IDENTIFICATION OF *CIS*-ELEMENTS AND TRANSACTING FACTORS INVOLVED IN THE ABIOTIC STRESS RESPONSES OF PLANTS

A Project Report submitted in partial fulfilment of the requirements for the degree of

MASTERS IN BIOINFORMATICS AND COMPUTATIONAL MOLECULAR BIOLOGY

of

RHODES UNIVERSITY

by

ATHLEE MACLEAR

February 2005

ABSTRACT

Many stress situations limit plant growth, resulting in crop production difficulties. Population growth, limited availability and over-utilization of arable land, and intolerant crop species have resulted in tremendous strain being placed on agriculturalists to produce enough to sustain the world's population. An understanding of the principles involved in plant resistance to environmental stress will enable scientists to harness these mechanisms to create stress-tolerant crop species, thus increasing crop production, and enabling the farming of previously unproductive land. This research project uses computational and bioinformatics techniques to explore the promoter regions of genes, encoding proteins that are up- or down-regulated in response to specific abiotic stresses, with the aim of identifying common patterns in the *cis*-elements governing the regulation of these abiotic stress responsive genes.

An initial dataset of fifty known genes encoding for proteins reported to be up- or down-regulated in response to plant stresses that result in water-deficit at the cellular level *viz.* drought, low temperature, and salinity, were identified, and a postgreSQL database created to store relevant information pertaining to these genes and the proteins encoded by them. The genomic DNA was obtained where possible, and the promoter and intron regions identified. The Neural Network Promoter Prediction (NNPP) software package was used to predict the transcription start signal (TSS) and the promoter searching software tool, TESS (Transcription Element Search Software) used to identify known and user-defined *cis*-elements within the promoter regions of these genes.

Currently available promoter prediction software analysis tools are reported to predict one promoter per kilobase of DNA, whilst functional promoters are thought to only occur one in 30-40 kilobases, which indicates that a large percentage of predictions are likely to be false positives (Pedersen *et al.*, 1999). NNPP was chosen as it was rated as the highest performing promoter prediction software tool by Fickett and Hatzigeorgiou (1997) in a thorough review of eukaryotic promoter prediction algorithms, however results were less than promising as very few predicted TSS were identified in the area 50 bps up- and downstream of the gene start site, where biologically functional TSSs are known to occur (Reese, 2000; Fickett and Hatzigeorgiou, 1997).

TESS results seemed to support the hypothesis that drought, low-temperature and high salinity plant stress response proteins have similar *cis*-elements in their promoter regions, and suggested links to various other gene regulation mechanisms *viz.* gibberellin-, light-, auxin- and development-regulated gene expression, highlighting the vast complexity of plant stress response processes. Although far from conclusive, results provide a valuable basis for future comparative promoter studies that will attempt to deduce possible common transcriptional initiation of abiotic stress response genes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABA	Abscisic Acid
ABRE	Abscisic Acid Responsive Element
AP2	*APETELA2*
API	Application Program Interface
BBF	rolB Domain Factor
BHLH	Basic Helix-loop-Helix
BLAST	Basic Local Alignment Search Tool
BZIP	Basic Leucine Zipper
CBF	C-Repeat Binding Factor
CDNA	Complementary DNA
CDS	Coding Sequence
CPRF-1	Common Plant Regulatory Factor 1
CRT	C-Repeat
DB	Database
DDBJ	DNA Data Bank of Japan
DNA	Deoxyribonucleic Acid
Dof2	DNA-Binding with One Finger 2
DRE	Dehydration Responsive Element
DREB	Dehydration Responsive Element Binding Protein
EBP/ EREBP	Ethylene-Responsive Element Binding Protein
EMBL	European Molecular Biology Laboratory
GA	Gibberellin
GAmyb	Gibberellin-Regulated Myb
GBF	G-box Binding Factor

GTF	General Transcription Factor
GUI	Graphical User Interface
HD	Homeodomain
HD-ZIP	Homeodomain-Leucine Zipper
HLH	Helix-Loop-Helix
Inr	Initiator
IO	Input-Output
JDBC	Java Database Connectivity
LEA	Late Embryogenesis Abundant
LIP	Low Temperature Induced Protein
LTRE	Low Temperature Responsive Element
MAPK	Mitogen-Activated Protein Kinases
Misc_feature	Miscellaneous Feature
Mol_type	Molecular Type
mRNA	Messenger Ribonucleic Acid
NLS	Nuclear Localisation Signal
NNPP	Neural Network Promoter Prediction
OBF	Ocs Element Binding Factor
ORDBMS	Object-Relational Database Management System
OSBZ8	*Oryza sativa* bZIP Protein 8
PBF	Prolamin Box Binding Factor
PDB	Protein Data Bank
PLACE	Plant *Cis*-acting Regulatory DNA Elements
PLD	Phospholipase D
SEF4	Soybean Embryo Factor 4
SPA	Storage Protein Activator

SsDBP-1	Single Strand DNA-Binding Protein 1
TBP-1	TATA-Binding Protein 1
TDNN	Time-Delay Neural Network
TESS	Transcription Element Search Software
TFD	Transcription Factor Database
TRES	Transcription Regulatory Element Search
TSS	Transcription Start Signal
WWW	World Wide Web

ACKNOWLEDGEMENTS

My sincere thanks and appreciation go to the following people for their contribution to this project report:

- Rhodes University, the National Research Foundation (NRF), and the National Bioinformatics Network (NBN) for funding the project.
- My supervisor, Dr. Graeme Bradley and co-supervisor, Dr. Fourie Joubert (University of Pretoria) for their support, guidance and encouragement.
- Dr. Greg Foster for his advice and help on programming, database design, and graphics.
- Shaun Bangay for his assistance and patience with endless system administration and installation problems and queries.
- My colleagues, Tim Akhurst, Bronwen Aken and Hamilton Ganesan for their help, companionship and encouragement throughout the year.
- My digs mates, Robyn Todd, Caryn McNamara, and Shane Govan for their support, patience and understanding.

CHAPTER 1 LITERATURE REVIEW

1.1. Introduction

The environment in which we live is dynamic and ever changing. Plants are unique in that they are sedentary organisms and are thus unable to escape the elements (Knight, 2000). As a result, they have evolved an abundance of stress response strategies to counteract the continual fluctuations in their immediate environment, and exhaustively monitor their surroundings and adjust their metabolic systems in order to maintain homeostasis (Bailey-Serres *et al.*, 1999; Knight, 2000; Knight and Knight, 2001; Pastori and Foyer, 2002). External conditions that adversely affect the growth, development and/or productivity of plants, are termed 'stresses' and may be biotic i.e. imposed by other organisms, or abiotic i.e. arising from an excess or deficit in the physical or chemical environment of the plant. Abiotic stresses that may cause plant damage include water-logging, drought, high or low temperature, excessive soil salinity, inadequate mineral nutrients in the soil, and an excess or deficit of light (Bray *et al.*, 2000).

This literature review outlines the abiotic stresses encountered by plants, specifically those causing water deficit at the cellular level, namely drought/dehydration, low temperature, and high salinity, and the mechanisms adopted by plants to respond to these environmental stresses.

1.1.1. Motivation

Statistics published by the U.S. Census Bureau in April 2004 indicate that the world population is increasing at a steady rate, and is expected to continue to do so in the foreseeable future (Figure 1). As a result, agriculture is under mounting pressure to provide for an ever-increasing volume of people, whilst at the same time competing with urban development for premium arable land.

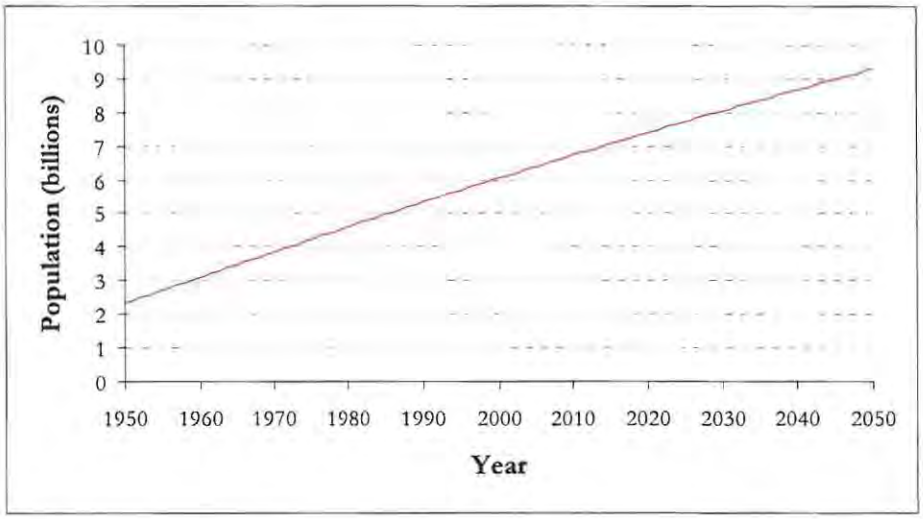

Figure 1: World Population: 1950-2050

(Source: U.S. Census Bureau, International Database, April 2004 version – Figure adapted from URL 18).

This trend is clearly demonstrated in Sub-Saharan Africa, where population growth rates are high and the vast majority of inhabitants poverty stricken or dying of starvation. This is evidence enough that agriculture is not meeting the needs of the population, be it from bad management/farming practises or sub-optimal conditions e.g. non-perennial water supply, high salinity soil, high diurnal temperature variation. This increasing demand coupled with diminishing resources, has driven scientific and commercial interest in unlocking the mechanisms by which plants respond to stress; the end goal of which would be to utilise this knowledge to enhance plant productivity in sub-optimal environments. Engineered mechanisms of stress tolerance will hopefully facilitate survival through periods of intense or prolonged stress, or maintain high plant productivity under conditions of moderate environmental stress (Oliver and Bewley, 1997; Bray *et al.*, 2000).

1.1.2. The Drought Environment

Simply defined, drought is a period with little or no rainfall. In terms of agriculture, drought is viewed as a dry spell leading to a drop in crop yields below that predicted under optimal water conditions. To the plant, however, drought is the combination of environmental stresses resulting from lack of rainfall (Thomas, 1997; Figure 2).

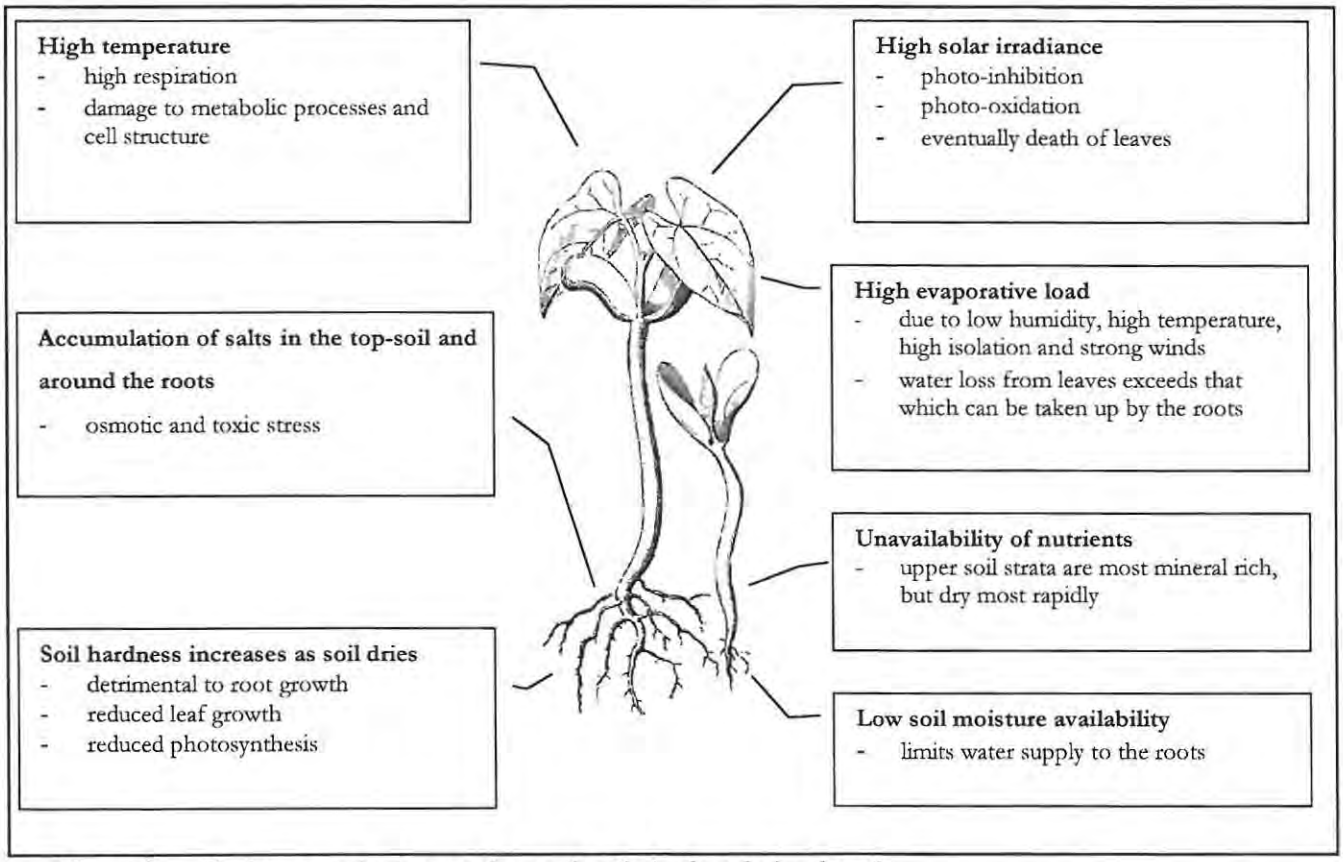

Figure 2: Environmental stresses acting on the whole plant during drought.

It is evident from figure 2 that a single environmental stress e.g. drought, results in a number of stresses when viewed from the whole plant perspective. It follows that plant responses are consequently multifaceted, and that there is likely considerable overlap in cellular responses elicited by different environmental stresses. This review therefore concentrates on those environmental stresses that cause water-deficit at the cellular level *viz.* drought/dehydration, high salinity and low temperature.

1.1.3. The role of water in the plant

Water performs a number of functions, which show different sensitivities to water stress. These are:

- Acting as a 'skeletal' material. Plants contain large vacuoles consisting primarily of water. These permit a small amount of dry matter to be spread over a large area or volume, enabling better exploration of the soil by the plant roots and more effective exploitation of the atmosphere and solar energy by the foliage (Thomas, 1997).
- Acts as a medium for metabolism, since many gases, salts and organic compounds are water-soluble (Thomas, 1997), e.g.
  - Nutrients are taken up from the soil in solution and transported to the leaves in the xylem flow.
  - Gas exchange for photosynthesis, occurs via the film of water in and on mesophyll cells.
  - Metabolites move through the plant in solution in the phloem.
- Maintains constant temperature by evaporating water from the leaves, thus cooling them and preventing overheating (Thomas, 1997).

1.2. The Stress Response Mechanism

Plants respond to different environmental signals in many different ways, however, a common chain of events can be identified which occurs in three stages (Figure 3):

- Sensing of environmental stress signals by specific receptor or sensor systems.
- Transduction of the signals by a sequence of processes that operate directly on metabolism or by altering gene expression. Metabolic changes can be rapid and induce fine changes, whilst the alteration of gene expression is slower and more energy consuming and thus is thought of as 'course control'.
- Initiation of a physiological response, enabling the phenotype to adapt to the current environment.

(Thomas, 1997; Xiong *et al.*, 2002)

The key components of the stress response relationship are outlined in Figure 3, namely the stress stimulus, signals, transducers, transcription regulators, target genes, and stress responses i.e. morphological, biochemical, and physiological changes (Pastori and Foyer, 2002).

Gene expression is primarily regulated at the initiation of transcription and is controlled by one or more interacting transcription factors that bind to specific *cis*-elements in the promoter region (Pastori and Foyer,

2002). In order to activate or repress transcription, transcription factors must be located in the nucleus, bind DNA and interact with the underlying transcription apparatus (Ramanjulu and Bartels, 2002). The rate of transcription under particular environmental conditions, whereby a specific promoter is activated or repressed is often governed by multiple protein-protein and/or DNA-protein interactions (Pastori and Foyer, 2002).

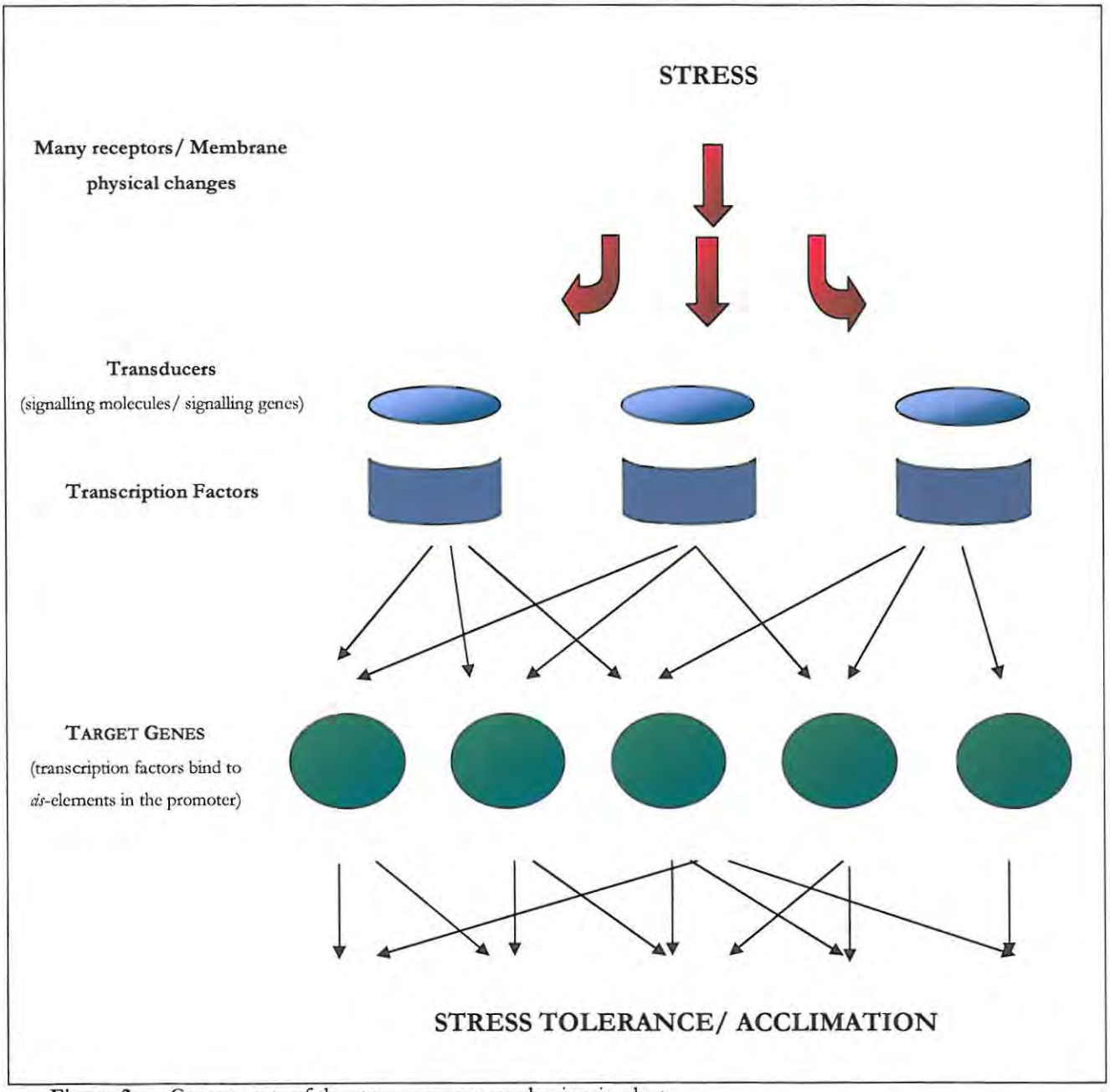

Figure 3: Components of the stress response mechanism in plants

(Figure adapted from Pastori and Foyer, 2002)

1.3. Biological Pathways Implicated in Drought, Salinity and Low-Temperature Stress-Response

The phyto-hormone, abscisic acid, is a key signal in response to environmental stresses relating to cellular water deficit, such as dehydration, high salinity and low temperature (Shinozaki and Yamaguchi-Shinozaki, 2000; Grover *et al.*, 2001; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). It was discovered in the 1950s affecting leaf abscission and bud dormancy, and has since been shown to mediate various developmental and physiological processes that affect agronomic crop performance, *viz.* embryo maturation, germination and the response of vegetative tissues to osmotic stress (Skriver and Mundy, 1990; Ramanjulu and Bartels, 2002).

Water stress i.e. dehydration, low-temperature and high salinity, is thought to elicit the accumulation of ABA, which then activates various stress-associated genes (Xing and Rajashekar, 2001; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). Kirch *et al.* (2002) report that all studies to date regarding the molecular events triggered in response to water deficit suggest a complex network of regulatory systems that mediate stress-induced gene expression, that involve both ABA-dependent and ABA-independent signal transduction pathways (Figure 4) (Kermode, 1997; Kirch *et al.*, 2002). At least four independent signal pathways have been identified under drought conditions. Two of these are ABA-dependent and two are ABA-independent. Additionally, two ABA-independent pathways are thought to function under low-temperature stress conditions. There is a common signal transduction pathway between dehydration and cold stress involving the DRE/CRT *cis*-acting element (Shinozaki and Yamaguchi-Shinozaki, 2000).

1.3.1. ABA-independent Regulation of dehydration-stress response

Many genes induced by exogenous ABA treatment are also induced by cold or dehydration in ABA-deficient (*aba*) or ABA-insensitive (*abi*) *Arabidopsis thaliana* mutants, which implies that these genes are not induced by the accumulation of endogenous ABA, however do respond to ABA (Shinozaki and Yamaguchi-Shinozaki, 2000; Chaves *et al.*, 2003). Two dehydration- and cold-inducible *Arabidopsis* genes that have been thoroughly researched are *rd29A/lti78/cor78* and *cor15a*. *Rd29A* transcription in *abi1* and *aba1* mutants indicates that cold- and drought-regulated expression does not require ABA (Shinozaki and Yamaguchi-Shinozaki, 2000).

Yamaguchi-Shinozaki and Shinozaki (1994) identified a *cis*-acting promoter element, dehydration-responsive element/C-repeat (DRE/CRT) that responds to osmotic stress but not to ABA. DRE is a 9-bp conserved sequence (i.e. TACCGACAT) that is essential for the initial rapid, ABA-independent induction of RD29A stress-responsive genes in *A. thaliana*. DRE-related motifs, which include the CCGAC motif that forms the core of the DRE sequence *viz.* CRT and low temperature-responsive element (LTRE), have since been identified in several other osmotic stress-responsive genes (Jaglo-Ottosen *et al.*, 1998; Shinozaki and Yamaguchi-Shinozaki, 2000; Grover *et al.*, 2001; Kirch *et al.*, 2002). Protein factors that bind to the DRE/CRT *cis*-element are classified into two groups, CBF1 (Stockinger *et al.*, 1997)/DREB1 and DREB2. Other promoter studies indicate that *cis*-elements other than DRE/CRT may be involved the ABA-

independent stress response. This suggests that there is a second ABA-independent signalling cascade involved in dehydration stress response (Kirch *et al.*, 2002; Chaves *et al.*, 2003).

1.3.2. ABA-dependent regulation of dehydration-stress response

A number of ABA-inducible genes depend on protein biosynthesis for their induction by ABA, suggesting that ABA-dependent pathways that are involved in the regulation of gene expression in response to dehydration-stress. Current research indicates that there are at least two ABA-dependent pathways governing drought stress response (Shinozaki and Yamaguchi-Shinozaki, 2000; Kirch *et al.*, 2002; Chaves *et al.*, 2003).

The slower ABA-dependent response of RD29A is controlled via a bZIP/ABRE (ABA responsive element) *cis*-element, which depends on the accumulation of endogenous ABA to trigger the signalling cascade (Kirch *et al.*, 2002; Chaves *et al.*, 2003).

A second ABA-dependent signalling pathway has been identified which requires *de novo* protein synthesis to initiate the expression of genes induced by dehydration (Guerrero and Mullet, 1986; Shinozaki and Yamaguchi-Shinozaki, 2000; Kirch *et al.*, 2002). In *A. thaliana*, the induction of a dehydration-responsive gene, *rd22*, is ABA-dependent, however the promoter does not include any sequence that corresponds to the consensus ABRE sequence, although it does contain recognition sites for some transcription factors i.e. MYC, MYB, and GT-1. The fact that *rd22* expression requires protein synthesis suggests that *rd22* expression might be regulated by trans-acting protein factors whose synthesis is dehydration- or ABA-induced (Iwasaki *et al.*, 1995; Shinozaki and Yamaguchi-Shinozaki, 2000).

This MYC/MYB system is thought to act in a slow and adaptive stress response process. The discrepancy in the timing of the induction of stress-responsive genes can be accounted for by the various regulatory elements functioning in their promoter regions *viz.* DRE/CRT, ABRE, and MYB/MYC (Shinozaki and Yamaguchi-Shinozaki, 2000).

The *rd29A* promoter, in *Arabidopsis* contains both the DRE and ABRE *cis*-elements. The ABRE *cis*-element and bZIP transcription factors function in ABA-responsive gene expression, whilst the DRE *cis*-element functions in ABA-independent gene expression, thus indicating that *rd29A* expression is mediated by more than one independent regulatory system. These results show that molecular responses to various environmental stresses may be controlled by both complex regulatory systems of gene expression and signal transduction, and by cross-talk between these mechanisms (Kermode, 1997; Shinozaki and Yamaguchi-Shinozaki, 2000).

1.3.3. Signal Perception

Knowledge of the mechanisms that govern dehydration stress perception and signalling is still very limited, however a number of postulations have been made on how plants initially sense stress. These include:

- Changes in membrane properties i.e. loss of turgor (Guerrero and Mullet, 1986; Guerrero *et al.*, 1990; Chaves *et al.*, 2003) and increased Ca^{2+} levels (Knight, 2000; Grover *et al.*, 2001; Xiong *et al.*, 2002; Kirch *et al.*, 2002). Cold, drought and salinity stress have been shown to induce transient Ca^{2+} influx to the cytoplasm, thus the Ca^{2+} channels responsible for this influx may represent one mechanism of stress perception (Knight, 2000; Chaves *et al.*, 2003). The activation of these channels is thought to result from physical changes in cellular structures i.e. loss of tugor due to water loss, or changes in membrane fluidity and cytoskeletal reorganisation as a result of low temperature (Knight, 2000; Xiong *et al.*, 2002).
- ATHK1 is transcriptionally upregulated in response to external osmolarity changes, and likely functions as an osmosensor transmitting a stress signal to a downstream MAPK (Mitogen-activated protein kinase) cascade (Urao *et al.*, 1999). The gene product of the ATHK1 gene (isolated by Urao *et al.*, 1999) in *A. thaliana* shows a high degree of homology with the yeast osmosensor SLN1. In addition, ATHK1 transcript levels were seen to accumulate under high and low osmolarity conditions (Urao *et al.*, 1999; Xiong *et al.*, 2002; Kirch *et al.*, 2002).
- Phospholipase D (PLD) plays an important role in the initial events leading to desiccation tolerance in *Craterostigma plantagineum* (Frank *et al.*, 2000; Chaves *et al.*, 2003). Phospholipases C, D, and A_2 are known to play a role in signalling pathways in animal cells by cleaving phospholipids present in the membrane, resulting in the formation of phosphatidic acid. Phosphatidic acid acts as a secondary messenger amplifying the original stress signal. A member of the phospholipase D (PLD) family was cloned from the resurrection plant, *C. plantagineum*. PLD activity in *C. plantagineum* was shown to increase significantly within minutes of exposure to dehydration (Kirch *et al.*, 2002; Xiong *et al.*, 2002). Additionally, Frank *et al.* (2000) report changes in the subcellular localization of the PLD protein.

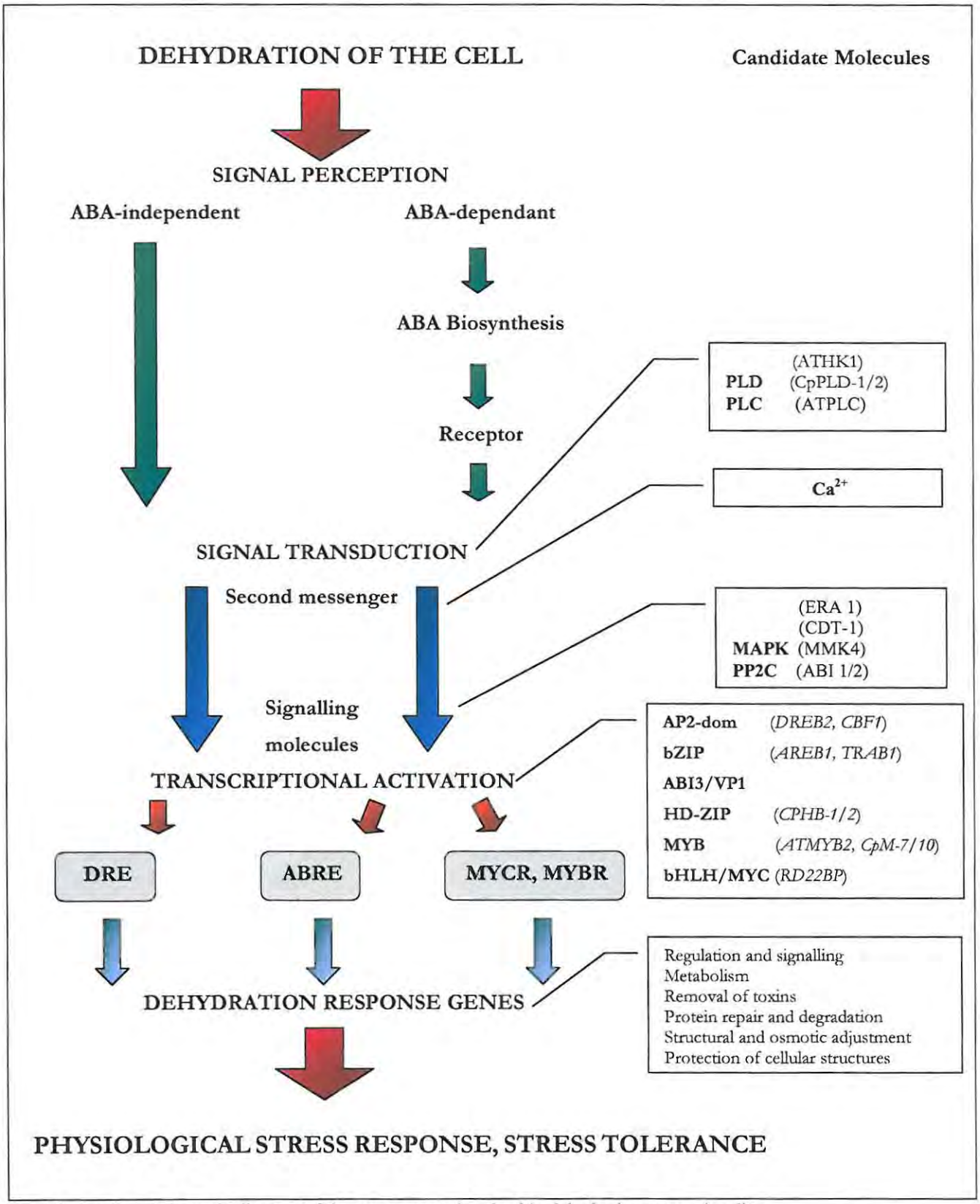

Figure 4: Schematic diagram of the components involved in dehydration-stress signaling

(Figure adapted from Kirch *et al.*, 2002)

1.3.4. Signalling Components

A number of factors have been proposed to act downstream of signal perception and upstream of transcription factor binding. A few of these have been summarised in table 1 below.

Table 1: Signal transduction components implicated in the expression of water-deficit stress-related genes

Component	Reference
Protein kinases	
AtDBF2 – salt, osmotic, heat and cold responsive, *A.thaliana*	Grover *et al.*, 2002
ARSK1 – dehydration, ABA, and salt stress responsive, *A.thaliana*	Hwang and Goodman, 1995
PKABA1 – ABA-responsive, *Oryza sativa*	Kirch *et al.*, 2002
RPK1 – dehydration, low temperature, salt and osmotic stress responsive, *P.sativum*	
Calcium-dependent protein kinases	
Dehydration-inducible, *A.thaliana*	Kirch *et al.*, 2002
Mitogen-activated kinases (MAPK)	
Activated by drought stress and/or ABA, *A.thaliana, C.plantagineum*	Kirch *et al.*, 2002
Induced by low temperature, drought and ABA, Alfalfa	
Phosphatases	
AtPTP1 – high salt inducible, and negatively regulated by cold, *A.thaliana*	Grover *et al.*, 2002
AtPP2C – ABA-inducible, *A.thaliana*	Kirch *et al.*, 2002
Ca^{2+} binding proteins	
AtCBL 1, 2, and 3 – cold, drought, salt and wounding responsive, *A.thaliana*	Grover *et al.*, 2002
AtCP – high salt-inducible, *A.thaliana*	
Phospholipases	
AtPLC – drought-induced, *A.thaliana*	Kirch *et al.*, 2002
PIP5K I – dehydration-responsive, *A.thaliana*	Xiong *et al.*, 2002
PLD – desiccation-inducible, *C.plantagineum*; ABA-responsive, barley.	

1.3.5. Transacting factors

Promoter *cis*-elements function in collaboration to recruit trans-acting, DNA-binding proteins that interact with RNA polymerase II at the exact time and location required for the initiation of transcription. Transcription factors are trans-acting proteins that bind to specific *cis*-elements (Ferl and Paul, 2000). A sizeable portion of all eukaryotic genomes are comprised of transcription factor genes, the majority of which can be classified into a selection of different, often large, gene families defined by their structural features, principally by the type of DNA-binding domain that they encode (Riechmann and Radcliffe, 2000), whilst

subfamilies are determined by the number and spacing of conserved residues occurring in the most similar domain (Liu *et al.*, 1999).

1.3.6. Plant Transcription Factors

A typical plant transcription factor contains a DNA-binding region, an oligomerisation site, a transcription regulation domain and a nuclear localisation signal (Liu *et al.*, 1999).

- *DNA-binding domains* – are often basic in character, and contain those amino acid residues determine the specificity of the protein by binding to the *cis*-acting element in the promoter. The base-recognition residues are often highly conserved, and their spatial organisation is important for specificity. Other residues enhance transcription factor binding by non-specific DNA contact through interaction with either phosphate or deoxyribose moieties. Many plant transcription factors possess both specific and non-specific DNA-binding domains, with the latter occasionally required for trans-activation of genes. The secondary structure of DNA-binding domains is also thought to affect their affinity and selectivity. Each plant transcription factor only contains one type of DNA-binding domain, which can occur in single or multiple copies e.g. most Myb-related proteins have two Myb domains (Liu *et al.*, 1999).
- *Oligomerisation site* – many plant transcription factors form hetero- and/or homo- oligomers, which affect the DNA-binding specificity, the affinity for promoter elements and the nuclear localisation of the transcription factor. The amino acid sequences of these domains are usually highly conserved, with each type combining with DNA-binding regions to form discrete three-dimensional arrangements. The oligomerisation domain length may vary within the same family of transcription factors. Variations in oligomerisation increase the flexibility of the transcription apparatus, allowing them the capacity to modulate gene expression (Liu *et al.*, 1999).
- *Transcription regulation domains* – are regions within the protein that tend to diverge from one another. It is these differences in their regulation domains that allow transcription factors of the same family to have distinct actions. Regulation domains, and hence transcription factors, function as either activators or repressors, depending on whether they inhibit or stimulate the transcription of target genes (Liu *et al.*, 1999).
- *Nuclear Localisation signals (NLSs)* – plant transcription factors contain nuclear localisation signals characterised by a core peptide enriched in arginine (R) and lysine (K) that allow them to selectively enter the nucleus. Flanking residues influence the action of the basic core. Within plant transcription factors, nuclear localisation signals vary in sequence, organisation and number (Liu *et al.*, 1999).

Plant transcription factor genes may be expressed constitutively or may be dependent on organ type, external stimulus (e.g. drought, low-temperature, high salinity, ABA), development stage or cell cycle (Liu *et al.*, 1999). Transcription factor genes may respond to multiple environmental cues, e.g. the low-temperature induced protein 15 (mLIP15) from maize, has the ability to bind to the wheat histone gene (*H3*) and the low-temperature-inducible gene (*Adh1*) promoters (Liu *et al.*, 1999). In addition, transcription factor genes of the

same family do not necessarily respond to the same environmental stimulus, e.g. members of the bZIP family have been shown to be induced by light (Liu *et al.*, 1999; Terzaghi *et al.*, 1997), abscisic acid (Lee *et al.*, 2003), drought (Ramanjulu and Bartels, 2002), auxin (Baumann *et al.*, 1999), and salicyclic acid (Liu *et al.*, 1999).

1.3.7. Plant transcription factors implicated in dehydration and/or water deficit stress response

Several classes of plant transcription factors have been identified as having a role in plant response to dehydration/water deficit stress response (Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002). These are summarised in table 2 and discussed below. It is the interaction between these factors and pre-existing factors that ultimately determines the response mechanism that results in stress-induced gene expression and stress adaptation (Riechmann and Ratcliffe, 2000; Ferl and Paul, 2000; Bray *et al.*, 2000; Ramanjulu and Bartels, 2002). The complexity of this response is shown in figure 4 above. In order to induce or inhibit transcription, transcription factors need to be located in the nucleus, bind to the DNA and interact with the basal transcription machinery, thus dehydration may affect one or a combination of these processes (Ramanjulu and Bartels, 2002).

Table 2: Plant transcription factors that function in dehydration/water-deficit signal transduction

Transcription factor class	DNA Consensus Sequence	Examples	References
Basic leucine zipper (bZIP) protein	CACGTG (ABRE) (G-box - ACGT)	Wheat – *EmBP-1* Tobacco – TAF-1 Rice – OSBZ8, OsZIP-1a, TRAB1 *A. thaliana* – ABF2, ABF3, ABF4, ABRE-binding proteins (AREB1 and AREB2)	(Ingram and Bartels, 1996; Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002; Grover *et al.*, 2001)
AP2 (APETALA2) EREBP domain	TACCGACAT (DRE/CRT/LTRE) (Core sequence – CCGAC)	DREB and CBF proteins i.e. DREB1A, DREB2A, CBF1, CBF2, CBF3	(Ingram and Bartels, 1996; Stockinger et al., 1997; Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002; Grover *et al.*, 2001)
HD-Zip (Homeodomain-leucine zipper) protein	HDE consensus CAAT(X)ATTG	*C. plantagineum – CPHB-1, CPHB-2*	(Ingram and Bartels, 1996; Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002; Grover *et al.*, 2001)
MYB-like protein	TGAACT	*A. thaliana – Atmyb2* *C. plantagineum – cpm-7, cpm-10*	(Ingram and Bartels, 1996; Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002; Grover *et al.*, 2001)
MYC-like Basic helix-loop-helix (bHLH) protein	CANNTG	*A. thaliana – rd22 BP1*	(Ingram and Bartels, 1996; Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002; Grover *et al.*, 2001)

1.3.7.1. bZIP proteins

The basic-leucine zipper (bZIP) domain contains a DNA-binding and dimerisation motif, and has been found in the transcription factors of a variety of eukaryotes. The DNA-binding motif is an approximately 25 amino acid stretch rich in basic residues, and immediately adjacent to the leucine zipper (ZIP) (Kirch *et al.*, 2002). bZIP proteins create a scissors-like dimmer that binds in the major groove of the DNA. Each

monomer contains a α-helix with a leucine at every seventh residue. These face one another in the dimer forming a structural arrangement that "zips" the helices together into a coiled coil, which holds the basic region in position within the major groove of the DNA (Ferl and Paul, 2000).

The G-box binding factors are a family of bZIP proteins that freely form heterodimers. The G-box motif (ACGT) is widespread in its distribution, thus heterodimers with slightly different binding affinities confer specificity among the diversity of promoters that contain the G-box (Ferl and Paul, 2000). Many plant bZIP proteins bind to elements with the G-box core sequence, ACGT. The ABA-responsive element (ABRE) is the best-known G-box *cis*-element in relation to dehydration stress tolerance and contains the palindromic motif CACGTG (Kirch *et al.*, 2002).

