Negative Information for Motif Discovery

by

Ken Takusagawa

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author .
Department of Electrical Engineering and Computer Science
July 31, 2003

Certified by .....
David Gifford
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Negative Information for Motif Discovery

by

Ken Takusagawa

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2003, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

In the field of computational biology, many previous DNA motif-discovery algorithms have suffered from discovering false motifs of sequences that are over-represented in all the intergenic sequences of a genome. Such motif discovery algorithms have mainly relied on only the "positive" intergenic regions to which a given transcription factor is thought to bind. This thesis will address the false motif problem by carefully incorporating information from the "negative" intergenic regions, regions to which the transcription factor does not bind. The algorithm presented in this thesis enumerates a large class of potential motifs (with possible wildcards), counts their occurrences in both positive and negative regions, and performs a statistical test to determine if a potential motif is over-represented in the positive intergenic regions compared to the negative intergenic regions. We present results of this algorithm on transcription factor binding data from yeast, and compare its perfomance against other motif discovery algorithms. Our results demonstrate that our method performs slightly better than other motif discovery algorithms.

Thesis Supervisor: David Gifford
Title: Professor

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motif discovery is one tool for unraveling the transcriptional regulatory network of an organism. The underlying model assumes that a transcription factor binds to a specific short sequence ("a motif") in the upstream intergenic region of a gene and affects the gene's expression in some way. By discovering a transcription factor's motif and scanning a genome for it, we can hypothesize which genes the transcription factor regulates. We can also make more elaborate hypotheses about combinatorial regulation of a gene by multiple transcription factors if multiple different motifs appear upstream of a gene.

Chromatin immunoprecipitation (ChIP) microarray experiments can determine to which intergenic regions a particular transcription factor binds on an entire genome[10]. After choosing an appropriate cutoff p-value, the intergenic regions are partitioned into two categories: those to which the transcription factor is thought to bind (which will be referred to as the "positive intergenic sequences") and those to which it does not bind (the "negative intergenic sequences"). This thesis will focus on discovering motifs in the ChIP experiments performed on 108 different transcription factors in *Saccharomyces cerevisiae* (baker's yeast)[10].

If an algorithm uses only the positive sequences for motif discovery, then it will likely discover many false motifs. Such false motifs are caused by sequences which appear frequently in all the intergenic sequences of a genome. In yeast, two prominent simple examples of such sequences are poly-A (long strings of consecutive

adenine nucleotides) and poly-CA (long strings of alternating cytosine and adenine nucleotides)[1].

This thesis will attempt to solve the false motif problem by incorporating the negative intergenic sequences. A potential motif must be significantly over-represented in the positive intergenic sequences when compared with the negative intergenic sequences.

1.1 Related work

There have been many past efforts to use negative intergenic sequences to derive a statistical test.

The very popular "Random Sequence Null Hypothesis" (so named in [3]) uses the negative sequences to discover the parameters of an n-th order background Markov model ($n = 0$ and $n = 3$ are popular). This approach greatly dilutes the information content of the negative intergenic sequences, and especially loses information about false motifs whose length is greater than the order of the Markov model.

Two papers describe a different approach to incorporate information from negative intergenic sequences. The approach pursued in this thesis will be similar to the following two citations. Vilo, *et al.*[12] cluster genes by their expression profiles and seek to discover motifs within each cluster. Their test for significance compares the occurrences of a potential motif within a cluster to the occurrences in all intergenic sequences. Their significance test compares the ratio of the two values against a binomial distribution. Barash, *et al.*[3] describe an alternative to the "Random Sequence Null Hypothesis", namely a "Random Selection Null Hypothesis". They perform a similar calculation to [12], but compare against a hyper-geometric distribution. (The difference appears to be the assumption of whether motif-containing sequences are selected "with replacement" or "without replacement" from all the sequences.)

Two other papers are also related to the work proposed in this thesis. Sinha [11] shows how to view motif discovery as a feature selection problem for classification. The algorithm presented in the paper requires input of positive and negative intergenic

sequences. Sinha generates the negative examples (intergenic sequences) artificially using a Markov model, but the framework presented the paper could easily use real negative intergenic sequences from ChIP experiments.

On the implementation side, this thesis develops a method of reducing the memory requirement of enumerative methods for motif discovery. Buhler and Tompa [6] also discuss this problem and propose other techniques.

1.2 Contributions of this thesis

With respect to the related work mentioned above, this thesis makes the following contributions to the field.

- Application of statistical tests similar to those described in [3, 12] on data from ChIP experiments, which were not available at the time [3, 12] were written.
- Generalization of the statistical tests to allow for weights on intergenic sequences.
- Word counting with wildcards. While the idea is not new, wildcards greatly increase the computation time and memory requirements and therefore require careful attention to implementation.
- A careful and detailed description of the implementation of the algorithm, including a filtering technique to reduce running time and a modified hash table technique to reduce memory requirement.

1.3 Structure of this thesis

This thesis will proceed in the following manner. Chapter 2 will discuss the algorithmic and implementation issues involved in this thesis's approach to motif discovery. Chapter 3 will discuss the statistical test and its mathematical aspects. Finally, Chapter 4 will present results and directions for future work.

Chapter 2

Word-counting Algorithm

This chapter has the following structure. In §2.1 we will introduce some terminology and conventions. §2.2 will define the word-counting problem, and describe how it will be used for motif discovery. §2.3 will describe two simple but slow algorithms for word counting. §2.4–2.10 then will discuss the design and implementation of principal algorithm of this thesis, a hybrid algorithm which combines the strengths of the two simple algorithms.

2.1 Sequences and Words

The term *sequence* denotes a concatenation of DNA nucleotides, chosen from the set $\{\mathcal{A}, \mathcal{C}, \mathcal{G}, \mathcal{T}\}$. Sequences will be typeset in $\mathcal{CALLIGRAPHIC}$ font. We will typically use the term *length* to describe the size of sequences and subsequences.

The term *word* denotes a concatenation of *word elements*. Word elements are nucleotides or *wildcard elements*. Words represent potential motifs. We will typically use the term *width* to describe the size of words and motifs.

Wildcard elements are specified by degenerate nucleotides which specify two or more possible real nucleotides that can occupy the position of the wildcard. There are a total of $2^4 - 1 = 15$ real nucleotides and wildcard elements. The 11 total wildcard elements $(15 - 4)$ can be denoted by their IUPAC code, or by a regular expression similar to the syntax used by the Unix command `grep`. This chapter will will use the

grep notation	IUPAC notation
[AC]	M
[AG]	R
[AT]	W
[CG]	S
[CT]	Y
[GT]	K
•	N

Table 2.1: Degenerate nucleotides used by the algorithm

grep notation	IUPAC notation
[ACG]	V
[ACT]	H
[AGT]	D
[CGT]	B

Table 2.2: Not used degenerate nucleotides

regular expression syntax. Words will be typeset using the `typewriter` font.

Of the 11 wildcard elements, the algorithm to be presented will only use 7 of them: the 6 double-degenerate nucleotides and the quadruple-degenerate ("gap") nucleotide. The elements used and unused are listed in Tables 2.1 and 2.2.

Borrowing from regular expression terminology, we say that a word *matches* a sequence if it is present somewhere in the sequence or the sequence's reversed complement. We will also say that a sequence is an *instantiation* of a word if the sequence matches the word and is the same length as the word.

2.1.1 Canonical Form of a Word

We will effectively search both DNA strands of the upstream region of a gene. Therefore, a word and its reversed complement are identical for the purposes of motif discovery. When one occurs, the other occurs on the other strand, as illustrated in Figure 2-1.

Furthermore, a word with a gap (a • wildcard) occurring at its end is identical to a word with the gap at the beginning (ignoring the corner case when such a word occurs at the end of an intergenic sequence). For example `AAGC••` is identical to

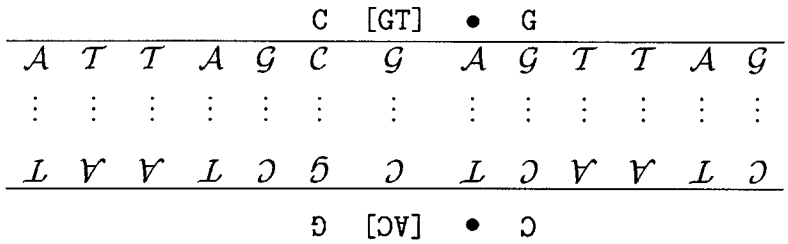


Figure 2-1: The word C[GT]•G matches the sequence on one strand. The reversed complemented word C•[AC]G matches on the complement strand.

word element	A	C	G	T	[AC]	[AG]	[AT]	[CG]	[CT]	[GT]	•
number	0	1	2	3	4	5	6	7	8	9	10

Table 2.3: Every word element has a number.

••AAGC and •AAGC•.

In order to eliminate duplicate words caused by reversed complements and gaps, we define a canonical form of a word. First we associate each word element with a number, which establishes a lexicographic ordering of words (Table 2.3).

With this ordering, we can compare a word with its reversed complement. Note that to complement a wildcard element, we simply complement the members of its character class.

We describe a predicate *is-canonical*(m) for a word m.

1. If a m begins with •, it is not canonical.

2. Otherwise, if a m is lexicographically greater than its reversed complement, it is not canonical.

3. Otherwise, m is canonical.

Rule 1 stipulates that gap elements can only occur at the end of a canonical word. For example, of AAT••, •AAT•, and ••AAT, only the first is canonical.

There is a subtle interaction between Rule 1 and Rule 2. Because the gap element • is lexicographically the greatest word element, it is assured that a word containing a gap element only at its end is canonical.

Under the rules specified above, there still do exist identical words that are canonical. For example, ATG• and CAT• are both canonical, but when one occurs, the other

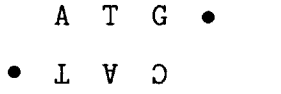


Figure 2-2: The words ATG• and CAT• are not quite reversed complements of each other, but when one occurs so does the other. Both words are canonical.

occurs on the complementary strand shifted by one base (Figure 2-2).

The presence of multiple identical canonical words does not affect the correctness of the algorithm, it merely creates extraneous output. A significant motif will be reported multiple times in each of its canonical forms. (And it was not worth the programmer effort to correct this bug.)

2.2 Basic problem

In order to do motif discovery with word counting, we need to first solve the following algorithmic problem. First, we define the set of words M which interest us. The experiments performed in this thesis searched for words of width 7 with up to 2 wildcard elements. Next, given two sets of sequences P ("positive") and A ("all") with $P \subseteq A$, we wish to determine for each word $m \in M$ which sequences in P and which in A match m.

The set P contains the intergenic sequences bound by the transcription factor whose motif we wish to discover. The negative set of unbound sequences is implicitly defined by $A \setminus P$. We choose to define the problem in terms of "positive" and "all" instead of "positive" and "negative" for convenience of implementation. It is slightly easier to use the same "all" set for all transcription factors, rather than explicitly calculating the negative set for each different transcription factor.

Having determined which sequences in P and A which match m, the statistical tests described in Chapter 3 can estimate the significance of m as a potential motif. Note that we want to know the identities of the sequences which match m, not just their count. Two of the statistical tests will make use of the lengths of the sequences for a more accurate result than just using counts.

The problem has the feature that the number of sequences in P is much fewer than

the number in A. Furthermore, we only care about words present in P. If a word is not present in P we do not care about its presence or lack thereof in A. Therefore, it would be quite reasonable for an algorithm to first restrict its attention to P.

2.3 Two simple algorithms

In this section we describe two simple but slow algorithms to solve the basic problem described above.

2.3.1 Exhaustive Enumeration

This technique simply enumerates the entire space of words in which we are interested and determines the presence of each word in P and A. (If a word does not exist in P we do not need to search for it in A.)

foreach $m \in M$ **do**

  if m matches somewhere in P **then**

    POSITIVES-WITH-M $\leftarrow$ *find*(m,P)

    ALL-WITH-M $\leftarrow$ *find*(m,A)

    test-for-significance(FILTERED-POSITIVES, FILTERED-ALL)

We will use the pseudocode notation "**foreach** $x \in S$" loosely throughout this chapter. It denotes a appropriate algorithm to iterate through the elements of S one by one. It typically is not implemented by first storing all the elements of S, then iterating through them. Frequently (for example in the enumeration of M) the most appropriate algorithm is recursive, and the statements within the **foreach** loop are evaluated at the base case of the recursion.

