THE BIOINFORMATICS AND EVOLUTION OF FUNGAL CELL WALL PROTEINS

by

JUAN E. CORONADO

A dissertation submitted to the Graduate Faculty in Biology in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York
2008

This manuscript has been read and accepted for the
Graduate Faculty in Biology in satisfaction of the
dissertation requirement for the degree of Doctor of Philosophy.

Peter N. Lipke

Date — Chair of Examining Committee

Laurel Eckhardt

Date — Executive Officer

Anne M. Dranginis

Susan L. Epstein

Weigang Qiu

Michael E. Steiper

Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract
THE BIOINFORMATICS AND EVOLUTION OF FUNGAL CELL WALL PROTEINS
by
Juan E. Coronado
Adviser: Professor Peter N. Lipke

Cell walls are hallmarks of the kingdom Fungi. Cell walls provide a myriad of functions for fungi: from osmotic protection to signal transduction. The mediators of these functions are cell wall proteins (CWPs) that are embedded in the carbohydrate network of the cell wall. Cell wall proteins have distinct characteristics: a secretory signal sequence in the N-terminus, repetitive and/or simple sequences, N and O glycosylation sites, disulfide-forming cysteine residues, and may have a membrane-anchoring glycosyl phosphatidyl inositol addition signal (GPI) at the C-terminus of the proteins.

Because CWPs often contain simple or low-complexity sequences rich in a few amino acids, sequence comparison tools- such as FASTA and BLAST- tend to align the most common residues in nonhomologous positions. These random alignments generate high similarity scores that deviate from the expected extreme value distribution and produce false statistical significance values (*e* values). We tested scoring matrices that compensate for the overrepresentation of some amino acids in any query sequence based on expected score (E) and information entropy (Q). These matrices were tested for sensitivity in finding true homologs, discrimination against nonhomologous and random sequences, conformance to the extreme value distribution, and accuracy of E values. Of the tested matrices, two matrices (called E and gtQ) gave reliable alignments in BLAST and FASTA searches, identified a consistent set of paralogs of the yeast cell wall test set

proteins, and improved the consistency of secondary structure predictions for cell wall proteins.

Iterative searches of the *S. cerevisiae* proteome with the modified matrices yielded a total of 171 known and putative cell wall proteins. The aligned segments were repeatedly subdivided and catalogued to identify 217 recurrent sequence motifs of length 8 amino acids or greater. Ninety-five percent (95%) of these motifs occur in more than one cell wall protein. The prevalence of these motifs supports the idea of fungal cell wall proteins as assemblies of recurrent building blocks, which through evolution can combinatorially produce many variants that can undergo natural selection.

When *S. cerevisiae* CWPs were compared with 17 other fungal proteomes, we found that cell wall protein profiles were consistent with fungal phylogenetic profiles, and only carbohydrate-modifying enzymes or chaperones were present in all or a majority of the fungal species. The 171 CWPs and 16 biosynthesis and biogenesis proteins were compared to the non-redundant (NR) database to look at the presence or absence of the proteins in other kingdoms of life. We found that carbohydrate-processing proteins and chaperones were the most conserved in other kingdoms, while invasins and adhesins were conserved only in less evolutionarily distant genera. The distribution of the cell wall related proteins provides evidence for a cell walled ancestor to all cells.

ACKNOWLEDGEMENTS

THE WORK IN THIS MANUSCRIPT IS AN ENDEAVOUR OF MANY VARIABLES: PARENTS, SIBLINGS, TEACHERS, COMMITTEE MEMBERS, SPOUSE, CHILDREN, ET AL. IN GENERAL, I WOULD LIKE TO THANK THEM ALL. IN PARTICULAR, I THANK MY TWO CHILDREN, **ANGELINA** AND **BERNARD**, AND MY WIFE AT THE TIME, BERLINDA, FOR THEIR SUPPORT DESPITE THE PRESSURES WITHOUT PRECEDENT OR REASON THAT WERE ENDURED. AND IT IS MY HOPE THAT THE SACRIFICES MADE WILL BE EVIDENT IN THE DEVELOPMENT OF ANGELINA AND BERNARD. THANK YOU ALL!

TABLE OF CONTENTS

ABSTRACT.....iii

ACKNOWLEDGEMENTS.....v

Chapter

1. INTRODUCTION.....1

2. METHODS AND CODE.....13

A. Matrix Modification Perl Script.....13

B. Cell wall Sequence Fragmentation.....30

C. Accessory Java Classes.....38

D. More Accessory Java Classes.....67

E. BLAST parser.....99

3. PUBLISHED MANUSCRIPTS

A. COMPOSITION-MODIFIED SCORING MATRICES IMPROVE IDENTIFICATION OF HOMOLOGS OF *S. cerevisiae* LOW-COMPLEXITY GLYCOPROTEINS.....114

1. References.....141

2. Figures and Tables.....145

B. Discovery of Recurrent Sequence Motifs in *Saccharomyces cerevisiae* Cell Wall Proteins.....154

1. References.....172

C. PHYLOGENETIC PROFILING OF CELL WALL PROTEINS.....174

1. References.....196

2. Figures and Tables.....202

4. CONCLUSION.....210

BIBLIOGRAPHY.....215

Introduction

In yeast form, *Saccharomyces cerevisiae* is a microscopic fungus that is usually between 5-10μm in length and 1-7μm in width. Its association with *Homo sapiens* can be traced probably back to the Neolithic revolution (10-12,000 years ago). *S. cerevisiae* has been used by the human race since ancient times to make beer, wine, and leavened bread. The energy in carbohydrates such as bread and fruits allows the yeast to multiply and produce alcohol and carbon dioxide as waste. Biological evidence, in particular rRNA, has been found in vases dated to 3150 B.C. In the tomb U-j of Scorpion I, one of the first kings of Egypt (Cavalieri, McGovern et al. 2003). Earlier molecular evidence has been found in Neolithic villages dating to 5000 B.C. (McGovern, Voigt et al. 1986, Ramishvili 1983).

S. cerevisiae is more than a servant to the human need for food and revelry. Its short life cycle, capacity to alternate generations between haploid and diploid organisms, manageable sporulation, and ease of culturing allowed *S. cerevisiae* to be the window into eukaryotic genetics, cell, and molecular biology. The many steps of the cell cycle were painstakingly elucidated using *S. cerevisiae* (Hartwell, Mortimer et al. 1973, Moir, Stewart et al. 1982, Reed 1980). The genetic regulatory process of mating type switching from alpha to **a** or vice versa was deduced in *S. cerevisiae* (Hicks, Herskowitz 1976). The process of anaerobic and aerobic respiration can be deductively studied through mutants in either process (Gancedo, Delgado 1984, Tzagoloff, Dieckmann 1990).

Through a sequencing project involving a number of labs , ending in 1996, the *S. cerevisiae* genome became the first eukaryote to be fully sequenced. Using genetic

analysis, biochemical techniques, gene predictions software, and comparative genomics, the sixteen chromosomes of *S. cerevisiae* have been found to have about 5800 predicted ORFs- with 4649 verified ORFs, 1145 uncharacterized ORFs (17.32%), and 815 pseudogenes (12.3%) (Goffeau, Barrell et al. 1996, Cherry, Ball et al. 1997, Spellman, Sherlock et al. 1998, Cho, Campbell et al. 1998, Allen, Salzberg 2005, Kellis, Patterson et al. 2003, Cliften, Sudarsanam et al. 2003, Hirschman, Balakrishnan et al. 2006). The yeast genome is relatively compact, where a protein-encoding gene is found every 2 kb of DNA on average. By contrast, the genome of the nematode worm *Caenorhabditis elegans* contains a potential protein-encoding gene every 6 kb , and in the human genome- some 30 kb or more of sequence must be examined in order to uncover such a gene (Caenorhabditis elegans Sequencing Consortium 1998, Lander, Linton et al. 2001, Venter, Adams et al. 2001). *S. cerevisiae* also contains ORFs in mitochondrial DNA (Foury, Roganti et al. 1998).

Cell Wall

The cell wall is a common characteristic of all fungi, but the constituents of the cell wall vary among species (Chaffin, Lopez-Ribot et al. 1998, Fontaine, Simenel et al. 2000, Grun, Hochstenbach et al. 2005, Kopecka, Fleet et al. 1995, Vaishnav, Bacon et al. 1998). The cell wall is a collage of carbohydrate and protein. *S. cerevisiae* contains an elastic, electron transparent, carbohydrate, inner cell wall layer that makes up to 66% of cell wall dry weight and is up to 200 nm thick. The carbohydrate mainly consists of glucose connected in a β1,3 orientation 1500 residues long. The β1,3 glucan is

synthesized at the cell membrane by the actions of Fsk1p and Fsk2p, with Rho1p being the regulatory subunit (Inoue, Takewaki et al. 1995, Mazur, Baginsky 1996). The inner cell wall is embellished with a small amount of β1,6 glucan (3-4% cell wall dry weight) and chitin (<2% cell wall dry weight). β1,6 glucan is also synthesized in the cell membrane by actions of unknown protein(s) (Shahinian, Bussey 2000). Chitin is made at the cell membrane by actions of chitin synthase I and II (Ford, Shaw et al. 1996, Cabib, Roh et al. 2001, Cabib, Blanco et al. 2007). The carbohydrates of the cell wall allow the cell wall to be flexible and strong through the presence of hydrogen bonds between chains. They also act as filters of molecules of certain molecular mass (De Nobel, Barnett 1991, De Nobel, Klis et al. 1990, de Nobel, Klis et al. 1990a, de Nobel, Klis et al. 1990b, De Nobel, Klis et al. 1991).

The protein component of the *S. cerevisiae* cell wall consists of an outer , electron dense, mannosylated protein layer connected to β1,3 glucan directly or indirectly through β1,6 glucan. Cell wall proteins serve a variety of functions. The cell wall requires maintenance provided by the following proteins that are produced as needed at different stages of the life or cell cycle: Crh1p, Crh2p, Crr1p,, Bgl2p, Egt2p, Pry3p, Cts1p, Dse2p, Dse4p, Scw11p, Cwp1, and Pir family (Goldman, Sullivan et al. 1995, Rodriguez-Pena, Cid et al. 2000, Molina, Gil et al. 2000, Cappellaro, Mrsa et al. 1998, Colman-Lerner, Chin et al. 2001, Lopez-Ribot, Alloush et al. 1996, Lopez-Ribot, Chaffin 1996, Kovacech, Nasmyth et al. 1996, Kapteyn, Van Den Ende et al. 1999, Kapteyn, Van Egmond et al. 1999, Terashima, Yabuki et al. 2000, Kapteyn, ter Riet et al. 2001, Jung, Levin 1999). Sag1p and Aga1p are involved in the sexual mating between alpha and **a**

cell types (Chen, Shen et al. 1995, Lipke, Kurjan 1992). Other proteins, such as Flo1p, Flo3p, Flo9p, Flo10p, and Flo11p- help cells flocculate, or bind to together, to increase chances of survival in different environments (Halme, Bumgarner et al. 2004, Cormack 2004, Lo, Dranginis 1996). Other cell wall proteins are involved in adaptations to growth conditions or metabolism: (Cappellaro, Mrsa et al. 1998, Kapteyn, Van Den Ende et al. 1999, Chu, DeRisi et al. 1998, Cohen, Sertil et al. 2001, Ram, Kapteyn et al. 1998, Kowalski, Kondo et al. 1995, Park, Jeong et al. 2005, Protchenko, Ferea et al. 2001, Sestak, Hagen et al. 2004).

Cell wall proteins have been chemically identified or predicted by comparing common sequence features with known proteins (De Groot, Hellingwerf et al. 2003, Caro, Tettelin et al. 1997). All cell wall proteins contain a signal sequence, which is a N-terminal hydrophobic sequence that the signal recognition particle binds, as the protein is being made by the ribosomes, and takes the synthetic complex (ribosome and nascent protein) to the rough endoplasmic reticulum (Egea, Stroud et al. 2005, Wild, Halic et al. 2004). Most, if not all, cell wall proteins have amino acid sequence signals that glycosylases recognize. The glycosylases add carbohydrate to asparagine, threonine, or serine residues (Gemmill, Trimble 1999, Girrbach, Strahl 2003, Yan, Lennarz 2005). The mannosylated residues can be elongated by glycolysases or trimmed by glucosidases as the protein is transported through the ER, golgi, and other cell membrane systems (Smith, Lupashin 2008, Hirschberg, Snider 1987, Herscovics 1999). Many cell wall proteins contain a glycosyl phosphatidyl inositol (GPI) addition signal, which is a C-terminal hydrophobic sequence that maintains the newly synthesized protein in the membranes of

the ER before being interchanged with the GPI glycolipid (Paulick, Bertozzi 2008, Pittet, Conzelmann 2007). Upon reaching the plasma membrane, the protein can remain at the cell membrane or be cleaved from the GPI anchor and translocated to the surrounding carbohydrate network (Pittet, Conzelmann 2007, Lesage, Bussey 2006, Klis, Boorsma et al. 2006, Frieman, Cormack 2004).

Fungal Genomics

Sequencing projects always follow the limitations of technology and genome size. Therefore, virus were the first to be completely sequenced, followed by the prokaryotes, and the next in line are the fungi (Goffeau, Barrell et al. 1996, Sanger 1981, Fleischmann, Adams et al. 1995). Due to our interests, the human genome, *Pan troglodytes* and other model organisms, *D. melanogaster*, *C. elegans*, *F. ripes*, *M. musculus* have been sequenced as well, but the number of complete genomes in each kingdom will remain proportional to genome size (Lander, Linton et al. 2001, Venter, Adams et al. 2001, Adams, Celniker et al. 2000, Aparicio, Chapman et al. 2002, Chimpanzee Sequencing and Analysis Consortium 2005, Mouse Genome Sequencing Consortium, Waterston et al. 2002). The kingdom fungi currently contain 180 genome projects, with 130 ascomycetes, 31 basidiomycetes, 7 zygomycetes, 7 microsporidia, 1 glomeromycetes, and 0 chytridiomycetes. Nineteen (19) of the ascomycetes , 3 basidiomycetes, and 1 microsporidia have been completely sequenced, respectively (Liolios, Tavernarakis et al. 2006). The sequencing projects have provided insight into ORF and regulatory sequence identification and prediction, molecular mechanisms of eukaryotic genome evolution, and

assignment of species characteristics to ORFs (Kellis, Patterson et al. 2003, Cliften, Sudarsanam et al. 2003, Bon, Casaregola et al. 2003, Harbison, Gordon et al. 2004, Scannell, Frank et al. 2007). The comparison of synteny, or gene order, in the chromosomes of *S. cerevisiae* and other ascomycetes species has provided evidence of a genome duplication event with a gradual loss of most duplicated ORFs that has created the present *S. cerevisiae* genome (Scannell, Frank et al. 2007).

Scoring Matrices

To gauge how similar two DNA or protein sequences are, a systematic measure that contains relevant biological or chemical information is required. A significant step in the creation of a method for scoring similarities was developed by Margaret Dayhoff et. al. (Dayhoff, Schwartz et al. 1978). Closely related sequences were aligned, and sequences with an average of about 1 substitution per 100 amino acids were selected to create the 1% point accepted mutation scoring matrix (PAM1) with log likelihood ratios, which take the $\log_2$ of the probability of having substitutions in related sequences divided by the probability having the same substitutions in random alignments. Scores greater than 1 represent common mutations, scores equal to zero represent neutral or random mutations, and scores less than 1 represent uncommon mutations. Subsequent scoring matrices for scoring protein alignments with greater evolutionary distance can be derived by multiplying the mutational probabilities of PAM1 to derived subsequent scoring matrices that can optimally score sequences with increasing divergence (PAM40, PAM62, PAM120 ,etc...). Due to the assumptions incorporated into the PAM matrices,

they did not detect divergent sequences as being similar. The BLOSUM (blocks substitution matrix) matrices addressed some of the shortcomings of PAM matrices by using more divergent sequences (up to 45% sequence identity) and increasing the sample of data by using the greater amount of sequence data available at the time (Henikoff, Henikoff 1992). Blocks of multiple alignments without gaps and with specific sequence identities, ranging from 90% to 45%, were used to generate ratios of observed substitutions divided by expected or random substitutions. The ratios were converted to positive or negative scores by taking the log to the base 2 of the ratio. The larger sample of data and the increased divergence among the sequences aligned produced more sensitive and selective matrices, making BLOSUM matrices the default matrices for sequence comparison (Henikoff, Henikoff 1993).

From multiple sequence alignments, it became evident that certain amino acid positions in the alignment were more conserved than other positions; by giving conserved positions higher scores, or log likelihood ratios, position specific scoring matrices (PSSMs) provided greater sensitivity and selectivity. PSSMs were able to detect relate sequences, or homologs, that had sequences mutated up to 80% (Altschul, Madden et al. 1997).

Bioinformatic Tools

Proteins are chains of amino acids that form 3D structures that enable the proteins to interact with other molecules in specific ways. The sequence of amino acids to a large extent determines the 3D shape of proteins. The sequence of amino acids can be

represented in a literal string, *e.g.* "ILIKEWRITING". The strings can be compared computationally, yielding similarity in sequence. The similarity in sequence can be used to predict function and structure of unknown proteins by comparing to the known structure and function of proteins. The sequence comparisons can also elucidate evolutionary relationships among proteins and the organisms the proteins come from. Needleman & Wunsch proposed a dynamic programming algorithm (developed in 1957 by Bellman) to globally align DNA sequences (Bellman, Kalaba 1957, Needleman, Wunsch 1970). Given a scoring criteria, the dynamic algorithm finds the maximum score for each possible alignment until you get the optimal (highest scoring) global alignment. The algorithm can be extended to protein sequence comparion using a different scoring matrix. Smith and Waterman made a slight modification of the Needleman-Wunsch algorithm to allow for local alignments instead of full-length alignments between sequences.The main difference from the Needleman-Wunsch algorithm is that negative scoring matrix cells for subalignments are set to zero, which renders the (thus positively scoring) local alignments visible. Backtracing starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment (Smith, Waterman 1981) .

Sequence comparison tools

The computational cost of computing with the dynamic programming algorithm was too great to be practical until 1990, so heuristic or approximations to the dynamic programming solutions were developed. The FASTA algorithm was one of the first

heuristics to be established. Regions with identical matches, either single sequence identities (ktup=1) or pairs of identities (ktup=2), between query and database sequences are identified by comparing the positions of each of the residues in a lookup table. The group of sequences with regions of similarity with the query are compared using the diagonal method or dot matrix plot, where regions of local or global similarities are identified. FASTA scores these regions with the PAM250 matrix; the top ten aligments are kept and trimmed at ends of the alignments to get the maximal score. The alignments with scores above a threshold are joined with penalty for gaps between regions of similarity. Finally, a Needleman-Wunsch-Sellers algorithm is used to optimally align a window of 32 amino acids centered on the highest scoring segment of the alignment. The program fits the similarity scores of the query sequence versus all database sequences to an extreme-value distribution to assess statistical significance for the optimal score between sequences (Pearson 2000).

Because of a ten fold increase in speed, the basic local alignment search tool (BLAST) became the most popular alignment tool. Words of size three (nucleotides or amino acids) are found in a sequence or database; aligned words scoring above a threshold are extended in the forward and backward direction. BLAST also introduced a sophisticated model for assessing statistical significance based on the score distribution of randomly aligned sequences with amino acid distributions similar to background, or observed, frequencies. The similarity score distribution of pairwise comparisons is an extreme-value distribution. BLAST introduced the Expected value (E value), a statistical measure of how likely a score was due to chance, which can be estimated from the

extreme-value distribution generated by random aligments (Altschul, Madden et al. 1997, Altschul, Gish et al. 1990).

Statistical significance based on the scores of randomly aligned sequences with average amino acid distribution worked well for most proteins; however, this was not the case for proteins with deviations from the average amino acid frequencies like cell wall proteins and other low-complexity proteins. The segments from the sequences with similar or same amino acids inflate scores through random alignments and produce falsely high significance (low probabilities of randomness). BLAST and other programs ignore segments with low information entropy to prevent these sequences from corrupting significance values. In removing the sequences, these programs prevent the possible alignments of similarity that these segments may possess. The aim of this thesis is to look at possible alignments of sequences with low-complexity segments .

Once pairwise comparisons are possible, other sophisticated algorithms, like position specific scoring matrix (PSSM) and hidden Markov models (HMMs), may be employed to garner more information from the sequences. From a multiple alignment of sequences gotten from pairwise alignments, a position specific scoring matrix (PSSM) with the size of the 20 amino acids by length of the multiple alignmentis constricted. Scores are generated for each amino acid for every position in the alignment; the scores are based on the frequency with which each amino acid occurs compared to the expected random occurrences for the particular amino acid (Altschul, Madden et al. 1997).

Hidden Markov models (HMMs) are similar to PSSMs, but have more flexibility; HMMs are probabilistic models of multiple alignments. Instead of having scores for

amino acid positions in a particular position in a sequence, HMMs have probabilities for matching an amino acid, observing a deletion or insertion. In addition, HMMs have other probabilities to make a more robust representation of an alignment: start and begin, n and c-terminal unaligned. Two popular HMMs comparison tools are Hmmer (Eddy 1998) and SAM (Hughey, Krogh 1996). These software packages use multiple alignments to generate models that can be used to search for sequences that fit the proposed models; HMMs have been shown to be slightly superior to PSSM methods (Madera, Gough 2002).

Support vector machines are machine learning algorithms that find an optimal plane or divider between data of any given nature; the data points usually stored in a vector of size *n* are analyzed in *n*-dimensional space, and an algorithm based on euclidean distance finds the optimal division in the *n*-dimensional space between the data points in the vector (Vapnik 1998, Yang 2004). Given scores in a vector of related and unrelated sequences, the algorithm finds the optimal division between true relationships from false relationships; the relationships can be sequence similarities, microarray data, protein structure, or combinations of different data types (Yang, Johnson 2005, Liao, Noble 2003, Wang, Zhu et al. 2008, Rapaport, Barillot et al. 2008, Klammer, Reynolds et al. 2008).

AIMS OF THESIS

The aims of this thesis are to find the high- and low-complexity sequence similarities among the cell wall proteins of *S. cerevisiae* and other fungi. The first

problem encountered is the identification and/or gathering of the set of possible cell wall proteins in *S. cerevisiae*. The second task is to minimize the low-complexity similarity problem, which is the random alignments of high frequency amino acids, such as serine and threonine, with each other. Thirdly, the sequence similarities in cell wall proteins are local, often confined to relatively short amino acid segments within two sequences, as opposed to N-terminal to C-terminal global sequence similarities between two protein sequences. Finally, it addresses the evolutionary question of how conserved *S. cerevisiae* cell wall proteins are in other fungi, and how the pattern of cell wall protein conservation applies to fungal and cell wall evolution.

Methods and Code

The matrix modifications and sequence comparison were accomplished with a PERL script. The perl program accepts individual or multiple sequences in FASTA format as input; the program will calculate the amino acid frequency of the sequence, then it will divide each amino acid frequency by the corresponding standard frequency to yield a ratio for each amino acid. The ratio is used to adjust the scores of the BLOSUM62 matrix to reflect the deviation of amino frequencies from the average composition. The program then runs a sequence comparison using BLAST, PSI-BLAST, FASTA, or SSEARCH- depending on which method is specified in the command line. BLAST and FASTA are the basic pairwise sequence comparisons tools. SSEARCH uses the Smith-Waterman alignment to produce the optimal scoring alignment. PSI-BLAST is the same as BLAST with composition-based statistics for one round of comparisons (-j 1). If the perl script is executed without command line arguments, it writes out to the screen a summary of the possible command line arguments.

Pseudo code

```
while more sequences in fasta file
do
     array = split( sequence)
     amino_acid_% =
     calculateAminoAcidPercentage(amino_acid_array)
     amino_acid_ratios=
     calculateAminoAcidRatios(amino_acid_%,
     background_amino_acid_%)
     modifiedScoringMatrix =
     multiplyMatrixScoresByAminoAcidRatios(BLOSUM62,
     amino_acid_ratios)
     lambdaModifiedMatrix=
          karlinAltschul(modifiedScoringMatrix)
     reScaledScoringMatrix =
     rescaleScoringMatrixToBlastStatistics( LambdaBLOSUM62,
```

```
     lambdaModifiedMatrix, modifiedScoringMatrix)
     runSequenceComparison(database, sequence,
     reScaledScoringMatrix)
done
```

Actual Code:

```
#!/bin/perl -w

# object-oriented code needed by MODPSC.pl script to run
properly: BioPerl and perl data structures
use POSIX;
use strict; use Data::Dumper; use Bio::SeqIO;
use Bio::Tools::Run::StandAloneBlast; use Bio::SearchIO;
use Bio::DB::Fasta; use Bio::Seq;
use Pod::Usage;
use Getopt::Std;

# options array to hold commandline arguments
my %opts;

#calling sub routine PargseArgs to get commandline arguments
&ParseArgs;

# sequence ID and description variables
my $id; my $desc;

# assign commandline arguments to variables or default
values to be used with BLAST or FASTA
# executable
my $seq_file =  $opts{'i'};
my $program = $opts{'P'} || "BLASTP";#print $program;
my $database = $opts{'d'} ||
"/export/home/juan/FISHER/blastDB/yeast.aa";
my $matrix_file = $opts{'M'} ||
"/export/home/juan/FISHER/FASTA/BLOSUM62.ori";
my $modmatrix = $opts{'k'} || "BLOSUM62";

# assign the SEG filtering variable to false if the matrix
modifications are specified on  the
# commandline, else assign true value to the $filter
variable
my $filter;
if ($program =~ /BLAST(PGP$|E$|Q$|H$|gtE$|gtQ$|hcE$|hcQ$)|
FASTA(E$|Q$|H$|hcE$|hcQ$|gtE$|gtQ$|gtE32$|gtQ32$)/ ){
  $filter = "F" ;
```

```
}elsif ($program =~ /BLASTP$|FASTAP$/){$filter = $opts{'F'}
|| "T"}

#assign commandline arguments to variables or default values
my $gap = $opts{'G'} || 11;
my $extension = $opts{'E'} || 1;
my $line_desc = $opts{'v'} || 500;
my $num_align = $opts{'b'} || 500;
my $evalue = $opts{'e'} || 10;

# assign variable to be used with the FASTA executable
my $gapFasta = $gap;
my $extFasta = $extension;

# reference to matrices hashes: originial and modified
matrices, respectively
my($blosum, %modblosum );
# copy the original matrix file to the matrix being
modified for each sequence
# prevents doing a search with a modified BLOSUM62 matrix
if BLAST is executed directly, not
# through this program
my @args2 = ("cp", "$matrix_file",
"/export/home/juan/FISHER/FASTA/$modmatrix");
system (@args2);

#  read in to the hash the original blosum62
$blosum = &blosum($matrix_file);

# read query FASTA sequence
my $seq_in = Bio::SeqIO->new(-file => "$seq_file" , '-
format' => 'Fasta');

# robinson & robinson amino acid frequencies gotten from
Robinson & Robinson (1990)
my @Robinson = (".07805",".05129",".04487",".05363",".
01924",".04264",".06294",".07377",".02199",".05142",".
09019",".05744", ".02243",".03856",".05203",".07120",".
05841",".01330",".03216",".06441",".002");

# amino acid symbol array
my @aa2 =
("A","R","N","D","C","Q","E","G","H","I","L","K","M","F","P
","S","T","W","Y","V", "X");
```

```
# populate the Robinson2 hash with amino acid symbol as key
and frequency as value
my %Robinson2;
for (my $i = 0;$i <= 20; $i++){
  $Robinson2 {$aa2[$i]} = $Robinson[$i];
}

# go through each sequence in the fasta file
while (my $seq = $seq_in->next_seq()) {
print "Processing sequence ", $seq->display_id(),"\n";

#  hash to maintain count of amino acids (key= amino acid
symbol; value=count)
my %count;

# hash to store the log likelihood of amino acid pair
frequencies , which is equal to the log( query-query #
amino  frequencies  divided by robinson-robinson
frequencies) (+ if  query-query is larger than
# robinson-robinson, 0 if equal, and - otherwise )
my %matrix;

my @PR;

# hash to hold the frequencies of all possible amino acid
pairs (20X20)
# proteomic amino acid pair frequencies
my %Rob_pairs;
# query amino acid pair frequencies,   , query and
proteomic amino acid pair frequencies
my (%aa_pairs, %aa_pairs1, %aa_pairs2);
                                                    d
# amino acid length of FASTA sequence
my $sum;
# low and high scores of the matrix
]'lol n kb
my $high;
my $low;

# constant to multiply matrix to re-calibrate the lambda
value to that of the original matrix
my $constant = 1;

#replace | and ( with _ in the sequence id
$id = $seq->display_id();
```

```
$id =~ s/[\(\)\:\/\|\s]/_/g;

# name of output file of sequence comparison
my $out = $opts{'O'} || "$id.$program.$gap.
$extension.out";#print $out, "\n";

# assign sequence description
$desc = $seq->desc();

# clean up the description
  if ($desc){
  $desc =~ s/[;\:\/\|\s]/_/g;
  }

# if the commandline stipulates any of the following
matrix modification
if ($program =~ /BLAST(P$|PGP$|E$|Q$|H$|gtE$|gtQ$|hcE$|hcQ
$)|FASTA(E$|Q$|H$|hcE$|hcQ$|gtE$|gtQ$|gtE32$|gtQ32$)|
SSEARCH(E$|Q$|H$|hcE$|hcQ$|gtE$|gtQ$|gtE32$|gtQ32$)/ ){

# get squence string, split the sequence , initialize the
count hash
my $str = $seq->seq();
$str =~ s/\*//g;$str =~ tr/a-z/A-Z/;

#split sequence into array
my @aa = split('', $str);
  foreach my $aa (@aa2){
    $count{$aa} = 0;
  }

#count the frequency of each amino acid, skipping X
foreach my $res (@aa) {
    unless ($res =~ /X/){
      $sum++;
      $count{$res}++
    }
  }


# calculate the amino acid pair frequencies using the amino
acid composition of the query sequence
# ($aa_pairs), of query sequence and the robinson
background, or proteomic, frequencies ( $aa_pairs2), # and
of the proteomic frequencies( $Rob_pairs)
```

```
# e.g., amino acid pair frequency of AA = amino frequency
of A * amino acid frequency of A
#e.g., amino acid pair frequency of AA = amino frequency of
A * robinson frequency of A
#e.g., amino acid pair frequency of AA = robinson frequency
of A * robinson frequency  of A
  foreach my $res_1 (keys %count) {
    foreach my $res_2 (keys %count) {
        $aa_pairs{$res_1. $res_2} = ($count{$res_1} *
$count{$res_2}) / ($sum * $sum);
        $aa_pairs2{$res_1. $res_2} = ($count{$res_1} *
$Robinson2{$res_2}) / ($sum);
        $Rob_pairs{$res_1. $res_2} = $Robinson2{$res_1} *
$Robinson2{$res_2};
       }
  }

# calculate the log likelihood of the query-query amino
frequencies / robinson-robinson frequencies
# To be used to determine whether or not to modify the
score in the BLOSUM62 matrix
foreach my $pair (keys %aa_pairs2) {
  unless ($aa_pairs2{$pair} == 0){
       $matrix{$pair} = sprintf "%.0f", log
( $aa_pairs2{$pair} / $Rob_pairs{$pair}  )  / log(2);
     }

# modify the matrix according to the Expected Score method
(Coronado et. al., 2006)
# the E modification
# if amino acid pair exists, modify accordingly; if XZB
amino acids appear leave the same
  if ($program =~ /BLASTE$|FASTAE$|SSEARCHE$/ ){
    foreach my $pair (keys %$blosum) {
      if ( $aa_pairs2{$pair}  )  {
                $modblosum{$pair} =   sprintf "%.0f",
(  ( $Rob_pairs{$pair} *$ {$blosum}{$pair} ) /
$aa_pairs2{$pair} ) ;

      }else {
        $modblosum{$pair} =  ( $ {$blosum}{$pair} );
      }
  }
}
```

```
# modify the matrix  to maintain the target frequencies
equal to the original matrix method
# (Coronado et. al., 2006): the Q modification
if ($program =~ /BLASTQ$|FASTAQ$|SSEARCHQ$/ ){
    foreach my $pair (keys %$blosum) {
      if ( $aa_pairs2{$pair} )  {
                $modblosum{$pair} =   sprintf "%.0f",(log
($Rob_pairs{$pair} / $aa_pairs2{$pair})/.318 + $ {$blosum}
{$pair});
      } else {
        $modblosum{$pair} =  ( $ {$blosum}{$pair} );
      }
 }

}
# modify the matrix  to maintain the Expected score  equal
to the original matrix for instances where
# the query-query amino acid frequencies pairs are greater
than the robinson-robinson pairs
# (Coronado et. al., 2006): the gtE modification
if ($program =~ /BLASTgtE$|FASTAgtE$|SSEARCHgtE$/ ){
    foreach my $pair (keys %$blosum) {
      if ( $matrix{$pair} and $matrix{$pair} > 0  )  {
           $modblosum{$pair} = sprintf "%.0f",
(  ( $Rob_pairs{$pair} *$ {$blosum}{$pair} ) /
$aa_pairs2{$pair} ) ;
        }else {
        $modblosum{$pair} =  ( $ {$blosum}{$pair} );
      }
  }

}


# To attempt to increase senstivity, scores were increased
32 fold for the gtE method as well as gap and #
extentension costs, this method could be used with FASTA
due to FASTA's flexible scoring matrix
# and gap costs parameters
if ($program =~ /FASTAgtE32$|SSEARCHgtE32$/ ){
           $gapFasta = $gap * 32;
           $extFasta = $extension * 32;
    foreach my $pair (keys %$blosum) {
      if ( $matrix{$pair} and $matrix{$pair} > 0  )  {
```

```
        $modblosum{$pair} = sprintf "%.0f",
(  ( $Rob_pairs{$pair} *($ {$blosum}{$pair} *32) ) /
$aa_pairs2{$pair} ) ;
             }else {
        $modblosum{$pair} =  ( $ {$blosum}{$pair}  * 32);
      }
    }

}



# same modification as Q above, but the modifications are
relegated to amino acid pairs with greater
# than background frequencies
  if ($program =~ /BLASTgtQ$|FASTAgtQ$|SSEARCHgtQ$/ ){
    foreach my $pair (keys %$blosum) {
      if ( $matrix{$pair} and $matrix{$pair} > 0 )  {
          $modblosum{$pair} =   sprintf "%.0f",(log
($Rob_pairs{$pair} / $aa_pairs2{$pair})/.318 + $ {$blosum}
{$pair});}
      }else {
        $modblosum{$pair} =  ( $ {$blosum}{$pair} );
      }
  }

 }


# To attempt to increase senstivity, scores were increased
32 fold for the gtQ method as well as gap and #
extentension costs, this method could be used with FASTA
due to FASTA's flexible scoring matrix
# and gap costs parameters
  if ($program =~ /FASTAgtQ32$|SSEARCHgtQ32$/ ){
          $gapFasta = $gap * 32;
           $extFasta = $extension * 32;
    foreach my $pair (keys %$blosum) {
      if ( $matrix{$pair}  and  $matrix{$pair}  > 0 )  {
                    $modblosum{$pair} =   sprintf "%.0f",
(((log ($Rob_pairs{$pair} / $aa_pairs2{$pair}))/.318 + $
{$blosum}{$pair}) * 32);
      }else {
        $modblosum{$pair} =  ( $ {$blosum}{$pair} * 32);
      }
```

```
    }
}

 if ($program =~ /BLAST(E$|Q$|H$|gtE$|gtQ$)|FASTA(E$|Q$|gtE
$|gtQ$)|SSEARCH(E$|Q$|gtE$|gtQ$)/ ){
 # sort the modified matrix to get the highest and lowest
score
    my @hilopair= sort { $modblosum{$a} <=>
$modblosum{$b} } keys %modblosum;
    $high = $modblosum{$hilopair[$#hilopair]};
    $low =  $modblosum{$hilopair[0]};

# make initialize  to hold the probability of getting a
particular score from the lowest to the highest
# score
    for (my $i=0;$i<= ($high-$low);$i++){
      $PR[$i] = 0;
    }

# fill in the array with the addition of all query-robinson
amino acid pair frequencies belong to the same # score,
which is the probability of getting a given score
    foreach my $pair (keys %aa_pairs2) {
      $PR[ $modblosum{$pair} + abs($low)] +=
$aa_pairs2{$pair};
    }

# run the GETLAMBDA sub routine to find the lambda for the
matrix; perl code is a translation of the # C code gotten
from Altschul @ NCBI
 my $lambda2 = &GETLAMBDA($high,$low,@PR);

 print "\nLambda value of Modified Matrix= $lambda2" , "\n";
# the lambda of original BLOSUM62 matrix
 my $lambda1 = .3176;

#  calculate constant to multiply of scores of modified
matrix to get the lambda back to the original
# value of .3176  or as close as possible to the value.
$constant = $lambda2 / $lambda1 ;
while ((my $pairs, my $score) = each (%modblosum))     {
 $modblosum{$pairs} =   sprintf "%.0f",($constant * $score);
    }

print "To get BLOSUM62 Lambda value (.3176), we multiply
```

```
matrix by the constant $constant", "\n";

# write the modified matrix to the BLOSUM62 file
 &WriteModBlosum(\%modblosum);

 print "BLOSUM62 has been overwritten\n";

# print the matrix to screen
 &WriteModBlosum2(\%modblosum);

  }

}

# run standard BLASTPGP (PSI-BLAST) without any matrix
modifications
if ($program =~ /BLAST(PGP$)/){
      my $out2 = Bio::SeqIO->new('-file' => ">$id.
1.fas" ,'-format' => 'Fasta');
     $out2->write_seq($seq);
     my @args = ("/usr/local/bin/blastpgp",  "-i", "$id.
1.fas","-d","$database","-M","$modmatrix","-F","$filter","-
G","$gap"
,"-E","$extension","-e","$evalue","-v","$line_desc","-
b","$num_align","-j","1" ); #,"-o","$id.out"
    print (@args,"\n");
    system (@args);
}

# run BLAST program without or with matrix modification
if ($program =~ /BLAST(P$|E$|Q$|H$|gtE$|gtQ$)/){
    my $out2 = Bio::SeqIO->new('-file' => ">$id.1.fas" ,'-
format' => 'Fasta');
   $out2->write_seq($seq);
     my @args = ("/usr/local/bin/blastall", "-p","blastp",
"-i", "$id.1.fas","-d","$database",
           "-M","$modmatrix","-F","$filter","-G","$gap","-
E","$extension","-e","$evalue","-v","$line_des   c","-
b","$num_align" );
     print (@args,"\n");
     system (@args);
}

# write  and print out matrix file to file or screen
 if ($program =~ /FASTAgtQ32$|FASTAgtE32$|SSEARCHgtQ32$|
```

```
SSEARCHgtE32$/ ){
           print "BLOSUM62 has been overwritten\n";
     &WriteModBlosum(\%modblosum);
          &WriteModBlosum2(\%modblosum);
 }

# run FASTA executable with or without modified matrices
 if ($program =~ /FASTA(P$|E$|Q$|gtE$|gtQ$|gtE32$|gtQ32$)/){
   $out =~ s/out$/fout/ ;
    my $out2 = Bio::SeqIO->new('-file' => ">$id.1.fas" ,'-
format' => 'Fasta');
   $out2->write_seq($seq);
    my @args;
   if ($filter =~/F/){
     @args = ("/usr/local/bin/fasta33","-
s","/export/home/juan/FISHER/FASTA/$modmatrix",
        "-q", "-E", "$evalue","-H","-f","$gapFasta","-
g","$extFasta","-d","$num_align",
      "$id.1.fas", "$database","-m","10","");#"-O","$out","-
m","10",
   }else{
     @args = ("/usr/local/bin/fasta33","-
s","/export/home/juan/FISHER/FASTA/$modmatrix",
        "-q", "-E", "$evalue","-H","-f","$gapFasta","-
g","$extFasta","-d","$num_align","-S",
     "$id.1.fas","$database","-m","10","");
   }
        print (@args,"\n");
   system (@args);
 }

#  run smith-waterman alignments, guaranteed to get highest
scoring local alignment, using the
# SSEARCH program from the FASTA package from William
Pearson
if ($program =~ /SSEARCH(P$|E$|Q$|H$|gtE$|gtQ$|hcE$|hcQ$|
gtE32$|gtQ32$)/){
   $out =~ s/out$/fout/ ;
   my $out2 = Bio::SeqIO->new('-file' => ">$id.1.fas" ,'-
format' => 'Fasta');
   $out2->write_seq($seq);
   my @args;
   if ($filter =~/F/){
     @args = ("/usr/local/bin/ssearch33", "-
s","/export/home/juan/FISHER/FASTA/$modmatrix",
```

```
        "-q", "-E","$evalue","-H","-f","$gapFasta","-
g","$extFasta","-d","$num_align","$id.1.fas",
     "$database","-m","10","");#"-O","$out",
   }else{
     @args = ("/usr/local/bin/ssearch33","-
s","/export/home/juan/FISHER/FASTA/$modmatrix","-q", "-E",
"$evalue","-H","-f","$gapFasta","-g","$extFasta","-
d","$num_align","-m","10","-S","$id.1.fas",
"$database","");

}
 print (@args,"\n");
  system (@args);

 }

}

my @rm = ("rm","*.1.fas");

system(@rm);

# copy original matrix file to BLOSUM62 file to make sure
you always start with the original
# BLOSUM 62 and some modified version of BLOSUM62
my @args3 = ("cp", "$matrix_file",
"/export/home/juan/FISHER/FASTA/$modmatrix");

system (@args3);

exit;



############################################################
#########################
Subroutine to write modified BLOSUM62 to the specified file
############################################################
#########################

sub WriteModBlosum {
    open (MODMAT, ">/export/home/juan/FISHER/FASTA/
$modmatrix") or die "Couldn't open for writing: $!\n";
    my @aa = ("A","R","N","D","C","Q","E","G","H",
"I","L","K","M","F","P","S","T","W","Y","V","X");
```

```
    print MODMAT "    ", join("  ",@aa), "\n";
    for (my $i=0; $i <=  $#aa; $i++) {
        print MODMAT  $aa[$i], "  ";
        for (my $j=0; $j <= $#aa; $j++) {
            printf MODMAT "%2d ",  $modblosum{$aa[$i].
$aa[$j]};
        }
        print MODMAT "\n";
    }
    print MODMAT "\n";
}

##############################################################
########################
Subroutine to print modified BLOSUM62 to the screen or
output log file
##############################################################
########################
sub WriteModBlosum2 {
   my
@aa=("A","R","N","D","C","Q","E","G","H","I","L","K","M","F
","P","S","T",
         "W","Y","V","X");
   print "    ", join("  ",@aa), "\n";
    for (my $i=0; $i <=  $#aa; $i++) {
        print   $aa[$i], "  ";
        for (my $j=0; $j <= $#aa; $j++) {
            printf  "%2d ",  $modblosum{$aa[$i].$aa[$j]};
       }
        print  "\n";
    }
    print  "\n";
}

##############################################################
########################
subroutine to read BLOSUM62 scores into a hash
##############################################################
########################
sub blosum {
    my $file = shift;
    my %blosum62;
    open(IN, "$file");
    my @lines;
    while(<IN>) {
```

```
        next if /^\#/;
        chomp;
        push @lines, $_;
    }
    my $firstline = shift @lines;
    my @aa = split (/\s+/, $firstline);
    shift @aa;
    foreach my $rest (@lines) {
        my @scores = split (/\s+/, $rest);
        my $aa = shift @scores;
        foreach (my $i = 0; $i <= $#scores; $i++) {
            $blosum62{$aa . $aa[$i] } = $scores[$i];
        }
    }
    return \%blosum62;
}


#############################################################
#########################
subroutine to calculate lambda of modified  BLOSUM62
matrix; perl code is a translation of C code gotten from
Stephen Altschul from the NCBI. The function uses the
Newton method to solve for lambda in the equation
```

$$\sum P_i * P_j e^{\lambda S_{ij}} = 1$$

```
#############################################################
#########################
sub GETLAMBDA{
# /* Calculate the parameter lambda ; assign parameters
passed to function*/
my $high = shift;
my $low = shift;
#my $ref = shift;
my @PR = @_;
my $range = $high - $low;
# /* up is upper bound on lambda */

my $up=0.5;
my $sum = 0.0;

do{

  $up *= 2.0;

  my $beta =exp($up);
```

```
  my $ftemp=exp($up*($low-1));

  for (my $i=0; $i<=$range; ++$i){

    $sum+= $PR[$i] * ($ftemp *= $beta);

  }
}while ($sum<1.0);

# /* avoid overflow from very large lambda*S */
while ($sum > 2.0){
   $up /= 2.0;
  my $beta=exp($up);
  my $ftemp=exp($up * ($low-1));
  $sum = 0.0;
  for (my $i=0; $i<=$range; ++$i){
    $sum+= $PR[$i] * ($ftemp *= $beta);
      }
}

  $up *= 2.0;

#*/     /* we moved past, now back up */

#  /*   for (lambda=j=0;j<25;++j) { */
my $lambda = 0.0;
my  $nit = 0;
# number of iterations
my $MAXIT = 25;

# precision estimate
my $TINY = 1e-6;
while ( $nit++ < $MAXIT ) {
  my $new = ($lambda+$up)/2.0;
  my $beta = exp($new);
  my $ftemp = exp($new*($low-1));
  $sum = 0.0;
  # /* multiply by exp(new) for each score */
  for (my $i=0;$i<=$range;++$i){
    $sum+= $PR[$i] * ($ftemp *= $beta);
}


if ($sum > 1.0 + $TINY){
```

```
    $up=$new;
} else {
    if ( abs($lambda - $new) < $TINY ){
      goto done;
    }
      $lambda = $new;
  }
 }

  if ($lambda <= 1e-10) {
    $lambda = -1.0;
    return 0;
  }
done:
 return ($lambda);

}

##############################################################
########################
subroutine to parse commandline arguments
##############################################################
########################

sub ParseArgs {
    getopt('hmPidMkFGEvbeO:', \%opts);
 pod2usage(VERBOSE => 3) if ($opts{m}) ;
  pod2usage(VERBOSE => 0) if ($opts{h}) ;
 $opts{i} or  pod2usage(VERBOSE => 0);
}

#=cut;
################## POD Documentation ##################
__END__
=head1 NAME
MODPSC (Modified Pair-wise Sequence Comparison)
=head1 SYNOPSIS
MODPSC.pl -id... [query file] [database file]
  Options:
    -help       brief help message
    -man        full documentation
    -P          BLAST[X] or  FASTA[X]; X = P(standard),E
(expected score),or Q (target frequency)
modification(e.g.,BLASTE).
 default = BLASTP
```

```
    -i         input file name[FASTA format]. required
    -O          output file name
    -d          database file. default = nr
    -M          location of original scoring matrix file
(e.g., /john/matrices/BLOSUM62.ori)
    -k          name of matrix to be used in the
comparions(e.g., BLOSUM62, PAM70): this matrix will be
modified sequence to
 sequence. default = BLOSUM62.
    -F          Seg Filter [F (off),T(on)]. default = T
    -G          intial gap cost. default = 11. For BLAST,
gap costs are specific to a given matrix. For FASTA, gaps
cost can
 be determined by user with less restriction.
    -E          gap extension cost. default = 1
    -v          number of alignments to show in output.
default = 250
    -b          number of one line descriptions to show in
output. default = 250
    -e          e-value cutoff. default = 10
=head1 OPTIONS
=over 4
=item B<-help>
Print  brief help message and exits.
=item B<-man>
Prints the manual page and exits.
=back
=head1 DESCRIPTION
 Using BIOPERL, read in sequence file in fasta format ;
calculate amino
 acid frequency; generate hash with amino acid pair
frequencies for both
 query and Robinson & Robinson frequencies; modify BLOSUM62
to maintain
 Expected Score constant in context of query AA%; calculate
Lambda of
 modify matrix; rescale scores to generate matrix w/
BLOSUM62
 Lambda value; run pair-wise comparison with the  modify
matrix.
=head1 FILES
=over 4
=back
=head1 REQUIRES
```

```
Perl 5.004
(http://www.perl.com/pub/a/language/info/software.html),
bioperl-1.2.3 (http://bioperl.org/Core/Latest)/, standal
one BLAST (ftp://ftp.ncbi.nlm.nih.gov/blast/executables/)
or FASTA (ftp.virginia.edu). User needs to copy the
BLOSUM62 matri
x file to BLOSUM62.ori file, which will not be changed from
comparison to comparison.
=head1 SEE ALSO
perl(1)
=head1 AUTHOR
Juan Coronado (graduate student)
coronado@genectr.hunter.cuny.edu
Hunter College
695 Park Avenue
New York, New York 10021
=cut
###################### End ############################
```