Several bZIP factors that bind ABA-responsive elements (ABREs) are candidates for ABA-responsive transcription factors that trigger gene expression by dehydration and/or ABA (Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002). The transcription of ABRE-binding proteins, AREB1 and AREB2, is drought-, high-salt- and ABA-responsive. Both proteins were shown to activate transcription of a reporter gene driven by ABRE and were dependant on ABA for their activity, thus it is thought that these proteins act as transcription factors in an ABA-dependant signalling pathway (Ramanjulu and Bartels, 2002).

The maize VP1 and *Arabidopsis* ABI3 proteins are homologues of transcription factors necessary for ABA activity in seeds (Kirch *et al.*, 2002). TRAB1, an ABRE-binding bZIP transcription factor from rice, has been shown to interact with VP1, suggesting that VP1 functions by binding to a factor that interacts with ABRE, thereby regulating ABA-induced transcription (Ramanjulu and Bartels, 2002). Thus VP1/ABI3 proteins are thought to be transcriptional co-activators that interact with ABRE-binding proteins (Kirch *et al.*, 2002).

1.3.7.2. AP2 (APETALA2) EREBP domain

Protein factors that specifically bind to the 9-bp DRE *cis*-element (previously discussed) have been identified in nuclear extracts prepared from either dehydrated or adequately watered *Arabidopsis* plants (Shinozaki and Yamaguchi-Shinozaki, 2000). DREB1A and DREB2A are DRE-binding proteins that specifically interact with the DRE *cis*-element in the promoter of the *Arabidopsis* gene, *rd29A*. Similarly, CBF-1 is a DRE/CRT-binding protein that binds to the CRT/DRE motif in several cold regulated genes (Shinozaki and Yamaguchi-Shinozaki, 2000; Grover *et al.*, 2001; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002).

DREB and CBF proteins contain a conserved DNA-binding motif, AP2 domain, that is also found in the EREBP (ethylene-responsive element binding protein) family and AP2 protein, which is encoded from the *Arabidopsis* floral homeotic gene, *APETELA2* (Shinozaki and Yamaguchi-Shinozaki, 2000; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). Protein factors that bind to the DRE/CRT *cis*-element are classified into two groups, CBF1/DREB1 and DREB2 (Kirch *et al.*, 2002). The groups contain similar AP2 domains but have

low sequence similarity outside that domain. There are three DREB1 proteins, *DREB1B, DREB1A,* and *DREB1C,* and two DREB2 proteins, DREB2A and DREB2B. DREB1B is identical to CBF1, and DREB1A and DREB1C, are homologous to CBF3 and CBF2 respectively. Both DREB1A and DREB2A bind specifically to DRE/CRT and function as transcriptional activators, however expression of the *DREB1A/CBF3* gene and its two homologues (i.e. *DREB1B/CBF1* and *DREB1C/CBF2*) is induced by low temperature stress, whereas the two *DREB2* genes are induced by dehydration (Shinozaki and Yamaguchi-Shinozaki, 2000; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). These results indicate that at least two separate families of C-repeat/DRE binding factors function as transcription factors in independent signal transduction pathways (Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002), i.e. DREB1 proteins function in cold-specific gene expression, whereas DREB2 proteins are involved in dehydration-specific gene expression (Shinozaki and Yamaguchi-Shinozaki, 2000).

1.3.7.3. HD-Zip (Homeodomain-leucine zipper) protein

The homeodomain (HD) is a DNA-binding domain that is conserved throughout eukaryotes (Kirch *et al.*, 2002). Homeodomain-leucine zipper (HD-ZIP) proteins, as yet only identified in plants, are putative transcription factors thought to regulate development and responses to environmental cues. They are characterised by a ZIP motif immediately following the homeodomain (Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). Five HD-Zip transcripts have been reported to be upregulated in response to dehydration. Two HD-ZIP family II genes from *C. plantagineum, CPHB-1* and *CPHB-2*, were shown to be dehydration-inducible, but showed different responses to ABA. This difference suggests that *CPHB-1* and *CPHB-2* act in different branches of the dehydration-induced network (Ramanjulu and Bartels, 2002). The accumulation of *ATHB-7* transcripts from *Arabidopsis*, were shown to be upregulated in response to water stress and ABA. *ATHB-6*, and *ATHB-12* were also identified as water-stress-responsive in an ABA-dependant transduction pathway (Lee *et al.*, 2001; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002).

1.3.7.4. MYB-protein

The MYB domain is classified as the DNA-binding domain of the proto-oncogene Myb, and is composed of three copies of an approximately 50 amino acid repeat containing conserved tryptophan residues. These tryptophan residues are organised 18-19 residues apart and form the hydrophobic core of the protein (Kirch *et al.*, 2002). Myb proteins appear to be a conserved family of ubiquitous transcription factors. The expression of Myb-related genes, *cpm7* and *cpm10*, from *C. plantagineum*, is upregulated by dehydration and ABA respectively. As the expression of *cpm7* is restricted to roots, which represent a plant's primary sensor of water deficit, it is thought that the encoded product acts as a transcription factor inducing genes responsive to dehydration (Neale *et al.*, 2000; Ramanjulu and Bartels, 2002). The *Atmyb2* gene from *Arabidopsis* is a homologue of Myb, and is dehydration-responsive. *Atmyb2* transcripts have also been shown to accumulate upon salt stress and in response to exogenous ABA (Kirch *et al.*, 2002).

1.3.7.5. MYC-like Basic helix-loop-helix (bHLH) protein

Helix-loop-helix (HLH) proteins are composed of two α-helices separated by an intervening loop. HLH factors always bind the DNA as a dimer – one helix of each monomer works to hold the dimer together by interacting with its counterpart, whilst the other pair forms a scissors-like structure where each "blade" fits into the adjacent major groove on the DNA (Ferl and Paul, 2000).

Myc-like proteins contain the basic helix-loop-helix (bHLH) domain, composed of two subdomains: the basic region responsible for DNA binding and the helix-loop-helix (HLH) region responsible for dimerisation (Ramanjulu and Bartels, 2002; Kirch *et al.*, 2002). bHLH proteins bind *cis*-elements with a CANNTG core consensus sequence, however flanking residues are also important for binding and determine the specificity of the protein (Kirch *et al.*, 2002). The *Arabidopsis rd22 BP1* gene encodes a Myc homologue transcription factor that has been shown to be dehydration-, high-salt- and ABA-responsive (Ramanjulu and Bartels, 2002).

1.4. Products of Genes Up-Regulated in Response to Water-Deficit

Environmental stresses that result in water-deficit trigger the expression of an array of new proteins that enable the plant to respond appropriately in order to adapt and survive under stress conditions. These proteins have been shown to perform a variety of cellular functions (Bray, 1993; Ingram and Bartels, 1996; Grover *et al.*, 2001; Ramanjulu and Bartels, 2002). These are summarised in Figure 5 and discussed in the text below.

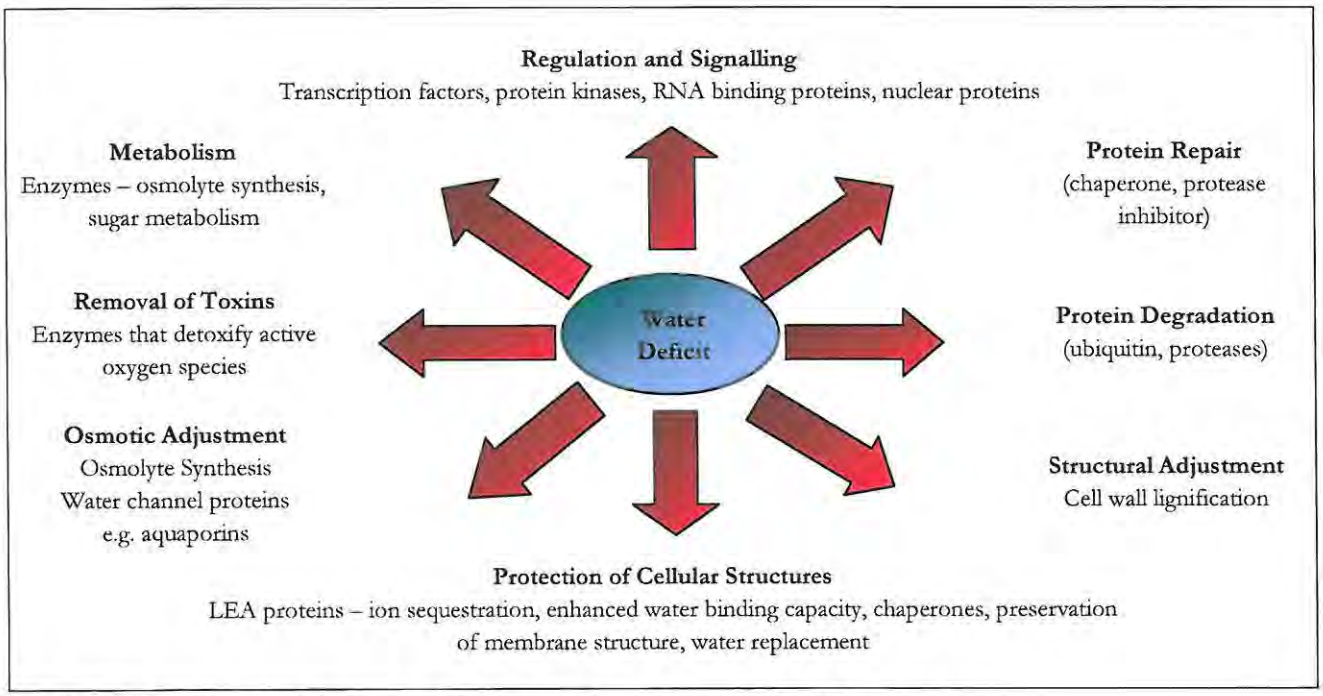

Figure 5: Predicted functions of the gene-products of water-deficit induced genes (Figure adapted from Bray, 1993)

1.4.1. Protection of Cellular Structures

A number of water-deficit-induced gene products are predicted to function in protecting cellular structures from damage caused by water loss. These genes, frequently called *lea*, encode late embryogenesis abundant (LEA) proteins (Bray, 1993; Bohnert *et al.*, 1995; Oliver and Bewley, 1997; Swamy and Smith, 1999; Bray *et al.*, 2000; Grover *et al.*, 2001; Ramanjulu and Bartels, 2002). They were initially identified as genes that are expressed during the maturation and desiccation phases of seed development (Bray, 1993). It has subsequently been shown that these genes are also expressed in vegetative tissues experiencing water-deficit resulting from drought/dehydration, osmotic and low-temperature stress (Bray, 1993; Oliver and Bewley, 1997; Ramanjulu and Bartels, 2002). *Lea* genes in seeds and vegetative tissue are also induced by ABA (Swamy and Smith, 1999; Ramanjulu and Bartels, 2002).

At least six groups of *lea* genes have been identified (Bray, 1993; Swamy and Smith, 1999; Bray *et al.*, 2000; Ramanjulu and Bartels, 2002). These are defined according to their amino acid sequence homologies, biochemical properties and alignment with proteins from cotton (Swamy and Smith, 1999; Ramanjulu and Bartels, 2002). The majority of the LEA proteins are predominantly hydrophilic, biased in amino acid composition, lacking in cysteine and tryptophan, and are predicted to be located in the cytoplasm (Bray, 1993; Ramanjulu and Bartels, 2002). Specific functions for each group of LEA proteins were proposed after careful analysis of the individual amino acid sequences and predicted structures of the proteins (Bray, 1993). Each group has a characteristic stretch of amino acid sequence that is highly conserved (Swamy and Smith, 1999; Ramanjulu and Bartels, 2002). The six groups of LEA proteins are summarised in table 3 below.

Table 3: The six groups of LEA proteins
(table adapted from Bray *et al.*, 2000; references consulted – Bray, 1993; Bray *et al.*, 2000; Swamy and Smith, 1999; Ingram and Bartels, 1996; Ramanjulu and Bartels, 2002)

Group	Representative Protein	Conserved Amino acids/ consensus sequence	Proposed function
Group 1 (D-19-LEA family)	Early Met-labelled protein (Em) (Wheat)	High percentage of charged residues and glycine residues	Enhanced water-binding capacity
Group 2 D-11-LEA (Dehydrins)	D-11 (Cotton)	EEKKGIMDKIKELPG	Chaperone function or one that preserves protein structure i.e. stabilises proteins structure
Group 3 D-7-LEA	D-7 (Cotton)	11-mer amino acid motif, TAQAAKEKAGE	Sequestration of ions that are concentrated during cellular dehydration
Group 4 D-95-LEA	D-95 (Soybean)	The N-terminus amino acid sequence which is thought to form an α-helix is conserved	Replaces water to preserve membrane structure
Group 5 D-113-LEA	D-113 (Cotton) LE25 (Tomato)	11-mer repeat – each amino acid in the motif has similar chemical properties to group 3	Sequesters ions during water loss
Group 6 D-29-LEA	D-29	Similar to group 3	May be similar to group 3 (Swamy and Smith, 1999); some references only list 5 groups (Bray *et al.*, 2000)

1.4.2. Osmotic Adjustment

Plants maintain total water potential during water deficit by osmotic adjustment (Bray, 1993; Bohnert *et al.*, 1995; Ingram and Bartels, 1996; Ramanjulu and Bartels, 2002). Cellular water potential is reduced to below the external water potential by decreasing osmotic potential, thus allowing water to move into the cell. This is achieved by the accumulation of osmolytes (compatible solutes) in the cytoplasm e.g. pyrroline-5-carboxylate synthase, betaine aldehyde dehydrogenase (Bray, 1993; Bohnert *et al.*, 1995; Ingram and Bartels, 1996; Ramanjulu and Bartels, 2002), and glycine betaine (Xing and Rajashekar, 2001). Genes that encode enzymes involved in the synthesis of these osmolytes have been identified, and in some cases, have been shown to be induced by water deficit (Bray, 1993; Bohnert *et al.*, 1995). Some examples of osmolytes include sugars, polyols, proline, quaternary ammonium compounds and tertiary sulfonium compounds (Bohnert *et al.*, 1995; Ramanjulu and Bartels, 2002; Pnueli *et al.*, 2002).

Ion and water channels are hypothesised to play an essential role in regulating water flux during drought-stress. This hypothesis has been supported by the isolation of channel protein genes that are induced in response to water deficit (Guerrero *et al.*, 1990; Bohnert *et al.*, 1995; Ingram and Bartels, 1996; Grover *et al.*, 2001). Channel proteins accumulate in the tonoplast during stress facilitating the movement of water or solutes from the vacuole to the cytoplasm resulting in the alteration of either the water content or the osmotic potential of the cytoplasm (Bray, 1993; Bohnert *et al.*, 1995).

The aquaporins are an important element of the cellular water transport mechanism (Bohnert *et al.*, 1995; Ramanjulu and Bartels, 2002). These are a complex family of channel proteins that facilitate the transport of water along transmembrane water potential gradients (Bohnert *et al.*, 1995; Ramanjulu and Bartels, 2002). Aquaporins can regulate the hydraulic conductivity of membranes and potentially increase water permeability by 10 – 20-fold. Dehydration mediates the regulation of expression and activity of these proteins (Ramanjulu and Bartels, 2002; Chaves *et al.*, 2003).

1.4.3. Structural Modification

Drought stress can result in adjustments in the chemical and physical properties of the cell wall e.g. wall extensibility (Ingram and Bartels, 1996). These alterations may include the genes encoding S-adenosylmethionine synthetase (Espartero *et al.*, 1994). Under normal conditions, increased expression of S-adenosylmethionine synthetase genes can be linked to regions where lignification is occurring, thus it is reasonable to conclude that increased expression in drought-stressed tissues could also be due to the lignification of the cell wall. Cell elongation is halted during prolonged drought stress, and seems to be followed by the initiation of lignification processes (Ingram and Bartels, 1996).

Suberisation is a process similar to lignification and may even have preceded lignification. In evolutionary terms, suberisation was of the utmost importance to plants developing the ability to live on land. Suberised

tissues are found in underground organs e.g. roots, and in periderm layers e.g. bark, and are formed as multilamellar domains composed of alternate polyaliphatic and polyaromatic layers. These layers contribute to cell wall strength and form impenetrable barriers that provide a mechanism to limit uncontrolled water loss by the whole plant (Croteau *et al.*, 2000).

1.4.4. Degradation and Repair

Genes encoding two types of protein-degrading mechanisms, namely proteases and ubiquitin, are induced by water deficit (Bray, 1993; Chaves *et al.*, 2003). Proteins with sequence similarity to proteases have been isolated in more than one plant species, and have been shown to be induced by drought e.g. *pA1494* from *A.thaliana* and *15a* from *P.sativum*. These enzymes are thought to degrade proteins irreparably damaged by the effects of abiotic stress conditions (Ingram and Bartels, 1996; Grover *et al.*, 2001) and to dispense with redundant proteins and depolymerise vascular storage peptides, thereby freeing up amino acids for use in the large-scale production of new proteins (Bray, 1993; Ingram and Bartels, 1996). *A. thaliana*, shows increased levels of mRNA encoding ubiquitin extension protein in the initial stages of drought stress. Ubiquitin extension protein is a fusion protein that is a precursor to active ubiquitin, and is derived by proteolytic processing. Ubiquitin functions in the degradation process by tagging proteins for destruction (Ingram and Bartels, 1996).

Counteracting the above-mentioned degradation mechanisms are chaperones and protease inhibitors, which are also induced by water deficit (Bray, 1993; Bohnert *et al.*, 1995). During drought stress, protein residues can be altered by chemical processes i.e. deamination, isomerisation, or oxidation, thus one can safely deduce that enzymes that play a role in protein repair are upregulated as a result (Ingram and Bartels, 1996; Oliver and Bewley, 1997; Ramanjulu and Bartels, 2002). For example, the synthesis and activity of L-isoaspartyl methyltransferase is enhanced by water-deficit in wheat seedlings, and converts modified L-isoaspartyl in damaged proteins back to L-aspartyl (Ramanjulu and Bartels, 2002). Two drought-induced gene products with similarity to heat shock proteins have been isolated by differential screening (Kiyosue *et al.*, 1994). These are most likely chaperonins, that function in protein repair by assisting other proteins, denatured or mis-folded during water stress, to refold into their native conformation (Bohnert *et al.*, 1995; Ingram and Bartels, 1996; Pnueli *et al.*, 2002; Haralampidis *et al.*, 2002).

1.4.5. Removal of toxins

The expression of enzymes involved in the removal of toxic intermediates produced during oxygenic metabolism (e.g. glutathione reductase, glutathione S-transferase, superoxide dismutase, and ascorbate peroxidase) increases in response to drought stress and is thought to be very important in stress tolerance (Bohnert *et al.*, 1995; Ingram and Bartels, 1996; Grover *et al.*, 2001). Declining leaf water content and resultant stomatal closure leads to reduced availability of CO_2 and the production of active oxygen species e.g. superoxide radicals. In addition, elevated photorespiratory and glycolate-oxidase activity go hand in hand

during drought, and result in the production of H_2O_2. It follows that genes encoding enzymes that detoxify active oxygen species have been shown to be upregulated in response to drought (Ingram and Bartels, 1996).

1.4.6. Regulation and Signalling (discussed in more detail in section 3)

Genes encoding proteins with probable involvement in the regulation and signalling process during periods of water deficit, i.e. protein kinases, calcium-dependent kinases, mitogen-activated kinases, phospholipases, and phosphatases have been identified. Transcription factors that recognise DNA elements within drought-induced promoters have also been identified *viz.* bZIP, AP2/EREBP, HD-Zip, MYB-like and MYC-like bHLH proteins (Bray, 1993; Shinozaki and Yamaguchi-shinozaki, 2000; Grover *et al.*, 2001; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002).

1.5. **Research Strategies**

Dehydration tolerance in plants has traditionally been investigated using three main approaches:

- Examining tolerant systems i.e. specific structures or species that can survive severe desiccation, *viz.* certain seeds and resurrection/ desiccation-tolerant plant species, for example, angiosperms, mosses, and ferns. A thorough molecular investigation of such tolerant systems should ideally reveal evidence of expressed genes that encode for desiccation tolerance (Bohnert *et al.*, 1995; Ingram and Bartels, 1996).
- Analysing mutants from genetic model species, for example *A. thaliana*. The advantage of using model species is that there is extensive genetic information, a wide variety of mutants, and the viability of positional gene cloning (Bohnert *et al.*, 1995; Ingram and Bartels, 1996).
- Analysing the effects of stress on agriculturally relevant plants. This method is useful because as a consequence of years of exhaustive breeding or *in vitro* selection, lines are available with varying degrees of tolerance, thus correlative evidence can be elucidated for genes implicated in drought response (Bohnert *et al.*, 1995; Ingram and Bartels, 1996).

The last few decades have seen dramatic growth in the amount of sequence data stored in public databases *viz.* GenBank, EMBL, and DDBJ (Figure 6). A significant amount of wet-bench research has already been done on plant responses to dehydration/water deficit stress, the bulk of which is freely available from the public databases. This project aims to collect this information into a central database, and using available software tools, perform an *in-silico* analysis of it.

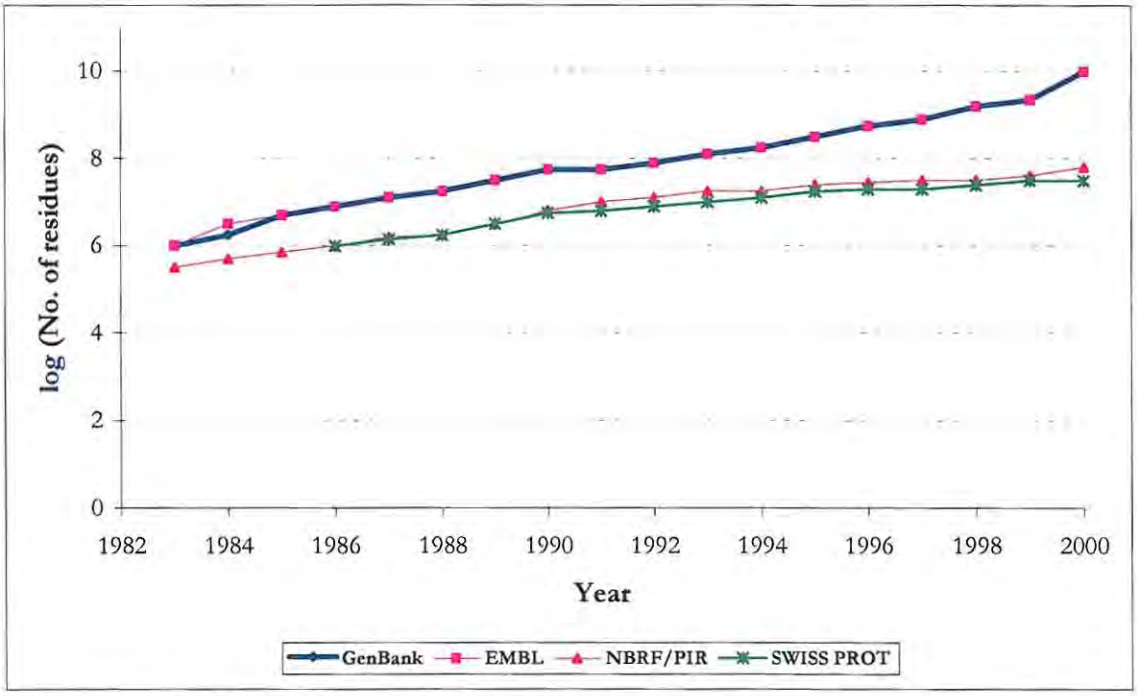

Figure 6: Growth of data stored in public biological databases
(Figure adapted from URL 36).

1.6. Project Outline

1.6.1. Problem Statement

Many stress situations limit plant growth, resulting in crop production difficulties (Hawkesford, 2001). The world population is steadily increasing, and will be for the foreseeable future. Arable land is in short supply and over-utilization of this land increases the salinity of the soil. The majority of crop species are intolerant to environmental stresses. The combination of these factors has resulted in a tremendous strain being placed on agriculturalists to produce enough to sustain the world's population.

An understanding of the principles involved in plant resistance to environmental stress will enable scientists to harness these mechanisms to create stress-tolerant crop species, thus increasing crop production, and enabling the farming of previously unproductive land (Hawkesford, 2001).

This project seeks to explore the promoter regions of genes, encoding proteins that are up- or down-regulated in response to specific abiotic stresses, with the aim of identifying common patterns, which can then be used to identify novel genes involved in specific plant stress responses.

1.6.2. Hypothesis

Drought, low-temperature and high salinity plant stress response proteins have similar *cis*-elements in their promoter regions.

1.6.3. Objectives

- Identify known genes encoding for proteins that have been reported to be up- or down-regulated in response to drought, low temperature and salinity plant stresses.
- Create a database to store relevant information pertaining to these genes and the proteins encoded by them.
- Using available software identify **known** *cis*-elements within these genes.
- Identify **novel** *cis*-elements within these genes.
- Identify significant patterns in the *cis*-elements.
- Identify novel genes that may be involved in drought, low temperature and salinity plant stress response.

CHAPTER 2 MATERIALS AND METHODS

2.1. System Configuration

- Operating System – Fedora Core 2 Linux, with kernel version 2.6.8-1.521.

2.2. Database Specifications

2.2.1. Database Development

PostgreSQL 7.4

URL: http://www.postgresql.org

PostgreSQL is an object-relational database management system (ORDBMS) based on POSTGRES, Version 4.2 (URL 10). The database, athlee_db, was created using PostgreSQL, in order to store sequence information of known genes encoding for proteins that are reported to be up- or down-regulated in response to drought, low temperature, and salinity plant stresses and relevant information pertaining to each gene.

2.2.2. Perl Database Interface

Pg.pm

URL: http://search.cpan.org/~mergl/pgsql_perl5-1.9.0/Pg.pm

Pg is a Perl5 extension for PostgreSQL, and allows the user access to all the functions of the Libpq interface of PostgreSQL. Pg uses blessed references as objects, with the Libpq functions serving as virtual methods after a new connection or result object is created. These functions can be divided into three categories, Connection, Result, and Large Objects (URL 11).

The Pg module was used to facilitate querying, manipulation, and insertion of data into athlee_db from within a perl script, and is identified at the top of the various perl scripts by "**use** Pg;".

2.3. Programming

2.3.1. Perl 5

Perl is the programming language of choice for bioinformatics as it simplifies many routine bioinformatics tasks, for example, it processes and manipulates long strings easily, thus is uniquely suited to sequence manipulation and retrival of information in flat files (ASCII text files) which is typically how data is stored in public biological databases such as GenBank and PDB (Tisdall, 2001).

Where possible perl scripts were written to automate the various data manipulations necessary to accomplish project objectives. These are detailed in the Appendices section and discussed in Chapter 3 of this report.

2.3.2. BioPerl-1.4

URL: http://www.bioperl.com

The Bioperl server is a web-based resource of modules, scripts, and web-links for developers of open-source Perl-based software tools for bioinformatics, genomics and life science research (URL 16). Specific Bioperl modules used are identified by the "use" keyword at the start of each perl script i.e. "**use** Bio::SearchIO;".

2.4. Inputs, Outputs, and Processes

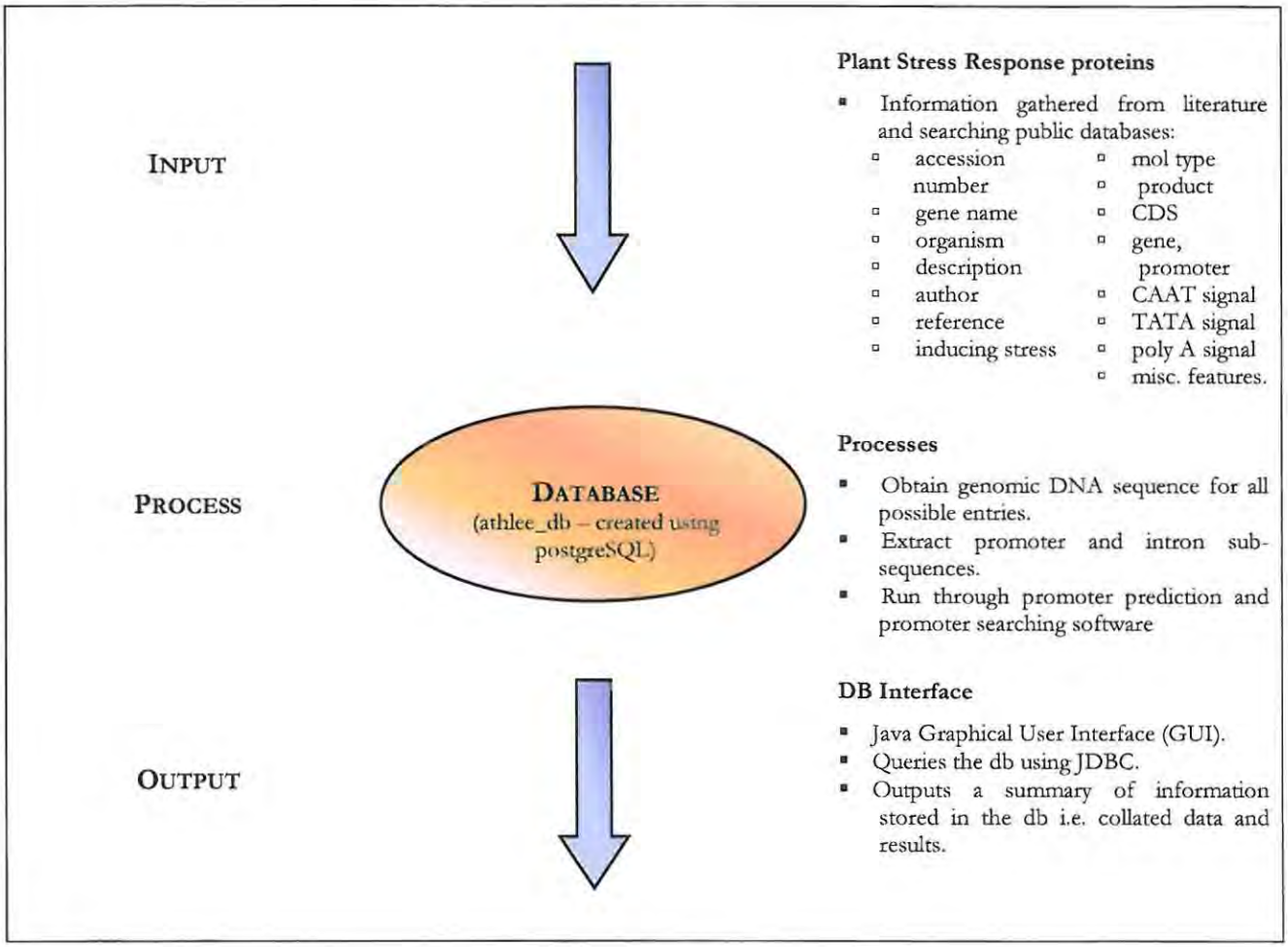

Figure 7: Project process diagram

2.5. **Promoter Prediction**

NNPP: Neural Network Promoter Prediction

- Version: NNPP2.2
- Program: fa2TDNNpred.linux – NNPP for LINUX (eukaryotic)
- URL: http://www.fruitfly.org/seq_tools/promoter.html

NNPP is an implementation of a feed-forward neural network model of the structural and compositional properties of eukaryotic and prokaryotic core-promoter regions. It uses the time-delay neural network architecture to make promoter predictions for DNA sequences and is applied to the question of predicting potential transcription start sites in genomic sequences (Reese, 2000; Fickett and Hatzigeorgiou, 1997; URL 3).

2.5.1. Time-delay Neural Networks (TDNN)

Neural networks are machine-learning methods that originally stem from the field of signal and speech recognition. Promoter modelling uses a particular type of neural network, the time-delay neural network (TDNN) architecture, initially developed for the analysis of speech sequence patterns in time series with local time shifts. Sequence patterns are transformed into input patterns by extracting a sub-sequence within a fixed window. The window is then moved over all positions in the sequence and the sub-sequences converted into input activities (Reese, 2000).

Essential promoter specific features that have to be learned by the network include (Reese, 2000):

- Recognition of sub-sequences occurring at non-fixed positions within the input window i.e. the sub-sequence is feature-independent of shifts in its position.
- Recognition of features albeit their appearance at various relative positions.

2.5.2. NNPP

NNPP combines two distinct neural networks, one for the TATA-box recognition and one for the *Inr* recognition, which were trained independently. During this initial training the statistical properties of the nucleotides in the binding sites were learned. The combined two-layered time-delay neural network then incorporates these individual networks and captures possibly important dependencies between the individual binding sites. The 51 base pair sequence, that spans the transcription start site from position –40 to +11 and includes the TATA-box and *Inr*, is used as input to the final TDNN. The initial single-feature time-delay neural networks are transferred into the combined TDNN and training is performed. The resulting neural network logs high order correlation between the various features and their relative distance into a complex weight matrix (Reese, 2000; Fickett and Hatzigeorgiou, 1997; URL 3).

NNPP was chosen as the promoter prediction tool of choice as it is freely available for download to academic institutions, thus could be installed and run locally, and was rated as the highest performing promoter prediction software tool by Fickett and Hatzigeorgiou (1997) in a thorough review of eukaryotic promoter prediction algorithms.

2.6. **Promoter Searching**

TESS: Transcription Element Search Software on the WWW, (Schug and Overton, 1997; URL 17).
URL: http://www.cbil.upenn.edu/tess

TESS is a web-based software tool that locates putative transcription factor binding sites (*cis*-elements) in a DNA sequence. The program performs string, filtered-string and weight matrices searches of a DNA sequence based on TRANSFAC and/or user-defined sequences, allowing mismatches and variable cut-offs (Schug and Overton, 1997; URL 3).

2.6.1. Database Preparation

TESS makes use of the TRANSFAC database as its transcription factor reference data source. TRANSFAC is a store of transcription factor binding sites, consensus sequences, and weight matrix data collated from the literature and maintained as a relational database (Schug and Overton, 1997; URL 21; URL 3; Rombauts *et al.*, 2003).

TRANSFAC is distributed in flat file and relational form. The flat-file form is composed of six EMBL files corresponding to six of the main relational tables. TESS pre-processes these flat files to generate a set of indexed tables, thus facilitating a much more rapid access and querying process (Schug and Overton, 1997).

2.6.2. Binding Site Searches

TESS performs two types of searching corresponding to the two forms of data accessed from TRANSFAC.

- String searching – uses the information in the SITES table
- Weight matrix searching – uses the information in the MATRIX table

Both searches attempt all alignments $A \in \mathcal{A}$ of a model of the binding site with the input sequence. The definition of 'model' is dependant on the search method being used. N and W_M indicate the length of the input sequence and model respectively. The string matching algorithm requires that the model lie entirely within the input sequence. Alignments are allowed in both the normal and reverse-complement sense, thus there are $2(N - W_M + 1)$ possible alignments for each model (Schug and Overton, 1997).

2.6.2.1. String Searching

The majority of transcription factors held in the FACTOR table of TRANSFAC are related to one or more entries in the SITES table indicating short sequences of DNA containing a binding site for a factor. During string searching, the model of the factors' binding site is merely the set of sequences taken searched one after the other. The sequence set for each factor is collected and ordered according to length, longest to shortest, prior to each search. The search program calculates the score of each legal alignment for every sequence. The positions of the sequence covered by the alignment are marked when an alignment score exceeds the minimum cut-off threshold. Subsequent alignments of shorter sequences that overlap with the marked bases are considered illegal and are not evaluated. All bases are released prior to searching for each new factor, thus the longest match that can be made to a specific location, given the cut-off parameter, is recorded (Schug and Overton, 1997).

The user is able to specify the following search parameters (Schug and Overton, 1997):

- Percentage mismatches allowed (recommended/default value = 10%)
- Minimum sequence length (default value = 6) – matches to sites that are too short are less significant and are likely to be excluded by a match with a longer sequence.
- Minimum score (recommended/default value = 12.0)
- Secondary scoring threshold – used in the Java display to highlight the very good matches.

TESS was chosen as the promoter searching tool of choice as it allows a filtered search of TRANSFAC (results are filtered according to a user-selected parameter e.g. organism classification) as well as the option to input user-defined search sequences.

2.7. **Database Interface**

2.7.1. Graphic Display

Java is a high-level, object-orientated programming language developed by Sun Microsystems in the early 1990s. It is accompanied by a library of extra software, the Java API, that provides the ability to create graphics, communicate over networks, interact with databases and so on.

A Graphical User Interface (GUI) is a program interface or well-defined layout of interactive graphical components, that utilises the computer's graphics capabilities in order to facilitate easy use of the program (Lewis and Loftus, 2003; URL 40).

A graphic display was created in JAVA to allow a user to search athlee_db (facilitated by the JDBC Java API – detailed below in section 2.7.2) and to display a concise summary of the search results.

2.7.2. Database Querying

Java Database Connectivity (JDBC) is a Java API that facilitates the execution of SQL statements by Java programs, thus allowing a java program to interact with any SQL-compliant database (URL 40). The PosgreSQL JDBC Driver enables a Java program to connect to a PostgreSQL database, using standard Java code (URL 9).

The JDBC API is considered the industry standard for database-independent connectivity between Java and a wide variety of databases. The JDBC API basically allows the user to do three things (URL 8):

- Establish a database connection
- Send SQL statements
- Process the results

CHAPTER 3 IMPLEMENTATION*

3.1. Database Design, Creation and Population

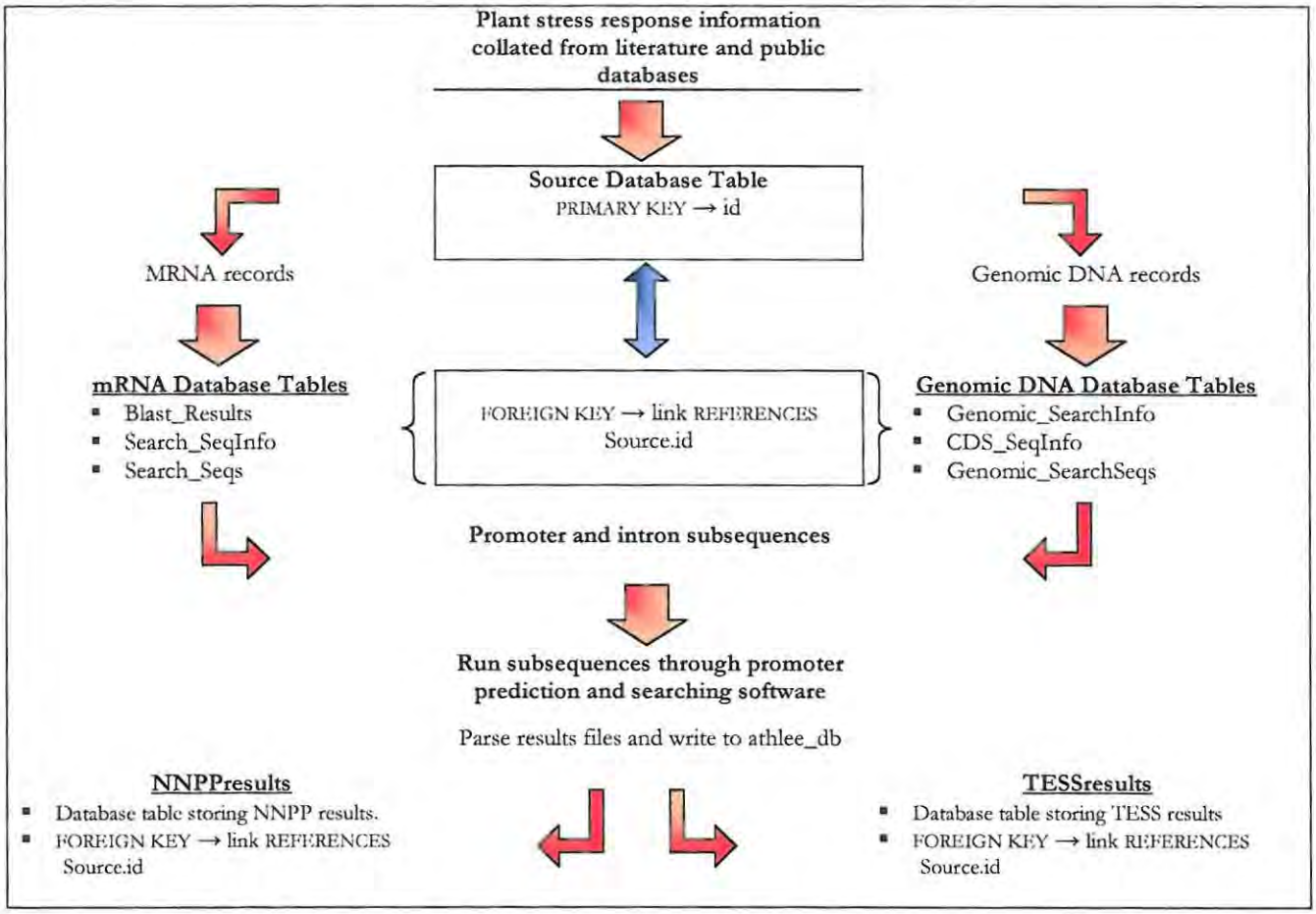

Figure 8: Outline of the flow of information through the database

An initial working set of fifty genes encoding for proteins that have been reported to be up- or down-regulated in response to drought, low temperature, and high salinity plant stresses, were identified by manually searching the available literature, and public databases. The search process could unfortunately not be automated as there seems to be no consistency in the nomenclature used i.e. dehydration, drought, water-deficit and water stress are used interchangeably when referring to plant stress associated with reduced water availability.