The problem with of exhaustive enumeration is there may be many words not in P. The size of the space of words increases exponentially with the width w of the word, while the number of words of width w in P is bounded by a constant times the total length of all the sequences in P.

This performance flaw this method begs for a very quick way of determining if

"m matches somewhere in P".

2.3.2 Window across positive regions

The alternative to enumerating all words is to scan each sequence of P with a window of width w, where w is the width of the motif we wish to discover.

foreach $s \in scan_w(P)$ **do**

// Comment: s is a length-w subsequence

foreach m which matches s **do**

ALL-POSITIVE-WORDS$\{m\}.add$(sequence number of s)

foreach $m \in$ keys of ALL-POSITIVE-WORDS

FILTERED-ALL $\leftarrow$ *find*(m,A)

test-for-significance(ALL-POSITIVE-WORDS$\{m\}$, FILTERED-ALL)

The problem of this method is the memory required to store ALL-POSITIVE-WORDS, which is a mapping from words to sets of sequences identifiers. A straightforward implementation would need to store every word (not just every subsequence) in P. This performance flaw begs for a way to reduce the memory requirement.

2.4 Hybrid approach

This thesis implemented a hybrid approach between the two simple algorithms above. The approach developed in the following sections is a modification of exhaustive enumeration using an imperfect but low-memory version of window scanning as a filter to quickly determine if a word matches somewhere in P.

Although the hybrid approach is faster than naive exhaustive enumeration, and uses less memory than naive implementation of window scanning, it should be noted that the purported performance flaws of §2.3.1 and §2.3.2 would not have been significant concerns for the experiments performed for this thesis. For the yeast genome (with approximately 3 million bases of intergenic sequence) and searching for motifs

of width $w = 7$ with up to 2 wildcards, modern machines have sufficient memory for even a naive implementation of window scanning. Furthermore, for many of the experiments, almost all possible 7-mers were present in P so filtering to determine what words are in P did not save very much computation. Therefore, the techniques described in the following sections will probably yield signficant benefits only in future studies on larger genomes or wider motifs.

The hybrid motif discovery algorithm consists of four major routines, which are illustrated in Figure 2-3. Sections 2.5 through 2.9 will describe each routine in detail.

2.5 Initial hashing

This routine corresponds to the window-scanning algorithm of §2.3.2. Its purpose is to create a data structure which can quickly determine if a word matches some sequence in the positive set P. However, unlike naive window-scanning, it will conserve memory, at the cost that the data structure will occasionally make mistakes in the determination.

foreach $i \in P$ **do**
  foreach $s \in [Subseq_w(i) \cup reverse\text{-}complement(Subseq_w(i))]$ **do**
    foreach $m \in AddWildcards_h(s)$ **do**
      if $is\text{-}canonical(m)$ **then**
        BUCKETS$[hash(m)].hit()$
  foreach *entry* of BUCKETS **do**
    BUCKETS$[entry].advance()$

P are the positive intergenic sequences, that is, those bound by the transcription factor whose motif we wish to discover.

$Subseq_w(i)$ are the length-w subsequences of i computed by scanning i with a width-w window. For example, $Subseq_3(\mathcal{ATTGA}) = \{\mathcal{ATT}, \mathcal{TTG}, \mathcal{TGA}\}$. The same subsequence might be evaluated multiple times if it occurs more than once.

The operation "$reverse\text{-}complement(W)$" reverses and DNA-base-pair complements every element of W. For example,

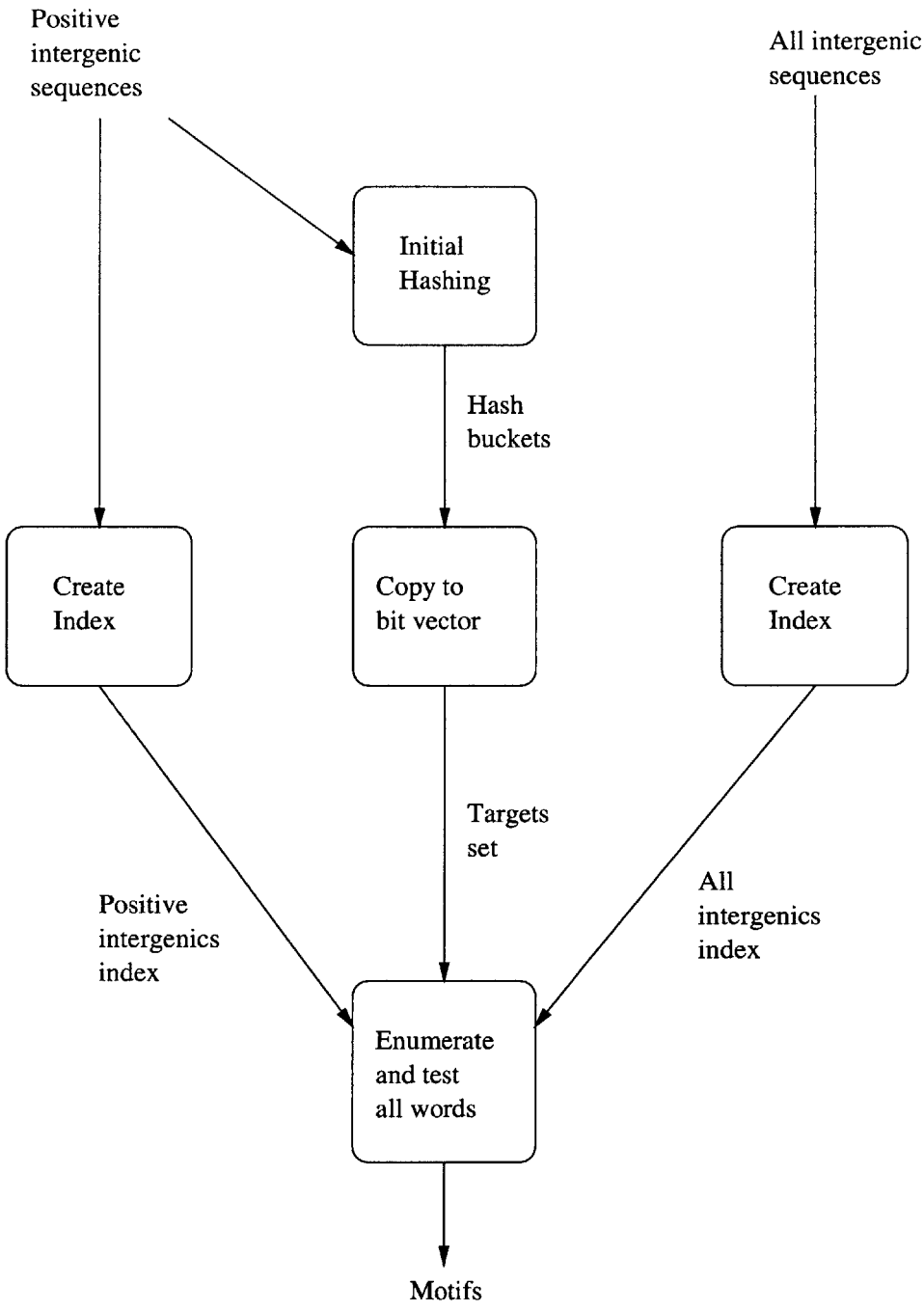


Figure 2-3: Major routines of the algorithm

ACG	AC[AG]	AC[CG]	AC[GT]	AC•
A[AC]G	A[AC][AG]	A[AC][CG]	A[AC][GT]	A[AC]•
A[CG]G	A[CG][AG]	A[CG][CG]	A[CG][GT]	A[CG]•
A[CT]G	A[CT][AG]	A[CT][CG]	A[CT][GT]	A[CT]•
A•G	A•[AG]	A•[CG]	A•[GT]	A••
[AC]CG	[AC]C[AG]	[AC]C[CG]	[AC]C[GT]	[AC]C•
[AC][AC]G	[AC][CG]G	[AC][CT]G	[AC]•G	[AG]CG
[AG]C[AG]	[AG]C[CG]	[AG]C[GT]	[AG]C•	[AG][AC]G
[AG][CG]G	[AG][CT]G	[AG]•G	[AT]CG	[AT]C[AG]
[AT]C[CG]	[AT]C[GT]	[AT]C•	[AT][AC]G	[AT][CG]G
[AT][CT]G	[AT]•G	•CG	•C[AG]	•C[CG]
•C[GT]	•C•	•[AC]G	•[CG]G	•[CT]G
••G				

Table 2.4: The result of $AddWildcards_2(\mathcal{ACG})$, given in lexicographic order (reading across columns).

$reverse\text{-}complement\{\mathcal{ATT}, \mathcal{TTG}, \mathcal{TGA}\} = \{\mathcal{AAT}, \mathcal{CAA}, \mathcal{TCA}\}$.

$AddWildcards_h(s)$ takes a sequence s and creates a set of words from s with up to h wildcard elements. The wildcards elements that can take the place of a given nucleotide are the three double-degenerate nucleotides consistent with the nucleotide it replaces, and the quadruple-degenerate nucleotide. The following table summarizes the wildcard element possibilities.

Nucleotide	... can be replaced with			
$\mathcal{A}$	•	[AC]	[AG]	[AT]
$\mathcal{C}$	•	[AC]	[CG]	[CT]
$\mathcal{G}$	•	[AG]	[CG]	[GT]
$\mathcal{T}$	•	[AT]	[CT]	[GT]

Thus, $AddWildcards_h(s)$ replaces up to h positions of s with a wildcard listed above. An example of $AddWildcards_2(\mathcal{ACG})$ is given in Table 2.4.

The $hash(m)$ function takes a word m and maps it to an integer in the range $[0 \ldots tableSize - 1]$ where *tableSize* is the size of the BUCKETS array. Most of the experiments used a *tableSize* of 10,000,000. The hashing function and its performance are discussed in detail in §2.6.

The BUCKETS array is an array of buckets. Each bucket is an object with the internal state variables *count* (an integer, initialized to zero) and *alreadyHit* (a boolean,

initialized to FALSE). Each bucket responds to the *hit*() and *advance*() messages in the following way:

```
hit()
    if (not alreadyHit) then
        alreadyHit ← TRUE
        count++

advance()
    alreadyHit ← FALSE
```

hit() ensures that multiple hits from the same intergenic region of sequences are only counted once.

This initial hashing phase is theoretically the memory bottleneck of the algorithm and BUCKETS has a critical feature that saves memory. Unlike a normal hashtable, BUCKETS only stores values, not key-value pairs. It is possible that multiple keys (words) will collide and hash to the same bucket.

The implementation of BUCKETS used one byte per bucket. The *alreadyHit* boolean was stored in the least significant bit of the byte, and the remaining upper seven bits stored *count* up to 127 hits. If a bucket is hit more than 127 times it was thresholded at 127.

In contrast, a true hashtable (with the ability to prevent collisions) will probably use far more than 1 byte per bucket. Storing the key requires space, perhaps 7 bytes for the width-7 words examined in this thesis. Storing the value requires 1 byte as described above. If "separate chaining" is used to resolve hashtable collisions (that is, each table entry contains a linked list of the keys which have collided), then another 4 bytes (assuming 32-bit memory addressing) are needed for the "next" pointer of the linked list. All together, a true hashtable might require 12 times more memory than the BUCKETS implementation.

Because BUCKETS allows for collisions, the initial hashing phase is imperfect. Later routines will examine the BUCKETS table to see if a word is present in the positive intergenic regions. If the value of the BUCKETS entry is positive, then the

word *might* be present. If the value is zero, then we know that the word was definitely *not* present.

2.6 Hashing function

Since words are composed of 11 possible word elements, the hashing function interprets a word as a integer expressed in base 11 and returns that value modulo *tableSize*. The value of each word element is given in Table 2.3 on page 17. If any number of wildcard elements were permitted (up to the width of the word) and non-canonical words were not eliminated, it is clear that this hash function evenly distributes all possible words.