Decompose.java

The Decompose.java class contains methods that execute the iterative motif identification algorithm using the BLAST sequence comparisons.

```
package hunter.cuny.edu;

# bring in required java code: biojava and java objects
import java.io.*;
import javax.sql.*;
import java.sql.*;
import java.util.*;
import java.net.*;
import org.biojava.bio.symbol.AlphabetManager;
import org.biojava.bio.seq.ProteinTools;
import org.biojava.bio.*;
import org.biojava.bio.program.sax.*;
import org.biojava.bio.program.ssbind.*;
import org.biojava.bio.search.*;
import org.biojava.bio.seq.db.*;
import org.xml.sax.*;
import org.biojava.bio.seq.io.SeqIOTools;
import org.biojava.bio.symbol.*;
import org.biojava.utils.ChangeVetoException;
import org.biojava.bio.seq.Sequence;
```

```
import org.biojava.bio.seq.SequenceIterator;
/**
*     This class will decompose a query sequence into
domains or fragments based on BLAST HSPs  * alignments: You
need to have sequence comparison  results within a
database, which you can
*    use to get ResultSets for input into the setSegmentMap
method present in this java class; you
*    would then save each segmentMap in a Vector, write the
vector to a file; the Vector will be used * as input to the
compareFrags method  to make sequence comparisons of the
sequences
*    fragment; the fragment compariosn results can be used
as input to the setSegmentMap to
*    decompose the fragments even further until no new
fragments are able to be made because of
*    fragment size limitations or no new local similarities
are identified and no new cuts are maded.
*    @author     Juan Coronado
*    @created    November 11, 2004
*/
public class DeCompose  {
   Map      segmentMap;
   SequenceDB  hashSeqDB      = new HashSequenceDB();
   Sequence    bioSeq         = null;
   ResultSet   strSeq         = null;
   boolean cuts = false;
   int[]       hspQueryStart, hspQueryEnd, hspsQuery    =
null;

/** if cuts were performed, returns true; else false **/
   boolean getCuts(){
      return cuts;
   }

/**
* sort from smallest to largest an array with start and end
alignment positions positions in the
*query    sequence
*/
void sort()  {
      for (int i = hspQueryEnd.length; --i >= 0; ) {
         boolean  flipped  = false;
         for (int j = 0; j < i; j++) {
```

```
            if (hspQueryEnd[j] > hspQueryEnd[j + 1]) {
               int  E  = hspQueryEnd[j];
               int  S  = hspQueryStart[j];
               hspQueryEnd[j] = hspQueryEnd[j + 1];
               hspQueryEnd[j + 1] = E;
               hspQueryStart[j] = hspQueryStart[j + 1];
               hspQueryStart[j + 1] = S;
               flipped = true;
            }

            /*order query_start from largest to smallest
(reverse ordering with reference to
                  query_end; so cuts at HSPs that have the
same query_end  will use the shortest */
            if (hspQueryEnd[j] == hspQueryEnd[j + 1] &&
hspQueryStart[j] < hspQueryStart[j+1]) {
               //   int  E  = hspQueryEnd[j];
               int  S  = hspQueryStart[j];
               //   hspQueryEnd[j] = hspQueryEnd[j + 1];
               //   hspQueryEnd[j + 1] = E;
               hspQueryStart[j] = hspQueryStart[j + 1];
               hspQueryStart[j + 1] = S;
               flipped = true;
            }

         }
     // no flips are made; means we are done
         if (!flipped) {
            return;
         }
      }
   }

   /**
    *  Gets the segmentMap attribute of the DeCompose object
    *
    * @return    The segmentMap value
    */
   public Map getSegmentMap() {
      return segmentMap;
   }


/** define fragments of at least cut(parameter) amino acids
in length from BLAST HSPs. Data
```

```
inputs needed: biojava Sequence object containing sequence
characteristics (ID, sequence string), a database resultset
with the sequence coordinates of start and end for each HSP
and ordered by query end or subject end for the reciprocal
comparisons (same sequence as query or subject); the number
of rows or HSPs, and the size of the minimum fragment
allowed (cut)
**/
 public void setFragMapVariable(Sequence bioSeq,  ResultSet
HspsOrderByQryEnd,
      ResultSet HspsOrderBySubEnd, int numRows, int cut) {
      this.bioSeq = bioSeq;
      String proteinID = bioSeq.getName();
      segmentMap = new HashMap();
      hspQueryEnd = new int[numRows];
      hspQueryStart = new int[numRows];

      try {
         /* assign hspQueryStart and hspQueryEnd values
from the ResultsSet gotten from the
     database for the protein as query and subject during
the sequence comparisons*/
         int          i   = 0;
         while (HspsOrderByQryEnd.next()) {
            hspQueryStart[i] =
HspsOrderByQryEnd.getInt("query_start");
            hspQueryEnd[i] =
HspsOrderByQryEnd.getInt("query_end");
            System.out.println(    i+" "+hspQueryStart[i] +"
"  +hspQueryEnd[i]);
            i++;
         }
         System.out.println(" query as subject");
         while (HspsOrderBySubEnd.next()) {
            hspQueryStart[i] =
HspsOrderBySubEnd.getInt("subj_start");
            hspQueryEnd[i] =
HspsOrderBySubEnd.getInt("subj_end");
            System.out.println(i+" "+hspQueryStart[i] +" "
+hspQueryEnd[i]);
            i++;
         }
         /* sort hspQueryStart and hspQueryEnd arrays */
         sort();
         // number of  segment
```

```
         int       hsp            = 1;
         // assign current cut position, that being 0 when
beginning the decomposition process
         int       currentCutPos  = 0;
         for (int k = 0; k < hspQueryEnd.length; k++) {
            int  query_start  = hspQueryStart[k];
            int  query_end    = hspQueryEnd[k];
            System.out.println("query_start:
"+hspQueryStart[k] +" query_end: "+hspQueryEnd[k]);
            /* if current cut position to HSP query_end is
at least "cut" AA */
            if (query_end-(currentCutPos+1) >= (cut-1) ) {
                  /* check to see if a segment of "cut"
AA or more separates the  HSPs from the
                   beginning of the fragment and the cut
will not leave a segment of less than "cut" AA  end
     cut site to end os quence,  proceed to cut from
currentCutPosition+1 to query_start-1 and        from
query_start to query_end */
               if ((query_start-1)-(currentCutPos+1) >=
(cut-1)
               && bioSeq.length() - (query_start+1)  >=
(cut-1)){
          // put segment sequence string and id into segment
Map for this protein sequence
                  segmentMap.put(proteinID
+"_"+hsp,bioSeq.subStr(currentCutPos+1, query_start-1));
                  System.out.print("HSP_cut#"+ hsp);
                  System.out.println(" :1cut from "+
(currentCutPos+1)+"   "+"to "+(query_start-1));
                  // problem with cutting at currentCutPos
+1: sometimes left one letter when currentCutPos =
     query_start in above
                  // if loop ; replaced
currentCutPos=query_start with currentCutPos=query_start-1;
should             fix problem 9-22-05
                  currentCutPos = query_start-1;
          // increase segment count by one
                  hsp += 1;
          // a cut occur; update cuts boolean to true
                  cuts=true;
          /*if a cut from the current cut position to query
end position does not leave a segment of             cut
amino acids in length, make a cut from current cut position
+1 to query_end*/
```

```
                    if (bioSeq.length() - (query_end+1)  >=
(cut-1)){
                       segmentMap.put(proteinID
+"_"+hsp,bioSeq.subStr(currentCutPos+1, query_end));
                       System.out.print("HSP_cut#"+ hsp);
                       System.out.println(" :2cut from "+
(currentCutPos+1)+"   "+"to "+query_end);
                       currentCutPos = query_end;
                       hsp += 1;
                    }
                 }
          /*if a cut from the current cut position to query
end position does not leave a segment of              cut
amino acids in length, make a cut from current cut position
+1 to query_end:
          taking the current cut position to query end*/
     else if (bioSeq.length() - (query_end+1)  >= (cut-1)){
                    /* else just cut from currentCutPosition
+1 to query_end */
                    segmentMap.put(proteinID
+"_"+hsp,bioSeq.subStr(currentCutPos+1, query_end));
                    System.out.print("HSP_cut#"+ hsp);
                    System.out.println(" :3cut from "+
(currentCutPos+1)+"   "+"to "+query_end);
                    currentCutPos = query_end;
                    hsp += 1;
                    cuts=true;
                 }

             }
                 /* if last HSP , and current position to
the end of sequence is equal to or
                    greater than minimum cut length, put rest
of sequence into the MAP . */

             if (k == (hspQueryEnd.length - 1) &&
bioSeq.length() - (currentCutPos+1) >= (cut-1)){
                segmentMap.put(proteinID
+"_"+hsp,bioSeq.subStr(currentCutPos +1, bioSeq.length()));
                System.out.println( "last cut# "+ hsp);
                System.out.println(" :cut from "+
(currentCutPos+1)+"   "+"to "+bioSeq.length());
             }

          }
```

```
      } catch (SQLException se) {
         System.out.println("We got an exception while
getting a result:this " +
            "shouldn't happen: we've done something really
bad.");
         se.printStackTrace();
         System.exit(1);
      } /*catch (BioException be) {}*/

   }

  /** compare fragments to each other after cuts are made
to query sequences ; this is required to perform the second
set of fragment  definitions if there needs to be any other
cuts in the query sequences as in repetitive sequences**/
   public void compareFrags(File vectorPath, String
program){
      Vector fragMapvector ;
      WriteObject wo = new WriteObject();
      /* get vector with protein fragments from disk */
      fragMapvector = (Vector) wo.getSaveObject(vectorPath);
      int num=1;
      String dir = "/opt/tomcat/webapps/modmat/tree";
      /* print fasta file for comparisons*/
      File fastaFile = new File(dir+File.separator
+vectorPath.getName()+"_fasta");

      if (fastaFile.exists()){
         fastaFile.delete();
      }
      for(int i=0;i < fragMapvector.size(); i++){
         /* get Map with proteinID_hsp# as keys and
sequence as values */
         Map seqFrags = (Map)fragMapvector.get(i);
         for (Iterator it=seqFrags.keySet().iterator();
it.hasNext(); ){
            /*get proteinID_hsp#*/
            String qID = (String) it.next();
            /* get sequence for  proteinID_hsp#*/
            String qStrSeq=(String)seqFrags.get(qID);
            try {
               Sequence          prot  =
ProteinTools.createProteinSequence(qStrSeq, qID);
               FileOutputStream  fos   = new
FileOutputStream
```

```
                    (fastaFile,true);
                /* write fasta file to disk */
                SeqIOTools.writeFasta(fos, prot);
                System.out.println(num++ +" " +qID);

            } catch (BioException be) {
                System.err.println("problem with
createProteinSequence");
            } catch (IOException e) {
                System.err.println("problem with
createProteinSequence IO:"+ e.getCause());
            }
        }

      }
     // format the newly created FASTA files to be used with
the sequence comparison with
     // composition modification of the scoring matrix
      String[]  formatdb  = {"/usr/local/bin/formatdb", "-
i", fastaFile.toString() };
      String[]  cmdLine  = {"/public/programs/MODPSC.pl",
"-P",program , "-i",
      fastaFile.toString(), "-d",fastaFile.toString(),"-G",
"13", "-E", "1", "-e", "1e-3"};

      String outFile = vectorPath.getName()+"_output";

      try {
     // run the commands defined above to format the
sequence files and run the BLAST
     // comparisons with modified matrices
         RunProcess  ps  = new RunProcess();
         ps.command(formatdb, new File(dir) );
         ps.command(cmdLine,outFile, new File(dir) );
         System.out.println("finished running process
formatdb MODPSC");
      } catch (IOException e) {
         System.out.println("error occurred during running
process formatdb MODPSC");
      }
```

```
    }

}
```

JcTools.java

This java class contains many useful methods that are used by the other java classes,

```
/*
 * JcTools.java
 *
 * Created on June 7, 2005, 9:41 AM
 */

package hunter.cuny.edu;
import NEXUS.NEXUSUsageBean;
import NEXUS.NEXUSWriter;
import Objects.TaxaBeanManager;
//import com.myjavatools.web.ClientHttpRequest;
import java.io.*; import java.util.*;
import org.biojava.bio.seq.io.*;import
org.biojava.bio.BioException;
import org.biojava.bio.dist.*; import
org.biojava.utils.ChangeVetoException;
import org.biojava.bio.seq.*;import java.net.*;import
java.sql.ResultSet;
import java.sql.SQLException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.methods.PostMethod;
import
org.apache.commons.httpclient.methods.multipart.FilePart;
import
org.apache.commons.httpclient.methods.multipart.MultipartRe
questEntity;
import org.apache.commons.httpclient.methods.multipart.Part;
import
org.apache.commons.httpclient.methods.multipart.StringPart;
import org.biojava.bio.BioError;
import org.biojava.bio.dist.Distribution;
import org.biojava.bio.symbol.*;
import org.biojava.bio.program.sax.*;
import org.biojava.bio.program.ssbind.*;
import org.biojava.bio.search.*;
```

```
import org.biojava.bio.seq.SequenceIterator;
import org.biojava.bio.seq.db.*;
import org.biojava.bio.symbol.Symbol;
import org.htmlparser.beans.LinkBean;
import org.htmlparser.util.ParserException;
import org.xml.sax.*;
import org.htmlparser.Parser;
import org.htmlparser.Tag;
import org.htmlparser.Text;
import org.htmlparser.tags.LinkTag;
import org.htmlparser.visitors.NodeVisitor;
/**
 *
 * @author  Juan Coronado
 */
public class JcTools {

   /** Creates a new instance of JcTools: a class to whole
my frequently called methods  **/
   public JcTools() {
   }


   /** get a Biojava SequenceIterator object to read FASTA
format from a given file,  **/
   public static SequenceIterator
getSequenceIterator( String file){
      SequenceIterator stream = null ;
      System.out.println("file="+file);
      try{
         BufferedReader br = new BufferedReader(new
FileReader(file));
         stream = SeqIOTools.readFastaProtein(br);

      }catch (FileNotFoundException ex) {
         ex.printStackTrace();
      }
      return stream;
   }

   /** this method will divide  a fasta-formatted db file
into multiple (numOfDivisions) fasta files;
    * truncates the sequence of each entry if signalp
(requires only the first 50 aminoa acids sequence)  is
   * true from truncStart to truncEnd
```

```
 **/
   public static void divideDB
      (String filepath, int numOfDivisions, int truncStart,
int truncEnd, boolean signalp){
      try {
         File file = new File(filepath);
         BufferedInputStream  indb = new
BufferedInputStream(new FileInputStream(file));
         //get the appropriate Alphabet
         AlphabetManager   am          = new
AlphabetManager();
         Alphabet          alpha       =
am.alphabetForName("PROTEIN-TERM");
         //get a SequenceDB of all sequences in the file
         //get a SequenceDB of all sequences in the file
         SequenceDB hashSeqDB = SeqIOTools.readFasta(indb,
alpha);
         int sizeOfDB = hashSeqDB.ids().size();
         System.out.println("sizeOfDB="+sizeOfDB);
         int i = 1;
         SequenceIterator  seqIt =
hashSeqDB.sequenceIterator();
         int seqCount = 0;
         double seqsPerFile = Math.rint((double)sizeOfDB/
(double)numOfDivisions);
         System.out.println("seqsPerFile="+seqsPerFile);
         while (i <= numOfDivisions){
            File divide = new File(file.getParent()
+File.separator+file.getName()+i);
            if (divide.exists()){ divide.delete();}
            FileOutputStream  fos   = new
FileOutputStream(divide,true);
            System.out.println(divide.toString());

            while (seqIt.hasNext()){
               Sequence prot  = seqIt.nextSequence();
//            System.out.println(prot.seqString());
               // create trucated  sequence for signalp
input
               if(signalp){
                  Sequence truncated  =null;
                  if (truncEnd <= prot.length()){
                     truncated  =
                        ProteinTools.createProteinSequence
                        (prot.subStr(truncStart,truncEnd),
```

```
prot.getName());
                    }else {
                       truncated  =
                          ProteinTools.createProteinSequence
                          (prot.subStr(truncStart,prot.length
()), prot.getName());
                    }
                    /* write fasta file to disk */
                    SeqIOTools.writeFasta(fos, truncated);
                 }else {
                    /* write fasta file to disk */
                    SeqIOTools.writeFasta(fos, prot);
                 }
                 //System.exit(0);
                 seqCount+=1;
                 double round = (double)seqCount/
(double)seqsPerFile;
                 // the equal sign here equate .9998498273014
equal to 1.0
                 // and i would be left with one sequence
missing
                 if (round > (double)i ){
                    System.out.println("\nround "+round+" has
ended.");
                    System.out.println("seqCount="+seqCount);
                    break;
                 }
              }
              System.out.println("\nseqCount: "+i+"
"+seqCount);
              i+=1;
              fos.close();
           }
        } catch (BioException ex) {
           ex.printStackTrace();
        }catch (FileNotFoundException ex) {
           ex.printStackTrace();
        } catch (IOException ex) {
           ex.printStackTrace();
        }

     }

  /** read FASTA format from file, count amino acids,
     * and generate a BIOJAVA amino acid distribution
```

```
object**/
   public static Distribution getAminoAcidDistr( String
file){
      Distribution aaDist = null;
      try{
         FiniteAlphabet aminoAcids =
ProteinTools.getAlphabet();
         //Distribution aaDist = new
SimpleDistribution(aminoAcids);

         aaDist =
DistributionFactory.DEFAULT.createDistribution(aminoAcids) ;
         BufferedReader br =
            new BufferedReader(new FileReader(file));
         SequenceIterator stream =
SeqIOTools.readFastaProtein(br);
         Map <String, Integer> aa_frqncy = new HashMap();
         int numOfResidues = 0;
         int numOfSeqs = 0;
         while (stream.hasNext()) {
            Sequence seq = stream.nextSequence();
            String[] seqArray = seq.seqString().split("");
     /* count amino acid sequence and put into the aa_frqncy
hash map  */
            for (int i=1; i < seqArray.length; i++){
               if (!
seqArray[i].toUpperCase().matches("[XBZU]")){
                  if (aa_frqncy.get(seqArray[i]) == null){
                     aa_frqncy.put(seqArray[i], 1);
                  }else{
                     aa_frqncy.put(seqArray[i],aa_frqncy.ge
t(seqArray[i])+1);
                  }
                  numOfResidues++;
               }
            }
            numOfSeqs++;
         }

         Iterator <String> it =
aa_frqncy.keySet().iterator();
         double totalPercent =0;
         System.out.println("average size of ORF :"
            + (double)numOfResidues/(double)numOfSeqs +"
numofOrfs="+numOfSeqs);
```

```
         while (it.hasNext()) {
            // Get key
            String key = it.next();
            /* get amino acid probability */
            double probability = (double)aa_frqncy.get(key)
/ (double)numOfResidues ;
            totalPercent += probability;
            System.out.println(key+" "+probability +" "+
numOfResidues+" "+totalPercent);
            // System.out.println(key+" "+probability );
                /* translated single letter amino acid to a
biojava Symbol
                  and set the probability in the
distribution*/
            aaDist.setWeight(
               aminoAcids.getTokenization("token").parseTok
en(key), probability );
             );

         }

      }catch (IOException e){
         System.out.println("Unable to read file:
"+e.getMessage());
      }catch (BioException be) {
         System.err.println( "hi " +be.getMessage());
      }catch (ChangeVetoException ise){
         System.err.println( ise.getMessage());
      }
      return aaDist;
   }

   /**shuffle sequences every window-size amino acids
(Fisher-Yates shuffle). Returns a shuffled char  array */
   public static char[]  shuffleEveryWindow(char[]
randSeq1, int window){
      for (int i = 0; i < randSeq1.length;){
         //System.out.println("i"+i);
         Random rand = new Random();
         /* if sequence is not an exact multiple of window
*/
         if(i+window-1 >= randSeq1.length){
            window = randSeq1.length - i;
         }
         /* make j == last index in window, and decrement */
```

```
         for (int j= (i+window)-1; j >= i;j-- ) {
            /* pick a number randomly  from (including) 0
to window(excluding) */
            int r =  rand.nextInt(window);
            //if indices are same-don't bother
            if ( j == i + r){ continue ;}
            /* swap char at these indices( j and i+r); */
            char temp = randSeq1[i+r];
            randSeq1[i+r]= randSeq1[j];
            randSeq1[j] = temp;
            //System.out.println("i"+i+"i+r"+(i+r)+"j"+j);
         }
         i +=window;
      }
//      System.out.println(randSeq1);
      return randSeq1;
   }

   /** make a sequence with the same amino acid composition
of the distribution; returns a char array
        with the same amino acid composition as the desired
distribution**/
   public static char[] makeSequenceWithDist(Distribution
d, int len){
      char[] randSeq1 = new char[len];
            int p =0;
      try {
         SymbolTokenization st =
d.getAlphabet().getTokenization("token");
         for(Iterator i = ((FiniteAlphabet)
d.getAlphabet()).iterator(); i.hasNext(); ) {
            AtomicSymbol s = (AtomicSymbol) i.next();
                /* selenocysteine not part of
distribution(would have NaN value)
                 ,but part of Biojava alphabet*/
            double aaProb = d.getWeight(s);
            String aa =st.tokenizeSymbol(s);
            // System.out.println("p "+ aaProb);
            if (aaProb >= 0 ){
               for (int j=0; j < Math.round(aaProb*len);j+
+){
                  randSeq1[p] = (aa.toCharArray())[0];
                  p++;
                        /*make sure that sequence is not
longer than specified length
```

```
                        *do to rounding */
                    if( p == len ){ break; }

                }
            }
         }
         /* if rounding produce a short sequence, add to
until specified length */
         while (p <len){
     // pick an amino acid randomly from the amino acid
distribution
            Symbol symDist =  JcTools.sampleSymbol(d) ;
            String aa =st.tokenizeSymbol(symDist);
            randSeq1[p]= (aa.toCharArray())[0];
            p++;

         }
         /*globally shuffle sequence */
         randSeq1 =
shuffleEveryWindow(randSeq1,randSeq1.length);

      } catch (IllegalSymbolException ire) {
         throw new BioError(
            "Unable to iterate over all symbols in alphabet
- " +
            "things changed beneath me!", ire
            );
      }catch (BioException be) {
         System.err.println( "hi " +be.getMessage());
      }
      return randSeq1;
   }

   /** pick a symbol from an Alphabet (protein, dna, etc..)
using the given distribution
    * of the alphabet **/
   public static Symbol sampleSymbol(Distribution d) {
      // pick a random probability from 0 to 1
      double p = Math.random();
      AtomicSymbol returnSym = null;
      try {
         for(Iterator i = ((FiniteAlphabet)
d.getAlphabet()).iterator(); i.hasNext(); ) {
            AtomicSymbol s = (AtomicSymbol) i.next();
            /* selenocysteine not part of
```

```
distribution(would have NaN value) but part of Biojava
          alphabet*/
     /*if the amino acid has a probability greater than
zero, subtract the probability from the random
     probability; if the result is negative return the
sampled symbol; otherwise continue until the     random
probability becomes negative after repeated subtractions of
amino acid probabilities in   the distribution*/
            if (d.getWeight(s) >= 0){
               p -= d.getWeight(s);
               if( p <= 0) {
                  //System.out.println("p <= 0");
                  returnSym = s;
                  break;
               }
            }
         }


      } catch (IllegalSymbolException ire) {
         throw new BioError(
            "Unable to iterate over all symbols in alphabet
- " +
            "things changed beneath me!", ire
            );
      }
      return returnSym;
   }

   /** make a random sequence of length "len" with the
composition of the distribution "d"; returns a
         a List for better manipulation of sequence than
using arrays**/
   public static List makeRandomSequence(Distribution d,
int len){
      List randSeq1 = new ArrayList();
      try{
         Map<String,Integer> aaNum = new HashMap();
         SymbolTokenization st =
d.getAlphabet().getTokenization("token");
         for (int j = 0; j < len; j++) {
            /* sample an amino acid from nr dist and append
to string buffer */
            Symbol symDist1 =  JcTools.sampleSymbol(d) ;
            String aa =st.tokenizeSymbol(symDist1);
```

```
            randSeq1.add( aa);
     // making sure the amino acid distribution is
corresponding to the new sequence
            if (aaNum.get(aa) == null){
               aaNum.put(aa,1);

            }else{
               aaNum.put(aa,aaNum.get(aa)+1);
            }
         }
         System.out.println(aaNum+"\n"+randSeq1);
      }catch (IllegalSymbolException ire) {
         throw new BioError(
            "Unable to iterate over all symbols in alphabet
- " +
            "things changed beneath me!", ire
            );
      }catch (BioException be) {
         System.err.println( "hi " +be.getMessage());
      }
      return randSeq1;
   }

   /** read output from FASTA programs fasta33, ssearch33,
make sure that -m 10 option
    * is usedl uses biojava to read FASTA output; returns a
List of  SeqSimilaritySearchResults object
     one for each sequence comparison */
   public static List readFastaOutput(URL url) {
      List results = null;
      try {
         XMLReader   parser   = (XMLReader)new
FastaSearchSAXParser();
         //make the SAX event adapter that will pass events
to a Handler.
         SeqSimilarityAdapter  adapter  = new
SeqSimilarityAdapter();
         //set the parsers SAX event adapter
         parser.setContentHandler(adapter);
         //The list to hold the SeqSimilaritySearchResults
         results = new ArrayList();
         //create the SearchContentHandler that will build
SeqSimilaritySearchResults
         //in the results List
         SearchContentHandler  builder  = new
```

```
BlastLikeSearchBuilder(results,
            new DummySequenceDB("queries"), new
DummySequenceDBInstallation());
         //register builder with adapter
         adapter.setSearchContentHandler(builder);
         //parse the file, after this the result List will
be populated with
         //SeqSimilaritySearchResults
         parser.parse(new InputSource(url.openStream()));
      } catch (IOException ex) {
         //IO problem, possibly file not found
         ex.printStackTrace();
      } catch (NoSuchElementException ex) {
         //no fasta sequences in the file
         ex.printStackTrace();
      } catch (Exception e) {
         e.printStackTrace();
      }
      return results;
   }

/* read file a from the file system ; returns a
bufferedreader object */
   public static BufferedReader readFile(String file){
      BufferedReader in = null;
      try{
         in = new BufferedReader(new FileReader(file));
      }catch(FileNotFoundException fnfe){
         System.err.println(fnfe.getMessage());
      }
      return in;
   }

   /** takes  a fasta protein sequence database, randomizes
window-size regions for each ORF
    *  repeat times for each ORF; makes a new database with
name as dbName.rand+window    **/
   public static void randomizeDBX(String dbName, int
window, int repeat){
      try{
         File dbFile = new File(dbName+".rand"+window);
         if ( dbFile.exists() ){ dbFile.delete(); }
         BufferedReader br =
            new BufferedReader(new FileReader(dbName));
         FileOutputStream fos = new FileOutputStream(
```

```
            dbFile,true);
          Sequence randSeq = null;
          String prot = null;
          Sequence seq = null;
          char[] seqArray = null;
          SequenceIterator stream =
SeqIOTools.readFastaProtein(br);
          while (stream.hasNext()) {
             seq = stream.nextSequence();
             System.out.println(seq.getName()+"
"+seq.length());
             for (int i=1; i<=repeat;i++){
                seqArray =
shuffleEveryWindow(seq.seqString().toCharArray(), window);
                prot = new String(seqArray);
     // create sequence object with id and randomized
sequence
                randSeq =
ProteinTools.createProteinSequence(prot,seq.getName()+"."+i
+"r");
                /* sequence to file */
                if (i == 1){
                   SeqIOTools.writeFasta(fos, seq);
                   System.out.println(seq.length()+"
"+seq.seqString());
                }
                SeqIOTools.writeFasta(fos, randSeq);
             }
          }
     // close file outputstream and bufferedreader objects
          fos.close();
          br.close();
       }catch (BioException be) {
          System.err.println( "hi " +be.getMessage());
       }catch(FileNotFoundException fnfe){
          System.err.println(fnfe.getMessage());
       }catch (IOException e){
          System.out.println("Unable to read file:
"+e.getMessage());
       }
    }

    /**send a post request to signalP prediction server,
receive response, and return the prediction
     * based on the  method choosen (nn,hmm, nn+hmm),
```

```
orgtype(euk, gram-,gram+)
    * batch mode ; give a fasta file ; get a Set array with
two sets(signalP+,signalP-);
    * uses Commons HttpCLient for connection; the method
will wait until results are returned from
    * server 10-14-2005;waitBtwnRequests: in minutes, the
time to wait between requests to server**/
   public static Set[] batchSignalP(String filePath, String
orgtype, String method,int trunc,
      double waitBtwnRequests) {
      System.out.println(" Checking for signal peptide");
      boolean nnPred = false ;boolean hmmPred = false;
boolean hmmNNpred =false;
      File file = new File(filePath);
      Set[] sets = new HashSet[2];
     //set up parametes to acces web site
      try {
         PostMethod post = new
PostMethod("http://www.cbs.dtu.dk/cgi-bin/nph-webface");
         Part[] parts = {
            new StringPart("configfile",
"/usr/opt/www/pub/CBS/services/SignalP-3.0/SignalP.cf"),
            new StringPart("orgtype", orgtype),
            new StringPart("method", method),
            new StringPart("graphmode", ""),
            new StringPart("format","short"),
            new StringPart("trunc",String.valueOf(trunc)),
            new FilePart("SEQSUB", file)
            //new StringPart("SEQSUB",file.getName())
         };
         post.setRequestEntity(new
MultipartRequestEntity(parts,post.getParams()) );
     // connect to web site
         HttpClient client = new HttpClient();
         client.getHttpConnectionManager().
            getParams().setConnectionTimeout(5000);
         int status = client.executeMethod(post);
         if (status == HttpStatus.SC_OK) {
            InputStream in = post.getResponseBodyAsStream();
            BufferedReader     inL          = new
BufferedReader(new InputStreamReader(in));
            String             strL;
           // a regular expression to parse the html
received from web site
      Pattern pattern = Pattern.compile("^.*href\\=\"(http.
```

```
+)\".+$");
            URL predictionUrl = null;
     // loop to identify the url with the html results
            while ((strL = inL.readLine()) != null) {
               System.out.println(strL);
               Matcher matcher = pattern.matcher(strL);
               if (matcher.matches()){
                  predictionUrl = new URL(matcher.group(1));
                  System.out.println(predictionUrl);
               }
            }

            // contains names of sequences with signal
peptide as predicted by signalp 3.0
            Set positive = new HashSet();
            // contains names of sequences with no signal
peptide as predicted by signalp 3.0
            Set negative = new HashSet();
            // are the results ready at the predictionUrl??
            boolean results = false;
            pattern = Pattern.compile("^[A-Z].+\\d
+\\.\\d\\d\\d.+$");
     // keep on trying to get results
            while(!results) {
               System.out.println("Trying to get results
from: "+predictionUrl.toString());
               //give time for the server to do the
prediction
               System.out.println("Waiting
"+waitBtwnRequests+" minute(s) before trying");
               Thread.sleep((int)Math.round(waitBtwnRequest
s*60*1000));
               // open link as inputStream and parse for
prediction
               inL  = new BufferedReader(new
InputStreamReader(predictionUrl.openStream()));
               while ((strL = inL.readLine()) != null) {
                  // outL.write(strL);
                  Matcher matcher = pattern.matcher(strL);
                  System.out.println(strL);
                  if (matcher.matches()){
          // remove "Sequence" and split the line at one or
more spaces or tab
                     String[] line =
strL.replaceAll("Sequence","").split(" +|\\t");
```

```
          //  take the 14th and 20th elements showing the
neural network and hidden markov model
          // predictions
                        String nn = line[13].trim(); String
hmm = line[20].trim();
                        System.out.println("prediction: "+nn+"
"+hmm );
                        System.out.println(Arrays.asList(line)
+"hello");
                        if (method.equals("nn+hmm") &&
nn.equals("Y") && hmm.equals("Y")){
                           //server cuts off  orf/gene names
that are too long; use the HMM  orf
                           // at index 14
          // System.out.println(line[0]);
          // positive.add(line[0]);
                           System.out.println("positive:
"+line[14]);
                           positive.add(line[14]);
                        }else{
                           System.out.println("negative:
"+line[14]);
                           negative.add(line[14]);
                        }
                        results=true;
                     }
                  }
                  inL.close();
               }
               sets[0]=positive;sets[1]=negative;
               post.releaseConnection();
            } else {
               System.err.println(
                  "Upload failed, response=" +
HttpStatus.getStatusText(status)
                  );
            }
         } catch (IOException e) {
            System.err.println("problems with
batchSignalP:"+e.getMessage());
         } catch (InterruptedException ex) {
            ex.printStackTrace();
         }
         return sets ;
      }
```

```
/**send a post request to tmHmm prediction server, receive
response, and return the prediction
    * batch mode ; give a fasta file ; get a Set array with
two sets(TM+,TM-);
    * uses Commons HttpCLient for connection; the method
will wait until results are returned from
    * server 10-14-2005;waitBtwnRequests: in minutes, the
time to wait between requests to server
    * signalpCut - the number of N-terminal amino acids
that may be resemble a signal sequence
    * gpiCut- the number of C-terminal amino acids that may
resemble a GPI anchor signal
    *discardGpiSignal - boolean expressing wish to discard
signal or gpi signals with transmembrane
    *domains not overlapping with either
**/
   public static Set[] batchCheckTransMembrane(String
filePath,double waitBtwnRequests,
      int signalpCut, int gpiCut,boolean discardGpiSignal) {
      System.out.println(" Checking for transMembrane ");

      File file = new File(filePath);
      Set[] sets = new HashSet[2];
      try {
     // set up parameters for post connection
         PostMethod post = new
PostMethod("http://www.cbs.dtu.dk/cgi-bin/nph-webface");
         Part[] parts = {
            new StringPart("configfile",
"/usr/opt/www/pub/CBS/services/TMHMM-2.0/TMHMM2.cf"),
            new StringPart("outform","-short"),
            new FilePart("seqfile", file)
            //new StringPart("SEQSUB",file.getName())
         };
         post.setRequestEntity(new
MultipartRequestEntity(parts,post.getParams()) );
         HttpClient client = new HttpClient();
         client.getHttpConnectionManager().
            getParams().setConnectionTimeout(5000);
     // connect to web site
         int status = client.executeMethod(post);
         if (status == HttpStatus.SC_OK) {
            InputStream in = post.getResponseBodyAsStream();
```

```
            BufferedReader     inL          = new
BufferedReader(new InputStreamReader(in));
            String             strL;
            //Pattern pattern =
Pattern.compile("^.*href\\=\"(http.+wait)\".+$");
            Pattern pattern =
Pattern.compile("^.*href\\=\"(http.+)\".+$");
            URL predictionUrl = null;
            while ((strL = inL.readLine()) != null) {
               System.out.println(strL);
               Matcher matcher = pattern.matcher(strL);
               if (matcher.matches()){
                  predictionUrl = new URL(matcher.group(1));
                  System.out.println("predictionUrl:"+predi
ctionUrl);
                  //break;
               }
            }
            post.releaseConnection();
            // contains names of sequences with signal
peptide as predicted by signalp 3.0
            Set positive = new HashSet();
            // contains names of sequences with no signal
peptide as predicted by signalp 3.0
            Set negative = new HashSet();
            // are the results ready at the predictionUrl??
            boolean results = false;
     // regular expression to parse html results
            pattern = Pattern.compile("^(.+)len=(\\d+).
+PredHel=(\\d+).+Topology=(.+)$");
            // open link as inputStream and parse for
prediction
            while(!results) {
               System.out.println("Trying to get results
from: "+predictionUrl.toString());
               //give time for the server to do the
prediction
               System.out.println("Waiting
"+waitBtwnRequests+" minute(s) before trying");
               Thread.sleep((int)Math.round(waitBtwnRequest
s*60*1000));
               inL  = new BufferedReader(new
InputStreamReader(predictionUrl.openStream()));
               while ((strL = inL.readLine()) != null) {
                  // outL.write(strL);
```

```
                    Matcher matcher = pattern.matcher(strL);
                    System.out.println("line: "+strL);
                    if (matcher.matches()){
                       String name = matcher.group(1).trim();
                       int len =
Integer.parseInt(matcher.group(2));
                       int numOfTm =
Integer.parseInt(matcher.group(3));
          // these booleans will remain false if
transmembrane does not overlap with gpi or signal
          // sequence
                       boolean signalp =false; boolean gpi =
false;
                       if (discardGpiSignal){
          // topology gives the coordinates of the
transmembrane helices
                          String topology =
matcher.group(4).trim().replaceAll("o","i")
                          .replaceAll("-","i");
                          String[] topologySegments
=topology.split("i");
                          System.out.println("topology:
"+Arrays.asList(topologySegments));
                          for (int i = 1 ; i <
topologySegments.length;){
                             int first =
Integer.parseInt(topologySegments[i] );
                             System.out.println("index:"+i
+"="+ first);
                             if (first <= signalpCut){
                                signalp = true;
                                System.out.println("signal
peptide: " +
                                   "a transmembrane domain
overlaps with the first "+signalpCut+" aa");
                             }
                             if (first > len - gpiCut){
                                gpi = true;
                                System.out.println("gpi
signal: " +
                                   "a transmembrane domain
overlaps with the last "+gpiCut+" aa");
                             }
                             i+=2;
                          }
```

```
                    }
                    System.out.println("numOfTm= "+
numOfTm);
          // if a no transmembrane or 1 transmembrane and
signal or gpi or 2 transmembrane and             //   gpi
and signal, then categorize as having no transmembrane
                    if( numOfTm == 0 || (numOfTm == 1 &&
(signalp|gpi)) ||
                        (numOfTm == 2 && gpi && signalp) ) {
                        negative.add(name);
                        System.out.println("no TM: "+name);
                    }else {
                        positive.add(name);
                        System.out.println("has TM: "+name);
                    }
                    results=true;
                 }

              }
              inL.close();
              //results=true;//only once
           }
     // put sets into set array to return as results
           sets[0]=positive;sets[1]=negative;

        } else {
           System.err.println(
              "In batchCheckTransMembrane: Upload failed,
response=" + HttpStatus.getStatusText(status)
              );
        }
      } catch (IOException e) {
         System.err.println("problems with
batchCheckTransMembrane:"+e.getMessage());
      } catch (InterruptedException ex) {
         ex.printStackTrace();
      }
      return sets ;
   }

/** uses the GPI-SOM server to predict proteins that have
C- and N-terminal signals for gpi and
* secretion signal
filePath- string representing to location of file with
```

```
fasta sequences to use as input
sequence**/
   public static Set[] checkForGpi(String filePath,String
sequence, double waitBtwnRequests) {
      System.out.println(" Checking for GPI ");
      File file = new File(filePath);
      Set[] sets = new HashSet[2];
     // set up parameters to pass onto website; need to
check website periodically to make sure
     //  address has not changed
      try {
         PostMethod post = new
PostMethod("http://genomics.unibe.ch/cgi-bin/gpi.cgi");
         Part[] parts = {
            new FilePart("fasta", file)
         };
         post.setRequestEntity(new
MultipartRequestEntity(parts,post.getParams()) );
         HttpClient client = new HttpClient();
         client.getHttpConnectionManager().
            getParams().setConnectionTimeout(5000);
     // connect to web site
         int status = client.executeMethod(post);
         if (status == HttpStatus.SC_OK) {
            InputStream in = post.getResponseBodyAsStream();
            BufferedReader     inL          = new
BufferedReader(new InputStreamReader(in));
            String             strL;
            Pattern pattern =
Pattern.compile("^.*href\\=(http.+)>Results.+$");
            URL gpiUrl = null;URL undecidedUrl= null;String
urlPath =null;
            while ((strL = inL.readLine()) != null) {
               System.out.println(strL);
               Matcher matcher = pattern.matcher(strL);
               if (matcher.matches()){
                  urlPath  = matcher.group(1);
          // parse out url with prediction results
                  System.out.println("urlPath: "+urlPath);
                  String gpiUrlPath
=urlPath.replaceFirst("ref=[\\d+]","ch=.pos.gpi");
                  String undecidedUrlPath =
urlPath.replaceFirst("ref=[\\d+]","ch=.und");
                  System.out.println("urlPath:
"+gpiUrlPath);
```

```
                    System.out.println("undecidedUrlPath:
"+undecidedUrlPath);
                    gpiUrl = new URL(gpiUrlPath);
                    undecidedUrl = new URL(undecidedUrlPath);
                 }
              }
              post.releaseConnection();
              // contains names of sequences with gpi &
signal peptide as predicted by GPI-SOM
              Set positiveGpi = new HashSet();
              // contains names of sequences with undecided
gpi &/signal peptide as predicted by GPI-SOM
              Set undedcidedGpi = new HashSet();
              // are the results ready at the predictionUrl??
              boolean results = false;
//         pattern = Pattern.compile("^>(.+[\\w\\|])\\s\\w.+
$");
              // the pipe gives problem with matching a
pattern; have to replace pipes with underscores
              pattern = Pattern.compile("^>.+$");
      // open link as inputStream and parse for prediction
              String geneName =null;
              while(!results) {
                 System.out.println("Trying to get results
from: "+gpiUrl.toString());
                 //give time for the server to do the
prediction
                 System.out.println("Waiting
"+waitBtwnRequests+" minute(s) before trying");
                 Thread.sleep((int)Math.round(waitBtwnRequest
s*60*1000));
                 inL  = new BufferedReader(new
InputStreamReader(gpiUrl.openStream()));
                 //did we finish getting the results for both
GPI-predictions and non-GPI predictions
                 boolean doneGpi =false;  boolean doneNotGpi
=false;
                 if ( inL.readLine() == null){
                    doneGpi =true;
                 }
                 while ((strL = inL.readLine()) != null) {
                    // outL.write(strL);
           // String replace = strL.replaceAll("\\|","_");
           //   Matcher matcher = pattern.matcher(replace );
                    Matcher matcher = pattern.matcher(strL );
```

```
          // System.out.println("line:"+replace );
                  System.out.println("line:"+strL );
                  if (matcher.matches()){
          // get gene name and add to positive gpi set
                     String[] split=strL.split(" ");
                     System.out.println(Arrays.asList(split
));
                     geneName
=split[0].replaceFirst(">","");
          // geneName =  geneName.replaceAll("_","\\|");
                     System.out.println("has GPI:
"+geneName);
                     positiveGpi.add(geneName);
                     doneGpi =true;
                  }
               }
               System.out.println("Trying to get results
from: "+undecidedUrl.toString());
               inL  = new BufferedReader(new
InputStreamReader(undecidedUrl.openStream()));
               if ( inL.readLine() == null){
                  doneNotGpi =true;
               }
               while ((strL = inL.readLine()) != null) {
                  Matcher matcher = pattern.matcher(strL);
                  System.out.println("line:"+strL);
                  if ( matcher.matches() ){
          // split the line at each space and take first
element of resulting array and remove “>”
                     geneName =strL.split(" ")
[0].replaceFirst(">","");
                     System.out.println("no GPI:
"+geneName);
                     undedcidedGpi.add(geneName);
                  }'
                  doneNotGpi =true;
               }
               inL.close();
               if (doneGpi | doneNotGpi){
                  results=true;
               }
            }
            sets[0]=positiveGpi;sets[1]=undedcidedGpi;
         } else {
            System.err.println(
```

```
                "Upload failed, response=" +
HttpStatus.getStatusText(status)
                );
            }
         } catch (IOException e) {
            System.err.println("problems with
checkForGPI:"+e.getMessage());
         } catch (InterruptedException ex) {
            ex.printStackTrace();
         }
         return sets ;
      }

      /**retrieves the set of ORFs annotated with the given GO
id**/
      public static Set getOrfsWithGOid(String[] goID){
         Set go = new HashSet();
         try {
            LinkBean lb = new LinkBean();
            BufferedReader inL = null;
            String strL = null;
            Pattern pattern = Pattern.compile("^.+/Y.+\\(\\d
+\\).+$");
            for (int s = 0; s < goID.length; s++){
               lb.setURL("http://db.yeastgenome.org/cgi-
bin/GO/go.pl?goid="+goID[s]);
               URL[] urls = lb.getLinks();
               for (int i = 0; i < urls.length; i++){
                  System.out.println(urls[i]);
                  if (urls[i].getPath().matches("^.*\\.xls$")){
                     inL = new BufferedReader(new
InputStreamReader(urls[i].openStream()));
                     while((strL = inL.readLine()) != null){
                        System.out.println( strL );
                        Matcher matcher =
pattern.matcher( strL );
                        if ( matcher.matches() ){
                           String name =
strL.split("[/|\\s*]")[1];
                           go.add( name );
                           System.out.println( name);
                        }
                     }
                  }
               }
```

```
            System.out.println(goID[s]+": "+go.size());
         }
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      return go;

   }

/** get ORF names from a FASTA file **/
   public static LinkedHashSet getOrfNames(String file ){
      //SequenceIterator seqIt =
JcTools.getSequenceIterator("/public/dbase/cerevisiae.aa");
      SequenceIterator seqIt =
JcTools.getSequenceIterator(file);
      LinkedHashSet cwpSce = new LinkedHashSet();
      try {
         while (seqIt.hasNext()){
            Sequence seq = seqIt.nextSequence();
            String orfName = seq.getName();
            cwpSce.add(orfName);
            //System.out.println(orfName+" "+i+"
"+averageST+" "+numOfST);
         }
      } catch (NoSuchElementException ex) {
         ex.printStackTrace();
      } catch (BioException ex) {
         ex.printStackTrace();
      }
      return cwpSce;
   }

   /** checks every window-size amino acids for Serine and
Threonine (ST) percentages above given percent;
    The file must be FASTA format **/
   public static Set getYeastSTorfs(int window, double
percent, String file ,String toBsavedAs){
      //SequenceIterator seqIt =
JcTools.getSequenceIterator("/public/dbase/cerevisiae.aa");
      SequenceIterator seqIt =
JcTools.getSequenceIterator(file);
      Set richST = new HashSet();
      try {
         while (seqIt.hasNext()){
            Sequence seq = seqIt.nextSequence();
```

```
            String orfName = seq.getName();
            String seqStr = seq.seqString();
            //look at windows of length 50 amino acids
            for(int i =0; i < (seq.length() - window)+1; i+
+){
               int numOfST = 0;
               //count amino acid serine or threonine at
each position in the window
               for (int j=i; j < window + i; j++){
                  char s = seqStr.charAt(j);
                  if (String.valueOf(s).matches("[ST]") ){
                     numOfST++;//System.out.print(s);
                  }
               }
               double averageST = (double)numOfST /
(double)window;
               if(averageST >= percent){
                  richST.add(orfName);
                  //go on to next sequence
                  break;
               }
            }
         }
         System.out.println("Number Orfs:"+richST.size()+"
win="+window+"@"+percent+"\n"+richST);
         serializeObject(toBsavedAs,richST);
      } catch (NoSuchElementException ex) {
         ex.printStackTrace();
      } catch (BioException ex) {
         ex.printStackTrace();
      }
      return richST;
   }

   /** returns the object that was serialized (saved) to a
file */
   public static Object readSerializedObject(String file){
      Object object = null;
      try {
         FileInputStream fis = new FileInputStream(file);
         ObjectInputStream ois = new ObjectInputStream(fis);
         object = ois.readObject();
         ois.close();
      } catch (FileNotFoundException ex) {
         ex.printStackTrace();
```

```
      }catch (ClassNotFoundException ex) {
         ex.printStackTrace();
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      return object;
   }

   /** write the object (save) to a file */
   public static void serializeObject(String file,Object
object){
      try {
         FileOutputStream fos = new FileOutputStream(file);
         ObjectOutputStream oos = new
ObjectOutputStream(fos);
         oos.writeObject(object);
         oos.close();
      } catch (FileNotFoundException ex) {
         ex.printStackTrace();
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }
   public static Set readOnePerLineStrings(String file){
      BufferedReader br =null;
      try {
         br = new BufferedReader(new FileReader(file));
      } catch (FileNotFoundException ex) {
         ex.printStackTrace();
      }
      String strL ;
      Set set = new HashSet();
      try {
         while ((strL = br.readLine()) != null) {
            set.add(strL);
         }
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      return set;
   }
/** returns a SequenceDB object using a FASTA file **/
   public static SequenceDB getHashSeqDB(String file,
String alphabet){
      BufferedInputStream indb;
```

```
      SequenceDB hashSeqDB =null;
      try {
         indb = new BufferedInputStream(new
FileInputStream(file));
         //get the appropriate Alphabet
         Alphabet          alpha       = new
AlphabetManager().alphabetForName(alphabet);
         //get a SequenceDB of all sequences in the file
         hashSeqDB = SeqIOTools.readFasta(indb, alpha);

      } catch (FileNotFoundException ex) {
         ex.printStackTrace();
      } catch (BioException ex) {
         ex.printStackTrace();
      }
      return hashSeqDB;
   }

   public static List findOrthologs(String orf, int
subj_species){
      GetDatabase  db  = new GetDatabase();
      ResultSet rs = null;
      List orthologs =new ArrayList();
      try {
         rs = db.getResultSet("select * from fun_ortho
where query_id='"+orf
               +"' AND subj_species="+subj_species);
         while(rs.next()){
            String orthoName = rs.getString("subj_id");
            orthologs.add(orthoName);
         }
      } catch (SQLException ex) {
         ex.printStackTrace();
      }
      return  orthologs;
   }

   public static String getTaxonomyLinksFromHtml(String
url){

//      LinkBean lb = new LinkBean();
//      BufferedReader inL = null;
//      String strL = null;
//      Pattern pattern = Pattern.compile("^.+/Y.+\\(\\d
+\\).+$");
```

```
//      lb.setURL(url);
//      URL[] urls = lb.getLinks();
//      for (int i = 0; i < urls.length; i++){
//         System.out.println(i+" "+urls[i]);
//      }
      System.out.println("parsing...");
      String taxonomy = null;
      Parser parser;
      try {
         parser = new Parser(url);
//         TitleAtrributeVisitor titleAtrributeVisitor =
new TitleAtrributeVisitor();
         org.htmlparser.Node[] nodes =
parser.extractAllNodesThatAre(LinkTag.class);
         for(int i=0; i < nodes.length; i++){
            System.out.println(i+nodes[i].getText());
            if(nodes[i].getText().matches("^.
+TITLE=.kingdom.+")){
               System.out.println(i+nodes[i].getText());
               taxonomy = nodes[i].getChildren().asString();
//               System.out.println(taxonomy);
               System.out.println("parsing ended ...");
               break;
            }else if (nodes[i].getText().matches("^.
+TITLE=.phylum.+")){
               System.out.println(i+nodes[i].getText());
               taxonomy =
"p"+nodes[i].getChildren().asString();
//               System.out.println(taxonomy);
               System.out.println("parsing ended ...");
               break;
            }else if (nodes[i].getText().matches("^.
+TITLE=.order.+")){
               System.out.println(i+nodes[i].getText());
               taxonomy =
"o"+nodes[i].getChildren().asString();
//               System.out.println(taxonomy);
               System.out.println("parsing ended ...");
               break;
            }
//            else  {
////               System.out.println(i+nodes[i].getText());
//               taxonomy = "probablyProtist";
////               System.out.println(taxonomy);
////               System.out.println("parsing ended ...");
```

```
////                break;
//             }
         }
//          parser.visitAllNodesWith (titleAtrributeVisitor);
//           parser.parse(new HasAttributeFilter("TITLE"));
      } catch (ParserException ex) {
         ex.printStackTrace();
      }
      System.out.println(taxonomy);
      return taxonomy;
   }

   /** takes the NCBI kindgom or higher taxonomy of species
and returns that classification **/
   public static String
getTaxonomyLinksFromHtmlKingdomOrBefore(String url){

      System.out.println("parsing...");
      String taxonomy = null;
      Parser parser;
      try {
         parser = new Parser(url);
//          TitleAtrributeVisitor titleAtrributeVisitor =
new TitleAtrributeVisitor();
         org.htmlparser.Node[] nodes =
parser.extractAllNodesThatAre(LinkTag.class);
         int i=15;// kingom taxonomy should be in this link
//          for(int j=0; j < 20; j++){
//             System.out.println(j+nodes[j].getText());
//
System.out.println(nodes[j].getChildren().asString());
//          }
         if(nodes[i].getText().matches("^.+TITLE=.kingdom.
+") ){
            System.out.println(i+nodes[i].getText());
            taxonomy = nodes[i].getChildren().asString();
//               System.out.println(taxonomy);
            System.out.println("parsing ended ..." +
taxonomy);
         }else if(nodes[i+1].getText().matches("^.
+TITLE=.kingdom.+") ){
            System.out.println( (i+1)+nodes[i+1].getText());
            taxonomy = nodes[i+1].getChildren().asString();
//               System.out.println(taxonomy);
            System.out.println("parsing ended ..." +
```

```
taxonomy);
         } else {
            System.out.println("no kingdom categorization;
taking group after domain");
            System.out.println( (i-1)+nodes[i-1].getText());
            taxonomy = nodes[i-1].getChildren().asString();
            if(taxonomy.equals("Eukaryota") ||
taxonomy.equals("Fungi/Metazoa group") ){
               //take the taxonomy after Eukaryota domain
or super kingdom classification
               taxonomy=nodes[i].getChildren().asString();
            }
            System.out.println("parsing
ended ..."+taxonomy);
         }
      } catch (ParserException ex) {
         ex.printStackTrace();
      }

      return taxonomy;
   }


   public static void main(String[] args) {
   }

}
```