The accession numbers were stored in a text file, and a perl script (Populate.pl – Appendix A1) written which reads these into an array[1], assigns a unique identifier to each record[2] (the first two characters of the accession number and a number that increments with the addition of each new record), and populates the id and accession fields[3] of the primary table in the database, the Source table. Accession numbers were not used as

* Superscript numbers in the text preceding each figure in this section correspond to those in the highlighted sections of Perl code in the figure that follows.

the primary key as some of the flatfiles contained more than one sequence of interest, thus the accession number was not a unique identifier for each record.

```
...
1. open (accessions, "SourceAccessions");
   @accessions = <accessions>;

2. for ($num = 0; $num < scalar@accessions; $num++){
       chomp @accessions[$num];
       $acc = @accessions[$num];
       $id = substr($acc,0,2) . "_" . $num;

3.     $res=$conn->exec("INSERT INTO Source (id, accession) VALUES ('$id', '$acc');");
   }
...
```

Figure 9: Highlighted sections from Populate.pl

Table 4 below outlines the structure of the Source table as well as the format and description of the information it was created to store. Similar summaries of the auxiliary tables are detailed in the appendices section of this report.

Table 4: Source Database Table Outline

Field Name	PostgreSQL Field Format	Description
Id	Text PRIMARY KEY	Unique identifying number for each new record generated by populate.pl. Foreign key of auxiliary tables reference Source.id
Accession Number	varchar (40)	Obtained from NCBI nucleotide database flat-file
Organism	varchar (60)	e.g. Arabidopsis thaliana
Gene Name	varchar (60)	Obtained from NCBI nucleotide database flat-file
Description	Text	Description of nucleotide
Author	Text	Primary author(s)
Reference	Text	Journal, vol (no.), pp (year)
Inducing stress/condition	Text	Low temperature, Drought/Dehydration , High salinity, Osmotic, Abscisic Acid
Mol_type	varchar (60)	Genomic DNA, mRNA
Product	Text	Protein product
CDS	Text	Identified Coding Region
Gene	Text	Gene-region
Promoter	Text	Promoter-region
CAAT_Signal	Text	Identified CAAT_Signal
TATA_Signal	Text	Identified TATA_Signal
PolyA_Signal	Text	Identified PolyA_Signal
Sequence	Text	Nucleotide sequence
Misc_Feature	Text	Putative/suggested features i.e. *cis*-elements identified in the promoter region

PopulateSource.pl (Appendix A2) then reads the id and accession numbers from the database[1], retrieves a stream of the corresponding seqIO objects from GenBank[2], and calls a series of sub-routines[3] (Appendix A2) to extract relative information and populate the remaining fields in the Source table[3], with the exception of the 'inducing_stress' field, which was entered manually.

```
1. $conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
   $conn->reset;

   $res=$conn->exec("SELECT id FROM Source;");
   while (@id_nos=$res->fetchrow){
       foreach $id(@id_nos){
           push(@ids,$id);
       }
   }

   $res=$conn->exec("SELECT accession FROM Source;");
   while (@row=$res->fetchrow){
       foreach $acc(@row){
           push(@accession_nrs,$acc);
       }
   }
   ...
2. $gb = new Bio::DB::GenBank();
   ...
   $seqio = $gb-> get_Stream_by_acc([@accession_nrs]);
   ...
3. while($clone = $seqio->next_seq) {
       $idnum = @ids[$x];
       FetchRefAnnotations($clone, $idnum);
       FetchDescription($clone, $idnum);
       FetchSequence($clone, $idnum);

       foreach $feat_object ($clone->get_SeqFeatures) {
           FetchGeneName($feat_object, $idnum);
           foreach $k(@seqfeatures){
               FetchSeqFeature($k, $feat_object, $idnum);
           }
           foreach $l(@sourceFeatures){
               FetchSourceFeatures($l, $feat_object, $idnum);
           }
       }
       $x++;
   ...
```

Figure 10: Highlighted sections from PopulateSource.pl

3.2. Retrieval of Genomic Promoter and Intron Sequences

The nucleotide information collated in the Source table primarily occurs in two forms, namely mRNA and genomic DNA. Sections 3.2.1 and 3.2.2 describe how the afore-mentioned source information was processed in order to extract the subsequences of interest *viz.* the promoter and intron regions.

3.2.1. mRNA Records

In order to obtain the genomic DNA for the mRNA records, a perl script, remoteBlast.pl (Appendix B1), selects all mRNA records from the Source table[1], writes them in FASTA format to an input file[2], performs a remote BLAST (program: blastn, evalue: 1e-10) of each record against the non-redundant nucleotide database[3], and writes the resulting BLAST report to individual output files[4].

```
...
1. $conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
   $conn->reset;

   Pg::doQuery ($conn,"SELECT accession, organism, seq FROM Source WHERE mol_type ~* 'mRNA'",\@ary);
   open (FILEIN,">mRNAinputseq");

2. for $i(0 .. $#ary){
      @orgName = split(/ /,$ary[$i][1]);
       print FILEIN ">$ary[$i][0]_$orgName[0]_$orgName[1]\n";
       print FILEIN "$ary[$i][2]*\n\n";
   }
   ...
3. my $prog = 'blastn';
   my $db = 'nr';
   my $e_val = '1e-10';

   my @params = ( '-prog'     =>$prog,
                  '-data'     =>$db,
                  '-expect'   =>$e_val,
                  '-readmethod'=>'SearchIO');

   my $factory = Bio::Tools::Run::RemoteBlast->new(@params);
   $factory->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
   ...
   my $str = Bio::SeqIO->new(-file=>'mRNAinputseq');
   ...
   my $r = $factory->submit_blast('mRNAinputseq');

   while(my @rids = $factory->each_rid){
      foreach my $rid(@rids){
         my $rc = $factory->retrieve_blast($rid);
         if(!ref($rc)){
            if($rc<0){
               $factory->remove_rid($rid);
            }
            print STDERR "." if($v>0);
            sleep 5;
         }else{
4.          my $result = $rc->next_result();
            my $filename = $result->query_name() "\.out";
            $factory->save_output($filename);
            $factory->remove_rid($rid);
            print "\nQuery Name:", $result->query_name(),"\n";

            while(my $hit = $result->next_hit){
               next unless ($v>0);
               print "\tHit name: ", $hit->name,"\n";
               while(my $hsp = $hit->next_hsp){
                  print "\t\tScore is ",$hsp->score,"\n";
                  ...
```

Figure 11: Highlighted sections from remoteBlast.pl

The BLAST output files[1] were then parsed (parse_blast.pl – Appendix B2) in order to filter out[2] all unwanted hits. Hits were filtered on the following three selection criteria:

- Must have greater than or equal to seventy-five percent identity with the query mRNA sequence
- Must be of the same species as the query sequence
- Must not be a cDNA or mRNA record

This process reduces the number of hits per query sequence to approximately 0 – 6 per query. Selected information on each hit was then extracted[3] and written[4] to the Blast_Results table (Appendix B3) in the database.

```
...
1. my $dir = "/home/g9610081/BlastReportFiles";
   my @blastreport_files = <$dir/*.out>;
...
   my $in = new Bio::SearchIO(-format => 'blast',
                              -file   => $filename);

2. while(my $result = $in->next_result) {
     while(my $hit = $result->next_hit) {
       while(my $hsp = $hit->next_hsp) {
         if($hsp->percent_identity >= 75) {
           if($hit->description =~ /[\s\w]+($species)[\s\w]+/i) {
             if($hit->description =~/[\s\w]+(mRNA|cDNA)+[\s\w]*/i) {
             } else {
             ...

3.                $hit_name = $hit->name;
                  $hit_accession = $hit->accession;
                  $hit_description = $hit->description;
                  $hit_description =~ s/\'/_/g;
                  $length = $hsp->length('total');
                  $percent_id = $hsp->percent_identity;
                  $e_value = $hsp->evalue;
                  $start_query = $hsp->start('query');
                  $end_query = $hsp->end('query');
                  $start_hit = $hsp->start('hit');
                  $end_hit = $hsp->end('hit');

                  if ($hit_name =~/^gb/){
                      $hit_seq=FetchGBSequence($hit_accession);
                  }else{
                      $hit_seq=FetchEMBLSequence($hit_accession);
                  }
                  ...

4.                $res=$conn->exec("INSERT INTO Blast_Results (accession, hit_name, hit_accession,
                  hit_description, length, percent_id, e_value, start_query, end_query, start_hit, end_hit, hit_seq)
                  VALUES ('$accession', '$hit_name', '$hit_accession', '$hit_description', '$length', '$percent_id',
                  '$e_value', '$start_query', '$end_query', '$start_hit', '$end_hit', '$hit_seq');");
                  ...
```

Figure 12: Highlighted sections from parse_blast.pl

The hits for each record were manually reviewed and where possible a series of consecutive hits on the same genomic sequence identified. The promoter and intron start and end positions were defined and entered into

the Search_SeqInfo table (Appendix B4), e.g. the blast results for query AB039924 included six consecutive hits with the genomic DNA sequence, AC002332. The evaluation process is detailed in table 5 below.

Table 5: Example of BLAST evaluation process

Query: AB039924 Hit: AC002332						
Query	4 – 174	175 – 325	324 - 410	410 – 508	504 - 632	631 - 935
Hit	50298 – 50128	49853 – 49703	49621 – 49535	49439 – 49341	49253 – 49125	48991 – 48688
						
Promoter and Intron Subsequences						
Promoter	46688 – 48687					
Intron 1	48992 – 49124					
Intron 2	49252 – 49340					
Intron 3	49440 – 49534					
Intron 4	49622 – 49702					
Intron 5	49854 – 50127					

SearchSeqFetch.pl (Appendix B5) and FetchIntrons.pl (Appendix B6) reads these values into a series of arrays[1], retrieves the corresponding nucleotide subsequences from GenBank[2], and writes them to the Search_seqs table[3] (Appendix B7).

```
...
1. $conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
   $conn->reset;
   ...

   $res=$conn->exec("SELECT TESSprom_start FROM Search_SeqInfo;");
      while (@rows1=$res->fetchrow){
         foreach $start1(@rows1){
            push(@startnrs1,$start1);
         }
      }
   }
   ...

2. foreach $entry(@hitaccession_nrs){
     $accession=@accession_nrs[$index];
     $promoter2000=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
     $promoter1000=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);
     ...

3. $res=$conn->exec("INSERT INTO Search_Seqs (accession, hit_accession, promoter2000, promoter1000) VALUES
   ('$accession', '$entry','$promoter2000', '$promoter1000');");

   $res=$conn->exec("UPDATE Search_Seqs SET seq = Blast_Results.hit_seq where hit_accession =
   Blast_Results.hit_accession;");
   ...

   sub FetchSequence {

   my ($search_term, $startnr, $endnr) = @_;
```

```
    $gb = new Bio::DB::GenBank();
    $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $gb->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->subseq($startnr,$endnr);
    $retrieved_seq;
}
```

Figure 13: Highlighted sections from SearchSeqFetch.pl

3.2.2. Genomic DNA records

GenomicSearchInfo.pl (Appendix B8) selects all genomic DNA records from the Source table, retrieves the corresponding promoter and intron seqFeatures from GenBank[1], and writes the start and end information to the Genomic_SearchInfo table[2] (Appendix B11). Additionally, GenomicSearchInfo.pl identifies split location CDS seqFeatures[3], retrieves the start and end information for each CDS sub-location, and writes it to the CDS_SeqInfo table (Appendix B9).

1.
```
...
$gb = new Bio::DB::GenBank();
$gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
$seqio = $gb-> get_Stream_by_acc([@accession_nrs]);
...
while($clone =  $seqio->next_seq) {
    foreach $feat_object ($clone->get_SeqFeatures) {
        FetchPromoter($feat_object, $ids[$x]);
        FetchIntron($feat_object, $ids[$x]);
        FetchCDSsubseqs($feat_object, $ids[$x]);
    }
    $x++;
}
...
```

2.
```
sub FetchPromoter {

my ($featobj, $identity) = @_;
    if ($featobj->primary_tag eq "promoter"){
        $start = $featobj->start;
        $res=$conn->exec("UPDATE Genomic_SearchInfo SET prom_start = '$start' WHERE link = '$identity',");
        $end = $featobj->end;       $res=$conn->exec("UPDATE Genomic_SearchInfo SET prom_end = '$end' WHERE
        link = '$identity',");
    }
}

sub FetchIntron {

my ($featobj1, $identity1) = @_;
    if ($featobj1->primary_tag eq "intron"){
        $start1 = $featobj1->start;
        $end1 = $featobj1->end;
        @tags = $featobj1->get_all_tags();
        if ($featobj1->has_tag('number')){
            @num = $featobj1->get_tag_values('number');
            $n = @num[0];
        }else{
            $n = 1;
        }
        $field = "intron" . $n;
        $field1 = $field . "_start";     $field2 = $field . "_end";
        $res=$conn->exec("UPDATE Genomic_SearchInfo SET $field1 = '$start1' WHERE link = '$identity1',");
        $res=$conn->exec("UPDATE Genomic_SearchInfo SET $field2 = '$end1' WHERE link = '$identity1',");
    }
```

```
}

3. sub FetchCDSsubseqs {

my ($featobj2, $identity2) = @_;
my $y=1;

    if ($featobj2->location->isa('Bio::Location::SplitLocationI') && $feat_object->primary_tag eq 'CDS') {
        foreach $location ($featobj2->location->sub_Location) {
            $End = $location->end;
            $Start = $location->start;
            $StartField = "Start$y";
            $EndField = "End$y";
            $res=$conn->exec("UPDATE CDS_SeqInfo SET $StartField = '$Start' WHERE link = '$identity2';");
            $res=$conn->exec("UPDATE CDS_SeqInfo SET $EndField = '$End' WHERE link = '$identity2';");
            $y++;
        }
    }
}
```

Figure 14: Highlighted sections from GenomicSearchInfo.pl

PopulateGenomic_SearchInfo.pl (Appendix B10) uses the CDS sub-location information to define the promoter and intron start and end positions[1], and writes this information to the Genomic_SearchInfo table[2].

```
...
1. $conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
   $conn->reset;

   for ($x = 1; $x < 14; $x++){
       $y = $x + 1;
       $IntronStart = "intron" . $x ."_start";
       $IntronEnd = "intron" . $x ."_end";
       $CDS_field1 = "CDS_SeqInfo.end$x";
       $CDS_field2 = "CDS_SeqInfo.start$y";
       ...

2. $res=$conn->exec("UPDATE Genomic_SearchInfo SET $IntronStart = $CDS_field1 + 1 WHERE link =
   CDS_SeqInfo.link;");
   $res=$conn->exec("UPDATE Genomic_SearchInfo SET $IntronEnd = $CDS_field2 - 1 WHERE link =
   CDS_SeqInfo.link;");
   ...
```

Figure 15: Highlighted sections from PopulateGenomic_SearchInfo.pl

As with the mRNA records, two perl scripts, SearchSeqFetch2.pl (Appendix B12) and FetchIntrons2.pl (Appendix B13), read these values into a series of arrays, retrieve the corresponding nucleotide subsequences from GenBank, and write them to the Genomic_SearchSeqs table (Appendix B14).

3.3. **Promoter Prediction and Searching**

3.3.1. Promoter prediction using NNPP

PromoterFastaFiles2.pl (Appendix C3) writes each promoter subsequence in FASTA format to individual text files[1], which are then used as input for NNPP (Neural Network Promoter Prediction – outlined in

section 2.5). The NNPP eukaryotic program, fa2TDNNpred.linux, was used with a threshold confidence value of 0.95. Reese (2000) reports false positive rates of 0.39% and 0.077% at confidence levels of 0.8 and 0.99 respectively for a set of representative eukaryotic sequences. A confidence level of 0.95 was thus chosen so as not to eliminate possibly interesting results whilst still maintaining a low false positive rate (approximately 0.14%).

```
1. $conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
   $conn->reset;

   Pg::doQuery($conn,"SELECT accession, hit_accession, promoter2000, link FROM Search_Seqs;",\@ary);

   for $i(0 .. $#ary){
       open (FILEIN, ">NNPPinputFiles\/$ary[$i][3]_mRNA_Promoter.fa");
       print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_PromoterSequence_2000bps\n";
       print FILEIN "$ary[$i][2]\n\n";
       close FILEIN;
   }
```

Figure 16: Highlighted sections from PromoterFastaFiles.pl

NNPPrun.pl (Appendix C4) reads all the NNPP input filenames into an array[1], runs each promoter FASTA sequence through NNPP[2], and writes the results to separate output files[3].

```
1. my $dir = "/home/g9610081/NNPPinputFiles";
   my @NNPP_fastaFiles = <$dir/*.fa>;

2. foreach $file (@NNPP_fastaFiles) {
       $resultFile = 'nnppRESULT_';
       $resultFile .= substr($file,30);
       open (FILEOUT, ">NNPPoutputFiles\/$resultFile");
       print FILEOUT "$resultFile\n\n";
       my $result = qx`/home/g9610081/NNPP/NNPP2.2/bin/fa2TDNNpred.linux -t 0.95 -r $file`;

3.     print FILEOUT "$result\n";
       close FILEOUT;
   }
```

Figure 17: Highlighted sections from NNPPrun.pl

The output files were then parsed by parseNNPP.pl (Appendix C5), which retrieves the highest scoring TSS (Transcription Start Site) prediction and the associated information and writes it to the NNPPresults Table (Appendix C6).

```
1. my $dir = "/home/g9610081/NNPPoutputFiles";
   my @NNPPresultFiles = <$dir/*.fa>;

2. foreach my $file (@NNPPresultFiles) {     #open file handles
       open(IN, "$file");
       open(OUT, ">>\/home\/g9610081\/NNPPresults");
```

```
...
while (<IN>) { #while reading the current NNPP report
...
    elsif ($_ =~ /^Prediction:\s( {8})\-+/) {
        push(@confidVals, $1);      #find the hit with the highest confidence value
        if ($confidVals[$x-1] > $confidVals[$index]){
            $index = $x-1; #record the index of the highest confidence value thus far
        }
    }
}
...
```

```
3. $res=$conn->exec("INSERT INTO NNPPresults (link, hitno, confidence, regionstart, regionend, signalstart, pattern,
   strand, length, forwardPattern) VALUES ('$Source', '$hitno[$index]', '$confidVals[$index]', '$PatternStart[$index]',
   '$PatternEnd[$index]', '$signalStart[$index]', '$patterns[$index]', '$strand[$index]', '$lgth', '$forwardPattern');");
...
```

Figure 18: Highlighted sections from parseNNPP.pl

3.3.2. Promoter searching using TESS

PromoterFastaFiles.pl[1] (Appendix C1) and IntronFastaFiles.pl[2] (Appendix C2) write the promoter and intron subsequences in FASTA format into a series of text files. These files were used as input for the promoter searching software tool, TESS (Transcription Element Search Software – outlined in section 2.6).

```
...
1. $conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
   $conn->reset;

   Pg::doQuery($conn,"SELECT accession, hit_accession, promoter2000, link FROM Search_Seqs;",\@ary);
   open (FILEIN,">TESSinputFiles\/mRNA_Promoter2000Fastas");

   for $i(0 .. $#ary){
       print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_PromoterSequence_2000bps\n";
       print FILEIN "$ary[$i][2]*\n\n";
   }

   close FILEIN;
   ...

2. $conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
   $conn->reset;

   Pg::doQuery($conn,"SELECT accession, hit_accession, intron1, link FROM Search_Seqs;",\@ary);
   open (FILEIN,">TESSinputFiles\/mRNA_Intron1Fastas");

   for $i(0 .. $#ary){
       print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_Intron1\n";
       print FILEIN "$ary[$i][2]\n\n";
   }

   close FILEIN;
   ...
```

Figure 19: Highlighted sections from PromoterFastaFiles.pl and IntronFastaFiles.pl

Two batch searches were performed. The first run was a string search for three user-defined DNA consensus sequences of plant transcription factors obtained from literature, namely ABRE, DRE/CRT/LTRE, and

MYB (refer to section 1.3.7). The second was a filtered string search of only plant strings from TRANSFAC. Apart from filtering the second search to include only plant sequences (Organism Classification = *viridiplantae*), the default parameters were used. These are detailed in Table 6 below.

Table 6: TESS Search parameters

Search Program selected	1. String-based Search Query 2. Filtered String-based Search Query
String Databases	1. Search *My* Strings ▪ CACGTG ▪ TACCGACAT ▪ TGAACT 2. Search TRANSFAC Strings ▪ Filters: Organism Classification = *viridiplantae*
String Scoring Parameters	
▪ Use only core positions for TRANSFAC strings	Yes
▪ Maximum Allowable String Mismatch %	10
▪ Minimum log-likelihood ratio score	12
▪ Minimum string length	6
Output Control	
▪ Secondary Lg-Likelihood Deficit	3.0
▪ Count significance threshold	1.0e-2

Although TESS is a web-based analysis tool, it includes a facility to email result reports to the user. These reports were saved into a separate directory and parsed by two perl scripts, ParseTESS.pl (Appendix C7) and ParseTESS2.pl (Appendix C8), in order to extract selected information[1], calculate relative positioning of predicted TESS sites on the corresponding nucleotide sequence[2] and write it to the TESSresults table[3] (Appendix C9).

```
...
1. my $dir = "/home/g9610081/TESSreports";
   my @TESSreports = <$dir/*.tessreport>;
   ...
   foreach $report (@TESSreports) {
      open(IN, "$report");
      ...
      while (<IN>) {
         chomp;
         if ($_ =~ /^SEQ\s+(>.+)/) {
            @fileName = split(/\|/,$1);
            @fileName1 = split(/_/,$fileName[1]);
            $fileNm = $fileName1[0] . "_" . $fileName1[1];
            open(OUT, ">>TESSoutputFiles\/$fileNm.tess");
            @heading = split(/_/, $fileName[$#fileName]);
         } elsif ($_ =~ /^HIT/) {
```

```
            @fields = split(/\s+/,$_);
            print OUT "$Source $seqTable $table $heading[1] $fields[2] $fields[3] $fields[4] $fields[5] $fields[6]
            $fields[7] $fields[8] $fields[9] $fields[10] $fields[11] $fields[12] $fields[13]\n\n";
        }
    }
    close (IN);
    close (OUT);
}
```

2.
```
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

my $dir = "/home/g9610081/TESSoutputFiles";
my @TESSoutputFiles = <$dir/*.tess>;

foreach $file (@TESSoutputFiles) {
    open(IN, "$file");
    while (<IN>) {
        @fields = split(/ /,$_); #split each line into an array
        $table = $fields[2]; #table

        if ($fields[3] eq 'PromoterSequence') {
            $fieldName = 'TESSprom'; #determine db field name
        }else{
            $fieldName = $fields[3];
        }
        $fieldStart = $fieldName . "_start";
        $fieldEnd = $fieldName . "_end";

        #read start position of promoter on original sequence
        $res=$conn->exec("SELECT $fieldStart FROM $table WHERE link = '$id';");
        $start = $res->fetchrow;

        #calculate start position of site on the original sequence
        $site_start = $start + $fields[6] - 2;
        ...
```

3.
```
        ...
        $res=$conn->exec("INSERT INTO TESSresults (link, site_name, source, site_start, site_length, sequence)
        VALUES ('$id', '$TRANSFACsite', '$Source', '$site_start', '$length', '$seq');");
    }
    close (IN);
}
```

Figure 20: Highlighted sections from ParseTESS.pl and ParseTESS2.pl

3.4. Database Graphical User Interface

A Graphical User Interface (GUI) was created using JAVA and JDBC, in order to facilitate easy searching of the athlee_db and concise display of the resulting information.

Figure 21 and 22 are schematic diagrams of the Search and SearchResults GUI respectively and indicate the primary panels forming the framework of the GUI and the layout managers controlling the positioning of the components held within them.

The search GUI allows the user to select a field name from the Source, TESSresults and NNPPresults tables, and enter search terms in the text fields provided (refer to Appendix D1 – DBsearch.java and Appendix D2 – DBsearchGUI.java). Clicking the 'Start Search' button at the bottom left of the screen triggers the ButtonListener class to call the Search.java class (Appendix D3).

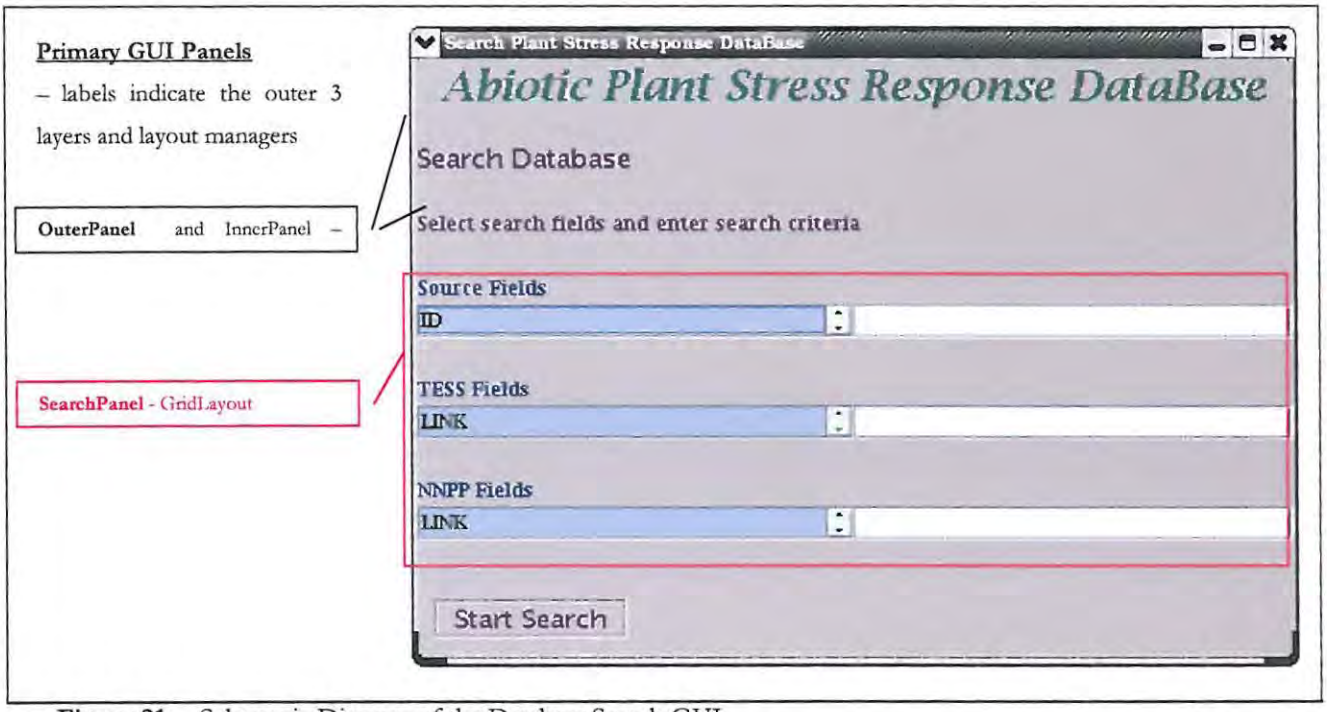

Figure 21: Schematic Diagram of the Database Search GUI

Search.java performs a search of the database using the keywords and fields selected by the user, returns the result as a list of record ids stored in an array, and triggers the search results GUI window to open, displaying the list of ids in the top left corner of the screen, as shown in figure 22 (refer to Appendix D4 – DBsearchResults.java).

The user now selects an id from the list, and clicks the 'Display Results' button. This triggers the ButtonListener to call the RetrieveSourceInfo, RetrieveTessInfo, and RetrieveNnppInfo classes (Appendices D5, D6 and D7) to extract a summary of information for the selected record from the corresponding database tables and display the results in the appropriate display components on the screen.

In order to display the results of another record in the result set, the user need only repeat the process; alternatively they may close the GUI window and rerun DBsearchGUI.java to perform another search.

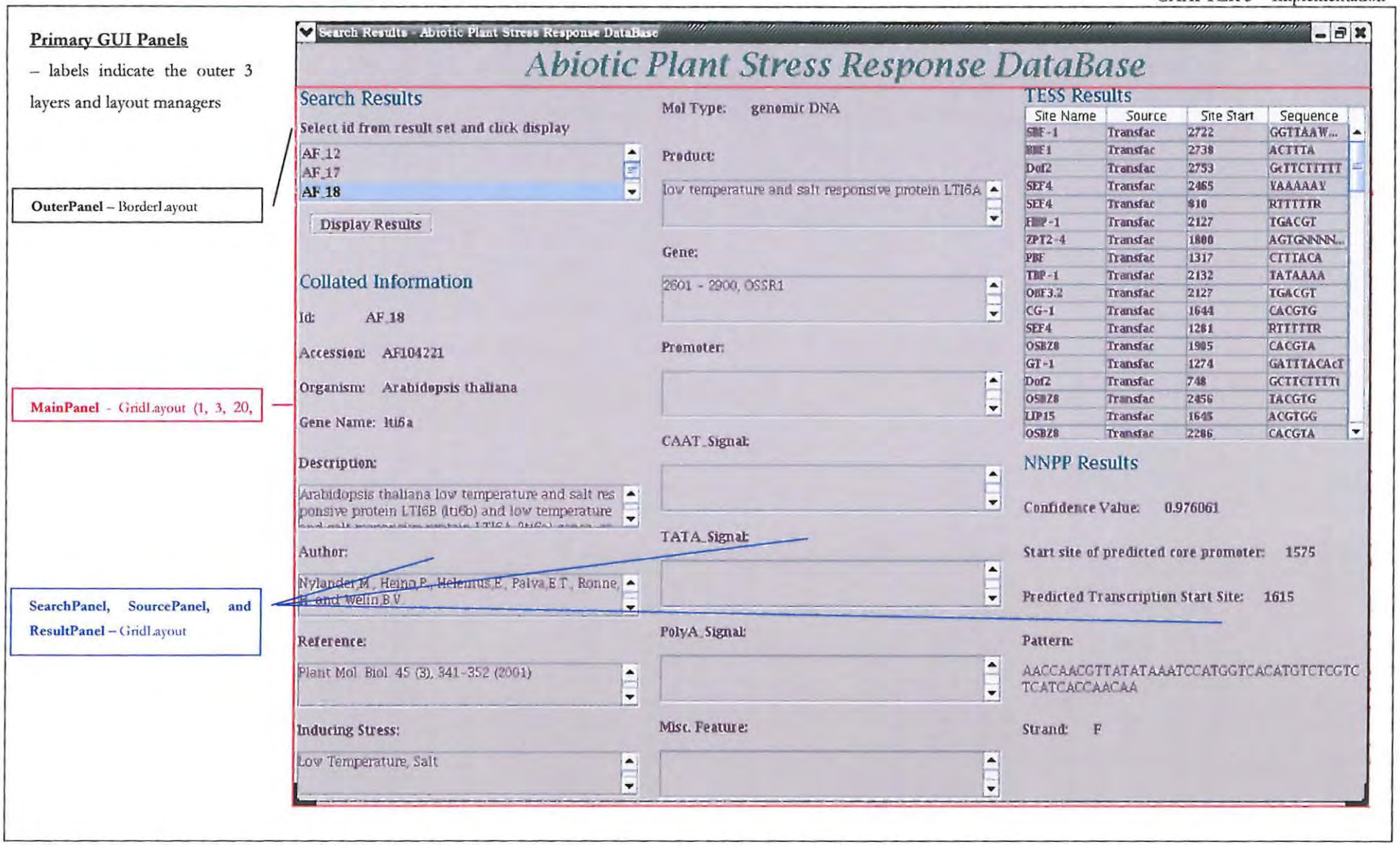

Figure 22: Schematic Diagram of the Database Search Results GUI

CHAPTER 4 RESULTS AND DISCUSSION

4.1. Database

The database was created in postgreSQL and initially populated with fifty entries obtained from literature and public databases. Entries were selected to represent genes reported to be up- or down-regulated in response to dehydration, low temperature and high salinity, as well as those reported to be regulated by a combination of the afore-mentioned abiotic plant stresses. The composition of the nucleotide sequence information of the dataset was approximately 3:2 mRNA:genomic DNA. The reason for this ratio is due to the limited amount of genomic DNA information available, which is primarily from plant genomes that have been completely sequenced, namely *A. thaliana* and *O. sativa*. mRNA entries were thus included to provide a dataset that was more representative of all plant species that contain genes regulated in response to abiotic plant stress.

Flow of information through athlee_db was, for the most part, automated using the various perl scripts discussed in chapter 3, however there were three stages where manual processing was necessary *viz.* the initial search process, BLAST hit evaluation and TESS batch runs. Future work will involve increasing the data set to include all known genes encoding for proteins that have been reported to be up- or down-regulated in response to drought, low temperature, and salinity plant stresses, at which stage these manual procedures will create significant "bottle-necking" of data throughput. Thus high priority should be placed on future automation of these processes if at all possible, in order to facilitate high throughput screening of new entries.

4.2. Promoter Prediction

Transcriptional initiation appears to be controlled by the coordinated binding of various proteins to *cis*-elements in the promoter, and in some cases to enhancer sequences. The combination, order, and positioning of these binding sites relative to one another and to the transcription start signal (TSS) are thought to suggest the particular expression context of the gene in question (Fickett and Hatzigeorgiou, 1997; Terai *et al.*, 2004).

The primary objective of this project was to determine whether or not drought, low-temperature and high salinity plant stress response proteins have similar *cis*-elements in their promoter regions, as a basis for attempting to deduce possible common transcriptional initiation of these genes. It was for this reason that we chose to include prediction of the transcription start signal.

Figure 24 depicts the relative position of the NNPP predicted Transcription Start Signals upstream of the gene start site. These points were obtained by subtracting the position of the predicted TSS (information held in the NNPPresults table) from the end of the promoter region (information held in the genomic_searchInfo and search_seqInfo tables for genomic DNA and mRNA records respectively), and then plotting against the

relative confidence value. The corresponding postgreSQL query statements are highlighted in figure 23 below - the sections highlighted in red indicate where the relative upstream position was retrieved from the database.

- SELECT nnppresults.link, nnppresults.confidence, genomic_searchinfo.tesspromend - nnppresults.signalstart FROM genomic_searchinfo, nnppresults WHERE genomic_searchinfo.link = nnppresults.link;
- SELECT nnppresults.link, nnppresults.confidence, search_seqinfo.tesspromend - nnppresults.signalstart FROM search_seqinfo, nnppresults WHERE search_seqinfo.link = nnppresults.link;

Figure 23: PostgreSQL database query statements used to retrieve NNPP predicted TSS information

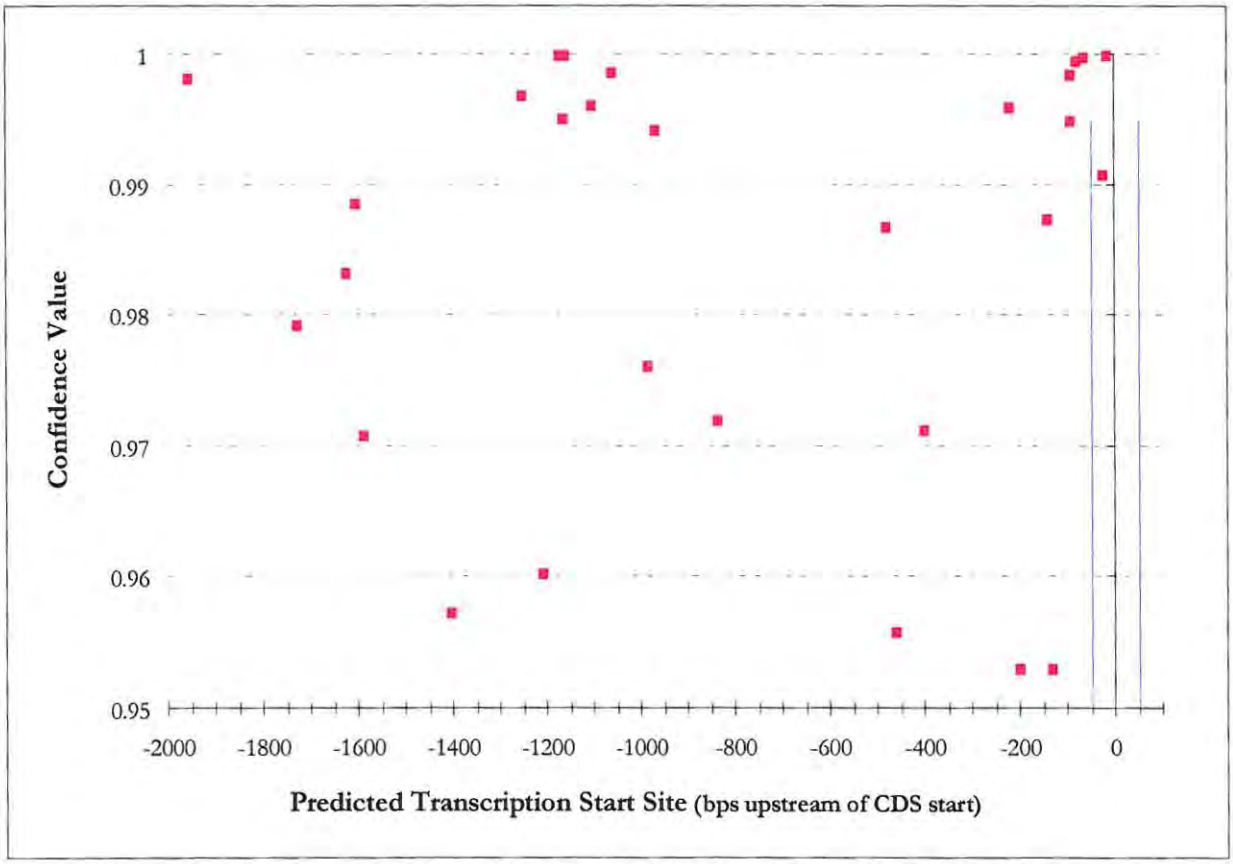

Figure 24: Relative upstream position of NNPP predicted Transcription Start Sites of sequences stored in athlee_db

Currently available promoter prediction software analysis tools are reported to predict one promoter per kilobase of DNA, whilst functional promoters are thought to only occur one in 30-40 kilobases, which indicates that a large percentage of predictions are likely to be false positives (Pedersen *et al.*, 1999; Rombauts *et al.*, 2003). NNPP predicts the TSS by combining TATA box and *Inr* recognition using a time-delay neural network architecture. These basal promoter features are known to occur in the region approximately 40 bps up- and downstream of the TSS (Reese, 2000; Fickett and Hatzigeorgiou, 1997; Pedersen *et al.*, 1999). The two vertical blue lines in figure 24 delimit the area 50 bps up- and downstream of the gene start site, thus the

NNPP predicted TSSs lying within or near this region are more likely to be accurate than those that are farther removed.

The core promoter is a binding region for RNA polymerase and the general transcription factors (GTFs). RNA polymerase II and the GTFs assemble together to form a pre-initiation complex with the core promoter in order to initiate basal transcription (Pedersen *et al.*, 1999). Figure 25 below is a diagrammatic representation of this complex, adapted from Pedersen *et al.* (1999).