However, the limit on the number of wildcard elements and the requirement that words be canonical causes the hash function to exhibit complicated behavior with respect to the *tableSize*, as illustrated in Figure 2-4. The figure gives the number of hash collisions (number of buckets containing more than one value) caused by hashing all the words in M, the canonical words of width 7 with up to 2 wildcard elements. The number of collisions varies very significantly for small changes in *tableSize*.

A Monte Carlo simulation computed that for *tableSize* = 10,000,000, the expected number of collisions for randomly distributed bucket hits is 19738.5, with a standard deviation of 134. When all words in M are hashed, the number of collisions was 19638, or just slightly better than the expected value. The best *tableSize* within a neighborhood of 100 of 10,000,000 is 10,000,096, with only 86 collisions among the 634,976 canonical words, or 140 standard deviations better than the expected value. Unfortunately this fact was discovered after all the experiments had been completed, so we did not use this more efficient table size. Furthermore, this efficient table size is specifically tailored for words of width 7 with up to 2 wildcard elements; other widths or number of wildcard elements will have different good table sizes.

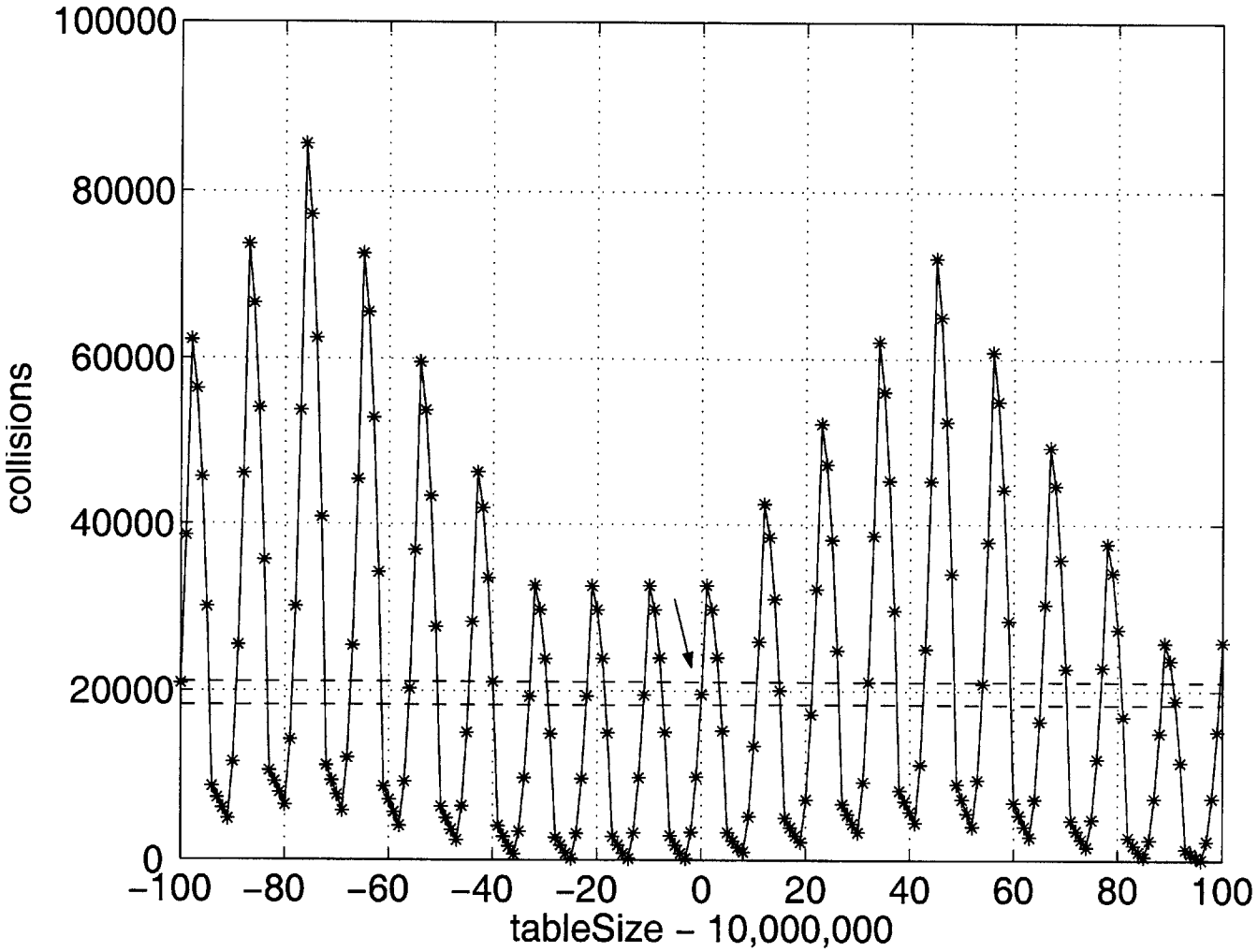


Figure 2-4: Number of collisions caused by hashing all 634,976 canonical words of width 7 with up to two wildcard elements, for *tableSize* in the neighborhood of 10,000,000. The nearly horizontal dashed lines mark a 10-standard deviation interval around the expected number of collisions for 634,976 random hits to a BUCKETS table of *tableSize*. The data point (0, 19638) is marked with an arrow.

2.7 Copy to bit vector

This routine selects the buckets which were hit at least twice in §2.5. The threshold of 2 was chosen extremely conservatively, after early experimentation chose a threshold too high that it lost significant motifs. Choosing this threshold wisely in the future will be discussed in §4.4.1.

The buckets to keep, namely those above or equal to the threshold, are marked in a bit vector TARGETS-SET of length *tableSize*. The implementation type of TARGETS-SET was the `vector<bool>` class in C++. The sole purpose of copying to a bit vector is to save memory (by a factor of 8). After copying, the buckets table can be deallocated, making space from the indexes in created §2.8.

During the copy operation, memory for both the buckets table and the bit vector is needed. However, the copy operation behaves well under virtual memory systems that predict future memory reads and writes by proximity to locations of previous reads and writes.

2.8 Create Index

This routine creates an associative mapping from length-w subsequences to identifiers of the intergenic sequences that contain the subsequence. The type of this container, expressed as C++, is `map < Sequence, set<int> >`, where `int`'s are used for the intergenic sequence identifier. We also experimented with `hash_map`'s, namely `__gnu_cxx::hash_map< Sequence , set<int> >`, an extension to the C++ Standard Template Library. `hash_map`'s were only slight faster than `map`'s, using the `gcc` 3.3 compiler and STL implementation.

This index is created by sliding a width-w window across each intergenic sequence and noting each subsequence seen. The *canonical* function used below returns the lexicographically lesser sequence of the sequence and its reversed complement.

for $0 \leq inumber < |\text{INPUT-SEQUENCES}|$ **do**

$i \leftarrow \text{INPUT-SEQUENCES}_{inumber}$

for $0 \leq j < length(i) - w$ **do**

$subseq \leftarrow i_{j \ldots j+w-1}$

$\text{INDEX}\{canonical(subseq)\}.add(inumber)$

return INDEX

This routine is used twice, once with INPUT-SEQUENCES $= P$, and once with INPUT-SEQUENCES $= A$.

2.8.1 Occurrences of words

The INDEX described above only indexes subsequences, not words. However, in §2.9, we will need to know the occurrences of words. We can find the occurrences of a word by taking the union of all possible instantiations of a word.

For example, the occurrences of the word `A[CT][GT]` is the following union:

$$\begin{aligned} & \text{INDEX}\{canonical(\mathcal{ACG})\} \\ \cup\ & \text{INDEX}\{canonical(\mathcal{ATG})\} \\ \cup\ & \text{INDEX}\{canonical(\mathcal{ACT})\} \\ \cup\ & \text{INDEX}\{canonical(\mathcal{ATT})\} \end{aligned}$$

The running time of the implementation of finding the occurrences of a word is exponential in the number of wildcards elements h (although with $h = 2$, the value is not large.) Furthermore, this routine behaves poorly with respect to the memory hierarchy of modern computers. Whether the index is implemented as a balanced tree (the C++ `map`) or as a hash table (`hash_map`), the keys (sequences) which are the instantiations of the word are likely to be scattered all over main memory. Therefore, the processor will likely stall waiting for memory to be read for each of the exponential number of instantiations of the word.

For these reasons, finding the occurrences of a word, especially the occurrences

of a word among all intergenic sequences A, for hundreds of thousands of words, was the most computationally time-consuming portion of the entire algorithm (not including the sum-of-products approximation of §3.4, which was so slow it was mostly abandoned).

2.9 Enumerate and Test all Motifs

This routine corresponds to the exhaustive enumeration of §2.3.1. However, the TARGETS-SET bit vector created in §2.7 is used as a quick filter to determine if a word is in the positive set P. Recall that if a bit corresponding to the hashed value of a word is true, then the word *might* be present. If false, then we know that the word was definitely not present (at least twice). The routine *find-occurrences-of-word* is described above in §2.8.1.

INDEX$_A$ $\leftarrow$ *create-index*(A)
INDEX$_P$ $\leftarrow$ *create-index*(P)
foreach $m \in M$ **do**
  if TARGETS-SET[m] **then**
    FILTERED-POSITIVES $\leftarrow$ *find-occurrences-of-word*(m,INDEX$_P$)
    if FILTERED-POSITIVES $\neq \emptyset$ **then**
      FILTERED-ALL $\leftarrow$ *find-occurrences-of-word*(m,INDEX$_A$)
      test-for-significance(FILTERED-POSITIVES, FILTERED-ALL)

2.10 Implementation verification

The word-counting algorithm was first rapid prototyped in the functional programming language Haskell, then re-implemented in C++. On small "toy" test cases (the Haskell implementation was too slow for use on real data), both algorithms gave the same results.

In agreement with other anecdotal reports comparing Haskell to other program-

ming languages, the C++ implementation took approximately 5 times more programmer effort than the Haskell implementation.

Chapter 3

Statistical tests

Having determined which sequences contain a word m, we now wish to determine if m occurs in the positive sequences P more often then would be expected by chance.

3.1 Null hypothesis

We must therefore first define a *null hypothesis* of what in fact is expected by chance. Biologically, the null hypothesis corresponds to the situation that m is not the motif for the transcription factor. To be able to statistically reject the null hypothesis, we must quantify what we expect to see if the null hypothesis is true.

This chapter will explore the "Random Selection Null Hypothesis" of [3], which states that when the null hypothesis is true, the positive sequences are "randomly selected" from among all the intergenic sequences. For their model, "randomly selected" means "all sequences are equally likely to be chosen without replacement". For this definition of "randomly selected", they give a formula for the probability that k sequences match the word m by chance alone.

$$P_{hyper}(k \mid n, K, N) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}} \tag{3.1}$$

where $n = |P|$, $N = |A|$, and K is the total number of sequences matching the word m. The above formula is the hyper-geometric probability distribution.

Using this formula we can calculate a p-value that the null hypothesis is true. The p-value sums the tail of the probability distribution for $k' \geq k$.

$$\text{p-value}(k) = \sum_{k'=k}^{n} P(k' \mid n, K, N) \tag{3.2}$$

In the above formula is $P = P_{hyper}$ for the hyper-geometric case; however, we will use this summation with other probability distributions P later in the chapter.

To minimize roundoff error, the sum is computed backwards starting from n going down to k:

```
for (kprime=n; kprime>=k; --kprime) {
   sum+=p(...)
}
```

As an aside, we note that to truly minimize roundoff error, the values to be summed ought be placed in a priority queue data structure such that the smallest value is always at the head of the queue. Each iteration, the smallest two values should be popped off the queue and the sum re-inserted into the queue. However, this intricate summing procedure was not implemented.