JcTools_2.java

This java class contains many useful methods that are used by the other java classes.

```
/*
 * JcTools.java
 *
 * Created on June 7, 2005, 9:41 AM
 */

package hunter.cuny.edu;
import com.sun.image.codec.jpeg.ImageFormatException;
import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGEncodeParam;
import com.sun.image.codec.jpeg.JPEGImageEncoder;
import java.awt.image.BufferedImage;
import java.io.BufferedInputStream;
import java.io.BufferedReader;
```

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FilenameFilter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.Set;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.methods.PostMethod;
import
org.apache.commons.httpclient.methods.multipart.MultipartRe
questEntity;
import org.apache.commons.httpclient.methods.multipart.Part;
import
org.apache.commons.httpclient.methods.multipart.StringPart;
import org.biojava.bio.BioException;
import org.biojava.bio.program.sax.BlastLikeSAXParser;
import
org.biojava.bio.program.ssbind.BlastLikeSearchBuilder;
import org.biojava.bio.program.ssbind.SeqSimilarityAdapter;
import org.biojava.bio.search.SearchContentHandler;
import org.biojava.bio.seq.DNATools;
import org.biojava.bio.seq.Sequence;
import org.biojava.bio.seq.SequenceIterator;
import org.biojava.bio.seq.db.DummySequenceDB;
import org.biojava.bio.seq.db.DummySequenceDBInstallation;
```

```
import org.biojava.bio.seq.db.HashSequenceDB;
import org.biojava.bio.seq.db.SequenceDB;
import org.biojava.bio.seq.io.SeqIOTools;
import org.biojava.bio.symbol.Alphabet;
import org.biojava.bio.symbol.AlphabetManager;
import org.biojava.bio.symbol.IllegalAlphabetException;
import org.biojava.bio.symbol.IllegalSymbolException;
import org.biojava.bio.symbol.SymbolList;
import org.biojava.utils.ChangeVetoException;/**
 *
 * @author  bioinfo
 */
import org.htmlparser.beans.StringBean;
import org.xml.sax.InputSource;
public class JcTools_2 {

   /** Creates a new instance of JcTools: a class to whole
my frequently called methods  ; methods in this are static;
they
require the following libraries: biojava, apache commons
httpclient **/
   public JcTools_2() {
   }

   public static void main(String[] args) {

   }

/** writes a FASTA file with sequences gotten from a list
of ORF names present in a FASTA datable (DBpath) **/
   public static void writeSequencesFromListOfGIs
      (Set listOfGis , String DBpath, String outfile,
String seqType) {
      try {
         BufferedReader br = new BufferedReader(new
FileReader(DBpath));
         SequenceIterator  seqIt = (SequenceIterator)
         SeqIOTools.fileToBiojava("fasta", seqType , br);
         HashSequenceDB subSetDb = new HashSequenceDB();
         while (seqIt.hasNext()){
            Sequence seq = seqIt.nextSequence();
            if ( listOfGis.contains( seq.getName() ) ){
               subSetDb.addSequence(seq);
            }
         }
```

```
         SeqIOTools.writeFasta(new
FileOutputStream(outfile),subSetDb);
      } catch (BioException ex) {
         ex.printStackTrace();
      } catch (FileNotFoundException ex) {
         ex.printStackTrace();
      }catch (ChangeVetoException ex) {
         ex.printStackTrace();
      }catch (IOException ex) {
         ex.printStackTrace();
      }
   }
   /** writes a FASTA file with sequences gotten from a
list of ORF names present in a FASTA datable (DBpath)
    for given number of sequences (numOfSeqs) **/
   public static void writeFastaSequencesFromListOfGIs
      (List listOfGis , String DBpath, String outfile,
String seqType, int numOfSeqs) {
      try {
         BufferedInputStream  indb = new
BufferedInputStream(new FileInputStream(DBpath));
         //get the appropriate Alphabet
         Alphabet          alpha       = new
AlphabetManager().alphabetForName("PROTEIN-TERM");
         SequenceDB genomeDB = SeqIOTools.readFasta(indb,
alpha);
         HashSequenceDB subSetDb = new HashSequenceDB();
         int p=1;
         Iterator <String> it = listOfGis.iterator();
         while (it.hasNext() && p <= numOfSeqs){
            Sequence seq = genomeDB.getSequence(it.next());
            subSetDb.addSequence(seq);
            p++;
         }
         SeqIOTools.writeFasta(new
FileOutputStream(outfile),subSetDb);
      } catch (BioException ex) {
         ex.printStackTrace();
      } catch (FileNotFoundException ex) {
         ex.printStackTrace();
      }catch (ChangeVetoException ex) {
         ex.printStackTrace();
      }catch (IOException ex) {
         ex.printStackTrace();
      }
```

```
   }

  /** get secondary structure predictions using the GOR4
website; you get a String array with a C (coil), E ,
H(helical)
    designation for each amino acid**/
   public static String[] getGORAssignments(String
sequence, double waitBtwnRequests){
      System.out.println(" Checking GOR assignments");
      String[] assignments = new String[sequence.length()];
      try {
         PostMethod post = new PostMethod("http://npsa-
pbil.ibcp.fr/cgi-bin/secpred_gor4.pl");
         Part[] parts = {
            new StringPart("notice", sequence),
            new StringPart("ali_width", "70"),

         };
         post.setRequestEntity(new
MultipartRequestEntity(parts,post.getParams()) );
         HttpClient client = new HttpClient();
         client.getHttpConnectionManager().
            getParams().setConnectionTimeout(5000);
         int status = client.executeMethod(post);
         if (status == HttpStatus.SC_OK) {
            InputStream in = post.getResponseBodyAsStream();
            BufferedReader     inL          = new
BufferedReader(new InputStreamReader(in));
            String             strL;
            Pattern pattern = Pattern.compile("^Prediction
result file.+href=(.+)>GOR4<.+$");
            URL predictionUrl = null;
            while ((strL = inL.readLine()) != null) {
               System.out.println(strL);
               Matcher matcher = pattern.matcher(strL);
               if (matcher.matches()){
                  predictionUrl = new URL("http://npsa-
pbil.ibcp.fr"+matcher.group(1));
                  System.out.println(predictionUrl);
               }
            }
            // are the results ready at the predictionUrl??
            boolean results = false;
            pattern = Pattern.compile("^([CEH])\\s+\\d+\\s
+\\d+.+$");
```

```
            //pattern = Pattern.compile("^[A-Z].+$");
            while(!results) {
               System.out.println("Trying to get results
from: "+predictionUrl.toString());
               //give time for the server to do the
prediction
               System.out.println("Waiting
"+waitBtwnRequests+" minute(s) before trying");
               Thread.sleep((int)Math.round(waitBtwnRequest
s*60*1000));
               // open link as inputStream and parse for
prediction
               inL  = new BufferedReader(new
InputStreamReader(predictionUrl.openStream()));
               int i = 0;
               while ((strL = inL.readLine()) != null) {
                  // outL.write(strL);
                  Matcher matcher = pattern.matcher(strL);
                  System.out.println(strL);
                  if (matcher.matches()){
                     String assignment = matcher.group(1);
                     assignments[i] = assignment;
                     i++;
                     results=true;
                  }
               }
               inL.close();
            }
            post.releaseConnection();
         } else {
            System.err.println(
               "Upload failed, response=" +
HttpStatus.getStatusText(status)
               );
         }
      } catch (IOException e) {
         System.err.println("problems with
batchSignalP:"+e.getMessage());
      } catch (InterruptedException ex) {
         ex.printStackTrace();
      }
      return assignments ;

   }
```

```
   /** write an bufferedimage to an output file in JPEG
format **/
   public static void writeImage(String outputName,
BufferedImage bufferedImage) {
      System.out.println("writing jpeg++++ outputName");
      try {
         FileOutputStream out = new
FileOutputStream(outputName);
         JPEGImageEncoder jpegEncoder =
JPEGCodec.createJPEGEncoder(out);
         JPEGEncodeParam encodeParam =
JPEGCodec.getDefaultJPEGEncodeParam(bufferedImage);
         encodeParam.setQuality((float)0.9, true);
         bufferedImage.flush();
         jpegEncoder.encode(bufferedImage, encodeParam);
      } catch (ImageFormatException ex) {
         ex.printStackTrace();
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }
    /** write an bufferedimage to an outputstream in JPEG
format **/
   public static void writeImage(OutputStream out,
BufferedImage bufferedImage) {
      System.out.println("writing jpeg++++");
      try {
         JPEGImageEncoder jpegEncoder =
JPEGCodec.createJPEGEncoder(out);
         JPEGEncodeParam encodeParam =
JPEGCodec.getDefaultJPEGEncodeParam(bufferedImage);
         encodeParam.setQuality((float)0.9, true);
         bufferedImage.flush();
         jpegEncoder.encode(bufferedImage, encodeParam);
      } catch (ImageFormatException ex) {
         ex.printStackTrace();
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }

/** returns the reverse complement for a nucleotide
sequence **/
   public static void  reverseCompletent(String
```

```
nucleotideSequence){
      try {
         //make a DNA SymbolList
         SymbolList symL =
DNATools.createDNA(nucleotideSequence);

         //reverse complement it
         symL = DNATools.reverseComplement(symL);

         //prove that it worked
//         System.out.println(symL.seqString());
      } catch (IllegalSymbolException ex) {
         //this will happen if you try and make the DNA seq
using non IUB symbols
         ex.printStackTrace();
      }catch (IllegalAlphabetException ex) {
         //this will happen if you try and reverse
complement a non DNA sequence using DNATools
         ex.printStackTrace();
      }
   }

/**parse the file, after this the result List will be
populated with
        SeqSimilaritySearchResults**/
   public static List readBLASToutput(String filePath){
      //The list to hold the SeqSimilaritySearchResults
      List results = new ArrayList();
      try {
         BufferedReader in         = new BufferedReader(new
FileReader(filePath));
         //make a BlastLikeSAXParser
         BlastLikeSAXParser    parser   = new
BlastLikeSAXParser();
         parser.setModeLazy();
         //make the SAX event adapter that will pass events
to a Handler.
         SeqSimilarityAdapter  adapter  = new
SeqSimilarityAdapter();
         //set the parsers SAX event adapter
         parser.setContentHandler(adapter);
                        //create the SearchContentHandler
that will build SeqSimilaritySearchResults
         //in the results List
         SearchContentHandler  builder  = new
```

```
BlastLikeSearchBuilder(results,
            new DummySequenceDB("queries"), new
DummySequenceDBInstallation());
         //register builder with adapter
         adapter.setSearchContentHandler(builder);
         //parse the file, after this the result List will
be populated with
         //SeqSimilaritySearchResults
         parser.parse(new InputSource(in));
      } catch (IOException ex) {
         //IO problem, possibly file not found
         ex.printStackTrace();
      } catch (NoSuchElementException ex) {
         //no fasta sequences in the file
         ex.printStackTrace();
      } catch (Exception e) {
         e.printStackTrace();
      }
      return results;
   }

    /**retrieves systematic deletion phenotype (viable or
inviable) as annotated by the SGD website **/
   public static Map getToBviableOrInviable(Set cwpNames){
      Map viability = new HashMap();
      Map cwpViability = new HashMap();
//   try {
      StringBean sb = new StringBean();
      int viable =1;   int inviable=1;   int
not_determined=1;
      for (Iterator<String>it
=cwpNames.iterator();it.hasNext();){
         String cwpName = it.next();
         sb.setURL("http://db.yeastgenome.org/cgi-
bin/locus.pl?locus="+cwpName);
//       sb.setURL("http://db.yeastgenome.org/cgi-
bin/locus.pl?locus=YIR019C");
         String text = sb.getStrings().replaceAll("\\n","
");
         if(text.matches("^.+ viable .+$")){
            System.out.println("matches="+ text);
            viability.put("viable",viable++);
            cwpViability.put(cwpName,"viable");
         }else if (text.matches("^.+ inviable .+$")) {
            System.out.println("no match="+ text);
```

```
            viability.put("inviable",inviable++);
            cwpViability.put(cwpName,"inviable");
         }else {
            System.out.println("problem with viability
determination:"+text);
            viability.put("not determined",not_determined+
+);
            cwpViability.put(cwpName,"not determined");
         }
      }
//    } catch (IOException ex) {
//       ex.printStackTrace();
//    }
      JcTools.serializeObject("cwpViability",cwpViability);
      JcTools.serializeObject("viability",viability);
      System.out.println("cwpViability="+cwpViability.size(
)+cwpViability);
      return viability;
   }


}
```

1.1.1.1.1.1.1 PlotJpeg.java
1.1.1.1.1.1.1.1 the main java class used to generate graphics for website and publications

```
/*
 * PlotJpeg.java
 *
 * Created on April 20, 2005, 1:10 PM
 */

package hunter.cuny.edu;
import ptolemy.plot.*;
import ptolemy.plot.plotml.PlotMLParser;
import com.microstar.xml.XmlException;
import javax.servlet.http.*;
import java.io.*;import java.util.*;

import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;
import com.sun.image.codec.jpeg.JPEGEncodeParam;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.awt.Component;
```

```
import java.awt.Dimension;
import java.awt.Color;
import java.awt.Font;
import org.biojava.bio.symbol.AlphabetManager;
import org.biojava.bio.*;
import org.biojava.bio.seq.db.*;
import org.biojava.bio.seq.io.SeqIOTools;
import org.biojava.bio.symbol.*;
import pal.alignment.*;
import java.util.ResourceBundle;
/**
 *
 * @author  bioinfo
 */
public class PlotJpeg {
    Histogram histPlot ;
    Plot plot;
    /** Creates a new instance of PlotJpeg */
    public PlotJpeg() {
        histPlot = new Histogram();
        histPlot.setColors(_colors);
        plot = new Plot();
        plot.setColors(_colors);

    }


    public Histogram createPlot(List list, String
setName ,String graph){
        // try{
        System.out.println("creating plot "+graph);
        ResourceBundle axes =
ResourceBundle.getBundle("graph.GraphAxes_en_US");
        System.out.println(graph+".title");
        histPlot.setTitle(axes.getString(graph+".title"));
        histPlot.setXLabel(axes.getString(graph+".x"));
        histPlot.setYLabel(axes.getString(graph+".y"));

        histPlot.read("DataSet: "+setName);
        // histPlot.setBinWidth(.05);
        // histPlot.setBinOffset(0);
        //histPlot.setSize(500,400);
        //histPlot.setXLog(false);
        for (int i=0; i < 301; i+=10){
            histPlot.addYTick(String.valueOf(i),(double)i);
```

```
        }
        Iterator listIt = list.iterator();
        while (listIt.hasNext()){
            histPlot.read( String.valueOf(listIt.next()));
        }
        int h = 500;
        int w = 800;
        histPlot.setSize(w,h);

        /*}catch (IOException ex) {

        }*/
        return histPlot;
    }

    public Histogram createPlot(Map map, String
setName,String graph){
        System.out.println("creating plot");
        // try{
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
        histPlot.setTitle(axes.getString(graph+".title"));
        histPlot.setXLabel(axes.getString(graph+".x"));
        histPlot.setYLabel(axes.getString(graph+".y"));
        histPlot.read("DataSet: "+setName);
        //histPlot.setBinWidth(1);
        //histPlot.setBinOffset(0);
        //histPlot.setSize(500,400);
        // histPlot.setXLog(false);
        Iterator keysIt = map.keySet().iterator();
        while (keysIt.hasNext()){
            histPlot.read( String.valueOf(map.get(keysIt.ne
xt())));
        }
        int h = 500;
        int w = 800;
        histPlot.setSize(w,h);


       /* }catch (IOException ex) {

        }*/
        return histPlot;
    }
```

```
    public Plot createPlot2(Map map, String setName,String
graph, int setNum ){
        System.out.println("creating plot");
        // try{
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
     //plot.setLabelFont("Arial-BOLD-18");
        plot.setTitle(axes.getString(graph+".title"));
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));
     plot.setLabelFont("Arial-BOLD-11");
        for (int i=0; i < 1301; i+=100){
            plot.addYTick(String.valueOf(i),(double)i);
        }
     for (int i=0; i < 1501; i+=10){
            plot.addXTick(String.valueOf(i),(double)i);
        }
        Iterator keysIt = map.keySet().iterator();
        // plot.setConnected(true,setNum);
     plot.setMarksStyle("dots");
        while (keysIt.hasNext()){
            Integer length = (Integer)keysIt.next();
            plot.addPoint(setNum, length,
(Integer)map.get(length),false);
        }
        int h = 275;
        int w = 430;
        // plot.addPoint(1,35,100,false);
        plot.setSize(w,h);

        plot.addLegend(setNum,setName);
     //plot.setConnected(true);
        // plot.setBars(true);
        //plot.zoom(20,0,90,1350);
        return plot;
    }
     public Plot createPlot2BW(Map map, String
setName,String graph, int setNum ){
        System.out.println("creating plot");
        // try{
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
     //plot.setLabelFont("Arial-BOLD-18");
        plot.setTitle(axes.getString(graph+".title"));
```

```
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));
     plot.setLabelFont("Arial-BOLD-11");
     plot.setColor(false);
        for (int i=0; i < 1301; i+=100){
            plot.addYTick(String.valueOf(i),(double)i);
        }
     for (int i=0; i < 1501; i+=10){
            plot.addXTick(String.valueOf(i),(double)i);
        }
        Iterator keysIt = map.keySet().iterator();
        // plot.setConnected(true,setNum);
     plot.setMarksStyle("various");
        while (keysIt.hasNext()){
            Integer length = (Integer)keysIt.next();
            plot.addPoint(setNum, length,
(Integer)map.get(length),false);
        }
        int h = 275;
        int w = 430;
        // plot.addPoint(1,35,100,false);
        plot.setSize(w,h);

        plot.addLegend(setNum,setName);
     //plot.setConnected(true);
        // plot.setBars(true);
        //plot.zoom(20,0,90,1350);
        return plot;
    }

    public Plot drawPlot(List list, List setNames ,String
graph){
        // try{
        System.out.println("creating plot "+graph);
        ResourceBundle axes =
ResourceBundle.getBundle("graph.GraphAxes_en_US");
        System.out.println(graph+".title");
        plot.setTitle(axes.getString(graph+".title"));
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));
        for (int i=0; i < 1001; i+=50){
            plot.addYTick(String.valueOf(i),(double)i);
        }
        for(int d=0; d <list.size(); d++){
            plot.addXTick((String)setNames.get(d),(double)
```

```
(d + 1));
            System.out.print(setNames.get(d)+" ");
            plot.addPoint(d, (double)d+1,
Double.valueOf(String.valueOf(list.get(d))), false);
        }
        int h = 500;
        int w = 800;
        plot.setSize(w,h);
        plot.setBars(1,0);
        plot.setXRange(1.0,16.0);
        /*}catch (IOException ex) {

        }*/
        return plot;
    }
    public Plot drawAAdistr(Map<String,Integer> map, String
setName,String graph,
    int dataset, String legend){
        System.out.println("creating plot");
        // try{
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
        plot.setTitle(axes.getString(graph+".title"));
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));

        //
        List aminoAcidList = new ArrayList();
aminoAcidList.addAll(map.keySet());
        Collections.sort(aminoAcidList);
        int x= 0;
        int numOfAA = map.get("numOfAA");
        System.out.println(numOfAA+" numOfAA "+ dataset);
        Iterator <String> keysIt = aminoAcidList.iterator();
        while (keysIt.hasNext()){
            String aminoAcid = keysIt.next();
            if (!aminoAcid.equals("numOfAA")){
                double percentage =
(double)map.get(aminoAcid) / (double)numOfAA ;
                System.out.print(aminoAcid+"&"+percentage
+"%");
                plot.addXTick(aminoAcid,(double)(x + 1));
                plot.addPoint(dataset, (double)x+1,
percentage, false);
                x++;
```

```
            }
        }
        int h = 500;
        int w = 800;
        plot.addLegend(dataset,legend);
        plot.setSize(w,h);

        plot.setBars(.5,.25);

       /* }catch (IOException ex) {

        }*/
        return plot;
    }

    public Plot drawRoc(List xy, String setName,String
graph, int setNum ){
        System.out.println("creating plot");
        // try{
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
        //plot.setTitle(axes.getString(graph+".title"));
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));
        int i=0;
        //plot.setMarksStyle("various");
        while (i < xy.size()){
            Integer x = (Integer)xy.get(i);
            i+=1;
            Integer y = (Integer)xy.get(i);
            plot.addPoint(setNum, x, y,true);
            i+=1;
        }
        //int h = 500;
        //int w = 800;
        // plot.addPoint(1,35,100,false);
        // plot.setSize(w,h);
        //plot.setMarksStyle("points");
        // plot.addLegend(setNum,setName);

        // plot.setBars(true);

        // plot.zoom(0,0,200,50);
        plot.zoom(0,-5,1000,1000);
        return plot;
```

```
    }

    public Plot drawYeastRoc(List xy, String setName,String
graph, int setNum ){
        System.out.println("creating plot");
        // try{
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
        //plot.setTitle(axes.getString(graph+".title"));
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));
        int i=0;
        //plot.setMarksStyle("various");
        while (i < xy.size()){
            Integer x = Integer.valueOf((String)xy.get(i));
            i+=1;
            Integer y = Integer.valueOf((String)xy.get(i));
            plot.addPoint(setNum, x, y,true);
            i+=1;
        }
        //int h = 500;
        //int w = 800;
        // plot.addPoint(1,35,100,false);
        //plot.setSize(w,h);
        //plot.setMarksStyle("points");
        plot.addLegend(setNum,setName);

        // plot.setBars(true);

        // plot.zoom(0,0,200,50);
        plot.zoom(0,0,150,50);
        return plot;
    }


     public Plot drawYeastRocLog(List xy, String
setName,String graph, int setNum ){
        System.out.println("creating plot");
        // try{
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
        //plot.setTitle(axes.getString(graph+".title"));
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));
        int i=0;
```

```
//   plot.setYLog(true);
        //plot.setMarksStyle("various");
        while (i < xy.size()){
            Integer x = Integer.valueOf((String)xy.get(i));
            i+=1;
            Integer y = Integer.valueOf((String)xy.get(i));
         System.out.println(setNum+" "+x+" "+y);
            plot.addPoint(setNum, x, y,true);
            i+=1;
        }
        //int h = 500;
        //int w = 800;
        // plot.addPoint(1,35,100,false);
        //plot.setSize(w,h);
        //plot.setMarksStyle("points");
        plot.addLegend(setNum,setName);

        // plot.setBars(true);

        // plot.zoom(0,0,200,50);
        plot.zoom(0,0,700,4);
        return plot;
    }//drawYeastRocLog

    /**
     * Write the jpeg image of the Component cnt to the
OutPutStream.
     *
     * If the component is a penal with subpanels you might
need to call
     * {@link JPanel#addNotify} and {@link JPanel#doLayout}.
     * If the server is running on Unix you will need to
use java 1.4 and specify
     * -Djava.awt.headless=true on the command Line or for
tomcat:
     *    edit catalina.sh in the $CATALINA_HOME/bin
directory and add a line
     *    CATALINA_OPTS='-Djava.awt.headless=true'
     */
    public void writeAsJPEG2(OutputStream out, Component
cnt) throws IOException
    {  Dimension s = cnt.getSize();
       cnt.doLayout();
       System.out.println("writing jpeg++++");
       BufferedImage bufferedImage =
```

```
       new BufferedImage((int)s.getWidth(),
(int)s.getHeight(),
       BufferedImage.TYPE_3BYTE_BGR );
       Graphics2D graphics = bufferedImage.createGraphics();
       graphics.setColor(Color.white);
       //graphics.fillRect(0, 0, (int)s.getWidth(),
(int)s.getHeight());
       cnt.print(graphics);
         graphics.setFont(new Font("Arial",Font.BOLD,17));
       graphics.drawString("Srch-Mat  Gap",468,60);
       graphics.setFont(new Font("Arial",Font.BOLD,20));
       graphics.drawString("A",50,50);
       graphics.drawString("B",55,450);
       // graphics.drawString("Srch-Mat  Gap",444,335);
       JPEGImageEncoder jpegEncoder =
JPEGCodec.createJPEGEncoder(out);
       JPEGEncodeParam encodeParam =
JPEGCodec.getDefaultJPEGEncodeParam(bufferedImage);
       encodeParam.setQuality((float)0.9, true);
       jpegEncoder.encode(bufferedImage, encodeParam);
       graphics.dispose();
       bufferedImage.flush();
    }

    public void writeAsJPEG(OutputStream out, Component
cnt) throws IOException
    {  Dimension s = cnt.getSize();
       //cnt.doLayout();
       System.out.println("writing jpeg++++");
       BufferedImage bufferedImage =
       new BufferedImage((int)s.getWidth(),
(int)s.getHeight(),
       BufferedImage.TYPE_3BYTE_BGR );
       Graphics2D graphics = bufferedImage.createGraphics();
       //graphics.setColor(Color.white);
       //graphics.fillRect(0, 0, (int)s.getWidth(),
(int)s.getHeight());
       cnt.print(graphics);
        // graphics.setFont(new Font("Arial",Font.BOLD,17));
       //graphics.drawString("Srch-Mat  Gap",468,60);
       //graphics.setFont(new Font("Arial",Font.BOLD,20));
       //graphics.drawString("A",50,50);
       //graphics.drawString("B",55,450);
       // graphics.drawString("Srch-Mat  Gap",444,335);
       JPEGImageEncoder jpegEncoder =
```

```
JPEGCodec.createJPEGEncoder(out);
       JPEGEncodeParam encodeParam =
JPEGCodec.getDefaultJPEGEncodeParam(bufferedImage);
       encodeParam.setQuality((float)0.9, true);
       jpegEncoder.encode(bufferedImage, encodeParam);
       graphics.dispose();
      bufferedImage.flush();
    }


    /**
     * Parse PlotML file into plot.
     */
    public void read(Plot plot, InputStream in) throws
IOException {
        PlotMLParser parser;
        parser = new PlotMLParser(plot);
        try {
            parser.parse(null, in);
        } catch (Exception ex) {
            String msg;
            if (ex instanceof XmlException)
            {   XmlException xmlex = (XmlException)ex;
                msg =
                "PlotMLFrame: failed to parse PlotML
data:\n"
                + "line: " + xmlex.getLine()
                + ", column: " + xmlex.getColumn()
                + "\nIn entity: " + xmlex.getSystemId()
                + "\n";
            } else
            {   msg = "PlotMLFrame: failed to parse PlotML
data:\n";
            }
            throw new IOException(msg);
        }
    }
    public Histogram createHistogram(HttpServletRequest
request,
    HttpServletResponse response, String graph, String
cutDir){
        System.out.println("creating Histogram ***");
        /*String urlString =
request.getRequestURL().toString()
        .replaceAll("plotServlet\\.do.*
```

```
$","").replaceFirst("^http://","");*/
        String urlString = "/opt/tomcat/webapps/modmat/";
        System.out.println("urlString= "+urlString);
        try {
            File decompDir = new File(urlString
+"Decomposition"+File.separator+cutDir);
            FilenameFilter  filter     =new
FilenameFilter() {
                //name.endsWith(".nxs")
                public boolean accept(File fdir, String
name) {
                    return (name.endsWith(".nxs")  );
                }

            };

            String[] clustalFiles = decompDir.list(filter);
            Map sizeOfEachMember = new HashMap();
            List numberOfMembers = new ArrayList();
            List famAlignmentLength = new ArrayList();

           /* if (graph.matches("numOfMembs")|
graph.matches("fragSize")|
            graph.matches("famSize"))  {
                // use PAL library to read clustalw
alignments of Fragment Families */
            for (int c=0; c < clustalFiles.length;c++ ){
                // System.out.println(clustalFiles[c]+"
name of alignment file");
                ReadAlignment clustal =
                new ReadAlignment(decompDir.toString()
+File.separator+clustalFiles[c]);
                //char[][] clustalMatrix =
clustal.getData();
                int idCount = clustal.getIdCount();
                numberOfMembers.add(idCount);
                int lengthAlignment = clustal.getLength();
                famAlignmentLength.add(lengthAlignment);
                //List fragFamily = new ArrayList();
                for (int i = 0; i < idCount; i++) {
                    String alignSeq =
clustal.getAlignedSequenceString(i);
                    /* replace gaps with nothing */
                    alignSeq    =
alignSeq.replaceAll("-","");
```

```
                    int len = alignSeq.length();
                    //sizeOfEachMember.add(len);
                    sizeOfEachMember.put(clustal.getIdentif
ier(i).getName(),len);
                    //
System.out.println(clustal.getIdentifier(i).getName()+" sEM
"+alignSeq+" sEM "+len);
                }
            }
            if (graph.matches("numOfMembs")){
                histPlot =
createPlot(numberOfMembers,"numOfMembs", graph );
            }else if (graph.matches("famSize")){
                histPlot =
createPlot(famAlignmentLength,"famSize",graph);
            }else if (graph.matches("fragSize")){
                histPlot =
createPlot(sizeOfEachMember,"fragSize",graph);
            }else if (graph.matches("coverage"))  {
                List famAlignmentCoverage = new ArrayList();
                File                dbFile2  = new File
                ("/public/dbase/cerevisiae.aa");
                BufferedInputStream indb = new
BufferedInputStream(dbFile2.toURL().openStream());
                Alphabet  alpha  = new
AlphabetManager().alphabetForName("PROTEIN-TERM");
                SequenceDB hashSeqDB =
SeqIOTools.readFasta(indb, alpha);
                BufferedReader  bf           = new
BufferedReader(
                new
FileReader("/public/sets/GO.GPI.total.gns"));
                String          geneName;
                List            proteinNames  = new
ArrayList();
                while ((geneName = bf.readLine()) != null) {
                    proteinNames.add(geneName);
                }
                Iterator it =proteinNames.iterator();
                while (it.hasNext()){
                    String gene = (String)it.next();
                    Iterator sEM
=sizeOfEachMember.keySet().iterator();
                    int geneLength =
```

```
hashSeqDB.getSequence(gene).length();
                    int  coverageLength = 0;
                    System.out.println(gene +" "+cutDir);
                    while(sEM.hasNext()){
                        String fragmentName =
(String)sEM.next();
                        //System.out.println(fragmentName+"
"+ sizeOfEachMember.get(fragmentName));
                        if (fragmentName.matches("^"+gene
+"_.+$")){
                            //coverageLength +=
(Integer.valueOf((String)sizeOfEachMember.get(fragmentName)
)).intValue();
                            coverageLength +=
((Integer)sizeOfEachMember.get(fragmentName)).intValue();
                            //
System.out.println(coverageLength);
                        }
                    }
                    System.out.println(gene
+"**"+coverageLength+
                    "*" + geneLength + "****"+
(((double)coverageLength)/((double)geneLength))*100.0);
                    famAlignmentCoverage.add(Math.round(((
double)coverageLength)/((double)geneLength))*100.0));
                }
                histPlot =
createPlot(famAlignmentCoverage,"Coverage",graph);
            }

        }catch (AlignmentParseException ape){
            System.out.println("error in parsing Alignment:
"+ ape);
        }catch (IOException e) {
            System.out.println("io problem: "+ e);
        }catch (BioException ex) {
            //not in fasta format or wrong alphabet
            ex.printStackTrace();
        }
        return histPlot;
    }

    public Histogram createHistogram(String graph){
        System.out.println("creating Histogram ***");
        /*String urlString =
```

```
request.getRequestURL().toString()
        .replaceAll("plotServlet\\.do.*
$","").replaceFirst("^http://","");*/
        String urlString =
"/opt/tomcat/webapps/modmat/Decomposition";
        System.out.println("urlString= "+urlString);
        try {
            File dir = new File(urlString);
            FilenameFilter  cutFilter     =new
FilenameFilter() {
                //name.endsWith(".nxs")
                public boolean accept(File fdir, String
name) {
                    return (name.startsWith("cut")  );
                }

            };
            String[] cutDirs = dir.list(cutFilter);
            for (int j =0; j < cutDirs.length;j++){
                File decompDir = new File(urlString
+File.separator+cutDirs[j]);
                FilenameFilter  filter     =new
FilenameFilter() {
                    //name.endsWith(".nxs")
                    public boolean accept(File fdir, String
name) {
                        return (name.endsWith(".nxs")  );
                    }

                };

                String[] clustalFiles =
decompDir.list(filter);
                Map sizeOfEachMember = new HashMap();
                List numberOfMembers = new ArrayList();
                List famAlignmentLength = new ArrayList();

           /* if (graph.matches("numOfMembs")|
graph.matches("fragSize")|
            graph.matches("famSize"))  {
                // use PAL library to read clustalw/nexus
alignments of Fragment Families */
                for (int c=0; c < clustalFiles.length;c++ ){
                    // System.out.println(clustalFiles[c]+"
name of alignment file");
```

```
                    ReadAlignment clustal =
                    new ReadAlignment(decompDir.toString()
+File.separator+clustalFiles[c]);
                    //char[][] clustalMatrix =
clustal.getData();
                    int idCount = clustal.getIdCount();
                    numberOfMembers.add(idCount);
                    int lengthAlignment =
clustal.getLength();
                    famAlignmentLength.add(lengthAlignment);
                    //List fragFamily = new ArrayList();
                    for (int i = 0; i < idCount; i++) {
                        String alignSeq =
clustal.getAlignedSequenceString(i);
                        /* replace gaps with nothing */
                        alignSeq    =
alignSeq.replaceAll("-","");

                        int len = alignSeq.length();
                        //sizeOfEachMember.add(len);
                        sizeOfEachMember.put(clustal.getIde
ntifier(i).getName(),len);
                        //
System.out.println(clustal.getIdentifier(i).getName()+" sEM
"+alignSeq+" sEM "+len);
                    }
                }
                if (graph.matches("numOfMembs")){
                    histPlot =
createPlot(numberOfMembers,cutDirs[j],graph);

                }else if (graph.matches("famSize")){
                    histPlot =
createPlot(famAlignmentLength,cutDirs[j],graph);
                }else if (graph.matches("fragSize")){
                    histPlot =
createPlot(sizeOfEachMember,cutDirs[j],graph);
                }else if (graph.matches("coverage"))  {
                    List famAlignmentCoverage = new
ArrayList();
                    File                dbFile2  = new File
                    ("/public/dbase/cerevisiae.aa");
                    BufferedInputStream indb = new
BufferedInputStream(dbFile2.toURL().openStream());
                    Alphabet  alpha  = new
```

```
AlphabetManager().alphabetForName("PROTEIN-TERM");
                    SequenceDB hashSeqDB =
SeqIOTools.readFasta(indb, alpha);
                    BufferedReader  bf            = new
BufferedReader(
                    new
FileReader("/public/sets/GO.GPI.total.gns"));
                    String          geneName;
                    List            proteinNames  = new
ArrayList();
                    while ((geneName = bf.readLine()) !=
null) {
                        proteinNames.add(geneName);
                    }
                    Iterator it =proteinNames.iterator();
                    while (it.hasNext()){
                        String gene = (String)it.next();
                        Iterator sEM
=sizeOfEachMember.keySet().iterator();
                        int geneLength =
hashSeqDB.getSequence(gene).length();
                        int  coverageLength = 0;
                        System.out.println(gene +"
"+cutDirs[j]);
                        while(sEM.hasNext()){
                            String fragmentName =
(String)sEM.next();
                            //System.out.println(fragmentNa
me+" "+ sizeOfEachMember.get(fragmentName));
                            if
(fragmentName.matches("^"+gene+"_.+$")){
                                //coverageLength +=
(Integer.valueOf((String)sizeOfEachMember.get(fragmentName)
)).intValue();
                                coverageLength +=
((Integer)sizeOfEachMember.get(fragmentName)).intValue();
                                //
System.out.println(coverageLength);
                            }
                        }
                        //
System.out.println(gene+"**"+coverageLength+
                        //                          "*" +
geneLength + "****"+(((double)coverageLength)/
((double)geneLength))*100.0);
```

```
                        if ((((double)coverageLength)/
((double)geneLength))*100.0 > 100.0){
                            System.out.println(gene+"
greater than 100");
                        }
                        famAlignmentCoverage.add(Math.round
((((double)coverageLength)/((double)geneLength))*100.0));
                    }
                    histPlot =
createPlot(famAlignmentCoverage,cutDirs[j], graph);
                }
            }// for j
        }catch (AlignmentParseException ape){
            System.out.println("error in parsing Alignment:
"+ ape);
        }catch (IOException e) {
            System.out.println("io problem: "+ e);
        }catch (BioException ex) {
            //not in fasta format or wrong alphabet
            ex.printStackTrace();
        }
        return histPlot;

    }

  /*public Plot createData(String graph){
        System.out.println("creating Plot ***");

        String urlString =
"/opt/tomcat/webapps/modmat/Decomposition";
        System.out.println("urlString= "+urlString);
        Map lengthVsNumOfMembs = new HashMap();
        try {
            File dir = new File(urlString);
            FilenameFilter  cutFilter     =new
FilenameFilter() {
                //name.endsWith(".nxs")
                public boolean accept(File fdir, String
name) {
                    return (name.startsWith("cut")  );
                }

            };
            String[] cutDirs = dir.list(cutFilter);
            for (int j =0; j < cutDirs.length;j++){
```

```
                File decompDir = new File(urlString
+File.separator+cutDirs[j]);
                FilenameFilter  filter     =new
FilenameFilter() {
                    //name.endsWith(".nxs")
                    public boolean accept(File fdir, String
name) {
                        return (name.endsWith(".nxs")  );
                    }

                };

                String[] clustalFiles =
decompDir.list(filter);


                // use PAL library to read clustalw/nexus
alignments of Fragment Families
                for (int c=0; c < clustalFiles.length;c++ ){
                    // System.out.println(clustalFiles[c]+"
name of alignment file");
                    ReadAlignment clustal =
                    new ReadAlignment(decompDir.toString()
+File.separator+clustalFiles[c]);

                    int lengthAlignment =
clustal.getLength();
                    int idCount = clustal.getIdCount();

                    if
(lengthVsNumOfMembs.get(lengthAlignment) == null){
                        //* initialize
                        lengthVsNumOfMembs.put(lengthAlignm
ent,idCount);
                    }else{
                        // take the number fragments seen
so far and add new idCount
                        int sum =
(Integer)lengthVsNumOfMembs.get(lengthAlignment);
                        lengthVsNumOfMembs.put(lengthAlignm
ent,idCount + sum);
                    }

                }// for each alignment file in this cut
directory
```

```
            }// for each cut directory
        }catch (AlignmentParseException ape){
            System.out.println("error in parsing Alignment:
"+ ape);
        }catch (IOException e) {
            System.out.println("io problem: "+ e);
        }
        if (graph.matches("lengthNumOfMembs")){
            plot =
createPlot2(lengthVsNumOfMembs,graph,graph);
        }
        return plot;

    } */

  /* public Plot createData(String graph, String cutDir){
        System.out.println("creating Plot ***");
        String urlString =
"/opt/tomcat/webapps/modmat/Decomposition";
        System.out.println("urlString= "+urlString);
        Map lengthVsNumOfMembs = new HashMap();
        try {
            File decompDir = new File(urlString
+File.separator+cutDir);
            FilenameFilter  filter     =new
FilenameFilter() {
                //name.endsWith(".nxs")
                public boolean accept(File fdir, String
name) {
                    return (name.endsWith(".nxs")  );
                }

            };

            String[] clustalFiles = decompDir.list(filter);


            // use PAL library to read clustalw/nexus
alignments of Fragment Families
            for (int c=0; c < clustalFiles.length;c++ ){
                // System.out.println(clustalFiles[c]+"
name of alignment file");
                ReadAlignment clustal =
```

```
                new ReadAlignment(decompDir.toString()
+File.separator+clustalFiles[c]);

                int lengthAlignment = clustal.getLength();
                int idCount = clustal.getIdCount();

                if (lengthVsNumOfMembs.get(lengthAlignment)
== null){
                    // initialize
                    lengthVsNumOfMembs.put(lengthAlignment,
idCount);
                }else{
                    // take the number fragments seen so
far and add the new idCount
                    int sum =
(Integer)lengthVsNumOfMembs.get(lengthAlignment);
                    lengthVsNumOfMembs.put(lengthAlignment,
idCount + sum);
                }

            }// for each alignment file in this cut
directory



        }catch (AlignmentParseException ape){
            System.out.println("error in parsing Alignment:
"+ ape);
        }catch (IOException e) {
            System.out.println("io problem: "+ e);
        }
        if (graph.matches("lengthNumOfMembs")){
            plot =
createPlot2(lengthVsNumOfMembs,graph,graph);
        }
        return plot;

    }*/

    /* generate a Map with the alignment file as key and
the percent of gaps in overhang regions of
     fragments in the alignment; to be used to assess the
degree of fragment definition by our
     simple HSP-based cutting algorithm */
    public Plot createOverHangPlot(String graph, String
```

```
cutDir){
        System.out.println("creating Plot ***");
        String urlString =
"/opt/tomcat/webapps/modmat/Decomposition";
        System.out.println("urlString= "+urlString+ cutDir);
        Map <String, Double>nxs_gaps = new HashMap();
        try {
            File decompDir = new File(urlString
+File.separator+cutDir);
            FilenameFilter  filter     =new
FilenameFilter() {
                //name.endsWith(".nxs")
                public boolean accept(File fdir, String
name) {
                    return (name.endsWith(".nxs")  );
                }

            };
            String[] clustalFiles = decompDir.list(filter);
            // use PAL library to read clustalw/nexus
alignments of Fragment Families */
            for (int c=0; c < clustalFiles.length;c++ ){
                // System.out.println(clustalFiles[c]+"
name of alignment file");
                ReadAlignment clustal =
                new ReadAlignment(decompDir.toString()
+File.separator+clustalFiles[c]);
                int lengthAlignment = clustal.getLength();
                int idCount = clustal.getIdCount();
                int overhangGaps = 0;
                System.out.println(clustalFiles[c]);
                /* for each sequence */
                for(int i=0; i <idCount; i++){
                    /*going from left to right, count all
gaps before first Amino Acid */
                    for(int f=0; f <lengthAlignment; f++){
                        if (clustal.isGap(i,f)){
                            System.out.print(clustal.getDat
a(i,f));
                            overhangGaps++;
                        }else{
                            System.out.println(clustal.getD
ata(i,f));
                            break;
                        }
```

```
                    }
                    /*going from right to left, count all
gaps before first Amino Acid */
                    for(int b=lengthAlignment-1; b >= 0;
b--){
                        if (clustal.isGap(i,b)){
                            System.out.print(clustal.getDat
a(i,b));
                            overhangGaps++;
                        }else{
                            System.out.println(clustal.getD
ata(i,b));
                            break;
                        }
                    }
                }
                double percent = (double)overhangGaps /
(double)(idCount*lengthAlignment);
                nxs_gaps.put(clustalFiles[c],percent);
            }// for each alignment file in this cut
directory
        }catch (AlignmentParseException ape){
            System.out.println("error in parsing Alignment:
"+ ape);
        }catch (IOException e) {
            System.out.println("io problem: "+ e);
        }
        // System.out.println(nxs_gaps);
        ResourceBundle axes =
ResourceBundle.getBundle( "graph.GraphAxes_en_US" );
        plot.setTitle(axes.getString(graph+".title"));
        plot.setXLabel(axes.getString(graph+".x"));
        plot.setYLabel(axes.getString(graph+".y"));

        Iterator <String>keysIt =
nxs_gaps.keySet().iterator();
        double aln =1;
        while (keysIt.hasNext()){
            String alnName = keysIt.next();
            plot.addPoint(0, aln,
nxs_gaps.get(alnName),true);
            aln++;
        }
        int h = 500;
        int w = 800;
```

```
        plot.setSize(w,h);
        //plot.setBars(true);

        return plot;

    }

    static protected Color[] _colors = {
        Color.BLUE,   // blue 1
        new Color(128,42,42),   // brown 2
        new Color(253,3,252),  // purple 3
        Color.RED , //red 4
        Color.GREEN,//green (blast-pgp from yellow) 5
        new Color(222,184,135), //burlywood 6
        new Color( 210,105,30 ),   // chocolate 7
        new Color(255,99,71),   // tomato 8
        new Color(255,20,147),   // deep pink 9
        new Color(250,128,114 ),   // salmon 10
        new Color(0,199,140), //turquoise 11

        new Color(0xff7f50),   // coral 12
        new Color(0x45ab1f),   // dark green-ish 13
        new Color(0x90422d),   // sienna-ish 14
        new Color(0xa0a0a0),   // grey-ish 15
        new Color(0x14ff14),   // green-ish 16
        new Color(184,201,197),   // blue-ish green 17

        Color.yellow,
        Color.DARK_GRAY,
        Color.PINK,
        new Color(123,104,238), //MediumSlateBlue
        new Color(0,0,139), //blue4
        new Color(34,139,34), //ForestGreen

    };

}
```

TabDelimitBlastOut.java

This class was used to parse the BLAST output and generate a tab-limited version to import into the database.