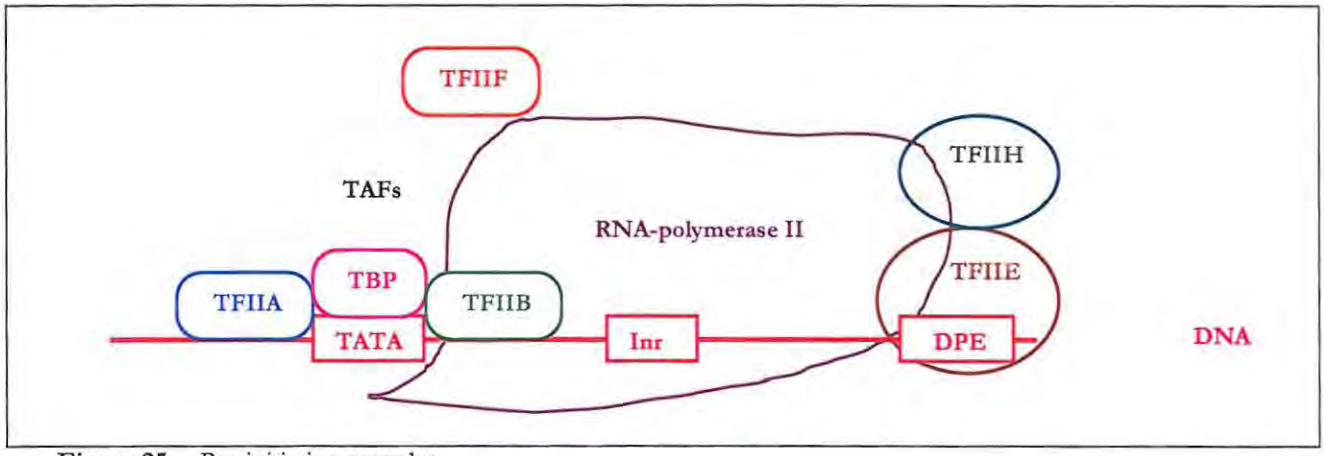

Figure 25: Pre-initiation complex
(adapted from Pedersen *et al.*, 1999).

Other promoter prediction algorithms approach promoter prediction by combining the recognition of this minimal set of transcription factor binding sites, with some description of the arrangement of these sites (Fickett and Hatzigeorgiou, 1997). TESS results identified two of these, namely TBP and TAF-1 (highlighted in yellow in Table 7 below), thus future promoter analysis could possibly include combining the NNPP results with those from a second promoter prediction tool of the afore-mentioned type, to provide clearer, more accurate results.

4.3. Promoter Searching

Table 7 below is a summary of the transcription factors predicted to bind to associated *cis*-elements in the promoter and intron regions of sequences stored in the database. This list was obtained from the following postgreSQL query statement, "SELECT DISTINCT site_name FROM tessresults;" and the associated information collated from the corresponding TRANSFAC flatfile (URL 21). The records highlighted in grey indicate those transcription factors where gene expression has been reported (in TRANSFAC) to be regulated, induced or repressed by environmental plant stresses/conditions, plant hormones and/or plant developmental stage.

Table 7: Summary of Transcription Factors predicted by TESS to bind in the promoter and intron regions of sequences stored in athlee_db
(Information collated from TRANSFAC flatfiles – URL 21)

Factor	Full Name	TRANSFAC Accession Number	DNA-binding domain	Induced/Repressed
BBF1	rolB domain factor	T02695	Dof-type zinc-finger	Auxin
BZIP910	BZIP910	T02789	Leucine zipper	-
BZIP911	BZIP911	T02790	Leucine zipper	-
CG-1	CG-1	T00126	-	-
CPRF-1	Common plant regulatory factor 1	T01091	Leucine zipper	Light
Dof2	DNA-binding with One Finger 2	T02690	Dof-type zinc-finger	-
EBP	Ethylene-responsive element binding protein	T02658	AP2 domain	Ethylene
EmBP-1b	EmBP-1b	T01098	Leucine zipper	ABA
GAmyb	Gibberellin-regulated Myb	T02679	-	Gibberellin
GBF1	G-box binding factor 1	T01078	Leucine zipper	Blue light
GBF12	G-box binding factor 12	T02676	Leucine zipper	-
GBF2	G-box binding factor 2	T01079	Leucine zipper	Blue light
GT-2	-	T01097	-	Light
HBP-1	-	T00354	Leucine zipper	-
HBP-1a	-	T00937	Leucine zipper	-
HBP-1a(1)	-	T01394	Leucine zipper	-
HBP-1a(c14)	-	T01395	Leucine zipper	-
HBP-1b	-	T00938	Leucine zipper	-
LIP15	Low temperature-induced protein 15	T02803	Leucine zipper	▪ Low temperature ▪ ABA ▪ Salt ▪ Anaerobiosis
MNB1a	-	T01059	Dof-type zinc-finger	Light
OBF3.1	Ocs element binding factor 3.1	T02662	Leucine zipper	-
Opaque-2	-	T00668	Leucine zipper	-
OBF3.2	Ocs element binding factor 3.2	T02663	Leucine zipper	-
OBF4	Ocs element binding factor 4	T02659	Leucine zipper	Ethylene
OBF5	Ocs element binding factor 5	T02661	Leucine zipper	-
OSBZ8	Oryza sativa bZIP protein 8	T02807	Leucine zipper	▪ Dehydration ▪ Development ▪ ABA ▪ Salt
PBF	Prolamin box binding factor	T02692	Dof-type zinc-finger	-

Factor	Full Name	TRANSFAC Accession Number	DNA-binding domain	Induced/Repressed
Putative ABRE	ABA Responsive Element Consensus Sequence	Taken from literature (refer to Table 2, section 1.3.7)	Basic Leucine Zipper (bZIP)	ABA
Putative DRE/CRT /LTRE	Dehydration Responsive Element/ C-repeat/ Low Temperature Responsive Element Consensus Sequence	Taken from literature (refer to Table 2, section 1.3.7)	AP2 (APETALA2) EREBP domain	▪ Dehydration ▪ Low temperature
Putative MYB-like	Myb-like Consensus Sequence	Taken from literature (refer to Table 2, section 1.3.7)	Myb-like	Dehydration
REB	-	T02808	Leucine zipper	-
RF2a	-	T02811	Leucine zipper	-
RITA-1	-	T02786	Leucine zipper	Seed development
ROM1	-	T02809	Leucine zipper	Development
ROM2	-	T02810	Leucine zipper	Development
SBF-1	-	T00739	-	-
SEF4	Soybean embryo factor 4	T01101	-	Development
SPA	Storage protein activator	T02787	Leucine zipper	Development
SsDBP-1	Single strand DNA-binding protein 1	T01178	-	-
TAF-1	-	T01090	Leucine zipper	▪ Dehydration ▪ ABA
TBP-1	TATA-binding protein 1	T00799	-	-
TGA1	-	T02794	Leucine zipper	-
TGA1a	-	T00829	Leucine zipper	Xenobiotic-stress
TGA1b	-	T00830	Leucine zipper	-
TGA2	-	T02795	Leucine zipper	-
TGA3	-	T02796	Leucine zipper	-
TGA6	-	T02797	Leucine zipper	-
ZIP-1A	-	T02804	Leucine zipper	ABA
Zmhox1a	-	T00922	Leucine zipper	-
ZPT2-1	-	T02325	Zinc-finger	-
ZPT2-2		T02447	Zinc-finger	▪ Dehydration ▪ Development ▪ Wounding ▪ Ultraviolet-B light

All the sequences stored in athlee_db are reported to be up- or down-regulated in response to one or more of the abiotic plant stresses of interest *viz.* dehydration, low temperature and high salinity. Additionally, many are reported to be up- or down-regulated in response to ABA. This information was stored in the 'inducing_stress' field of the Source table in athlee_db.

Table 8 below summarises these inducing stresses (as reported in literature) and the transcription factors predicted to bind to associated *cis*-elements in the promoter and intron regions of sequences reported to be regulated in response to them.

This information was retrieved from the athlee_db with the following postgreSQL query statement:

SELECT DISTINCT tessresults.site_name, source.inducing_stress FROM tessresults, source WHERE source.inducing_stress ~* '**$Inducing Stress**'[1] AND source.id = tessresults.link;

Table 8: TESS sites found on promoter/intron sequences and associated abiotic stresses

	High Salinity	ABA	Dehydration	Low Temperature
BBF1	X	X	X	X
BZIP910		X	X	X
BZIP911	X	X	X	X
CG-1	X	X	X	X
CPRF-1	X		X	X
Dof2	X	X	X	X
EBP	X	X	X	
EmBP-1b			X	
GAmyb			X	
GBF1	X	X	X	X
GBF12			X	
GBF2	X		X	X
GBF9			X	X
GT-1	X	X	X	X
GT-2			X	
HBP-1	X	X	X	X
HBP-1a		X	X	X
HBP-1a(1)			X	X
HBP-1a(c14)	X	X		
HBP-1b	X		X	X
LIP15	X	X	X	X
MNB1a	X	X	X	X
OBF3.1	X	X	X	
Opaque-2		X		X
OBF3.2	X		X	X
OBF4	X			
OBF5			X	
OSBZ8	X	X	X	X
PBF	X	X	X	X
Putative ABRE	X	X	X	X
Putative DRE/CRT/LTRE			X	
Putative MYB-like	X	X	X	X
REB	X	X	X	X
RF2a		X		
RITA-1	X	X	X	X
ROM1		X		

[1] $Inducing Stess indicates where each inducing factor/abiotic stress indicated in table 8, was inserted for each query i.e. high salinity, ABA, dehydration, low temperature, osmotic.

	High Salinity	ABA	Dehydration	Low Temperature
ROM2			x	x
SBF-1	x	x	x	x
SEF4	x	x	x	x
SPA	x	x	x	x
SsDBP-1	x	x		
TAF-1	x	x	x	x
TBP-1	x	x	x	x
TGA1	x	x	x	x
TGA1a			x	
TGA1b		x		
TGA2	x		x	
TGA3				x
TGA6	x		x	
ZIP-1A			x	x
Zmhox1a	x	x	x	x
ZPT2-1	x	x	x	
ZPT2-2	x		x	x

The phyto-hormone, abscisic acid (ABA), is a key signal in response to environmental stresses relating to cellular water deficit, such as dehydration, high salinity and low temperature (Shinozaki and Yamaguchi-Shinozaki, 2000; Grover *et al.*, 2001; Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). Dehydration is thought to elicit the accumulation of ABA, which then activates various stress-associated genes (Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). Studies to date regarding the molecular events triggered in response to water deficit suggest a complex network of regulatory systems that mediate stress-induced gene expression, that involve both ABA-dependent and ABA-independent signal transduction pathways. The slower ABA-dependent response is thought to involve mediation via a bZIP/ABRE (ABA responsive element) *cis*-element, which depends on the accumulation of endogenous ABA to trigger the signalling cascade (Kirch *et al.*, 2002).

Examples of ABA Responsive Elements (ABRE) taken from literature, as laid out in table 2, namely EmBP-1, TAF-1, OSBZ8, were identified by TESS in the promoter regions of genes reported to be induced under afore-mentioned abiotic stress conditions (refer to records highlighted in green in table 8 above). EmBP-1 was predicted to bind in the promoter region of genes regulated in response to dehydration, whilst TAF-1 and OSBZ8 were predicted to bind in the promoter region of genes regulated in response to ABA and high salinity, dehydration and low temperature plant stresses.

These results suggest that the ABRE *cis*-element may mediate gene expression in response to low temperature and high salinity plant stresses, as well as in response to dehydration. Alternatively, it may indicate that similar genes are regulated in response to one or more of the afore-mentioned stress albeit by different regulatory mechanisms.

Research done by Shinozaki and Yamaguchi-Shinozaki (2000) on the *Arabidopsis rd29A* promoter suggests that the second of these hypotheses is more likely. The *rd29A* promoter was found to contain both the DRE and ABRE *cis*-elements. These are reported to function in ABA-independent and ABA-dependent gene expression respectively, which indicated that *rd29A* expression was mediated by more than one independent regulatory system.

LIP15 (low temperature induced protein 15) was identified by TESS in the promoter region of genes reported to be induced by ABA and high salinity, dehydration, and low temperature plant stresses. In addition, the putative ABRE and Myb-like (consensus sequences taken from literature – refer to table 2, section 1.3.7) were identified in the promoter region of genes reported to be induced by ABA and high salinity, dehydration, and low temperature plant stresses. These results therefore further corroborate the hypothesis that there is considerable overlap in the mechanisms governing the regulation of genes induced in response to the aforementioned abiotic stresses.

A large number of the transcription factors predicted to bind in the promoter and intron regions of our data set are of the basic leucine zipper family. Members of this family of transcription factors exhibit differential, although often overlapping, binding preference for *cis*-elements containing an ACGT core sequence. Binding specificity is determined by the nucleotides flanking the core ACGT sequence. ACGT elements have been associated with abscisic acid, auxin, wounding, salicyclic acid, and anaerobiosis and with the expression of genes involved in light-regulated processes e.g. photosynthesis, and flavonoid (antioxidant) biosynthesis (Meier and Gruissem, 1994; Martinez-Garcia *et al.*, 1998). A major group of these bZIP transcription factors bind to G-box *cis*-elements which contain the palindromic CACGTG motif (Meier and Gruissem, 1994; Martinez-Garcia *et al.*, 1998). The ABA-responsive element (ABRE) is the best-known G-box *cis*-element in relation to dehydration stress tolerance (Kirch *et al.*, 2002).

Martinez-Garcia *et al.* (1998) suggest that a number of bZIP transcription factors may be able to interact with any particular ACGT target, and that the result of such multiple interactions depends on the relative affinity of each bZIP factor for the ACGT target, the relative abundance of these factors under prevailing environmental conditions or at each phase of development, as well as the availability of other interacting proteins.

One can thus speculate that differential specificity for the ACGT target sequence plays a central role in the expression of abiotic stress responsive genes. Table 9 below outlines selected examples taken from athlee_db where various ACGT-binding transcription factors induced under different conditions were predicted by TESS to bind at overlapping positions in the same promoter. This data seems to support the hypothesis that

differential specificity for the ACGT target sequence plays a central role in regulating the expression of abiotic stress responsive genes, under different environmental conditions, at specific stages in development and in response to various physiological cues.

Table 9: Examples taken from athlee_db that support the hypothesis that the ACGT target sequence plays a central role in the expression of abiotic stress responsive genes

Database Id	Site Name	Source	Site start	Site length	Sequence
D1_2	RITA-1	TRANSFAC	5336	10	GTG**ACGT**CAC
	OSBZ8		5367	6	T**ACGT**G
S7_27	RITA-1	TRANSFAC	62	10	GAC**ACGT**GtC
	EmBP-1b		61	11	GGAC**ACGT**GgC
AJ_25	RITA-1	TRANSFAC	670	10	GTG**ACGT**GGc
	LIP15		671	6	**ACGT**GG
X8_43	GBF1	TRANSFAC	22021	8	CC**ACGT**GG
	Putative ABRE	User Defined	22022	6	C**ACGT**G

Transcription factors induced by environmental and physiological cues other than abiotic stresses relating to water deficit, were also predicted by TESS to bind to *cis*-elements within the promoter region of some of the gene sequences stored in athlee_db (highlighted in grey in table 7 above). Those that are worth mentioning *viz.* gibberellin, light, development and auxin, are outlined in more detail in sections 4.3.1 – 4.3.4 and possible links to water stress response discussed in the text that follows.

4.3.1. Gibberellin

The transcription factor, GAmyb, was predicted by TESS to bind to its associated *cis*-element, GARE (Gibberellin Response Element), in the promoter region of genes regulated in response to dehydration. The expression of GAmyb is reported to be induced by gibberellin (GA) (URL 21; Cercos *et al.*, 1999; Gubler *et al.*, 1999).

Gibberellins are a large family of phyto-hormones that control an array of plant development processes e.g. fruit development, seed germination and maturation, stem elongation and leaf expansion (Cercos *et al.*, 1999; Sun, 2000; Olszewski *et al.*, 2000; Ishida *et al.*, 2004).

Sun (2000) suggests that the basal state of GA signalling is most probably repressive, and that a GA signal is required in order to activate the GA pathway to stimulate growth and development. Transcriptional initiation of GA responsive genes requires the transcription factor GAmyb (Cercos *et al.*, 1999; Gubler *et al.*, 1999; Sun, 2000; Olszewski *et al.*, 2000). ABA has been shown to suppress the majority of GA responses (Cercos *et al.*, 1999; Olszewski *et al.*, 2000), thus the regulation of many GA-inducible genes is hypothesised to be co-ordinately controlled by GA and ABA via the expression of GAmyb (Cercos *et al.*, 1999).

Dehydration elicits the accumulation of ABA, which then activates various stress-associated genes (Kirch *et al.*, 2002; Ramanjulu and Bartels, 2002). The second of the two ABA-dependent signalling pathways implicated in dehydration stress response (discussed in section 1.3.2) requires *de novo* protein synthesis to initiate the expression of genes induced by dehydration (Shinozaki and Yamaguchi-Shinozaki, 2000; Kirch *et al.*, 2002). In *A. thaliana*, the induction of a dehydration-responsive gene, *rd22*, is ABA-dependent, however the promoter does not include any sequence that corresponds to the consensus ABRE sequence, although it does contain recognition sites for some transcription factors i.e. MYC, MYB, and GT-1 (Iwasaki *et al.*, 1995; Shinozaki and Yamaguchi-Shinozaki, 2000).

It is thus possible that this second ABA-dependent pathway functions in the regulation of GA responsive genes involved in plant growth processes, whereby dehydration-induced ABA suppresses synthesis of GAmyb under dehydration stress conditions.

4.3.2. Auxin

The auxin-induced (URL21) transcription factor, BBF, was predicted by TESS to bind to associated *cis*-elements in the promoter region of genes regulated in response to abiotic plant stresses. Baumann *et al.* (1999) report that BBF is involved in tissue-specific and auxin-regulated gene expression.

Auxin signal transduction is known to mediate a diverse range of plant processes. These include the regulation of a number of processes during plant growth and development, e.g. cell division and elongation, abscission, flowering (Hooley, 1998), the ubiquitination of target proteins marking them for degradation (Kepinski and Leyser, 2002), and the activation of stomatal opening (Schroeder *et al.*, 2001).

One could, thus, make several deductions as to the role auxin-induced BBF plays in the regulation of abiotic stress responsive genes, depending on the function of the encoded protein. For example, *A. thaliana*, shows increased levels of mRNA encoding ubiquitin extension protein in the initial stages of drought stress. Ubiquitin extension protein is a fusion protein that is a precursor to active ubiquitin, and is derived by proteolytic processing. Ubiquitin functions in the apoptotic process by tagging proteins for degradation (Ingram and Bartels, 1996). Protein degradation plays a duel role under abiotic stress conditions in that it degrades proteins irreparably damaged by stress and is thought to function in dispensing with redundant proteins and depolymerising vascular storage peptides, thereby freeing up amino acids for use in the large-scale production of new proteins (Bray, 1993; Ingram and Bartels, 1996). One could thus speculate that the auxin-induced transcription factor, BBF, may function to upregulate abiotic stress responsive genes encoding proteins involved in protein degradation. Alternatively, one could suggest that BBF is involved in the

downregulation of genes encoding proteins facilitating plant growth and development processes in response to abiotic stress.

4.3.3. Light

The transcription factors, CPRF-1, GBF1, GBF2, GT-2, MNB1a and ZPT2-2 were predicted by TESS to bind to associated *cis*-elements in the promoter region of genes regulated in response to abiotic plant stresses. Expression of the afore-mentioned transcription factors is reported to be light-induced (URL 21).

Plants have evolved mechanisms to monitor the quality, quantity, direction and duration of light conditions in order to ensure optimal growth via photosynthesis (Kircher *et al.*, 1999; Kim *et al.*, 2004). It follows that light-induced transcription factors likely effect changes in gene expression relating to plant growth and development.

CO_2 acquisition from the atmosphere for assimilation during photosynthesis, is facilitated by stomatal opening, a process which is driven by H^+ extrusion through the plasma membrane H^+-ATPases, resulting in hyperpolarisation of the membrane which is thought to force K^+ uptake by the guard cells. Guard cells are known to respond to various physiological indicators, including red and blue light, CO_2, plant pathogens, phytohormones e.g. ABA, GA, auxin and cytokinin and other environmental cues (Schroeder *et al.*, 2001).

Stomatal opening can be activated by auxins, red and blue light (Schroeder *et al.*, 2001). Conversely, ABA which accumulates in response to water deficit results in partial stomatal closing thus inhibiting photosynthesis and minimising water loss via transpiration (Mansfield and Atkinson, 1990; Schroeder *et al.*, 2001; Chaves *et al.*, 2003). Inhibition of photosynthesis via stomatal adjustment has a two-fold benefit for the drought-stressed plant in that it reduces free radical production resulting from an excess of photosynthetic excitation energy, and water loss by transpiration (Chaves *et al.*, 2003).

Auxin and gibberellin have also been implicated in the activation of stomatal adjustment (Schroeder *et al.*, 2001) and are known to function in the regulation of various processes during plant growth development (Cercos *et al.*, 1999; Sun, 2000; Olszewski *et al.*, 2000; Hooley, 1998). As previously discussed in sections 4.3.1 and 4.3.2, auxin- and gibberellin-induced transcription factors were predicted to bind to associated *cis*-elements in the promoter region of genes regulated in response to abiotic plant stresses stored in athlee_db, thus one can speculate that these phytohormones function in a similar way to regulate growth and development via the mediation of photosynthesis.

Terzaghi *et al.* (1997) suggest that light-regulated phosphorylation of GBF in the cytoplasm may affect its nuclear uptake and consequently the activation of target genes. Phosphorylation cascades have also been

implicated in the regulation of genes in response to abiotic plant stresses, specifically Phospholipase D, which is reported to increase significantly within minutes of exposure to dehydration (Kirch *et al.*, 2002; Xiong *et al.*, 2002; refer to section 1.3.3). This points to differential expression of abiotic stress responsive genes, based on changes in cytosolic phosphorylation levels modulating the NLS activity of various transcription factors by as yet unknown mechanisms.

Three adaptive responses, namely homeostasis, growth control and stress damage control and repair, are essential for plant survival under abiotic stress conditions (Olszewski *et al.*, 2000). One can hypothesise that abiotic stress responsive genes that contain *cis*-elements in their promoters that bind light-induced transcription factors function in all of the afore-mentioned adaptive responses. Under physiological conditions, one may predict that light-induced transcription factors are allowed into the nucleus where they activate genes involved in plant growth processes. Similarly, under water stress conditions, abiotic stress-induced transcription factors are likely to function in an antagonistic manner by repressing growth. Figure 26 below is a diagrammatic representation of this hypothesis.

Bohnert *et al.* (1995) report that the abiotic stress tolerant plant species, *Mesembryanthemum crystallinum*, exhibits stomatal opening only at night thereby reducing water loss by transpiration. This points to a further explanation of our observation of light-induced transcription factor binding sites in the promoter region of genes known to be regulated in response to abiotic plant stress.

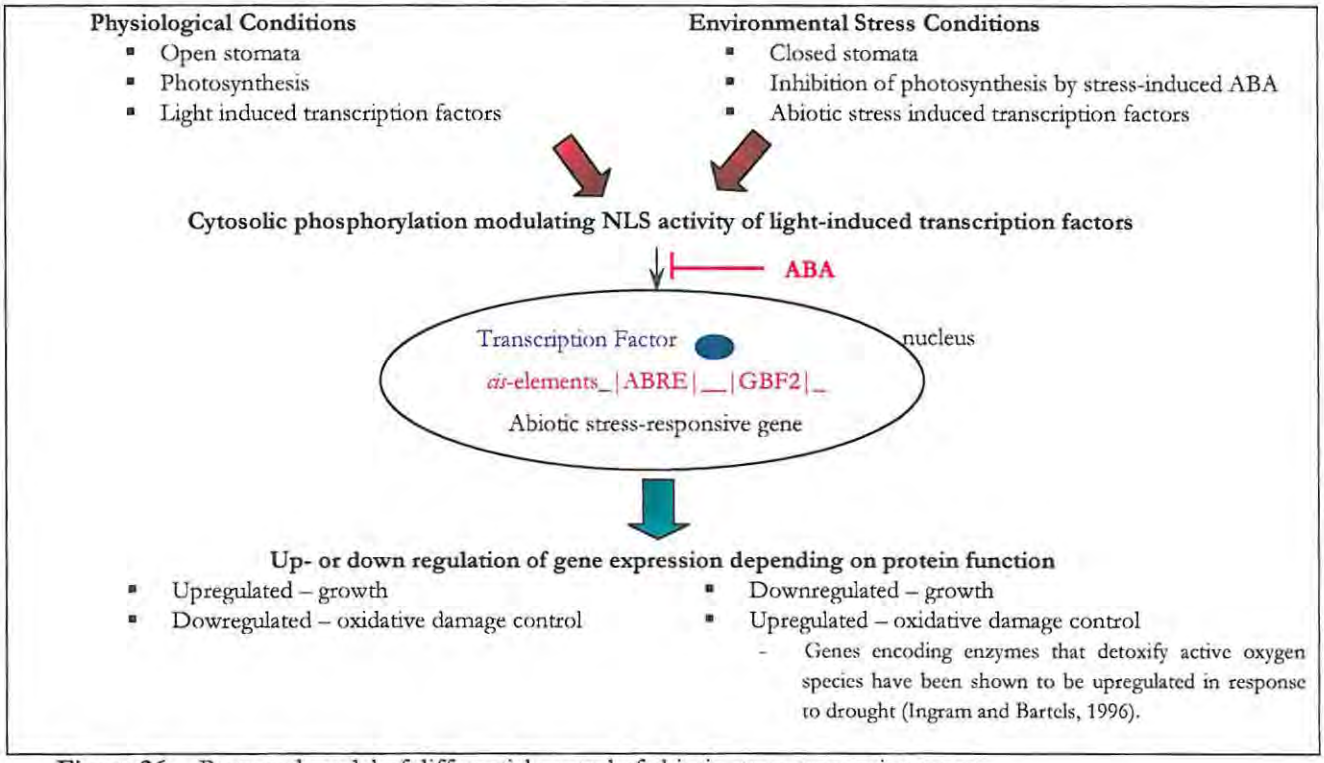

Figure 26: Proposed model of differential control of abiotic stress responsive genes

4.3.4. Development

The development-induced transcription factors RITA-1 and ROM-1 (URL 21) were predicted by TESS to bind in the promoter region of genes reported to be regulated in response to abiotic plant stresses resulting in water deficit. Izawa *et al.* (1994) report that RITA-1 is expressed in developing rice seeds, and speculate that it may function in the regulation of genes expressed during seed development. A bZIP transcription factor, ROM1, was isolated from immature bean embryos; highest levels were observed during the cotyledon stage and decreased rapidly at the onset of maturation. ROM1 was suggested to function as a transcriptional repressor of lectin and storage protein genes (Chern *et al.*, 1996).

During development, seeds undergo a process of varying histodifferentiation which is characterised by an increase in fresh mass, reserve accumulation, accompanied by a rapid rise in dry mass content, and maturation drying. During maturation drying, dry mass accumulation is halted and fresh mass decreases as there is a net water loss from the seeds. Prior to maturation drying, seeds acquire desiccation tolerance and the ability to germinate (Pammenter *et al.*, 1999).

ABA concentrations in seeds were found to increase prior to dehydration and not in response to dehydration, and functions to prevent precocious germination and promote the acquisition of desiccation tolerance (Ho, 1982). ABA levels decrease as seed development progresses, resulting in reduced tissue-sensitivity to ABA, and thus germination is no longer inhibited (Hetherington and Quatrano, 1991). Proteins responsible for desiccation tolerance are upregulated by ABA during seed maturation (Ingram and Bartels, 1996). In addition, seed germination is strongly inhibited by abscisic acid, high salinity and low water availability (Finkelstein and Lynch, 2000).

The acquisition of desiccation tolerance during seed maturation, and the inhibition of seed germination by ABA and abiotic plant stresses relating to water deficit, points to a definite overlap in the mechanisms regulating gene expression during development and in vegetative tissue exposed to abiotic stress. It is, therefore, not surprising that development-induced transcription factors were predicted by TESS to bind in the promoter region of genes reported to be regulated in response to abiotic stress, and that different transcription factors mediate expression of similar genes during seed development and in response to abiotic stress.

CHAPTER 5 CONCLUSION

A postgreSQL database was created and initially populated with fifty entries obtained from literature and public databases representative of genes reported to be up- or down-regulated in response to dehydration, low temperature and high salinity, as well as those reported to be regulated by a combination of the afore-mentioned abiotic plant stresses. Information pertaining to each of these genes was collated and stored in the database, athlee_db. Nucleotide sequences were processed in order to retrieve subsequences of interest *viz.* the promoter and intron sequences. These subsequences were run through promoter prediction and promoter searching software analysis tools (NNPP & TESS). Output files were then parsed and results written to the database, athlee_db and a java GUI created to facilitate easy searching of the database and concise display of the resulting information.

NNPP was chosen as it was rated as the highest performing promoter prediction software tool by Fickett and Hatzigeorgiou (1997) in a thorough review of eukaryotic promoter prediction algorithms, however results were less than promising as very few predicted TSS were identified in the area 50 bps up- and downstream of the gene start site, where biologically functional TSSs are known to occur (Reese, 2000; Fickett and Hatzigeorgiou, 1997).

TESS results seemed to support the hypothesis that drought, low-temperature and high salinity plant stress response proteins have similar *cis*-elements in their promoter regions, and suggested links to various other gene regulation mechanisms *viz.* gibberellin-, light-, auxin- and development-regulated gene expression, highlighting the vast complexity of plant gene regulation.

ABA responsive elements (ABREs) were found in many of the abiotic stress responsive genes stored in athlee_db, confirming the involvement of ABA-dependent signalling pathways in response to dehydration. Its presence in the promoter region of genes up- and down-regulated in response to other abiotic stresses relating to water deficit *viz.* high salinity and low-temperature, indicate a possible convergent signalling mechanism for all stresses relating to water deficit.

The identification of LIP15 and the ABRE- and Myb-like consensus sequences in the promoter region of genes reported in literature to be regulated in response to water stress, further corroborates the hypothesis that there is considerable overlap in the mechanisms governing the regulation of genes induced in response to water deficit.

The ACGT target sequence (core sequence of G-box elements) plays a central role in the expression of abiotic stress responsive genes, as many of the TESS-predicted transcription factors bind to *cis*-elements containing this core sequence.

The presence of GAmyb binding sites in many of the dehydration-responsive gene promoter sequences held in athlee_db, suggests the involvement of the ABA-dependent pathway which may activate *de novo* protein synthesis which, in turn, regulates GAmyb mediated gene expression, thereby repressing GA-regulated growth and development processes.

The predicted binding of the auxin-induced transcription factor, BBF, in the promoter region of genes regulated in response to water stress is not surprising as auxin is implicated in the mediation of target protein ubiquitination. This suggests that BBF may function to up-regulate abiotic stress responsive genes encoding proteins involved in apoptosis, thereby ensuring the integrity of the plants cells.

Plants have evolved mechanisms to monitor the quality, quantity, direction and duration of light conditions in order to ensure optimal growth via photosynthesis (Kirscher *et al.*, 1999). It follows that light-induced transcription factors likely effect changes in gene expression relating to plant growth and development. Therefore, under abiotic stress conditions, the expression of these genes would need to be down-regulated in order to repress plant growth and development. The majority of the light-induced transcription factors predicted by TESS to bind in the promoter region of abiotic stress responsive genes stored in athlee_db were also identified in genes reported to be regulated by ABA (Table 8).

ABA is known to cause partial stomatal closure thereby inhibiting photosynthesis and consequently growth and development. ABA may, therefore, indirectly or directly regulate the phosphorylation of light-induced transcription factors, resulting in the repression of proteins involved in plant growth and development.

The acquisition of desiccation tolerance during seed maturation, and the inhibition of seed germination by ABA and abiotic plant stresses relating to water deficit, points to a definite overlap in the mechanisms regulating gene expression during development and in vegetative tissue exposed to abiotic stress. It is, therefore, not surprising that development-induced transcription factors were predicted by TESS to bind in the promoter region of genes reported to be regulated in response to abiotic stress, and that different transcription factors mediate expression of similar genes during seed development and in response to abiotic stress.

Future investigations could include comparative promoter analysis using TRES (Transcription Regulatory Element Search – URL 15). Instead of searching a single sequence, TRES makes use of known information on transcription factor binding sites/*cis*-elements from PLACE (Plant *cis*-acting regulatory DNA elements –

Higo *et al.*, 1999; URL 19), TRANSFAC (URL 21) and ooTFD (URL 4) databases to perform a simultaneous search of several related sequences, thus facilitating the identification of common regulatory motifs (Katti *et al.*, 2000). Furthermore it is able to predict possible new motifs and gives information about the conservation of identified motifs relative to one another, the TSS and the TATA-box (Katti *et al.*, 2000). In addition, a software package developed by Terai *et al.* (2004) could be used to provide information regarding the order, presence and spatial arrangement of conserved motifs within the promoter region. This information would provide valuable insight into the precise mechanism of transcriptional regulation of these genes, as the order of and distance separating elements within the promoter is suggested to play an important role in determining gene expression (Terai *et al.*, 2004). Finally possible novel genes involved in drought, low temperature and high salinity plant stress response could be identified by developing a consensus nucleotide sequence and performing a BLAST-like search against complete and near-complete plant genomes (Appendix E).

Further research objectives and database improvements include:

- Automation of the initial search procedure and evaluation of BLAST results for mRNA records
- Increasing the data set to include all known genes encoding for proteins that have been reported to be up- or down-regulated in response to drought, low temperature, and salinity plant stresses.
- Development of a java applet to graphically display results in the form of a sequence diagram.

Although far from conclusive, this preliminary study demonstrates the value of this approach in collating the extensive data available on plant abiotic stress responses in order to gain a holistic perspective of this research area, and a valuable basis for future comparative promoter studies which will attempt to deduce possible common transcriptional initiation of stress response genes.

APPENDICES

Appendix A1 – populate.pl

```
#!/usr/bin/perl

# Athlee Maclear
# MSc Bioinformatics & Computational Molecular Biology 2004
# Rhodes University, Grahamstown, South Africa

# populate.pl
#           - reads a list of accession numbers stored in a text file
#           - creates a unique id to be used as the PRIMARY KEY
#           - inserts each accession and associated id as a new record into athlee_db

#Additional Perl Modules required
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#open file-handle to read in accessions
open(accessions, "SourceAccessions");

@accessions = <accessions>;

for ($num = 0; $num < scalar@accessions; $num++) {

   chomp @accessions[$num];

   $acc = @accessions[$num];
   $id = substr ($acc, 0, 2) . "_" . $num;    #create id

   #read into athlee_db
   $res = $conn->exec("INSERT INTO Source (id, accession) VALUES ('$id', '$acc');");
}

#close file-handle
close(accessions);
```

Appendix A2 – populateSource.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#populateSource.pl
#     - reads accession and id from athlee_db, Source Table
#     - retrieves corresponding sequence, and selected seqFeatures and annotations from GenBank
#     - populates the relative fields in the Source table

#Additional Perl Modules required
use Bio::Perl;
use Bio::SearchIO;
use Bio::Annotation::Collection;
use Bio::SeqIO;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#read Source.id from db and push onto an array
$res = $conn->exec("SELECT id FROM Source;");

while (@id_nos=$res->fetchrow){
   foreach $id(@id_nos){
      push(@ids,$id);
   }
}

#read Source.accession from db and push onto an array
$res=$conn->exec("SELECT accession FROM Source;");

while (@row=$res->fetchrow){
   foreach $acc(@row){
      push(@accession_nrs,$acc);
   }
}

#define source and seq features to be retrieved
@sourceFeatures = ('organism', 'mol_type', 'product');
@seqfeatures  =  ('CDS',  'gene',  'promoter',  'CAAT_signal',  'TATA_signal',  'polyA_signal',
'misc_feature');

#access GenBank
$gb = new Bio::DB::GenBank();
$gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );

#stream of seqIO objects for list of accessions
$seqio = $gb-> get_Stream_by_acc([@accession_nrs]);
```

```
$x=0; #declare increment variable

#while-loop to go through stream of seqIO objects
while($clone =  $seqio->next_seq) {
   $idnum = @ids[$x];

   #parse seqIO object and id to subroutines
   FetchRefAnnotations($clone, $idnum);
   FetchDescription($clone, $idnum);
   FetchSequence($clone, $idnum);

   #use foreach loop to go through SeqFeatures from each seqIO object
   foreach $feat_object ($clone->get_SeqFeatures) {
      FetchGeneName($feat_object, $idnum);

      #fetch selected source and seq features
      foreach $k(@seqfeatures){
         FetchSeqFeature($k, $feat_object, $idnum);
      }

      foreach $l(@sourceFeatures){
         FetchSourceFeatures($l, $feat_object, $idnum);
      }
   }

   #increment x
   $x++;
}

##############################################
sub FetchSeqFeature {

my ($key, $featobj, $identity) = @_;

   #retrieve start, end, and tag values for defined seqFeature
   if ($featobj->primary_tag eq "$key"){

      my $start = $featobj->start;
      my $end = $featobj->end;
      my $field = $featobj->primary_tag;
      @tags = $featobj->get_all_tags();

      if ($featobj->has_tag('note')){

         @notes = $featobj->get_tag_values('note');
         $note = @notes[0];
      }

      #write to db
      print $field, " ", $start, " ", $end, " ", $note, "\n";
```

```
        $res=$conn->exec("UPDATE Source SET $field = '$start - $end, $note' WHERE id =
        '$identity';");
    }
}

sub FetchRefAnnotations {

my ($seqio1, $ident1) = @_;

    #retrieve annotation from seqIO object
    my $anno_collection = $seqio1->annotation();

    #get all reference annotations
    my @annotations = $anno_collection->get_all_Annotations('reference');

    #retrieve original author and location and write to db
    my $authors = @annotations[0]->authors();
    $res=$conn->exec("UPDATE Source SET author = '$authors' WHERE id = '$ident1';");

    my $ref = @annotations[0]->location();
    $res=$conn->exec("UPDATE Source SET reference = '$ref' WHERE id = '$ident1';");
}

sub FetchSourceFeatures {

my ($sourceFeat, $seqio2, $ident2) = @_;

    #retrieve selected source features
    if ($seqio2->has_tag("$sourceFeat")){

        my @feat = $seqio2->get_tag_values("$sourceFeat");
        my $feature = @feat[0];

        #write to db
        $res=$conn->exec("UPDATE Source SET $sourceFeat = '$feature' WHERE id = '$ident2';");
    }
}

sub FetchGeneName {

my ($seqio3, $ident3) = @_;

    #retrieve gene name
    if ($seqio3->has_tag("gene")){
        my @geneName = $seqio3->get_tag_values("gene");
        my $gene = @geneName[0];

        #write to db
        $res=$conn->exec("UPDATE Source SET gene_name = '$gene' WHERE id = '$ident3';");
    }
}
```

```
sub FetchDescription {

my ($seqio4, $ident4) = @_;

   #retrieve description and write to db
   my $description = $seqio4->desc();

   $res=$conn->exec("UPDATE Source SET description = '$description' WHERE id = '$ident4';");
}

sub FetchSequence {

my ($seqio5, $ident5) = @_;

   #retrieve sequence and write to db
   my $retrieved_seq = $seqio5->seq();