Because the probabilities being summed are frequently numbers of extremely small magnitudes, they are best manipulated as log-probabilities. The equations below express the logarithm of a sum in terms of the logarithms of the values being summed.

$$\log(a+b) \tag{3.3}$$

$$= \log a + \log(1 + \exp(\log b - \log a)) \qquad (a \geq b) \tag{3.4}$$

$$= \log b + \log(1 + \exp(\log a - \log b)) \qquad (a < b) \tag{3.5}$$

3.2 Sequences chosen by length

Instead of "all sequences equally likely" as the behavior under the null hypothesis, this thesis proposes the following alternative:

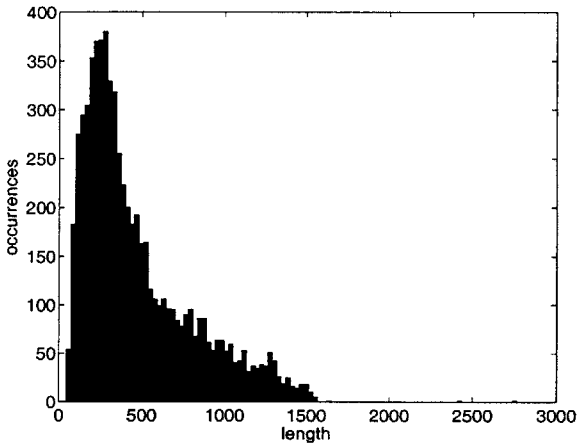


Figure 3-1: Distribution of integenic sequence lengths.

Sequences are selected with probability proportional to the sequence's length.

The motivation for this alternative stems from the problem that the sequences from the ChIP experiments were of different lengths. Barash, *et al.* did not concern themselves with sequences of varying length because they explicitly chose their sequences to be all the same length (1000 nucleotides upstream of from the start of the ORF). They also specifically developed their hyper-geometric statistic under the assumption that all sequences are the same length. However, the data from ChIP experiments do not afford us this luxury (Figure 3-1).

The modification is plausible: given no other knowledge about the transcription factor, a longer sequence is more likely to contain the transcription factor's true motif. However, the modification assumes that the intergenic sequences from the ChIP experiment (which occasionally cut a long intergenic sequence into several pieces) were not constructed in special some way that violates this assumption of "selection probability proportional to length." We use the following notation for the lengths of all, positive, and negative sequences.

$$AL = \{\text{length}(s) \mid s \in A\} \tag{3.6}$$

$$PL = \{\text{length}(s) \mid s \in P\} \tag{3.7}$$

$$NL = \{\text{length}(s) \mid s \notin P\} \tag{3.8}$$

Note that the quantities are *bags* (or multi-sets) because different sequences of identical lengths are allowed in each bag. We use use the symbol $\uplus$ to mean bag union, for example $PL \uplus NL = AL$.

Having defined the null hypothesis, we can now attempt to derive a formula (analogous to the hyper-geometric distribution formula) for the probability that k sequences which match the word are selected. Unfortunately, I was unable to derive an efficiently (polynomial-time) computable formula for this probability. (But I was frustratingly unable to prove any complexity bounds, such as NP-completeness, either.) Therefore, below we discuss two efficiently computable approximations. (The second, despite being polynomial-time, will be too slow for practical use.)

3.3 Binomial approximation

Instead of selecting sequences without replacement, one approximation is to select sequences with replacement. The probability of selecting exactly k sequences is binomial:

$$P_{binom}(k \mid n, K, N) = \binom{n}{k} r^k (1-r)^{n-k}. \tag{3.9}$$

where r is the proportion of total sequence (total lengths) containing the word.

$$r = \frac{\sum PL}{\sum AL}$$

The binomial approximation, being quick to calculate, was the principal statistical test used in this thesis.

3.4 Sum of products approximation

The second approximation has functional form similar to the hyper-geometric distribution. Its weakness is that apart from its "inspirations" described below, I was unable to derive any formal justification as to why it is an approximation, or how

good an approximation it is. Although this approximation was calculated for the few hundred very significant (by binomial approximation) motifs, it was too slow to use as a general statistical measure of significance. Therefore, this section serves mostly as a curious mathematical aside.

The inspirations for this approximation came from two different directions. The first was to take the hyper-geometric formula (Equation 3.1) and modify it to allow for sequences of different lengths. The simple modification chosen was to replace the binomial function with something different. The second inspiration came while laboriously writing out the expressions for the exact choose-without-replacement probability for small sets PL and NL. It was noted that the resulting expressions frequently contained *products* (and sums of products) of lengths with each other.

We also define two straightforward constraints that we would like the approximation to obey. We require approximation take less than exponential time to calculate. We require that the approximation behave like a probability. That is, the values must be bounded between zero and unity, and they must sum appropriately to unity so that a meaningful p-value can be calculated.

From the inspirations and constraints, we define the following function to replace the binomial: *Sum-prod*(S, k) is the sum of the products of all size-k sub-bags of a bag S. (We are using bags instead of sets to allow for multiple identical elements.) This function was chosen for its simplicity and elegance; it probably is not the only possible choice satisfying the constraints.

$$\textit{Sum-prod}(S, k) = \sum_{\substack{T \subseteq S \\ |T|=k}} \left(\prod_{t \in T} t \right) \tag{3.10}$$

Sum-prod can be computed efficiently using dynamic programming, using the following recurrence relation (for $k > 0$):

$$\textit{Sum-prod}(S \uplus \{x\}, k) = \textit{Sum-prod}(S, k) + x \cdot \textit{Sum-prod}(S, k-1) \tag{3.11}$$

The recurrence follows from the fact that the set of sub-bags of $S \uplus \{x\}$ can be partitioned in two: those not containing x and those that do. For the latter, x can be factored out of each term, resulting in second term of Equation 3.11. The recurrence uses the following base cases.

$$\begin{aligned} \textit{Sum-prod}(S, 0) &= 1 && (3.12) \\ \textit{Sum-prod}(\emptyset, k) &= 0 \qquad (k > 0) && (3.13) \\ \textit{Sum-prod}(S, k) &= 0 \qquad (k < 0) && (3.14) \end{aligned}$$

Note that Eq. 3.12 holds for $S = \emptyset$.

It does not matter what order the elements are added. In the first set of equations below, a is added first, then b. In the second set, b then a. Both orderings expand via the recurrence to the same expression.

$$\begin{aligned} & SP([S \uplus a] \uplus b, k) && (3.15) \\ = \; & SP(S \uplus a, k) + b \cdot SP(S \uplus a, k-1) && (3.16) \\ = \; & [SP(S, k) + a \cdot SP(S, k-1)] + b \cdot [SP(M, k-1) + a \cdot SP(M, k-2)] && (3.17) \\ = \; & SP(S, k) + a \cdot SP(S, k-1) + b \cdot SP(M, k-1) + ab \cdot SP(M, k-2) && (3.18) \end{aligned}$$

$$\begin{aligned} & SP([S \uplus b] \uplus a, k) && (3.19) \\ = \; & SP(S \uplus b, k) + a \cdot SP(S \uplus b, k-1) && (3.20) \\ = \; & [SP(S, k) + b \cdot SP(S, k-1)] + a \cdot [SP(M, k-1) + b \cdot SP(M, k-2)] && (3.21) \\ = \; & SP(S, k) + b \cdot SP(S, k-1) + a \cdot SP(M, k-1) + ab \cdot SP(M, k-2) && (3.22) \end{aligned}$$

(We occasionally write *SP* for *Sum-prod* for compactness.)

We observe that when all lengths are unity, the function reduces to the binomial

function:

$$\binom{n}{k} = \textit{Sum-prod}(\{\underbrace{1, 1, \ldots, 1}_{n}\}, k) \tag{3.23}$$

Inspired by this observation, first we note the occurrences of the binomial function in the hyper-geometric distribution in Equation 3.1 on page 31. We define a function with similar form, but with the binomial function replaced with *Sum-prod*:

$$P_{\textit{Sum-prod}}(k \mid n, PL, NL, AL) = \frac{\textit{Sum-prod}(PL, k) \cdot \textit{Sum-prod}(NL, n-k)}{\textit{Sum-prod}(AL, n)} \tag{3.24}$$

Example computations of *Sum-prod* are given in §3.6.

3.4.1 $P_{\textit{Sum-prod}}$ behaves like a probability

Despite the *ad hoc* set of "inspirations" upon which this approximation was based, we find (this was actually just stroke of good luck) that $P_{\textit{Sum-prod}}$ behaves appropriately as a probability, so a p-value can be computed by summing the tail of the distribution with Equation 3.2 on page 32.

We will first show that Equation 3.24 sums to unity when summed over k, that is:

$$\sum_{k=0}^{n} \frac{\textit{Sum-prod}(A, k) \cdot \textit{Sum-prod}(B, n-k)}{\textit{Sum-prod}(A \uplus B, n)} = 1 \tag{3.25}$$

It will follow as a simple corollary that the value is bounded between zero and 1. Multiplying Eq. 3.25 by the denominator yields:

$$\textit{Sum-prod}(A \uplus B, n) = \sum_{k=0}^{n} \textit{Sum-prod}(A, k)\textit{Sum-prod}(B, n-k) \tag{3.26}$$

We will prove this equation via induction on the elements of B. First, for the base

case $B = \emptyset$, the left-hand side simplifies trivially:

$$\textit{Sum-prod}(A \uplus \emptyset, n) = \textit{Sum-prod}(A, n) \tag{3.27}$$

For the right-hand side

$$\begin{aligned} & \sum_{k=0}^{n} \textit{Sum-prod}(A, k)\textit{Sum-prod}(\emptyset, n-k) && (3.28)\\ = \;& \textit{Sum-prod}(A, n)\textit{Sum-prod}(\emptyset, n-n) && (3.29)\\ = \;& \textit{Sum-prod}(A, n) \cdot 1 && (3.30)\\ = \;& \textit{Sum-prod}(A, n). && (3.31) \end{aligned}$$

Eq. 3.29 results from the base cases of Eq. 3.12 and Eq. 3.13. The only nonzero term is occurs when $k = n$.

Next, we assume the inductive hypothesis and show that Eq. 3.26 is true for $B \leftarrow B \uplus \{b\}$. The left-hand side expands to

$$\begin{aligned} & \textit{Sum-prod}(A \uplus [B \uplus \{b\}], n) && (3.32)\\ = \;& \textit{Sum-prod}([A \uplus B] \uplus \{b\}, n) && (3.33)\\ = \;& \textit{Sum-prod}(A \uplus B, n) + b \cdot \textit{Sum-prod}(A \uplus B, n-1) && (3.34) \end{aligned}$$

Eq. 3.34 results from the recurrence (Eq. 3.11) with $S = A \uplus B$. The right hand side

expands to:

$$\sum_{k=0}^{n} SP(A,k)SP(B \uplus \{b\}, n-k) \tag{3.35}$$

$$= \sum_{k=0}^{n} SP(A,k)[SP(B,n-k) + b \cdot SP(B,n-k-1)] \tag{3.36}$$

$$= \sum_{k=0}^{n} SP(A,k)SP(B,n-k) + b \cdot \sum_{k=0}^{n} SP(A,k)SP(B,n-k-1) \tag{3.37}$$

$$= SP(A \uplus B, n) \tag{3.38}$$

$$+b \cdot \left[\sum_{k=0}^{n-1} SP(A,k)SP(B,n-k-1) + SP(A,n)SP(B,n-n-1)\right] \tag{3.39}$$

$$= SP(A \uplus B, n) + b \cdot [SP(A \uplus B, n-1) + SP(A,n)SP(B,-1)] \tag{3.40}$$

$$= SP(A \uplus B, n) + b \cdot [SP(A \uplus B, n-1) + SP(A,n) \cdot 0] \tag{3.41}$$

$$= SP(A \uplus B, n) + b \cdot SP(A \uplus B, n-1) \tag{3.42}$$

Eq. 3.36 results from applying the recurrence (Eq. 3.11). Eq. 3.37 distributes $SP(A,k)$ over the bracketed sum and separates the equation into two sums. Expression 3.38 uses the inductive hypothesis for $SP(A \uplus B, n)$. Expression 3.39 separates out the special case $k = n$ from the sum. Eq. 3.40 uses the inductive hypothesis for and $SP(A \uplus B, n-1)$ Eq. 3.41 uses the base case Eq. 3.14 for negative k.