```
package hunter.cuny.edu;
import java.io.*;
import java.util.*;
```

```
import java.net.*;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.biojava.bio.*;
import org.biojava.bio.program.sax.*;
import org.biojava.bio.program.ssbind.*;
import org.biojava.bio.search.*;
import org.biojava.bio.seq.db.*;
import org.xml.sax.*;
//import org.jdom.*;
/**
 *  Description of the Class
 *
 * @author     bioinfo
 * @created    October 26, 2004
 */
public class TabDelimitBlastOut {

    private  List  results;
    private  URL   url2;
    private File blastout;
    public static double green =  0.99;
    public static double yellow = 0.74;
    public static double orange =  0.49;
    public static double red =  0.24;
    public Map entropyValues, kValues, lambda;
    /**
     *Constructor for the TabDelimitBlastOut object
     *
     * @param  blastout  Description of the Parameter
     */

    public TabDelimitBlastOut() {}

    public TabDelimitBlastOut(File blastout) {
     this.blastout = blastout;
     System.out.println(blastout);
     try {

         BufferedReader in         = new BufferedReader(new
FileReader(blastout));

         //make a BlastLikeSAXParser
         BlastLikeSAXParser    parser   = new
BlastLikeSAXParser();
```

```
        parser.setModeLazy();

        //make the SAX event adapter that will pass events
to a Handler.
        SeqSimilarityAdapter  adapter  = new
SeqSimilarityAdapter();

        //set the parsers SAX event adapter
        parser.setContentHandler(adapter);

        //The list to hold the SeqSimilaritySearchResults
        results = new ArrayList();

        //create the SearchContentHandler that will build
SeqSimilaritySearchResults
        //in the results List
        SearchContentHandler  builder  = new
BlastLikeSearchBuilder(results,
        new DummySequenceDB("queries"), new
DummySequenceDBInstallation());

        //register builder with adapter
        adapter.setSearchContentHandler(builder);

        //parse the file, after this the result List will
be populated with
        //SeqSimilaritySearchResults
        parser.parse(new InputSource(in));
     } catch (IOException ex) {
        //IO problem, possibly file not found
        ex.printStackTrace();
     } catch (NoSuchElementException ex) {
        //no fasta sequences in the file
        ex.printStackTrace();
     } catch (Exception e) {
        e.printStackTrace();
     }
    }
    /**
     *  Description of the Method
     */
    void tabDelimitHspsFile() {
     //list the hits
     try{
        /* write out a file with tad delimited fields to be
```

```
inputted into db;
           the file contains all hsps from the blast output
*/
         BufferedWriter bw = new BufferedWriter(new
FileWriter(blastout.toString()+"_tdHsps"));
         int  hsp_id  = 0;
         for (Iterator i = results.iterator();
i.hasNext(); ) {
          try {
              SeqSimilaritySearchResult  result  =
(SeqSimilaritySearchResult) i.next();
              Annotation                 anno    =
result.getAnnotation();
              for (Iterator k = result.getHits().iterator();
k.hasNext(); ) {
               SeqSimilaritySearchHit  hit  =
(SeqSimilaritySearchHit) k.next();
                for (Iterator l =
hit.getSubHits().iterator(); l.hasNext(); ) {
                   SimpleSeqSimilaritySearchSubHit  subHit
= (SimpleSeqSimilaritySearchSubHit) l.next();
                   String                           queryId
= (String) anno.getProperty("queryId");
                   String
subjectId  = hit.getSubjectID();
                   if (!subjectId.matches(queryId)) {
                    bw.write(hsp_id + "\t");
                    bw.write(queryId + "\t");
                    bw.write(subjectId + "\t");
                    bw.write(subHit.getQueryStart() + "\t");
                    bw.write(subHit.getQueryEnd() + "\t");
                    bw.write(subHit.getSubjectStart() +
"\t");
                    bw.write(subHit.getSubjectEnd() + "\t");
                    bw.write(subHit.getScore()+"\t");
                    bw.write(subHit.getEValue()+"\n");

                                    /* for (Iterator m =
subHit.getAlignment().getLabels().iterator();
m.hasNext(); ) {
                                        Object      label
= (Object) m.next();
                                        //System.out.print(l
abel+ "\t");
                                        SymbolList
```

```
symbolList  =
subHit.getAlignment().symbolListForLabel(label);

     System.out.print(symbolList.seqString() + "\t");

                                    } */
                    hsp_id += 1;
                    //System.out.print("\n");
                   }
                 }
                }
           } catch (NoSuchElementException ex) {
               //no fasta sequences in the file
               ex.printStackTrace();
           } catch (Exception e) {
               e.printStackTrace();
           }
          }
          bw.close();
     } catch (IOException e) {
     }
    }
    void tabDelimitHsps(String tableName) {
     //list the hits
     entropyValues = new HashMap();
     kValues = new HashMap();
     lambda =new HashMap();
     GetDatabase        db          = new GetDatabase();
         /* write out a file with tad delimited fields to be
inputted into db;
           the file contains all hsps from the blast output
*/

     int  hsp_id  = 0;
     for (Iterator i = results.iterator(); i.hasNext(); ) {
         try {
          SeqSimilaritySearchResult  result  =
(SeqSimilaritySearchResult) i.next();
          Annotation                 anno    =
result.getAnnotation();
          entropyValues.put(anno.getProperty("queryId"),
          anno.getProperty("H"));
          kValues.put(anno.getProperty("queryId"),
          anno.getProperty("K"));
          lambda.put(anno.getProperty("queryId"),
```

```
          anno.getProperty("Lambda"));
          for (Iterator k = result.getHits().iterator();
k.hasNext(); ) {
              SeqSimilaritySearchHit  hit  =
(SeqSimilaritySearchHit) k.next();
              for (Iterator l = hit.getSubHits().iterator();
l.hasNext(); ) {
               SimpleSeqSimilaritySearchSubHit  subHit     =
(SimpleSeqSimilaritySearchSubHit) l.next();
               String                           queryId    =
(String) anno.getProperty("queryId");
               String                           subjectId  =
hit.getSubjectID();
               if (!subjectId.matches(queryId)) {
                   /*System.out.println("INSERT INTO
"+tableName+" VALUES " +
                   "("+hsp_id+",'"+ queryId+"','"+ subjectId
+"',"+subHit.getQueryStart()+","
                   +subHit.getQueryEnd()
+","+subHit.getSubjectStart()+","
                   +subHit.getSubjectEnd()
+","+subHit.getScore()+","
                   + subHit.getEValue()+")");*/
                   db.xecuteUpdate("INSERT INTO "+tableName
+" VALUES " +
                   "("+hsp_id+",'"+ queryId+"','"+ subjectId
+"',"+subHit.getQueryStart()+","
                   +subHit.getQueryEnd()
+","+subHit.getSubjectStart()+","
                   +subHit.getSubjectEnd()
+","+subHit.getScore()+",'"
                   + String.valueOf(subHit.getEValue())
+"')");


                                  /* for (Iterator m =
subHit.getAlignment().getLabels().iterator();
m.hasNext(); ) {
                                       Object      label
= (Object) m.next();
                                       //System.out.print(l
abel+ "\t");
                                       SymbolList
symbolList  =
subHit.getAlignment().symbolListForLabel(label);
```

```
     System.out.print(symbolList.seqString() + "\t");

                                    } */
                    hsp_id += 1;
                    //System.out.print("\n");
                 }
                }

          }
         } catch (NoSuchElementException ex) {
          //no fasta sequences in the file
          ex.printStackTrace();
         } catch (Exception e) {
          e.printStackTrace();
         }
     }



    }


    void tabDelimitHsps(String tableName, String
modification, int querySpecies, int subjectSpecies) {
     //list the hits
     entropyValues = new HashMap();
     kValues = new HashMap();
     lambda =new HashMap();
     GetDatabase        db          = new GetDatabase();
         /* write out a file with tad delimited fields to be
inputted into db;
           the file contains all hsps from the blast output
*/
     int  hsp_id  = 0;
     for (Iterator i = results.iterator(); i.hasNext(); ) {
         try {
          SeqSimilaritySearchResult  result  =
(SeqSimilaritySearchResult) i.next();
          Annotation                 anno    =
result.getAnnotation();
          entropyValues.put(anno.getProperty("queryId"),
          anno.getProperty("H"));
          kValues.put(anno.getProperty("queryId"),
          anno.getProperty("K"));
```

```
          lambda.put(anno.getProperty("queryId"),
          anno.getProperty("Lambda"));
          for (Iterator k = result.getHits().iterator();
k.hasNext(); ) {
              SeqSimilaritySearchHit  hit  =
(SeqSimilaritySearchHit) k.next();
              for (Iterator l = hit.getSubHits().iterator();
l.hasNext(); ) {
               SimpleSeqSimilaritySearchSubHit  subHit     =
(SimpleSeqSimilaritySearchSubHit) l.next();
               String                           queryId    =
(String) anno.getProperty("queryId");
               String                           subjectId  =
hit.getSubjectID();
               if (!subjectId.matches(queryId)) {
                   /*System.out.println("INSERT INTO
"+tableName+" VALUES " +
                   "("+hsp_id+",'"+ queryId+"','"+ subjectId
+"',"+subHit.getQueryStart()+","
                   +subHit.getQueryEnd()
+","+subHit.getSubjectStart()+","
                   +subHit.getSubjectEnd()
+","+subHit.getScore()+","
                   + subHit.getEValue()+")");*/
                   db.xecuteUpdate("INSERT INTO "+tableName
+" VALUES " +
                   "("+hsp_id+",'"+ queryId+"','"+ subjectId
+"',"+subHit.getQueryStart()+","
                   +subHit.getQueryEnd()
+","+subHit.getSubjectStart()+","
                   +subHit.getSubjectEnd()
+","+subHit.getScore()+",'"
                   + String.valueOf(subHit.getEValue())
+"'"+","+querySpecies+","+subjectSpecies+",'"+
                   modification+"')");
                   /* for (Iterator m =
subHit.getAlignment().getLabels().iterator();
m.hasNext(); ) {
                     Object      label       = (Object)
m.next();
                     //System.out.print(label+ "\t");
                      SymbolList  symbolList  =
subHit.getAlignment().symbolListForLabel(label);

System.out.print(symbolList.seqString() + "\t");
```

```
                    } */
                   hsp_id += 1;
                   //System.out.print("\n");
                 }
                }

          }
         } catch (NoSuchElementException ex) {
          //no fasta sequences in the file
          ex.printStackTrace();
         } catch (Exception e) {
          e.printStackTrace();
         }
     }



    }



    public List getBioverseNet(int hits){
     List homologyNet = new ArrayList();
     //XmlWriter xmlWriter = new XmlWriter();

     for (Iterator i = results.iterator(); i.hasNext(); ) {

         SeqSimilaritySearchResult  res  =
(SeqSimilaritySearchResult) i.next();
         Annotation                 anno    =
res.getAnnotation();
         List resHits = res.getHits();
         String queryId    = (String)
anno.getProperty("queryId");

         /* if the query contains equal to or more (int)hits
*/
         if (resHits.size() >= hits){
          for (Iterator k = resHits.iterator(); k.hasNext();
) {
              SeqSimilaritySearchHit  hit  =
(SeqSimilaritySearchHit) k.next();
              String  subjectId  = hit.getSubjectID();
              if (!subjectId.matches(queryId)) {
               double evalue = hit.getEValue();
```

```
                double confidence = 0;
                if (evalue <= 1e-20){
                    confidence = green;
                }else if(evalue <= 1e-10) {
                    confidence = yellow;
                }else if (evalue <=1e-6){
                    confidence = orange;
                }else if (evalue <= 1e-3){
                    confidence = red;
                }
                // System.out.println(queryId+"-"+subjectId
+"#"+confidence);
                homologyNet.add(queryId);
                homologyNet.add(subjectId);
                homologyNet.add(confidence);
               }
            }
           }
     }
     // return((String[])homologyNet.toArray(new
String[homologyNet.size()]));
     return homologyNet;
    }
    public Map getEntropyValues(){
     return entropyValues;
    }
    public Map getKValues(){
     return kValues;
    }
    public Map getLambdas(){
     return lambda;
    }
    /**
     *  The main program for the TabDelimitBlastOut class
     *
     * @param  args             The command line arguments
     * @exception  IOException  Description of the Exception
     */
    public static void main(String[] args) throws
IOException {
     //URL blastout = new URL(args[0]);
     TabDelimitBlastOut  tdBO  = new TabDelimitBlastOut(new
File(args[0]));
     tdBO.tabDelimitHspsFile();
```

```
    }
    // main

    void simpleParseOfFasta
    (String file, String tableName, String modification,
int querySpecies, int subjectSpecies) {
     GetDatabase  db  = new GetDatabase();
     try {
         BufferedReader br = new BufferedReader(new
FileReader(file));
         String line = null;
         Pattern query =  Pattern.compile("^\\s+\\d+>>>([a-
zA-Z_0-9:\\-\\.\\|]+)\\s.+$");
         Pattern subject =  Pattern.compile("^>>([a-zA-
Z_0-9:\\-\\.\\|]+)\\s.+$");
         String queryId = null; String subjectId =
null;String evalue = null;
         // query start, query end, subject start, subject
end , score
         String qs; String qe; String ss; String se;String
score ;
         int hsp_id =1;
         while( (line = br.readLine()) != null ){
          Matcher match = query.matcher(line);
          Matcher match2 = subject.matcher(line);
          //System.out.println(line);
          if ( match.matches() ){
             System.out.println(line);
              queryId = match.group(1);
          }
          if ( match2.matches()){
              subjectId = match2.group(1);
              //first line after the line with the subject
id
              System.out.println(line);
              line =br.readLine();
              System.out.println(line);
              String[] first = line.split(" +");
              System.out.println(Arrays.asList(first));
              score = first[10]; evalue = first[12];
              // second line after the line with the subject
id
               line =br.readLine();
              System.out.println(line);
              String[] startEnds = line.replaceAll("[\\
```

```
(\\)]","").split(" +")[11].split(":");
                System.out.println(Arrays.asList(startEnds));
                qs = startEnds[0].replaceAll("-\\d+","");
                qe= startEnds[0].replaceAll("^\\d+-","");
                ss =startEnds[1].replaceAll("-\\d+","");
                se =startEnds[1].replaceAll("^\\d+-","");
                db.xecuteUpdate("INSERT INTO "+tableName+"
VALUES " +
                "("+hsp_id+",'"+ queryId+"','"+ subjectId
+"',"+qs+","
                +qe+","+ss+","+se+","+score+",'"
                + evalue+"'"+","+querySpecies
+","+subjectSpecies+",'"+
                modification+"')");
                /*System.out.println("INSERT INTO "+tableName
+" VALUES " +
                "("+hsp_id+",'"+ queryId+"','"+ subjectId
+"',"+qs+","
                +qe+","+ss+"," +se+","+score+",'"
                + evalue+"'"+","+querySpecies
+","+subjectSpecies+",'"+
                modification+"')");*/
                hsp_id++;
            }


          }

      } catch (FileNotFoundException ex) {
          ex.printStackTrace();
      }catch (IOException ex) {
          ex.printStackTrace();
      }
     }

}
// class TabDelimitBlastOut
```

Shell Scripts

The fungal cell wall comparisons were done using these shell scripts.

```
#!/bin/bash
file=$1
#database=/home/juan/blastDB/orf_trans.fasta
```

```
#database=/export/home/juan/blastDB/paradoxus.aa
#database=/export/home/juan/blastDB/paradoxus.2.aa
#database=/export/home/juan/blastDB/bayanus.aa
#database=/export/home/juan/blastDB/mikatae.aa
#database=/export/home/juan/blastDB/castelli.aa
#database=/export/home/juan/blastDB/kudriavzevii.aa
#database=/export/home/juan/blastDB/kluyveri.aa
#database=/export/home/juan/blastDB/Calbicans.aa
#database=/home/bioinfo/FISHER/blastDB/Cglabrata.aa
#database=/export/home/juan/blastDB/yeast.aa.rand

#database=/public/dbase/yeast.aa
database=$6
matrix=/home/bioinfo/FISHER/FASTA/BLOSUM62.ori
e=$4
for evalue in  $5 #1e-9 1e-6 1e-3 #"1e-2" "1e-3" "1e-1"
do
for g in $3 #9 10 11 12 13 #9 10 11 12 13
do
for prog in $2 #BLASTE BLASTQ BLASTgtE BLASTgtQ BLASTPGP
do
seqCycle=1
for seq in *FAS
do
cycle=1
hits_cycle=1
echo $seq
cp $seq $seqCycle.$cycle.$prog.$g.$e.$evalue.fas
cp $seq result.$seqCycle.$cycle.$prog.$g.$e.$evalue.fas
while [[ $hits_cycle -ne 0 && $hits_cycle -lt 700 ]]
    do
     echo "CYCLE = $cycle ..........."
        echo -n "Number of query for cycle $cycle => "
        grep -c "^>" $seqCycle.$cycle.$prog.$g.$e.
$evalue.fas
        /public/programs/MODPSC-closure.pl -P $prog -i
$seqCycle.$cycle.$prog.$g.$e.$evalue.fas -d      $database
-M $matrix
 -k BLOSUM62 -G $g -E $e -v 250 -b 250 -e $evalue -F F >
$seqCycle.$cycle.$prog.$g.$e.$evalue.out2
        #~/FISHER/E-scrdbprot/MODPSC-PGP.pl -P $prog -i
query.$cycle.$prog.$g.$e.$evalue.fas -d $database -M
$matrix -k BLOSUM62 -G $g -E $e -v 250 -b 250 -e $evalue -F
F
        echo "blast finished."
```

```
        #cat *.out > $seqCycle.$cycle.$prog.$g.$e.
$evalue.out2
        #rm *.out
        echo "blast reports in $seqCycle.$cycle.$prog.$g.
$e.$evalue.out2"
                /public/programs/parse_blastOP $seqCycle.
$cycle.$prog.$g.$e.$evalue.out2 $database
echo "$seqCycle.$cycle.$prog.$g.$e.$evalue.out2 $database"
#exit
        echo "blastOP finished"
        echo "Done for cycle $cycle ........"
        cat *.hits > blast.hits.fas
        rm *.hits
        echo -n "Number of hits for cycle $cycle: => "
        grep -c "^>" blast.hits.fas
        /public/programs/make-uniq.pl blast.hits.fas

        echo "Done make-uniq.pl for cycle $cycle ........"
        echo -n "Number of unique hits for cycle $cycle: =>
"
        grep -c "^>" uniq.hits.fas
        cat result.$seqCycle.*.$prog.$g.$e.$evalue.fas
>total-hits.$seqCycle.$prog.$g.$e.$evalue.fas
                /public/programs/new-hits.pl total-hits.
$seqCycle.$prog.$g.$e.$evalue.fas uniq.hits.fas
        echo "Done extracting new hits for cycle
$cycle ......"
        echo "$cycle  uniq `grep -c "^>" uniq.hits.fas`" >>
iblast.$g.$e.$evalue.$prog.$seqCycle
        rm uniq.hits.fas
        cycle=$((cycle+1))
        echo -n "Number of new hits for cycle $cycle: => "
        grep -c "^>" new.hits.fas
        echo "$cycle new `grep -c "^>" new.hits.fas`" >>
iblast.$g.$e.$evalue.$prog.$seqCycle
        cp new.hits.fas result.$seqCycle.$cycle.$prog.$g.
$e.$evalue.fas
        echo "New hits in result.$cycle.fas"
        cp new.hits.fas $seqCycle.$cycle.$prog.$g.$e.
$evalue.fas
        echo "Query seq for next cycle copied"
        echo ""
        hits_cycle=`grep -c "^>" new.hits.fas`
        #rm *.ssout2
        rm  new.hits.fas
```