   $res=$conn->exec("UPDATE Source SET seq = '$retrieved_seq' WHERE id = '$ident5';");
}
```

Appendix B1 – remoteBlast.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#remoteBlast.pl
# - selects accession, organism, seq from athlee_db, Source Table where mRNA data available
# - write seqs in FASTA format to an input file
# - remote BLAST - parameters => blastn, nr, 1e-10

#Additional Perl Modules required
use Bio::Tools::Run::RemoteBlast;
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#query db for only mRNA records
Pg::doQuery($conn,"SELECT accession, organism, seq FROM Source WHERE mol_type ~*
'mRNA';",\@ary);

#open file handle to write info to text file
open (FILEIN,">mRNAinputseq");

#for-loop to go through array of info read from the db
for $i(0 .. $#ary){
   #split organism name into array i.e. "Arabidopsis thaliana" -> ("Arabidopsis","thaliana")
   @orgName = split(/ /,$ary[$i][1]);

   #write info for all records in FASTA format into an input file
   print FILEIN ">$ary[$i][0]_$orgName[0]_$orgName[1]\n";
   print FILEIN "$ary[$i][2]*\n\n";
}

#close file handle
close FILEIN;

print "Finished generating BLAST input file...\n"; #track progress of program

#BLAST parameters
my $prog = 'blastn'; #nucleotide-nucleotide blast
my $db = 'nr'; #non-redundant db
my $e_val = '1e-10'; #threshold e-value

my @params = ('-prog'       =>$prog,
              '-data'       =>$db,
              '-expect'     =>$e_val,
              '-readmethod'=>'Search10');
```

```
#Remote BLAST
my $factory = Bio::Tools::Run::RemoteBlast->new(@params);
$factory->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );

#$v is just to turn messages on and off
my $v = 1;

#Read sequences in FASTA format from input file
my $str = Bio::SeqIO->new(-file=>'mRNAinputseq');

print "File read...\n"; #track progress

#submit_blast($input)
my $r = $factory->submit_blast('mRNAinputseq');

print STDERR "waiting..." if($v>0);

#while there are still more BLAST hits
while(my @rids = $factory->each_rid){
   foreach my $rid(@rids){
      my $rc = $factory->retrieve_blast($rid);
      if(!ref($rc)){
         if($rc<0){
            $factory->remove_rid($rid);
         }
         print STDERR "." if($v>0);
         sleep 5;
      }else{
         my $result = $rc->next_result();
         #save the output
         my $filename = $result->query_name()."\.out";
         $factory->save_output($filename);
         $factory->remove_rid($rid);
         print "\nQuery Name:", $result->query_name(),"\n";

         while(my $hit = $result->next_hit){
            next unless ($v>0);
            print "\tHit name: ", $hit->name,"\n";

            while(my $hsp = $hit->next_hsp){
               print "\t\tScore is ",$hsp->score,"\n";
            }
         }
      }
   }
}
```

Appendix B2 – parse_blast.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#parse_blast.pl
#           - parses Blast Report files filtering out unwanted hits
#           - writes selected info to an output file and to athlee_db, Blast_Results table
#           - retrieves and writes hit sequences to athlee_db, Blast_Results table

#Additional Perl Modules required
use Bio::Perl;
use Bio::SearchIO;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#read all .out filenames in BlastReportFiles directory into and array
my $dir = "/home/g9610081/BlastReportFiles";
my @blastreport_files = <$dir/*.out>;

#open file-handles to write info to text file
open(files,">filenames");
open (genomic_hits,">Genomic_Hits");

#foreach Blast report retrieve just the filename from the file path
foreach $i(@blastreport_files){
   $file = substr($i,15);
   print files "$file\n"; #write to a text file
}

#close file handle
close(files);

#open file handle to read in info
open(filenames,"filenames");
@files = <filenames>;

#declare increment variable
$x=0;

#retrieve accession and organism from each filename
foreach $filename (@files){
   chomp $filename;
   @name = split(/_/,$filename);
   $accession = @name[0];
   @name[2] =~ /(\w+)(.out$)/i;
   $species = $1;
```

```
$genus = @name[1];

#parse each Blast report file
my $in = new Bio::SearchIO(-format => 'blast',
                           -file   => $filename);

#filter results
while(my $result = $in->next_result) {
   while(my $hit = $result->next_hit) {
      while(my $hsp = $hit->next_hsp) {
       #filter level 1 => percent identity
       if($hsp->percent_identity >= 75) {
           #level 2 => same species as query
          if($hit->description =~ /[\s\w]+($species)[\s\w]+/i) {
            #level 3 => filter out mRNA/cDNA hits
            if($hit->description =~/[\s\w]+(mRNA|cDNA)+[\s\w]*/i) {
            } else {
               $x++; #increment x

               #save output to text file
               print genomic_hits "$x Hit:              ", $hit->name,
                                  "\nHit Accession_no:", $hit->accession,
                                  "\nHit Description: ", $hit->description,
                                  "\nLength:          ", $hsp->length('total'),
                                  "\nPercent_id:      ", $hsp->percent_identity,
                                  "\nE-value:         ", $hsp->evalue,
                                  "\nStart (Query):   ", $hsp->start('query'),
                                  "\nEnd (Query):     ", $hsp->end('query'),
                                  "\nStart (Hit):     ", $hsp->start('hit'),
                                  "\nEnd (Hit):       ", $hsp->end('hit');

               print genomic_hits "\n\n\n";

               #declare and instantiate db variables
               $hit_name = $hit->name;
               $hit_accession = $hit->accession;
               $hit_description = $hit->description;
               $hit_description =~ s/\'/_/g;
               $length = $hsp->length('total');
               $percent_id = $hsp->percent_identity;
               $e_value = $hsp->evalue;
               $start_query = $hsp->start('query');
               $end_query = $hsp->end('query');
               $start_hit = $hsp->start('hit');
               $end_hit = $hsp->end('hit');

               #fetch hit sequences
               if ($hit_name =~/^gb/){
                  $hit_seq=FetchGBSequence($hit_accession);
               }else{
                  $hit_seq=FetchEMBLSequence($hit_accession);
               }
```

```
                    #write output to athlee_db, Blast_Results table
                    $res=$conn->exec("INSERT INTO Blast_Results (accession, hit_name,
                    hit_accession, hit_description, length, percent_id, e_value, start_query,
                    end_query, start_hit, end_hit, hit_seq) VALUES ('$accession', '$hit_name',
                    '$hit_accession', '$hit_description', '$length', '$percent_id', '$e_value',
                    '$start_query', '$end_query', '$start_hit', '$end_hit', '$hit_seq');");
                }
            }
        }
    }
  }
}

print "$x iterations\n";

#close file handles
close(filenames);
close(genomic_hits);

#####################################################################
sub FetchGBSequence{

my $search_term = $_[0];
    chomp $search_term;
    $gb = new Bio::DB::GenBank(); #access GenBank
    $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $gb->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->seq();
    $retrieved_seq;
}

sub FetchEMBLSequence{

my $search_term = $_[0];
    chomp $search_term;
    $embl = new Bio::DB::EMBL(); #access EMBL
    $embl->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $embl->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->seq();
    $retrieved_seq;
}
```

Appendix B3 – Blast_Results Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	text	Accession number of the query mRNA record
Hit Name	text	e.g. gb|AC003970.1|
Hit Accession	text	Hit accession number
Hit Description	text	Description of hit nucleotide
Length	int	Length of hit with the query
Percent Id	int	Percent identity between hit and query
E-Value	int	E-value score of match
Start query	int	Start position of match on the query sequence
End query	int	End position of match on the query sequence
Start hit	int	Start position of match on the hit sequence
End hit	int	End position of match on the hit sequence
Hit seq	text	Hit Sequence
Link	text	FOREIGN KEY referencing Source.id

Appendix B4 – Search_SeqInfo Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	varchar(80)	Accession number of the query mRNA record
Hit Accession	varchar(80)	e.g. gb|AC003970.1|
TESSprom_start	int	2000 bps upstream of gene start
TESSprom_end	int	1bp upstream of gene start
Intron1_start	Int	Start position of Intron 1
Intron1_end	int	End position of Intron 1
Intron2_start	int	Start position of Intron 2
Intron2_end	int	End position of Intron 2
		
Intron8_start	int	Start position of Intron 8
Intro8_end	int	End position of Intron 8
TRESprom_start	int	1000 bps upstream of gene start
TRESprom_end	int	1bp upstream of gene start
Link	text	FOREIGN KEY referencing Source.id

Appendix B5 – searchSeqFetch.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#searchSeqFetch.pl
#     - reads promoter start and end info from athlee_db, Search_SeqInfo table
#     - uses info to retrieve promoter subsequences
#     - writes promoter subseqs to athlee_db, Search_Seqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Search_SeqInfo table
$res=$conn->exec("SELECT accession FROM Search_SeqInfo;");

while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT hit_accession FROM Search_SeqInfo;");

while (@rows=$res->fetchrow){
   foreach $line(@rows){
      push(@hitaccession_nrs,$line);
   }
}

$res=$conn->exec("SELECT TESSprom_start FROM Search_SeqInfo;");

while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}

$res=$conn->exec("SELECT TESSprom_end FROM Search_SeqInfo;");

while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}
```

```
$res=$conn->exec("SELECT TRESpromarea_start FROM Search_SeqInfo;");

while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
      push(@startnrs2,$start2);
   }
}

$res=$conn->exec("SELECT TRESpromarea_end FROM Search_SeqInfo;");

while (@rows22=$res->fetchrow){
   foreach $end2(@rows22){
      push(@endnrs2,$end2);
   }
}

#declare increment variable
$index = 0;

#retrieve promoter subsequences and write to athlee_db, Search_Seqs table
foreach $entry(@hitaccession_nrs){
   print "Fetching $entry...\n"; #track progress

   $accession=@accession_nrs[$index];
   $promoter2000=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
   $promoter1000=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);

   print "Writing promoter sequence to database...\n"; #track progress

   $res=$conn->exec("INSERT INTO Search_Seqs (accession, hit_accession, promoter2000,
   promoter1000) VALUES ('$accession', '$entry','$promoter2000', '$promoter1000');");
   $res=$conn->exec("UPDATE Search_Seqs SET seq = Blast_Results.hit_seq where hit_accession =
   Blast_Results.hit_accession;");

   $index++; #increment
}

##################################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;
   $gb = new Bio::DB::GenBank(); #access GenBank
   $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
   $seqobj = $gb->get_Seq_by_acc($search_term);
   $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve sub-sequence
   $retrieved_seq;
}
```

Appendix B6 – FetchIntrons.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#FetchIntrons.pl
#       - reads intron start and end info from athlee_db, Search_SeqInfo table
#       - uses info to retrieve intron subsequences
#       - writes intron subseqs to athlee_db, Search_Seqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Search_SeqInfo table
$res=$conn->exec("SELECT accession FROM Search_SeqInfo;");

while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT link FROM Search_SeqInfo;");

while (@row1=$res->fetchrow){
   foreach $ident(@row1){
      push(@ids,$ident);
   }
}

$res=$conn->exec("SELECT hit_accession FROM Search_SeqInfo;");

while (@rows=$res->fetchrow){
   foreach $line(@rows){
      push(@hitaccession_nrs,$line);
   }
}

$res=$conn->exec("SELECT intron1_start FROM Search_SeqInfo;");

while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}
```

```
$res=$conn->exec("SELECT intron1_end FROM Search_SeqInfo;");

while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}

$res=$conn->exec("SELECT intron2_start FROM Search_SeqInfo;");

while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
      push(@startnrs2,$start2);
   }
}

$res=$conn->exec("SELECT intron2_end FROM Search_SeqInfo;");

while (@rows22=$res->fetchrow){
   foreach $end2(@rows22){
      push(@endnrs2,$end2);
   }
}

$res=$conn->exec("SELECT intron3_start FROM Search_SeqInfo;");

while (@rows3=$res->fetchrow){
   foreach $start3(@rows3){
      push(@startnrs3,$start3);
   }
}

$res=$conn->exec("SELECT intron3_end FROM Search_SeqInfo;");

while (@rows33=$res->fetchrow){
   foreach $end3(@rows33){
      push(@endnrs3,$end3);
   }
}

$res=$conn->exec("SELECT intron4_start FROM Search_SeqInfo;");

while (@rows4=$res->fetchrow){
   foreach $start4(@rows4){
      push(@startnrs4,$start4);
   }
}

$res=$conn->exec("SELECT intron4_end FROM Search_SeqInfo;");
```

```
while (@rows44=$res->fetchrow){
   foreach $end4(@rows44){
      push(@endnrs4,$end4);
   }
}

$res=$conn->exec("SELECT intron5_start FROM Search_SeqInfo;");

while (@rows5=$res->fetchrow){
   foreach $start5(@rows5){
      push(@startnrs5,$start5);
   }
}

$res=$conn->exec("SELECT intron5_end FROM Search_SeqInfo;");

while (@rows55=$res->fetchrow){
   foreach $end5(@rows55){
      push(@endnrs5,$end5);
   }
}

$res=$conn->exec("SELECT intron6_start FROM Search_SeqInfo;");

while (@rows6=$res->fetchrow){
   foreach $start6(@rows6){
      push(@startnrs6,$start6);
   }
}

$res=$conn->exec("SELECT intron6_end FROM Search_SeqInfo;");

while (@rows66=$res->fetchrow){
   foreach $end6(@rows66){
      push(@endnrs6,$end6);
   }
}

$res=$conn->exec("SELECT intron7_start FROM Search_SeqInfo;");

while (@rows7=$res->fetchrow){
   foreach $start7(@rows7){
      push(@startnrs7,$start7);
   }
}

$res=$conn->exec("SELECT intron7_end FROM Search_SeqInfo;");

while (@rows77=$res->fetchrow){
   foreach $end7(@rows77){
      push(@endnrs7,$end7);
   }
```

```
}

$res=$conn->exec("SELECT intron8_start FROM Search_SeqInfo;");

while (@rows8=$res->fetchrow){
   foreach $start8(@rows8){
      push(@startnrs8,$start8);
   }
}

$res=$conn->exec("SELECT intron8_end FROM Search_SeqInfo;");

while (@rows88=$res->fetchrow){
   foreach $end8(@rows88){
      push(@endnrs8,$end8);
   }
}

#declare increment variable
$index = 0;

#retrieve intron subsequences and write to athlee_db, Search_Seqs table
foreach $entry(@hitaccession_nrs){
   #id of record
   $id = @ids[$index];

   print "Fetching introns from hit: $entry...\n"; #track progress

   $intron1=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
   $intron2=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);
   $intron3=FetchSequence($entry,@startnrs3[$index],@endnrs3[$index]);
   $intron4=FetchSequence($entry,@startnrs4[$index],@endnrs4[$index]);
   $intron5=FetchSequence($entry,@startnrs5[$index],@endnrs5[$index]);
   $intron6=FetchSequence($entry,@startnrs6[$index],@endnrs6[$index]);
   $intron7=FetchSequence($entry,@startnrs7[$index],@endnrs7[$index]);
   $intron8=FetchSequence($entry,@startnrs8[$index],@endnrs8[$index]);

   #track progress
   print "Writing entry to database...\n";

   #write to athlee_db, Search_Seqs table
   $res=$conn->exec("UPDATE Search_Seqs SET intron1 = '$intron1' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron2 = '$intron2' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron3 = '$intron3' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron4 = '$intron4' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron5 = '$intron5' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron6 = '$intron6' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron7 = '$intron7' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron8 = '$intron8' WHERE link = '$id';");

   #increment
   $index++;
```

```
}

#if no subseq to retrieve, enters a '1' - clear from db
$res=$conn->exec("UPDATE Search_Seqs SET intron1 = '' WHERE intron1 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron2 = '' WHERE intron2 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron3 = '' WHERE intron3 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron4 = '' WHERE intron4 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron5 = '' WHERE intron5 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron6 = '' WHERE intron6 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron7 = '' WHERE intron7 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron8 = '' WHERE intron8 = '1';");

##################################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;
   $gb = new Bio::DB::GenBank(); #access GenBank
   $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
   $seqobj = $gb->get_Seq_by_acc($search_term);
   $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve sub-seq
   $retrieved_seq;
}
```

Appendix B7 – Search_Seqs Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	varchar(80)	Accession number of the query mRNA record
Hit Accession	varchar(80)	e.g. gb|AC003970.1|
Promoter2000	Text	2000 bps putative promoter subsequence
Intron1	Text	Intron subsequences
↕	↕	↕
Intron8	Text	
Promoter1000	Text	1000 bps promoter subsequence (TRES only takes 1000bps)
Link	Text	FOREIGN KEY referencing Source.id

Appendix B8 – GenomicSearchInfo.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#GenomicSearchInfo.pl
#    - retrieves promoter, intron and CDS seqFeatures for Genomic records
     - writes start and end info to athlee_db, Genomic_SearchInfo table

#Additional Perl Modules required
use Bio::Perl;
use Bio::SeqIO;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#select link (= Source.id -> foreign key) and push onto an array
$res=$conn->exec("SELECT link FROM Genomic_SearchInfo;");

while (@id_nos=$res->fetchrow){
   foreach $id(@id_nos){
      push(@ids,$id);
   }
}

#select accession and push onto an array
$res=$conn->exec("SELECT accession FROM Genomic_SearchInfo;");

while (@row=$res->fetchrow){
   foreach $acc(@row){
      push(@accession_nrs,$acc);
   }
}

#access GenBank and retrieve seqIO stream
$gb = new Bio::DB::GenBank();
$gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
$seqio = $gb-> get_Stream_by_acc([@accession_nrs]);

#declare increment variable
$x=0;

#retrieve promoter, intron and CDS seqFeatures from each seqIO object
while($clone =  $seqio->next_seq) {
   foreach $feat_object ($clone->get_SeqFeatures) {
      FetchPromoter($feat_object, $ids[$x]);
      FetchIntron($feat_object, $ids[$x]);
      FetchCDSsubseqs($feat_object, $ids[$x]);
```

```
    }
    $x++; #increment x
}

#####################################################
sub FetchPromoter {

my ($featobj, $identity) = @_;
#retrieve promoter seqFeature and write start and end values to athlee_db, Genomic_SearchInfo
if ($featobj->primary_tag eq "promoter"){
    $start = $featobj->start;
    $res=$conn->exec("UPDATE Genomic_SearchInfo SET prom_start = '$start' WHERE link =
    '$identity';");
    $end = $featobj->end;
    $res=$conn->exec("UPDATE Genomic_SearchInfo SET prom_end = '$end' WHERE link =
    '$identity';");
    }
}

sub FetchIntron {

my ($featobj1, $identity1) = @_;
#retrieve intron seqFeatures
if ($featobj1->primary_tag eq "intron"){
    $start1 = $featobj1->start;
    $end1 = $featobj1->end;
    @tags = $featobj1->get_all_tags();

    #get intron number
    if ($featobj1->has_tag('number')){
        @num = $featobj1->get_tag_values('number');
        $n = @num[0];
    }else{
        $n = 1;
    }

    $field = "intron" . $n;
    $field1 = $field . "_start";
    $field2 = $field . "_end";

    #write start and end values to athlee_db, Genomic_SearchInfo table
    $res=$conn->exec("UPDATE Genomic_SearchInfo SET $field1 = '$start1' WHERE link =
    '$identity1';");

    $res=$conn->exec("UPDATE Genomic_SearchInfo SET $field2 = '$end1' WHERE link =
    '$identity1';");
    }
}

sub FetchCDSsubseqs {

my ($featobj2, $identity2) = @_;
```

```
my $y=1; #exon number

  #retrieve CDS splitLocation seqFeatures
    #i.e. CDS => >1 subseq (exons) - can therefore define introns
  if ($featobj2->location->isa('Bio::Location::SplitLocationI')
                                        && $feat_object->primary_tag eq 'CDS')  {
    #retrieve start and end of each exon
    foreach $location ($featobj2->location->sub_Location) {
      $End = $location->end;
      $Start = $location->start;
      $StartField = "Start$y";
      $EndField = "End$y";

      #write info to athlee_db, CDS_SeqInfo table
      $res=$conn->exec("UPDATE CDS_SeqInfo SET $StartField = '$Start' WHERE link =
      '$identity2';");

      $res=$conn->exec("UPDATE CDS_SeqInfo SET $EndField = '$End' WHERE link =
      '$identity2';");

      $y++; #increment y
    }
  }
}
```

Appendix B9 – CDS_SeqInfo Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	Text	Accession number of the query mRNA record
Link	Text	FOREIGN KEY referencing Source.id
Start1	Text	Start and end positions of CDS split-location seqFeatures – indicate where the exons start and end. PopulateGenomicSearchInfo.pl uses these values to define the intron regions
End1	Text	
↕	↕	↕
Start9	Text	
End9	Text	

Appendix B10 – populateGenomic_SearchInfo.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#populateGenomic_SearchInfo.pl
#    - defines intron and promoter start and end positions from info obtained from
#      splitLocation CDS seqFeatures
#    - writes info to athlee_db, Genomic_SearchInfo tables

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

for ($x = 1; $x < 14; $x++){
   $y = $x + 1;

   $IntronStart = "intron" . $x ."_start";
   $IntronEnd = "intron" . $x ."_end";
   $CDS_field1 = "CDS_SeqInfo.end$x";
   $CDS_field2 = "CDS_SeqInfo.start$y";

   print $IntronStart, " ", $CDS_field1, "\n";

   $res=$conn->exec("UPDATE Genomic_SearchInfo SET $IntronStart = $CDS_field1 + 1 WHERE link =
   CDS_SeqInfo.link;");

   $res=$conn->exec("UPDATE Genomic_SearchInfo SET $IntronEnd = $CDS_field2 - 1 WHERE link =
   CDS_SeqInfo.link;");
}

$res=$conn->exec("UPDATE Genomic_SearchInfo SET Prom_start = CDS_SeqInfo.start1 - 2000 WHERE
link = CDS_SeqInfo.link;");
$res=$conn->exec("UPDATE Genomic_SearchInfo SET Prom_end = CDS_SeqInfo.start1 - 1 WHERE link =
CDS_SeqInfo.link;");
```

Appendix B11 – Genomic_SearchInfo Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	Text	Accession number of the genomic DNA record
Prom_start	Int	Upstream of gene start
Prom_end	Int	1bp upstream of gene start
Intron1_start	Int	Start position of Intron 1
Intron1_end	Int	End position of Intron 1
Intron2_start	Int	Start position of Intron 2
Intron2_end	Int	End position of Intron 2
↕	↕	↕
Intron13_start	Int	Start position of Intron 13
Intro13_end	Int	End position of Intron 13
TESSprom_start	Int	2000 bps upstream of gene start
TESSprom_end	Int	1bp upstream of gene start
Link	Text	FOREIGN KEY referencing Source.id

Appendix B12 – searchSeqFetch2.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa
#searchSeqFetch2.pl
#     - reads promoter start and end info from athlee_db, Genomic_SearchInfo table
#     - uses info to retrieve promoter subsequences
#     - writes promoter subseqs to athlee_db, Genomic_SearchSeqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Genomic_SearchInfo table
$res=$conn->exec("SELECT accession FROM Genomic_SearchInfo;");
while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT link FROM Genomic_SearchInfo;");
while (@row1=$res->fetchrow){
   foreach $number1(@row1){
      push(@ids,$number1);
   }
}

$res=$conn->exec("SELECT prom_start FROM Genomic_SearchInfo;");
while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}

$res=$conn->exec("SELECT prom_end FROM Genomic_SearchInfo;");
while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}

$res=$conn->exec("SELECT TESSprom_start FROM Genomic_SearchInfo;");
while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
      push(@startnrs2,$start2);
```

```
    }
}

$res=$conn->exec("SELECT TESSprom_end FROM Genomic_SearchInfo;");
while (@rows22=$res->fetchrow){
    foreach $end2(@rows22){
        push(@endnrs2,$end2);
    }
}

#declare increment variable
$index = 0;

#retrieve promoter subsequences and write to athlee_db, Genomic_SearchSeqs table
foreach $entry(@accession_nrs){
    print "Fetching $entry...\n"; #track progress

    $accession = @accession_nrs[$index];
    $id = @ids[$index];
    $prom=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
    $prom2000=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);

    print "Writing promoter sequences to the database...\n"; #track progress

    $res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter = '$prom' WHERE link = '$id';");
    $res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter2000 = '$prom2000' WHERE link =
    '$id';");

    $index++; #increment
}

#if no subseq to retrieve, enters a '1' - clear from db
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter = '' WHERE promoter = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter2000 = '' WHERE promoter2000 = '1';");

##############################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;

    $gb = new Bio::DB::GenBank(); #access GenBank
    $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $gb->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve subseqs
    $retrieved_seq;
}
```

Appendix B13 – FetchIntrons2.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#FetchIntrons2.pl
#    - reads intron start and end info from athlee_db, Genomic_SearchInfo table
#    - uses info to retrieve intron subsequences
#    - writes intron subseqs to athlee_db, Search_Seqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Genomic_SearchInfo table
$res=$conn->exec("SELECT accession FROM Genomic_SearchInfo;");
while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT link FROM Genomic_SearchInfo;");
while (@rows=$res->fetchrow){
   foreach $line(@rows){
      push(@ids,$line);
   }
}

$res=$conn->exec("SELECT intron1_start FROM Genomic_SearchInfo;");
while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}

$res=$conn->exec("SELECT intron1_end FROM Genomic_SearchInfo;");
while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}

$res=$conn->exec("SELECT intron2_start FROM Genomic_SearchInfo;");
while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
```

```
      push(@startnrs2,$start2);
   }
}

$res=$conn->exec("SELECT intron2_end FROM Genomic_SearchInfo;");
while (@rows22=$res->fetchrow){
   foreach $end2(@rows22){
      push(@endnrs2,$end2);
   }
}

$res=$conn->exec("SELECT intron3_start FROM Genomic_SearchInfo;");
while (@rows3=$res->fetchrow){
   foreach $start3(@rows3){
      push(@startnrs3,$start3);
   }
}

$res=$conn->exec("SELECT intron3_end FROM Genomic_SearchInfo;");
while (@rows33=$res->fetchrow){
   foreach $end3(@rows33){
      push(@endnrs3,$end3);
   }
}

$res=$conn->exec("SELECT intron4_start FROM Genomic_SearchInfo;");
while (@rows4=$res->fetchrow){
   foreach $start4(@rows4){
      push(@startnrs4,$start4);
   }
}

$res=$conn->exec("SELECT intron4_end FROM Genomic_SearchInfo;");
while (@rows44=$res->fetchrow){
   foreach $end4(@rows44){
      push(@endnrs4,$end4);
   }
}

$res=$conn->exec("SELECT intron5_start FROM Genomic_SearchInfo;");
while (@rows5=$res->fetchrow){
   foreach $start5(@rows5){
      push(@startnrs5,$start5);
   }
}

$res=$conn->exec("SELECT intron5_end FROM Genomic_SearchInfo;");
while (@rows55=$res->fetchrow){
   foreach $end5(@rows55){
      push(@endnrs5,$end5);
   }
}
```

```
$res=$conn->exec("SELECT intron6_start FROM Genomic_SearchInfo;");
while (@rows6=$res->fetchrow){
   foreach $start6(@rows6){
      push(@startnrs6,$start6);
   }
}

$res=$conn->exec("SELECT intron6_end FROM Genomic_SearchInfo;");
while (@rows66=$res->fetchrow){
   foreach $end6(@rows66){
      push(@endnrs6,$end6);
   }
}

$res=$conn->exec("SELECT intron7_start FROM Genomic_SearchInfo;");
while (@rows7=$res->fetchrow){
   foreach $start7(@rows7){
      push(@startnrs7,$start7);
   }
}

$res=$conn->exec("SELECT intron7_end FROM Genomic_SearchInfo;");
while (@rows77=$res->fetchrow){
   foreach $end7(@rows77){
      push(@endnrs7,$end7);
   }
}

$res=$conn->exec("SELECT intron8_start FROM Genomic_SearchInfo;");
while (@rows8=$res->fetchrow){
   foreach $start8(@rows8){
      push(@startnrs8,$start8);
   }
}

$res=$conn->exec("SELECT intron8_end FROM Genomic_SearchInfo;");
while (@rows88=$res->fetchrow){
   foreach $end8(@rows88){
      push(@endnrs8,$end8);
   }
}

$res=$conn->exec("SELECT intron9_start FROM Genomic_SearchInfo;");
while (@rows9=$res->fetchrow){
   foreach $start9(@rows9){
      push(@startnrs9,$start9);
   }
}

$res=$conn->exec("SELECT intron9_end FROM Genomic_SearchInfo;");
while (@rows99=$res->fetchrow){
```

```
    foreach $end9(@rows99){
        push(@endnrs9,$end9);
    }
}

$res=$conn->exec("SELECT intron10_start FROM Genomic_SearchInfo;");
while (@rows10=$res->fetchrow){
    foreach $start10(@rows10){
        push(@startnrs10,$start10);
    }
}

$res=$conn->exec("SELECT intron10_end FROM Genomic_SearchInfo;");
while (@rows1010=$res->fetchrow){
    foreach $end10(@rows1010){
        push(@endnrs10,$end10);
    }
}

$res=$conn->exec("SELECT intron11_start FROM Genomic_SearchInfo;");
while (@rows11=$res->fetchrow){
    foreach $start11(@rows11){
        push(@startnrs11,$start11);
    }
}

$res=$conn->exec("SELECT intron11_end FROM Genomic_SearchInfo;");
while (@rows1111=$res->fetchrow){
    foreach $end11(@rows1111){
        push(@endnrs11,$end11);
    }
}

$res=$conn->exec("SELECT intron12_start FROM Genomic_SearchInfo;");
while (@rows12=$res->fetchrow){
    foreach $start12(@rows12){
        push(@startnrs12,$start12);
    }
}

$res=$conn->exec("SELECT intron12_end FROM Genomic_SearchInfo;");
while (@rows1212=$res->fetchrow){
    foreach $end12(@rows1212){
        push(@endnrs12,$end12);
    }
}

$res=$conn->exec("SELECT intron13_start FROM Genomic_SearchInfo;");
while (@rows13=$res->fetchrow){
    foreach $start13(@rows13){
        push(@startnrs13,$start13);
    }
```

```
}

$res=$conn->exec("SELECT intron13_end FROM Genomic_SearchInfo;");
while (@rows1313=$res->fetchrow){
   foreach $end13(@rows1313){
      push(@endnrs13,$end13);
   }
}

#declare increment variable
$index = 0;

#retrieve intron subsequences and write to athlee_db, Genomic_SearchInfo table
foreach $entry(@accession_nrs){
   #id of record
   $id = @ids[$index];

   print "Fetching introns from: $entry...\n"; #track progress

   $intron1=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
   $intron2=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);
   $intron3=FetchSequence($entry,@startnrs3[$index],@endnrs3[$index]);
   $intron4=FetchSequence($entry,@startnrs4[$index],@endnrs4[$index]);
   $intron5=FetchSequence($entry,@startnrs5[$index],@endnrs5[$index]);
   $intron6=FetchSequence($entry,@startnrs6[$index],@endnrs6[$index]);
   $intron7=FetchSequence($entry,@startnrs7[$index],@endnrs7[$index]);
   $intron8=FetchSequence($entry,@startnrs8[$index],@endnrs8[$index]);
   $intron9=FetchSequence($entry,@startnrs9[$index],@endnrs9[$index]);
   $intron10=FetchSequence($entry,@startnrs10[$index],@endnrs10[$index]);
   $intron11=FetchSequence($entry,@startnrs11[$index],@endnrs11[$index]);
   $intron12=FetchSequence($entry,@startnrs12[$index],@endnrs12[$index]);
   $intron13=FetchSequence($entry,@startnrs13[$index],@endnrs13[$index]);

   print "Writing entry to database...\n"; #track progress

   #write to athlee_db, Genomic_Search_Seqs table
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron1 = '$intron1' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron2 = '$intron2' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron3 = '$intron3' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron4 = '$intron4' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron5 = '$intron5' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron6 = '$intron6' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron7 = '$intron7' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron8 = '$intron8' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron9 = '$intron9' WHERE link = '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron10  =  '$intron10'  WHERE  link  =
   '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron11  =  '$intron11'  WHERE  link  =
   '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron12  =  '$intron12'  WHERE  link  =
   '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron13  =  '$intron13'  WHERE  link  =
```

```
    '$id';");

    $index++; #increment
}

#if no subseq to retrieve, enters a '1' - clear from db
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron1 = '' WHERE intron1 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron2 = '' WHERE intron2 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron3 = '' WHERE intron3 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron4 = '' WHERE intron4 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron5 = '' WHERE intron5 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron6 = '' WHERE intron6 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron7 = '' WHERE intron7 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron8 = '' WHERE intron8 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron9 = '' WHERE intron9 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron10 = '' WHERE intron10 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron11 = '' WHERE intron11 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron12 = '' WHERE intron12 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron13 = '' WHERE intron13 = '1';");

##################################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;

    $gb = new Bio::DB::GenBank(); #access GenBank
    $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $gb->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve sub-seq
    $retrieved_seq;
}
```

Appendix B14 – Genomic_SearchSeqs Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	Text	Accession number of the query mRNA record
Link	Text	FOREIGN KEY referencing Source.id
Promoter	Text	Promoter subsequence
Intron1	Text	Intron subsequences
↕	↕	↕
Intron13	Text	
Promoter2000	Text	2000bp promoter subsequence

Appendix C1 – PromoterFastaFiles.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#PromoterFastaFiles.pl
#    - reads promoter sequences from athlee_db, Search_Seqs and Genomic_SearchSeqs tables
#    - writes in FASTA format to a series of input files
#    - used as input for TESS - promoter searching program

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#mRNA Promoter Sequences
Pg::doQuery($conn,"SELECT accession, hit_accession, promoter2000, link FROM
Search_Seqs;",\@ary);
open (FILEIN,">TESSinputFiles\/mRNA_Promoter2000Fastas");

for $i(0 .. $#ary){
   print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_PromoterSequence_2000bps\n";
   print FILEIN "$ary[$i][2]*\n\n";
}
close FILEIN;

Pg::doQuery($conn,"SELECT accession, hit_accession, promoter1000, link FROM
Search_Seqs;",\@ary1);
open (FILEIN1,">TRESinputFiles\/mRNA_Promoter1000Fastas");

for $j(0 .. $#ary1){
   print FILEIN1 ">ID|$ary1[$j][3]_Source|$ary1[$j][0]_Hit|$ary1[$j][1]_PromoterSequence_1000bps\n";
   print FILEIN1 "$ary1[$j][2]*\n\n";
}
close FILEIN1;

#Genomic DNA Promoter Sequences
Pg::doQuery($conn,"SELECT accession, link, promoter2000 FROM Genomic_SearchSeqs;",\@ary2);
open (FILEIN2,">TESSinputFiles\/GenomicDNA_Promoter2000Fastas");

for $k(0 .. $#ary2){
   print FILEIN2 ">ID|$ary2[$k][1]_Accession|$ary2[$k][0]_PromoterSequence_2000bps\n";
   print FILEIN2 "$ary2[$k][2]*\n\n";
}
close FILEIN2;
```

Appendix C2 – IntronFastaFiles.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#IntronFastaFiles.pl
#    - reads intron sequences from athlee_db, Search_Seqs and Genomic_SearchSeqs tables
#    - writes in FASTA format to a series of input files
#    - used as input for TESS - promoter searching program

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#mRNA Intron Sequences
Pg::doQuery($conn,"SELECT accession, hit_accession, intron1, link FROM Search_Seqs;",\@ary);
open (FILEIN,">TESSinputFiles\/mRNA_Intron1Fastas");

for $i(0 .. $#ary){
   print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_Intron1\n";
   print FILEIN "$ary[$i][2]\n\n";
}
close FILEIN;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron2, link FROM Search_Seqs;",\@ary1);
open (FILEIN1,">TESSinputFiles\/mRNA_Intron2Fastas");

for $j(0 .. $#ary1){
   print FILEIN1 ">ID|$ary1[$j][3]_Source|$ary1[$j][0]_Hit|$ary1[$j][1]_Intron2\n";
   print FILEIN1 "$ary1[$j][2]\n\n";
}
close FILEIN1;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron3, link FROM Search_Seqs;",\@ary2);
open (FILEIN2,">TESSinputFiles\/mRNA_Intron3Fastas");

for $k(0 .. $#ary2){
   print FILEIN2 ">ID|$ary2[$k][3]_Source|$ary2[$k][0]_Hit|$ary2[$k][1]_Intron3\n";
   print FILEIN2 "$ary2[$k][2]\n\n";
}
close FILEIN2;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron4, link FROM Search_Seqs;",\@ary3);
open (FILEIN3,">TESSinputFiles\/mRNA_Intron4Fastas");

for $l(0 .. $#ary3){
```

```
   print FILEIN3 ">ID|$ary3[$l][3]_Source|$ary3[$l][0]_Hit|$ary3[$l][1]_Intron4\n";
   print FILEIN3 "$ary3[$l][2]\n\n";
}
close FILEIN3;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron5, link FROM Search_Seqs;",\@ary4);
open (FILEIN4,">TESSinputFiles\/mRNA_Intron5Fastas");

for $m(0 .. $#ary4){
   print FILEIN4 ">ID|$ary4[$m][3]_Source|$ary4[$m][0]_Hit|$ary4[$m][1]_Intron5\n";
   print FILEIN4 "$ary4[$m][2]\n\n";
}
close FILEIN4;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron6, link FROM Search_Seqs;",\@ary5);
open (FILEIN5,">TESSinputFiles\/mRNA_Intron6Fastas");

for $n(0 .. $#ary5){
   print FILEIN5 ">ID|$ary5[$n][3]_Source|$ary5[$n][0]_Hit|$ary5[$n][1]_Intron6\n";
   print FILEIN5 "$ary5[$n][2]\n\n";
}
close FILEIN5;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron7, link FROM Search_Seqs;",\@ary6);
open (FILEIN6,">TESSinputFiles\/mRNA_Intron7Fastas");

for $o(0 .. $#ary6){
   print FILEIN6 ">ID|$ary6[$o][3]_Source|$ary6[$o][0]_Hit|$ary6[$o][1]_Intron7\n";
   print FILEIN6 "$ary6[$o][2]\n\n";
}
close FILEIN6;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron8, link FROM Search_Seqs;",\@ary8);
open (FILEIN8,">TESSinputFiles\/mRNA_Intron8Fastas");

for $pp(0 .. $#ary8){
   print FILEIN8 ">ID|$ary8[$pp][3]_Source|$ary8[$pp][0]_Hit|$ary8[$pp][1]_Intron8\n";
   print FILEIN8 "$ary8[$pp][2]\n\n";
}
close FILEIN8;

#Genomic DNA Intron Sequences
Pg::doQuery($conn,"SELECT accession, link, intron1 FROM Genomic_SearchSeqs;",\@ary11);
open (FILEIN11,">TESSinputFiles\/GenomicDNA_Intron1Fastas");

for $q(0 .. $#ary11){
   print FILEIN11 ">ID|$ary11[$q][1]_Accession|$ary11[$q][0]_Intron1\n";
   print FILEIN11 "$ary11[$q][2]\n\n";
}
close FILEIN11;

Pg::doQuery($conn,"SELECT accession, link, intron2 FROM Genomic_SearchSeqs;",\@ary22);
```

APPENDICES

Appendix A1 – populate.pl

```
#!/usr/bin/perl

# Athlee Maclear
# MSc Bioinformatics & Computational Molecular Biology 2004
# Rhodes University, Grahamstown, South Africa

# populate.pl
#           - reads a list of accession numbers stored in a text file
#           - creates a unique id to be used as the PRIMARY KEY
#           - inserts each accession and associated id as a new record into athlee_db

#Additional Perl Modules required
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#open file-handle to read in accessions
open(accessions, "SourceAccessions");

@accessions = <accessions>;

for ($num = 0; $num < scalar@accessions; $num++) {

   chomp @accessions[$num];

   $acc = @accessions[$num];
   $id = substr ($acc, 0, 2) . "_" . $num;    #create id

   #read into athlee_db
   $res = $conn->exec("INSERT INTO Source (id, accession) VALUES ('$id', '$acc');");
}