Since Eq. 3.34 equals Eq. 3.42 we have proved the inductive case. Therefore, Eq. 3.26 holds for any B (of finite size).

Assuming that all sequence lengths are positive, conclude that the value of Eq. 3.24 is always non-negative. Since the sum over k is unity, and no values are negative, we conclude that $P_{Sum\text{-}prod}$ is always bounded between 0 and 1.

3.5 Which approximation was used?

Of the approximations given in this chapter, the algorithm principally used the choose-with-replacement binomial approximation. The sum-of-products approximation, despite the speedup from dynamic programming, was still to slow for use. Note that the running time of sum-of-products approximation is $O(N^2)$, where $N = |AL|$,

because of the $O(N^2)$ entries in the dynamic programming tableau. For the yeast genome, $N \approx 6700$, so millions of arithmetic operations would have been needed for each (of the hundreds of thousands) of words examined.

3.6 Sample computations

Consider a simple scenario with positive sequence with lengths $PL = \{p_1 = 50, p_2 = 51\}$ and negative sequences with lengths $NL = \{n_1 = 2, n_2 = 3\}$. The positive and negative lengths are extremely different to highlight the differences in the different approaches. To simplify notation, let W be the total length of all sequences, $W = p_1 + p_2 + n_1 + n_2 = 106$.

The exact probability of selecting exactly k positive sequences in $n = 2$ draws is given below:

$k = 0$	$\frac{n_1}{W} \cdot \frac{n_2}{W-n_1} + \frac{n_2}{W} \cdot \frac{n_1}{W-n_2}$	$=$	$\frac{621}{567736}$	$\approx$ 0.00109
$k = 1$	$\frac{n_1}{W} \cdot \frac{p_1+p_2}{W-n_1} + \frac{n_2}{W} \cdot \frac{p_1+p_2}{W-n_2}$ $+\frac{p_1}{W} \cdot \frac{n_1+n_2}{W-p_1} + \frac{p_2}{W} \cdot \frac{n_1+n_2}{W-p_2}$	$=$	$\frac{27205}{206206}$	$\approx$ 0.13193
$k = 2$	$\frac{p_1}{W} \cdot \frac{p_2}{W-p_1} + \frac{p_2}{W} \cdot \frac{p_1}{W-p_2}$	$=$	$\frac{28305}{32648}$	$\approx$ 0.86698
	Sum	$=$	1	

The hyper-geometric probability (Equation 3.1) of choosing exactly k positive sequences in $n = 2$ draws is given below. For this example, we have $N = 4$ and $K = 2$. The percentage error with respect to the exact value calculated above is given in the final column.

k	Value	%Error
$k=0$	$\frac{\binom{2}{0}\binom{2}{2}}{\binom{4}{2}} = \frac{1}{6} \approx 0.16667$	15137%
$k=1$	$\frac{\binom{2}{1}\binom{2}{1}}{\binom{4}{2}} = \frac{2}{3} \approx 0.66667$	405%
$k=2$	$\frac{\binom{2}{2}\binom{2}{0}}{\binom{4}{2}} = \frac{1}{6} \approx 0.16667$	-81%
	Sum $= 1$	

The binomial approximation yields the following values and errors. We use $r = \frac{p_1+p_2}{W} = \frac{101}{106}$.

k	Value	%Error
$k=0$	$\binom{2}{0}r^0(1-r)^2 = \frac{25}{11236} \approx 0.00222$	103%
$k=1$	$\binom{2}{1}r(1-r) = \frac{505}{5608} \approx 0.08989$	-32%
$k=2$	$\binom{2}{2}r^2(1-r)^0 = \frac{10201}{11236} \approx 0.90789$	4.7%
	Sum $= 1$	

The dynamic programming tableau for $\mathit{Sum\text{-}prod}(AL, n) = \mathit{Sum\text{-}prod}(\{50, 51, 2, 3\}, n)$ is given below. Note that the rows for $n = 3$ and $n = 4$ are not needed for the computation; they are given for completeness.

Sum-prod(S,n)	$S=\{50,51,2,3\}$	$\{51,2,3\}$	$\{2,3\}$	$\{3\}$	$\emptyset$
$n=0$	1	1	1	1	1
$n=1$	106	56	5	3	0
$n=2$	$50\cdot 56+261=3061$	$51\cdot 5+6=261$	$2\cdot 3+0=6$	0	0
$n=3$	$50\cdot 261+306=13356$	$51\cdot 6+0=306$	0	0	0
$n=4$	$50\cdot 306+0=15300$	0	0	0	0

The tableau was filled from the upper right hand corner using the template below:

$\{A,\ldots\}$	
$\vdots$	$\vdots$
	B
$\boxed{A\cdot B+C}$	C

The other tableaux for PL and NL individually are trivial and are not shown here. The sum-of-products approximation yields the following values and errors.

k	Value				%Error
$k=0$	$\frac{\textit{Sum-prod}(PL,0)\cdot\textit{Sum-prod}(NL,2)}{\textit{Sum-prod}(PL\uplus NL,2)}$	$=$	$\frac{6}{3061}$	$\approx$ 0.00196	79%
$k=1$	$\frac{\textit{Sum-prod}(PL,1)\cdot\textit{Sum-prod}(NL,1)}{\textit{Sum-prod}(PL\uplus NL,2)}$	$=$	$\frac{505}{3061}$	$\approx$ 0.16498	25%
$k=2$	$\frac{\textit{Sum-prod}(PL,2)\cdot\textit{Sum-prod}(NL,0)}{\textit{Sum-prod}(PL\uplus NL,2)}$	$=$	$\frac{2550}{3061}$	$\approx$ 0.83306	-3.9%
	Sum	$=$	1		

3.7 Comparing the approximations

Table 3.1 compares the relative errors of the three approximations for the simple scenario above. It is believed that this small example is representative of the results for larger sets. The sum-of-products approximation does the best, however, it is too slow to use for large problems. The binomial approximation does better than the

k	Hyper-geometric	Binomial	*Sum-prod*
$k = 0$	15137%	103%	79%
$k = 1$	405%	-32%	25%
$k = 2$	-81%	4.7%	-3.9%

Table 3.1: Relative errors for the small sample computation.

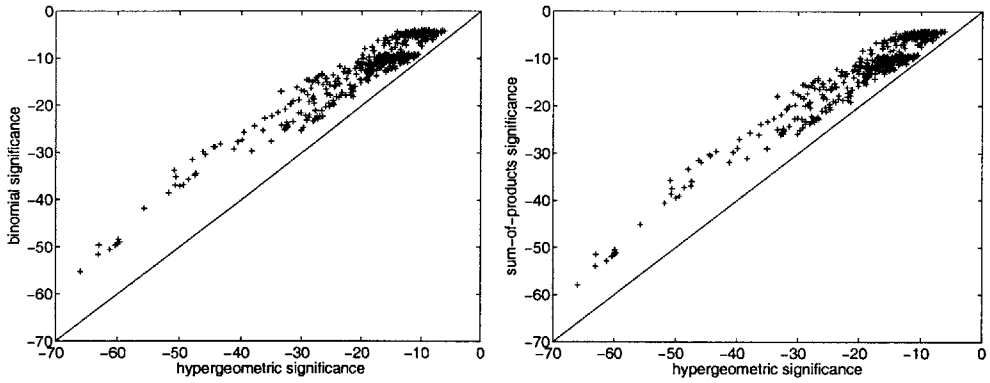


Figure 3-2: Comparing the hyper-geometric significance with binomial and sum-of-products.

hyper-geometric, so the binomial will be used as the principal method of determining statistical significance for most of the experiments.

For a few hundred highly significant (by the binomial approximation) motifs discovered, the sum-of-products approximation and the hyper-geometric significance was also calculated. Figures 3-2 and 3-3 plot their corresponding values. Admittedly, the set of points might be biased because of the selection. However, the figures seem to indicate that the binomial significance is a good substitution for the expensive-to-calculate sum-of-products significance (with the binomial slightly understating significance) while the hyper-geometric significance tends to overstate significance.

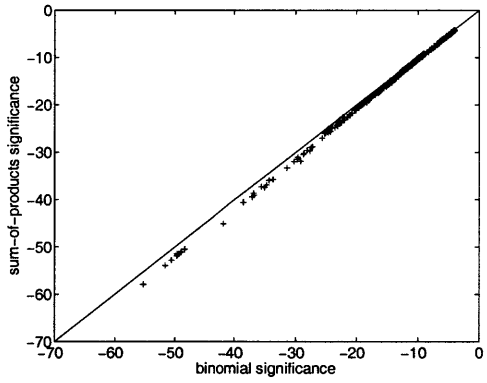


Figure 3-3: Comparing the sum-of-products significance with binomial significance.

Chapter 4

Results and Discussion

This chapter is organized into the following sections. §4.1 validates the algorithm by attempting to replicate known motifs. §4.2 assesses the running-time performance of the algorithm. §4.3 presents potential new motifs discovered by the algorithm. Finally, §4.4 discusses ideas for future work.

4.1 Validation

This section measures and compares the algorithm's motif discovery performance. For an absolute measure, the algorithm was run on binding data for transcription factors whose motifs were previously discovered and confirmed biologically. For a comparative measure, the same data were analyzed with the motif discovery programs MEME[1] and MDscan [9]. The algorithm was also run on differently processed binding data for each transcription factor to determine the effect of the type binding data on motif discovery.

4.1.1 Program parameters

MDscan was run through the web interface with the following parameters:

- Motif width: 7
- Number of top sequences to look for candidate motifs: 10

- Number of candidate motifs for scanning the rest sequences: `20`
- Report the top final `10` motifs found
- Precomputed genome background model: `S. cerevisiae intergenic`

MEME was run with the command-line parameters `-dna -w 7 -nmotifs 10 -revcomp -bfile $MEME/tests/yeast.nc.6.freq`. The parameters direct MEME attempt to discover 10 motifs of width 7 on either strand using the pre-computed order-6 Markov background model of the yeast non-coding regions.

4.1.2 Binding data

Three different criteria for positive sequences were used. That is, three different methods were used to determine which sequences are bound by a transcription factor. The first two are a simple p-value threshold on the ChIP experiment[10] (*not* related to the p-values calculated the statistical tests of Chapter 3). The last uses the GRAM gene modules described in [2] which incorporates both binding data and clustered expression level data.

1. Bound probes from Young Lab, cutoff p-value 0.001
2. Bound probes from Young Lab, cutoff p-value 0.0001
3. Gene modules under YPD [2] (see Appendix A).

4.1.3 Scoring method

To score the performance of both this thesis's algorithm, and MEME and MD-Scan, the discovered motifs were compared against the consensus sequences for transcription factors (Table 4.1) which were gathered from the literature.

To compare a discovered motif with the consensus, all possible alignments of the motif (or its reverse complement) and the consensus are tried, and alignment yielding the smallest distance is taken as the score of the discovered motif.

Transcription Factor	Consensus motif sequence
ABF1	TCRNNNNNNACG
CBF1	RTCACRTG
GAL4	CGGNNNNNNNNNNCCG
GCN4	TGACTCA
GCR1	CTTTCC
HAP2	CCAATNA
HAP3	CCAATNA
HAP4	CCAATNA
HSF1	GAANNTTTCNNGAA
INO2	ATGTGAAA
MATa1	TGATGTANNT
MCM1	CCNNNWWRGG
MIG1	WWWWSYGGGG
PHO4	CACGTG
RAP1	RMACCCANNCAYY
REB1	CGGGTRR
STE12	TGAAACA
SWI4	CACGAAA
SWI6	CACGAAA
YAP1	TTACTAA

Table 4.1: Consensus sequences

A	$(1, 0, 0, 0)$
[AG]	$(\frac{1}{2}, 0, \frac{1}{2}, 0)$
[GT]	$(0, 0, \frac{1}{2}, \frac{1}{2})$
•	$(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$

Table 4.2: Some example 4-tuple probabilities

The distance between two elements, for example [AG] and [GT] is the squared Euclidean distance of the probability distribution specified by the wildcard or base. In other words, each base or wildcard represents a 4-tuple specifying the respective probability of $\mathcal{A}$, $\mathcal{C}$, $\mathcal{G}$, or $\mathcal{T}$. The 4-tuples for a few example word elements are listed in Table 4.2.