```
        rm blast.hits.fas
        rm *.1.fas
    done #while
        seqCycle=$((seqCycle+1))
done #for seq
done #for prog
done #for g
done #for evalue
exit 0
```

Composition-modified Matrices Improve Homolog Identification for Low-complexity Yeast Glycoproteins

Juan Coronado[1], Oliver Attie[1], Susan L. Epstein[2], Weigang Qiu[1], Peter N. Lipke[1,*]

[1]Department of Biological Sciences and Center for Gene Structure and Function, Hunter College of City University of New York, New York, NY 10021, USA

[2]Department of Computer Science, Hunter College of City University of New York, New York NY 10021, USA

Corresponding Author

Peter Lipke

Dept. of Biological Sciences

Hunter College

695 Park Ave.

New York, NY 10021

(212)-772-6235

fax: 212-772-5227

lipke@genectr.hunter.cuny.edu

Keywords: BLAST, FASTA, Yeast cell wall, BLOSUM62, *Saccharomyces cerevisiae,*

Running title: Glycoprotein search and alignment

Yeast glycoproteins are representative of low-complexity sequences, those rich in a few types of amino acids. Low-complexity protein sequences comprise more than 10% of the proteome, but are poorly aligned by existing methods. Under default conditions, BLAST and FASTA use the scoring matrix BLOSUM62, which is optimized for sequences with diverse amino acid composition. Because low-complexity sequences are rich in a few amino acids, these tools tend to align the most common residues in non-homologous positions, thereby generating anomalously high scores, deviations from the expected extreme value distribution, and small *e*-values. This anomalous scoring prevents BLOSUM62-based BLAST and FASTA from identifying correct homologs for proteins with low-complexity sequences including *Saccharomyces cerevisiae* wall proteins. We have devised and empirically tested scoring matrices that compensate in different ways for the over-representation of some amino acids in any query sequence. These matrices were tested for sensitivity in finding true homologs, discrimination against non-homologous and random sequences, conformance to the extreme value distribution, and accuracy of *e*-values. Of the tested matrices, the two best (called E and gtQ) gave reliable alignments in BLAST and FASTA searches, identified a consistent set of paralogs of the yeast cell wall test-set proteins, and improved the consistency of secondary structure predictions for cell wall proteins. The PERL and shell scripts, customized databases, and supplemental sensitivity curves presented in this paper are available at http://wallace.hunter.cuny.edu/~jc/docs/.

The ability to accurately align protein sequences is central to inference about the evolutionary history of genes, and therefore to the evolution of organelles and organisms as well. In addition, homology modeling and even functional inference through annotation of similar sequences depends on alignment accuracy. For low-complexity sequences, such as fungal cell wall proteins, errors caused by anomalous high scores for non-homologous sequences will inevitably lead to erroneous inferences for evolution, structure and function.

Low-Complexity Sequences in the Proteome. Proteins with low-complexity sequences are common and functionally important, but are not well aligned by existing procedures. These proteins are rich in a few amino acids, and thus have overall composition significantly different from the "average" compositions seen in the multiple alignments used to construct the BLOSUM alignment scoring matrices and for the BLAST statistical analyses . About 10% of known protein sequences have overall low-complexity; eukaryotic genomes and some bacterial pathogens contain even higher percentages of low-complexity sequences . The NCBI non-redundant database currently contains approximately 3.2 million sequences. Thus, there are about 320,000 low-complexity sequences, which cannot be accurately compared or aligned, and therefore cannot be compared on any large scale, either functionally or evolutionarily. In addition, there are low-complexity segments in half of all proteins . These segments also cannot be reliably aligned, and so are currently "masked" by SEG or similar procedures and then ignored by the alignment tools . In globular proteins, low-complexity sequences tend to occur as loops within and between globular domains , regions often important for protein

function. Recent papers have highlighted the need to solve this problem, and a logical solution is modification of scoring matrices to compensate for composition of the query sequence .

Fungal cell wall proteins are representative of low-complexity sequences; they average 35% Ser and Thr residues, with some 100-residue segments composed almost exclusively of these two amino acids . As a result, wall proteins are normally aligned only after SEG filtering to remove the low-complexity segments, so sequence comparisons cannot be made for the low-complexity regions. If there were rapid search and alignment protocols that could compare such compositionally-biased segments, then both evolutionary and structural comparisons could be attempted.

The major alignment problem for low-complexity sequences is called *low-complexity corruption* . Intuitively, low-complexity corruption results from the alignment of high-frequency residues. In fungal cell wall proteins, the problem is most egregious for Ser, Thr, Pro, Ala, and Val. This phenomenon gives high alignment scores and low *e*-values to non-homologous pairs of protein segments (high-scoring pairs or *HSP's*). For example, alignments of Ser with Ser and Thr with Thr in cell wall proteins give alignment scores of +4 and +5, respectively in BLOSUM62, the standard scoring matrix. Because the residue alignment scores are summed over the segments being aligned, the many pairs of aligned Ser and Thr residues will give a high summed total alignment score, even if the frequent amino acids are randomly distributed in the sequences. Indeed, in searches using low-complexity proteins as the query sequence there are enough abnormally high-scoring pairs that the distribution of all scores is skewed by the over-

representation of high scores (see Fig. 1B). The skew means that the score distribution deviates from the expected extreme value distribution, and *e*-values calculated from the scores are invalid because the underlying distribution is different. On low-complexity sequences, this combination of anomalous high scores and small *e*-values appears with any search and alignment tool that uses BLOSUM matrices, including BLAST, FASTA, and the initial alignments in PSI-BLAST. Thus, if the alignment scores for frequent amino acids were reduced appropriately, alignments of these residues would not artificially inflate the scores to generate HSP's from sequences with similar amino acid composition but dissimilar sequence.

. Matrices other than BLOSUM have been shown more appropriate for sequences of non-average composition . For example, to make discriminatory matrices and predict hydrophobic and transmembrane segments in proteins, the specialized matrices PHAT and SLIM use the background frequencies present in transmembrane alignments, instead of standard amino acid frequencies . Similarly, position-specific scoring matrices are used to predict coiled-coil structures and in all iterative searches after the first in PSI-BLAST . The effectiveness of these specialized matrices on their intended targets attests to the fact that adjustment of matrices to account for amino-acid composition in the query and target sequences can be highly discriminating and sensitive.

Goals and evaluation criteria. To improve the alignment of low-complexity sequences we have developed and tested modifications to produce scoring matrices adjusted for the composition of each query. The goal is to prevent alignments of sequences that are compositionally similar but non-homologous, and to generate

statistically significant, homology-driven alignments of low-complexity segments necessary for structural and evolutionary studies of the low-complexity portion of the proteome.

Each matrix modification method was evaluated on the following criteria, as summarized in Table 1: *sensitivity* (ability to find a high number of homologs) for both low-complexity and high-complexity query sequences; *discrimination* against randomized sequences and non-homologous proteins with similar amino acid composition; *conformance* with the expected extreme value distribution of alignment scores that should be generated during search; *accuracy* of derived *e*-values; and *computational efficiency.* The results demonstrated that two of the composition-based matrices are powerful adaptations for BLAST and FASTA searches and alignments for low-complexity yeast glycoprotein sequences.

Methods

Summary of methods. We searched for sequences similar to each of 10 yeast cell wall proteins. These proteins have low-complexity regions that constitute 40-100% of the ORF length. For each query sequence, the amino acid frequencies were determined, and the scoring matrix was altered by the rules described below. The modified scoring matrices (Table 2) were rescaled to the same κ and λ statistical parameters as the standard scoring matrix (BLOSUM62), so that the reported *e*-values are distributed similarly to those from BLOSUM62-based searches of high complexity sequences (Supplemental Table S1, NR query set). A similar re-scaling strategy is used in PSI-BLAST . (Note that

additional mathematical definitions and relationships are described in supplemental material.) The query sequence was then used as the query in BLAST or FASTA. HSP's were ranked by *e*-values.

FASTA calculates *e*-values for each search by comparison to scores generated by randomized query sequences. Therefore, the *e*-values reported for FASTA searches are appropriate for each query sequence and scoring matrix.

Matrix Modification Q. One way to change scoring matrices is to adjust each scoring element S_{ij} to compensate for the probability of a match at random. This approach keeps the target frequencies Q_{ij} equal to the standard target frequencies, in the hope that this will reduce random alignments of frequently-appearing amino acids. Each new matrix element S^*_{ij} can be calculated from

$$P_i P_j \exp\{\lambda S_{ij}\} = P^*_i P_j \exp\{\lambda S^*_{ij}\} = Q_{ij} \quad (1)$$

where P_i is the probability of occurrence of an individual amino acid i, P^*_i is the probability of amino acid i in the query sequence, and the new score is calculated from S_{ij} and P_j. P_j and Q_{ij} are taken to be unchanged, so one compensates for the low-complexity in the query, but not in the database sequence. λ predicts the width of the extreme value score distribution. In essence, each score S^*_{ij} is reduced or raised to compensate for the degree to which the frequency for i in the query sequence differs from the frequency for i in the standard ratios used in BLOSUM62 . The new matrix will have the same target frequencies in the context of the amino-acid composition of the query sequence that the original matrix had in the context of standard amino-acid composition. Because target frequencies Q_{ij} are kept constant, equation (1) guarantees that the λ of the matrix in the

context of amino-acid composition of the query sequence should not change. BLAST, however, requires that the matrix entries be integers, so λ does change after rounding of the score. For each search, λ^* can be set to the λ of the original matrix by multiplying each score by the ratio of the λ^* of the unscaled matrix to the λ of original matrix, as described earlier . We call this *matrix modification Q*, for target frequency.

Matrix Modification E. The problem of complexity corruption can be thought of in another manner. The expected score E of a given matrix is

$$E = \Sigma P_i P_j S_{ij} \qquad (2)$$

The BLAST statistical model requires the value E in equation (2) to be negative . If the probability of amino acid *i* in the query sequence is larger than the standard probability for *i* used in the database or score distribution simulation, the expected score for the *ij* pair will unduly contribute to the total score of alignments, and will select for randomly-aligned segments that have amino-acid composition similar to the query. Once again, we can adjust the score of the matrix to compensate for the fluctuation in the amino-acid composition of a query from the standard amino-acid composition, and yet retain the intrinsic property (i.e., expected score, E) of the matrix in the context of the query's amino acid composition as follows:

$$P_i P_j S_{ij} = P^*_i P_j S^*_{ij} \qquad (3)$$

We call this *matrix modification E*, for expected score. This modification significantly changes the value of λ, which is then reset according to equation (1).

Supplementary Fig. S1 shows the impact of matrix modifications on positive and negative scores, relative to the ratio between the probability that amino acid *i* occurs in

the query and the standard Robinson and Robinson probability for that amino acid. These matrix modifications decrease positive scores for frequent amino-acid pairs but increase negative ones. Matrix modification E increases the negative scores for frequent pairs, but such a negative score never becomes positive nor does a positive score ever become negative. In contrast, Q modifications can convert a negative score into a positive one or vice versa. As a result, the two types of matrix modifications produced distinct total scores and alignments.

gt and 32 Modifications. A "greater than" (*gt*) matrix modification was also implemented. Under this modification, scores are reduced only if a residue is more frequent in the query sequence than in "standard" frequencies calculated by Robinson and Robinson (i.e., $P^*_i / P_i > 1$). When applied to matrix modification E or to matrix modification Q, this produces scoring matrices *gtE* and *gtQ*, respectively.

PSI-BLAST uses BLAST-PGP, with a 32-fold scale-up of BLOSUM62 to enhance sensitivity during the first round of comparisons. We have used the same scaling factor to augment the BLOSUM62 matrix before adjusting for amino acid composition deviation. This generates the gtE32 and gtQ32 matrices; gap costs are also scaled up (Table2). The gtE32 and gtQ32 matrix modifications were implemented for FASTA only.

Implementation. As a test of the effects of composition-based matrix modifications, we carried out searches on two sets of proteins. The first was a test to find homologs of low-complexity yeast cell wall proteins in a combined database of the yeast proteome and three complete sets of randomized yeast ORF pseudo-sequences. Randomizations of the sequences were *global* (the entire sequence randomized for each

ORF) or *local*. For local randomizations, the sequence was randomized within contiguous windows of 12 residues. This window length corresponds to that of the SEG filter, and maintains the local entropy of the sequences. For searches with cell wall proteins as queries, HSP's with authentic yeast ORFs were counted as "True" hits and HSP's with randomized sequences were counted as "False." This designation favors non-discriminating matrices such as BLOSUM62, because some non-homologous sequences were counted as "True" hits for the tests in Fig. 2. Inspection of alignments (e.g., Fig. 1) and comparison of annotations (Table 4) showed that these non-homologous "TRUE" hits were not reported by in searches with gtQ or E matrices. The other search set was the Aravind dataset, which contains 103 domain-specific query sequences and a total of 1005 true positives in the yeast proteome, curated by E.L. Aravind and E.V. Koonin . We used the curators' definitions of "True" and "False" hits.

To test the sensitivity and selectivity of pairwise search algorithms for high-complexity sequences with the modified scoring matrices, stand-alone versions of BLAST (version 2.2.2) and FASTA (version 3) were used. As recommended , gap costs of 9-13 were used with BLAST, and lower costs, 5-9, for FASTA searches, and the gap extension cost was set at 1. Each search used one of the scoring matrices (described in the previous section) based upon the amino acid frequencies in the individual query sequence. The notations that describe each type of search with each type of matrix are summarized in Table 2.

All BLAST searches were implemented using the command-line executable `blastall` with the BLAST-*x* matrices, or the command-line executable `blastpgp` with

the BLAST-PGP matrix and the composition-based statistics flag on (`-t T`). Both command-line executables produce gapped pairwise alignments, but `blastpgp` uses composition-based statistics to assess significance and can be used to generate PSSM from first round hits. The PSSM is used to score the second round of searches in PSI-BLAST. To preserve comparability, `blastpgp` searches were relegated to one round (`-j 1`). The FASTA searches were conducted with the command-line executable `fasta34`. Command-line options were default unless specified otherwise. All matrix modifications searches were PERL and BASH shell scripts, executed on a Sun Microsystems SunBlade100 workstation running Debian GNU Linux. Searches were performed without SEG filtering unless specifically designated, and were repeated for several different gap values.

Transitive closure tests. We tested whether the similarity sets were closed for yeast cell wall proteins. These tests compared output from BLAST-PGP, FASTA-B, and the four matrix modifications that are sufficiently sensitive and discriminating to support searches with low-complexity sequences: -E, -gtQ, -gtQ32, and -gtE32. These searches used the 10 yeast cell wall proteins as query sequences to search the Yeast Protein Database (retrieved from NCBI). Searches were done with gap costs shown in Fig. 2. HSP's with *e*-values less than the specified cut-off for distinct new proteins in each round became the query set for the next round, still against the same database. This process continued until no new, distinct proteins with *e*-values below the specified cut-off were obtained .

Comparisons of the transitive closure sets were performed using a Java web

application (diverge.hunter.cuny.edu:8080/funome) and other Java code. The WAR file for web application is available at http://wallace.hunter.cuny.edu/~jc/docs. The GPI-protein set was taken from . Using the GO terms "cell wall (sensu fungi)" and "cell wall organization and biogenesis," the Gene Ontology sets were obtained from the Saccharomyces Genome Database site at (http://www.yeastgenome.org/). The authors curated the "cell wall protein" and "non-cell wall protein" classifications in Table 4.

Availability. The PERL and shell scripts, customized databases, and supplemental sensitivity curves described in this paper can be obtained at http://wallace.hunter.cuny.edu/~jc/docs/.

Results

General Approach. Two types of score adjustment, with several variations of each, were designed and evaluated in tests using a variety of query sequences and databases summarized in Table 1. The score adjustments and queries are described below.

Score Changes for Frequently Occurring Amino Acids. The E and Q matrix modification methods reduce the alignment score S_{ij} for aligned residues i and j for amino acids occurring at high frequency in a query sequence, but preserve the net negative value for the matrix that accurate statistical analyses of the alignments require . Each modification method yields a different scoring matrix for each query sequence. Each modification method and its variants compensate in different ways for the deviation from the standard Robinson & Robinson frequencies used to derive the gapped BLAST statistical parameters for BLOSUM62, as summarized in the Methods section . The E method keeps constant the expected score of the matrix; while the Q method keeps the

target frequencies Q_{ij} constant, where Q_{ij} is the expected frequency that a residue *i* in one sequence is replaced by *j* in randomly aligned sequences . These frequencies are determined in a set of standard alignments under BLOSUM62.

All matrix modifications are summarized in Table 2 and described in detail in the Methods section. Throughout, we append suffixes to indicate which modifications were applied to a search method (Table 2). For example, A "BLAST-BF" search indicates that unmodified BLOSUM62 ("B") was used, with SEG filtering ("F"). FASTA-gtE32 indicates that we carried out a FASTA search with three modifications to the BLOSUM62 matrix: E, gt, and 32. We adopt the BLAST filtering criterion as a working definition for a low-complexity sequence, that is, one with Shannon entropy less than 2.2 over a window of at least 12 amino acid residues .

Query sets. The cell wall query set for most searches with low-complexity queries was a group of 10 cell wall GPI-class mannoproteins : Cwp2p, Sag1p, Ssr1p, Tip1p, Sed1p, Tir1p, Flo11p, Aga1p, Flo1p, and Fig2p, with respective lengths 92, 650, 238, 210, 338, 254, 1367, 725, 1537, and 1609 . These sequences are representative of GPI-anchored fungal cell wall proteins, and include 6 unique genes, 2 members of the *FLO* gene family, and two members of the *TIR/TIP* family. These and other cell wall proteins are mosaics of high-complexity and low-complexity segments .

Tests with high-complexity queries used a standard dataset of 103 yeast signal transduction proteins as queries in searches of the *S. cerevisiae* proteome and 3 copies of the proteome with the ORF sequences randomized .

Effects of matrix modifications on searches with low-complexity query

sequences. The problem of low-complexity corruption is illustrated in Fig. 1. BLOSUM62-based BLAST or FASTA searches with yeast cell wall proteins as queries identified homologs with highly similar sequences (Fig. 1A), but also returned HSP's with randomized sequences and non-homologous proteins, even when score statistics were adjusted by PGP or when low-complexity regions were masked with SEG (Fig. 1B-D). These alignments were based on high frequencies of matched Ser and Thr residues, and therefore identified many non-homologous sequences as highly similar, a known consequence of low-complexity corruption . In the BLAST-B search, the highest scoring match to Muc1p was a random pseudoprotein segment derived from Dan4p. Similarly, the three highest scoring matches to Fig2p ($e<10^{-62}$) were randomized versions of Muc1p. Like the BLAST-BLOSUM62 searches, BLAST-BF and BLAST-PGP, which uses composition-based statistical analyses with BLOSUM62, gave matches in which >80% of the identities were Ser or Thr (Fig. 1C and D). Other residues were seldom aligned. PGP also identified a large number of best hits with similar composition but unlikely homology: among the highest scoring matches for Aga1p was Snt1p, a histone deacetylase subunit, and for Muc1p the third highest scoring match was to the Sec31p subunit of the ER protein translocation pore. These proteins are unlikely to be homologous on the basis of functional analogy, cellular localization, or alignment of conserved sequence motifs. In addition, BLAST-B, PGP, and BF searches identified many randomized sequences as HSP's with $e < 10^{-3}$.

Alignments were greatly improved after matrix scores were adjusted to reflect the composition of the query sequences. Of the matrix variants listed in Table 2, the E and

gtQ variations with BLAST or FASTA, and gtQ32 with FASTA as well gave more specific alignments. (Our website http://diverge.hunter.cuny.edu:8080/modmat has automated, composition-based matrix modifications and search capability for any query sequence.) E matrices were highly specific; they required regions of extensive identity to achieve HSP's with significant *e*-values. The Muc1p/Bsc1p homology (Fig. 1A) was the only significant hit for any of the three query proteins illustrated in Fig. 1. gtQ matrices showed more high quality HSP's, a result of acquisition of significant scores over even relatively short but highly similar segments (Fig. 1D). All of the significant HSP's were to proteins that are also localized to cell walls. Note that with gtQ the best match for Aga1p was in a segment that was aligned with a randomized Muc1p pseudoprotein in the best-match of the BLOSUM62-based search (panel B).

Thus, the alignments showed that searches with BLOSUM62 matrices were subject to low-complexity corruption, even with PGP statistics or SEG filtering. These findings were confirmed in the structural comparisons and the sensitivity and transitive closure tests described below. In contrast, gtQ matrices were highly sensitive, reaching significant *e*-values in relatively short segments of both-low complexity and high-complexity composition. The E matrices were highly discriminatory, and identified only long HSP's with high likelihood of homology.

Structural correlations and matrix modification. Alignments are especially important in structural comparisons. There are few structures known for low-complexity proteins, and indeed structures for low-complexity sequences are severely under-represented in the Protein databank . Therefore, apparent matches to non-homologous

sequences may be used mistakenly as the basis for alignment and modeling. Use of gtQ or E matrices can assure better alignments and more accurate structural predictions.

If aligned regions are homologous, they should have similar secondary structures . We tested the composition-modified matrices as predictors of concordant secondary structure predictions for pairs of HSP's with $e \leq 10^{-3}$. The cell wall query proteins were used to search the *S. cerevisiae* genome database. Each aligned sequence segment was used as input for GOR IV, a secondary structure predictor that does not depend on BLOSUM62-based alignment to homologous sequences . The GOR IV secondary structure predictions of α-helix or β-sheet were compared (Table 3). The gtQ matrices gave the highest degree of concordance, over 80%, followed by PGP and E matrices. However, the concordance values with PGP had high variance, due to inclusion of non-homologous HSP's (e.g., Fig. 1). We repeated the test for the subset of HSP's with $10^{-5} \geq e \geq 10^{-30}$, values for the alignments most likely to be relevant for such predictions. For these HSP's, E and gtQ matrices out-performed BLOSUM62-based matrices. Again, PGP searches had poor concordance and the greatest standard deviation (not shown), indicating variation in the quality of the matches, as expected in situations where HSP's include non-homologous matches. Thus, use of modified matrices significantly improved the reliability of secondary structure predictions.

Sensitivity and discrimination. Sensitivity curves are a standard way to illustrate the effectiveness of search strategies . These graphs (e.g., Fig. 2) illustrate sensitivity (number of homologs identified as HSP's) as horizontal displacement and discrimination (number of false hits identified as HSP's) as vertical displacement. Thus good

performance is indicated by a curve that has a long horizontal component with minimal verticality apparent only at the right hand end of the curve. Previous work has defined false hits either as randomized sequences of similar composition to the true hits , or as proteins known to be non-homologous . To test the composition-modified matrices for discrimination against non-homologous, low-complexity sequences similar to the query sequences, we searched the cell wall protein query set against the S. cerevisiae genome combined with the locally randomized pseudoprotein sequences described in the Methods section.

Fig. 2 shows sensitivity plots for the cell wall protein query set against the *S. cerevisiae* proteome and three locally randomized copies. All tested matrix modification methods performed better than BLAST with B or BF and FASTA-B, which were unable to discriminate between authentic and randomized sequences. BLAST-PGP, which uses composition-based statistics with BLOSUM62, found 25 true hits (including the 10 query sequences themselves) at *e*-values below that of the first false hit. Among the modified matrix searches, BLAST-E was highly discriminatory (found very few false hits even at large *e*-values). The gtQ matrices showed by far the best sensitivity (105 true hits with lower *e*-values than the best-scoring false hit). Thus, FASTA-gtQ32 identified the 10 query sequences and 95 paralogs of the query proteins at *e*-values that excluded false hits, whereas BLAST-PGP identified only 15 paralogs.

Transitive closure tests. We used transitive closure as an empirical test of the usefulness of the composition-based matrix modifications. The 10 cell wall proteins were used as query sequences in BLAST and FASTA searches. Each query was used with

different matrices derived from its own composition. The ORFs corresponding to all hits with $e < 10^{-3}$ were used as the query sequences in the next round of searches, again with scoring matrices derived from each specific composition. This procedure was repeated until no new HSP's were identified. If a search method discriminates between similar and non-similar sequences, transitive closure should terminate after identifying a relatively small set of sequences. On the other hand, low-complexity corruption or other artifacts will result in frequent identification of non-homologous proteins with low *e*-values. The consequences will include a larger number of search rounds to achieve closure, and the significant "hits" will potentially include much of the proteome.

As expected, BLAST-B failed to achieve closure on the low-complexity query sequences, even with a cut-off value of $e \leq 10^{-9}$. With a standard cut-off of $e \leq 10^{-3}$ there were increasingly many new hits in each round, with a total of 863 sequences after 5 rounds (15% of the yeast proteome; Fig. 3 and Table 4). BLAST-BF also failed to close. The other methods achieved closure in 3-10 rounds (Table 4). There were 192 different ORFs identified in one or more of the searches with composition-modified matrices. Of these, 47 ORFs were identified in all searches, with 1 more ORF identified by 5 of the 6 modified matrix methods. Thus, there was a core of 48 hits that were most similar to the query sequences.

BLAST-PGP, was the most sensitive method that closed, but it did not discriminate against non-homologous sequences. The BLAST-PGP test identified 135 hits not found in any other search. Most of these extra hits were due to low-complexity corruption, similar to that seen in Figs. 1 and 2. The alignments were rich in pairings of

non-homologous Ser and Thr, and there were multiple different alignments in the same segments of the protein pairs with the same score. Such multiple equivalent HSP's are typical of low-complexity corruption. Furthermore, the vast majority of these hits were in proteins that are unlikely to be related to cell wall proteins (Table 4).

We reasoned that the most likely homologs of the query sequences would be other cell wall and cell-surface proteins, since their composition and domain structures are similar to each other and substantially different from those of globular proteins . Therefore, we functionally classified the hits identified in the transitive closure tests. The 343 ORFs identified in any modified matrix search or BLAST-PGP or FASTA-BF were labeled Cell Wall or Not Cell Wall, either in accordance with the Gene Ontology (GO) database or as curated by the authors. BLAST-PGP and FASTA-BF searches included many non-cell wall proteins among the significant hits . In contrast, searches with E and gtQ composition-modified matrices identified a highly similar set of ORFs, almost all of which were classified as cell wall proteins in either BLAST or FASTA. A complete list of hits for BLAST-PGP and composition-modified matrix searches is shown in supplemental Table S4.

Effects of matrix modifications on searches with high-complexity query sequences. To assess the effects of composition-based matrix modification on searches with high-complexity sequences, we also tested our methods in searches with globular (high-complexity) proteins as queries. The Aravind dataset is a set of curated signal-transduction proteins within the *S. cerevisiae* proteome . 103 of these proteins were used as queries in BLAST and FASTA searches, counting the number of alignments with

curated "true" and "false" homologs within the previously-established criterion that $e \leq 10^{-2}$ (Table 5;). As previously reported, BLAST with BLOSUM62 was the most sensitive method, returning 46% of the known homologs at this e value . Among the composition-modified matrices, searches with gtQ performed well, with 82-86% of BLOSUM62's sensitivity in BLAST and 75% sensitivity in FASTA searches. B and gtQ had similar discrimination against False hits. Again, the E matrices were highly discriminatory and gave no false hits, but the searches were less sensitive. Thus, composition-modified matrices provided moderately lower sensitivity, but similar (gtQ) or increased (E) discrimination in searches with sequences whose composition is near the Robinson and Robinson average.

Score distributions. The reliability of e-values depends on the statistical distribution of the alignment scores, which must conform to the extreme value Gumbel distribution . We tested this conformance for BLOSUM62 and the gtQ and E modifications. Each test used as queries a 1000-residue segment from Flo1p, a randomized sequence with the same composition as the yeast cell wall query dataset, and a random sequence of the same composition as the Robinson and Robinson high-complexity dataset. As in previous tests of searches and matrices Altschul, 1996 #3}, each query was tested for Smith-Waterman alignments against 4 databases, each of size 10^4: high-complexity sequences randomized globally, high-complexity sequences locally randomized, and low-complexity sequences randomized globally, and low-complexity sequences randomized locally. For each search, the 10^4 alignment scores were binned and compared to expected scores in the extreme value distribution with Pearson's χ^2 test. The

distributions of alignment scores generated by the composition-modified matrices, as they should be, were similar to the extreme-value distribution with $p<0.005$. However, in BLOSUM62-based searches for low-complexity sequences in both low-complexity databases, the *p* value was > 0.03-0.07. Thus, BLOSUM62 conformed less well than the modified matrices to an extreme value distribution. The detailed data appears in Supplemental Table S1.

The score distributions were used to estimate the statistical parameters κ and λ of the distributions as well . For FASTA searches, assuming conformance with the extreme value distribution, κ and λ are calculated and *e*-values are derived from the distribution for each search . In contrast, standard BLAST assumes values for these parameters that were derived from empirical estimates in gapped searches of high-complexity sequences. It is noteworthy that for the BLOSUM62-based searches of cell wall queries against the randomized cell wall pseudo-sequences, the value of κ was as much as 10^6 times greater than the standard value of 0.0243. This difference is probably the major source of inaccuracy of *e*-values and subsequent low-complexity corruption in low-complexity searches using BLOSUM62. In contrast, the composition-modified matrices generated score distributions with κ values that differed from the standard less than 4-fold. The λ values were all close to the BLAST-assumed value of 0.24, again with the exceptions of the BLOSUM62-based cell wall searches against the low-complexity and low-complexity pseudo-sequence databases (Supplemental Table S1).

Another test for conformance is probability plots of the inverse Poisson distribution *p*-values for alignment scores. Although such plots are often used to compare

scores for two samplings of a population, they can also be used to illustrate the number of scores at each probability in two distributions . The plots in Supplemental Fig. S2 show the cumulative fraction of scores above given index scores for comparisons of the E and gtQ matrices compared to the distribution in BLAST-PGP search of the high-complexity query and database . The plots are linear, as expected for comparable score distributions.

***e*-values for false hits.** In an extreme value distribution, the mean best *e*-value of false hits should be 1 . We therefore calculated this quantity for each matrix modifications in both BLAST and FASTA searches using high- and low-complexity queries. In searches with high-complexity queries, all matrices had mean first false hit scores between 0.41 and 11.7 (Supplemental Table S2). Again, E matrices were the most discriminatory, and had the largest *e*-values for false hits. In contrast, in searches with low-complexity queries, the composition-modified matrices far outperformed BLOSUM62. For BLOSUM62, even with SEG filtering or composition-modified statistics, the mean *e*-values for the first false hits were between 10^{-3} and 10^{-46}. Furthermore, the best-scoring false hit in a BLOSUM62-based search had an $e = 10^{-110}$. In contrast, the modified matrices generated mean *e*-values between 10^{-2} and 10^{2}. Thus, on high-complexity searches, the E and gtQ modifications produced *e*-values close to 1 for the first false hit, as expected. On low-complexity sequences, the E and gtQ modifications produced *e*-values much closer to the expected value of 1 than in searches with BLOSUM62.

Computational Efficiency. In BLAST, the major computational burden is the time needed to extend the 2-4 letter words from the query sequence that find similarity to sequences in the database . We therefore measured computation time in BLAST and

FASTA. BLAST-E and BLAST-gtQ ran faster than BLAST-B and BLAST-PGP for low-complexity sequences, both on the *S. cerevisiae* genome database and on the database that consisted of the genome with randomized sequences (Table 6). The maximum difference was about a 25-fold speed up for the BLAST-E search with low-complexity queries. For high-complexity sequences, E matrices were slightly more efficient and gtQ matrices were 40% slower than standard BLAST methods. In contrast, composition-based matrix modifications had little effect on the scan times for searches by FASTA (Data not shown).

Discussion

There is an acute need for bioinformatic tools that align and compare low-complexity sequences. Most available programs merely identify or mask such segments . We have shown that strategies that base alignment scores on the frequency of specific amino acids in the query sequence greatly improve the reliability and usefulness of BLAST and FASTA searches for low-complexity query sequences. These E and gtQ matrix modification methods decreased the scores for common residues, and were highly discriminatory against non-homologous sequences. The searches using these matrices identified a consistent set of paralogs of known yeast wall proteins (Table 4). These proteins share homologous sequence regions and motifs that have not been identified in BLOSUM62-based searches (Coronado et al. in preparation). Searches with composition-modified matrices also improved structural concordance in aligned sequences (Table 3).

The modified matrices yielded alignment scores in BLAST and FASTA that

conformed to the extreme value distribution (Table S1, Fig. S1), and generated *e*-values more accurate than BLOSUM62-based searches (Table S2) for low-complexity sequences. In searches with high-complexity queries, the distributions also conformed to the expected extreme value distribution, but the increased discriminatory power of the modified matrices decreased sensitivity somewhat (Table 5). This finding is consistent with a report that BLOSUM62 is the most sensitive matrix for searches with high-complexity sequences .

Transitive closure with modified-matrix searches identified a consistent set of yeast proteins. The transitive closure tests demonstrated that searches with E or gtQ modified matrices reliably identified apparent homologs of cell wall query sequences (Table 4; GO annotation and manually curated sets). In contrast, BLOSUM62-based searches with standard statistics did not close, and hit a large fraction of the yeast proteome. The transitive closure test closed with BLAST-PGP, but the majority of the hits with $e < 10^{-3}$ were not cell wall-related proteins (Table 4). Indeed, inspection revealed that most of them were low-complexity sequences in mobile elements or RNA processing enzymes.

The BLAST and FASTA transitive closure tests with the three best-performing composition-based matrices (BLAST with E or gtQ, and FASTA with E, gtQ, or gtQ32) identified 61 apparent homologs of the yeast cell wall proteins with alignment *e*-values smaller than 10^{-3}. Of those apparent homologs, 48 were retrieved by all five of these modified matrix searches; FASTA-E retrieved only these 48 ORFs. One additional ORF, Ylr110c, was retrieved by the four other modified-matrix searches. Nine more ORFs

were identified by BLAST-gtQ, FASTA-gtQ, and FASTA-gtQ32. Based on inspection of the significant alignments and resistance to low-complexity corruption, the E and gtQ modifications used in BLAST, or used with high gap costs in FASTA, define a consistent set of potentially homologous low-complexity proteins efficiently and accurately (Tables 4 and S4).

Other matrix modifications. Matrices modified for composition of both query and target sequence might further increase sensitivity, but at the cost of calculating a new matrix for each HSP. An analysis of reciprocal hits in the transitive closure test shows that query-based modifications were sufficient to find all known paralogous pairs (see supplemental material).

In a different approach, Yu, Altschul and Wootton have proposed composition-based modifications of BLOSUM scoring matrices to do alignments of low-complexity sequences without SEG filtering. The scoring matrices in are corrected by keeping the total entropy of each matrix constant, a strategy to maximize sensitivity for queries of unusual composition. Thus, these modifications would apply to a different aspect of the low-complexity search and alignment problem. The consequences of such matrices at a large scale have not yet been published.

Structural Consequences. Disordered regions of proteins often include low-complexity sequences. DISORDER, a scoring matrix specific for disordered regions of structurally well-characterized proteins improves scores for homologous protein pairs with 40-50% identity . The discrimination ability is similar to BLOSUM62, and the increase in sensitivity appears to be 2-fold. In contrast, the E and gtQ matrices increased

discrimination for any query sequence, and gtQ showed a greater sensitivity. The result was greater agreement in predicted secondary structures of the aligned segments.

Summary. We have presented several ways to normalize alignment scores and statistical parameters for individual query sequences (Table 2). Of these, the E and gtQ modifications support sensitive, discriminating, accurate search and scoring statistics for proteins or segments whose amino acid composition is far outside the Robinson & Robinson amino acid frequencies originally used to estimate the statistical parameters of λ and κ.

Scoring matrix modifications E and gtQ rendered SEG filtering unnecessary, and generated alignment scores that conformed to the extreme value distribution, which BLOSUM62-based searches could not do for these sequences of unusual composition. The composition-based matrix modifications also generated score distributions with statistical parameters much closer to those assumed in gapped BLAST statistics, so the resultant *e*-values were more accurate than those from BLOSUM62, and at least as accurate as composition-based statistics in BLAST-PGP. Therefore, BLAST or FASTA with the E or gtQ modified matrices showed great resistance to low-complexity corruption, and reliably identified apparent homologs of these important, low-complexity sequences without masking out the low-complexity segments. Furthermore, for these sequences the efficiency of BLAST was improved, and the efficiency of FASTA was not significantly changed. For query sequences containing low-complexity regions, BLAST-gtQ, and FASTA-gtQ32 were the most sensitive search methods, and had good discrimination against non-homologous sequences of similar amino acid composition.

Matrix modification E with either BLAST or FASTA searches had maximal discrimination against non-homologous sequences, but was somewhat less sensitive. The results presented here demonstrate that composition-based matrix modifications discriminate against non-homologous alignments and therefore make possible accurate comparative studies of low-complexity sequences. This accuracy is necessary for phylogenetics, and for structural comparisons.

Another benefit of these matrices will be an analogous improvement in the accuracy of genomic annotations, which are often based on functional analogies for homologous sequences. For instance, transitive closure identified a set of 48 sequences in *S. cerevisiae* that are similar to the cell wall protein queries. Searches through fungal genomes have revealed that apparent homologs of these proteins are present in other ascomycetes and basidiomycetes (Coronado et al. in preparation). These homologies, in turn, imply commonalities in cell wall structure and function for fungi whose walls are not as well characterized as those of *S. cerevisiae*.

Acknowledgments

This work was funded by the NIH/NCRR/RCMI program (grant RR 03037), NIH/MBRS-SCORE program (grant S06 GM 60654), and the Howard Hughes Medical Institute (grant 52002679). J.C. is a Fellow of the NSF-MAGNET and RCMI programs.

Literature Cited

1. **Alba, M. M., R. A. Laskowski, and J. M. Hancock.** 2002. Detecting cryptically simple protein sequences using the SIMPLE algorithm. Bioinformatics **18:**672-8.

2. **Altschul, S. F.** 1998. Generalized affine gap costs for protein sequence alignment. Proteins **32:**88-96.

3. **Altschul, S. F., and W. Gish.** 1996. Local alignment statistics. Methods Enzymol **266:**460-80.

4. **Altschul, S. F., W. Gish, W. Miller, E. W. Myers, and D. J. Lipman.** 1990. Basic local alignment search tool. J Mol Biol **215:**403-10.

5. **Altschul, S. F., T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman.** 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res **25:**3389-402.

6. **Altschul, S. F., J. C. Wootton, E. M. Gertz, R. Agarwala, A. Morgulis, A. A. Schaffer, and Y. K. Yu.** 2005. Protein database searches using compositionally adjusted substitution matrices. Febs J **272:**5101-9.

7. **Berger, B., D. B. Wilson, E. Wolf, T. Tonchev, M. Milla, and P. S. Kim.** 1995. Predicting coiled coils by use of pairwise residue correlations. Proc Natl Acad Sci U S A **92:**8259-63.

8. **Caro, L. H., H. Tettelin, J. H. Vossen, A. F. Ram, H. van den Ende, and F. M. Klis.** 1997. In silicio identification of glycosyl-phosphatidylinositol-anchored plasma-membrane and cell wall proteins of Saccharomyces cerevisiae. Yeast **13:**1477-89.

9. **Chambers, J., W. Cleveland, B. Kleiner, and P. Tukey.** 1983. Graphical
114
Methods of Data Analysis. Wadsworth.

10. **Colussi, P. A., and P. Orlean.** 1997. The essential Schizosaccharomyces pombe gpil+ gene complements a bakers' yeast GPI anchoring mutant and is required for efficient cell separation. Yeast **13:**139-50.

11. **De Groot, P. W., K. J. Hellingwerf, and F. M. Klis.** 2003. Genome-wide identification of fungal GPI proteins. Yeast **20:**781-96.

12. **Dwight, S. S., M. A. Harris, K. Dolinski, C. A. Ball, G. Binkley, K. R. Christie,**

D. G. Fisk, L. Issel-Tarver, M. Schroeder, G. Sherlock, A. Sethuraman, S. Weng, D. Botstein, and J. M. Cherry. 2002. Saccharomyces Genome Database (SGD) provides secondary gene annotation using the Gene Ontology (GO). Nucleic Acids Res **30:**69-72.

13. **Garnier, J., J. F. Gibrat, and B. Robson.** 1996. GOR method for predicting protein secondary structure from amino acid sequence. Methods Enzymol **266:**540-53.

14. **Gerstein, M.** 1998. Measurement of the effectiveness of transitive sequence comparison, through a third 'intermediate' sequence. Bioinformatics **14:**707-14.

15. **Grigorescu, A., M. H. Chen, H. Zhao, P. C. Kahn, and P. N. Lipke.** 2000. A CD2-based model of yeast alpha-agglutinin elucidates solution properties and binding characteristics. IUBMB Life **50:**105-13.

16. **Henikoff, S., and J. G. Henikoff.** 1992. Amino acid substitution matrices from protein blocks. Proc Natl Acad Sci U S A **89:**10915-9.
17. **Henikoff, S., and J. G. Henikoff.** 1993. Performance evaluation of amino acid substitution matrices. Proteins **17:**49-61.

18. **Karlin, S., and S. F. Altschul.** 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. Proc Natl Acad Sci U S A **87:**2264-8.

19. **Li, X., P. Romero, M. Rani, A. K. Dunker, and Z. Obradovic.** 1999. Predicting Protein Disorder for N-, C-, and Internal Regions. Genome Inform Ser Workshop Genome Inform **10:**30-40.

20. **Lipke, P. N., and J. Kurjan.** 1992. Sexual agglutination in budding yeasts: structure, function, and regulation of adhesion glycoproteins. Microbiol Rev **56:**180-94.

21. **Liu, J., H. Tan, and B. Rost.** 2002. Loopy proteins appear conserved in evolution. J Mol Biol **322:**53-64.

22. **Lupas, A.** 1996. Prediction and analysis of coiled-coil structures. Methods Enzymol **266:**513-25.

23. **Muller, T., S. Rahmann, and M. Rehmsmeier.** 2001. Non-symmetric score matrices and the detection of homologous transmembrane proteins. Bioinformatics **17 Suppl 1:**S182-9.

24. **Nandi, T., D. Dash, R. Ghai, B. R. C, K. Kannan, S. K. Brahmachari, C.**

Ramakrishnan, and S. Ramachandran. 2003. A novel complexity measure for comparative analysis of protein sequences from complete genomes. J Biomol Struct Dyn **20:**657-68.

25. **Ng, P. C., J. G. Henikoff, and S. Henikoff.** 2000. PHAT: a transmembranespecific substitution matrix. Predicted hydrophobic and transmembrane. Bioinformatics **16:**760-6.

26. **Pearson, W. R.** 1998. Empirical statistical estimates for sequence similarity searches. J Mol Biol **276:**71-84.

27. **Pearson, W. R.** 2000. Flexible sequence similarity searching with the FASTA3 program package. Methods Mol Biol **132:**185-219.

28. **Popolo, L., and M. Vai.** 1999. The Gas1 glycoprotein, a putative wall polymer cross-linker. Biochim Biophys Acta **1426:**385-400.

29. **Promponas, V. J., A. J. Enright, S. Tsoka, D. P. Kreil, C. Leroy, S. Hamodrakas, C. Sander, and C. A. Ouzounis.** 2000. CAST: an iterative algorithm for the complexity analysis of sequence tracts. Complexity analysis of sequence tracts. Bioinformatics **16:**915-22.

30. **Radivojac, P., Z. Obradovic, C. J. Brown, and A. K. Dunker.** 2002. Improving sequence alignments for intrinsically disordered proteins. Pac Symp Biocomput:589-600.

31. **Schaffer, A. A., L. Aravind, T. L. Madden, S. Shavirin, J. L. Spouge, Y. I. Wolf, E. V. Koonin, and S. F. Altschul.** 2001. Improving the accuracy of PSIBLAST protein database searches with composition-based statistics and other refinements. Nucleic Acids Res **29:**2994-3005.

32. **Sim, K. L., and T. P. Creamer.** 2002. Abundance and distributions of eukaryote protein simple sequences. Mol Cell Proteomics **1:**983-95.
33. **Wise, M. J.** 2001. 0j.py: a software tool for low complexity proteins and protein domains. Bioinformatics **17 Suppl 1:**S288-95.

34. **Wootton, J. C., and S. Federhen.** 1996. Analysis of compositionally biased regions in sequence databases. Methods Enzymol **266:**554-71.

35. **Yona, G., N. Linial, and M. Linial.** 2000. ProtoMap: automatic classification of protein sequences and hierarchy of protein families. Nucleic Acids Res **28:**49-55.

36. **Yu, Y. K., and S. F. Altschul.** 2005. The construction of amino acid substitution

matrices for the comparison of proteins with non-standard compositions. Bioinformatics **21:**902-11.

37. **Yu, Y. K., J. C. Wootton, and S. F. Altschul.** 2003. The compositional adjustment of amino acid substitution matrices. Proc Natl Acad Sci U S A **100:**15688-93.

Table 1. *Guide to validation tests*

Test purpose	Methods and/or searches (matrices)	Data set			Results
		Query	Database (positive control)	Database (negative control)	
Sensitivity/discrimination (low complexity)	Sensitivity curves, BLAST and FASTA (11 modified and unmodified matrices)	10 cell wall proteins	Yeast proteome	Locally randomized yeast proteome (pseudoprotein sequences)	Fig. 2
Application in homology searching	Transitive closure, BLAST and FASTA (11 modified and unmodified matrices)	10 cell wall proteins	GO and manually annotated cell wall proteins in yeast proteome	Non-cell wall proteins according to GO and manual annotations	Fig. 3, Table 4
Sensitivity/discrimination (high complexity)	Sensitivity curves, BLAST and FASTA (11 modified and unmodified matrices)	Aravind (103 query sequences)	Aravind (true hits)	Aravind (false hits)	Table 5
Conformance to extreme value distribution	Distribution of scores, BLAST and FASTA with B, PGP gtQ, and E	Flo1p fragment, random sequences of low and high complexity	None	10,000 globally and locally random sequences with low and high complexity	Table S1; chi-square tests, κ and λ estimates
	Distribution of scores (11 modified and unmodified matrices)	Aravind (103 query sequences)	Homologous sequences in yeast proteome	Nonhomologous sequences in yeast proteome	Fig. S2
Accuracy of false-hit *e* values	Mean and best *e* values	10 cell wall proteins; Aravind	None	Nonhomologous sequences in yeast proteome	Table S2

Table 2. Summary of search methods and modified scoring matrices.

Search Tool	Matrix	Comment
BLAST	B	Standard BLOSUM62 (SEG filter not used)
	BF	Standard BLOSUM62 with SEG filtering
	PGP	BLOSUM62 with 32–fold expanded scaling and 32-fold gap costs; score distributions adjusted to reflect the composition of the query.
	E	Adjust scores to maintain Expected Score equal
	Q	Adjust scores to maintain Target Frequencies equal
	gtE	BLOSUM62 with E modifications for over-represented amino acids
	gtQ	BLOSUM62 with Q modifications for over-represented amino acids
FASTA	B	Standard BLOSUM62 (SEG filter not used)
	BF	Standard BLOSUM62 with SEG filtering
	E	Adjust scores to maintain Expected Score equal
	Q	Adjust scores to maintain Target Frequencies equal
	gtE	BLOSUM62 with E modifications for over-represented amino acids
	gtQ	BLOSUM62 with Q modifications for over-represented amino acids
	gtE32	32 * gtE modifications and 32* gap costs
	gtQ32	32 * gtQ modifications and 32* gap costs

Matrix	$10^{-3} \geq e$		$10^{-5} \geq e \geq 10^{-30}$	
	aligned residues[1]	concordance H+E (%)[2]	aligned residues[1]	concordance H+E (%)[2]
E	2845	80	2763	81
gtQ	3010	82	650	65
B	3937	74	1812	51
PGP	6245	63	243	58

Table 3. Concordance of GOR IV secondary structure predictions for cell-wall-related aligned sequences. The cell wall query set was used for BLAST searches of the *S. cerevisiae* genome. GOR IV was used to predict the conformation of all sequences in all HSP's within the designated range of *e* values. Each residue predicted to be in α-helix (H) or β-sheet (E) conformation was compared to its aligned partner and scored as concordant if the conformation predictions were identical.

[1] Number of aligned residues predicted to be in α-helix (H) or β-sheet (E) conformation in all HSP's with the designated *e* values.

[2] Per cent instances where both members of an aligned pair of residues are predicted to be in the same α-helical or β-sheet conformation.

Search	Matrix	Rounds to close	Hits		GO Cell Wall		Curated			
			Type	Number	Number	% of Hits	Cell Wall	Wall Biogenesis	Not Cell Wall	Unknown or Ambiguous
BLAST	B	>5	total	863	66	8%	n.d.[1]	n.d.	n.d.	n.d.
	BF	>8	total	784	60	8%	n.d.	n.d	n.d.	n.d
	PGP	7	total	192	28	15%	41	6	122	23
			unique[2]	135	4	3%	0	5	122	8
	E	3	total	48	18	38%	35	0	0	13
			unique	0	0	0%	0	0	0	0
	gtQ	4	total	64	26	41%	46	2	2	14
			unique	13	5	38%	8	2	2	1
FASTA	B	13	total	397	51	13%	n.d.	n.d.	n.d.	n.d.
	BF	10	total	158	43	27%	61	6	60	31
			unique	16	2	12%	2	0	14	0
	E	3	total	48	18	38%	35	0	0	13
			unique	0	0	0%	0	0	0	0
	gtQ	3	total	51	21	41%	38	0	0	13
			unique	0	0	0%	0	0	0	0
	gtQ32	3	total	51	21	41%	38	0	0	13
			unique	0	0	0%	0	0	0	0

Table 4: Comparison of transitive closure sets

[1] n.d. not determined

[2]Hits were classified as unique if they were identified only in the specified search. For example, transitive closure with BLAST-PGP identified 192 homologous ORFs, of which 135 were identified only in the BLAST-PGP search, and 57 were also identified in at least one other search.

Method	True homologs $e \leq 10^{-2}$	False hits $e \leq 10^{-2}$	Per cent of BLAST-B sensitivity
BLAST-B[1]	460	3	100%
BLAST-PGP	434	2	94%
BLAST-BF	436	2	95%
BLAST-E	231	0	50%
BLAST-Q	348	1	76%
BLAST-gtE	401	1	87%
BLAST-gtQ	390	3	85%
FASTA-B	388	0	84%
FASTA-BF	398	0	86%
FASTA-E	242	0	53%
FASTA-Q	223	80	48%
FASTA-gtE	339	1	74%
FASTA-gtE32	318	0	69%
FASTA-gtQ	319	0	69%
FASTA-gtQ32	345	1	75%

Table 5. Results of modified-matrix searches on the Aravind dataset.

[1]BLAST-B identified 45.8% of the total "true" homologs.

Query Set	Matrix	Computation time (sec)	
		S. cerevisiae genome	*S. cerevisiae* genome w. random sequences
Cell Wall proteins	E	3	9
	gtQ	8	26
	BLOSUM-PGP	60	230
	BLOSUM62	55	213
Aravind	E	15	43
	gtQ	24	80
	BLOSUM-PGP	18	56
	BLOSUM62	17	55

Table 6. Efficiency of modified-matrix BLAST searches.

Figure Legends

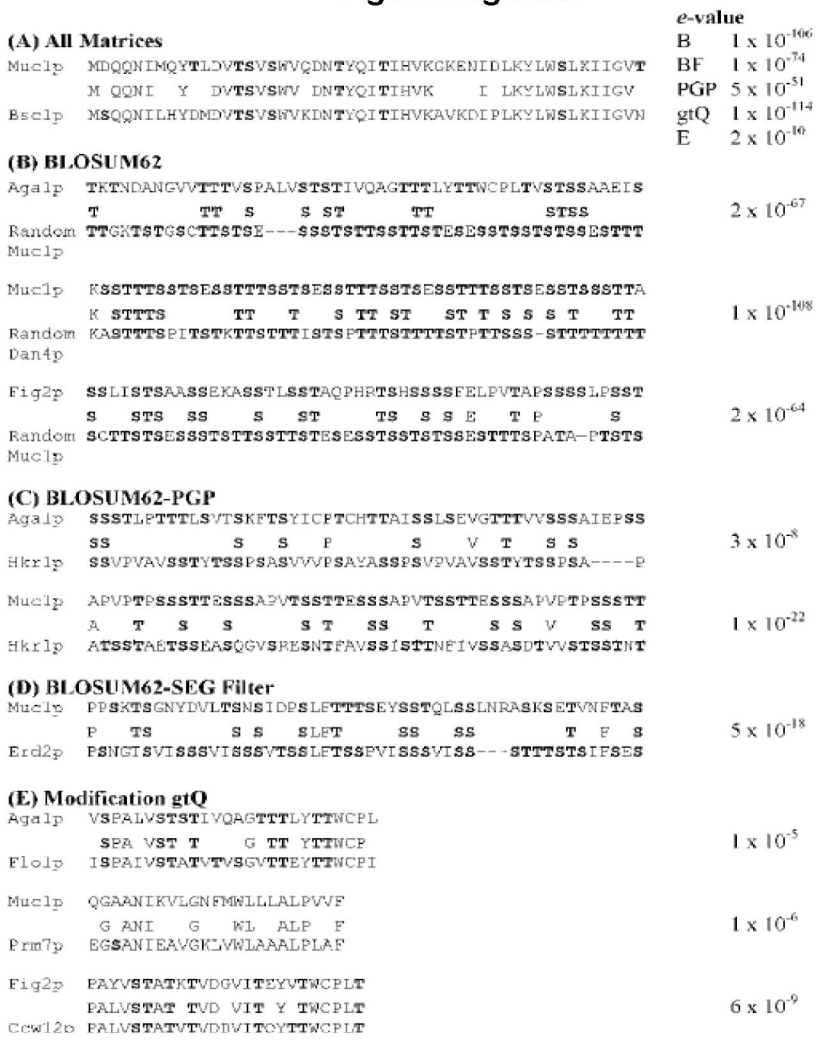

Figure 1. Alignments for best-scoring HSP's ($e \leq 10^{-3}$) for Aga1p, Muc1p, and Fig2p. The first 50 aligned residues in each alignment are shown, and identities are shown between the query sequence and the similar sequence. The bolded residues S and T are over-represented in the query sequences. A) Muc1p/Bsc1p alignment reported in all searches. *e*-Values for the alignments are shown on the left. B-D) Other highest scoring alignments and *e*-values for BLAST searches with each listed matrix. There were no other HSP's with $e < 10^{-3}$ for the BLAST-E searches.

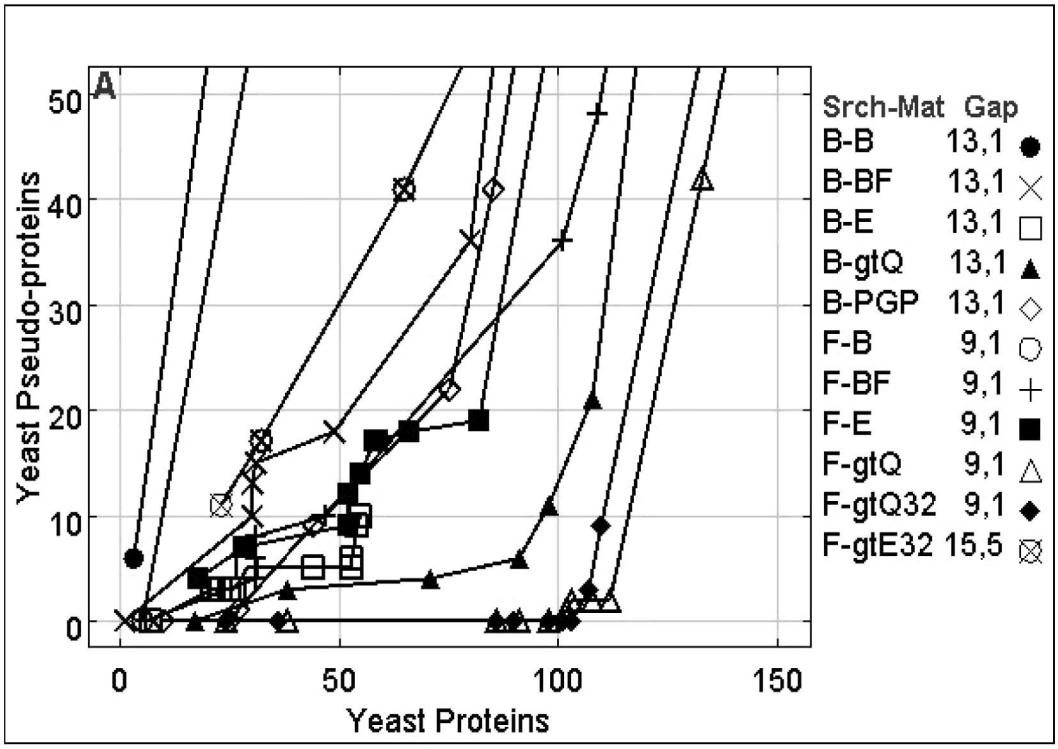

Figure 2. Sensitivity curves for BLAST (B) and FASTA (F) with different scoring matrices (Mat). Matrix B is the traditional BLOSUM62 matrix; E, Q, gtE, gtQ, gtE32, gtQ32 are described in the text. A) *S. cerevisiae* cell wall proteins searched against the yeast proteome and three locally randomized copies of the yeast proteome. Search output was binned into groups of hits by *e* -value (10^{-130}, 10^{-8}, 5x 10^{-8}, 10^{-7}, 10^{-5}, 5 x 10^{-2}, 10^{-1}, 5 x 10^{-1}, 1, 5, 10). True positives (yeast ORFs) and False positives (locally-randomized yeast pseudo-protein sequences) were counted and plotted for each group of hits. "Gap" is the gap costs (cost to open, cost to extend). For PGP, gtE32, and gtQ32 modifications, the listed gap values were multiplied by 32 before alignments were evaluated. The designated gap penalties gave maximal discrimination for each tested matrix.

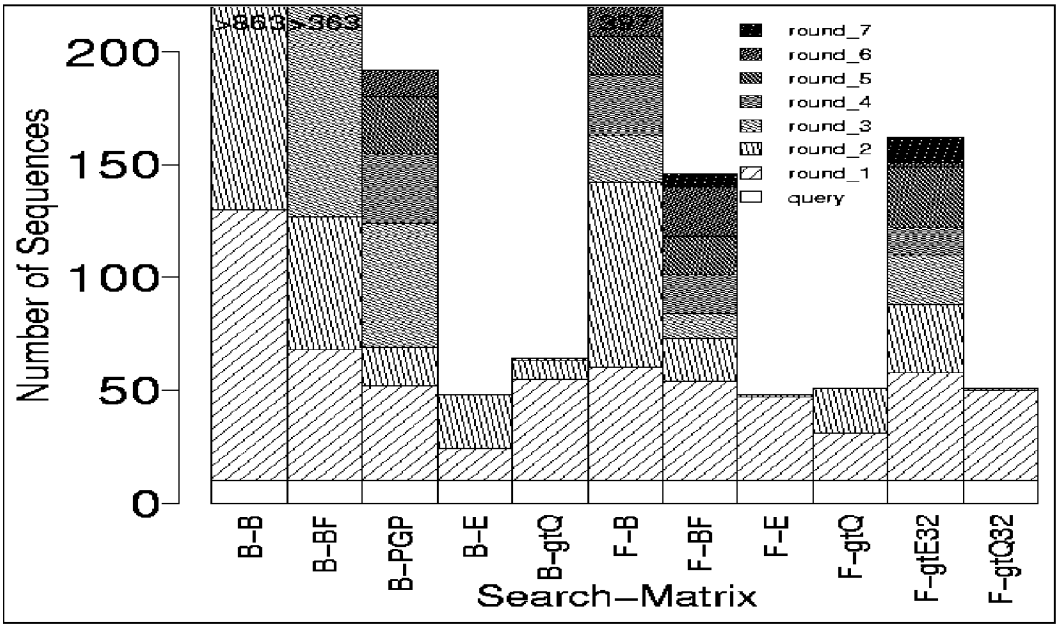

Figure 3. Transitive closure trial of BLAST-PGP, -E, -gtQ, FASTA-B, -E, -gtQ, -gtE32, -gtQ32, respectively. Iterative searches of the yeast cell wall protein query set (10 proteins) against the SGD yeast database were run as described in the text. Cut-off *e*-value was 10^{-3} for all searches. The hits marked "Query" are the identities to the query sequences, which have the smallest *e*-values in the first round searches. The Accession numbers for the identified ORF's are shown in Supplementary Table S3.

Discovery of Recurrent Sequence Motifs in *Saccharomyces cerevisiae* Cell Wall Proteins

Juan E. Coronado[1], Susan L. Epstein[2], Wei-Gang Qiu[1], Peter N. Lipke[1*]

[1]Department of Biological Sciences and Center for Gene Structure and Function, Hunter College of City University of New York, New York, NY 10021, USA

[2]Department of Computer Science, Hunter College of City University of New York, New York NY 10021, USA

*To whom inquiries should be addressed (current address): Peter Lipke, Dept. of Biology, Brooklyn College, 2900 Bedford Ave., Brooklyn, NY 11210 USA. (718)-951-5000 X1949; plipke@brooklyn.cuny.edu

Abstract

This paper describes a procedure for the discovery of recurrent substrings in amino acid sequences of proteins, and its application to fungal cell walls. The evolutionary origins of fungal cell walls are an open biological question. This question can be approached by studies of similarity among the sequences and sub-sequences of fungal wall proteins and by comparison to proteins in animals. We describe here how we have discovered building blocks, represented as recurrent sequence motifs (sub-sequences), within fungal cell wall proteins. These motifs have not been systematically identified before, because the low Shannon entropy of the cell wall sequences has hindered searches for local sequence similarities by sequence alignments. Nonetheless, our new, composition-based scoring matrices for local alignment searches now support statistically valid alignments for such low entropy sequences (Coronado et al. 2006. Euk. Cell 5: 628-637). We have now searched for similarities in a set of 171 known and putative cell wall proteins from baker's yeast, *Saccharomyces cerevisiae.* The aligned segments were repeatedly subdivided and catalogued to identify 217 recurrent sequence motifs of length 8 amino acids or greater. 95% of these motifs occur in more than one cell wall protein. The median length of the motifs is 22 amino acid residues, considerably shorter than protein domains. For many cell wall proteins, these motifs collectively account for more than half of their amino acids. The prevalence of these motifs supports the idea of fungal cell wall proteins as assemblies of recurrent building blocks.

Introduction

Discovery collects and organizes information in ways that are meaningful to the user. The identification of regularities in a large knowledge base is of particular interest when their existence is supported by other information. This paper describes such a discovery. We hypothesize that in unusual low-complexity protein-based sequences there are recurrent *motifs*, macros that provide building blocks for protein construction. The external support for our hypothesis is evolution. The principle results of this paper are the discovery of such motifs for yeast cell wall proteins in the fungus *Saccharomyces cerevisiae* (bakers' yeast) and the visually-oriented methods used to find them.

The evolutionary history of cell walls in fungi is an intriguing question. Fungi are a sister group to the animals, a non-walled kingdom, and both groups are postulated to descend from a common ancestor without a wall . The question is therefore "How did fungal walls evolve, and what materials were used to construct this phylogenetically unique cellular structure?" In fact, anecdotal evidence suggests that recurrent sequence motifs are common in fungal wall proteins . If this observation were shown to be generally true, then we could hypothesize that such motifs are "building blocks" that are replicated to make up a substantial and functionally critical portion of the proteins in the wall.

This question can be approached by comparative studies of the genes that encode the proteins in the walls, and comparisons of evolutionary history of the proteins and their component parts. Studies of molecular evolution depend upon the comparison of protein *sequences* (variable-length strings on a 20-letter alphabet of amino acid residues).

Comparisons of sequence similarities and differences allow the inference of gene divergence and re-arrangements, and therefore of evolutionary history. The occurrence of similar sequences in two different organisms or in multiple copies in one organism results from *homology,* mutual inheritance from a common ancestor. Homologous sequences diverge at a rate dependent upon the mutation rate and the strength of selection for or against changes in the sequence. Sequences tend to be more strongly conserved if they have a beneficial function, and in such cases there is evolutionary pressure to preserve the inherited sequences unchanged. If a sequence is not beneficial it may be neutral and allowed to mutate freely, or a sequence may be harmful, in which case mutations that abrogate its function are positively selected.

Some sequences or *fragments* (substrings) occur multiple times in the genome of an organism. *Paralogs* are fragment recurrences within a single organism due to duplications of the DNA during replication within this organism or in its ancestors. Duplicated copies can be recombined into other parts of the genome by transposition . Such duplications may persist in the genomes unless they are selected against. Like other homologous sequences, paralogous sequences can be beneficial, neutral, or harmful. The rate of accumulation of mutational substitution in paralogs is an indicator of the evolutionary pressure for or against mutation and of the time since the paralogs' creation by duplication.

The origin and evolution of fungal cell walls are problems whose solutions have been hampered by the lack of good methods to identify and compare the glycoproteins that predominate in fungal walls . Although 103 of the proteins in the *S. cerevisiae*

genome are known or predicted to be cell wall proteins , only a few of the proteins in the genome have known biological function (e.g., see Table 1). Since sequence similarity suggests functional similarity, our knowledge base should also include sequences similar to known cell wall proteins. Similarity between two sequences can be measured by the quality of the best alignment between them. (An alignment creates a one-to-one mapping between the sequences, and permits the insertion of *gaps*, sub-sequences of blanks.) The *score* of an alignment measures its quality; identical or functionally similar residues should be paired, and the number and length of the gaps minimized. The *e-value* (labeled "Expect" in Figure 1) is a statistical estimate of the probability of an alignment score that is the same or greater between two random sequences . Thus we seek proteins in the genome that have high-scoring (i.e., low *e*-value) alignments with known cell wall proteins.

Gene[a]	ORF[b]	Fragment Number	Sequence of Fragment	GO Annotation[c]		
				Location	Process	Function
Motif 12						
	YPL282C	1	VTRVITGVPWYSTRL	u	U	u
DAN2	YLR037C	2	VTRVITGVPWYSTRL	cw	U	u
	YMR325W	3	VTRVITGVPWYSTRL	u	U	u
	YOR394W	4	VTRVITGVPWYSTRL	u	U	u
	YLL025W	5	VTRVITGVPWYSTRL	u	U	u
	YIR041W	6	VTRVITGVPWYSTRL	u	U	u
	YKL224C	7	VTRVITGVPWYSTRL	u	U	u
PAU3	YCR104W	8	VTRVITGVPWYSTRL	u	U	u
PAU6	YNR076W	9	VTRMITGVPWYSTRL	u	U	u
DAN4	YJR151C	10	---MITGVPWYSTRL	cw	U	u
	YLL064C	11	VTRMITGVPWYSTRL	u	U	u
DAN1	YJR150C	12	VTRMITGVPWYSTRL	cw	St	u
	YOL161C	13	VTRMITGVPWYSTRL	u	U	u
PAU5	YFL020C	14	VTRVITGVPWYSSRL	u	U	u
PAU1	YJL223C	15	VTRMITGVPWYSSRL	u	U	u
	YIL176C	16	VTRMITGVPWYSSRL	u	U	u
	YGR294W	17	VTRMITGVPWYSSRL	u	U	u
PAU2	YEL049W	18	VTRMITGVPWYSSRL	u	U	u
	YBL108C-A	19	VTRMITGVPWYSSRL	u	U	u
	YAL068C	20	---MITGVPWYSSRL	u	U	u
PAU4	YLR461W	21	---MITGVPWYSSRL	u	U	u
	YDR542W	22	---MITGVPWYSSRL	u	U	u
	YGL261C	23	---MITGVPWYSSRL	u	U	u
	YHL046C	24	---MITGVPWYSSRL	u	U	u
DAN3	YBR301W	25	---MITGVPWYSSRL	u	U	u
TIR2	YOR010C	26	VSKMLTMVPWYSSRL	cw	Sr	u
TIR1	YER011W	27	VSKMLTMVPWYSSRL	cw	Sr	cwc
TIR4	YOR009W	28	---MLTMVPWYSSRL	cw	U	u
Motif 25[d]						
PST1	YDR055W	1	IYISDTSLQSVDGFSALKKVNVFNVNNNKKLTSIKSPVETVSDSLQFSFNGNQTKITFDD	cw	Cwob	u
ECM33	YBR078W	2	IIVSDTTLESVEGFSTLKKVNVFNINNNRYLNSFQSSLESVSDSLQFSSNGDNTTLAFDN	pm/cw	Cwob	u
	YCL048W	3	IVISDTSLANIENFNKVQEIDTFNINNNRFLETIHSNVKTIRGQFSVHANAKELELEMPH	pm	cwob/swa	u
SPS2	YDR522C	4	--ISDTALTSIDYFNNVKKVDIFNINNNRFLENLFASLESVTKQLTVHSNAKELELDLSN	cw	Swa	u

[a] Name given where assigned

[b] From *Saccharomyces* Genome Database

[c] Gene Ontology Annotation: u, unknown; cw, cell wall; pm, plasma membrane; st, sterol transport; sr, stress response; cwob, cell wall organization and biogenesis; swa, spore wall assembly; cwc, cell wall constituent.

[d] The first 60 amino acids of the 177-amino acid HSP

Table 1. Members of fragment families 12 and 25. The ORF name is the reference from the *Saccharomyces* Genome Database. The gene name, aligned sequences in single letter code, and Gene Ontology annotation are shown. Acronyms are listed and defined in the Appendix.

Existing powerful tools for sequence alignment accept a *query* sequence and return sequences most similar to it. These tools assume that the query has high *Shannon entropy* (high diversity in sequence elements, with no individual element at greater frequency than about 15%). In the sequences for cell wall proteins in *S. cerevisiae*, however, a few letters of the alphabet are over-represented; cell wall proteins are especially rich in the amino acids serine (symbols S or Ser), threonine (T or Thr) and a few others. As a result, these sequences have low Shannon entropy and the standard search methods, BLAST and FASTA , cannot discriminate between sequences similar to the query and dissimilar ones of similar composition . This problem, called *low-complexity corruption*, is also present in other low-entropy proteins, including mammalian mucins and other glycoproteins. Low-complexity corruption is caused by alignment scores that are based on high scores from matrices appropriate for high-entropy sequences.

Throughout this work, we have enhanced standard search tools with our *gtQ matrices.* described in . These composition-modified scoring matrices define high-scoring residue pairs, and are calculated for each query sequence. They reduce the score for aligned residues *i* and *j* in proportion to the likelihood of a random *ij* alignment in sequences of similar composition to the query sequence. Two important criteria for good alignments are *discrimination* (the ability to distinguish strings with similar sequence from those with similar amino acids composition but different sequence) and *sensitivity* (the ability to identify a maximal number of similar sequences). BLAST searches with gtQ matrices (*BLAST-gtQ searches*) have improved discrimination and do not sacrifice

sensitivity for cell-wall protein queries against fungal genome databases .

To interpret a genome, DNA sequences on the successive 3-element substrings of the 4-letter nucleic acid alphabet are rewritten as successive single elements in the 20-letter amino acid alphabet. We use the term *Open Reading Frame (ORF)* here to denote a potential protein sequence over the 20-letter alphabet. An ORF is always delimited by pre-specified start and stop signals. For statistical reasons, ORFs are defined to be at least 75 amino acids long; shorter ORFs are biologically present but rare. When an ORF has been demonstrated to exist biochemically or genetically, it is also called a *protein sequence.* (As a result, each gene in a genome that could encode a protein has an ORF name, and many also have a gene name and a protein name.) The set of all protein sequences and other ORFs from baker's yeast (*Saccharomyces cerevisiae*) was our knowledge base. In *S. cerevisiae*, gene names are italicized with three capital letters and a numeral, e.g. *DAN1* and *ECM33.*

Motifs are recurring sub-sequences found in one or more ORFs. We used BLAST-gtQ searches to find and align homologs of cell wall proteins and ORFs in *S. cerevisiae*. BLAST reports High Scoring Pairs (*HSPs*), well-aligned pairs of sequences consisting of whole sequences or sub-sequences that have high alignment scores and low *e*-values. We then devised a strategy to define and compare the sequence motifs that are paralogous within the *S. cerevisiae* wall proteome, the set of proteins that are located in the cell wall or are homologous to known cell wall proteins . Our results greatly constrain the possible models for evolution of fungal cell walls.

Results

We carried out BLAST-gtQ searches to identify the paralogs of known cell wall proteins in the yeast proteome. New protein sequences or ORFs identified as homologs were then used as queries in a second round, and the process was repeated until no new HSPs were identified. The HSPs from all these searches were then used to define a set of recurrent sequence motifs that make up a large part of the sequences of cell wall proteins (the cell wall *proteome*).

Identification of paralogs of cell wall proteins. The query set was the 103 *S. cerevisiae* cell wall proteins annotated as cell wall in the Gene Ontology (GO) database or identified as glycosylphosphatidyl inositol-anchored (GPI-anchored) proteins . BLAST-gtQ searches of the *S. cerevisiae* genome identified 1597 HSPs with *e* values $\leq 10^{-5}$. Two examples of HSPs are shown in Figure 1. The search found such HSPs in 68 other ORFs in *S. cerevisiae.* Thus a total of 171 ORFs, including the original 103 queries, were identified as cell wall components or their paralogs.