#close file-handle
close(accessions);
```

Appendix A2 – populateSource.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#populateSource.pl
#     - reads accession and id from athlee_db, Source Table
#     - retrieves corresponding sequence, and selected seqFeatures and annotations from GenBank
#     - populates the relative fields in the Source table

#Additional Perl Modules required
use Bio::Perl;
use Bio::SearchIO;
use Bio::Annotation::Collection;
use Bio::SeqIO;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#read Source.id from db and push onto an array
$res = $conn->exec("SELECT id FROM Source;");

while (@id_nos=$res->fetchrow){
   foreach $id(@id_nos){
      push(@ids,$id);
   }
}

#read Source.accession from db and push onto an array
$res=$conn->exec("SELECT accession FROM Source;");

while (@row=$res->fetchrow){
   foreach $acc(@row){
      push(@accession_nrs,$acc);
   }
}

#define source and seq features to be retrieved
@sourceFeatures = ('organism', 'mol_type', 'product');
@seqfeatures  =  ('CDS',  'gene',  'promoter',  'CAAT_signal',  'TATA_signal',  'polyA_signal',
'misc_feature');

#access GenBank
$gb = new Bio::DB::GenBank();
$gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );

#stream of seqIO objects for list of accessions
$seqio = $gb-> get_Stream_by_acc([@accession_nrs]);
```

```
$x=0; #declare increment variable

#while-loop to go through stream of seqIO objects
while($clone =  $seqio->next_seq) {
   $idnum = @ids[$x];

   #parse seqIO object and id to subroutines
   FetchRefAnnotations($clone, $idnum);
   FetchDescription($clone, $idnum);
   FetchSequence($clone, $idnum);

   #use foreach loop to go through SeqFeatures from each seqIO object
   foreach $feat_object ($clone->get_SeqFeatures) {
      FetchGeneName($feat_object, $idnum);

      #fetch selected source and seq features
      foreach $k(@seqfeatures){
         FetchSeqFeature($k, $feat_object, $idnum);
      }

      foreach $l(@sourceFeatures){
         FetchSourceFeatures($l, $feat_object, $idnum);
      }
   }

   #increment x
   $x++;
}

################################################
sub FetchSeqFeature {

my ($key, $featobj, $identity) = @_;

   #retrieve start, end, and tag values for defined seqFeature
   if ($featobj->primary_tag eq "$key"){

      my $start = $featobj->start;
      my $end = $featobj->end;
      my $field = $featobj->primary_tag;
      @tags = $featobj->get_all_tags();

      if ($featobj->has_tag('note')){

         @notes = $featobj->get_tag_values('note');
         $note = @notes[0];
      }

      #write to db
      print $field, " ", $start, " ", $end, " ", $note, "\n";
```

```
        $res=$conn->exec("UPDATE Source SET $field = '$start - $end, $note' WHERE id =
        '$identity';");
    }
}

sub FetchRefAnnotations {

my ($seqio1, $ident1) = @_;

    #retrieve annotation from seqIO object
    my $anno_collection = $seqio1->annotation();

    #get all reference annotations
    my @annotations = $anno_collection->get_all_Annotations('reference');

    #retrieve original author and location and write to db
    my $authors = @annotations[0]->authors();
    $res=$conn->exec("UPDATE Source SET author = '$authors' WHERE id = '$ident1';");

    my $ref = @annotations[0]->location();
    $res=$conn->exec("UPDATE Source SET reference = '$ref' WHERE id = '$ident1';");
}

sub FetchSourceFeatures {

my ($sourceFeat, $seqio2, $ident2) = @_;

    #retrieve selected source features
    if ($seqio2->has_tag("$sourceFeat")){

        my @feat = $seqio2->get_tag_values("$sourceFeat");
        my $feature = @feat[0];

        #write to db
        $res=$conn->exec("UPDATE Source SET $sourceFeat = '$feature' WHERE id = '$ident2';");
    }
}

sub FetchGeneName {

my ($seqio3, $ident3) = @_;

    #retrieve gene name
    if ($seqio3->has_tag("gene")){
        my @geneName = $seqio3->get_tag_values("gene");
        my $gene = @geneName[0];

        #write to db
        $res=$conn->exec("UPDATE Source SET gene_name = '$gene' WHERE id = '$ident3';");
    }
}
```

```
sub FetchDescription {

my ($seqio4, $ident4) = @_;

   #retrieve description and write to db
   my $description = $seqio4->desc();

   $res=$conn->exec("UPDATE Source SET description = '$description' WHERE id = '$ident4';");
}

sub FetchSequence {

my ($seqio5, $ident5) = @_;

   #retrieve sequence and write to db
   my $retrieved_seq = $seqio5->seq();

   $res=$conn->exec("UPDATE Source SET seq = '$retrieved_seq' WHERE id = '$ident5';");
}
```

Appendix B1 – remoteBlast.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#remoteBlast.pl
# - selects accession, organism, seq from athlee_db, Source Table where mRNA data available
# - write seqs in FASTA format to an input file
# - remote BLAST - parameters => blastn, nr, 1e-10

#Additional Perl Modules required
use Bio::Tools::Run::RemoteBlast;
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#query db for only mRNA records
Pg::doQuery($conn,"SELECT accession, organism, seq FROM Source WHERE mol_type ~*
'mRNA';",\@ary);

#open file handle to write info to text file
open (FILEIN,">mRNAinputseq");

#for-loop to go through array of info read from the db
for $i(0 .. $#ary){
   #split organism name into array i.e. "Arabidopsis thaliana" -> ("Arabidopsis","thaliana")
   @orgName = split(/ /,$ary[$i][1]);

   #write info for all records in FASTA format into an input file
   print FILEIN ">$ary[$i][0]_$orgName[0]_$orgName[1]\n";
   print FILEIN "$ary[$i][2]*\n\n";
}

#close file handle
close FILEIN;

print "Finished generating BLAST input file...\n"; #track progress of program

#BLAST parameters
my $prog = 'blastn'; #nucleotide-nucleotide blast
my $db = 'nr'; #non-redundant db
my $e_val = '1e-10'; #threshold e-value

my @params = ('-prog'      =>$prog,
              '-data'      =>$db,
              '-expect'    =>$e_val,
              '-readmethod'=>'SearchIO');
```

```
#Remote BLAST
my $factory = Bio::Tools::Run::RemoteBlast->new(@params);
$factory->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );

#$v is just to turn messages on and off
my $v = 1;

#Read sequences in FASTA format from input file
my $str = Bio::SeqIO->new(-file=>'mRNAinputseq');

print "File read...\n"; #track progress

#submit_blast($input)
my $r = $factory->submit_blast('mRNAinputseq');

print STDERR "waiting..." if($v>0);

#while there are still more BLAST hits
while(my @rids = $factory->each_rid){
   foreach my $rid(@rids){
      my $rc = $factory->retrieve_blast($rid);
      if(!ref($rc)){
         if($rc<0){
            $factory->remove_rid($rid);
         }
         print STDERR "." if($v>0);
         sleep 5;
      }else{
         my $result = $rc->next_result();
         #save the output
         my $filename = $result->query_name()."\.out";
         $factory->save_output($filename);
         $factory->remove_rid($rid);
         print "\nQuery Name:", $result->query_name(),"\n";

         while(my $hit = $result->next_hit){
            next unless ($v>0);
            print "\tHit name: ", $hit->name,"\n";

            while(my $hsp = $hit->next_hsp){
               print "\t\tScore is ",$hsp->score,"\n";
            }
         }
      }
   }
}
```

Appendix B2 – parse_blast.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#parse_blast.pl
#           - parses Blast Report files filtering out unwanted hits
#           - writes selected info to an output file and to athlee_db, Blast_Results table
#           - retrieves and writes hit sequences to athlee_db, Blast_Results table

#Additional Perl Modules required
use Bio::Perl;
use Bio::SearchIO;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#read all .out filenames in BlastReportFiles directory into and array
my $dir = "/home/g9610081/BlastReportFiles";
my @blastreport_files = <$dir/*.out>;

#open file-handles to write info to text file
open(files,">filenames");
open (genomic_hits,">Genomic_Hits");

#foreach Blast report retrieve just the filename from the file path
foreach $i(@blastreport_files){
   $file = substr($i,15);
   print files "$file\n"; #write to a text file
}

#close file handle
close(files);

#open file handle to read in info
open(filenames,"filenames");
@files = <filenames>;

#declare increment variable
$x=0;

#retrieve accession and organism from each filename
foreach $filename (@files){
   chomp $filename;
   @name = split(/_/,$filename);
   $accession = @name[0];
   @name[2] =~ /(\w+)(.out$)/i;
   $species = $1;
```

```
$genus = @name[1];

#parse each Blast report file
my $in = new Bio::SearchIO(-format => 'blast',
                           -file   => $filename);

#filter results
while(my $result = $in->next_result) {
   while(my $hit = $result->next_hit) {
      while(my $hsp = $hit->next_hsp) {
       #filter level 1 => percent identity
       if($hsp->percent_identity >= 75) {
          #level 2 => same species as query
          if($hit->description =~ /[\s\w]+($species)[\s\w]+/i) {
            #level 3 => filter out mRNA/cDNA hits
            if($hit->description =~/[\s\w]+(mRNA|cDNA)+[\s\w]*/i) {
            } else {
               $x++; #increment x

               #save output to text file
               print genomic_hits "$x Hit:              ", $hit->name,
                                  "\nHit Accession_no:", $hit->accession,
                                  "\nHit Description: ", $hit->description,
                                  "\nLength:          ", $hsp->length('total'),
                                  "\nPercent_id:      ", $hsp->percent_identity,
                                  "\nE-value:         ", $hsp->evalue,
                                  "\nStart (Query):   ", $hsp->start('query'),
                                  "\nEnd (Query):     ", $hsp->end('query'),
                                  "\nStart (Hit):     ", $hsp->start('hit'),
                                  "\nEnd (Hit):       ", $hsp->end('hit');

               print genomic_hits "\n\n\n";

               #declare and instantiate db variables
               $hit_name = $hit->name;
               $hit_accession = $hit->accession;
               $hit_description = $hit->description;
               $hit_description =~ s/\'/_/g;
               $length = $hsp->length('total');
               $percent_id = $hsp->percent_identity;
               $e_value = $hsp->evalue;
               $start_query = $hsp->start('query');
               $end_query = $hsp->end('query');
               $start_hit = $hsp->start('hit');
               $end_hit = $hsp->end('hit');

               #fetch hit sequences
               if ($hit_name =~/^gb/){
                  $hit_seq=FetchGBSequence($hit_accession);
               }else{
                  $hit_seq=FetchEMBLSequence($hit_accession);
               }
```

```
                #write output to athlee_db, Blast_Results table
                $res=$conn->exec("INSERT INTO Blast_Results (accession, hit_name,
                hit_accession, hit_description, length, percent_id, e_value, start_query,
                end_query, start_hit, end_hit, hit_seq) VALUES ('$accession', '$hit_name',
                '$hit_accession', '$hit_description', '$length', '$percent_id', '$e_value',
                '$start_query', '$end_query', '$start_hit', '$end_hit', '$hit_seq');");
              }
            }
          }
        }
      }
    }
}

print "$x iterations\n";

#close file handles
close(filenames);
close(genomic_hits);

#################################################################
sub FetchGBSequence{

my $search_term = $_[0];
    chomp $search_term;
    $gb = new Bio::DB::GenBank(); #access GenBank
    $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $gb->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->seq();
    $retrieved_seq;
}

sub FetchEMBLSequence{

my $search_term = $_[0];
    chomp $search_term;
    $embl = new Bio::DB::EMBL(); #access EMBL
    $embl->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $embl->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->seq();
    $retrieved_seq;
}
```

Appendix B3 – Blast_Results Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	text	Accession number of the query mRNA record
Hit Name	text	e.g. gb|AC003970.1|
Hit Accession	text	Hit accession number
Hit Description	text	Description of hit nucleotide
Length	int	Length of hit with the query
Percent Id	int	Percent identity between hit and query
E-Value	int	E-value score of match
Start query	int	Start position of match on the query sequence
End query	int	End position of match on the query sequence
Start hit	int	Start position of match on the hit sequence
End hit	int	End position of match on the hit sequence
Hit seq	text	Hit Sequence
Link	text	FOREIGN KEY referencing Source.id

Appendix B4 – Search_SeqInfo Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	varchar(80)	Accession number of the query mRNA record
Hit Accession	varchar(80)	e.g. gb|AC003970.1|
TESSprom_start	int	2000 bps upstream of gene start
TESSprom_end	int	1bp upstream of gene start
Intron1_start	Int	Start position of Intron 1
Intron1_end	int	End position of Intron 1
Intron2_start	int	Start position of Intron 2
Intron2_end	int	End position of Intron 2
↕	↕	↕
Intron8_start	int	Start position of Intron 8
Intro8_end	int	End position of Intron 8
TRESprom_start	int	1000 bps upstream of gene start
TRESprom_end	int	1bp upstream of gene start
Link	text	FOREIGN KEY referencing Source.id

Appendix B5 – searchSeqFetch.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#searchSeqFetch.pl
#     - reads promoter start and end info from athlee_db, Search_SeqInfo table
#     - uses info to retrieve promoter subsequences
#     - writes promoter subseqs to athlee_db, Search_Seqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Search_SeqInfo table
$res=$conn->exec("SELECT accession FROM Search_SeqInfo;");

while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT hit_accession FROM Search_SeqInfo;");

while (@rows=$res->fetchrow){
   foreach $line(@rows){
      push(@hitaccession_nrs,$line);
   }
}

$res=$conn->exec("SELECT TESSprom_start FROM Search_SeqInfo;");

while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}

$res=$conn->exec("SELECT TESSprom_end FROM Search_SeqInfo;");

while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}
```

```
$res=$conn->exec("SELECT TRESpromarea_start FROM Search_SeqInfo;");

while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
      push(@startnrs2,$start2);
   }
}

$res=$conn->exec("SELECT TRESpromarea_end FROM Search_SeqInfo;");

while (@rows22=$res->fetchrow){
   foreach $end2(@rows22){
      push(@endnrs2,$end2);
   }
}

#declare increment variable
$index = 0;

#retrieve promoter subsequences and write to athlee_db, Search_Seqs table
foreach $entry(@hitaccession_nrs){
   print "Fetching $entry...\n"; #track progress

   $accession=@accession_nrs[$index];
   $promoter2000=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
   $promoter1000=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);

   print "Writing promoter sequence to database...\n"; #track progress

   $res=$conn->exec("INSERT INTO Search_Seqs (accession, hit_accession, promoter2000,
   promoter1000) VALUES ('$accession', '$entry','$promoter2000', '$promoter1000');");
   $res=$conn->exec("UPDATE Search_Seqs SET seq = Blast_Results.hit_seq where hit_accession =
   Blast_Results.hit_accession;");

   $index++; #increment
}

##################################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;
   $gb = new Bio::DB::GenBank(); #access GenBank
   $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
   $seqobj = $gb->get_Seq_by_acc($search_term);
   $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve sub-sequence
   $retrieved_seq;
}
```

Appendix B6 – FetchIntrons.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#FetchIntrons.pl
#       - reads intron start and end info from athlee_db, Search_SeqInfo table
#       - uses info to retrieve intron subsequences
#       - writes intron subseqs to athlee_db, Search_Seqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Search_SeqInfo table
$res=$conn->exec("SELECT accession FROM Search_SeqInfo;");

while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT link FROM Search_SeqInfo;");

while (@row1=$res->fetchrow){
   foreach $ident(@row1){
      push(@ids,$ident);
   }
}

$res=$conn->exec("SELECT hit_accession FROM Search_SeqInfo;");

while (@rows=$res->fetchrow){
   foreach $line(@rows){
      push(@hitaccession_nrs,$line);
   }
}

$res=$conn->exec("SELECT intron1_start FROM Search_SeqInfo;");

while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}
```

```
$res=$conn->exec("SELECT intron1_end FROM Search_SeqInfo;");

while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}

$res=$conn->exec("SELECT intron2_start FROM Search_SeqInfo;");

while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
      push(@startnrs2,$start2);
   }
}

$res=$conn->exec("SELECT intron2_end FROM Search_SeqInfo;");

while (@rows22=$res->fetchrow){
   foreach $end2(@rows22){
      push(@endnrs2,$end2);
   }
}

$res=$conn->exec("SELECT intron3_start FROM Search_SeqInfo;");

while (@rows3=$res->fetchrow){
   foreach $start3(@rows3){
      push(@startnrs3,$start3);
   }
}

$res=$conn->exec("SELECT intron3_end FROM Search_SeqInfo;");

while (@rows33=$res->fetchrow){
   foreach $end3(@rows33){
      push(@endnrs3,$end3);
   }
}

$res=$conn->exec("SELECT intron4_start FROM Search_SeqInfo;");

while (@rows4=$res->fetchrow){
   foreach $start4(@rows4){
      push(@startnrs4,$start4);
   }
}

$res=$conn->exec("SELECT intron4_end FROM Search_SeqInfo;");
```

```
while (@rows44=$res->fetchrow){
   foreach $end4(@rows44){
      push(@endnrs4,$end4);
   }
}

$res=$conn->exec("SELECT intron5_start FROM Search_SeqInfo;");

while (@rows5=$res->fetchrow){
   foreach $start5(@rows5){
      push(@startnrs5,$start5);
   }
}

$res=$conn->exec("SELECT intron5_end FROM Search_SeqInfo;");

while (@rows55=$res->fetchrow){
   foreach $end5(@rows55){
      push(@endnrs5,$end5);
   }
}

$res=$conn->exec("SELECT intron6_start FROM Search_SeqInfo;");

while (@rows6=$res->fetchrow){
   foreach $start6(@rows6){
      push(@startnrs6,$start6);
   }
}

$res=$conn->exec("SELECT intron6_end FROM Search_SeqInfo;");

while (@rows66=$res->fetchrow){
   foreach $end6(@rows66){
      push(@endnrs6,$end6);
   }
}

$res=$conn->exec("SELECT intron7_start FROM Search_SeqInfo;");

while (@rows7=$res->fetchrow){
   foreach $start7(@rows7){
      push(@startnrs7,$start7);
   }
}

$res=$conn->exec("SELECT intron7_end FROM Search_SeqInfo;");

while (@rows77=$res->fetchrow){
   foreach $end7(@rows77){
      push(@endnrs7,$end7);
   }
```

```
}

$res=$conn->exec("SELECT intron8_start FROM Search_SeqInfo;");

while (@rows8=$res->fetchrow){
   foreach $start8(@rows8){
      push(@startnrs8,$start8);
   }
}

$res=$conn->exec("SELECT intron8_end FROM Search_SeqInfo;");

while (@rows88=$res->fetchrow){
   foreach $end8(@rows88){
      push(@endnrs8,$end8);
   }
}

#declare increment variable
$index = 0;

#retrieve intron subsequences and write to athlee_db, Search_Seqs table
foreach $entry(@hitaccession_nrs){
   #id of record
   $id = @ids[$index];

   print "Fetching introns from hit: $entry...\n"; #track progress

   $intron1=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
   $intron2=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);
   $intron3=FetchSequence($entry,@startnrs3[$index],@endnrs3[$index]);
   $intron4=FetchSequence($entry,@startnrs4[$index],@endnrs4[$index]);
   $intron5=FetchSequence($entry,@startnrs5[$index],@endnrs5[$index]);
   $intron6=FetchSequence($entry,@startnrs6[$index],@endnrs6[$index]);
   $intron7=FetchSequence($entry,@startnrs7[$index],@endnrs7[$index]);
   $intron8=FetchSequence($entry,@startnrs8[$index],@endnrs8[$index]);

   #track progress
   print "Writing entry to database...\n";

   #write to athlee_db, Search_Seqs table
   $res=$conn->exec("UPDATE Search_Seqs SET intron1 = '$intron1' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron2 = '$intron2' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron3 = '$intron3' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron4 = '$intron4' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron5 = '$intron5' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron6 = '$intron6' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron7 = '$intron7' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Search_Seqs SET intron8 = '$intron8' WHERE link = '$id';");

   #increment
   $index++;
```

```
}

#if no subseq to retrieve, enters a '1' - clear from db
$res=$conn->exec("UPDATE Search_Seqs SET intron1 = '' WHERE intron1 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron2 = '' WHERE intron2 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron3 = '' WHERE intron3 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron4 = '' WHERE intron4 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron5 = '' WHERE intron5 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron6 = '' WHERE intron6 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron7 = '' WHERE intron7 = '1';");
$res=$conn->exec("UPDATE Search_Seqs SET intron8 = '' WHERE intron8 = '1';");

##################################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;
   $gb = new Bio::DB::GenBank(); #access GenBank
   $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
   $seqobj = $gb->get_Seq_by_acc($search_term);
   $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve sub-seq
   $retrieved_seq;
}
```

Appendix B7 – Search_Seqs Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	varchar(80)	Accession number of the query mRNA record
Hit Accession	varchar(80)	e.g. gb|AC003970.1|
Promoter2000	Text	2000 bps putative promoter subsequence
Intron1	Text	Intron subsequences
↕	↕	↕
Intron8	Text	
Promoter1000	Text	1000 bps promoter subsequence (TRES only takes 1000bps)
Link	Text	FOREIGN KEY referencing Source.id

Appendix B8 – GenomicSearchInfo.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#GenomicSearchInfo.pl
#    - retrieves promoter, intron and CDS seqFeatures for Genomic records
     - writes start and end info to athlee_db, Genomic_SearchInfo table

#Additional Perl Modules required
use Bio::Perl;
use Bio::SeqIO;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#select link (= Source.id -> foreign key) and push onto an array
$res=$conn->exec("SELECT link FROM Genomic_SearchInfo;");

while (@id_nos=$res->fetchrow){
   foreach $id(@id_nos){
      push(@ids,$id);
   }
}

#select accession and push onto an array
$res=$conn->exec("SELECT accession FROM Genomic_SearchInfo;");

while (@row=$res->fetchrow){
   foreach $acc(@row){
      push(@accession_nrs,$acc);
   }
}

#access GenBank and retrieve seqIO stream
$gb = new Bio::DB::GenBank();
$gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
$seqio = $gb-> get_Stream_by_acc([@accession_nrs]);

#declare increment variable
$x=0;

#retrieve promoter, intron and CDS seqFeatures from each seqIO object
while($clone =  $seqio->next_seq) {
   foreach $feat_object ($clone->get_SeqFeatures) {
      FetchPromoter($feat_object, $ids[$x]);
      FetchIntron($feat_object, $ids[$x]);
      FetchCDSsubseqs($feat_object, $ids[$x]);
```

```
    }
    $x++; #increment x
}

#####################################################
sub FetchPromoter {

my ($featobj, $identity) = @_;
#retrieve promoter seqFeature and write start and end values to athlee_db, Genomic_SearchInfo
if ($featobj->primary_tag eq "promoter"){
    $start = $featobj->start;
    $res=$conn->exec("UPDATE Genomic_SearchInfo SET prom_start = '$start' WHERE link =
    '$identity';");
    $end = $featobj->end;
    $res=$conn->exec("UPDATE Genomic_SearchInfo SET prom_end = '$end' WHERE link =
    '$identity';");
    }
}

sub FetchIntron {

my ($featobj1, $identity1) = @_;
#retrieve intron seqFeatures
if ($featobj1->primary_tag eq "intron"){
    $start1 = $featobj1->start;
    $end1 = $featobj1->end;
    @tags = $featobj1->get_all_tags();

    #get intron number
    if ($featobj1->has_tag('number')){
        @num = $featobj1->get_tag_values('number');
        $n = @num[0];
    }else{
        $n = 1;
    }

    $field = "intron" . $n;
    $field1 = $field . "_start";
    $field2 = $field . "_end";

    #write start and end values to athlee_db, Genomic_SearchInfo table
    $res=$conn->exec("UPDATE Genomic_SearchInfo SET $field1 = '$start1' WHERE link =
    '$identity1';");

    $res=$conn->exec("UPDATE Genomic_SearchInfo SET $field2 = '$end1' WHERE link =
    '$identity1';");
    }
}

sub FetchCDSsubseqs {

my ($featobj2, $identity2) = @_;
```

```
my $y=1; #exon number

   #retrieve CDS splitLocation seqFeatures
      #i.e. CDS => >1 subseq (exons) - can therefore define introns
   if ($featobj2->location->isa('Bio::Location::SplitLocationI')
                                                && $feat_object->primary_tag eq 'CDS') {
      #retrieve start and end of each exon
      foreach $location ($featobj2->location->sub_Location) {
         $End = $location->end;
         $Start = $location->start;
         $StartField = "Start$y";
         $EndField = "End$y";

         #write info to athlee_db, CDS_SeqInfo table
         $res=$conn->exec("UPDATE CDS_SeqInfo SET $StartField = '$Start' WHERE link =
         '$identity2';");

         $res=$conn->exec("UPDATE CDS_SeqInfo SET $EndField = '$End' WHERE link =
         '$identity2';");

         $y++; #increment y
      }
   }
}
```

Appendix B9 – CDS_SeqInfo Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	Text	Accession number of the query mRNA record
Link	Text	FOREIGN KEY referencing Source.id
Start1	Text	Start and end positions of CDS split-location seqFeatures – indicate where the exons start and end. PopulateGenomicSearchInfo.pl uses these values to define the intron regions
End1	Text	
↕	↕	↕
Start9	Text	
End9	Text	

Appendix B10 – populateGenomic_SearchInfo.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#populateGenomic_SearchInfo.pl
#    - defines intron and promoter start and end positions from info obtained from
#      splitLocation CDS seqFeatures
#    - writes info to athlee_db, Genomic_SearchInfo tables

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

for ($x = 1; $x < 14; $x++){
   $y = $x + 1;

   $IntronStart = "intron" . $x ."_start";
   $IntronEnd = "intron" . $x ."_end";
   $CDS_field1 = "CDS_SeqInfo.end$x";
   $CDS_field2 = "CDS_SeqInfo.start$y";

   print $IntronStart, " ", $CDS_field1, "\n";

   $res=$conn->exec("UPDATE Genomic_SearchInfo SET $IntronStart = $CDS_field1 + 1 WHERE link =
   CDS_SeqInfo.link;");

   $res=$conn->exec("UPDATE Genomic_SearchInfo SET $IntronEnd = $CDS_field2 - 1 WHERE link =
   CDS_SeqInfo.link;");
}

$res=$conn->exec("UPDATE Genomic_SearchInfo SET Prom_start = CDS_SeqInfo.start1 - 2000 WHERE
link = CDS_SeqInfo.link;");
$res=$conn->exec("UPDATE Genomic_SearchInfo SET Prom_end = CDS_SeqInfo.start1 - 1 WHERE link =
CDS_SeqInfo.link;");
```

Appendix B11 – Genomic_SearchInfo Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	Text	Accession number of the genomic DNA record
Prom_start	Int	Upstream of gene start
Prom_end	Int	1bp upstream of gene start
Intron1_start	Int	Start position of Intron 1
Intron1_end	Int	End position of Intron 1
Intron2_start	Int	Start position of Intron 2
Intron2_end	Int	End position of Intron 2
↕	↕	↕
Intron13_start	Int	Start position of Intron 13
Intro13_end	Int	End position of Intron 13
TESSprom_start	Int	2000 bps upstream of gene start
TESSprom_end	Int	1bp upstream of gene start
Link	Text	FOREIGN KEY referencing Source.id

Appendix B12 – searchSeqFetch2.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa
#searchSeqFetch2.pl
#     - reads promoter start and end info from athlee_db, Genomic_SearchInfo table
#     - uses info to retrieve promoter subsequences
#     - writes promoter subseqs to athlee_db, Genomic_SearchSeqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Genomic_SearchInfo table
$res=$conn->exec("SELECT accession FROM Genomic_SearchInfo;");
while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT link FROM Genomic_SearchInfo;");
while (@row1=$res->fetchrow){
   foreach $number1(@row1){
      push(@ids,$number1);
   }
}

$res=$conn->exec("SELECT prom_start FROM Genomic_SearchInfo;");
while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}

$res=$conn->exec("SELECT prom_end FROM Genomic_SearchInfo;");
while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}

$res=$conn->exec("SELECT TESSprom_start FROM Genomic_SearchInfo;");
while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
      push(@startnrs2,$start2);
```

```
    }
}

$res=$conn->exec("SELECT TESSprom_end FROM Genomic_SearchInfo;");
while (@rows22=$res->fetchrow){
    foreach $end2(@rows22){
        push(@endnrs2,$end2);
    }
}

#declare increment variable
$index = 0;

#retrieve promoter subsequences and write to athlee_db, Genomic_SearchSeqs table
foreach $entry(@accession_nrs){
    print "Fetching $entry...\n"; #track progress

    $accession = @accession_nrs[$index];
    $id = @ids[$index];
    $prom=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
    $prom2000=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);

    print "Writing promoter sequences to the database...\n"; #track progress

    $res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter = '$prom' WHERE link = '$id';");
    $res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter2000 = '$prom2000' WHERE link =
    '$id';");

    $index++; #increment
}

#if no subseq to retrieve, enters a '1' - clear from db
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter = '' WHERE promoter = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET promoter2000 = '' WHERE promoter2000 = '1';");

##################################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;

    $gb = new Bio::DB::GenBank(); #access GenBank
    $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $gb->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve subseqs
    $retrieved_seq;
}
```

Appendix B13 – FetchIntrons2.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#FetchIntrons2.pl
#    - reads intron start and end info from athlee_db, Genomic_SearchInfo table
#    - uses info to retrieve intron subsequences
#    - writes intron subseqs to athlee_db, Search_Seqs table

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#retrieve info from athlee_db, Genomic_SearchInfo table
$res=$conn->exec("SELECT accession FROM Genomic_SearchInfo;");
while (@row=$res->fetchrow){
   foreach $number(@row){
      push(@accession_nrs,$number);
   }
}

$res=$conn->exec("SELECT link FROM Genomic_SearchInfo;");
while (@rows=$res->fetchrow){
   foreach $line(@rows){
      push(@ids,$line);
   }
}

$res=$conn->exec("SELECT intron1_start FROM Genomic_SearchInfo;");
while (@rows1=$res->fetchrow){
   foreach $start1(@rows1){
      push(@startnrs1,$start1);
   }
}

$res=$conn->exec("SELECT intron1_end FROM Genomic_SearchInfo;");
while (@rows11=$res->fetchrow){
   foreach $end1(@rows11){
      push(@endnrs1,$end1);
   }
}

$res=$conn->exec("SELECT intron2_start FROM Genomic_SearchInfo;");
while (@rows2=$res->fetchrow){
   foreach $start2(@rows2){
```

```
      push(@startnrs2,$start2);
   }
}

$res=$conn->exec("SELECT intron2_end FROM Genomic_SearchInfo;");
while (@rows22=$res->fetchrow){
   foreach $end2(@rows22){
      push(@endnrs2,$end2);
   }
}

$res=$conn->exec("SELECT intron3_start FROM Genomic_SearchInfo;");
while (@rows3=$res->fetchrow){
   foreach $start3(@rows3){
      push(@startnrs3,$start3);
   }
}

$res=$conn->exec("SELECT intron3_end FROM Genomic_SearchInfo;");
while (@rows33=$res->fetchrow){
   foreach $end3(@rows33){
      push(@endnrs3,$end3);
   }
}

$res=$conn->exec("SELECT intron4_start FROM Genomic_SearchInfo;");
while (@rows4=$res->fetchrow){
   foreach $start4(@rows4){
      push(@startnrs4,$start4);
   }
}

$res=$conn->exec("SELECT intron4_end FROM Genomic_SearchInfo;");
while (@rows44=$res->fetchrow){
   foreach $end4(@rows44){
      push(@endnrs4,$end4);
   }
}

$res=$conn->exec("SELECT intron5_start FROM Genomic_SearchInfo;");
while (@rows5=$res->fetchrow){
   foreach $start5(@rows5){
      push(@startnrs5,$start5);
   }
}

$res=$conn->exec("SELECT intron5_end FROM Genomic_SearchInfo;");
while (@rows55=$res->fetchrow){
   foreach $end5(@rows55){
      push(@endnrs5,$end5);
   }
}
```

```
$res=$conn->exec("SELECT intron6_start FROM Genomic_SearchInfo;");
while (@rows6=$res->fetchrow){
   foreach $start6(@rows6){
      push(@startnrs6,$start6);
   }
}

$res=$conn->exec("SELECT intron6_end FROM Genomic_SearchInfo;");
while (@rows66=$res->fetchrow){
   foreach $end6(@rows66){
      push(@endnrs6,$end6);
   }
}

$res=$conn->exec("SELECT intron7_start FROM Genomic_SearchInfo;");
while (@rows7=$res->fetchrow){
   foreach $start7(@rows7){
      push(@startnrs7,$start7);
   }
}

$res=$conn->exec("SELECT intron7_end FROM Genomic_SearchInfo;");
while (@rows77=$res->fetchrow){
   foreach $end7(@rows77){
      push(@endnrs7,$end7);
   }
}

$res=$conn->exec("SELECT intron8_start FROM Genomic_SearchInfo;");
while (@rows8=$res->fetchrow){
   foreach $start8(@rows8){
      push(@startnrs8,$start8);
   }
}

$res=$conn->exec("SELECT intron8_end FROM Genomic_SearchInfo;");
while (@rows88=$res->fetchrow){
   foreach $end8(@rows88){
      push(@endnrs8,$end8);
   }
}

$res=$conn->exec("SELECT intron9_start FROM Genomic_SearchInfo;");
while (@rows9=$res->fetchrow){
   foreach $start9(@rows9){
      push(@startnrs9,$start9);
   }
}

$res=$conn->exec("SELECT intron9_end FROM Genomic_SearchInfo;");
while (@rows99=$res->fetchrow){
```

```
   foreach $end9(@rows99){
      push(@endnrs9,$end9);
   }
}

$res=$conn->exec("SELECT intron10_start FROM Genomic_SearchInfo;");
while (@rows10=$res->fetchrow){
   foreach $start10(@rows10){
      push(@startnrs10,$start10);
   }
}

$res=$conn->exec("SELECT intron10_end FROM Genomic_SearchInfo;");
while (@rows1010=$res->fetchrow){
   foreach $end10(@rows1010){
      push(@endnrs10,$end10);
   }
}

$res=$conn->exec("SELECT intron11_start FROM Genomic_SearchInfo;");
while (@rows11=$res->fetchrow){
   foreach $start11(@rows11){
      push(@startnrs11,$start11);
   }
}

$res=$conn->exec("SELECT intron11_end FROM Genomic_SearchInfo;");
while (@rows1111=$res->fetchrow){
   foreach $end11(@rows1111){
      push(@endnrs11,$end11);
   }
}

$res=$conn->exec("SELECT intron12_start FROM Genomic_SearchInfo;");
while (@rows12=$res->fetchrow){
   foreach $start12(@rows12){
      push(@startnrs12,$start12);
   }
}

$res=$conn->exec("SELECT intron12_end FROM Genomic_SearchInfo;");
while (@rows1212=$res->fetchrow){
   foreach $end12(@rows1212){
      push(@endnrs12,$end12);
   }
}

$res=$conn->exec("SELECT intron13_start FROM Genomic_SearchInfo;");
while (@rows13=$res->fetchrow){
   foreach $start13(@rows13){
      push(@startnrs13,$start13);
   }
```

```
}

$res=$conn->exec("SELECT intron13_end FROM Genomic_SearchInfo;");
while (@rows1313=$res->fetchrow){
   foreach $end13(@rows1313){
      push(@endnrs13,$end13);
   }
}

#declare increment variable
$index = 0;

#retrieve intron subsequences and write to athlee_db, Genomic_SearchInfo table
foreach $entry(@accession_nrs){
   #id of record
   $id = @ids[$index];

   print "Fetching introns from: $entry...\n"; #track progress

   $intron1=FetchSequence($entry,@startnrs1[$index],@endnrs1[$index]);
   $intron2=FetchSequence($entry,@startnrs2[$index],@endnrs2[$index]);
   $intron3=FetchSequence($entry,@startnrs3[$index],@endnrs3[$index]);
   $intron4=FetchSequence($entry,@startnrs4[$index],@endnrs4[$index]);
   $intron5=FetchSequence($entry,@startnrs5[$index],@endnrs5[$index]);
   $intron6=FetchSequence($entry,@startnrs6[$index],@endnrs6[$index]);
   $intron7=FetchSequence($entry,@startnrs7[$index],@endnrs7[$index]);
   $intron8=FetchSequence($entry,@startnrs8[$index],@endnrs8[$index]);
   $intron9=FetchSequence($entry,@startnrs9[$index],@endnrs9[$index]);
   $intron10=FetchSequence($entry,@startnrs10[$index],@endnrs10[$index]);
   $intron11=FetchSequence($entry,@startnrs11[$index],@endnrs11[$index]);
   $intron12=FetchSequence($entry,@startnrs12[$index],@endnrs12[$index]);
   $intron13=FetchSequence($entry,@startnrs13[$index],@endnrs13[$index]);

   print "Writing entry to database...\n"; #track progress

   #write to athlee_db, Genomic_Search_Seqs table
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron1 = '$intron1' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron2 = '$intron2' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron3 = '$intron3' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron4 = '$intron4' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron5 = '$intron5' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron6 = '$intron6' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron7 = '$intron7' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron8 = '$intron8' WHERE link = '$id';");
   $res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron9 = '$intron9' WHERE link = '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron10  =  '$intron10'  WHERE  link  =
   '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron11  =  '$intron11'  WHERE  link  =
   '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron12  =  '$intron12'  WHERE  link  =
   '$id';");
   $res=$conn->exec("UPDATE  Genomic_SearchSeqs  SET  intron13  =  '$intron13'  WHERE  link  =
```

```
    '$id';");

    $index++; #increment
}

#if no subseq to retrieve, enters a '1' - clear from db
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron1 = '' WHERE intron1 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron2 = '' WHERE intron2 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron3 = '' WHERE intron3 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron4 = '' WHERE intron4 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron5 = '' WHERE intron5 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron6 = '' WHERE intron6 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron7 = '' WHERE intron7 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron8 = '' WHERE intron8 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron9 = '' WHERE intron9 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron10 = '' WHERE intron10 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron11 = '' WHERE intron11 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron12 = '' WHERE intron12 = '1';");
$res=$conn->exec("UPDATE Genomic_SearchSeqs SET intron13 = '' WHERE intron13 = '1';");

##################################################
sub FetchSequence {

my ($search_term, $startnr, $endnr) = @_;

    $gb = new Bio::DB::GenBank(); #access GenBank
    $gb->proxy(['http'], 'http://g9610081:733700829@cache3.ru.ac.za:3128' );
    $seqobj = $gb->get_Seq_by_acc($search_term);
    $retrieved_seq = $seqobj->subseq($startnr,$endnr); #retrieve sub-seq
    $retrieved_seq;
}
```

Appendix B14 – Genomic_SearchSeqs Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Accession	Text	Accession number of the query mRNA record
Link	Text	FOREIGN KEY referencing Source.id
Promoter	Text	Promoter subsequence
Intron1	Text	Intron subsequences
↕	↕	↕
Intron13	Text	
Promoter2000	Text	2000bp promoter subsequence

Appendix C1 – PromoterFastaFiles.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#PromoterFastaFiles.pl
#    - reads promoter sequences from athlee_db, Search_Seqs and Genomic_SearchSeqs tables
#    - writes in FASTA format to a series of input files
#    - used as input for TESS - promoter searching program

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#mRNA Promoter Sequences
Pg::doQuery($conn,"SELECT accession, hit_accession, promoter2000, link FROM
Search_Seqs;",\@ary);
open (FILEIN,">TESSinputFiles\/mRNA_Promoter2000Fastas");

for $i(0 .. $#ary){
   print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_PromoterSequence_2000bps\n";
   print FILEIN "$ary[$i][2]*\n\n";
}
close FILEIN;

Pg::doQuery($conn,"SELECT accession, hit_accession, promoter1000, link FROM
Search_Seqs;",\@ary1);
open (FILEIN1,">TRESinputFiles\/mRNA_Promoter1000Fastas");

for $j(0 .. $#ary1){
   print FILEIN1 ">ID|$ary1[$j][3]_Source|$ary1[$j][0]_Hit|$ary1[$j][1]_PromoterSequence_1000bps\n";
   print FILEIN1 "$ary1[$j][2]*\n\n";
}
close FILEIN1;