For example, the distance between [AG] and [GT] is

$$\left(\frac{1}{2} - 0\right)^2 + (0 - 0)^2 + \left(\frac{1}{2} - \frac{1}{2}\right)^2 + \left(0 - \frac{1}{2}\right)^2 = \frac{1}{2}$$

Note that no square root is taken in the distance. The distance between two aligned words is the sum of the distances for each position. In order to allow alignments to hang off of the edge of each other (especially when the consensus is of a different length than discovered motif), the consensus sequences is implicitly assumed to be padded infinitely on either end by •. After computing the sum of the distances, the value is floored from below at 0.833. This value was chosen "by eye" as a value indicating a "nearly perfect" match.

We use the squared Euclidean distance instead of the more standard KL divergence for the distance between two probability distributions. The KL divergence $D(p\|q)$ becomes infinite if the approximating distribution q has zero probability for an event that has positive probability under p. This unfortunate behavior occurs when the consensus and the discovered motif disagree completely at a position, for example, matching G against A.

Each motif discovery program outputs many candidate motifs. MEME and MD-Scan were configured to produce 10, while this thesis's algorithm produced as many were above a significance threshold of 10^{-4}. Therefore, both the single best (high-

est significance) motif, and the best motif among the top 10 motifs reported by each algorithm are scored. A discovered motif scored "correct" if its distance from the consensus is less than 2.083. (This weird constant is $\frac{300}{144}$, an artifact the implementation multiplying probabilities by a factor of 12 so that arithmetic could be performed using integers only.) This threshold was chosen "by eye" as a value for which discovered motifs seemed close to consensus motifs.

It should be noted that choosing from only the top 10 (or top 1) ranked motifs is a somewhat problematic way of comparing motif discovery algorithms, and might appear to penalize this thesis's algorithm more than other algorithms. If the algorithm is fooled by a significant false motif (perhaps the binding site of a different transcription factor that always binds with the transcription factor in question) then it might be fooled many times over by small variations of the word (for an example of such variations, see Table 4.7), all of which may be more significant than the true motif. The many variations of the false motif may push the ranking of the true motif out of the top 10.

Nevertheless, we will charge along with the top-1 and top-10 comparison method, but we also present Figure 4-1 for a different perspective. The figure examines the best motif among all the significant (threshold 10^{-4}) words discovered by the algorithm, and then compares its significance with the significance of the most significant motif. The figure illustrates that although the best motif may rank lower than tenth, its significance often comparable to the most significant motif.

4.1.4 Validation results

Table 4.3 gives the correct motifs found by the algorithm and other motif-discovery algorithms on different data sets. We can make the following observations:

- The best performance was this thesis's algorithm using binding data with threshold p-value 0.001.
- Choosing a more rigorous threshold for the binding data, namely 0.0001, resulted in poorer performance, most likely because of insufficient positive inter-

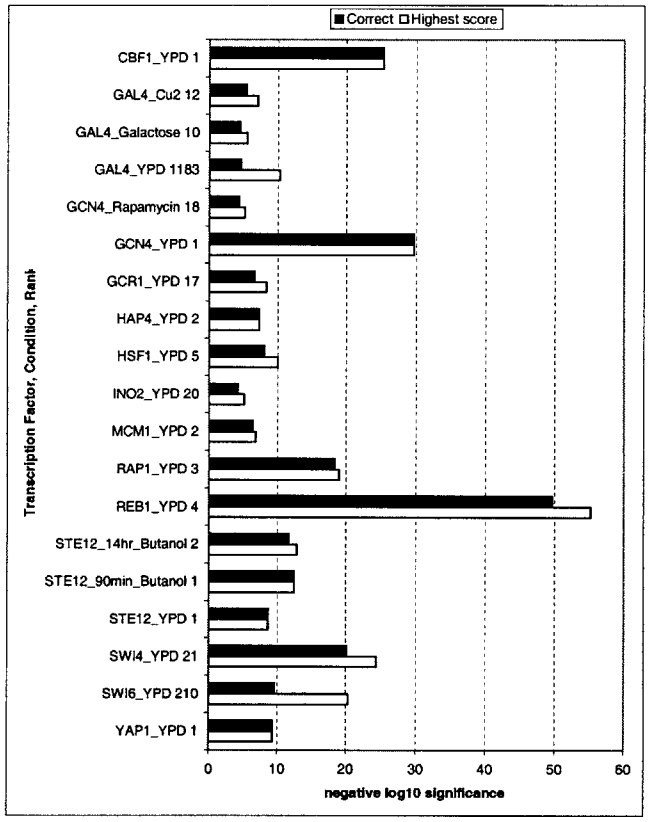


Figure 4-1: Significance of most-correct motif (black) versus significance of the top-scoring motif (white). The number after each transcription factor is the rank of the most-correct motif. The "most-correct" motif is the motif with the smallest distance from the consensus sequence among all significant words discovered. Note that distance is floored at 0.83; if two words have the same distance, the one with the higher (smaller) rank is preferred.

genic sequences for a significant result.

- Using GRAM modules algorithm to define the set of positive intergenic sequences did slightly poorer than using the raw binding data. However, the modules result did find two correct motifs, for HAP2 and HAP3, that the raw binding data did not (at the cost of failing to find GAL4, MCM1, PHO4, and SWI4.)

- The algorithm finds slightly more correct motifs than MEME or MDscan.

alg: thesis data: p=0.001 choose: 10	14	CBF1	GAL4	GCN4	GCR1	HAP4	HSF1	INO2	MCM1	PHO4	RAP1	REB1	STE12	SWI4	YAP1
alg: MDscan data: p=0.001 choose: 10	12	CBF1	GCN4	HAP4	HSF1	INO2	MATa1	PHO4	RAP1	REB1	STE12	SWI4	YAP1		
alg: MEME data: p=0.001 choose: 10	10	CBF1	GAL4	GCN4	GCR1	HSF1	PHO4	RAP1	REB1	STE12	SWI4				
alg: thesis data: p=0.001 choose: 1	10	CBF1	GAL4	GCN4	HAP4	HSF1	PHO4	RAP1	REB1	STE12	YAP1				
alg: MDscan data: p=0.001 choose: 1	9	CBF1	GCN4	HSF1	PHO4	RAP1	REB1	STE12	SWI4	YAP1					
alg: MEME data: p=0.001 choose: 1	0														
alg: thesis data: GRAM choose: 10	12	CBF1	GCN4	GCR1	HAP2	HAP3	HAP4	HSF1	INO2	RAP1	REB1	STE12	YAP1		
alg: thesis data: GRAM choose: 1	9	CBF1	GCN4	HAP2	HAP3	HAP4	INO2	RAP1	REB1	STE12					
alg: thesis data: p=0.0001 choose: 10	12	GAL4	GCN4	HAP2	HAP4	HSF1	MCM1	PHO4	RAP1	REB1	STE12	SWI4	YAP1		
alg: MEME data: p=0.0001 choose: 10	12	GCN4	HAP3	HAP4	HSF1	MATa1	MIG1	PHO4	RAP1	REB1	STE12	SWI4	SWI6		
alg: thesis data: p=0.0001 choose: 1	9	GCN4	HAP2	HAP4	HSF1	MCM1	PHO4	REB1	STE12	YAP1					

Table 4.3: Verified consensus motifs. The first column gives the algorithm, which positive sequences were used, and the number of top ranked motifs which were tested for correctness. The second column is the number of correct motifs, followed by the transcription factors whose correct motifs were found.

4.2 Algorithm Running Time

This section will present analysis running time of the algorithm over the many binding data sets for many transcription factors. Because no effort was made to ensure a random sampling of experiments, the results in this section are only intended to be a rough indication of the performance of the algorithm. Furthermore, the times reported are wall-clock times, so some of the measurements are skewed by other users of the compute cluster where the calculations were performed.

Figure 4-2 gives a histogram of wall-clock running time for the various binding data sets, on a 1.667 GHz Athlon system. Figure 4-3 plots the running time versus the number of positive sequences. The running time flattens out after 50 bound sequences because of saturation: all possible words are present.

On the average, 96.6% of the running time was spent on the exhaustive enumeration of all words (§2.9). Figure 4-4 illustrates the running time of that routine versus the number of words evaluated (words whose hashed value was marked in the bitvector created in §2.7). The running time is roughly linearly dependent of the number of words. The interpolated line yields an estimate of 2.0 milliseconds per evaluated word. The figure illustrates that, for non-saturated experiments, filtering by the bitvector (and to a lesser extent, skipping words that *find-occurrences-of-word* reports are not present in the positive sequences) is effective in reducing running time.

Of the time not spent on exhaustive enumeration, the next most time-consuming task was creating the index INDEX_A in §2.8 of all intergenic sequences. This typically took about 30 seconds. This operation is always exactly the same for all experiments—the only reason the time varied was because of user load. Some time could have been saved by pre-computing INDEX_A. However, it was decided that 30 seconds was not too long to wait.

The initial hashing (§2.5) also took significant running time. Its running time with respect to the number of positive sequences is shown in Figure 4-5. Note that the *tableSize* was 10,000,000 for the experiments.

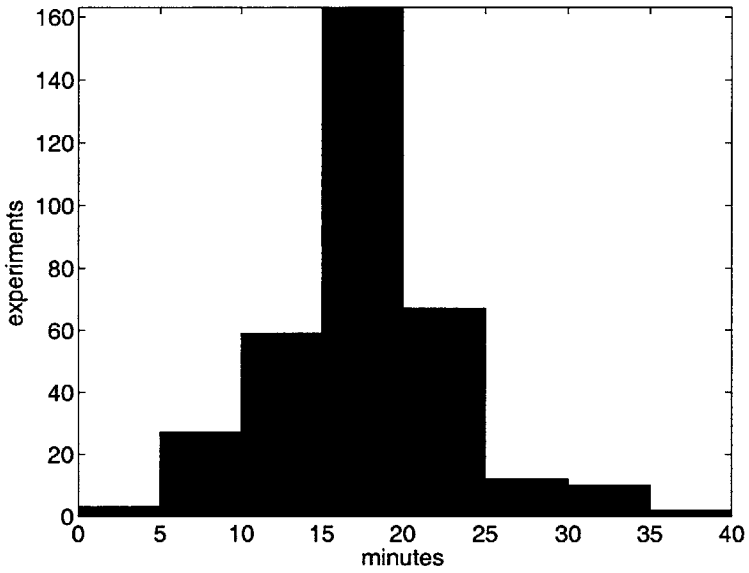


Figure 4-2: Histogram of running times of the algorithm, in minutes.

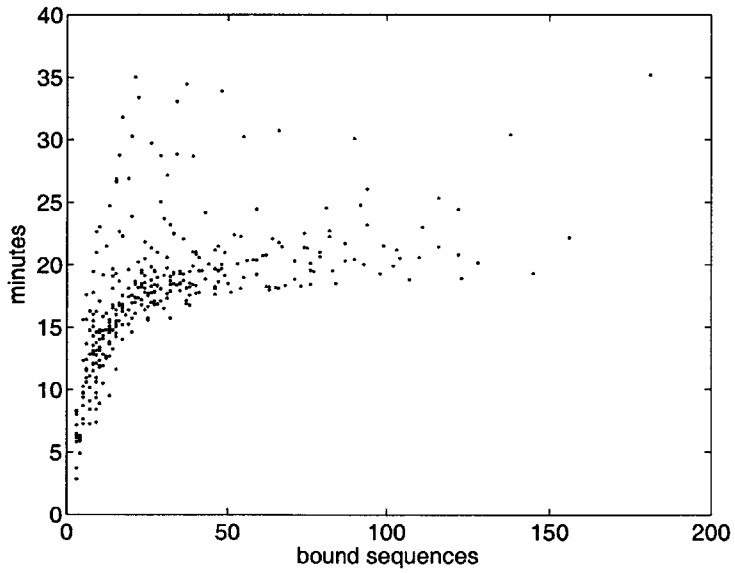


Figure 4-3: The running time, in minutes, versus the number of positive sequences. The asymptotic effect (most visible in the lower envelope) is due to saturation; nearly all words of width seven with up to two wildcard elements were evaluated.