```
Query= YAL063C
>YHR213W YHR213W SGDID:S0001256, Chr VIII from 539147-539743,
     Uncharacterized ORF
     Length = 198

 Score = 403 bits (949), Expect = e-112
 Identities = 150/179 (83%), Positives = 161/179 (89%)

Query: 140 MTGYFLPPQTGSYTFKFATVDDSAILSVGGSIAFECCAQEQPPITSTNFTINGIKPWNGS 199
           MTGYFLPPQT SYTF+FA VDDSAILSVGG +AFECCAQEQPPITST+FTINGIKPW GS
Sbjct: 1   MTGYFLPPQTSSYTFRFAKVDDSAILSVGGNVAFECCAQEQPPITSTDFTINGIKPWQGS 60

Query: 200 PPDNITGTVYMYAGFYYPMKIVYSNAVAWGTLPISVTLPDGTTVSDDFEGYVYTFDNNLS 259
            PDNI G VYMYAG+YYP+K+VYSNAV+WGTLPISV LPDGTTVSDDFEGYVYTFD++LS
Sbjct: 61  LPDNIGGTVYMYAGYYYPLKVVYSNAVSWGTLPISVELPDGTTVSDDFEGYVYSFDDDLS 120

Query: 260 QPNCTIPDPSNYTVSTTITTTEPWTGTFTSTSTEMTTVTGTNGVPTDETVIVIRTPTTA 318
           Q NCTIPDPS +T S   TTTE WTGTFTSTSTEMTTVTGTNG PTDETVIV + PTTA
Sbjct: 121 QSNCTIPDPSKHTTS*IVTTTTELWTGTFTSTSTEMTTVTGTNGQPTDETVIVAKAPTTA 179

 Score = 90.2 bits (205), Expect = 2e-18
 Identities = 53/63 (84%), Positives = 56/63 (88%), Gaps = 1/63 (1%)
Query: 815 LVTTTTEPWTGTFTSTSTEMTTITGTNGQPTDETVIIVKTPTTAISSSLSSSSG-QITSF 873
           +VTTTTE WTGTFTSTSTEMTT TGTNGQPTDETVI+ K PTTA SSSLSSSS  QITS
Sbjct: 136 *IVTTTTELWTGTFTSTSTEMTTVTGTNGQPTDETVIVAKAPTTATSSSLSSSSSEQITSS 195