#Genomic DNA Promoter Sequences
Pg::doQuery($conn,"SELECT accession, link, promoter2000 FROM Genomic_SearchSeqs;",\@ary2);
open (FILEIN2,">TESSinputFiles\/GenomicDNA_Promoter2000Fastas");

for $k(0 .. $#ary2){
   print FILEIN2 ">ID|$ary2[$k][1]_Accession|$ary2[$k][0]_PromoterSequence_2000bps\n";
   print FILEIN2 "$ary2[$k][2]*\n\n";
}
close FILEIN2;
```

Appendix C2 – IntronFastaFiles.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#IntronFastaFiles.pl
#    - reads intron sequences from athlee_db, Search_Seqs and Genomic_SearchSeqs tables
#    - writes in FASTA format to a series of input files
#    - used as input for TESS - promoter searching program

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#mRNA Intron Sequences
Pg::doQuery($conn,"SELECT accession, hit_accession, intron1, link FROM Search_Seqs;",\@ary);
open (FILEIN,">TESSinputFiles\/mRNA_Intron1Fastas");

for $i(0 .. $#ary){
   print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_Intron1\n";
   print FILEIN "$ary[$i][2]\n\n";
}
close FILEIN;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron2, link FROM Search_Seqs;",\@ary1);
open (FILEIN1,">TESSinputFiles\/mRNA_Intron2Fastas");

for $j(0 .. $#ary1){
   print FILEIN1 ">ID|$ary1[$j][3]_Source|$ary1[$j][0]_Hit|$ary1[$j][1]_Intron2\n";
   print FILEIN1 "$ary1[$j][2]\n\n";
}
close FILEIN1;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron3, link FROM Search_Seqs;",\@ary2);
open (FILEIN2,">TESSinputFiles\/mRNA_Intron3Fastas");

for $k(0 .. $#ary2){
   print FILEIN2 ">ID|$ary2[$k][3]_Source|$ary2[$k][0]_Hit|$ary2[$k][1]_Intron3\n";
   print FILEIN2 "$ary2[$k][2]\n\n";
}
close FILEIN2;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron4, link FROM Search_Seqs;",\@ary3);
open (FILEIN3,">TESSinputFiles\/mRNA_Intron4Fastas");

for $l(0 .. $#ary3){
```

```
   print FILEIN3 ">ID|$ary3[$l][3]_Source|$ary3[$l][0]_Hit|$ary3[$l][1]_Intron4\n";
   print FILEIN3 "$ary3[$l][2]\n\n";
}
close FILEIN3;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron5, link FROM Search_Seqs;",\@ary4);
open (FILEIN4,">TESSinputFiles\/mRNA_Intron5Fastas");

for $m(0 .. $#ary4){
   print FILEIN4 ">ID|$ary4[$m][3]_Source|$ary4[$m][0]_Hit|$ary4[$m][1]_Intron5\n";
   print FILEIN4 "$ary4[$m][2]\n\n";
}
close FILEIN4;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron6, link FROM Search_Seqs;",\@ary5);
open (FILEIN5,">TESSinputFiles\/mRNA_Intron6Fastas");

for $n(0 .. $#ary5){
   print FILEIN5 ">ID|$ary5[$n][3]_Source|$ary5[$n][0]_Hit|$ary5[$n][1]_Intron6\n";
   print FILEIN5 "$ary5[$n][2]\n\n";
}
close FILEIN5;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron7, link FROM Search_Seqs;",\@ary6);
open (FILEIN6,">TESSinputFiles\/mRNA_Intron7Fastas");

for $o(0 .. $#ary6){
   print FILEIN6 ">ID|$ary6[$o][3]_Source|$ary6[$o][0]_Hit|$ary6[$o][1]_Intron7\n";
   print FILEIN6 "$ary6[$o][2]\n\n";
}
close FILEIN6;

Pg::doQuery($conn,"SELECT accession, hit_accession, intron8, link FROM Search_Seqs;",\@ary8);
open (FILEIN8,">TESSinputFiles\/mRNA_Intron8Fastas");

for $pp(0 .. $#ary8){
   print FILEIN8 ">ID|$ary8[$pp][3]_Source|$ary8[$pp][0]_Hit|$ary8[$pp][1]_Intron8\n";
   print FILEIN8 "$ary8[$pp][2]\n\n";
}
close FILEIN8;

#Genomic DNA Intron Sequences
Pg::doQuery($conn,"SELECT accession, link, intron1 FROM Genomic_SearchSeqs;",\@ary11);
open (FILEIN11,">TESSinputFiles\/GenomicDNA_Intron1Fastas");

for $q(0 .. $#ary11){
   print FILEIN11 ">ID|$ary11[$q][1]_Accession|$ary11[$q][0]_Intron1\n";
   print FILEIN11 "$ary11[$q][2]\n\n";
}
close FILEIN11;

Pg::doQuery($conn,"SELECT accession, link, intron2 FROM Genomic_SearchSeqs;",\@ary22);
```

```
open (FILEIN22,">TESSinputFiles\/GenomicDNA_Intron2Fastas");

for $r(0 .. $#ary22){
   print FILEIN22 ">ID|$ary22[$r][1]_Accession|$ary22[$r][0]_Intron2\n";
   print FILEIN22 "$ary22[$r][2]\n\n";
}
close FILEIN22;

Pg::doQuery($conn,"SELECT accession, link, intron3 FROM Genomic_SearchSeqs;",\@ary33);
open (FILEIN33,">TESSinputFiles\/GenomicDNA_Intron3Fastas");

for $s(0 .. $#ary33){
   print FILEIN33 ">ID|$ary33[$s][1]_Accession|$ary33[$s][0]_Intron3\n";
   print FILEIN33 "$ary33[$s][2]\n\n";
}
close FILEIN33;

Pg::doQuery($conn,"SELECT accession, link, intron4 FROM Genomic_SearchSeqs;",\@ary44);
open (FILEIN44,">TESSinputFiles\/GenomicDNA_Intron4Fastas");

for $t(0 .. $#ary44){
   print FILEIN44 ">ID|$ary44[$t][1]_Accession|$ary44[$t][0]_Intron4\n";
   print FILEIN44 "$ary44[$t][2]\n\n";
}
close FILEIN44;

Pg::doQuery($conn,"SELECT accession, link, intron5 FROM Genomic_SearchSeqs;",\@ary55);
open (FILEIN55,">TESSinputFiles\/GenomicDNA_Intron5Fastas");

for $u(0 .. $#ary55){
   print FILEIN55 ">ID|$ary55[$u][1]_Accession|$ary55[$u][0]_Intron5\n";
   print FILEIN55 "$ary55[$u][2]\n\n";
}
close FILEIN55;

Pg::doQuery($conn,"SELECT accession, link, intron6 FROM Genomic_SearchSeqs;",\@ary66);
open (FILEIN66,">TESSinputFiles\/GenomicDNA_Intron6Fastas");

for $v(0 .. $#ary66){
   print FILEIN66 ">ID|$ary66[$v][1]_Accession|$ary66[$v][0]_Intron6\n";
   print FILEIN66 "$ary66[$v][2]\n\n";
}
close FILEIN66;

Pg::doQuery($conn,"SELECT accession, link, intron7 FROM Genomic_SearchSeqs;",\@ary77);
open (FILEIN77,">TESSinputFiles\/GenomicDNA_Intron7Fastas");

for $w(0 .. $#ary77){
   print FILEIN77 ">ID|$ary77[$w][1]_Accession|$ary77[$w][0]_Intron7\n";
   print FILEIN77 "$ary77[$w][2]\n\n";
}
close FILEIN77;
```

```
Pg::doQuery($conn,"SELECT accession, link, intron8 FROM Genomic_SearchSeqs;",\@ary88);
open (FILEIN88,">TESSinputFiles\/GenomicDNA_Intron8Fastas");

for $x(0 .. $#ary88){
   print FILEIN88 ">ID|$ary88[$x][1]_Accession|$ary88[$x][0]_Intron8\n";
   print FILEIN88 "$ary88[$x][2]\n\n";
}
close FILEIN88;

Pg::doQuery($conn,"SELECT accession, link, intron9 FROM Genomic_SearchSeqs;",\@ary99);
open (FILEIN99,">TESSinputFiles\/GenomicDNA_Intron9Fastas");

for $y(0 .. $#ary99){
   print FILEIN99 ">ID|$ary99[$y][1]_Accession|$ary99[$y][0]_Intron9\n";
   print FILEIN99 "$ary99[$y][2]\n\n";
}
close FILEIN99;

Pg::doQuery($conn,"SELECT accession, link, intron10 FROM Genomic_SearchSeqs;",\@ary110);
open (FILEIN110,">TESSinputFiles\/GenomicDNA_Intron10Fastas");

for $z(0 .. $#ary110){
   print FILEIN110 ">ID|$ary110[$z][1]_Accession|$ary110[$z][0]_Intron10\n";
   print FILEIN110 "$ary110[$z][2]\n\n";
}
close FILEIN110;

Pg::doQuery($conn,"SELECT accession, link, intron11 FROM Genomic_SearchSeqs;",\@ary111);
open (FILEIN111,">TESSinputFiles\/GenomicDNA_Intron11Fastas");

for $a(0 .. $#ary111){
   print FILEIN111 ">ID|$ary111[$a][1]_Accession|$ary111[$a][0]_Intron11\n";
   print FILEIN111 "$ary111[$a][2]\n\n";
}
close FILEIN111;

Pg::doQuery($conn,"SELECT accession, link, intron12 FROM Genomic_SearchSeqs;",\@ary112);
open (FILEIN112,">TESSinputFiles\/GenomicDNA_Intron12Fastas");

for $b(0 .. $#ary112){
   print FILEIN112 ">ID|$ary112[$b][1]_Accession|$ary112[$b][0]_Intron12\n";
   print FILEIN112 "$ary112[$b][2]*\n\n";
}
close FILEIN112;
```

Appendix C3 – PromoterFastaFiles2.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#PromoterFastaFiles2.pl
#    - reads promoter sequences from athlee_db, Search_Seqs and Genomic_SearchSeqs tables
#    - writes in FASTA format into individual input files
#    - used as input for NNPP - promoter prediction program

#Additional Perl Modules required
use Bio::Perl;
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#mRNA Promoter Sequences
Pg::doQuery($conn,"SELECT accession, hit_accession, promoter2000, link FROM
Search_Seqs;",\@ary);

for $i(0 .. $#ary){
   open (FILEIN, ">NNPPinputFiles\/$ary[$i][3]_mRNA_Promoter.fa");
   print FILEIN ">ID|$ary[$i][3]_Source|$ary[$i][0]_Hit|$ary[$i][1]_PromoterSequence_2000bps\n";
   print FILEIN "$ary[$i][2]\n\n";
   close FILEIN;
}

#Genomic DNA Promoter Sequences
Pg::doQuery($conn,"SELECT accession, link, promoter2000 FROM Genomic_SearchSeqs;",\@ary2);

for $k(0 .. $#ary2){
   open (FILEIN,">NNPPinputFiles\/$ary2[$k][1]_genomicDNA_Promoter.fa");
   print FILEIN ">ID|$ary2[$k][1]_Accession|$ary2[$k][0]_PromoterSequence_2000bps\n";
   print FILEIN "$ary2[$k][2]\n\n";
   close FILEIN;
}
```

Appendix C4 – NNPPrun.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#NNPPrun.pl
#    - reads all NNPP input filenames into an array
#    - runs each FASTA sequences through NNPP
#    - writes output to individual output files

#read all NNPP input filenames into an array
my $dir = "/home/g9610081/NNPPinputFiles";
my @NNPP_fastaFiles = <$dir/*.fa>;

foreach $file (@NNPP_fastaFiles) {
   $resultFile = 'nnppRESULT_';
   $resultFile .= substr($file,30);

   #open file handle to write results to an output file
   open (FILEOUT, ">NNPPoutputFiles\/$resultFile");

   print FILEOUT "$resultFile\n\n";

   #run through NNPP eukaryotic promoter prediction program - installed locally
   my $result = qx`/home/g9610081/NNPP/NNPP2.2/bin/fa2TDNNpred.linux -t 0.95 -r $file`;

   print FILEOUT "$result\n";

   #close file handle
   close FILEOUT;
}
```

Appendix C5 – parseNNPP.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#parseNNPP.pl
#    - parses all NNPP output files
#    - retrieves highest scoring Transcription Start Site prediction and associated info
#    - writes info to athlee_db, NNPPresults table

#Additional Perl Modules required
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#read all NNPP output filenames from directory into an array
my $dir = "/home/g9610081/NNPPoutputFiles";
my @NNPPresultFiles = <$dir/*.fa>;

my $index = 0; #declare increment variable

foreach my $file (@NNPPresultFiles) {
   #open file handles
   open(IN, "$file");
   open(OUT, ">>\/home\/g9610081\/NNPPresults");

   #declare and initialise variables
   my @hitno = (); my @PatternStart = (); my @PatternEnd = (); my @signalStart = ();
   my @patterns = (); my @confidVals = (); my @strand = (); my $x = 0; my $flag = 0;

   #get unique record identifier from filename
   my @name = split(/\//, $file);
   my @name1 = split(/_/, $name[$#name]);
   my $Source = $name1[1] . "_" . $name1[2];

   #determine if genomic DNA or mRNA record
   my $mol_type = $name1[3];
   if ($mol_type =~ /^mRNA/g) {
      $table = 'Search_SeqInfo';
      $seqTable = 'Search_Seqs';
   }else{
      $table = 'Genomic_SearchInfo';
      $seqTable = 'Source';
   }

   #read promoter start and end info from the relative table in athlee_db
   $res=$conn->exec("SELECT tessprom_start FROM $table WHERE link = '$Source';");
   $PromStart = $res->fetchrow;
```

```
$res=$conn->exec("SELECT tessprom_end FROM $table WHERE link = '$Source';");
$PromEnd = $res->fetchrow;

print OUT ">$Source - Highest Predicted NNPP Transcription Start Site";
while (<IN>) { #while reading the current NNPP report
  if($_ =~ /^(No Hits).*/){
    #put 0 in the various arrays as a place-holder when no hits are found
    my $status = 0;
    push(@hitno, $status);
    push(@PatternStart, $status);
    push(@PatternEnd, $status);
    push(@signalStart, $status);
    push(@patterns, $status);
    push(@confidVals, $status);
    push(@strand, $status);
    $index = 0;
    last;
  }
  elsif ($_ =~ /Prediction for ID.*\d{3,4}\sbases.*$/){
    $lg = split(/ /, $_); #retrieve query length from NNPP result file
    $lgth = substr($lg[3], 1);
  }
  elsif ($_ =~ /.*(reverse strand).*/) { #determine if hits are on the reverse strand
    $flag = 1;
  }
  #read hit information
  elsif ($_ =~ /^Hit (\d+).{2}position.{2}(\d+).{4}(\d+),\ssignal start = (\d+)/){
    $x++;
    push(@hitno, $1);
    $start = $2;
    $end = $3;
    $signal = $4;

    if($flag){ #if hit on reverse strand
      $str = "R";
      push(@strand,$str);
      #Pattern in reverse format but not complement; numbering from back strand!
      #To get forward numbering:
        #y(position_iforward_strand) = N(length_of_sequence) - x(position_in_reverse_seq) + 1
      $start = $lgth - $start + 1;
      $end = $lgth - $end + 1;
      $signal = $lgth - $signal +1;
    } else {
      $str = "F";
      push(@strand,$str);
      $start = $2;
      $end = $3;
      $signal = $4;
    }
    #calculate positions on original sequence
    $region_start = ($PromStart + $start) - 2;
    $region_end = ($PromStart + $end) - 2;
```

```
        $signal_start = ($PromStart + $signal) - 2;
        push(@PatternStart, $region_start);
        push(@PatternEnd, $region_end);
        push(@signalStart, $signal_start)
      }
      elsif ($_ =~ /^Pattern:\s(.*)/) {
        $pattern = $1;
        push(@patterns, $1);
      }
      elsif ($_ =~ /^Prediction:\s(.{8})\-+/) {
        push(@confidVals, $1);
        #find the hit with the highest confidence value
        if ($confidVals[$x-1] > $confidVals[$index]){
          $index = $x-1; #record the index of the highest confidence value thus far
        }
      }
    }

    #check to see if positions on the original sequence are calculated correctly
    if ($seqTable eq 'Source'){
      $idField = 'id';
    }else{
      $idField = 'link';
    }

    $res1=$conn->exec("SELECT seq FROM $seqTable WHERE $idField = '$Source';");
    $sequence = $res1->fetchrow;
    $forwardPattern = substr($sequence, $PatternStart[$index], 51);
    print $Source, " ", $strand[$index], " ", $mol_type, "\n";
    print $patterns[$index], "\n";
    print $forwardPattern, "\n\n";

    #insert parsed results into the db
    $res=$conn->exec("INSERT INTO NNPPresults (link, hitno, confidence, regionstart, regionend,
    signalstart, pattern, strand, length, forwardPattern) VALUES ('$Source', '$hitno[$index]',
    '$confidVals[$index]', '$PatternStart[$index]', '$PatternEnd[$index]',
    '$signalStart[$index]', '$patterns[$index]', '$strand[$index]', '$lgth',
    '$forwardPattern');");

    #print parsed results to a text file
    print OUT "\nLength:\t\t\t", $lgth, "\nHit:\t\t\t", $hitno[$index], "\nConfidence:\t\t",
    $confidVals[$index], "\nRegion:\t\t\t", $PatternStart[$index], " - ", $PatternEnd[$index],
    "\nTSS Signal Start:\t", $signalStart[$index], "\nPattern:\t\t", $patterns[$index],
    "\nStrand:\t\t\t", $strand[$index], "\nForward Pattern:\t", $forwardPattern, "\n\n";
}

#close file handles
close (IN);
close (OUT);
```

Appendix C6 – NNPPresults Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Link	Text	FOREIGN KEY referencing Source.id
Hit Number	Int	NNPP hit where TSS with highest confidence value located
Confidence	Text	Confidence value of TSS prediction
RegionStart	Int	Start position of hit
RegionEnd	Int	End position of hit
SignalStart	Int	Position where putative TSS located on hit sequence
Pattern	Text	Hit sequence
Strand	Text	TSS located on forward (F) or reverse (R) strand
Length	Int	Length of hit region

Appendix C7 – parseTESS.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#parseTESS.pl
#    - parses all TESS report files
#    - retrieves selected information
#    - writes info to a .tess output file

#read all TESS report filenames from directory into an array
my $dir = "/home/g9610081/TESSreports";
my @TESSreports = <$dir/*.tessreport>;

foreach $report (@TESSreports) {
   #open file handle to read TESS report files
   open(IN, "$report");

   #extract info from filename
   @name = split(/\//, $report);
   @name1 = split(/_/, $name[$#name]);
   @name2 = split(/\./,$name1[$#name1]);
   $Source = $name2[0];

   #determine if you're dealing with an mRNA or genomic DNA record
   if ($name[$#name] =~ /^mRNA/g) {
      $table = 'Search_SeqInfo';
      $seqTable = 'Search_Seqs';
   }else{
      $table = 'Genomic_SearchInfo';
      $seqTable = 'Source';
```

```
}

while (<IN>) { #while reading the current TESS report
   chomp;

   #extract record name
   if ($_ =~ /^SEQ\s+(>.+)/) {
      @fileName = split(/\|/,$1);
      @fileName1 = split(/_/,$fileName[1]);
      $fileNm = $fileName1[0] . "_" . $fileName1[1];

      #open file_handle to write parsed info to output file
      open(OUT, ">>TESSoutputFiles\/$fileNm.tess");
      @heading = split(/_/, $fileName[$#fileName]);

   #retrieve HIT info
   } elsif ($_ =~ /^HIT/)  {
      @fields = split(/\s+/,$_);

      #write to output file
      print OUT "$Source  $seqTable  $table  $heading[1]  $fields[2]  $fields[3]  $fields[4]
      $fields[5] $fields[6]  $fields[7]  $fields[8]  $fields[9]  $fields[10]  $fields[11]
      $fields[12] $fields[13]\n\n";
   }
}

#close file handles
close (IN);
close (OUT);
}
```

Appendix C8 – parseTESS2.pl

```
#!/usr/bin/perl

#Athlee Maclear
#MSc Bioinformatics & Computational Molecular Biology 2004
#Rhodes University, Grahamstown, South Africa

#parseTESS2.pl
#    - parses all TESS output files
#    - writes info into athlee_db, TESSresults table

#Additional Perl Modules required
use Pg;

#db connection
$conn = Pg::connectdb("dbname=athlee_db user=g9610081 password=asm120478");
$conn->reset;

#read all TESS output filenames from directory into an array
my $dir = "/home/g9610081/TESSoutputFiles";
my @TESSoutputFiles = <$dir/*.tess>;

foreach $file (@TESSoutputFiles) {
   #open file handle to read TESS output file
   open(IN, "$file");

   #extract record identifier from filename
   @name = split(/\./, $file);
   @name1 = split(/\//, $name[0]);
   $id = $name1[$#name1];

   while (<IN>) { #while reading the current TESS output file
      chomp;

      @fields = split(/  /,$_); #split each line into an array
      $table = $fields[2]; #table

      if ($fields[3] eq 'PromoterSequence') {
         $fieldName = 'TESSprom'; #determine db field name
      }else{
         $fieldName = $fields[3];
      }

      $fieldStart = $fieldName . "_start";
      $fieldEnd = $fieldName . "_end";

      #read start position of promoter on original sequence
      $res=$conn->exec("SELECT $fieldStart FROM $table WHERE link = '$id';");
      $start = $res->fetchrow;

      #calculate start position of site on the original sequence
      $site_start = $start + $fields[6] - 2;
```

```
    #length of site
    $length = $fields[8];

    #Check to see if $site_start defined correctly
    $seqTable = $fields[1];

    if ($seqTable eq 'Source'){
       $idField = 'id';
    }else{
       $idField = 'link';
    }

    $res1=$conn->exec("SELECT seq FROM $seqTable WHERE $idField = '$id';");
    $sequence = $res1->fetchrow;
    $site_seq = substr($sequence, $site_start, $length);

    print $site_seq, "\n";

    #TRANSFAC site name
    $TRANSFACsite = substr($fields[4],6);

    #site sequence
    $seq = $fields[9];

    #Source - user-defined or from TRANSFAC
    $Source = $fields[0];

    #insert into athlee_db, TESSresults table
    $res=$conn->exec("INSERT INTO TESSresults (link, site_name, source, site_start,
    site_length, sequence) VALUES ('$id', '$TRANSFACsite', '$Source', '$site_start',
    '$length', '$seq');");
  }

  #close file handles
  close (IN);
}
```

Appendix C9 – TESSresults Table

Field Name	PostgreSQL Field Format	Description
Id	serial PRIMARY KEY	Auto-incrementing number used as the primary key for the table
Link	Text	FOREIGN KEY referencing Source.id
Site Name	Text	Site name located
Source	Text	TRANSFAC or UserDefined Site
Site_Start	Int	Start position of site
Site_End	Int	End position of site
Sequence	Text	Sequence of site located

Appendix D1 – Dbsearch.java

```
//Athlee Maclear
//MSc Bioinformatics & Computational Molecular Biology 2004
//Rhodes University, Grahamstown, South Africa
//DBsearch.java - creates a GUI interface to search athlee_db
//                - user selects search criteria
//

//import library classes needed
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class DBsearch
{
   //Declare and initialize variables
   private int WIDTH = 550;
   private int HEIGHT = 400;

   private String searchFieldA = "";
   private String searchFieldB = "";
   private String searchFieldC = "";

   //Declare containers and components
   private JFrame frame;
   private JPanel searchPanel1, searchPanel2, searchPanel3, resultSelection;
   private JPanel instructionPanel, searchPanel, buttonPanel;
   private JPanel innerPanel, outerPanel;
   private JLabel title, instructionLabel1, instructionLabel2, label1, label2, label3;
   private JButton search;
   private JTextField searchTerm1, searchTerm2, searchTerm3;
   private JList searchList1, searchList2, searchList3;
   private JScrollPane mainScroll, scrollLst1, scrollLst2, scrollLst3, scrollLst4;

   //Declare and initialize display variables
   private Color titleColour = Color.getHSBColor(0.5F,0.5F,0.5F);
   private Color labelColour = Color.getHSBColor(0.7F,0.5F,0.5F);
   private Color label1Colour = Color.getHSBColor(0.8F,0.4F,0.5F);
   private Font titleFont = new Font("Serif", Font.BOLD + Font.ITALIC, 26);
   private Font labelFont1 = new Font("Serief", Font.BOLD, 16);
   private Font labelFont2 = new Font("Serif", Font.BOLD, 12);
   private Font listFont = new Font("Serif", Font.BOLD, 12);

   // Declare and initialize data array
   private String [] sourceFields ={"ID",
            "ACCESSION",
            "GENE_NAME",
            "ORGANISM",
            "DESCRIPTION",
            "AUTHOR",
            "REFERENCE",
```

```
        "MOL_TYPE",
        "PRODUCT"
        };

private String [] tessFields ={"LINK",
        "SITE_NAME",
        "SOURCE",
        };

private String [] nnppFields = {"LINK",
        "CONFIDENCE",
        "STRAND",
        };

private String [] results;

//Constructor
public DBsearch()
{
   //Instantiate and define GUI components
   frame = new JFrame ("Search Plant Stress Response DataBase");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.setPreferredSize(new Dimension(WIDTH, HEIGHT));

   title = new JLabel("Abiotic Plant Stress Response DataBase", SwingConstants.CENTER);
   title.setFont(titleFont);
   title.setForeground(titleColour);

   instructionLabel1 = new JLabel("Search Database", SwingConstants.LEFT);
   instructionLabel1.setFont(labelFont1);
   instructionLabel1.setForeground(label1Colour);

   instructionLabel2 = new JLabel("Select search fields and enter search criteria",
   SwingConstants.LEFT);
   instructionLabel2.setFont(labelFont2);
   instructionLabel2.setForeground(label1Colour);

   searchList1 = new JList(sourceFields);
   searchList1.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
   searchList1.setSelectedIndex(0);
   searchList1.setVisibleRowCount(1);
   searchList1.setFont(listFont);
   searchList1.setBackground(Color.lightGray);
   searchList1.setForeground(Color.black);
   searchList1.setSize(30, 30);
   searchList1.addListSelectionListener(new ListListener());
   scrollLst1 = new JScrollPane(searchList1);

   searchTerm1 = new JTextField(10);

   searchList2 = new JList(tessFields);
   searchList2.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
searchList2.setSelectedIndex(0);
searchList2.setVisibleRowCount(1);
searchList2.setFont(listFont);
searchList2.setBackground(Color.lightGray);
searchList2.setForeground(Color.black);
searchList2.setSize(30, 30);
searchList2.addListSelectionListener(new ListListener());
scrollLst2 = new JScrollPane(searchList2);

searchTerm2 = new JTextField(10);

searchList3 = new JList(nnppFields);
searchList3.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
searchList3.setSelectedIndex(0);
searchList3.setVisibleRowCount(1);
searchList3.setFont(listFont);
searchList3.setBackground(Color.lightGray);
searchList3.setForeground(Color.black);
searchList3.setSize(30, 30);
searchList3.addListSelectionListener(new ListListener());
scrollLst3 = new JScrollPane(searchList3);

searchTerm3 = new JTextField(10);

search = new JButton("  Start Search  ");
search.addActionListener(new ButtonAListener());
search.setBackground(Color.lightGray);
search.setForeground(label1Colour);
search.setFont(labelFont1);
search.setSize(50,50);
search.setBorder(BorderFactory.createEtchedBorder());

label1 = new JLabel("Source Fields");
label1.setFont(labelFont2);
label1.setForeground(labelColour);

label2 = new JLabel("TESS Fields");
label2.setFont(labelFont2);
label2.setForeground(labelColour);

label3 = new JLabel("NNPP Fields");
label3.setFont(labelFont2);
label3.setForeground(labelColour);

//1st tier panels - innermost
instructionPanel = new JPanel();
instructionPanel.setLayout(new GridLayout(5,1));
instructionPanel.add(Box.createRigidArea(new Dimension(2,0)));
instructionPanel.add(instructionLabel1);
instructionPanel.add(Box.createRigidArea(new Dimension(2,0)));
instructionPanel.add(instructionLabel2);
instructionPanel.add(Box.createRigidArea(new Dimension(2,0)));
```

```
instructionPanel.setBackground(Color.lightGray);

searchPanel1 = new JPanel();
searchPanel1.setLayout(new GridLayout(1,2));
searchPanel1.add(scrollLst1);
searchPanel1.add(searchTerm1);
searchPanel1.setBackground(Color.lightGray);

searchPanel2 = new JPanel();
searchPanel2.setLayout(new GridLayout(1,2));
searchPanel2.add(scrollLst2);
searchPanel2.add(searchTerm2);
searchPanel2.setBackground(Color.lightGray);

searchPanel3 = new JPanel();
searchPanel3.setLayout(new GridLayout(1,2));
searchPanel3.add(scrollLst3);
searchPanel3.add(searchTerm3);
searchPanel3.setBackground(Color.lightGray);

//2nd tier panels - intermediate
searchPanel = new JPanel();
searchPanel.setLayout(new GridLayout(9,1));
searchPanel.add(label1);
searchPanel.add(searchPanel1);
searchPanel.add(Box.createRigidArea(new Dimension(2,0)));
searchPanel.add(label2);
searchPanel.add(searchPanel2);
searchPanel.add(Box.createRigidArea(new Dimension(2,0)));
searchPanel.add(label3);
searchPanel.add(searchPanel3);
searchPanel.add(Box.createRigidArea(new Dimension(2,0)));
searchPanel.setBackground(Color.lightGray);

buttonPanel = new JPanel();
buttonPanel.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 10));
buttonPanel.add(search);
buttonPanel.setBackground(Color.lightGray);

//3rd tier panels - outermost
innerPanel = new JPanel();
innerPanel.setLayout(new BorderLayout());
innerPanel.add(instructionPanel, BorderLayout.NORTH);
innerPanel.add(searchPanel, BorderLayout.CENTER);
innerPanel.add(buttonPanel, BorderLayout.SOUTH);
innerPanel.setBackground(Color.lightGray);

outerPanel = new JPanel();
outerPanel.setLayout(new BorderLayout());
outerPanel.setPreferredSize(new Dimension(WIDTH, HEIGHT));
outerPanel.setBackground(Color.lightGray);
outerPanel.add(title, BorderLayout.NORTH);
```

```
        outerPanel.add(innerPanel, BorderLayout.CENTER);

        frame.getContentPane().add(outerPanel);
    }

    //Inner classes - event listeners
    private class ListListener implements ListSelectionListener
    {
        public void valueChanged(ListSelectionEvent evt)
        {
        //retrieve user-selected search fields
        searchFieldA = sourceFields[searchList1.getSelectedIndex()];
        searchFieldB = tessFields[searchList2.getSelectedIndex()];
        searchFieldC = nnppFields[searchList3.getSelectedIndex()];
        }
    }

    private class ButtonAListener implements ActionListener
    {
        public void actionPerformed (ActionEvent event)
        {
            //retrieve user-inputted search terms
            String searchTermA = searchTerm1.getText();
            String searchTermB = searchTerm2.getText();
            String searchTermC = searchTerm3.getText();

            //instantiate call to Search class - uses JDBC to search db
            Search mySearch = new Search(searchFieldA, searchTermA, searchFieldB, searchTermB,
            searchFieldC, searchTermC);

            //method of Search.java that returns an array of ids retrieved from db search
            results = mySearch.searchDB();

            //parse results array to result GUI and display
            DBsearchResults resultsConverter = new DBsearchResults(results);
            resultsConverter.display();
        }
    }

    //GUI display method
    public void display()
    {
        frame.pack();
        frame.setVisible(true);
    }
}
```

Appendix D2 – DBsearchGUI.java

```
//Athlee Maclear
//MSc Bioinformatics & Computational Molecular Biology 2004
//Rhodes University, Grahamstown, South Africa
//DBsearchGUI.java - displays GUI inteface for athlee_db
//

public class DBsearchGUI
{
   public static void main (String[] args)
   {
      //declare and instatiate instance of DBsearch class
      DBsearch converter = new DBsearch();
      //call display method of DBsearch class to display GUI
      converter.display();
   }
}
```

Appendix D3 – Search.java

```
//Athlee Maclear
//MSc Bioinformatics & Computational Molecular Biology 2004
//Rhodes University, Grahamstown, South Africa
//Search.java - class that searches athlee_db using user-selected search terms and fields
//            - called by the button listener class of DBsearch.java
//

//import library classes needed
import java.sql.*;
import java.util.ArrayList;

public class Search {
   //Declare variables
   private String field1, search1, field2, search2, field3, search3, query;
   private ArrayList ids;
   private String[] result;

   //Constructor
   public Search (String field1, String search1,String field2, String search2, String field3,
   String search3)
   { this.field1 = field1;
      this.search1 = search1;
      this.field2 = field2;
      this.search2 = search2;
      this.field3 = field3;
      this.search3 = search3;
   }

   //searchDB methods - returns an array of String objects
   public String[] searchDB()
   {
```

```
        //declare and initialise variables
        this.ids = ids;
        this.query = query;
        this.result = result;
        ids = new ArrayList();

        //PostgreSQL Query statement
        query = "Select DISTINCT Source.id from Source where Source." + field1 + " like '" +
        search1 + "' and TessResults." + field2 + " like '" + search2  + "' and nnppResults." +
        field3 + " like '" + search3  + "' and Source.id = TessResults.link";

        //JDBC driver url
        String url = "jdbc:postgresql:athlee_db";
        Connection con;
        PreparedStatement st;

        try {
           Class.forName("org.postgresql.Driver");

        } catch(java.lang.ClassNotFoundException e) {
           System.err.print("ClassNotFoundException: ");
           System.err.println(e.getMessage());
        }

        try {
           con = DriverManager.getConnection (url, "athlee", "sargent"); //Connect to athlee_db
           st = con.prepareStatement(query); //JDBC preparation of query statement
           ResultSet rs = st.executeQuery();//Execute query statement
           int x = 0;

           while (rs.next()) { //while reading result set
              //read record id from result set and store in ArrayList
              String id = rs.getString("ID");
              ids.add(x, id);
              x++;
           }

           //Convert ArrayList to an array of string objects
           result = (String[]) ids.toArray(new String[0]);

           //close db statement and connection
           st.close();
           con.close();

        } catch(SQLException ex) {
           System.err.print("SQLException: ");
           System.err.println(ex.getMessage());
        }
        return result; //return array of ids
     }
}
```