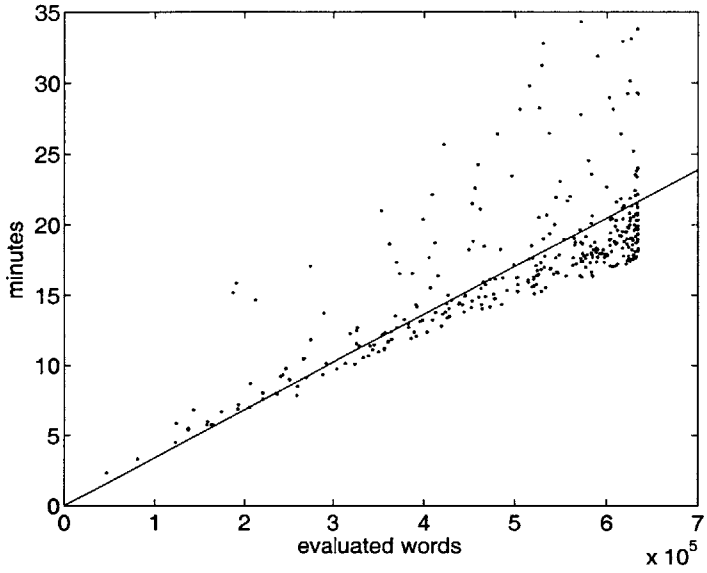


Figure 4-4: A graph of the running time of the exhaustive enumeration stage versus the number of words present in the positive sequences. The interpolated line yields an estimate of 2.0 milliseconds per evaluated word.

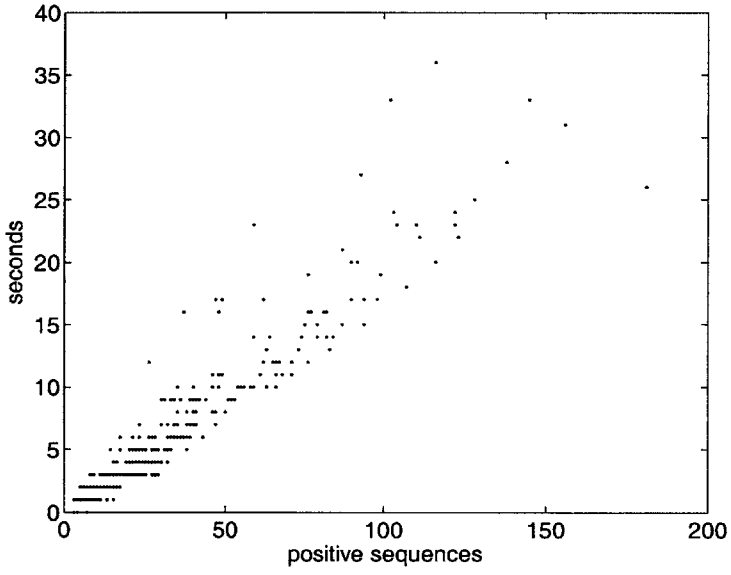


Figure 4-5: The running time of the initial hashing routine, in seconds, versus the number of positive sequences.

4.2.1 Performance scaling experiments

A motif width of 7 with up to 2 wildcard elements was chosen because of running time constraints. Any wider or more wildcard elements would have taken too much time to run on the approximately 340 data sets for different transcription factors, conditions, and binding thresholds.

However, to measure the how the algorithm scales with problem size, the algorithm was run with different motif widths and number of holes. The experiments were run on a representative (somewhat large) example data set, STE12 under YPD conditions with the binding data threshold at 0.001, which has 74 positive sequences. Figure 4-6 shows the run times on the 1.667GHz AMD Athlon system. For all the experiments, the *tableSize* was $1.7 \cdot 10^9$, approximately the largest that could fit in main memory.

The larger *tableSize* allows us to estimate the time spent on the on the loop around the call to the *advance*() method in the initial hashing routine (§2.5 on page 21), which becomes quite expensive for large value of *tableSize*. For words of width 7 and up to two wildcard elements, the running time of just the initial hashing routine at *tableSize* 10,000,000 took 14 seconds, while the running time at *tableSize* $1.7 \cdot 10^9$ took 550 seconds. From these data we can estimate that a single call to *advance*() took approximately $4 \cdot 10^{-9}$ seconds, and accounted for 540 of the 550 seconds (98%) of the computation time of the initial hashing routine at *tableSize* $1.7 \cdot 10^9$.

Figure 4-7 shows how many buckets were hit during the initial hashing routine. Conventional wisdom directs that a hash table should not allowed to become more than 0.1 full, which was violated in some of the higher points of the figure. The figure has one more data point for $(11, \square)$ than Figure 4-6 because the initial hashing routine completed for that point, but we ran out of patience to allow the exhaustive enumeration of all words to complete.

It should be disclosed that Figure 4-6 were done on an early version of the code which in fact gave incorrect results. Nevertheless, it is believed that the running times and the BUCKETS table fullness results reported are approximately correct. With many of the data points taking over 10 hours, it was undesirable to repeat the

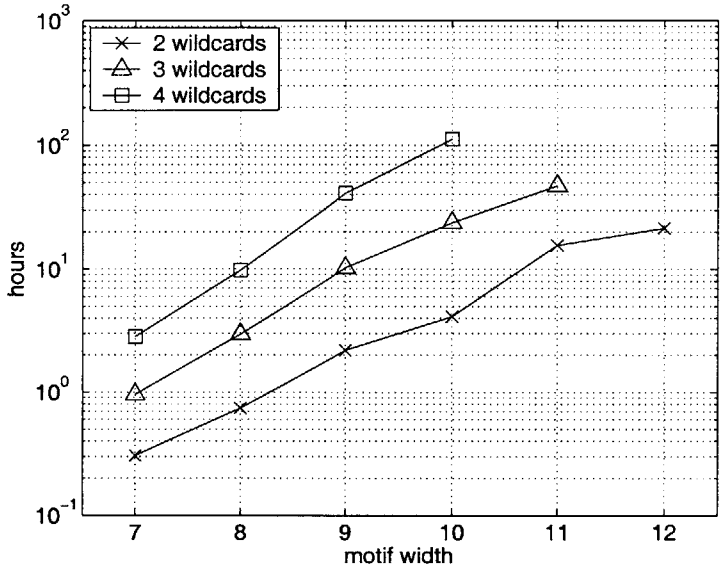


Figure 4-6: Running time in hours for motif discovery on the STE12_YPD data set for wider motifs and more allowed wildcards.

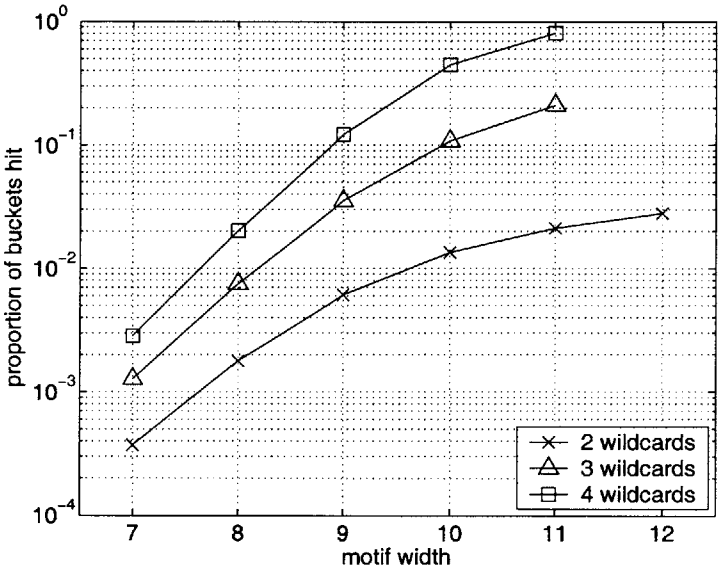


Figure 4-7: Illustration of how full (as a percentage of the $1.7 \cdot 10^9$ entries) the BUCKETS table became for various motif sizes.

experiment.

4.3 New motifs

Tables 4.4 and 4.5 give the top-scoring motifs for some transcription factors not listed in Table 4.1. These are candidates for further investigation. The score used to decide "top-scoring" was the sum-of-products p-value of §3.4. The positive sequences used for the table were the bound sequences at p-value 0.001.

Because computing the sum-of-products p-value is so computationally expensive, the motifs were first discovered with the binomial p-value. For each experiment, the top 10 motifs with binomial significance greater than $10^{-9.5}$ were re-evaluated with the sum-of-products p-value. The computation of a single sum-of-products p-value took about 15 seconds.

From discussion with a colleague, we note that the motifs for CIN5, GAT3, GLN3, IME4, YAP5, and YAP6 are probably not correct, while those for BAS1, FKH1, FKH2, INO4, and SUM1 are consistent with what is known about the transcription factors[7].

4.3.1 Results on shuffled data

To judge the background level of motifs, the algorithm was also run on random sets of intergenic sequences. Ideally, these runs should produce no significant motifs. Twenty-five random trials were run for each of 20, 40, 80, 120, and 160 randomly chosen sequences (for a total of 125 trials). Five of the 125 experiments discovered a total of 11 motifs with binomial significances greater than 10^{-4}, with most significant motif having significance $10^{-4.7}$. These falsely significant were more likely to be found when there were fewer positive sequences, as 8 of the 11 motifs were found in data sets with 20 positive sequences.

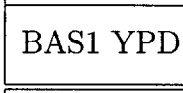
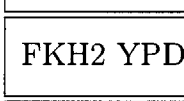
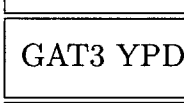
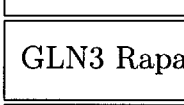
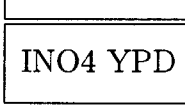
TF + Condition	Motif	*Sum-prod*	Binomial	Hypergeometric
BAS1 YPD		-11.24	-10.99	-14.71
CIN5 YPD		-11.36	-10.86	-19.67
DIG1 14hr Butanol		-13.65	-13.16	-22.30
FHL1 Rapamycin		-28.84	-27.28	-39.88
FHL1 YPD		-37.41	-35.12	-50.61
FKH1 YPD		-11.06	-10.85	-14.72
FKH2 YPD		-12.54	-12.16	-18.49
GAT3 YPD		-16.49	-15.90	-21.14
GLN3 Rapamycin		-11.73	-11.46	-16.65
IME4 YPD		-12.42	-12.16	-15.22
INO4 YPD		-12.28	-12.14	-14.36

Table 4.4: Top scoring motifs discovered for transcription factors not on Table 4.1 with sum-of-products significance greater than 10^{-10}. The significance values are $\log_{10}$ of the p-value.