Query: 874 ITS 876
           ITS
Sbjct: 196 ITS 198
```

Figure 1: Two low-complexity BLAST HSPs generated for the query yeast ORF YAL063c with gtQ scoring. The top HSP shows an alignment of amino acids 140-318 of ORF YAL063c with amino acids 1-179 of ORF YHR213w. The bottom HSP is residues 815-876 of YAL063c with 136-198 of YHR213w. In the middle rows of the alignments, identical amino acids are repeated and similar amino acids score a "+". Note that amino acid residues 136-179 of YHR213w (in italics) align similarly with two regions of the query sequence (positions 275-318 in the first match and 815-858 in the second).

Identification and alignment of cell wall sequence motifs. The next challenge was to determine whether the HSPs contained recurrent sequence motifs. Traditionally, such motifs are found by sequence similarity within functional regions of proteins or by searches for recurrent sub-sequences within an ORF (motif searches). For cell wall

proteins, however, few functional regions are known, and their low-complexity regions make searches for repeats slow, insensitive to variations within motifs, and unable to discriminate against sub-sequences consisting of a single amino acid (repeats of a single letter) . We therefore adopted an approach that divides HSPs into unique sub-sequences (those fragments without homologs) and recurrent sub-sequences (those with at least one homolog). This approach is illustrated in Figure 2.

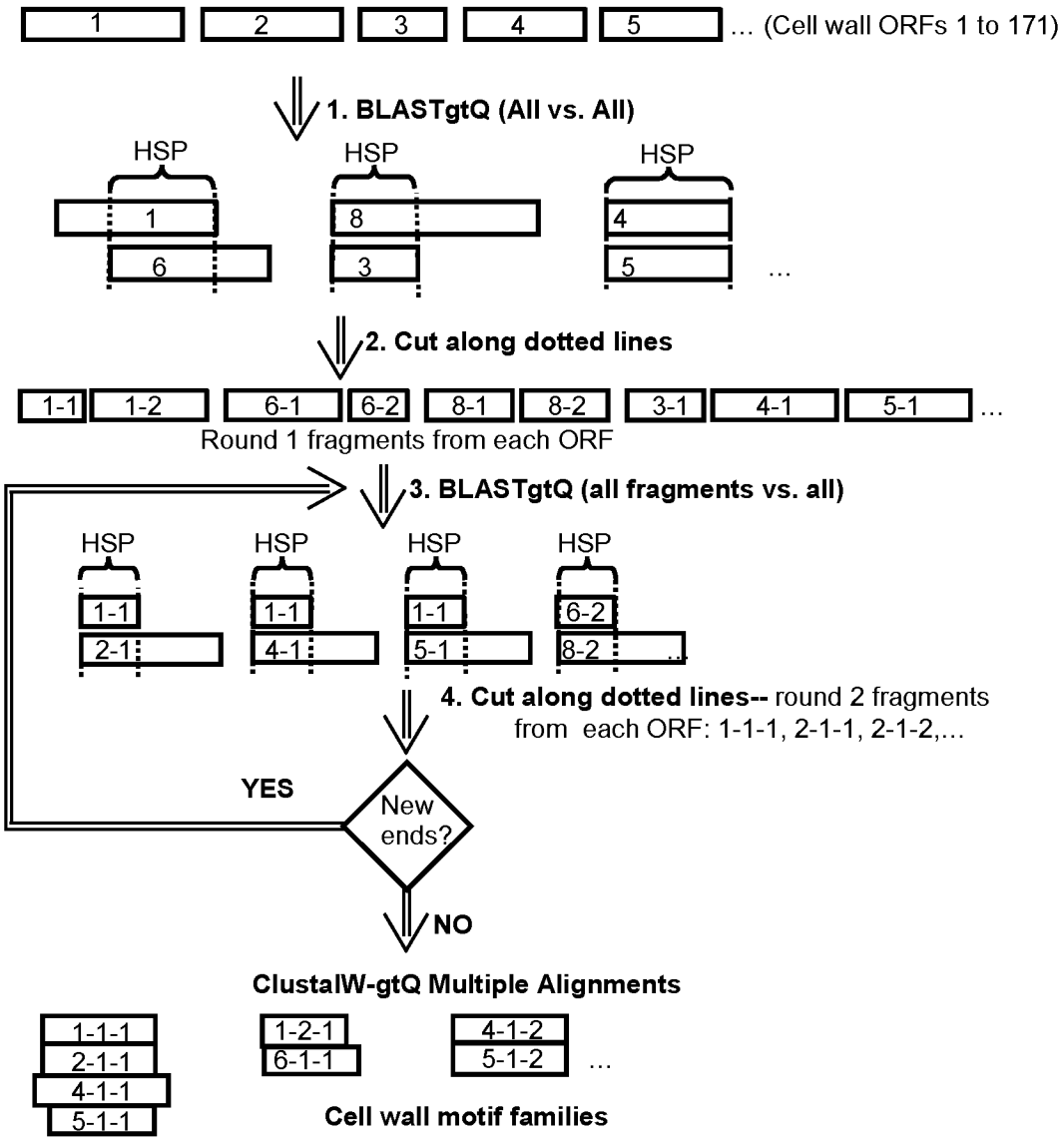

Fig. 2. Fragmentation process. (1) All 171 putative cell wall proteins were compared with BLAST-gtQ searches. Pairwise sequence alignments with $e \leq 10^{-5}$ detected homologies, denoted as vertically aligned segments. Note that more than one good alignment from different portions of a query is possible. (2) The boundaries of these alignments were treated as cuts to produce fragments with lengths $\geq n$. (3) All fragments were compared with BLAST-gtQ searches, and all new fragment alignments with length $\geq n$ and $e \leq 10^{-3}$ were used to cut the fragments as in step (2). Because the gtQ scoring matrices are modified based on the composition of each query string, statistically significant alignments were produced even with short fragments like 1-1. (4) Fragment alignment and cutting continued until there were no new fragments with length $\geq n$. Finally, fragments with sequence homology were aligned with CLUSTALW-gtQ to produce cell wall motif families.

Both ORFs in a BLAST HSP with $e \leq 10^{-5}$ were partitioned into fragments at the boundaries of the match. Each resulting fragment of length at least n was then used as the query in a new BLAST-gtQ search against all other wall protein fragments of length $\geq n$. The newly-aligned fragments were again cut at their boundaries as long as both resulting fragments would have length $\geq n$, and the process was repeated until no new fragments were identified with length $\geq n$ and $e \leq 10^{-3}$. (Because of the reduced length of the query strings, searches after the first round used a cut-off of $e \leq 10^{-3}$ rather than $e \leq 10^{-5}$.) The result was a set of sequence fragments that were either unique or were similar to as many as 41 other sub-sequences in the 171 protein database. Those fragments that had at least one other similar sequence constitute the set of recurrent *cell wall motifs*. The number of cell wall motifs identified depended on the motif minimum length n. We investigated n values from 8 to 20; $n = 20$ identified 156 motifs, while $n = 8$ identified the most, 217. Because $n = 8$ gave the maximum number of cell wall motifs, we chose this set for further analysis.

Characteristics of cell wall motifs. Mutually paralogous motifs were aligned by

CLUSTALW, again using composition-dependent gtQ scoring. CLUSTALW aligns a set of previously identified similar sequences in parallel, rather than pairwise, the way BLAST does . Each set of mutually aligned motifs was called a *motif family.* These motif families are available at http://diverge.hunter.cuny.edu:8080/modmat/misc.do?action=ShowCutDirs. The cell wall motifs ranged in length from 8 to 507 amino acids for $n = 8$. Ninety percent (90%) of the 217 motifs occurred once in each of multiple ORFs. The other fragments (10%) were present as two or more repeats in at least one ORF, and half of these (5%) occurred only as repeats in a single ORF. Thus 95% of the motifs were sub-sequences present in multiple ORFs.

Two representative alignments of cell wall motifs are shown in Table 1. Cell wall motif 12 is a highly similar group of 28 short fragments, each occurring exactly once in an ORF. Such conservation of sequence is characteristic of recently duplicated sequences or of strong function conservation (and therefore sequence conservation) after more ancient origin of the copies. The former interpretation is unlikely for this motif family, given its widespread occurrence in sequences that are otherwise not homologous, because it would be unusual for all copies to have entered the genome at the same recent time. Because many recombination mechanisms (e.g., transposition) tend to insert multiple copies of sequences in one locus, it is also interesting that these motifs occur only once per ORF. It is possible that multiple insertions would be detrimental and therefore selected against. There are other cell wall motifs which do display a tendency to multiple insertions in the same ORF .

There is great diversity in size, membership, and evolutionary rate of cell wall

motifs. Cell wall motif 25, the other example in Table 1, is longer (177 amino acids), present in 4 ORFs and has more sequence divergence. Motifs 12 and 25 illustrate the range in sequence length, frequency of occurrence, and divergence among HSPs. Cell wall motif 12 is highly conserved: there are 7 apparent amino acid polymorphisms (differences in amino acid sequence), while the other 8 residues remain constant. In contrast, fragment family 25 shows 73 polymorphic sites in the first 240 amino acids, 18-fold more substitutions per position in the alignment.

Most of the motifs are short, although a few are as long as ORFs and represent entire gene sequences that have been duplicated. For motifs of length $n \geq 8$, the median length is 22 amino acids; and 74% of them are of length 30 or less. These lengths are much shorter than protein domains, the longer sequences that define functional units of globular proteins. Such domains typically fold with discrete topologies, and have a median length near 100 amino acids.

We determined the prevalence of motifs in the cell wall proteins, by asking what fractions of the ORF sequences consisted of motifs. Twenty-eight of the original 171 query proteins contained no motifs (Figure 3). The other 143 wall proteins fell into three populations. About 34 proteins had some motifs, with a mean motif content of about 20% of their sequences. The largest population (97 proteins) was composed mostly of motifs, with a mean motif content of about 65%. Surprisingly, 12 protein sequences were composed wholly of motifs. Thus, almost 70% of the wall proteins had motif

content over 50%.

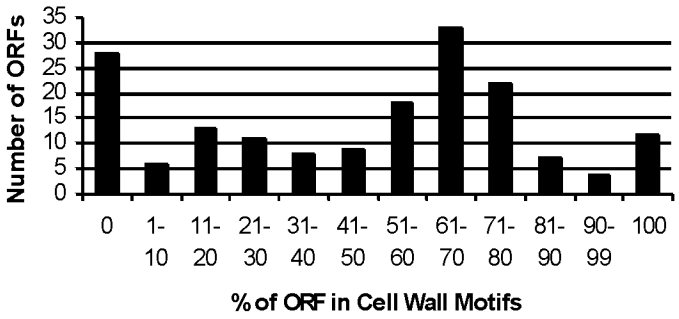

Figure 3. Fraction of cell wall protein sequences that are covered by motifs. The lengths of the fragments derived from a single cell wall protein were added and divided by the length of the protein.

Cluster graphs. The relationships among the members of individual cell wall motif families can be represented as graphs (Fig. 4). These graphs reveal sequence characteristics less accessible from traditional BLAST and CLUSTALW alignments. Motifs with many edges (similarities) cluster more closely; those with fewer similarities protrude from the cluster. Thus it is possible to distinguish quickly between close and

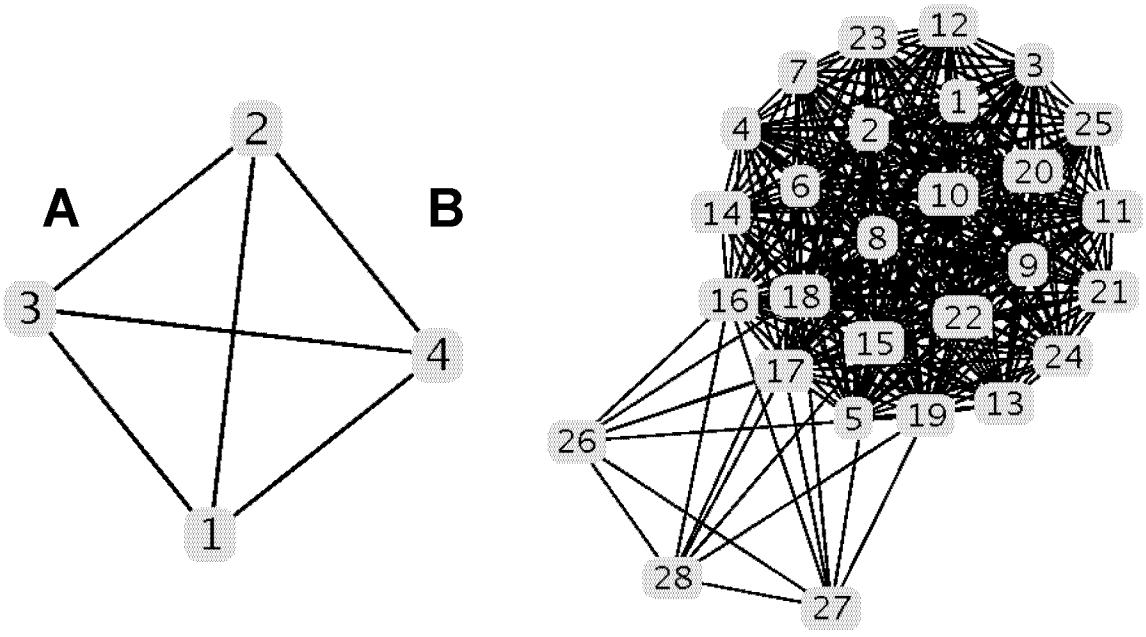

Figure 4. Graphs of motif families 25 (A) and 12 (B). The numbers indicate the order of the vertices in the multiple alignments in Table 1. For clarity all edges are the same thickness, but the graph can also be shown with thicker lines corresponding to smaller BLAST statistical *e*-values. Each edge represents a high-scoring alignment between a pair of motifs.

more distant relationships.

In Figure 4, the vertices represent the motif sequences in Table 1, and the edges represent a BLAST-gtQ alignment for each pair below the scoring threshold. The "`prefuse` JAVA graph library was used to construct a graph with ORFs as vertices and BLAST similarities as edges . Although omitted here for clarity, the thicknesses of the edges can represent the BLAST *e* value, with lower *e* values producing thicker edges. The `prefuse` library generates the diagram by simulating an environment where vertices repel each other while edges attract based on their thickness. Motif 25 family is a *clique* (every possible edge appears among its vertices), showing close relationships between the sequences. Motif family 12 is nearly a clique; 25 of the 28 members are completely interconnected. The three remaining sequences, however, form a clique (at the lower left in Figure 4B) with multiple edges (significant similarity) to 8 members of the larger clique Thus, the cluster graph demonstrates the inter-relationships among the individual members of the motif families. Others families are less clique-like (not shown).

Summary.

The shared functional and structural components of fungal cell wall proteins were largely unknown due to a lack of computational methods for detecting significant evolutionary relatedness among low-complexity sequences . Eukaryotic proteins are mosaics of domains, which are typically sequences of length 100 or more. Many of the sequences we identify here, however, are considerably shorter, and therefore their detection is more de-

manding computationally. We have used compositionally modified gtQ matrices to identify homologs of known yeast cell wall proteins, and to carry out a search for recurrent sub-sequences among them. The resultant aligned pairs were the basis for division into those subsequences that occurred only once and those that occurred more often, the latter called *motifs* in analysis of protein sequences. We have calculated the fraction of these proteins that are covered by motifs, and used graphs to illustrate relationships among them. To our knowledge, this is the first systematic search and analysis of motifs in fungal cell wall proteins. We conclude that there is a relatively limited set of between 156 and 217 sequence motifs present in the set of 171 putative wall proteins. A large proportion of the total sequence length represented by these proteins is part of one or more recurrent motifs. Some of these motifs have been noticed anecdotally before (such as the *DAN/PAU* and *PIR/TIR* gene families and some of the cysteine-rich motifs;) but the majority of motifs were newly identified in our analysis. The motifs have large ranges in length, frequency of occurrence, and rate of evolution. Many of the motifs occur in many ORFs, and a few are multiply repeated within individual ORFs.

This paper provides the first computational method to identify evolutionarily-conserved sequence motifs in the low-complexity part of a proteome. The result is the discovery of a set of relatively short sequence motifs that comprise a large fraction of the total length of the genes, consistent with the idea that these short motifs may be fundamental "building blocks" for fungal cell wall proteins. This is supported by how closely related the sequences in each motif family are, and by the prevalence of motifs in the sequences of fungal cell wall proteins. Thus, any theory for the evolutionary origin of

cell wall proteins must account for the prevalence of these motifs. Detailed analyses of the structure and function of specific motifs will lead to further insights into structure and evolution of the wall proteins.

Acknowledgements

This work was supported by NIH program grants from RCMI (G12 RR030307) and NIGMS-SCORE (S06 GM060654) to Hunter College.

Literature Cited

1. **Alba, M. M., R. A. Laskowski, and J. M. Hancock.** 2002. Detecting cryptically simple protein sequences using the SIMPLE algorithm. Bioinformatics **18:**672-8.

2. **Altschul, S. F., T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman.** 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res **25:**3389-402.

3. **Ashburner, M., C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock.** 2000. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. Nat Genet **25:**25-9.

4. **Brendel, V., P. Bucher, I. R. Nourbakhsh, B. E. Blaisdell, and S. Karlin.** 1992. Methods and algorithms for statistical analysis of protein sequences. Proc Natl Acad Sci U S A **89:**2002-6.

5. **Caro, L. H., H. Tettelin, J. H. Vossen, A. F. Ram, H. van den Ende, and F. M. Klis.** 1997. In silicio identification of glycosyl-phosphatidylinositol-anchored plasma-membrane and cell wall proteins of Saccharomyces cerevisiae. Yeast **13:**1477-89.

6. **Coronado, J. E., O. Attie, S. L. Epstein, W. G. Qiu, and P. N. Lipke.** 2006. Composition-Modified Matrices Improve Identification of Homologs of Saccharomyces cerevisiae Low-Complexity Glycoproteins. Eukaryotic Cell **5:**628-37.

7. **De Groot, P. W., K. J. Hellingwerf, and F. M. Klis.** 2003. Genome-wide identification of fungal GPI proteins. Yeast **20:**781-96.

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

9. **Ecker, M., R. Deutzmann, L. Lehle, V. Mrsa, and W. Tanner.** 2006. Pir Proteins of Saccharomyces cerevisiae Are Attached to beta-1,3-Glucan by a New Protein-Carbohydrate Linkage. J Biol Chem **281:**11523-9.

10. **Embley, T. M., and W. Martin.** 2006. Eukaryotic evolution, changes and challenges. Nature **440:**623-30.

11. **Gumbel, E. J.** 1958. Statistics of extremes. Columbia University Press, New York,.

12. **Heer, J., S. K. Card, and J. A. Landay.** 2005. Presented at the Conference on human factors in computing. Portland, OR.

13. **Higgins, D. G., J. D. Thompson, and T. J. Gibson.** 1996. Using CLUSTAL for multiple sequence alignments. Methods Enzymol **266:**383-402.

14. **Huang, J. Y., and D. L. Brutlag.** 2001. The EMOTIF database. Nucleic Acids Res **29:**202-4.

15. **Lipke, P. N., and J. Kurjan.** 1992. Sexual agglutination in budding yeasts: structure, function, and regulation of adhesion glycoproteins. Microbiol Rev **56:**180-94.

16. **Mrsa, V., T. Seidl, M. Gentzsch, and W. Tanner.** 1997. Specific labeling of cell wall proteins by biotinylation. Identification of four covalently linked Omannosylated proteins of Saccharomyces cerevisiae. Yeast **13:**1145-54.

17. **Pearson, W. R.** 2000. Flexible sequence similarity searching with the FASTA3 program package. Methods Mol Biol **132:**185-219.

18. **Steenkamp, E. T., J. Wright, and S. L. Baldauf.** 2006. The protistan origins of animals and fungi. Mol Biol Evol **23:**93-106.

19. **Verstrepen, K. J., A. Jansen, F. Lewitter, and G. R. Fink.** 2005. Intragenic tandem repeats generate functional variability. Nat Genet **37:**986-90.

20. **Verstrepen, K. J., and F. M. Klis.** 2006. Flocculation, adhesion and biofilm formation in yeasts. Mol Microbiol **60:**5-15.

21. **Verstrepen, K. J., T. B. Reynolds, and G. R. Fink.** 2004. Origins of variation in the fungal cell surface. Nat Rev Microbiol **2:**533-40.

22. **Wainright, P. O., G. Hinkle, M. L. Sogin, and S. K. Stickel.** 1993. Monophyletic origins of the metazoa: an evolutionary link with fungi. Science **260:**340-2.

23. **Yu, Y. K., and S. F. Altschul.** 2005. The construction of amino acid substitution matrices for the comparison of proteins with non-standard compositions. Bioinformatics **21:**902-11.

Conserved Processes and Lineage-Specific Proteins in Fungal Cell Wall Evolution

Juan E. Coronado[1,2], Saad Mniemneh[3], Susan L. Epstein[3], Weigang Qiu[1,2], Peter N. Lipke[1,2,4]*

[1]Department of Biological Sciences and [2]Center for Gene Structure and Function, and [3]Computer Science, Hunter College, City University of New York, New York, NY 10021, USA

[4]Department of Biology, Brooklyn College of City University of New York, Brooklyn, NY 11210, USA

[§]To whom correspondence should be addressed at e-mail: plipke@brooklyn.cuny.edu

Running Title: Fungal Cell Wall Evolution

Keywords: glycoprotein, sequence alignment, GPI, opisthokonts, eukaryote evolution

The cell wall is a defining organelle that differentiates fungi from its sister clades in the opisthokont superkingdom. With a sensitive technique to align low-complexity protein sequences, we have identified 187 cell wall-related proteins in *Saccharomyces cerevisiae*, and determined the presence or absence of homologs in 17 other fungal genomes. There were both conserved and lineage-specific cell wall proteins, and the degree of conservation was strongly correlated with protein function. Some functional classes were poorly conserved and lineage-specific: adhesins, structural wall glycoprotein components, and un-annotated ORFs. These proteins are primarily those that are constituents of the walls themselves. On the other hand, glycosyl hydrolases and transferases, proteases, lipases, proteins in the glycosyl phosphatidyl-inositol-protein synthesis pathway, and chaperones were strongly conserved. Many of these proteins are also conserved in other eukaryotes and are associated with wall synthesis in plants. This gene conservation, along with known similarities in wall architecture, implies that the basic architecture of fungal walls is ancestral to the divergence of the ascomycetes and basidiomycetes. The contrasting lineage-specificity of wall-resident proteins implies diversification. Therefore, fungal cell walls consist of rapidly diversifying proteins that are assembled by the products of an ancestral and conserved set of genes.

This detailed phylogenetic study of homologs to fungal cell wall proteins partially resolves the question of their origin. Cell walls are defining organelles that differentiate fungi from their sister clades in the opisthokonts including the animals . Although fungal cell walls consist of both carbohydrates and proteins, they are not homologous to cell walls in other organisms, such as plants or bacteria. It is, therefore, reasonable to ask whether fungal cell walls evolved independently in multiple lineages or diverged from a common ancestral origin. At one extreme, all or most cell wall components might be conserved in all fungal lineages, so that the same set of orthologous macromolecules would constitute cell walls across all existing fungal species. At the other extreme, few cell wall components might be conserved in structure or function during fungal evolution, so that fungal cell wall components would have lineages that are highly species-specific. Fungi consist of the phyla Ascomycota, Basidiomycota (together called Dikarya or sometimes "higher fungi"), Glomeromycota, Microsporidia, and two polyphyletic groups: Zygomycetes and Chytridiomycetes . In general, fungi are walled in at least part of the life cycle, and chitin is usually found as a wall component, although it is sometimes minor.

General features of fungal cell walls. Fungal walls appear to share a conserved basic design, although chemical and molecular analyses have been rare, except in ascomycetes and basidiomycetes. The major polysaccharides are glucans, chitin or chitosan , and mannans . GPI anchored proteins have also been found whenever sought . The best studied fungal cell walls are from bakers yeast, *Saccharomyces cerevisiae* , and the opportunistic pathogen *Candida albicans* . These two share basic design with other

ascomycetes: β-linked fibrous polysaccharides crosslinked by glycoproteins . The polysaccharides are assembled into fibrils and crosslinked to the Ser-rich and Thr-rich glycoproteins through modified GPI-glycans at the C-terminal and/or by ester bonds . Disulfide bonds between glycoproteins also contribute to cell wall integrity . Many *S. cerevisiae* wall components and assembly mechanisms have been found in the walls of other fungi, so *S. cerevisiae* appears to be a valid general model . The synthetic and secretion pathways that export fungal cell wall components are similar to those of extracellular glycoproteins in other eukaryotes . The protein components of *S. cerevisiae* cell walls are structurally and functionally heterogeneous, but have several common sequence features that enable their extra-cellular localization . These motifs include N-terminal secretion signal sequences, N- and O-glycosylation motifs, intermolecular disulfide bonding, and often hydrophobic C-terminal GPI addition signals. As in other eukaryotes, membrane translocation occurs in the Endoplasmic Reticulum (ER), and glycosylation takes place in the ER and Golgi, as the fungal cell wall components traverse the vesicular secretion pathway .

Molecular evolution and fungal cell walls. Fungal cell wall evolution has not been systematically investigated by sequence analysis, primarily because standard protein alignment techniques have proved inadequate for many fungal cell wall proteins. Alignment algorithms typically use scoring matrices based on amino acid composition of globular proteins , but fungal cell wall proteins have a very different composition. In *S. cerevisiae* wall proteins, amino acids S, T, A, V, G and P are present at higher percentages, and K, L, N, D, and E are more scarce . This composition gives

disproportionately large scores to alignments of over-represented residues, even when they are in non-homologous positions, a problem called "low-complexity corruption" . We devised a technique to minimize low-complexity corruption with scoring matrices adjusted for the frequency of each amino acid in the query sequence . Such query-specific matrices lower the score for matches of residues common in the query sequence, and so yield alignments free from random pairings of common residues .

Sequence-specific matrices allow identification and alignment of homologous sequences among fungal cell wall proteins . We have compared *S. cerevisiae* cell wall proteins to open reading frames (*ORFs*) in 18 other complete genomes and to the NR database . This large collection of ORFs supports the comparison of cell wall-related sequences in diverse fungal species. Our results show that fungal walls are both conserved in origin and diversified in composition.

Methods

Identification of cell wall proteins and their paralogs in *S. cerevisiae*. We identified a set of 103 *S. cerevisiae* cell wall proteins based on their annotation in the *S. cerevisiae* gene ontology (GO) database . This set was the union of the 85 entries GO-annotated as "cell wall" with the (overlapping) set of 74 proteins in *S. cerevisiae* known or predicted to have a signal sequence and a glycosyl phosphatidyl inositol (GPI) attachment signal but no transmembrane domains .

These 103 proteins were then used as queries in searches against the *S. cerevisiae* genome sequence database (SGD). To avoid low-complexity corruption, all searches reported here used BLAST with the query-specific *gtQ* scoring matrices . A *gtQ* matrix compensates for over-representation of any residue, while it preserves the negative value of the entire matrix. Thus it is much more highly discriminating, but does not sacrifice sensitivity. A transitive closure procedure conducted multiple rounds of searches against the *S. cerevisiae* genome, until no new homologous ORFs were identified. In a given round, new ORFs that participated in High Scoring Pairs (*HSPs*) with $e<10^{-5}$ became the query set for the next round, still against the same database. This process continued until no additional proteins with *e*-values below the specified cut-off were obtained . This method identified a total of 171 proteins, including the original 103 queries as *S. cerevisiae* cell wall components or their paralogs. To these genes we added 16 other sequences that are not annotated as cell wall in GO in SGD, but are known to be associated with wall synthesis and biogenesis . These 187 ORFs are listed in Table S1 and are referred to in the remainder of this paper as *S. cerevisiae cell wall-related proteins.*

Cell wall-related proteins from *Yarrowia lipolytica.* We needed an independent set of queries that could be used to test the relationship of homology occurrence and phylogenetic distance (Fig. 4). Therefore we identified a set of putative cell wall-related proteins from the ascomycete *Yarrowia lipolytica* as query sequences. Because annotation is sparse in species other than *S. cerevisiae,* we used the ORFs predicted to have GPI anchors, which are common in cell wall proteins in *S. cerevisiae*, *C. albicans,* and other fungi . *Y. lipolytica* putative cell wall-related sequences were identified as follows: we used the GPI-SOM server to find 237 proteins with predicted GPI signals and secretion signals . Of these 237 proteins, the tmHmm prediction server (http://www.cbs.dtu.dk/services/TMHMM/) identified 188 proteins without transmembrane domains. The Fungal BIG-PI server (http://mendel.imp.ac.at/gpi/fungi_server.html) was slightly more conservative, and identified a subset of 149 (79%) as potential GPI-anchored proteins . Like the *S. cerevisiae* cell wall proteins, 115 of the *Y. lipolytica* proteins contained regions with high S and T content: > 50 consecutive residues with > 30% S and T.

Genome comparisons. The genomes were retrieved from their respective databases (Table S2). The 187 *S. cerevisiae* cell wall-related ORFs were used as initial queries in BLAST-*gtQ* searches for sequence similarity in 17 other fungal genomes (listed in Supplementary Table S2). The 188 GPI-anchored proteins from *Y. lipolytica* were also used as a query set in homology searches of the other fungal genomes, but the results were not analyzed in detail because many of the ORFs have minimal annotations or ones based only on similarity to *S. cerevisiae* sequences.

Species tree. The power of any phylogenetic analysis depends on comparison to a reliable phylogenetic tree for the sampled organisms. Therefore, a tree was generated based on comparison of amino acid sequences in all orthologous ORFs in all 18 fungal genomes . A species tree was produced from a neighbor joining of the amino acid identity distance matrix .

Degree of conservation. Presence/absence of an ORF on an internal node of the tree was inferred from maximum parsimony, with a post-order tree traversal algorithm . The Degree of Conservation value of an ORF (in Figs. 3 and 5 and Table S1) was determined as the proportion of the total length of branches on which this ORF is present over the total tree branch length. The branches where an ORF is present are counted only if the ORF is present on both the parent and the child nodes.

Results

Phylogenetics of fungal proteomes. A species tree was generated based on the average amino acid identity of all orthologs between all pairs of genomes. Such widespread comparisons are more reliable than trees based on 18s RNA or on other limited sequence comparisons . Figure 1 shows the resultant neighbor-joining tree, congruent to other phylogenetic analyses of the fungi . *S. cerevisiae, S. paradoxus, S. kudriavzevii, and S. bayanus* form a closely related group and represent *Saccharomyces sensu stricto*, . The clade *Saccharomyces sensu lato* includes also *Candida glabrata, S. castelli, Kluyveromyces lactis,* and *S. kluyveri.* More distant ascomycetes and the basidiomycetes were clearly resolved, and the microsporidian *Encephalitozoon cuniculi* served as an outgroup.

The amino acid distance tree in Figure 1 shows that major fungal lineages have accumulated extensive sequence differences. These differences imply that the lineages split long ago, with early divergence of the basidiomycetes from the ascomycetes, and of the filamentous ascomycetes *Aspergillus fumigatus* and *Neurospora crassa.* from the yeasts. The basidiomycetes *Ustilago maydis* and *Cryptococcus neoformans* are themselves as divergent as the budding yeast *S. cerevisiae* and the fission yeast *Schizosaccaharomyces pombe* (~48% amino acid differences).

The functions of *S. cerevisiae* cell wall-related proteins and their paralogs. We identified a set of 187 cell wall-related proteins as those annotated in the SGD, GPI-anchored proteins, and their paralogs identified in transitive closure . For the 187 cell wall-related proteins, GO annotations report that 108 (63%) are localized either to the cell

wall (86 proteins) or to similar cell wall-associated spaces such as periplasmic (3 proteins) and extracellular (13 proteins). A large set (46 ORFs: 27%) has unknown location or no annotation. Only about 10% of the proteins are reported with intracellular localization including paralogs of chaperones and glycolytic enzymes. Those two classes of proteins are primarily cytoplasmic, but some proteins in each class are also proposed to be localized in cell walls in *S. cerevisiae* and in *C. albicans* on the bases of analyses of isolated walls and/or immunoassays on intact cells .

The distribution of molecular function for these 187 (as annotated by the SGD) includes several major classes (Fig. 2). Sixty-one proteins have no known molecular function (33%); 33 (18%) are glycosyl transferases, hydrolases, or transglycosylases (collectively called "carbohydrate-active enzymes" in the CAZY database [http://www.cazy.org], but abbreviated as "glycosylases" in subsequent discussions); 16 (9%) are involved in unfolded protein binding (called "chaperones" subsequently); 13 (7%) are structural constituents of the cell wall; and 8 (4%) have protease activity. The GO annotation for biological process most often associated with the 187 *S. cerevisiae* cell wall-related proteins was "cell wall organization and biogenesis." Forty-eight proteins, including chaperones and glycosylases, were annotated this way.

Occurrence of cell wall protein homologs in fungi. To study the conservation of cell wall proteins across the fungi, the 187 *S. cerevisiae* cell wall-related proteins were used as queries in BLAST-gtQ searches against 17 other fungal genomes. Sequences that aligned with *e*-value < 10^{-10} were counted as homologs. A homology was designated as an orthology only if the members of the HSP were reciprocal best hits and had at least 30%

of the aligned amino acids were identities across at least 70% of the length of the shorter member of the pair .

Figure 3 illustrates the occurrence of homologs and orthologs in two functional classes of cell wall proteins. Among the adhesins and invasins shown at the top in Fig. 3, those participating in mating were conserved within the closely-related *Saccharomyces sensu stricto* species. For the *FLO* family of mannose binding flocculins and invasins, homologs were also found in *Saccharomyces sensu lato* species and *D. hanseni*, but not in the more distant yeasts or the filamentous ascomycetes. In contrast, the glycosylases at the bottom in the figure show a different conservation pattern. For many members of this large functional class, there were homologs in each of the ascomycete and basidiomycete genomes. Thus these two functional classes of genes show different patterns of sequence conservation. Table S1 lists the 187 *S. cerevisiae* cell wall-related proteins.

The occurrence pattern for most genes followed the expected general pattern: there were homologs in the species most closely-related to *S. cerevisiae*, and the probability of recognizable homology decreased as phylogenetic distance increased. Fig. 4 illustrates this trend; it shows decreasing homolog identification with increasing phylogenetic distance. To confirm that this trend was not specific to *S. cerevisiae* and its cell wall-related proteins, we carried out a similar search and analysis for homologs of 188 potential GPI-anchored wall proteins from *Y. lipolytica* (a set of similar size). Most of the other species were similarly distant and had homologs to about 35% of the *Y. lipolytica* proteins. The corresponding points in Fig. 4 cluster near the occurrence-distance curve for *S. cerevisiae*, except for *Y. lipolytica/E. cuniculi* comparison, which

identified homologs to only 10 of the *Y. lipolytica* proteins.

Degree of conservation is related to molecular function. The origins of cell walls can be understood in terms of the roles of the conserved and non-conserved cell wall ORFs. For each function of the 187 ORFs in the *S. cerevisiae* wall (Fig. 2), we determined the *degree of conservation,* as detailed in the methods. The different functional classes differed significantly in their degrees of conservation (Fig. 5). Several functional classes were poorly conserved, with Degree of Conservation values below 0.4 (on the right in Fig. 5). These protein classes included a small set of Fe-transport-related proteins, the uncharacterized ORFs, cell wall structural components, and adhesins. The uncharacterized ORFs include many with known location in the cell wall but no known phenotype in gene deletions (Table S1). Thus these uncharacterized ORFs may encode proteins without unique or essential molecular activities, or those replaceable with other gene products . In contrast, components of the GPI-synthesis pathway, lipases, proteases, metabolic enzymes, glycosylases, and chaperones were strongly conserved, and were present in most tested fungi. In summary, a general observation is that the biosynthetic capability for cell walls is well conserved, but non-catalytic wall components, including the adhesins, and structural and putative structural proteins are not.

Cell wall-related proteins occur across the eukaryotes. Comparisons of the *S. cerevisiae* cell wall-related proteins against the Genbank NR database identified homologs in 137 ascomycetes species other than the genomes used in the initial fungal comparisons. Homologs were also found in 30 basidiomycete species, 7 zygomycete species, and 2 chytridiomycetes species. The GPI biosynthetic enzymes, metabolic

enzymes, and chaperones were most likely to have homologs (Fig. 6).

Searches from the 187 *S. cerevisiae* cell wall-related proteins into the NCBI non-redundant protein database identified similar sequences in organisms outside of the fungi as well (Fig. 6). Forty of the 187 proteins had homologs in other kingdoms. These 40 ORFs were in the functional groups most conserved in the fungi, including glycosylases, GPI synthetic proteins, aspartyl proteases, a group of related proteins important for plant pathogenesis, and two cytoplasmic groups with known or suspected cell wall involvement: chaperones and metabolic enzymes . Because these kinds of genes are present in many eukaryote kingdoms, they are likely to be ancestral to all eukaryotes, including fungi.

Discussion

What are the origins of fungal cell walls? The cell surface is at the front line of species adaptation. While the cell wall needs to maintain a basic functionality like other organelles, it is expected to evolve more quickly because it is directly exposed to biotic and abiotic selective forces. Cell surface structures are a defining feature of some of the deepest evolutionary lineages including bacteria, archea, fungi, plants and animals. Indeed, innovations in cell surface structures may be responsible for the success and persistence of major evolutionary lineages . One way to test for a causal relation between the evolution of the cell wall and the cell as a whole is to study the evolutionary conservation of cell wall macromolecular components. For example, a key evolutionary innovation in a lineage should be conserved and shared by all descendants, without repeated gain and loss.

Cavalier-Smith has argued that fungal cell walls are derived from ancestral chitinous walls of dessication-resistant cysts in many protist clades . Certainly, chitinous cysts and spores are present in other eukaryote kingdoms including Amoebozoa (such as *Entamoeba*), Chromalveolata (*Phytophthora*), and Excavata (*Giardia*:). In addition, *S. cerevisiae* walls share extracellular disulfide crosslinks and GPI anchored cell adhesion molecules with Amoebozoa, perhaps the earliest diverged eukaryotes (Fig. 6;). Cavalier-Smith's hypothesis motivated our use of genomics to identify genes that are key to wall biogenesis and structure, and to test which are conserved and which are subject to adaptive selection.

The deep roots for the ascomycete-basidiomycete split show large amino acid

distances among all homologs in the fungi (and Fig. 1), This observation implies that cell wall genes could also have diverged broadly. Therefore the identification of homologs in some cell wall proteins among distantly related organisms implies greater-than-average sequence conservation. In other words, anciently diverged fungal clades should retain sequence similarity only in the most conserved of the ancestral cell wall genes. The sequence comparisons summarized in this study demonstrate precisely that.

Prevalence of cell wall-related sequences in fungi. The evolutionary distance between two fungal species is correlated with the divergence of their cell wall proteins (Fig.4). The *Saccharomyces sensu stricto* species have homologs for at least 166 of the 187 *S. cerevisiae* cell wall-related ORFs, with amino acid identity of at least 86%. The number of conserved wall genes decreases in more distant fungal clades. *Saccharomyces sensu lato* yeasts have homologs for at least 70% of the *S. cerevisiae* cell wall-related proteins; filamentous ascomycetes had fewer. Among the homologs not found in filamentous ascomycetes were genes encoding extracellular proteins, including cell wall structural proteins, adhesins, and extracellular members of the GO classification "cell wall organization and biosynthesis." Also not conserved between yeast and filamentous ascomycetes were many of the proteins involved in the cell wall stress response pathway.

The two basidiomycete genomes in our study had homologs for 68 of the *S. cerevisiae* putative cell wall proteins (36%). The intracellular components were mostly conserved, except for the cell wall response pathway components. Among the cell-wall localized proteins, all but six of the glycosylases were conserved, as were lipases and proteases. Twelve GPI-anchored proteins from *S. cerevisiae* had basidiomycete

homologs. The large number of homologs to the *Y. lipolytica* GPI proteins supports this concept of conservation of GPI-anchored proteins, previously noted in genomic analyses of several fungi . Thus basidiomycetes share with *S. cerevisiae* intracellular and extracellular activities that are critical for biosynthesis and assembly of fungal cell walls.

In the search of the non-redundant NCBI database, almost every major fungal phylum contained homologs of some *S. cerevisiae* cell wall proteins (Fig. 6). (The single exception, Glomeromycota, is probably due to the small number of sequences in Genbank.) Thus homologs of some of the *S. cerevisiae* cell wall-related proteins are likely to be present throughout the fungi.

Differences in divergence of functional classes of cell wall-related proteins. A major finding of this analysis is that that the degree of conservation among cell wall proteins is strongly correlated with the cellular roles of those proteins (Fig. 5). Many of the proteins that reside in *S. cerevisiae* cell walls have diverged to the extent that homologs were not recognized in organisms outside of *Saccharomyces sensu lato.* The least-conserved include the three Fit iron transport proteins, and the largest class: homologs of unannotated *S. cerevisiae* cell wall ORFs. Homologs of unannotated sequences were often found only in the *sensu stricto* group (Table S1 and Fig. 5). These poorly conserved proteins probably have important but non-catalytic roles in walls.

Among the adhesins, most have homologs in the *Saccharomyces sensu lato* group, but not in more distant ascomycetes (Figs. 3 and Table S1). There is anecdotal evidence that adhesins are poorly conserved, because they are subject to sexual isolation and to diversifying selection for adaptation to different environments . The structural cell wall

proteins (in the *CWP, TIR, PAU,* and *DAN* families) are also poorly conserved. This observation may reflect the idea that this class is rich in proteins with very low complexity compositions. Within each sequence only short segments are under selective pressure to retain sequence: the secretion and GPI signals, and the relatively short glycosylation and transglycosylation sites . The majority of each sequence has low complexity, and would evolve faster than other parts of the genome (48).

Two protein classes show intermediate levels of conservation: the cell wall stress response pathways and cell wall biogenesis pathways. Their intermediate mean conservation scores result from their composition, a mixture of poorly conserved and highly conserved proteins. In general, those proteins resident in the walls are poorly conserved. Within these two classes, the plasma membrane and intracellular proteins are a mixture of conserved and non-conserved sequences (Table S1).

Strongly conserved proteins. In contrast, the strongly conserved functional classes are key biosynthetic and processing enzymes that must be useful in wall biosynthesis and assembly. These include the glycosylases, GPI synthetic enzymes, proteases, and lipases. These proteins include many orthologs that may function in homologous roles in wall biogenesis across the fungi. Among the most highly conserved are three sets of partially redundant glycosylases: the Chs chitin synthases, the Gas transglcosidases, and the Fks β1,3 glucan synthases. Other highly conserved components were peptidases, phosphatases, lipases, and enzymes of the N-acetylglucoasmine synthesis pathway (Table S1).

It is not surprising that chaperone sequences and triose phosphate dehydrogenases

(in the Metabolic Enzyme class) are strongly conserved, because these are genes whose sequences are highly conserved across the biome . Their primary locations are intracellular. Although their role as cell wall proteins is less well-known and somewhat controversial, it is well documented in two ascomycetous yeasts: *S. cerevisiae* and *C. albicans* . Given their controversial or auxiliary role in cell walls and their dual roles as wall-resident and key metabolic activities, these classes could be excluded from our analysis ("Chaperones" and "Metabolism" in Figs. 4 and 6). That exclusion would in fact strengthen our key observation, that wall biosynthetic genes generally have conserved sequences, and wall-resident proteins do not.

Modes of sequence divergence in cell wall components. In contrast to the homology of cell wall biosynthetic pathways, the actual composition of cell walls is lineage-specific. This dichotomy is similar to the distinction in genomics between conserved "house-keeping" genes and a faster-evolving set of "accessory" genes . Two evolutionary mechanisms could give rise to such diversity and lineage-specificity in the "accessory-like" genes: either rapid sequence divergence or frequent loss and substitution of the proteins. In the first evolutionary scenario, a homologous core set of cell wall proteins exists, but they are conserved only in protein structure and function, not in sequence. For instance, the greatest mean amino acid distance between genomes of *Saccharomyces sensu stricto* species is 16% (Fig. 1). The *S. cerevisiae* wall proteins generally have homologs within the *sensu stricto* group, so the sequences conserved in this clade must have been conserved within the time scale corresponding to this difference. Rapid evolution would make homology unrecognizable in more distant

groups. This result is consistent with the observation that many of these sequences are rich in low-complexity sequences, which are evolutionarily less constrained . In such a case, proteins resident in fungal cell walls may have a single origin but have diverged among different fungal lineages, through either neutral or adaptive divergence.

In the second evolutionary scenario, fungal cell wall architecture is conserved as a whole, with frequent additions and deletions of specific macromolecular components during lineage diversification. In this case, fungal cell walls are more analogous to a cell organelle that has evolved independently among lineages with few underlying homologous components. Functional features of fungal cell walls have been maintained by natural selection, similar to the multiple emergences of fins among different vertebrate groups. These fins show functional homology and anatomic convergence despite their multiple origins. Possible fungal examples might be the re-localization of novel proteins to the wall following acquisition of secretion signals and GPI anchor sequences or the internal repeats that include Gln residues to be esterified to wall glucans . Either of these additions could happen by recombination or by insertion of foreign DNA .

Whether the cause is a rapid sequence divergence or frequent loss and substitution, it appears beyond dispute that fungal cell wall components evolve faster than do core metabolic protein sequences (Fig. 5). Fast evolution in a large number of cell wall-related ORFs may be driven by adaptive divergence. Unlike other cell organelles, cell walls in fungi are in contact with the external environment, play a direct role in cell adaptation, and must have evolved as highly specialized and dynamic structures for colonization, host immune system evasion, signal transduction, transport, and structural

maintenance. Fungi have continuously put these structures to the test through natural selection, and the selection continues. Cell wall proteins that are harmful or neutral to the cell can be mutated or removed from the genome. As a result, the organism loses a maladaptive factor and gains efficiency in its growth and replication rates. Given that natural selection favors improved fitness and that the organism can gain two benefits from a single such event, the likelihood of selection for such mutations and deletions increases.

Conservation of cell wall biogenesis. Several classes of fungal cell wall-related proteins are common to eukaryotes: glycosylases, proteases, proteins needed for GPI synthesis, the poorly characterized Pry proteins, as well as chaperones. The presence of these classes throughout the fungi and in several eukaryote kingdoms (Fig. 6) implies that they are ancestral. These genes represent a conserved set of metabolic activities that were coordinated in the formation of cell walls. Many of the conserved glycosylases have plant homologs with annotations that show roles in plant cell wall biogenesis. These proteins include glucosidases Dse4, Spr1, and Exg1, mannosidases involved in glycoprotein biogenesis (Mns1 and Mnl1), and the Fks β1,3 glucan synthase subunits. The roles in plant wall biogenesis include synthesis and processing of callose (β1,3 glucan), cellulose, and glycoproteins.

The enzymes of chitin metabolism are extremely highly conserved. Two enzymes in the biosynthetic pathway for N-acetyl glucosamine (Gfa1 and YMR84w) are conserved in ascomycetes and basidiomycetes, as are chitin deacetylases (Cda1 and Cda2)(Table S1). Fungal chitin synthases (Chs) are homologous to those in other fungi,

metazoa, amoebozoa, and chromalveolata (Fig. 6). This high degree of conservation is consistent with an ancestral and continuing role of chitin in cell walls of many eukaryotes

.

Given the conservation of wall biosynthetic sequences, the broad distribution of specific wall glycoconjugates, and the similarity in roles of plant and fungal glycosylases, it is reasonable to infer that a common ancestor to fungi and other eukaryotes was an organism with an extensive carbohydrate chemistry repertoire. The commonalities identified here suggest that each eukaryotic lineage has conserved the mechanism to generate diverse extracellular structures, but has altered the products of the mechanism in their ancestor to form cell walls (in plants) or some other extracellular matrices (in metazoans). This conserved mechanism uses GPI-synthesis and processing enzymes, carbohydrate-processing enzymes, and chaperones to enable the localization and proper folding of the proteins .

In summary, genes that encode the proteins that make up fungal cell walls have evolved so fast that their homology is often not recognized, even within *Saccharomyces sensu lato*. This rapid divergence appears to be driven by the great diversity of strong selective pressures on cell interactions, including mating, colony and biofilm formation, pathogenesis, and immune escape. On the other hand, there is a conserved core of sequences that are involved in wall biogenesis throughout the fungi and in other eukaryote kingdoms. The conserved metabolic capabilities in Fig. 6 are apparently ancestral. Their conservation implies that the ability to organize a wall may predate the divergence of the plants from the opisthokonts including the fungi.

ACKNOWLEDGMENTS

We thank Eugene Koonin for his comments. Supported by NIH/NIGMS SCORE grants S06 GM 060654 and S06 GM 076168, and NIH/RCMI grant RR 030307. JEC was supported in part by a fellowship from NSF MAGNET-STEM.

Literature Cited

1. **Altschul, S. F., T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller,and D. J. Lipman.** 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res **25:**3389-402.

2. **Bendtsen, J. D., H. Nielsen, G. von Heijne, and S. Brunak.** 2004. Improved prediction of signal peptides: SignalP 3.0. J Mol Biol **340:**783-95.

3. **Brodsky, J. L., and G. Chiosis.** 2006. Hsp70 molecular chaperones: emerging roles in human disease and identification of small molecule modulators. Curr Top Med Chem **6:**1215-25.

4. **Cabib, E.** 2004. The septation apparatus, a chitin-requiring machine in budding yeast. Arch Biochem Biophys **426:**201-7.

5. **Cavalier-Smith, T.** 2006. Cell evolution and Earth history: stasis and revolution. Philos Trans R Soc Lond B Biol Sci **361:**969-1006.

6. **Cavalier-Smith, T.** 2004. Only six kingdoms of life. Proc Biol Sci **271:**1251-62.

7. **Cavalier-Smith, T.** 2002. The phagotrophic origin of eukaryotes and phylogenetic classification of Protozoa. Int J Syst Evol Microbiol **52:**297-354.

8. **Cavalier-Smith, T.** 2001. What are Fungi?, p. 1-37. *In* D. J. McLaughlin, E. C. McLaughlin, and P. A. Lemke (ed.), The Mycota. Springer, Berlin.

9. **Coronado, J. E., O. Attie, S. L. Epstein, W. G. Qiu, and P. N. Lipke.** 2006. Composition-modified matrices improve identification of homologs of *Saccharomyces cerevisiae* low-complexity glycoproteins. Eukaryot Cell **5:**628-37.

10. **Coutinho, P. M., E. Deleury, G. J. Davies, and B. Henrissat.** 2003. An evolving hierarchical family classification for glycosyltransferases. J Mol Biol **328:**307-17.

11. **Das, S., K. Van Dellen, D. Bulik, P. Magnelli, J. Cui, J. Head, P. W. Robbins, and J. Samuelson.** 2006. The cyst wall of *Entamoeba invadens* contains chitosan (deacetylated chitin). Mol Biochem Parasitol **148:**86-92.

12. **De Groot, P. W., A. D. de Boer, J. Cunningham, H. L. Dekker, L. de Jong, K. J. Hellingwerf, C. de Koster, and F. M. Klis.** 2004. Proteomic analysis of *Candida albicans* cell walls reveals covalently bound carbohydrate-active enzymes and adhesins. Eukaryot Cell **3:**955-65.

13. **De Groot, P. W., K. J. Hellingwerf, and F. M. Klis.** 2003. Genome-wide

identification of fungal GPI proteins. Yeast **20:**781-96.

14. **de Nobel, J. G., F. M. Klis, J. Priem, T. Munnik, and H. van den Ende.** 1990. The glucanase-soluble mannoproteins limit cell wall porosity in *Saccharomyces cerevisiae*. Yeast **6:**491-9.

15. **Delgado, M. L., J. E. O'Connor, I. Azorin, J. Renau-Piqueras, M. L. Gil, and D. Gozalbo.** 2001. The glyceraldehyde-3-phosphate dehydrogenase polypeptides encoded by the *Saccharomyces cerevisiae TDH1, TDH2* and *TDH3* genes are also cell wall proteins. Microbiology **147:**411-7.

16. **Dranginis, A. M., J. M. Rauceo, J. E. Coronado, and P. N. Lipke.** 2007. A biochemical guide to yeast adhesins: glycoproteins for social and antisocial occasions. Microbiol Mol Biol Rev **71:**282-94.

17. **Ecker, M., R. Deutzmann, L. Lehle, V. Mrsa, and W. Tanner.** 2006. Pir proteins of *Saccharomyces cerevisiae* are attached to beta-1,3-glucan by a new protein-carbohydrate linkage. J Biol Chem **281:**11523-9.

18. **Eisenhaber, B., G. Schneider, M. Wildpaner, and F. Eisenhaber.** 2004. A sensitive predictor for potential GPI lipid modification sites in fungal protein sequences and its application to genome-wide studies for *Aspergillus nidulans, Candida albicans, Neurospora crassa, Saccharomyces cerevisiae* and *Schizosaccharomyces pombe*. J Mol Biol **337:**243-53.

19. **Fankhauser, N., and P. Maeser.** 2006. KohGPI: Identification of GPI-anchor signals by a Kohonen Self Organizing Map. http://gpi.unibe.ch/.

20. **Fitch, W.** 1971. Toward defining the course of evolution: Minimum change for a specified tree topology. Systematic Zool. **20:**406-416.

21. **Fu, X., W. Jiao, and Z. Chang.** 2006. Phylogenetic and biochemical studies reveal a potential evolutionary origin of small heat shock proteins of animals from bacterial class A. J Mol Evol **62:**257-66.

22. **Gerstein, M.** 1998. Measurement of the effectiveness of transitive sequence comparison, through a third 'intermediate' sequence. Bioinformatics **14:**707-14.
23. **Hecker, M.** 2003. A proteomic view of cell physiology of *Bacillus subtilis*--bringing the genome sequence to life. Adv Biochem Eng Biotechnol **83:**57-92.

24. **Hochstenbach, F., F. M. Klis, H. van den Ende, E. van Donselaar, P. J. Peters, and R. D. Klausner.** 1998. Identification of a putative alpha-glucan synthase essential for cell wall construction and morphogenesis in fission yeast. Proc Natl Acad Sci U S A **95:**9161-6.