Appendix D4 – DBsearchResults.java

```
//Athlee Maclear
//MSc Bioinformatics & Computational Molecular Biology 2004
//Rhodes University, Grahamstown, South Africa
//DBsearchResults.java - creates a GUI inteface to display results from athlee_db
//                     - user selects record id from result set
//                     - and clicks display button to display result summary for id selected
//

//import library classes needed
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.BorderFactory;
import javax.swing.event.*;
import javax.swing.table.*;

public class DBsearchResults
{
   //declare and initialize variables
   private int WIDTH = 1000;
   private int HEIGHT = 700;

   private String selectedId = "";

   //Declare containers and components
   private JFrame frame;
   private JPanel instructionPanel, resultsListPanel, buttonPanel, tablePanel;
   private JPanel Panel1, Panel2, Panel3, Panel4, tessPanel, nnppPanel;
   private JPanel searchPanel, sourcePanel, resultPanel;
   private JPanel mainPanel, outerPanel;
   private JLabel title, Label1, Label2;
   private JLabel header1, idLabel, accLabel, orgLabel, geneNameLabel,descLabel, authLabel;
   private JLabel refLabel, indStressLabel, molTypeLabel, prodLabel, geneLabel, promLabel;
   private Jlabel caatLabel, tataLabel, polyaLabel, featureLabel, header2, header3;
   private Jlabel confidence, regStart, sigStart, pattern, strand;
   private JTextArea descText, authText, refText, indStressText, prodText, geneText, promText;
   private JTextArea caatText, tataText, polyaText, featureText, patternText;
   private JButton display, graphics;
   private JList resultIdList;
   private JScrollPane scrollLst1, scrollLst2, scrollLst3, scrollLst4, scrollLst5, scrollLst6;
   private JScrollPane scrollLst7, scrollLst8, scrollLst9, scrollLst10, scrollLst11;
   private JScrollPane scrollLst12, scrollLst13;
   private JTable table;
   private JTableHeader tableHeader;

   //Declare and initialize display variables
   private Color titleColour = Color.getHSBColor(0.5F,0.4F,0.5F);
   private Color label1Colour = Color.getHSBColor(0.6F,0.6F,0.5F);
   private Color label2Colour = Color.getHSBColor(0.8F,0.4F,0.5F);
   private Font titleFont = new Font("Serif", Font.BOLD + Font.ITALIC, 30);
   private Font labelFont2 = new Font("Serif", Font.PLAIN, 12);
```

```
private Font headerFont = new Font("Serif", Font.BOLD, 16);
private Font listFont = new Font("Serif", Font.BOLD, 12);
private Font tableFont = new Font("Serif", Font.BOLD, 10);

// Declare and initialize data arrays
private String [] results;
private String [] sourceResults;
private String[] nnppResults;
private String[] columnNames = {"Site Name",
                                "Source",
                                "Site Start",
                                "Sequence"
                                };

private String[][] tessResults;
private String [][] cells = new String [100][4];

//Constructor
public DBsearchResults(String [] resultIds)
{
   results = resultIds; //reads in array of result ids

   //Instantiate and define GUI components
   frame = new JFrame ("Search Results - Abiotic Plant Stress Response DataBase");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   frame.setPreferredSize(new Dimension(WIDTH, HEIGHT));

   title = new JLabel("Abiotic Plant Stress Response DataBase", SwingConstants.CENTER);
   title.setFont(titleFont);
   title.setForeground(titleColour);

   Label1 = new JLabel("Search Results", SwingConstants.LEFT);
   Label1.setFont(headerFont);
   Label1.setForeground(label1Colour);

   Label2 = new JLabel("Select id from result set and click display", SwingConstants.LEFT);
   Label2.setFont(listFont);
   Label2.setForeground(label2Colour);

   resultIdList = new JList(results);
   resultIdList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
   resultIdList.setSelectedIndex(0);
   resultIdList.setVisibleRowCount(1);
   resultIdList.setFont(listFont);
   resultIdList.setBackground(Color.lightGray);
   resultIdList.setForeground(label2Colour);
   resultIdList.setSize(30, 30);
   resultIdList.addListSelectionListener(new ResultsListListener());
   scrollLst1 = new JScrollPane(resultIdList);

   display = new JButton("  Display Results  ");
   display.addActionListener(new ButtonListener());
```

```
display.setBackground(Color.lightGray);
display.setForeground(label2Colour);
display.setFont(listFont);
display.setBorder(BorderFactory.createEtchedBorder());

header1 = new JLabel("Collated Information");
header1.setFont(headerFont);
header1.setForeground(label1Colour);

idLabel = new JLabel("Id: ");
idLabel.setForeground(label2Colour);
idLabel.setFont(listFont);
accLabel = new JLabel("Accession: ");
accLabel.setForeground(label2Colour);
accLabel.setFont(listFont);
orgLabel = new JLabel("Organism: ");
orgLabel.setForeground(label2Colour);
orgLabel.setFont(listFont);
geneNameLabel = new JLabel("Gene Name: ");
geneNameLabel.setForeground(label2Colour);
geneNameLabel.setFont(listFont);
descLabel = new JLabel("Description: ");
descLabel.setForeground(label2Colour);
descLabel.setFont(listFont);
authLabel = new JLabel("Author: ");
authLabel.setForeground(label2Colour);
authLabel.setFont(listFont);
refLabel = new JLabel("Reference: ");
refLabel.setForeground(label2Colour);
refLabel.setFont(listFont);
indStressLabel = new JLabel("Inducing Stress: ");
indStressLabel.setForeground(label2Colour);
indStressLabel.setFont(listFont);
molTypeLabel = new JLabel("Mol Type: ");
molTypeLabel.setForeground(label2Colour);
molTypeLabel.setFont(listFont);
prodLabel = new JLabel("Product: ");
prodLabel.setForeground(label2Colour);
prodLabel.setFont(listFont);
geneLabel = new JLabel("Gene: ");
geneLabel.setForeground(label2Colour);
geneLabel.setFont(listFont);
promLabel = new JLabel("Promoter: ");
promLabel.setForeground(label2Colour);
promLabel.setFont(listFont);
caatLabel = new JLabel("CAAT_Signal: ");
caatLabel.setForeground(label2Colour);
caatLabel.setFont(listFont);
tataLabel = new JLabel("TATA_Signal: ");
tataLabel.setForeground(label2Colour);
tataLabel.setFont(listFont);
polyaLabel = new JLabel("PolyA_Signal: ");
```

```
polyaLabel.setForeground(label2Colour);
polyaLabel.setFont(listFont);
featureLabel = new JLabel("Misc. Feature: ");
featureLabel.setForeground(label2Colour);
featureLabel.setFont(listFont);

descText = new JTextArea(5,20);
descText.setBackground(Color.lightGray);
descText.setForeground(label2Colour);
descText.setFont(labelFont2);
descText.setLineWrap(true);
scrollLst2 = new JScrollPane(descText);

authText = new JTextArea(5,20);
authText.setBackground(Color.lightGray);
authText.setForeground(label2Colour);
authText.setFont(labelFont2);
authText.setLineWrap(true);
scrollLst3 = new JScrollPane(authText);

refText = new JTextArea(5,20);
refText.setBackground(Color.lightGray);
refText.setForeground(label2Colour);
refText.setFont(labelFont2);
refText.setLineWrap(true);
scrollLst4 = new JScrollPane(refText);

indStressText = new JTextArea(5,20);
indStressText.setBackground(Color.lightGray);
indStressText.setForeground(label2Colour);
indStressText.setFont(labelFont2);
indStressText.setLineWrap(true);
scrollLst5 = new JScrollPane(indStressText);

prodText = new JTextArea(5,20);
prodText.setBackground(Color.lightGray);
prodText.setForeground(label2Colour);
prodText.setFont(labelFont2);
prodText.setLineWrap(true);
scrollLst6 = new JScrollPane(prodText);

geneText = new JTextArea(5,20);
geneText.setBackground(Color.lightGray);
geneText.setForeground(label2Colour);
geneText.setFont(labelFont2);
geneText.setLineWrap(true);
scrollLst7 = new JScrollPane(geneText);

promText = new JTextArea(5,20);
promText.setBackground(Color.lightGray);
promText.setForeground(label2Colour);
promText.setFont(labelFont2);
```

```
promText.setLineWrap(true);
scrollLst8 = new JScrollPane(promText);

caatText = new JTextArea(5,20);
caatText.setBackground(Color.lightGray);
caatText.setForeground(label2Colour);
caatText.setFont(labelFont2);
caatText.setLineWrap(true);
scrollLst9 = new JScrollPane(caatText);

tataText = new JTextArea(5,20);
tataText.setBackground(Color.lightGray);
tataText.setForeground(label2Colour);
tataText.setFont(labelFont2);
tataText.setLineWrap(true);
scrollLst10 = new JScrollPane(tataText);

polyaText = new JTextArea(5,20);
polyaText.setBackground(Color.lightGray);
polyaText.setForeground(label2Colour);
polyaText.setFont(labelFont2);
polyaText.setLineWrap(true);
scrollLst11 = new JScrollPane(polyaText);

featureText = new JTextArea(5,20);
featureText.setBackground(Color.lightGray);
featureText.setForeground(label2Colour);
featureText.setFont(labelFont2);
featureText.setLineWrap(true);
scrollLst12 = new JScrollPane(featureText);

header2 = new JLabel("TESS Results");
header2.setFont(headerFont);
header2.setForeground(label1Colour);

table = new JTable(cells,columnNames);
table.setBackground(Color.lightGray);
table.setForeground(label2Colour);
table.setFont(tableFont);
scrollLst13 = new JScrollPane(table);

header3 = new JLabel("NNPP Results");
header3.setFont(headerFont);
header3.setForeground(label1Colour);

confidence = new JLabel("Confidence Value: ");
confidence.setForeground(label2Colour);
confidence.setFont(listFont);
regStart = new JLabel("Start site of predicted core promoter: ");
regStart.setForeground(label2Colour);
regStart.setFont(listFont);
sigStart = new JLabel("Predicted Transcription Start Site: ");
```

```
sigStart.setForeground(label2Colour);
sigStart.setFont(listFont);
pattern = new JLabel("Pattern: ");
pattern.setForeground(label2Colour);
pattern.setFont(listFont);
strand = new JLabel("Strand: ");
strand.setForeground(label2Colour);
strand.setFont(listFont);

patternText = new JTextArea(2,20);
patternText.setBackground(Color.lightGray);
patternText.setForeground(label2Colour);
patternText.setFont(labelFont2);
patternText.setLineWrap(true);

tablePanel = new JPanel();
tablePanel.setLayout(new BorderLayout());
tablePanel.add(table.getTableHeader(), BorderLayout.PAGE_START);
tablePanel.add(scrollLst13, BorderLayout.CENTER);
tablePanel.setBackground(Color.lightGray);

//1st tier of panels - innermost
instructionPanel = new JPanel();
instructionPanel.setLayout(new GridLayout(2,1));
instructionPanel.add(Label1);
instructionPanel.add(Label2);
instructionPanel.setBackground(Color.lightGray);

resultsListPanel = new JPanel();
resultsListPanel.setLayout(new GridLayout(1,1));
resultsListPanel.add(scrollLst1);
resultsListPanel.setBackground(Color.lightGray);

buttonPanel = new JPanel();
buttonPanel.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 10));
buttonPanel.add(display);
buttonPanel.setBackground(Color.lightGray);

//2nd tier of panels - intermediate 1
Panel1 = new JPanel();
Panel1.setLayout(new GridLayout(3,1));
Panel1.add(instructionPanel);
Panel1.add(resultsListPanel);
Panel1.add(buttonPanel);
Panel1.setBackground(Color.lightGray);

Panel2 = new JPanel();
Panel2.setLayout(new GridLayout(5,1));
Panel2.add(header1);
Panel2.add(idLabel);
Panel2.add(accLabel);
Panel2.add(orgLabel);
```

```
Panel2.add(geneNameLabel);
Panel2.setBackground(Color.lightGray);

Panel3 = new JPanel();
Panel3.setLayout(new GridLayout(4,1));
Panel3.add(descLabel);
Panel3.add(scrollLst2);
Panel3.add(authLabel);
Panel3.add(scrollLst3);
Panel3.setBackground(Color.lightGray);

Panel4 = new JPanel();
Panel4.setLayout(new GridLayout(4,1));
Panel4.add(refLabel);
Panel4.add(scrollLst4);
Panel4.add(indStressLabel);
Panel4.add(scrollLst5);
Panel4.setBackground(Color.lightGray);

tessPanel = new JPanel();
tessPanel.setLayout(new BorderLayout());
tessPanel.add(header2, BorderLayout.NORTH);
tessPanel.add(tablePanel, BorderLayout.CENTER);
tessPanel.setBackground(Color.lightGray);

nnppPanel = new JPanel();
nnppPanel.setLayout(new GridLayout(8,1));
nnppPanel.add(header3);
nnppPanel.add(confidence);
nnppPanel.add(regStart);
nnppPanel.add(sigStart);
nnppPanel.add(pattern);
nnppPanel.add(patternText);
nnppPanel.add(strand);
nnppPanel.add(Box.createRigidArea(new Dimension(2,0)));
nnppPanel.setBackground(Color.lightGray);

//3rd tier of panels - intermediate 2
searchPanel = new JPanel();
searchPanel.setLayout(new GridLayout(4,1));
searchPanel.add(Panel1);
searchPanel.add(Panel2);
searchPanel.add(Panel3);
searchPanel.add(Panel4);
searchPanel.setBackground(Color.lightGray);

sourcePanel = new JPanel();
sourcePanel.setLayout(new GridLayout(15,1));
sourcePanel.add(molTypeLabel);
sourcePanel.add(prodLabel);
sourcePanel.add(scrollLst6);
sourcePanel.add(geneLabel);
```

```
        sourcePanel.add(scrollLst7);
        sourcePanel.add(promLabel);
        sourcePanel.add(scrollLst8);
        sourcePanel.add(caatLabel);
        sourcePanel.add(scrollLst9);
        sourcePanel.add(tataLabel);
        sourcePanel.add(scrollLst10);
        sourcePanel.add(polyaLabel);
        sourcePanel.add(scrollLst11);
        sourcePanel.add(featureLabel);
        sourcePanel.add(scrollLst12);
        sourcePanel.setBackground(Color.lightGray);

        resultPanel = new JPanel();
        resultPanel.setLayout(new GridLayout(2,1));
        resultPanel.add(tessPanel);
        resultPanel.add(nnppPanel);
        resultPanel.setBackground(Color.lightGray);

        //4th tier of panels - intermediate 3
        mainPanel = new JPanel();
        mainPanel.setLayout(new GridLayout(1,3,20,20));
        mainPanel.add(searchPanel);
        mainPanel.add(sourcePanel);
        mainPanel.add(resultPanel);
        mainPanel.setBackground(Color.lightGray);

        //5th tier - outermost
        outerPanel = new JPanel();
        outerPanel.setLayout(new BorderLayout());
        outerPanel.setPreferredSize(new Dimension(WIDTH, HEIGHT));
        outerPanel.setBackground(Color.lightGray);
        outerPanel.add(title, BorderLayout.NORTH);
        outerPanel.add(mainPanel, BorderLayout.CENTER);

        frame.getContentPane().add(outerPanel);
    }
    //GUI display method
    public void display()
    {
        frame.pack();
        frame.setVisible(true);
    }

    //Inner classes - event listeners
    private class ResultsListListener implements ListSelectionListener
    {
        public void valueChanged(ListSelectionEvent evt)
        {
            //Get user-select id from result set
            selectedId = results[resultIdList.getSelectedIndex()];
        }
```

```
    }

    private class ButtonListener implements ActionListener
    {
        public void actionPerformed (ActionEvent event)
        {
            //Declare and instatiate new instance of RetrieveSourceInfo.java,
            RetrieveTessInfo.java, and RetrieveNnppInfo.java classes
            RetrieveSourceInfo myResultSourceSet = new RetrieveSourceInfo(selectedId);
            RetrieveTessInfo myResultTessSet = new RetrieveTessInfo(selectedId);
            RetrieveNnppInfo myResultNnppSet = new RetrieveNnppInfo(selectedId);

            //Call the getResults method of each of the above classes to retrieve summary of
            information for selected id
            sourceResults = myResultSourceSet.getResults();
            tessResults = myResultTessSet.getResults();
            nnppResults = myResultNnppSet.getResults();

            //Update JTable with TESS results for selected id
            for (int x = 0; x < tessResults.length; x++){
                for (int y = 0; y < tessResults[0].length; y++){
                    table.setValueAt(tessResults[x][y], x, y);
                }
            }

            //Update GUI components with result information
            confidence.setText("Confidence Value:       " + nnppResults[0]);
            regStart.setText("Start site of predicted core promoter:      " + nnppResults[1]);
            sigStart.setText("Predicted Transcription Start Site:      " + nnppResults[2]);
            patternText.setText(nnppResults[3]);
            strand.setText("Strand:       " + nnppResults[4]);
            idLabel.setText("Id:             " + sourceResults[0]);
            accLabel.setText("Accession:     " + sourceResults[1]);
            orgLabel.setText("Organism:     " + sourceResults[2]);
            geneNameLabel.setText("Gene Name:   " + sourceResults[3]);
            descText.setText(sourceResults[4]);
            authText.setText(sourceResults[5]);
            refText.setText(sourceResults[6]);
            indStressText.setText(sourceResults[7]);
            molTypeLabel.setText("Mol Type:       " + sourceResults[8]);
            prodText.setText(sourceResults[9]);
            geneText.setText(sourceResults[10]);
            promText.setText(sourceResults[11]);
            caatText.setText(sourceResults[12]);
            tataText.setText(sourceResults[13]);
            polyaText.setText(sourceResults[14]);
            featureText.setText(sourceResults[15]);
        }
    }
}
```

Appendix D5 – RetrieveSourceInfo.java

```
//Athlee Maclear
//MSc Bioinformatics & Computational Molecular Biology 2004
//Rhodes University, Grahamstown, South Africa
//RetrieveSourceInfo.java - retrieves information for user-selected id from Source table
//

//import library classes needed
import java.sql.*;
import java.util.ArrayList;

public class RetrieveSourceInfo
{
   //declare variables
   private String id, query;
   private ArrayList retrievedResults;
   private String[] result;

   //Constructor
   public RetrieveSourceInfo (String id)
   {
      this.id = id; //reads in user-selected id from DbsearchResults GUI
   }

   //getResults method - returns array of string objects
   public String[] getResults()
   {
      //declare and instantiate variables
      retrievedResults = new ArrayList();

      //PostgreSQL query statement
      query = "Select Source.id, Source.accession, Source.organism, Source.gene_name,
      Source.description, Source.author, Source.reference, Source.inducing_stress,
      Source.mol_type, Source.product, Source.gene, Source.promoter, Source.caat_signal,
      Source.tata_signal, Source.polya_signal, Source.misc_feature from Source where Source.id
      = '" + id +"'";

      String url = "jdbc:postgresql:athlee_db"; //JDBC driver url
      Connection con;
      PreparedStatement st;

      try {
         Class.forName("org.postgresql.Driver");
      } catch(java.lang.ClassNotFoundException e) {
         System.err.print("ClassNotFoundException: ");
         System.err.println(e.getMessage());
      }
      try {
         //Connect to athlee_db and JDBC preparation and execution of query statement
         con = DriverManager.getConnection (url, "athlee", "sargent");
         st = con.prepareStatement(query);
         ResultSet rs = st.executeQuery();
```

```
        int x = 0;
        //while reading query result set
        while (rs.next()) {
           //retrieve selected information to update DbsearchResults GUI
           String id = rs.getString("ID");
           String acc = rs.getString("ACCESSION");
           String org = rs.getString("ORGANISM");
           String geneName = rs.getString("GENE_NAME");
           String desc = rs.getString("DESCRIPTION");
           String auth = rs.getString("AUTHOR");
           String ref = rs.getString("REFERENCE");
           String indStress = rs.getString("INDUCING_STRESS");
           String molType = rs.getString("MOL_TYPE");
           String prod = rs.getString("PRODUCT");
           String gene = rs.getString("GENE");
           String prom = rs.getString("PROMOTER");
           String caat = rs.getString("CAAT_SIGNAL");
           String tata = rs.getString("TATA_SIGNAL");
           String polya = rs.getString("POLYA_SIGNAL");
           String misc = rs.getString("MISC_FEATURE");
           //Add retrieved info to ArrayList
           retrievedResults.add(id);
           retrievedResults.add(acc);
           retrievedResults.add(org);
           retrievedResults.add(geneName);
           retrievedResults.add(desc);
           retrievedResults.add(auth);
           retrievedResults.add(ref);
           retrievedResults.add(indStress);
           retrievedResults.add(molType);
           retrievedResults.add(prod);
           retrievedResults.add(gene);
           retrievedResults.add(prom);
           retrievedResults.add(caat);
           retrievedResults.add(tata);
           retrievedResults.add(polya);
           retrievedResults.add(misc);
        }
        //Convert ArrayList to array of String objects
        result = (String[]) retrievedResults.toArray(new String[0]);
        //Close db statement and connection
        st.close();
        con.close();

     } catch(SQLException ex) {
        System.err.print("SQLException: ");
        System.err.println(ex.getMessage());
     }
     return result; //return array of result information
  }
}
```

Appendix D6 – RetrieveTessInfo.java

```
//Athlee Maclear
//MSc Bioinformatics & Computational Molecular Biology 2004
//Rhodes University, Grahamstown, South Africa
//RetrieveTessInfo.java - retrieves information for user-selected id from TESSresults table
//

//import library classes needed
import java.sql.*;
import java.util.ArrayList;

public class RetrieveTessInfo
{
   //declare variables
   private String id, query;
   private ArrayList retrievedResults;
   private String[] result;
   private String[][] resultTable;

   //Constructor
   public RetrieveTessInfo (String id)
   {
      this.id = id; //reads in user-selected id from DbsearchResults GUI
   }

   //getResults method - returns a 2D array of String objects
   public String[][] getResults()
   {
      //declare and instantiate variables
      retrievedResults = new ArrayList();

      //PostgreSQL query statement
      query = "Select TessResults.Site_Name, TessResults.Source, TessResults.Site_start,
      TessResults.sequence where Source.id = '" + id + "' and Source.id = TessResults.link";

      String url = "jdbc:postgresql:athlee_db"; //JDBC Driver url
      Connection con;
      PreparedStatement st;

      try {
         Class.forName("org.postgresql.Driver");

      } catch(java.lang.ClassNotFoundException e) {
         System.err.print("ClassNotFoundException: ");
         System.err.println(e.getMessage());
      }

      try {
         //Connect to athlee_db and JDBC preparation and execution of query statement
         con = DriverManager.getConnection (url, "athlee", "sargent");
         st = con.prepareStatement(query);
         ResultSet rs = st.executeQuery();
```

```
        //while reading result set
        while (rs.next()) {
           //retrieve selected information to update DbsearchResults GUI
           String site = rs.getString("SITE_NAME");
           String source = rs.getString("SOURCE");
           int start = rs.getInt("SITE_START");
           String seq = rs.getString("SEQUENCE");
           String start_pos = Integer.toString(start);
           //add retrieved info to ArrayList
           retrievedResults.add(site);
           retrievedResults.add(source);
           retrievedResults.add(start_pos);
           retrievedResults.add(seq);
        }
        //Convert ArrayList to array of String objects
        result = (String[]) retrievedResults.toArray(new String[0]);

        //convert array to 2D array to update JTable in DBsearchResults GUI
        int rows = result.length/4;
        resultTable = new String [rows][4];
        int n = 0;

        do {
           for (int x = 0; x < rows; x++){
              for (int y = 0; y < 4; y++){
                 resultTable[x][y] = result[n];
                 System.out.println(resultTable[x][y]);
                 n++;
              }
           }
        }while (n < result.length - 1);
        //Close db statement and connection
        st.close();
        con.close();

     } catch(SQLException ex) {
        System.err.print("SQLException: ");
        System.err.println(ex.getMessage());
     }
     //return 2D array of TESS results for selected id
     return resultTable;
  }
}
```

Appendix D7 – RetrieveNnppInfo.java

```
//Athlee Maclear
//MSc Bioinformatics & Computational Molecular Biology 2004
//Rhodes University, Grahamstown, South Africa
//RetrieveNnppInfo.java - retrieves information for user-selected id from NNPPresults table
//

//import library classes needed
import java.sql.*;
import java.util.ArrayList;

public class RetrieveNnppInfo
{
   //declare variables
   private String id, query;
   private ArrayList retrievedResults;
   private String[] result;

   //Constructor
   public RetrieveNnppInfo (String id)
   {
      this.id = id; //reads in user-selected id from DBsearchResults GUI
   }

   //getResults method - returns an array of String objects
   public String[] getResults()
   {
      //declare and instantiate variables
      retrievedResults = new ArrayList();
      //PostgreSQL query statement
      query = "Select NnppResults.Confidence, NnppResults.RegionStart, nnppResults.SignalStart,
      nnppResults.pattern, nnppResults.Strand where Source.id = '" + id + "' and Source.id =
      nnppResults.link";

      String url = "jdbc:postgresql:athlee_db"; //JDBC Driver Url
      Connection con;
      PreparedStatement st;

      try {
         Class.forName("org.postgresql.Driver");

      } catch(java.lang.ClassNotFoundException e) {
         System.err.print("ClassNotFoundException: ");
         System.err.println(e.getMessage());
      }

      try {
         //Connect to athlee_db and JDBC preparation and execution of query statement
         con = DriverManager.getConnection (url, "athlee", "sargent");
         st = con.prepareStatement(query);
         ResultSet rs = st.executeQuery();
```

```
        //while reading result set
        while (rs.next()) {
          //retrieve selected information to update DbsearchResults GUI
          String confid = rs.getString("CONFIDENCE");
          int start = rs.getInt("REGIONSTART");
          int sig = rs.getInt("SIGNALSTART");
          String pat = rs.getString("PATTERN");
          String strand = rs.getString("STRAND");
          String start_pos = Integer.toString(start);
          String sig_pos = Integer.toString(sig);

          //add retrieved info to ArrayList
          retrievedResults.add(confid);
          retrievedResults.add(start_pos);
          retrievedResults.add(sig_pos);
          retrievedResults.add(pat);
          retrievedResults.add(strand);
        }
        //Convert ArrayList to array of String objects
        result = (String[]) retrievedResults.toArray(new String[0]);

        //Close db statement and connection
        st.close();
        con.close();

      } catch(SQLException ex) {
        System.err.print("SQLException: ");
        System.err.println(ex.getMessage());

      } catch(NullPointerException exc) {
        System.err.print("NullPointerException: ");
        System.err.println(exc.getMessage());
        throw exc;
      }

      //return array of NMPP results for selected id
      return result;
    }
}
```

Appendix E – Plant Genome Projects (information obtained from GOLD (http://www.genomesonline.org/) and TIGR (http://www.tigr.org) webpages)

Organism	Status	Link
Arabidopsis thaliana	Complete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=3702 http://ukcrop.net/agr/
Oryza sativa spp. Indica cultivar-group	Complete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4530
Oryza sativa spp. Japonica	Complete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4530
Avena sativa (oat)	Incomplete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4498&query=/
Brassica napus	Incomplete	http://ukcrop.net/brassica.html
Brassica oleracea C (Jbo and BoB)	Incomplete	http://ukcrop.net/brassica.html
Brassica rapa A (JBr)	Incomplete	http://ukcrop.net/brassica.html
Brassica rapa pekinensis	Incomplete	http://ukcrop.net/brassica.html
Chlorarachnion reptans	Incomplete	
Coffea Arabica (coffee)	Incomplete	http://arara.lbi.ic.unicamp.br/cafe/
Eucalyptus	Incomplete	http://www.agrf.org.au/future_initiatives.html
Glycine max (Soybean)	Incomplete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=3847&query=/
Gossypium hirsutum L cv. Maxxa (cotton)	Incomplete	http://cottongenomecenter.ucdavis.edu/
Hordeum vulgare var. distichum (barley)	Incomplete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4513&query=/
Lolium perenne	Incomplete	
Lotus japonicus	Incomplete	http://www.tigr.org/tigr-scripts/tgi/T_index.cgi?species=l_japonicus
Lycopersicon esculentum cv. Heinz 1706 (tomato)	Incomplete	http://www.genome.clemson.edu/projects/other/tomato/
Lycopersicon esculentum (tomato)	Incomplete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4081&query=/
Medicago sativa (alfalfa)	Incomplete	
Medicago Truncatula	Incomplete	http://www.noble.org/medicago/index.htm
Medicago truncatula LGIII	Incomplete	http://www.tigr.org/tigr-scripts/tgi/T_index.cgi?species=medicago
Musa (banana) acuminata calcutta 4	Incomplete	http://www.inibap.org/
Oryza sativa ssp. japonicac.v. Nipponbare	Incomplete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4530
Phaseolus vulgaris (bean)	Incomplete	http://beangenes.cws.ndsu.nodak.edu/
Populus balsamifera subsp. Trichocarpa	Incomplete	http://genome.jgi-psf.org/poplar0/poplar0.home.html
Triticum aestivum (bread wheat) – EST data	Incomplete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4565&query=/
Thalassiosira pseudonana	Incomplete	http://genome.jgi-psf.org/thaps1/thaps1.home.html
Zea mays (corn)	Incomplete	http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4577&query=/

REFERENCES

Bailey-Serres, J., Rochaix, J., Wassenegger, M. and Filipowicz, W. (1999) Plants, their organelles, viruses and transgenes reveal the mechanisms and relevance of post-transcriptional processes, *EMBO J*, **18(19)**, 5153-5158.

Baumann, K., De Paolis, A., Costantino, P. and Gualberti, G. (1999) The DNA Binding Site of the Dof Protein NtBBF1 is Essential for Tissue-Specific and Auxin-Regulated Expression of the *rolB* Oncogene in Plants, *Plant Cell*, **11**, 323-333.

Bohnert, H.J., Nelson, D.E. and Jensen, R.G. (1995) Adaptations to environmental stresses, *Plant Cell*, **7**, 1099-1111.

Bray, E.A. (1993) Molecular Responses to Water Deficit, *Plant Physiol.*, **103**, 1035-1040.

Bray, E.A., Bailey-Serres, J. and Weretilnyk, E. (2000) Responses to Abiotic Stresses *in Biochemistry and Molecular Biology of Plants* (Eds. Buchanan, B., Gruissem, W. and Jones, R.) 1158-1203, American Society of Plant Physiologists, Rockville, Maryland, U.S.A.

Cercos, M., Gomez-Cadenas, A. and Ho, T.D. (1999) Hormonal regulation of a cysteine proteinase gene, EPB-1, in barley aleurone layers:*cis*- and trans-acting elements involved in the co-ordinated gene expression regulated by gibberellins and abscisic acid, *Plant J*, **19(2)**, 107-118.

Chaves, M.M., Moaroco, J.P. and Pereira, J.S. (2003) Understanding plant responses to drought – from genes to the whole plant, *Funct. Plant Biol.*, **30**, 239-264.

Chern, M., Eiben, H.G. and Bustos, M.M. (1996) The developmentally regulated bZIP factor ROM1 modulates transcription from lectin and storage protein genes in bean embryos, *Plant J*, **10(1)**, 135-148.

Croteau, R., Kutchan, T.M. and Lewis, N.G. (2000) Natural Products (Secondary metabolites) *in Biochemistry and Molecular Biology of Plants* (Eds. Buchanan, B., Gruissem, W. and Jones, R.) 1250-1318, American Society of Plant Physiologists, Rockville, Maryland, U.S.A.

Espartero, J., Pintor-Toro, J.A. and Pardo, J.M. (1994) Differential accumulation of S-adenosylmethionine synthetase transcripts in response to salt stress, *Plant Mol. Biol.*, **25(2)**, 217-227.

Ferl, R. and Paul, A. (2000) Genome Organisation and Expression *in Biochemistry and Molecular Biology of Plants* (Eds. Buchanan, B., Gruissem, W. and Jones, R.) 312-357, American Society of Plant Physiologists, Rockville, Maryland, U.S.A.

Fickett, J.W. and Hatzigeorgiou, A.G. (1997) Eukaryotic Promoter Recognition, *Genome Res.*, **7**, 861-878.

Finkelstein, R.R. and Lynch, T.J. (2000) Abscisic acid inhibition of radicle emergence but not seedling growth is suppressed by sugars, *Plant Physiol.*, **122**, 1179 – 1186.

Frank, W., Munnik, T., Kerkmann, K., Salamini, F. and Bartels, D. (2000) Water deficit triggers phospholipase D activity in the resurrection plant *Craterostigma plantagineum*, *Plant Cell*, **12**, 111-123.

Grover, A., Kapoor, A., Lakshmi, O.S., Agarwal, S., Sahi, C., Katiyar-Agarwal, S., Agarwal, M. and Dubey, H. (2001) Understanding molecular alphabets of the plant abiotic stress responses, *Curr. Sci.*, **80**, 206-216.

Gubler, F., Raventos, D., Keys, M., Watts, R., Mundy, J. and Jacobsen, J.V. (1999) Target genes and regulatory domains of the GAMYB transcriptional activator in cereal aleurone, *Plant J*, **17(1)**, 1 – 9.

Guerrero, F. and Mullet, J.E. (1986) Increased Abscisic Acid Biosynthesis during Plant Dehydration Requires Transcription, *Plant Physiol.*, **80**, 588-591.

Guerrero, F.D., Jones, J.T. and Mullet, J.E. (1990) Turgor-responsive gene transcription and RNA levels increase rapidly when pea shoots are wilted. Sequence and expression of three inducible genes, *Plant Mol. Biol.*, **15(1)**, 11-26.

Haralampidis, K., Milioni, D., Rigas, S. and Hatzopoulos, P. (2002) Combinatorial Interaction of *Cis* Elements Specifies the Expression of the *Arabidopsis AtHsp90-1* Gene, *Plant Physiol.*, **129**, 1138-1149.

Hawkesford, M.J. (2001) Introduction: The Molecular Analysis of Plant Adaptation to the Environment *in Molecular Analysis of Plant Adaptation to the Environment* (Eds. Hawkesford, M.J. and Buchner, P.) 1-15, Kluwer Academic Publishers, Dordrecht.

Hetherington, A.M. and Quatrano, R.S. (1991) Mechanisms of Action of Abscisic Acid at the cellular level, *New Phytol.*, **119**, 9-32.

Higo, K., Ugawa, Y., Iwamoto, M. and Korenaga, T. (1999) Plant *cis*-acting regulatory DNA elements (PLACE) database: 1999, *Nucleic Acids Res.*, **27(1)**, 297-300.

Ho, T.D. (1982) The mechanism of abscisic acid during seed germination *in The physiology and biochemistryof seed development, dormancy and germination* (Ed. Khann, A.A.) 299-319, Elsevier Biomedical Press.

Hooley, R. (1998) Auxin Signaling: Homing in with Targeted Genetics, *Plant Cell*, **10**, 1581-1583.

Hwang, I. and Goodman, H.M. (1995) An *Arabidopsis thaliana* root-specific kinase homolog is induced by dehydration, ABA, and NaCl, *Plant J*, **8(1)**, 37-43.

Ingram, J. and Bartels, D. (1996) The Molecular Basis of Dehydration Tolerance in Plants, *Annu. Rev. Plant Mol. Biol.*, **47**, 377-403.

Ishida, S., Fukazawa, J., Yuasa, T. and Takahashi, Y. (2004) Involvement of 14-3-3 Signaling Protein Binding in the Functional Regulation of the Transcriptional Activator REPRESSION OF SHOOT GROWTH by Gibberellins, *Plant Cell*, **16**, 2641-2651.

Iwasaki, T., Yamaguchi-shinozaki, K. and Shinozaki, K. (1995) Identification of a *cis*-acting regulatory region of a gene in *Arabidopsis thaliana* whose induction by dehydration is mediated by abscisic acid and requires protein synthesis, *Mol. Gen. Genet.*, **247**, 391-398.

Izawa, T., Foster, R., Nakajima, M., Shimamoto, K. and Chua, N. (1994) The Rice bZIP Transcriptional Activator RITA-1 is Highly Expressed during Seed Development, *Plant Cell*, **6**, 1277-1287.

Jaglo-Ottosen, K.R., Gilmour, S.J., Zarka, D.G., Schabenberger, O. and Thomashow, M.F. (1998) *Arabidopsis* CBF1 overexpression induces COR genes and enhances freezing tolerance, *Science*, **280**, 104-106.

Katti, M.V., Sakharkar, M.K., Ranjekar, P.K. and Gupta, V.S. (2000) TRES: comparative promoter analysis, *Bioinformatics*, **16(8)**, 739-740.

Kepinski, S. and Leyser, O. (2002) Ubiquitination and Auxin Signaling: A Degrading Story, *Plant Cell*, **Supplement**, S81-S95.

Kermode, A.R. (1997) Approaches to elucidate the basis of desiccation-tolerance in seeds, *Seed Sci. Res.*, **7**, 75-95.

Kim, J., Shen, Y., Han, Y., Park, J., Kirchenbauer, D., Soh, M., Nagy, F., Schafer, E. and Song, P. (2004) Phytochrome Phosphorylation Modulates Light Signaling by Influencing the Protein-Protein Interaction, *Plant Cell*, **16**, 2629-2640.

Kirch, H., Phillips, J. and Bartels, D. (2003) Dehydration-stress signal transduction *in Plant Signal Transduction* (Eds. Scheel, D. and Wasternack, C.) 140-164, Oxford University Press Inc., New York, U.S.A.

Kirscher, S., Wellmer, F., Nick, P., Rugner, A., Schafer, E. and Harter, K. (1999) Nuclear import of the Parsley bZIP transcription factor CPRF2 is regulated by Phytochrome Photoreceptors, *J Cell Biol.*, **144(2)**, 201 – 211.

Kiyosue, T., Yamaguchi-Shinozaki, K. and Shinozaki, K. (1994) Cloning of cDNAs for genes that are early-responsive to dehydration stress (ERDs) in *Arabidopsis thaliana L.*: identification of three ERDs as HSP cognate genes, *Plant Mol. Biol.*, **25(5)**, 791-798.

Knight, H. (2000) Calcium signalling during abiotic stress in plants, *Int. Rev. Cytol.*, **195**, 269-324.

Knight, H. and Knight, M.R. (2001) Abiotic stress signalling pathways: specificity and cross-talk, *Trends Plant Sci.*, **6**, 262-267.

Lee, Y., Lu, C., Casaretto, J. and Yu, S. (2003) An ABA-responsive bZIP protein, OsBZ8, mediates sugar repression of α-amylase gene expression, *Physiol. Plant.*, **119**, 78-86.

Lee, Y., Oh, H., Cheon, C., Hwang, I., Kim, Y. and Chun, J. (2001) Structure and Expression of the *Arabidopsis thaliana* Homeobox Gene *Athb-12*, *Biochem. Biophys. Res. Commun.*, **284**, 133-141.

Lewis, J. and Loftus. W. (2003) Java Software Solutions: foundations of program design 3rd edition, Pearson Education, Inc., Boston.

Liu, L., White, M.J. and MacRae, T.H. (1999) Transcription factors and their genes in higher plants – functional domains, evolution and regulation, *Eur. J. Biochem.*, **262**, 247-257.

Mansfield, T.A. and Atkinson, C.J. (1990) Stomatal Behaviour in Water Stressed Plants *in Stress responses in Plants: Adaptation and Acclimation Mechanisms*, 241-264, Wiley-Liss, Inc.

Martinez-Garcia, J.F., Moyano, E., Alcocer, M.J.C. and Martin, C. (1998) Two bZIP proteins from *Antirrhinum* flowers preferentially bind a hybrid C-box/G-box motif and help to define a new sub-family of bZIP transcription factors, *Plant J*, **13(4)**, 489-505.

Neale, A.D., Blomstedt, C.K., Bronson, P., Le, T., Guthridge, K., Evans, J., Gaff, D.F. and Hamill, J.D. (2000) The isolation of genes from the resurrection grass *Sporobolus stapfianus* which are induced during severe drought stress, *Plant Cell Environ.*, **23**, 265-277.

Oliver, M.J. and Bewley, J.D. (1997) Dessication tolerance of plant tissues: a mechanistic overview, *Hortic. Rev.*, **18**, 171-205.

Olszewski, N., Sun, T. and Gubler, F. (2000) Gibberellin Signaling: Biosynthesis, Catabolism, and Response Pathways, *Plant Cell*, **Supplement**, S61 – S80.

Pammenter, N.W. and Berjak, P. (1999) A review of recalcitrant seed physiology in relation to desiccation-tolerance mechanisms, *Seed Sci. Res.*, **9**, 13-37.

Pastori, G.M. and Foyer, C.H. (2002) Common Components, Networks, and Pathways of Cross-Tolerance to Stress. The Central Role of "Redox" and Abscisic Acid-Mediated Controls, *Plant Physiol.*, **129**, 460-468.

Pedersen, A.G., Baldi, P., Chauvin, Y. and Brunak, S. (1999) The Biology of Eukaryotic Promoter Prediction – a Review, *Comput. Chem.*, **23(3-4)**, 191-207.

Pnueli, L., Hallak-Herr, E., Rozenberg, M., Cohen, M., Goloubinoff, P., Kaplan, A. and Mittler, R. (2002) Molecular and biochemical mechanisms associated with dormancy and drought tolerance in the desert legume *Retama raetam*, *Plant J*, **31(3)**, 319-330.

Ramanjulu, S. and Bartels, D. (2002) Drought- and desiccation-induced modulation of gene expression in plants, *Plant Cell Environ.*, **25**, 141-151.

Reese, M.G. (2000) PhD Thesis: Computational prediction of gene structure and regulation in the genome of *Drosophila melanogaster*, UC Berkeley/University of Hohenheim, 24-47.
URL: http://www.fruitfly.org/~martinr/doc/PhDThesisReese2000.pdf.

Riechmann, J.L. and Ratcliffe, O.J. (2000) A genomic perspective on plant transcription factors, *Curr. Opin. Plant Biol.*, **3**, 423-434.

Rombauts, S., Florquin, K., Lescot, M., Marchal, K., Rouze, P. and Van de Peer, Y. (2003) Computational approaches to identify promoters and *cis*-regulatory elements in plant genomes, *Plant Physiol.*, **132**, 1162-1176.

Schroeder, J.I., Allen, G.J., Hugouvieux, V., Kwak, J.M. and Waner, D. (2001) Guard Cell Signal Transduction, *Annu. Rev. Plant Physiol. Plant Mol. Biol.*, **52**, 627-658.

Schug, J. and Overton, G.C. (1997) TESS: Transcription Element Search Software on the WWW, Technical Report CBIL-TR-1997-1001-v0.0, Computational Biology and Informatics Laboratory, School of Medicine, University of Pennsylvania.
URL: http://www.cbil.upenn.edu/tess

Shinozaki, K. and Yamaguchi-Shinozaki, K. (1996) Molecular responses to drought and cold stress, *Curr. Opin. Biotechnol.*, **7**, 161-167.

Shinozaki, K. and Yamaguchi-Shinozaki, K. (2000) Molecular responses to dehydration and low temperature: differences and cross-talk between two stress signalling pathways, *Curr. Opin. Biotechnol.*, **3**, 217-223.

Skriver, K. and Mundy, J. (1990) Gene Expression in Response to Abscisic Acid and Osmotic Stress, *Plant Cell*, **2**, 503-512.

Stockinger, E.J., Gilmour, S.J and Thomashow, M.F. (1997) *Arabidopsis thaliana* CBF1 encodes an AP2 domain-containing transcriptional activator that binds to the C-repeat/DRE, a *cis*-acting DNA regulatory element that stimulates transcription in response to low-temperature and water deficit, *Proc. Natl. Acad. Sci. USA*, **94**, 1035-1040.

Sun, T. (2000) Gibberellin Signal Transduction, *Curr. Opin. Plant Biol.*, **3**, 374-380.

Swamy, P.M. and Smith, B.N. (1999) Role of absisic acid in plant stress tolerance, *Curr. Sci.*, **76**, 1220-1227.

Terai, G. and Takagi, T. (2004) Predicting rules on organization of *cis*-regulatory elements, taking the order of elements into account, *Bioinformatics*, **20**, 1119-1128.

Terzaghi, W.B., Bertekap, R.L.(Jr.), and Cashmore, A.R. (1997) Intracellular Localization of GBF proteins and blue light-induced import of GBF2 fusion proteins into the nucleus of cultured *Arabidopsis* and soybean cells, *Plant J*, **11(5)**, 967-982.

Thomas, H. (1997) Drought Resistance in Plants *in Mechanisms of Environmental Stress Resistance in Plants* (Basra, A.S. and Basra, R.K.) 1-41, Harwood Academic Publishers, Amsterdam, The Netherlands.

Tisdall, J. (2001) Getting started with Perl *in Beginning Perl for Bioinformatics*, 20-30, O'Reilly Publishers.

Urao, T., Yakubov, B., Yamaguchi-Shinozaki, K., Seki, M., Hirayama, T., and Shinozaki, K. (1999) A transmembrane hybrid-type histidine kinase in Arabidopsis functions as an osmosensor, *Plant Cell*, **11**, 1743-1754.

Xing, W. and Rajashekar, C.B. (2001) Glycine betaine involvement in freezing tolerance and water stress in *Arabidopsis thaliana*, *Environ. Exp. Bot.*, **46**, 21-28.

Xiong, L., Schumaker, K.S. and Zhu, J. (2002) Cell Signalling during Cold, Drought, and Salt Stress, *Plant Cell*, Supplement, 165-183.

Yamaguchi-Shinozaki, K. and Shinozaki, K. (1994) A novel *cis*-acting element in an *Arabidopsis* gene is involved in responsiveness to drought, low-temperature, or high-salt stress, *Plant Cell*, **6**, 1061

URL References

URL 1. http://arara.lbi.ic.unicamp.br/cafe/

URL 2. http://beangenes.cws.ndsu.nodak.edu/

URL 3. http://bip.weizmann.ac.il/ws/120204_sbd/methods-finding-promoter.pdf

URL 4. http://cmgm.stanford.edu/help/manual/databases/tfd.html#what

URL 5. http://cottongenomecenter.ucdavis.edu/

URL 6. http://genome.jgi-psf.org/poplar0/poplar0.home.html

URL 7. http://genome.jgi-psf.org/thaps1/thaps1.home.html

URL 8. http://java.sun.com/products/jdbc/

URL 9. http://jdbc.postgresql.org

URL 10. http://s2k-ftp.C.S.Berkley.EDU:8000/postgres/postgres.html

URL 11. http://search.cpan.org/~mergl/pgsql_perl5-1.9.0/Pg.p

URL 12. http://ukcrop.net/agr/

URL 13. http://ukcrop.net/brassica.html

URL 14. http://www.agrf.org.au/future_initiatives.html

URL 15. http://www.bic.nus.edu.sg:8888/tres

URL 16. http://www.bioperl.com

URL 17. http://www.cbil.upenn.edu/tess/

URL 18. http://www.census.gov/ipc/www/world.html

URL 19. http://www.dna.affrc.go.jp/PLACE/

URL 20. http://www.fruitfly.org/seq_tools/promoter.html

URL 21. http://www.gene-regulation.com/cgi-bin/pub/databases/transfac/search.cgi

URL 22. http://www.genome.clemson.edu/projects/other/tomato/

URL 23. http://www.genomesonline.org/

URL 24. http://www.inibap.org/

URL 25. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=3702

URL 26. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=3847&query=/

URL 27. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4081&query=/

URL 28. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4498&query=/

URL 29. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4513&query=/

URL 30. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4530

URL 31. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4530

URL 32. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4565&query=/

URL 33. http://www.ncbi.nlm.nih.gov/mapview/map_search.cgi?taxid=4577&query=/

URL 34. http://www.noble.org/medicago/index.htm

URL 35. http://www.postgresql.org

URL 36. http://www.sanbi.ac.za/mrc/tdr2004/

URL 37. http://www.tigr.org

URL 38. http://www.tigr.org/tigr-scripts/tgi/T_index.cgi?species=l_japonicus

URL 39. http://www.tigr.org/tigr-scripts/tgi/T_index.cgi?species=medicago

URL 40. http://www.webopedia.com