25. **James, T. Y., F. Kauff, C. L. Schoch, P. B. Matheny, V. Hofstetter, C. J. Cox, G. Celio, C. Gueidan, E. Fraker, J. Miadlikowska, H. T. Lumbsch, A. Rauhut, V. Reeb, A. E. Arnold, A. Amtoft, J. E. Stajich, K. Hosaka, G. H. Sung, D. Johnson, B. O'Rourke, M. Crockett, M. Binder, J. M. Curtis, J. C. Slot, Z. Wang, A. W. Wilson, A. Schussler, J. E. Longcore, K. O'Donnell, S. Mozley-Standridge, D. Porter, P. M. Letcher, M. J. Powell, J. W. Taylor, M. M. White, G. W. Griffith, D. R. Davies, R. A. Humber, J. B. Morton, J. Sugiyama, A. Y. Rossman, J. D. Rogers, D. H. Pfister, D. Hewitt, K. Hansen, S. Hambleton, R. A. Shoemaker, J. Kohlmeyer, B. Volkmann-Kohlmeyer, R. A. Spotts, M. Serdani, P. W. Crous, K. W. Hughes, K. Matsuura, E. Langer, G. Langer, W. A. Untereiner, R. Lucking, B. Budel, D. M. Geiser, A. Aptroot, P. Diederich, I. Schmitt, M. Schultz, R. Yahr, D. S. Hibbett, F. Lutzoni, D. J. McLaughlin, J. W. Spatafora, and R. Vilgalys.** 2006. Reconstructing the early evolution of Fungi using a six-gene phylogeny. Nature **443:**818-22.

26. **Jose-Estanyol, M., F. X. Gomis-Ruth, and P. Puigdomenech.** 2004. The eightcysteine
motif, a versatile structure in plant proteins. Plant Physiol Biochem **42:**355-65.

27. **Kapteyn, J. C., H. Van Den Ende, and F. M. Klis.** 1999. The contribution of cell wall proteins to the organization of the yeast cell wall. Biochim Biophys Acta **1426:**373-83.

28. **Klis, F. M., A. Boorsma, and P. W. De Groot.** 2006. Cell wall construction in *Saccharomyces cerevisiae*. Yeast **23:**185-202.

29. **Klis, F. M., P. de Groot, and K. Hellingwerf.** 2001. Molecular organization of the cell wall of *Candida albicans*. Med Mycol **39 Suppl 1:**1-8.

30. **Klotz, S. A., M. L. Pendrak, and R. C. Hein.** 2001. Antibodies to alpha5beta1 and alpha(v)beta3 integrins react with *Candida albicans* alcohol dehydrogenase. Microbiology **147:**3159-64.

31. **Konstantinidis, K. T., and J. M. Tiedje.** 2004. Trends between gene content and genome size in prokaryotic species with larger genomes. Proc Natl Acad Sci U S A **101:**3160-5.

32. **Leal, J. A., B. Gomez-Miranda, A. Prieto, and M. Bernabe.** 1993. Differences in cell wall polysaccharides of several species of *Eupenicillium*. FEMS Microbiol Lett **108:**341-5.

33. **Lee, J. H., S. Waffenschmidt, L. Small, and U. Goodenough.** 2007. Betweenspecies

analysis of short-repeat modules in cell wall and sex-related hydroxyproline-rich glycoproteins of *Chlamydomonas*. Plant Physiol **144:**1813-26.

34. **Lehle, L., S. Strahl, and W. Tanner.** 2006. Protein glycosylation, conserved from yeast to man: a model organism helps elucidate congenital human diseases. Angew Chem Int Ed Engl **45:**6802-18.

35. **Lesage, G., and H. Bussey.** 2006. Cell wall assembly in *Saccharomyces cerevisiae*. Microbiol Mol Biol Rev **70:**317-43.

36. **Liolios, K., N. Tavernarakis, P. Hugenholtz, and N. C. Kyrpides.** 2006. The Genomes On Line Database (GOLD) v.2: a monitor of genome projects worldwide. Nucleic Acids Res **34:**D332-4.

37. **Lipke, P. N., and J. Kurjan.** 1992. Sexual agglutination in budding yeasts: structure, function, and regulation of adhesion glycoproteins. Microbiol Rev **56:**180-94.

38. **Lipke, P. N., and R. Ovalle.** 1998. Cell wall architecture in yeast: new structure and new challenges. J Bacteriol **180:**3735-40.

39. **Liti, G., and E. J. Louis.** 2005. Yeast evolution and comparative genomics. Annu Rev Microbiol **59:**135-53.

40. **Lopez-Ribot, J. L., H. M. Alloush, B. J. Masten, and W. L. Chaffin.** 1996. Evidence for presence in the cell wall of *Candida albicans* of a protein related to the Hsp70 family. Infect Immun **64:**3333-40.

41. **Lopez-Ribot, J. L., and W. L. Chaffin.** 1996. Members of the Hsp70 family of proteins in the cell wall of *Saccharomyces cerevisiae*. J Bacteriol **178:**4724-6.

42. **Lu, C. F., J. Kurjan, and P. N. Lipke.** 1994. A pathway for cell wall anchorage of *Saccharomyces cerevisiae* alpha-agglutinin. Mol Cell Biol **14:**4825-33.

43. **Maia, J. C.** 1994. Hexosamine and cell wall biogenesis in the aquatic fungus *Blastocladiella emersonii*. Faseb J **8:**848-53.

44. **Mort-Bontemps, M., L. Gay, and M. Fevre.** 1997. CHS2, a chitin synthase gene from the oomycete *Saprolegnia monoica*. Microbiology **143 (Pt 6):**2009-20.

45. **Mrsa, V., M. Ecker, S. Strahl-Bolsinger, M. Nimtz, L. Lehle, and W. Tanner.** 1999. Deletion of new covalently linked cell wall glycoproteins alters the electrophoretic mobility of phosphorylated wall components of *Saccharomyces cerevisiae*. J Bacteriol **181:**3076-86.

46. **Ngondi-Ekome, J., F. Thiebault, J. M. Strub, A. Van Dorsselaer, R. Bonaly, C. Contino-Pepin, M. Wathier, B. Pucci, and J. Coulon.** 2003. Study on agglutinating factors from flocculent *Saccharomyces cerevisiae* strains. Biochimie **85:**133-43.

47. **Nickerson, W. J.** 1974. Chemical composition of cell walls and membranes of yeasts. Ann N Y Acad Sci **235:**105-8.

48. **Nickerson, W. J., and G. Falcone.** 1956. Enzymatic reduction of disulfide bonds in cell wall protein of baker's yeast. Science **124:**318-9.

49. **Romov, P. A., F. Li, P. N. Lipke, S. L. Epstein, and W. G. Qiu.** 2006. Comparative genomics reveals long, evolutionarily conserved, low-complexity islands in yeast proteins. J Mol Evol **63:**415-25.

50. **Schaffer, A. A., L. Aravind, T. L. Madden, S. Shavirin, J. L. Spouge, Y. I. Wolf, E. V. Koonin, and S. F. Altschul.** 2001. Improving the accuracy of PSIBLAST protein database searches with composition-based statistics and other refinements. Nucleic Acids Res **29:**2994-3005.

51. **Simpson, A. G., and A. J. Roger.** 2004. The real 'kingdoms' of eukaryotes. Curr Biol **14:**R693-6.

52. **Southard, S. B., C. A. Specht, C. Mishra, J. Chen-Weiner, and P. W. Robbins.** 1999. Molecular analysis of the *Candida albicans* homolog of *Saccharomyces cerevisiae MNN9*, required for glycosylation of cell wall mannoproteins. J Bacteriol **181:**7439-48.

53. **Sprague, G. F., Jr.** 1991. Genetic exchange between kingdoms. Curr Opin Genet Dev **1:**530-3.

54. **Steenkamp, E. T., J. Wright, and S. L. Baldauf.** 2006. The protistan origins of animals and fungi. Mol Biol Evol **23:**93-106.

55. **Van Dellen, K., S. K. Ghosh, P. W. Robbins, B. Loftus, and J. Samuelson.** 2002. *Entamoeba histolytica* lectins contain unique 6-Cys or 8-Cys chitin-binding domains. Infect Immun **70:**3259-63.

56. **Verstrepen, K. J., A. Jansen, F. Lewitter, and G. R. Fink.** 2005. Intragenic tandem repeats generate functional variability. Nat Genet **37:**986-90.

57. **Ward, H. D., J. Alroy, B. I. Lev, G. T. Keusch, and M. E. Pereira.** 1988. Biology of *Giardia lamblia*. Detection of N-acetyl-D-glucosamine as the only

surface saccharide moiety and identification of two distinct subsets of trophozoites by lectin binding. J Exp Med **167:**73-88.

58. **Weig, M., L. Jansch, U. Gross, C. G. De Koster, F. M. Klis, and P. W. De Groot.** 2004. Systematic identification *in silico* of covalently bound cell wall proteins and analysis of protein-polysaccharide linkages of the human pathogen *Candida glabrata*. Microbiology **150:**3129-44.

59. **West, C. M.** 2003. Comparative analysis of spore coat formation, structure, and function in *Dictyostelium*. Int Rev Cytol **222:**237-93.

60. **Yarbrough, P. O., M. A. Hayden, L. A. Dunn, P. S. Vermersch, M. R. Klass, and R. M. Hecht.** 1987. The glyceraldehyde-3-phosphate dehydrogenase gene family in the nematode, *Caenorhabditis elegans*: isolation and characterization of one of the genes. Biochim Biophys Acta **908:**21-33.

61. **Yin, Q. Y., P. W. de Groot, H. L. Dekker, L. de Jong, F. M. Klis, and C. G. de Koster.** 2005. Comprehensive proteomic analysis of *Saccharomyces cerevisiae* cell walls: identification of proteins covalently attached via glycosylphosphatidylinositol remnants or mild alkali-sensitive linkages. J Biol Chem **280:**20894-901.

62. **Yona, G., N. Linial, and M. Linial.** 2000. ProtoMap: automatic classification of protein sequences and hierarchy of protein families. Nucleic Acids Res **28:**49-55.

63. **Yoshida, M., N. Sakuragi, K. Kondo, and E. Tanesaka.** 2006. Cleavage with phospholipase of the lipid anchor in the cell adhesion molecule, csA, from *Dictyostelium discoideum*. Comp Biochem Physiol B Biochem Mol Biol **143:**138-44.

64. **Yu, Y. K., and S. F. Altschul.** 2005. The construction of amino acid substitution matrices for the comparison of proteins with non-standard compositions. Bioinformatics **21:**902-11.

Figure Legends

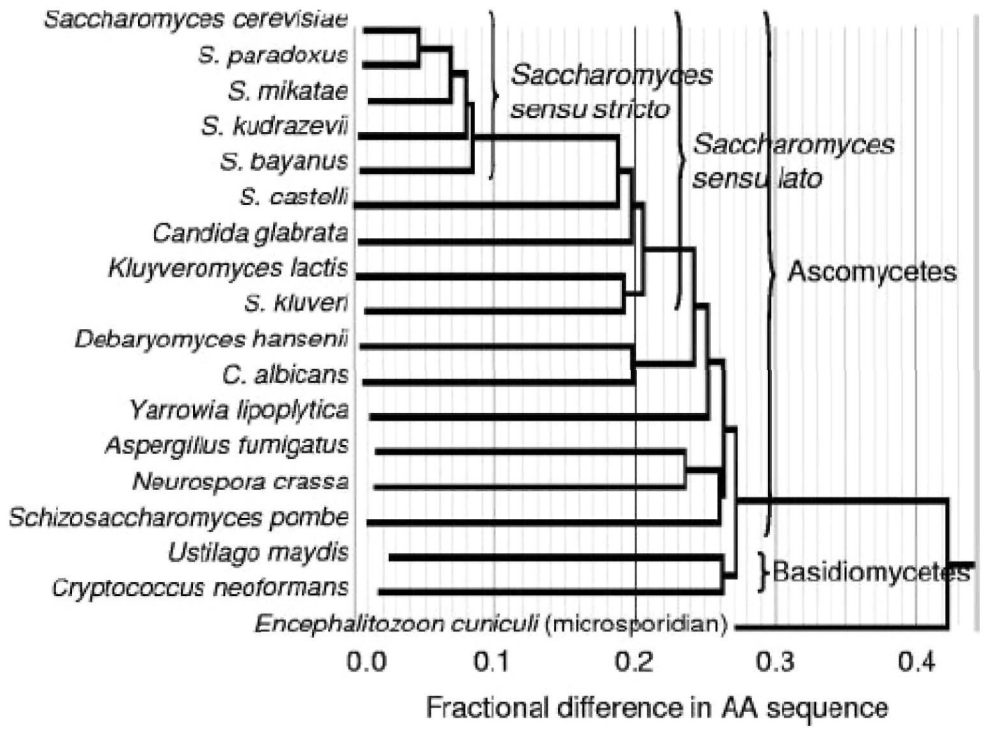

Fig. 1. Neighbor-joining phylogenetic tree based on amino acid substitution rates for all orthologs in 18 fungal genomes.

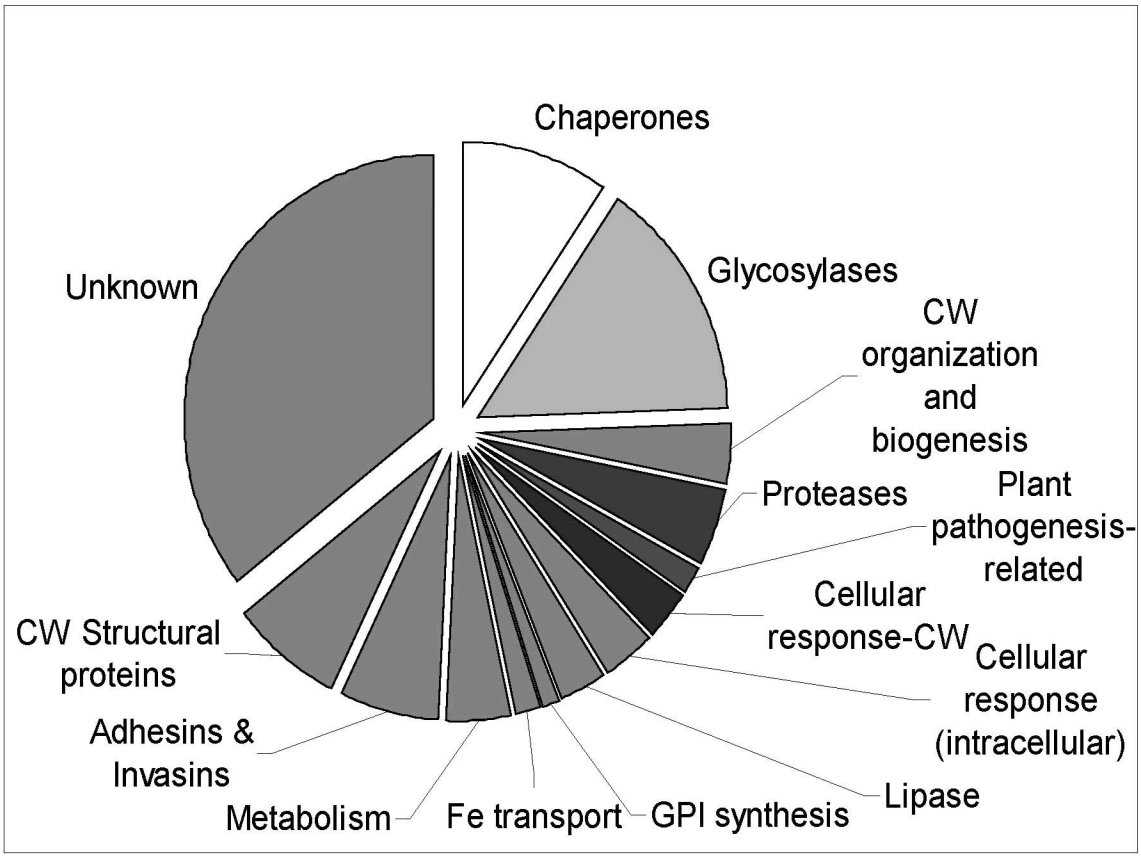

Fig. 2. Cellular function groups of *S. cerevisiae* cell wall-related proteins. The size of each sector is proportional to the number of ORFs in that group. "CW" is cell wall.

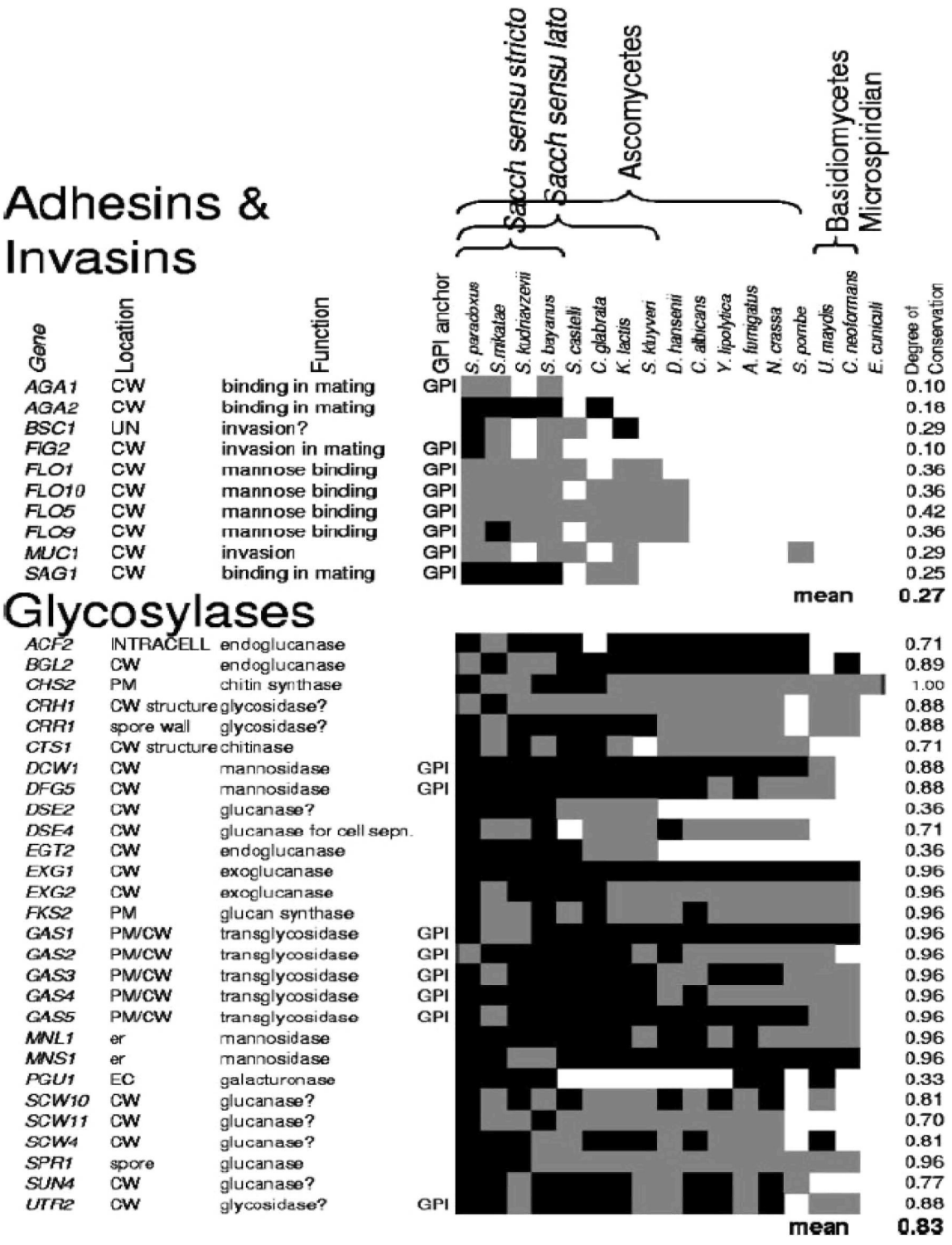

Fig. 3. Presence of homologs and orthologs for two functional groups of wall-related genes in 17 fungi. The query sequences were *S. cerevisiae* cell wall-related

proteins in the Adhesin and Glycosylase functional groups. Presence of one or more homologs with e $<10^{-10}$ is shown as a gray square, and presence of an ortholog is shown as a black square (38). On the right is the total conservation score for each gene, with the group mean conservation score at the bottom of the group. Cellular locations are: CW, cell wall; PM, plasma membrane; ER, endoplasmic reticulum; UN, unknown. The column labeled "GPI anchor" shows those proteins predicted to have GPI anchors.

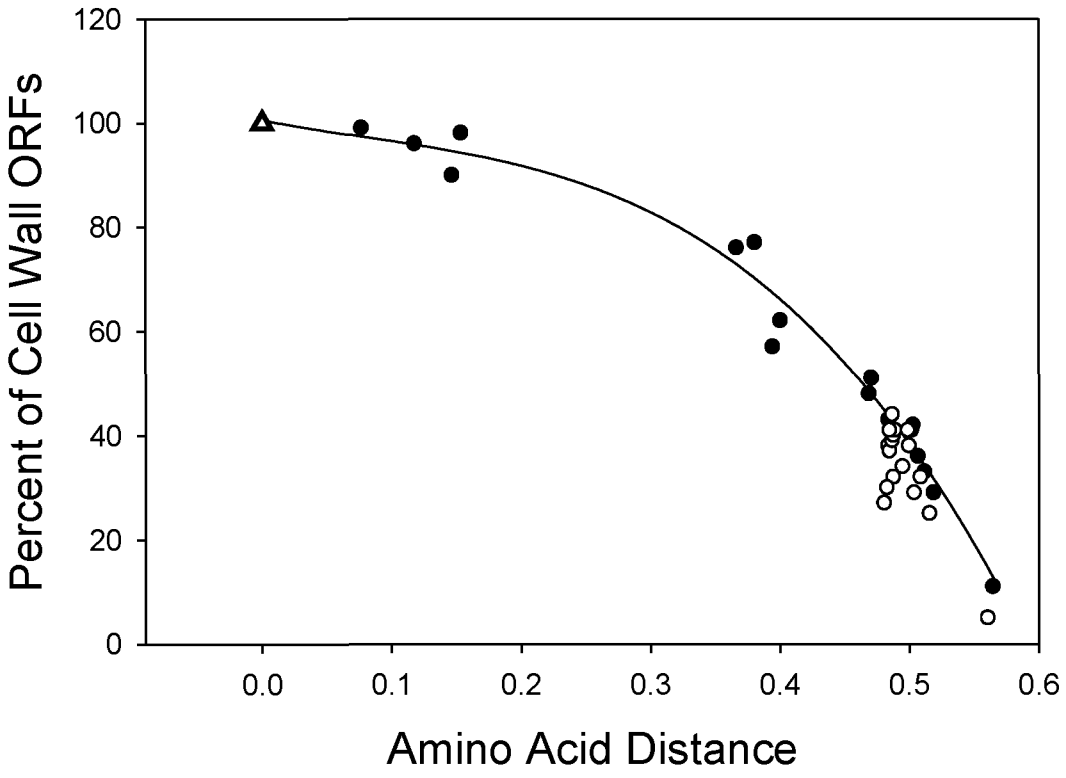

Fig. 4. Relationship between homolog identification and phylogenetic distance. For each tested genome, the fraction of query sequences that had a homolog is plotted against phylogenetic distance derived from Fig. 1. The query protein sets were (●) all *S. cerevisiae* cell wall-related proteins, or (○) *Y. lipolytica* GPI-anchored proteins. The top, left-most point (△) corresponds to the searches of *S. cerevisiae* queries against the *S. cerevisiae* genome and to searches of the *Y. lipolytica* queries against the *Y. lipolytica* genome.

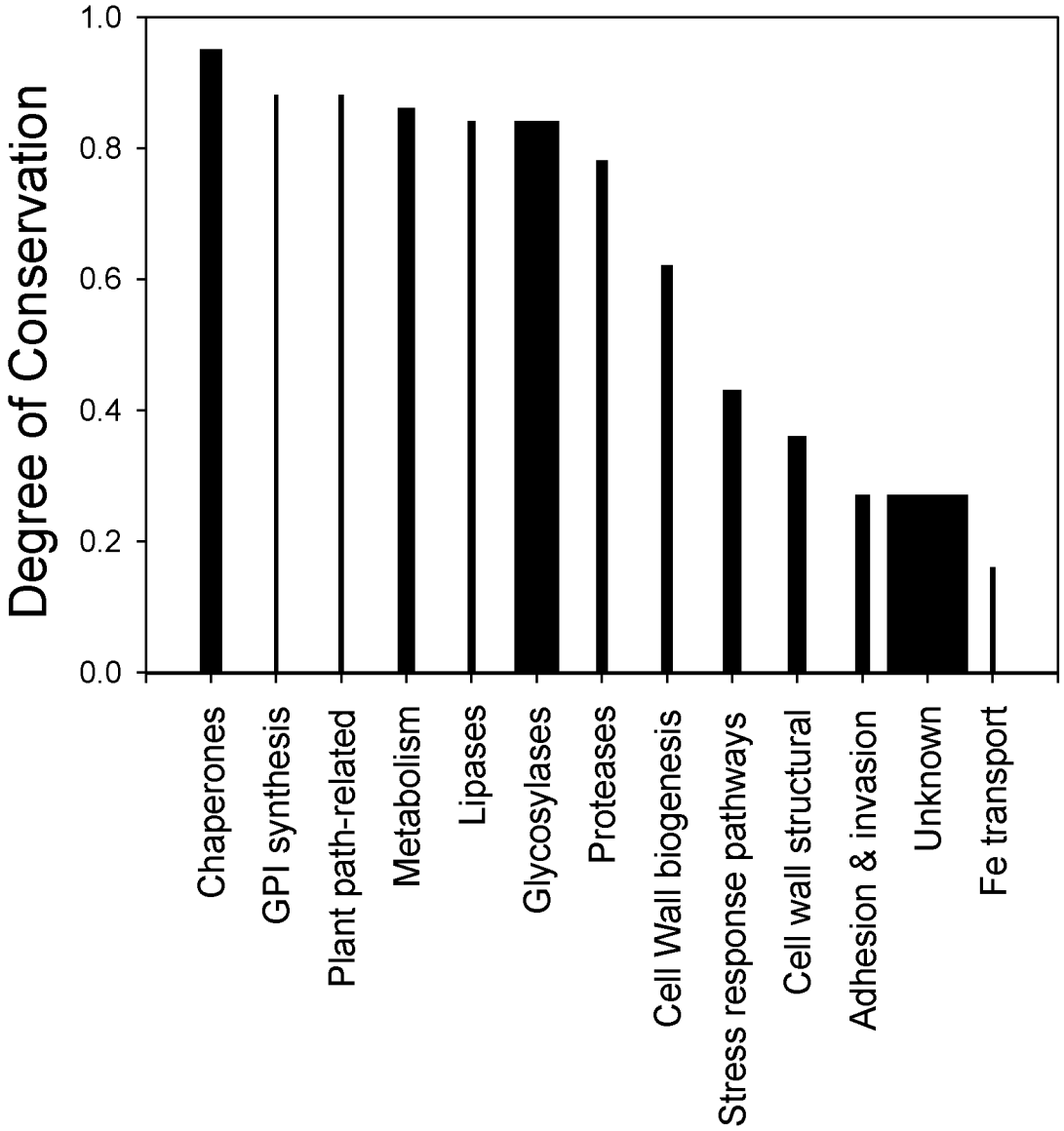

Fig. 5. Degree of Conservation for 13 functional groups of cell wall proteins. The width of each bar is proportional to the number of ORFs in that group. Degree of conservation was calculated as described in Methods.

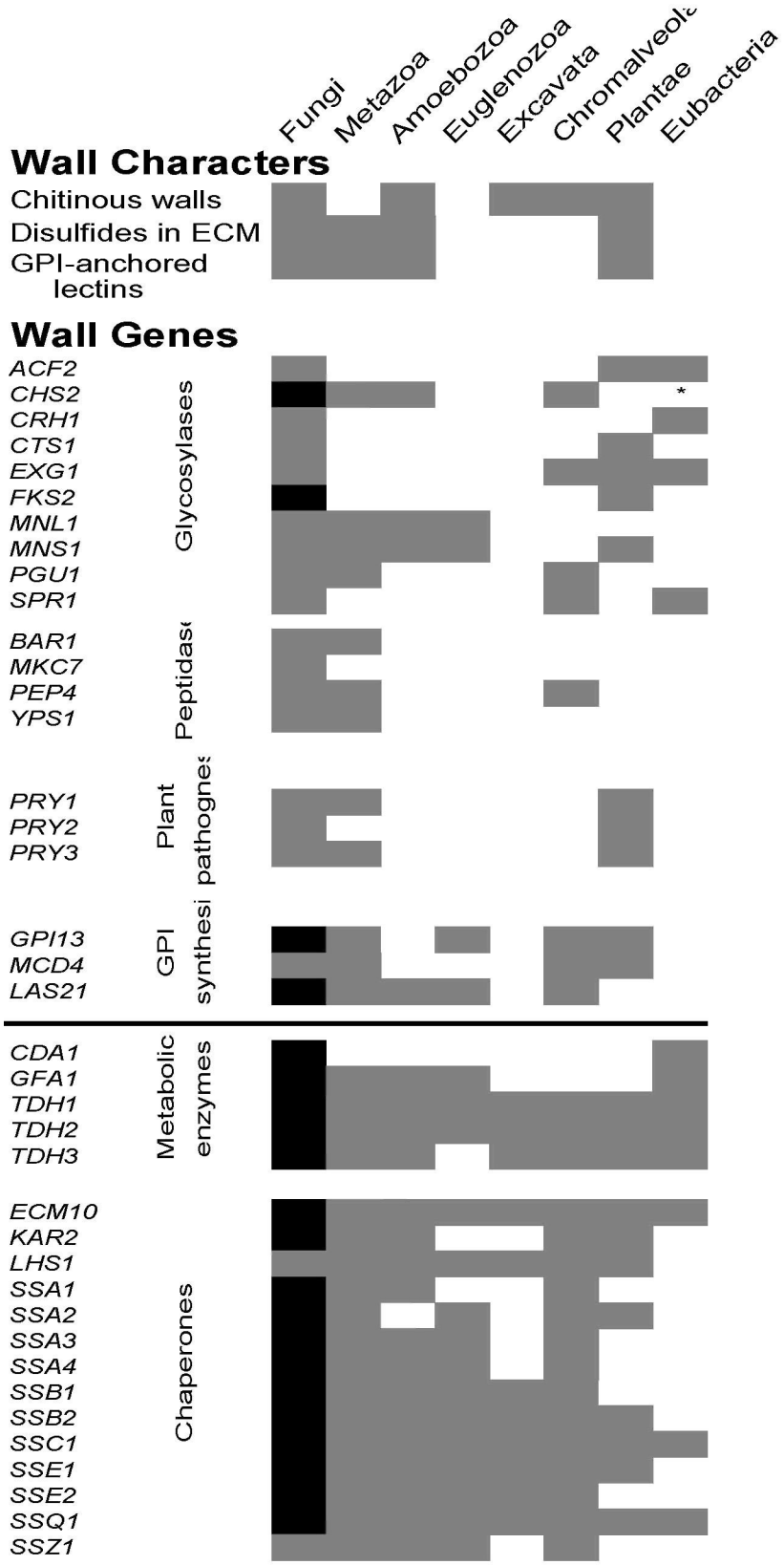
Fungi
Metazoa
Amoebozoa
Euglenozoa
Excavata
Chromalveola
Plantae
Eubacteria
Wall Characters
Chitinous walls
Disulfides in ECM
GPI-anchored lectins
Wall Genes
ACF2
CHS2
CRH1
CTS1
EXG1
FKS2
MNL1
MNS1
PGU1
SPR1
Glycosylases
BAR1
MKC7
PEP4
YPS1
Peptidases
PRY1
PRY2
PRY3
Plant pathogenesis
GPI13
MCD4
LAS21
GPI synthesis
CDA1
GFA1
TDH1
TDH2
TDH3
Metabolic enzymes
ECM10
KAR2
LHS1
SSA1
SSA2
SSA3
SSA4
SSB1
SSB2
SSC1
SSE1
SSE2
SSQ1
SSZ1
Chaperones

Fig. 6. Occurrence of wall characters and homologs of *S. cerevisiae* cell wall-related ORFs in other kingdoms and domains. Top: conserved characters in walls are shown as grey shading where known to occur. Note that absence of shading is not definitive, because it may represent lack of data. Extracellular matrix is abbreviated "ECM." Bottom: Each *S. cerevisiae* cell wall-related ORF was the query sequence for a BLAST-gtQ search through the NR database. Homologs (grey squares) were characterized by kingdom and domain. Black squares designate ORFs present in each of the 18 fungal genomes searched and in all fungal phyla except Glomeromycota. * There was a single bacterial *CHS2* homolog, and its uniqueness and *e*-value, $e<10^{-18}$, imply that it derives from horizontal gene transfer. A horizontal line sets off the two classes of proteins that are primarily intracellular.

Conclusion

To reduce the chances of getting random low-complexity sequence alignments, we modified scoring matrices, such as BLOSUM62, to compensate for deviations in amino acid composition from average amino acid composition. These matrices reduce the score for a pair of aligned residues in proportion to the likelihood that they will align randomly in sequences of similar composition to the query sequence. For both low- and high-complexity query sequences, this modification significantly increases discrimination (the ability to distinguish similar sequences from randomized sequences and non-homologous proteins with similar amino acid composition), with only minimal reduction in sensitivity (the ability to find a high number of homologs). As a result, alignments relying on such scoring matrices significantly decrease the probability of generating false HSPs (high-scoring pairs). The best of our techniques is the gtQ matrices, which were used in all subsequent studies in this thesis . These matrices generate statistically significant, homology-driven alignments of low-complexity segments necessary for structural and evolutionary studies of the low-complexity portion of the proteome. These matrices reduce the score for a pair of aligned residues in proportion to the likelihood that they align randomly in sequences of similar composition to the query sequence.

The distribution of the extracellular proteins seems to be distinct from the distribution of cytoplasmic proteins. From an evolutionary perspective, the amino acid distributions of extracellular proteins are under different selections pressures from the intracellular proteins. The extracellular proteins must accumulate amino acids that are tolerant of environmental conditions such as pH, oxidative and reductive chemicals, and

varying osmolarity. The twenty amino acids present in eukaryotes have a definite threshold for functionality. In cell wall proteins we see an increase in the frequency of valine, alanine, isoleucine, lysine, proline, serine, and threonine, comprising a set of small hydrophobic or hydrophilic amino acids in an attempt to optimize the relationship between functionality of the protein and the capacity to resist adverse solvent conditions.

Extracellular proteins require flexibility to generate a wide range of binding affinities to the high number of possible ligands to the external structures the extracellular proteins may come into contact with. The more defined a protein tertiary structure needs to be, the more information (higher entropy) its protein sequence must possess. In protein sequences, higher informational entropy is accomplished by having more different amino acids dictate the most confined structure possible, as required by the function of the protein. Extracellelular proteins are selected in many instances to have sections of low informational entropy, that is, to use a few amino acids in a given section of the protein, to have a less defined 3D structure and thus more flexibility, which can be selected for at the periphery of the cell.

For cell wall proteins, the process of going from the place where they are made, or translated from mRNA, to being linked in the carbohydrate network requires information to be embedded in the amino acid sequence of the proteins. The N-terminus contains a hydrophobic sequence that is bound by the signal recognition particle and brought to the rough endoplasmic reticulum, where translation is completed and the N-terminal hydrophobic sequence is cleaved, releasing the rest of the protein into the lumen of the endoplasmic reticulum. For proteins with a hydrophobic sequence at the C-terminus, the

hydrophobic sequence is replaced with a glycosyl phosphatidyl inositol lipid anchor (GPI) in the ER. Signals for N-and O-glycoslyation at asparagine and serine or threonine residues, respectively, allow for glycosylases to start to add the sugar mannose to proteins in the endoplasmic reticulum and to end the addition or modification of sugars in the golgi. Once golgi vesicles fuse with the cell membrane, the GPI may be cleaved from the protein, and the protein is bound to usually β1,6 glucan- becoming part of the cell wall. The enzymes that perform the cleaving of the GPI and/or the translocation of the remaining protein into the carbohydrate network have not been identified.

The 171 cell wall related *S. cerevisiae* proteins were used as queries in BLAST with gtQ searches, to identify homologs in 17 other fungal genomes. We grouped sets of homologous proteins by function, and catalogued the patterns of occurrence. There was an unexpected pattern: the enzymes and other proteins mediating wall biosynthesis (e.g., the glycosylases) are highly conserved; indeed many of them have homologs in other eukaryote kingdoms. In contrast, the proteins resident in the walls (e.g. the adhesins) are highly lineage specific. Other lineage-specific groups include wall structural proteins and many wall-localized proteins of unknown function. These other groups, like the adhesins, are low complexity and have frequently repeated modular sequences. Thus, wall-resident proteins in *S. cerevisiae* share an unusual set of sequence characters (motifs and low-complexity sequences), and their pattern of occurrence in only closely-related species shows that they diverge rapidly. The *Saccharomyces sensu stricto* species have homologs for 90% of the 171 *S. cerevisiae* cell wall-related ORFs. *Saccharomyces sensu lato* yeasts have homologs for 70% of the *S. cerevisiae* cell wall-related proteins; filamentous

ascomycetes had fewer. The pattern is similar when GPI-anchored proteins from the phylogenetically-distant yeast *Yarrowia lipolytica* were used as the initial query sequences instead of *S. cerevisiae*: wall biosynthetic proteins were conserved, and putative adhesins and wall structural proteins were not. The conservation of wall biosynthetic genes across eukaryote kingdoms supports Cavalier-Smith's hypothesis that chitinous walls are ancestral to divergence of the eukaryote kingdoms (Cavalier-Smith 2001).

Only a few cell wall related proteins are conserved across fungi. They are chaperones and carbohydrated-modifying enzymes, which are biochemically important to intracellular activities. But peripheral cell wall proteins are not conserved; these proteins are not essential to the survival of fungi *per se*. Evolutionarily, these non-conserved proteins play an important part in the diversification and niche searching capabilities of fungi species. Through evolutionary and meiotic combinatorics, segments of different cell wall proteins genes recombine among different genes (Verstrepen and Fink 2006). The recombined gene variants may pose an environmental advantage over other variants that do not contain the recombined gene, and thus the recombined variants are selected for. The distributionof cell-wall related proteins is consistent with the phylogenetic distribution of fungi. The closer two fungi are evolutionarily related, the more cell wall proteins they have in common; therefore, studying the divergence of cell wall related proteins can lead to a clearer understanding of species similarities or differences where differences in intracellular proteins do not suffice. The consistency in the phylogenetic profiles among the fungal genomes implies that CWPs are an important factor in the

production of new species or strains of fungi. The small cohort of conserved CWPs in fungi suggests that the majority of CWPs assist fungal cells in optimizing the differences in their habitats and are not vital to the survival of the cell *per se*, but they are important in creating efficient niches where the fungal cells can thrive.

BIBLIOGRAPHY for General Introduction and Conclusion.

ADAMS, M.D., CELNIKER, S.E., HOLT, R.A., EVANS, C.A., GOCAYNE, J.D., AMANATIDES, P.G., SCHERER, S.E., LI, P.W., HOSKINS, R.A., GALLE, R.F., GEORGE, R.A., LEWIS, S.E., RICHARDS, S., ASHBURNER, M., HENDERSON, S.N., SUTTON, G.G., WORTMAN, J.R., YANDELL, M.D., ZHANG, Q., CHEN, L.X., BRANDON, R.C., ROGERS, Y.H., BLAZEJ, R.G., CHAMPE, M., PFEIFFER, B.D., WAN, K.H., DOYLE, C., BAXTER, E.G., HELT, G., NELSON, C.R., GABOR, G.L., ABRIL, J.F., AGBAYANI, A., AN, H.J., ANDREWS-PFANNKOCH, C., BALDWIN, D., BALLEW, R.M., BASU, A., BAXENDALE, J., BAYRAKTAROGLU, L., BEASLEY, E.M., BEESON, K.Y., BENOS, P.V., BERMAN, B.P., BHANDARI, D., BOLSHAKOV, S., BORKOVA, D., BOTCHAN, M.R., BOUCK, J., BROKSTEIN, P., BROTTIER, P., BURTIS, K.C., BUSAM, D.A., BUTLER, H., CADIEU, E., CENTER, A., CHANDRA, I., CHERRY, J.M., CAWLEY, S., DAHLKE, C., DAVENPORT, L.B., DAVIES, P., DE PABLOS, B., DELCHER, A., DENG, Z., MAYS, A.D., DEW, I., DIETZ, S.M., DODSON, K., DOUP, L.E., DOWNES, M., DUGAN-ROCHA, S., DUNKOV, B.C., DUNN, P., DURBIN, K.J., EVANGELISTA, C.C., FERRAZ, C., FERRIERA, S., FLEISCHMANN, W., FOSLER, C., GABRIELIAN, A.E., GARG, N.S., GELBART, W.M., GLASSER, K., GLODEK, A., GONG, F., GORRELL, J.H., GU, Z., GUAN, P., HARRIS, M., HARRIS, N.L., HARVEY, D., HEIMAN, T.J., HERNANDEZ, J.R., HOUCK, J., HOSTIN, D., HOUSTON, K.A., HOWLAND, T.J., WEI, M.H., IBEGWAM, C., JALALI, M., KALUSH, F., KARPEN, G.H., KE, Z., KENNISON, J.A., KETCHUM, K.A., KIMMEL, B.E., KODIRA, C.D., KRAFT, C., KRAVITZ, S., KULP, D., LAI, Z., LASKO, P., LEI, Y., LEVITSKY, A.A., LI, J., LI, Z., LIANG, Y., LIN, X., LIU, X., MATTEI, B., MCINTOSH, T.C., MCLEOD, M.P., MCPHERSON, D., MERKULOV, G., MILSHINA, N.V., MOBARRY, C., MORRIS, J., MOSHREFI, A., MOUNT, S.M., MOY, M., MURPHY, B., MURPHY, L., MUZNY, D.M., NELSON, D.L., NELSON, D.R., NELSON, K.A., NIXON, K., NUSSKERN, D.R., PACLEB, J.M., PALAZZOLO, M., PITTMAN, G.S., PAN, S., POLLARD, J., PURI, V., REESE, M.G., REINERT, K., REMINGTON, K., SAUNDERS, R.D., SCHEELER, F., SHEN, H., SHUE, B.C., SIDEN-KIAMOS, I., SIMPSON, M., SKUPSKI, M.P., SMITH, T., SPIER, E., SPRADLING, A.C., STAPLETON, M., STRONG, R., SUN, E., SVIRSKAS, R., TECTOR, C., TURNER, R., VENTER, E., WANG, A.H., WANG, X., WANG, Z.Y., WASSARMAN, D.A., WEINSTOCK, G.M., WEISSENBACH, J., WILLIAMS, S.M., WOODAGET, WORLEY, K.C., WU, D., YANG, S., YAO, Q.A., YE, J., YEH, R.F., ZAVERI, J.S., ZHAN, M., ZHANG, G., ZHAO, Q., ZHENG, L., ZHENG, X.H., ZHONG, F.N., ZHONG, W., ZHOU, X., ZHU, S., ZHU, X., SMITH, H.O., GIBBS, R.A., MYERS, E.W., RUBIN, G.M. and VENTER, J.C., 2000. The genome sequence of Drosophila melanogaster. *Science (New York, N.Y.),* **287**(5461), pp. 2185-2195.

ALLEN, J.E. and SALZBERG, S.L., 2005. JIGSAW: integration of multiple sources of evidence for gene prediction. *Bioinformatics (Oxford, England),* **21**(18), pp. 3596-3603.

ALTSCHUL, S.F., GISH, W., MILLER, W., MYERS, E.W. and LIPMAN, D.J., 1990.

Basic local alignment search tool. *J Mol Biol,* **215**(3), pp. 403-10.

ALTSCHUL, S.F., MADDEN, T.L., SCHAFFER, A.A., ZHANG, J., ZHANG, Z., MILLER, W. and LIPMAN, D.J., 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res,* **25**(17), pp. 3389-402.

APARICIO, S., CHAPMAN, J., STUPKA, E., PUTNAM, N., CHIA, J.M., DEHAL, P., CHRISTOFFELS, A., RASH, S., HOON, S., SMIT, A., GELPKE, M.D., ROACH, J., OH, T., HO, I.Y., WONG, M., DETTER, C., VERHOEF, F., PREDKI, P., TAY, A., LUCAS, S., RICHARDSON, P., SMITH, S.F., CLARK, M.S., EDWARDS, Y.J., DOGGETT, N., ZHARKIKH, A., TAVTIGIAN, S.V., PRUSS, D., BARNSTEAD, M., EVANS, C., BADEN, H., POWELL, J., GLUSMAN, G., ROWEN, L., HOOD, L., TAN, Y.H., ELGAR, G., HAWKINS, T., VENKATESH, B., ROKHSAR, D. and BRENNER, S., 2002. Whole-genome shotgun assembly and analysis of the genome of Fugu rubripes. *Science (New York, N.Y.),* **297**(5585), pp. 1301-1310.

BELLMAN, R. and KALABA, R., 1957. Dynamic Programming and Statistical Communication Theory. *Proceedings of the National Academy of Sciences of the United States of America,* **43**(8), pp. 749-751.

BON, E., CASAREGOLA, S., BLANDIN, G., LLORENTE, B., NEUVEGLISE, C., MUNSTERKOTTER, M., GULDENER, U., MEWES, H.W., VAN HELDEN, J., DUJON, B. and GAILLARDIN, C., 2003. Molecular evolution of eukaryotic genomes: hemiascomycetous yeast spliceosomal introns. *Nucleic acids research,* **31**(4), pp. 1121-1135.

CABIB, E., BLANCO, N., GRAU, C., RODRIGUEZ-PENA, J.M. and ARROYO, J., 2007. Crh1p and Crh2p are required for the cross-linking of chitin to beta(1-6)glucan in the Saccharomyces cerevisiae cell wall. *Molecular microbiology,* **63**(3), pp. 921-935.

CABIB, E., ROH, D.H., SCHMIDT, M., CROTTI, L.B. and VARMA, A., 2001. The yeast cell wall and septum as paradigms of cell growth and morphogenesis. *The Journal of biological chemistry,* **276**(23), pp. 19679-19682.

CAENORHABDITIS ELEGANS SEQUENCING CONSORTIUM, 1998. Genome sequence of the nematode C. elegans: a platform for investigating biology. *Science (New York, N.Y.),* **282**(5396), pp. 2012-2018.

CAPPELLARO, C., MRSA, V. and TANNER, W., 1998. New potential cell wall glucanases of Saccharomyces cerevisiae and their involvement in mating. *Journal of Bacteriology,* **180**(19), pp. 5030-5037.

CARO, L.H., TETTELIN, H., VOSSEN, J.H., RAM, A.F., VAN DEN ENDE, H. and KLIS, F.M., 1997. In silicio identification of glycosyl-phosphatidylinositol-anchored plasma-membrane and cell wall proteins of Saccharomyces cerevisiae. *Yeast (Chichester, England),* **13**(15), pp. 1477-1489.

CAVALIERI, D., MCGOVERN, P.E., HARTL, D.L., MORTIMER, R. and POLSINELLI, M., 2003. Evidence for S. cerevisiae fermentation in ancient wine. *Journal of Molecular Evolution,* **57 Suppl 1**, pp. S226-32.
CAVALIER-SMITH,T., 2001. What are Fungi?, p. 1-37. *In* D. J. McLaughlin, E. C.McLaughlin, and P. A. Lemke (ed.), The Mycota. Springer, Berlin.

CHAFFIN, W.L., LOPEZ-RIBOT, J.L., CASANOVA, M., GOZALBO, D. and MARTINEZ, J.P., 1998. Cell wall and secreted proteins of Candida albicans: identification, function, and expression. *Microbiology and molecular biology reviews : MMBR,* **62**(1), pp. 130-180.

CHEN, M.H., SHEN, Z.M., BOBIN, S., KAHN, P.C. and LIPKE, P.N., 1995. Structure of Saccharomyces cerevisiae alpha-agglutinin. Evidence for a yeast cell wall protein with multiple immunoglobulin-like domains with atypical disulfides. *The Journal of biological chemistry,* **270**(44), pp. 26168-26177.

CHERRY, J.M., BALL, C., WENG, S., JUVIK, G., SCHMIDT, R., ADLER, C., DUNN, B., DWIGHT, S., RILES, L., MORTIMER, R.K. and BOTSTEIN, D., 1997. Genetic and physical maps of Saccharomyces cerevisiae. *Nature,* **387**(6632 Suppl), pp. 67-73.

CHIMPANZEE SEQUENCING AND ANALYSIS CONSORTIUM, 2005. Initial sequence of the chimpanzee genome and comparison with the human genome. *Nature,* **437**(7055), pp. 69-87.

CHO, R.J., CAMPBELL, M.J., WINZELER, E.A., STEINMETZ, L., CONWAY, A., WODICKA, L., WOLFSBERG, T.G., GABRIELIAN, A.E., LANDSMAN, D., LOCKHART, D.J. and DAVIS, R.W., 1998. A genome-wide transcriptional analysis of the mitotic cell cycle. *Molecular cell,* **2**(1), pp. 65-73.

CHU, S., DERISI, J., EISEN, M., MULHOLLAND, J., BOTSTEIN, D., BROWN, P.O. and HERSKOWITZ, I., 1998. The transcriptional program of sporulation in budding yeast. *Science (New York, N.Y.),* **282**(5389), pp. 699-705.

CLIFTEN, P., SUDARSANAM, P., DESIKAN, A., FULTON, L., FULTON, B., MAJORS, J., WATERSTON, R., COHEN, B.A. and JOHNSTON, M., 2003. Finding functional features in Saccharomyces genomes by phylogenetic footprinting. *Science (New York, N.Y.),* **301**(5629), pp. 71-76.

COHEN, B.D., SERTIL, O., ABRAMOVA, N.E., DAVIES, K.J. and LOWRY, C.V., 2001. Induction and repression of DAN1 and the family of anaerobic mannoprotein genes in Saccharomyces cerevisiae occurs through a complex array of regulatory sites. *Nucleic acids research,* **29**(3), pp. 799-808.

COLMAN-LERNER, A., CHIN, T.E. and BRENT, R., 2001. Yeast Cbk1 and Mob2 activate daughter-specific genetic programs to induce asymmetric cell fates. *Cell,* **107**(6), pp. 739-750.

CORMACK, B., 2004. Can you adhere me now? Good. *Cell,* **116**(3), pp. 353-354.

DAYHOFF, M.O., SCHWARTZ, R.M. and ORCUTT, B.C., 1978. *Atlas of Protein Sequence and Structure Vol. 5, suppl. 3.* suppl. 3. edn. Washington, D.C.: National Biomedical Research Foundation.

DE GROOT, P.W., HELLINGWERF, K.J. and KLIS, F.M., 2003. Genome-wide identification of fungal GPI proteins. *Yeast,* **20**(9), pp. 781-96.

DE NOBEL, J.G. and BARNETT, J.A., 1991. Passage of molecules through yeast cell walls: a brief essay-review. *Yeast (Chichester, England),* 7(4), pp. 313-323.

DE NOBEL, J.G., KLIS, F.M., MUNNIK, T., PRIEM, J. and VAN DEN ENDE, H., 1990. An assay of relative cell wall porosity in Saccharomyces cerevisiae, Kluyveromyces lactis and Schizosaccharomyces pombe. *Yeast (Chichester, England),* **6**(6), pp. 483-490.

DE NOBEL, J.G., KLIS, F.M., PRIEM, J., MUNNIK, T. and VAN DEN ENDE, H., 1990a. The glucanase-soluble mannoproteins limit cell wall porosity in Saccharomyces cerevisiae. *Yeast,* **6**(6), pp. 491-9.

DE NOBEL, J.G., KLIS, F.M., PRIEM, J., MUNNIK, T. and VAN DEN ENDE, H., 1990b. The glucanase-soluble mannoproteins limit cell wall porosity in Saccharomyces cerevisiae. *Yeast (Chichester, England),* **6**(6), pp. 491-499.

DE NOBEL, J.G., KLIS, F.M., RAM, A., VAN UNEN, H., PRIEM, J., MUNNIK, T. and VAN DEN ENDE, H., 1991. Cyclic variations in the permeability of the cell wall of Saccharomyces cerevisiae. *Yeast (Chichester, England),* 7(6), pp. 589-598.

EDDY, S.R., 1998. Profile hidden Markov models. *Bioinformatics (Oxford, England),* **14**(9), pp. 755-763.

EGEA, P.F., STROUD, R.M. and WALTER, P., 2005. Targeting proteins to membranes: structure of the signal recognition particle. *Current opinion in structural biology,* **15**(2), pp. 213-220.

FLEISCHMANN, R.D., ADAMS, M.D., WHITE, O., CLAYTON, R.A., KIRKNESS, E.F., KERLAVAGE, A.R., BULT, C.J., TOMB, J.F., DOUGHERTY, B.A. and MERRICK, J.M., 1995. Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science (New York, N.Y.),* **269**(5223), pp. 496-512.

FONTAINE, T., SIMENEL, C., DUBREUCQ, G., ADAM, O., DELEPIERRE, M., LEMOINE, J., VORGIAS, C.E., DIAQUIN, M. and LATGE, J.P., 2000. Molecular organization of the alkali-insoluble fraction of Aspergillus fumigatus cell wall. *The Journal of biological chemistry,* **275**(36), pp. 27594-27607.

FORD, R.A., SHAW, J.A. and CABIB, E., 1996. Yeast chitin synthases 1 and 2 consist of a non-homologous and dispensable N-terminal region and of a homologous moiety essential for function. *Molecular & general genetics : MGG,* **252**(4), pp. 420-428.

FOURY, F., ROGANTI, T., LECRENIER, N. and PURNELLE, B., 1998. The complete sequence of the mitochondrial genome of Saccharomyces cerevisiae. *FEBS letters,* **440**(3), pp. 325-331.

FRIEMAN, M.B. and CORMACK, B.P., 2004. Multiple sequence signals determine the distribution of glycosylphosphatidylinositol proteins between the plasma membrane and cell wall in Saccharomyces cerevisiae. *Microbiology (Reading, England),* **150**(Pt 10), pp. 3105-3114.

GANCEDO, C. and DELGADO, M.A., 1984. Isolation and characterization of a mutant from Saccharomyces cerevisiae lacking fructose 1,6-bisphosphatase. *European journal of biochemistry / FEBS,* **139**(3), pp. 651-655.

GEMMILL, T.R. and TRIMBLE, R.B., 1999. Overview of N- and O-linked oligosaccharide structures found in various yeast species. *Biochimica et biophysica acta,* **1426**(2), pp. 227-237.

GIRRBACH, V. and STRAHL, S., 2003. Members of the evolutionarily conserved PMT family of protein O-mannosyltransferases form distinct protein complexes among themselves. *The Journal of biological chemistry,* **278**(14), pp. 12554-12562.

GOFFEAU, A., BARRELL, B.G., BUSSEY, H., DAVIS, R.W., DUJON, B., FELDMANN, H., GALIBERT, F., HOHEISEL, J.D., JACQ, C., JOHNSTON, M., LOUIS, E.J., MEWES, H.W., MURAKAMI, Y., PHILIPPSEN, P., TETTELIN, H. and OLIVER, S.G., 1996. Life with 6000 genes. *Science (New York, N.Y.),* **274**(5287), pp. 546, 563-7.

GOLDMAN, R.C., SULLIVAN, P.A., ZAKULA, D. and CAPOBIANCO, J.O., 1995. Kinetics of beta-1,3 glucan interaction at the donor and acceptor sites of the fungal glucosyltransferase encoded by the BGL2 gene. *European journal of biochemistry / FEBS,* **227**(1-2), pp. 372-378.

GRUN, C.H., HOCHSTENBACH, F., HUMBEL, B.M., VERKLEIJ, A.J., SIETSMA, J.H., KLIS, F.M., KAMERLING, J.P. and VLIEGENTHART, J.F., 2005. The structure of cell wall alpha-glucan from fission yeast. *Glycobiology,* **15**(3), pp. 245-257.

HALME, A., BUMGARNER, S., STYLES, C. and FINK, G.R., 2004. Genetic and epigenetic regulation of the FLO gene family generates cell-surface variation in yeast. *Cell,* **116**(3), pp. 405-415.

HARBISON, C.T., GORDON, D.B., LEE, T.I., RINALDI, N.J., MACISAAC, K.D., DANFORD, T.W., HANNETT, N.M., TAGNE, J.B., REYNOLDS, D.B., YOO, J.,

JENNINGS, E.G., ZEITLINGER, J., POKHOLOK, D.K., KELLIS, M., ROLFE, P.A., TAKUSAGAWA, K.T., LANDER, E.S., GIFFORD, D.K., FRAENKEL, E. and YOUNG, R.A., 2004. Transcriptional regulatory code of a eukaryotic genome. *Nature,* **431**(7004), pp. 99-104.

HARTWELL, L.H., MORTIMER, R.K., CULOTTI, J. and CULOTTI, M., 1973. Genetic Control of the Cell Division Cycle in Yeast: V. Genetic Analysis of cdc Mutants. *Genetics,* **74**(2), pp. 267-286.

HENIKOFF, S. and HENIKOFF, J.G., 1993. Performance evaluation of amino acid substitution matrices. *Proteins,* **17**(1), pp. 49-61.

HENIKOFF, S. and HENIKOFF, J.G., 1992. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A,* **89**(22), pp. 10915-9.

HERSCOVICS, A., 1999. Importance of glycosidases in mammalian glycoprotein biosynthesis. *Biochimica et biophysica acta,* **1473**(1), pp. 96-107.

HICKS, J.B. and HERSKOWITZ, I., 1976. Interconversion of Yeast Mating Types I. Direct Observations of the Action of the Homothallism (HO) Gene. *Genetics,* **83**(2), pp. 245-258.

HIRSCHBERG, C.B. and SNIDER, M.D., 1987. Topography of glycosylation in the rough endoplasmic reticulum and Golgi apparatus. *Annual Review of Biochemistry,* **56**, pp. 63-87.

HIRSCHMAN, J.E., BALAKRISHNAN, R., CHRISTIE, K.R., COSTANZO, M.C., DWIGHT, S.S., ENGEL, S.R., FISK, D.G., HONG, E.L., LIVSTONE, M.S., NASH, R., PARK, J., OUGHTRED, R., SKRZYPEK, M., STARR, B., THEESFELD, C.L., WILLIAMS, J., ANDRADA, R., BINKLEY, G., DONG, Q., LANE, C., MIYASATO, S., SETHURAMAN, A., SCHROEDER, M., THANAWALA, M.K., WENG, S., DOLINSKI, K., BOTSTEIN, D. and CHERRY, J.M., 2006. Genome Snapshot: a new resource at the Saccharomyces Genome Database (SGD) presenting an overview of the Saccharomyces cerevisiae genome. *Nucleic acids research,* **34**(Database issue), pp. D442-5.

HUGHEY, R. and KROGH, A., 1996. Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Computer applications in the biosciences : CABIOS,* **12**(2), pp. 95-107.

INOUE, S.B., TAKEWAKI, N., TAKASUKA, T., MIO, T., ADACHI, M., FUJII, Y., MIYAMOTO, C., ARISAWA, M., FURUICHI, Y. and WATANABE, T., 1995. Characterization and gene cloning of 1,3-beta-D-glucan synthase from Saccharomyces cerevisiae. *European journal of biochemistry / FEBS,* **231**(3), pp. 845-854.

JUNG, U.S. and LEVIN, D.E., 1999. Genome-wide analysis of gene expression regulated

by the yeast cell wall integrity signalling pathway. *Molecular microbiology,* **34**(5), pp. 1049-1057.

KAPTEYN, J.C., TER RIET, B., VINK, E., BLAD, S., DE NOBEL, H., VAN DEN ENDE, H. and KLIS, F.M., 2001. Low external pH induces HOG1-dependent changes in the organization of the Saccharomyces cerevisiae cell wall. *Molecular microbiology,* **39**(2), pp. 469-479.

KAPTEYN, J.C., VAN DEN ENDE, H. and KLIS, F.M., 1999. The contribution of cell wall proteins to the organization of the yeast cell wall. *Biochim Biophys Acta,* **1426**(2), pp. 373-83.

KAPTEYN, J.C., VAN EGMOND, P., SIEVI, E., VAN DEN ENDE, H., MAKAROW, M. and KLIS, F.M., 1999. The contribution of the O-glycosylated protein Pir2p/Hsp150 to the construction of the yeast cell wall in wild-type cells and beta 1,6-glucan-deficient mutants. *Molecular microbiology,* **31**(6), pp. 1835-1844.

KELLIS, M., PATTERSON, N., ENDRIZZI, M., BIRREN, B. and LANDER, E.S., 2003. Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature,* **423**(6937), pp. 241-254.

KLAMMER, A.A., REYNOLDS, S.M., BILMES, J.A., MACCOSS, M.J. and NOBLE, W.S., 2008. Modeling peptide fragmentation with dynamic Bayesian networks for peptide identification. *Bioinformatics (Oxford, England),* **24**(13), pp. i348-56.

KLIS, F.M., BOORSMA, A. and DE GROOT, P.W., 2006. Cell wall construction in Saccharomyces cerevisiae. *Yeast,* **23**(3), pp. 185-202.

KOPECKA, M., FLEET, G.H. and PHAFF, H.J., 1995. Ultrastructure of the cell wall of Schizosaccharomyces pombe following treatment with various glucanases. *Journal of structural biology,* **114**(2), pp. 140-152.

KOVACECH, B., NASMYTH, K. and SCHUSTER, T., 1996. EGT2 gene transcription is induced predominantly by Swi5 in early G1. *Molecular and cellular biology,* **16**(7), pp. 3264-3274.

KOWALSKI, L.R., KONDO, K. and INOUYE, M., 1995. Cold-shock induction of a family of TIP1-related proteins associated with the membrane in Saccharomyces cerevisiae. *Molecular microbiology,* **15**(2), pp. 341-353.

LANDER, E.S., LINTON, L.M., BIRREN, B., NUSBAUM, C., ZODY, M.C., BALDWIN, J., DEVON, K., DEWAR, K., DOYLE, M., FITZHUGH, W., FUNKE, R., GAGE, D., HARRIS, K., HEAFORD, A., HOWLAND, J., KANN, L., LEHOCZKY, J., LEVINE, R., MCEWAN, P., MCKERNAN, K., MELDRIM, J., MESIROV, J.P., MIRANDA, C., MORRIS, W., NAYLOR, J., RAYMOND, C., ROSETTI, M., SANTOS, R., SHERIDAN, A., SOUGNEZ, C., STANGE-THOMANN, N., STOJANOVIC, N.,

SUBRAMANIAN, A., WYMAN, D., ROGERS, J., SULSTON, J., AINSCOUGH, R., BECK, S., BENTLEY, D., BURTON, J., CLEE, C., CARTER, N., COULSON, A., DEADMAN, R., DELOUKAS, P., DUNHAM, A., DUNHAM, I., DURBIN, R., FRENCH, L., GRAFHAM, D., GREGORY, S., HUBBARD, T., HUMPHRAY, S., HUNT, A., JONES, M., LLOYD, C., MCMURRAY, A., MATTHEWS, L., MERCER, S., MILNE, S., MULLIKIN, J.C., MUNGALL, A., PLUMB, R., ROSS, M., SHOWNKEEN, R., SIMS, S., WATERSTON, R.H., WILSON, R.K., HILLIER, L.W., MCPHERSON, J.D., MARRA, M.A., MARDIS, E.R., FULTON, L.A., CHINWALLA, A.T., PEPIN, K.H., GISH, W.R., CHISSOE, S.L., WENDL, M.C., DELEHAUNTY, K.D., MINER, T.L., DELEHAUNTY, A., KRAMER, J.B., COOK, L.L., FULTON, R.S., JOHNSON, D.L., MINX, P.J., CLIFTON, S.W., HAWKINS, T., BRANSCOMB, E., PREDKI, P., RICHARDSON, P., WENNING, S., SLEZAK, T., DOGGETT, N., CHENG, J.F., OLSEN, A., LUCAS, S., ELKIN, C., UBERBACHER, E., FRAZIER, M., GIBBS, R.A., MUZNY, D.M., SCHERER, S.E., BOUCK, J.B., SODERGREN, E.J., WORLEY, K.C., RIVES, C.M., GORRELL, J.H., METZKER, M.L., NAYLOR, S.L., KUCHERLAPATI, R.S., NELSON, D.L., WEINSTOCK, G.M., SAKAKI, Y., FUJIYAMA, A., HATTORI, M., YADA, T., TOYODA, A., ITOH, T., KAWAGOE, C., WATANABE, H., TOTOKI, Y., TAYLOR, T., WEISSENBACH, J., HEILIG, R., SAURIN, W., ARTIGUENAVE, F., BROTTIER, P., BRULS, T., PELLETIER, E., ROBERT, C., WINCKER, P., SMITH, D.R., DOUCETTE-STAMM, L., RUBENFIELD, M., WEINSTOCK, K., LEE, H.M., DUBOIS, J., ROSENTHAL, A., PLATZER, M., NYAKATURA, G., TAUDIEN, S., RUMP, A., YANG, H., YU, J., WANG, J., HUANG, G., GU, J., HOOD, L., ROWEN, L., MADAN, A., QIN, S., DAVIS, R.W., FEDERSPIEL, N.A., ABOLA, A.P., PROCTOR, M.J., MYERS, R.M., SCHMUTZ, J., DICKSON, M., GRIMWOOD, J., COX, D.R., OLSON, M.V., KAUL, R., RAYMOND, C., SHIMIZU, N., KAWASAKI, K., MINOSHIMA, S., EVANS, G.A., ATHANASIOU, M., SCHULTZ, R., ROE, B.A., CHEN, F., PAN, H., RAMSER, J., LEHRACH, H., REINHARDT, R., MCCOMBIE, W.R., DE LA BASTIDE, M., DEDHIA, N., BLOCKER, H., HORNISCHER, K., NORDSIEK, G., AGARWALA, R., ARAVIND, L., BAILEY, J.A., BATEMAN, A., BATZOGLOU, S., BIRNEY, E., BORK, P., BROWN, D.G., BURGE, C.B., CERUTTI, L., CHEN, H.C., CHURCH, D., CLAMP, M., COPLEY, R.R., DOERKS, T., EDDY, S.R., EICHLER, E.E., FUREY, T.S., GALAGAN, J., GILBERT, J.G., HARMON, C., HAYASHIZAKI, Y., HAUSSLER, D., HERMJAKOB, H., HOKAMP, K., JANG, W., JOHNSON, L.S., JONES, T.A., KASIF, S., KASPRYZK, A., KENNEDY, S., KENT, W.J., KITTS, P., KOONIN, E.V., KORF, I., KULP, D., LANCET, D., LOWE, T.M., MCLYSAGHT, A., MIKKELSEN, T., MORAN, J.V., MULDER, N., POLLARA, V.J., PONTING, C.P., SCHULER, G., SCHULTZ, J., SLATER, G., SMIT, A.F., STUPKA, E., SZUSTAKOWSKI, J., THIERRY-MIEG, D., THIERRY-MIEG, J., WAGNER, L., WALLIS, J., WHEELER, R., WILLIAMS, A., WOLF, Y.I., WOLFE, K.H., YANG, S.P., YEH, R.F., COLLINS, F., GUYER, M.S., PETERSON, J., FELSENFELD, A., WETTERSTRAND, K.A., PATRINOS, A., MORGAN, M.J., DE JONG, P., CATANESE, J.J., OSOEGAWA, K., SHIZUYA, H., CHOI, S., CHEN, Y.J. and INTERNATIONAL HUMAN GENOME SEQUENCING CONSORTIUM, 2001. Initial sequencing and analysis of the human genome. *Nature,* **409**(6822), pp. 860-921.

LESAGE, G. and BUSSEY, H., 2006. Cell wall assembly in Saccharomyces cerevisiae. *Microbiology and molecular biology reviews : MMBR,* **70**(2), pp. 317-343.

LIAO, L. and NOBLE, W.S., 2003. Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships. *Journal of computational biology : a journal of computational molecular cell biology,* **10**(6), pp. 857-868.

LIOLIOS, K., TAVERNARAKIS, N., HUGENHOLTZ, P. and KYRPIDES, N.C., 2006. The Genomes On Line Database (GOLD) v.2: a monitor of genome projects worldwide. *Nucleic acids research,* **34**(Database issue), pp. D332-4.

LIPKE, P.N. and KURJAN, J., 1992. Sexual agglutination in budding yeasts: structure, function, and regulation of adhesion glycoproteins. *Microbiological reviews,* **56**(1), pp. 180-194.

LO, W.S. and DRANGINIS, A.M., 1996. FLO11, a yeast gene related to the STA genes, encodes a novel cell surface flocculin. *Journal of Bacteriology,* **178**(24), pp. 7144-7151.

LOPEZ-RIBOT, J.L., ALLOUSH, H.M., MASTEN, B.J. and CHAFFIN, W.L., 1996. Evidence for presence in the cell wall of Candida albicans of a protein related to the hsp70 family. *Infect Immun,* **64**(8), pp. 3333-40.

LOPEZ-RIBOT, J.L. and CHAFFIN, W.L., 1996. Members of the Hsp70 family of proteins in the cell wall of Saccharomyces cerevisiae. *Journal of Bacteriology,* **178**(15), pp. 4724-4726.

MADERA, M. and GOUGH, J., 2002. A comparison of profile hidden Markov model procedures for remote homology detection. *Nucleic Acids Res,* **30**(19), pp. 4321-8.

MAZUR, P. and BAGINSKY, W., 1996. In vitro activity of 1,3-beta-D-glucan synthase requires the GTP-binding protein Rho1. *The Journal of biological chemistry,* **271**(24), pp. 14604-14609.

MCGOVERN, P.E., VOIGT, M.M., GLUSKER, D.L. and EXNER, L.J., 1986. Neolithic resinated wine. *Nature,* (381), pp. 480-481.

MOIR, D., STEWART, S.E., OSMOND, B.C. and BOTSTEIN, D., 1982. Cold-sensitive cell-division-cycle mutants of yeast: isolation, properties, and pseudoreversion studies. *Genetics,* **100**(4), pp. 547-563.

MOLINA, M., GIL, C., PLA, J., ARROYO, J. and NOMBELA, C., 2000. Protein localisation approaches for understanding yeast cell wall biogenesis. *Microscopy research and technique,* **51**(6), pp. 601-612.

MOUSE GENOME SEQUENCING CONSORTIUM, WATERSTON, R.H.,

LINDBLAD-TOH, K., BIRNEY, E., ROGERS, J., ABRIL, J.F., AGARWAL, P., AGARWALA, R., AINSCOUGH, R., ALEXANDERSSON, M., AN, P., ANTONARAKIS, S.E., ATTWOOD, J., BAERTSCH, R., BAILEY, J., BARLOW, K., BECK, S., BERRY, E., BIRREN, B., BLOOM, T., BORK, P., BOTCHERBY, M., BRAY, N., BRENT, M.R., BROWN, D.G., BROWN, S.D., BULT, C., BURTON, J., BUTLER, J., CAMPBELL, R.D., CARNINCI, P., CAWLEY, S., CHIAROMONTE, F., CHINWALLA, A.T., CHURCH, D.M., CLAMP, M., CLEE, C., COLLINS, F.S., COOK, L.L., COPLEY, R.R., COULSON, A., COURONNE, O., CUFF, J., CURWEN, V., CUTTS, T., DALY, M., DAVID, R., DAVIES, J., DELEHAUNTY, K.D., DERI, J., DERMITZAKIS, E.T., DEWEY, C., DICKENS, N.J., DIEKHANS, M., DODGE, S., DUBCHAK, I., DUNN, D.M., EDDY, S.R., ELNITSKI, L., EMES, R.D., ESWARA, P., EYRAS, E., FELSENFELD, A., FEWELL, G.A., FLICEK, P., FOLEY, K., FRANKEL, W.N., FULTON, L.A., FULTON, R.S., FUREY, T.S., GAGE, D., GIBBS, R.A., GLUSMAN, G., GNERRE, S., GOLDMAN, N., GOODSTADT, L., GRAFHAM, D., GRAVES, T.A., GREEN, E.D., GREGORY, S., GUIGO, R., GUYER, M., HARDISON, R.C., HAUSSLER, D., HAYASHIZAKI, Y., HILLIER, L.W., HINRICHS, A., HLAVINA, W., HOLZER, T., HSU, F., HUA, A., HUBBARD, T., HUNT, A., JACKSON, I., JAFFE, D.B., JOHNSON, L.S., JONES, M., JONES, T.A., JOY, A., KAMAL, M., KARLSSON, E.K., KAROLCHIK, D., KASPRZYK, A., KAWAI, J., KEIBLER, E., KELLS, C., KENT, W.J., KIRBY, A., KOLBE, D.L., KORF, I., KUCHERLAPATI, R.S., KULBOKAS, E.J., KULP, D., LANDERS, T., LEGER, J.P., LEONARD, S., LETUNIC, I., LEVINE, R., LI, J., LI, M., LLOYD, C., LUCAS, S., MA, B., MAGLOTT, D.R., MARDIS, E.R., MATTHEWS, L., MAUCELI, E., MAYER, J.H., MCCARTHY, M., MCCOMBIE, W.R., MCLAREN, S., MCLAY, K., MCPHERSON, J.D., MELDRIM, J., MEREDITH, B., MESIROV, J.P., MILLER, W., MINER, T.L., MONGIN, E., MONTGOMERY, K.T., MORGAN, M., MOTT, R., MULLIKIN, J.C., MUZNY, D.M., NASH, W.E., NELSON, J.O., NHAN, M.N., NICOL, R., NING, Z., NUSBAUM, C., O'CONNOR, M.J., OKAZAKI, Y., OLIVER, K., OVERTON-LARTY, E., PACHTER, L., PARRA, G., PEPIN, K.H., PETERSON, J., PEVZNER, P., PLUMB, R., POHL, C.S., POLIAKOV, A., PONCE, T.C., PONTING, C.P., POTTER, S., QUAIL, M., REYMOND, A., ROE, B.A., ROSKIN, K.M., RUBIN, E.M., RUST, A.G., SANTOS, R., SAPOJNIKOV, V., SCHULTZ, B., SCHULTZ, J., SCHWARTZ, M.S., SCHWARTZ, S., SCOTT, C., SEAMAN, S., SEARLE, S., SHARPE, T., SHERIDAN, A., SHOWNKEEN, R., SIMS, S., SINGER, J.B., SLATER, G., SMIT, A., SMITH, D.R., SPENCER, B., STABENAU, A., STANGE-THOMANN, N., SUGNET, C., SUYAMA, M., TESLER, G., THOMPSON, J., TORRENTS, D., TREVASKIS, E., TROMP, J., UCLA, C., URETA-VIDAL, A., VINSON, J.P., VON NIEDERHAUSERN, A.C., WADE, C.M., WALL, M., WEBER, R.J., WEISS, R.B., WENDL, M.C., WEST, A.P., WETTERSTRAND, K., WHEELER, R., WHELAN, S., WIERZBOWSKI, J., WILLEY, D., WILLIAMS, S., WILSON, R.K., WINTER, E., WORLEY, K.C., WYMAN, D., YANG, S., YANG, S.P., ZDOBNOV, E.M., ZODY, M.C. and LANDER, E.S., 2002. Initial sequencing and comparative analysis of the mouse genome. *Nature,* **420**(6915), pp. 520-562.

NEEDLEMAN, S.B. and WUNSCH, C.D., 1970. A general method applicable to the

search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology,* **48**(3), pp. 443-453.

PARK, Y.S., JEONG, H.S., SUNG, H.C. and YUN, C.W., 2005. Sed1p interacts with Arn3p physically and mediates ferrioxamine B uptake in Saccharomyces cerevisiae. *Current genetics,* **47**(3), pp. 150-155.

PAULICK, M.G. and BERTOZZI, C.R., 2008. The Glycosylphosphatidylinositol Anchor: A Complex Membrane-Anchoring Structure for Proteins. *Biochemistry,* .

PEARSON, W.R., 2000. Flexible sequence similarity searching with the FASTA3 program package. *Methods Mol Biol,* **132**, pp. 185-219.

PITTET, M. and CONZELMANN, A., 2007. Biosynthesis and function of GPI proteins in the yeast Saccharomyces cerevisiae. *Biochimica et biophysica acta,* **1771**(3), pp. 405-420.

PROTCHENKO, O., FEREA, T., RASHFORD, J., TIEDEMAN, J., BROWN, P.O., BOTSTEIN, D. and PHILPOTT, C.C., 2001. Three cell wall mannoproteins facilitate the uptake of iron in Saccharomyces cerevisiae. *The Journal of biological chemistry,* **276**(52), pp. 49244-49250.

RAM, A.F., KAPTEYN, J.C., MONTIJN, R.C., CARO, L.H., DOUWES, J.E., BAGINSKY, W., MAZUR, P., VAN DEN ENDE, H. and KLIS, F.M., 1998. Loss of the plasma membrane-bound protein Gas1p in Saccharomyces cerevisiae results in the release of beta1,3-glucan into the medium and induces a compensation mechanism to ensure cell wall integrity. *Journal of Bacteriology,* **180**(6), pp. 1418-1424.

RAMISHVILI, R., 1983. New materials on the history of viniculture in Georgia (in Georgian, with Russian summary). *Hist Archaeol Ethnol Art Hist Ser,* (2), pp. 125-140.

RAPAPORT, F., BARILLOT, E. and VERT, J.P., 2008. Classification of arrayCGH data using fused SVM. *Bioinformatics (Oxford, England),* **24**(13), pp. i375-82.

REED, S.I., 1980. The selection of S. cerevisiae mutants defective in the start event of cell division. *Genetics,* **95**(3), pp. 561-577.

RODRIGUEZ-PENA, J.M., CID, V.J., ARROYO, J. and NOMBELA, C., 2000. A novel family of cell wall-related proteins regulated differently during the yeast life cycle. *Molecular and cellular biology,* **20**(9), pp. 3245-3255.

SANGER, F., 1981. Determination of nucleotide sequences in DNA. *Science (New York, N.Y.),* **214**(4526), pp. 1205-1210.

SCANNELL, D.R., FRANK, A.C., CONANT, G.C., BYRNE, K.P., WOOLFIT, M. and WOLFE, K.H., 2007. Independent sorting-out of thousands of duplicated gene pairs in

two yeast species descended from a whole-genome duplication. *Proceedings of the National Academy of Sciences of the United States of America,* **104**(20), pp. 8397-8402.

SESTAK, S., HAGEN, I., TANNER, W. and STRAHL, S., 2004. Scw10p, a cell-wall glucanase/transglucosidase important for cell-wall stability in Saccharomyces cerevisiae. *Microbiology (Reading, England),* **150**(Pt 10), pp. 3197-3208.

SHAHINIAN, S. and BUSSEY, H., 2000. beta-1,6-Glucan synthesis in Saccharomyces cerevisiae. *Molecular microbiology,* **35**(3), pp. 477-489.

SMITH, R.D. and LUPASHIN, V.V., 2008. Role of the conserved oligomeric Golgi (COG) complex in protein glycosylation. *Carbohydrate research,* **343**(12), pp. 2024-2031.

SMITH, T.F. and WATERMAN, M.S., 1981. Identification of common molecular subsequences. *Journal of Molecular Biology,* **147**(1), pp. 195-197.

SPELLMAN, P.T., SHERLOCK, G., ZHANG, M.Q., IYER, V.R., ANDERS, K., EISEN, M.B., BROWN, P.O., BOTSTEIN, D. and FUTCHER, B., 1998. Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization. *Molecular biology of the cell,* **9**(12), pp. 3273-3297.

TERASHIMA, H., YABUKI, N., ARISAWA, M., HAMADA, K. and KITADA, K., 2000. Up-regulation of genes encoding glycosylphosphatidylinositol (GPI)-attached proteins in response to cell wall damage caused by disruption of FKS1 in Saccharomyces cerevisiae. *Molecular & general genetics : MGG,* **264**(1-2), pp. 64-74.

TZAGOLOFF, A. and DIECKMANN, C.L., 1990. PET genes of Saccharomyces cerevisiae. *Microbiological reviews,* **54**(3), pp. 211-225.

VAISHNAV, V.V., BACON, B.E., O'NEILL, M. and CHERNIAK, R., 1998. Structural characterization of the galactoxylomannan of Cryptococcus neoformans Cap67. *Carbohydrate research,* **306**(1-2), pp. 315-330.

VAPNIK, V.N., 1998. Statistical Learning Theory. Wiley-interscience.

VENTER, J.C., ADAMS, M.D., MYERS, E.W., LI, P.W., MURAL, R.J., SUTTON, G.G., SMITH, H.O., YANDELL, M., EVANS, C.A., HOLT, R.A., GOCAYNE, J.D., AMANATIDES, P., BALLEW, R.M., HUSON, D.H., WORTMAN, J.R., ZHANG, Q., KODIRA, C.D., ZHENG, X.H., CHEN, L., SKUPSKI, M., SUBRAMANIAN, G., THOMAS, P.D., ZHANG, J., GABOR MIKLOS, G.L., NELSON, C., BRODER, S., CLARK, A.G., NADEAU, J., MCKUSICK, V.A., ZINDER, N., LEVINE, A.J., ROBERTS, R.J., SIMON, M., SLAYMAN, C., HUNKAPILLER, M., BOLANOS, R., DELCHER, A., DEW, I., FASULO, D., FLANIGAN, M., FLOREA, L., HALPERN, A., HANNENHALLI, S., KRAVITZ, S., LEVY, S., MOBARRY, C., REINERT, K., REMINGTON, K., ABU-THREIDEH, J., BEASLEY, E., BIDDICK, K., BONAZZI, V.,

BRANDON, R., CARGILL, M., CHANDRAMOULISWARAN, I., CHARLAB, R., CHATURVEDI, K., DENG, Z., DI FRANCESCO, V., DUNN, P., EILBECK, K., EVANGELISTA, C., GABRIELIAN, A.E., GAN, W., GE, W., GONG, F., GU, Z., GUAN, P., HEIMAN, T.J., HIGGINS, M.E., JI, R.R., KE, Z., KETCHUM, K.A., LAI, Z., LEI, Y., LI, Z., LI, J., LIANG, Y., LIN, X., LU, F., MERKULOV, G.V., MILSHINA, N., MOORE, H.M., NAIK, A.K., NARAYAN, V.A., NEELAM, B., NUSSKERN, D., RUSCH, D.B., SALZBERG, S., SHAO, W., SHUE, B., SUN, J., WANG, Z., WANG, A., WANG, X., WANG, J., WEI, M., WIDES, R., XIAO, C., YAN, C., YAO, A., YE, J., ZHAN, M., ZHANG, W., ZHANG, H., ZHAO, Q., ZHENG, L., ZHONG, F., ZHONG, W., ZHU, S., ZHAO, S., GILBERT, D., BAUMHUETER, S., SPIER, G., CARTER, C., CRAVCHIK, A., WOODAGE, T., ALI, F., AN, H., AWE, A., BALDWIN, D., BADEN, H., BARNSTEAD, M., BARROW, I., BEESON, K., BUSAM, D., CARVER, A., CENTER, A., CHENG, M.L., CURRY, L., DANAHER, S., DAVENPORT, L., DESILETS, R., DIETZ, S., DODSON, K., DOUP, L., FERRIERA, S., GARG, N., GLUECKSMANN, A., HART, B., HAYNES, J., HAYNES, C., HEINER, C., HLADUN, S., HOSTIN, D., HOUCK, J., HOWLAND, T., IBEGWAM, C., JOHNSON, J., KALUSH, F., KLINE, L., KODURU, S., LOVE, A., MANN, F., MAY, D., MCCAWLEY, S., MCINTOSH, T., MCMULLEN, I., MOY, M., MOY, L., MURPHY, B., NELSON, K., PFANNKOCH, C., PRATTS, E., PURI, V., QURESHI, H., REARDON, M., RODRIGUEZ, R., ROGERS, Y.H., ROMBLAD, D., RUHFEL, B., SCOTT, R., SITTER, C., SMALLWOOD, M., STEWART, E., STRONG, R., SUH, E., THOMAS, R., TINT, N.N., TSE, S., VECH, C., WANG, G., WETTER, J., WILLIAMS, S., WILLIAMS, M., WINDSOR, S., WINN-DEEN, E., WOLFE, K., ZAVERI, J., ZAVERI, K., ABRIL, J.F., GUIGO, R., CAMPBELL, M.J., SJOLANDER, K.V., KARLAK, B., KEJARIWAL, A., MI, H., LAZAREVA, B., HATTON, T., NARECHANIA, A., DIEMER, K., MURUGANUJAN, A., GUO, N., SATO, S., BAFNA, V., ISTRAIL, S., LIPPERT, R., SCHWARTZ, R., WALENZ, B., YOOSEPH, S., ALLEN, D., BASU, A., BAXENDALE, J., BLICK, L., CAMINHA, M., CARNES-STINE, J., CAULK, P., CHIANG, Y.H., COYNE, M., DAHLKE, C., MAYS, A., DOMBROSKI, M., DONNELLY, M., ELY, D., ESPARHAM, S., FOSLER, C., GIRE, H., GLANOWSKI, S., GLASSER, K., GLODEK, A., GOROKHOV, M., GRAHAM, K., GROPMAN, B., HARRIS, M., HEIL, J., HENDERSON, S., HOOVER, J., JENNINGS, D., JORDAN, C., JORDAN, J., KASHA, J., KAGAN, L., KRAFT, C., LEVITSKY, A., LEWIS, M., LIU, X., LOPEZ, J., MA, D., MAJOROS, W., MCDANIEL, J., MURPHY, S., NEWMAN, M., NGUYEN, T., NGUYEN, N., NODELL, M., PAN, S., PECK, J., PETERSON, M., ROWE, W., SANDERS, R., SCOTT, J., SIMPSON, M., SMITH, T., SPRAGUE, A., STOCKWELL, T., TURNER, R., VENTER, E., WANG, M., WEN, M., WU, D., WU, M., XIA, A., ZANDIEH, A. and ZHU, X., 2001. The sequence of the human genome. *Science (New York, N.Y.),* **291**(5507), pp. 1304-1351.

VERSTREPEN, K.J., REYNOLDS, T.B., and FINK, G.R., 2004. Origins of variation in the fungal cell surface. Nat Rev Microbiol **2**:533-40.

WANG, L., ZHU, J. and ZOU, H., 2008. Hybrid huberized support vector machines for microarray classification and gene selection. *Bioinformatics (Oxford, England),* **24**(3),

pp. 412-419.

WILD, K., HALIC, M., SINNING, I. and BECKMANN, R., 2004. SRP meets the ribosome. *Nature structural & molecular biology,* **11**(11), pp. 1049-1053.

YAN, A. and LENNARZ, W.J., 2005. Unraveling the mechanism of protein N-glycosylation. *The Journal of biological chemistry,* **280**(5), pp. 3121-3124.

YANG, Z.R., 2004. Biological applications of support vector machines. *Briefings in bioinformatics,* **5**(4), pp. 328-338.

YANG, Z.R. and JOHNSON, F.C., 2005. Prediction of T-cell epitopes using biosupport vector machines. *Journal of chemical information and modeling,* **45**(5), pp. 1424-1428.