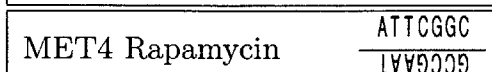
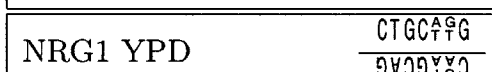
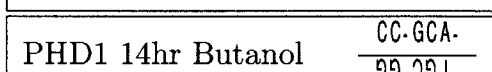
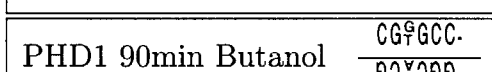
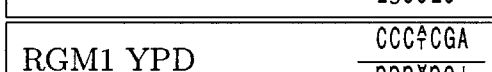
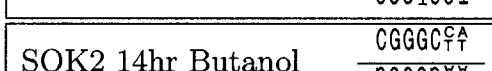
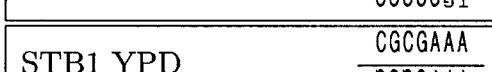
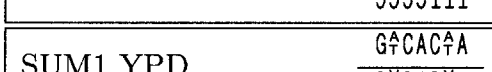
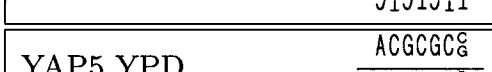
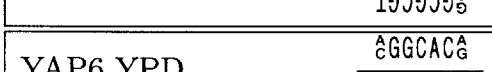
TF + Condition	Motif	*Sum-prod*	Binomial	Hypergeometric
MAC1 Cu2		-11.20	-11.00	-14.75
MBP1 YPD		-21.14	-20.14	-25.40
MET4 Rapamycin		-10.45	-10.25	-13.13
MET4 YPD		-10.90	-10.78	-13.08
NRG1 YPD		-12.04	-11.65	-19.00
PHD1 14hr Butanol		-17.90	-17.00	-33.46
PHD1 90min Butanol		-13.94	-13.33	-20.92
PHD1 YPD		-11.27	-10.86	-20.01
RGM1 YPD		-13.11	-12.91	-15.94
SOK2 14hr Butanol		-13.93	-13.28	-19.06
STB1 YPD		-11.00	-10.91	-12.36
SUM1 YPD		-11.60	-11.38	-17.18
YAP5 YPD		-12.49	-11.94	-16.98
YAP6 YPD		-11.92	-11.44	-18.78

Table 4.5: Top scoring motifs (continued from Table 4.4) discovered for transcription factors not on Table 4.1 with sum-of-products significance greater than 10^{-10}. The significance values are $\log_{10}$ of the p-value.

4.4 Future work

We conclude this chapter with ideas for future work.

4.4.1 Improving the running time

The algorithm can be made faster. Because so much of the run time is in the *find-occurrences-of-word* routine (§2.8.1), we will concentrate our attention on and around that routine.

First, we can lower the number of times the routine is called by causing fewer bits to be marked in TARGETS-SET. One simple way to do this is to use a threshold higher than 2 for the number of times a bucket must be hit in order to be included in TARGETS-SET. Figure 4-8 and Table 4.6 give the values of the most significant word for given number of occurrences in a set of positive sequences. For example, if we know a word occurs 40 times in the positive sequences, the greatest significance it *could* have is -29.6. Because we typically demand a significance of at least 10^{-9}, a threshold of 8 or 9 instead of 2 would have caused fewer bits to be marked in TARGETS-SET. Unfortunately Figure 4-8 can only be calculated after having set the threshold too low. However, it may be possible to reuse the values for future studies on the yeast genome.

Second, we can also be more intelligent about when a word "hits" during initial hashing (§2.5). For example, not only must it be present in P, but we may require that the word also be significant with respect to a Markov background model of intergenic sequences.

Finally, we can try to improve the running time of *find-occurrences-of-word* itself. This thesis's implementation exhaustively enumerated all instantiations of a word. A different data structure (such as a trie or suffix-tree[5, 12]) for storing the length-w subsequences may be able to compute *find-occurrences-of-word* faster.

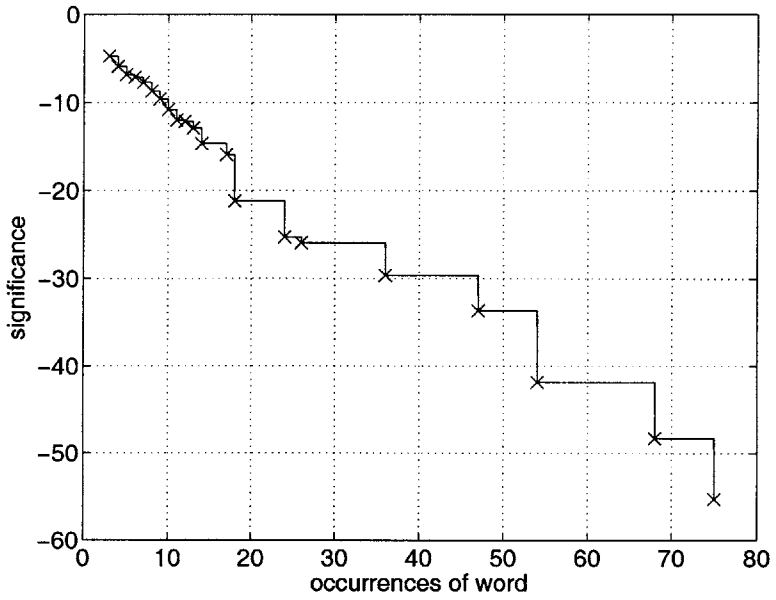


Figure 4-8: The lower envelope of the most significant word for given number of occurrences in a positive set. The values of the $\times$ points are given in Table 4.6.

Occurrences	Most significant word
3	-4.77
4	-5.94
5	-6.88
6	-7.14
7	-7.75
8	-8.73
9	-9.62
10	-10.83
11	-12.01
12	-12.15
13	-12.92
14	-14.68
17	-15.91
18	-21.17
24	-25.27
26	-25.93
36	-29.64
47	-33.68
54	-41.86
68	-48.30
75	-55.27

Table 4.6: The lower envelope points of Figure 4-8.

4.4.2 Further processing discovered motifs

The significant motifs discovered by this algorithm can be used as input to further refinement and post-processing.

One can use the discovered motif as the seed (starting point) for expectation maximization (as done by [3], [1], [8]). Expectation maximization returns a position-specific scoring matrix (PSSM), a finer grained representation (than a word) of the probabilities of nucleotides at each position of a motif. Expectation maximization can also widen a motif, if motif discovered by word-counting is initially padded on either side by the PSSM equivalent of the gap element.

The other type of post-processing eliminates the many near-identical discovered motifs. The word-counting algorithm frequently reports many similar motifs which are all similar through wildcard substitution or alignment. For example, the top 20 most significant motifs found for FHL1_YPD (using binding data with significance p=0.001) are shown in Table 4.7. Sinha[4], and Barash[3] discuss techniques of reducing the number of similar motifs.

4.4.3 Improvements to the statistical test

The statistical test developed in Chapter 3 can make use of more information for a better measure of significance.

In §3.2 we defined the null hypothesis behavior "random selection" to be selection with probability proportional to length. A straightforward modification could instead use the number of different subsequences of a sequences as its probability (appropriately normalized). As an extreme example, consider a very long sequence consisting of a repeat of a single nucleotide. While long, it offers few possibilities of where a transcription factor might bind. Such a long repetitive sequence ought to be selected with low probability.

Continuing in this manner, other biological prior knowledge can be incorporated into the prior probability that a sequence is selected. Such knowledge might (hypothetically) involve the location of the sequence on the chromosome, knowledge about

			C	C	[AG]	T	A	C	A
		[CT]	C	C	[AG]	T	A	C	
			C	C	[AG]	[GT]	A	C	A
			C	C	[AG]	T	[AG]	C	A
			C	C	[AG]	T	A	C	[AC]
			C	C	G	T	A	C	A
			C	C	[AG]	T	A	C	[AG]
			C	C	[AG]	[CT]	A	C	A
			C	C	[AG]	[AT]	A	C	A
			C	C	G	T	[AG]	C	A
			C	C	[AG]	T	A	[CG]	A
			C	[CG]	[AG]	T	A	C	A
			C	C	[AG]	T	[AC]	C	A
			C	C	[AG]	•	A	C	A
			C	C	•	T	A	C	A
C	A	[CT]	C	C	[AG]	T			
			C	C	G	T	A	C	[AG]
			C	[CG]	G	T	A	C	A
			C	C	[AG]	T	A	C	[AT]
			C	C	G	T	[AC]	C	A

Table 4.7: Top twenty significant motifs found for FHL1_YPD

the gene which the sequence precedes, or other genetic markers.

Biologically, we must question the assumption of independence (modulo choosing without replacement) between the $n = |P|$ random selections from A. For example, it would be reasonable to hypothesize that if two sequences are very similar, they would likely both be selected, or neither.

Not only can we incorporate biologically relevant information into the prior probability of the binding, but we can also try to incorporate more information about the binding event itself. Currently, the algorithm only makes use of the binary presence ("yes" or "no") of words in sequences. It could, for example, incorporate the following features:

- Number of occurrences of the word in the sequence
- Position of the occurrence(s) with respect to the start of transcription or other genetic markers in the sequence

- Strand of the occurrence of the word
- p-value of the binding event.

Because we use the identity of sequences, rather than just counts, we could try to calculate the probability of a specific set BM of bound-and-matching sequences. However, it becomes less clear how to define the p-value, although a plausible definition might sum over all sets of size n containing BM as a subset.

4.4.4 Toward larger challenges

As noted in §2.4, the 7-mer (with two wildcard) motif discovery problem on yeast was too small for the hybrid algorithm presented to be signficantly more effective than other naive approaches. Fortunately, however, larger challenges await.

For yeast, we can attempt to discover wider motifs, gapped and other poly-syllabic motif structures, and composite motifs of caused by the binding of multiple transcription factors. The composite motif search might be done by introducing an additional wildcard element which which can take the place of one or more consecutive nucleotides in sequence (somewhat like the behavior of rubber lengths in the text-formatting program LaTeX).

Beyond yeast, of course, are the many organisms whose genomes have been recently sequenced, including human. It will be only a matter of time before ChIP experiments are performed on those organisms. We expect that to do worthwhile motif discovery on larger and more complicated genomes, careful attention will have to be paid to the statistical issues and improvements mentioned above.

Appendix A

Gene Modules under YPD

This appendix documents the intergenic sequences used for motif discovery of gene modules from the GRAM algorithm[2]. Transcription factors are given in **bold**, genes are given *italics*, and a gene's upstream intergenic sequence is given in sans serif in parentheses following each gene. An intergenic sequence may occur more than once for a transcription factor (in the case of head-to-head genes). For such cases, only one copy of the sequence is given as input to motif discovery.

The list of genes for each transcription factor was provided by Ziv Bar-Joseph. The mapping from a gene to upstream intergenic sequence was originally created by Josh Barnett in the file `gene-probe-mapping.txt`. The intergenic sequences used in the ChIP experiment[10] were provided by D. Benjamin Gordon.

The mapping from gene to upstream intergenic sequence is somewhat messy, probably due to disagreements of what genes are real. Some genes may have more than one upstream intergenic sequence, because the ChIP experiment occasionally cut a long intergenic sequence into several pieces.

ABF1: *YBL039C* (iYBL039C); *YDL051W* (iYDL052C); *YDL081C* (iYDL081C); *YDR172W* (iYDR171W); *YDR280W* (iYDR279W); *YDR361C* (iYDR361C); *YDR365C* (iYDR365C); *YDR398W* (iYDR397C); *YER165W* (iYER164W); *YGL169W* (itK(CUU)G2); *YGR128C* (iYGR128C); *YGR187C* (iYGR187C); *YHR020W* (iYHR019C); *YHR064C* (iYHR064C); *YJL069C* (iYJL069C); *YJR063W* (iYJR062C); *YKL014C* (iYKL014C); *YKL078W* (iYKL079W); *YKL099C* (iYKL099C); *YKL143W* (iYKL144C); *YKL144C* (iYKL144C); *YKL172W* (iYKL173W); *YKL191W* (iYKL192C); *YKR056W* (iYKR055W); *YKR057W* (iYKR056W); *YKR059W* (iYKR058W); *YKR081C* (iYKR081C); *YLL011W* (iYLL012W); *YLL034C* (iYLL034C); *YLL044W* (iYLL046C); *YLR003C* (iYLR003C); *YLR029C* (iYLR029C); *YLR180W* (iYLR179C); *YLR222C* (iYLR222C); *YLR264W* (iYLR263W); *YLR401C* (iYLR401C); *YMR079W* (iYMR078C); *YMR093W* (iYMR092C); *YMR217W* (iYMR216C); *YMR260C* (iYMR260C); *YNL113W* (iYNL115C); *YNL119W* (iYNL121C); *YNL186W* (iYNL187W); *YNL209W* (iYNL210W); *YNL255C* (iYNL255C); *YNL313C* (iYNL313C); *YNR038W* (iYNR037C); *YNR046W* (iYNR045W); *YNR053C* (iYNR053C); *YOL077C* (iYOL077C); *YOR056C* (iYOR056C); *YOR078W* (iYOR077W); *YOR145C* (iYOR146W); *YOR167C*

(iYOR167C); *YOR168W* (iYOR167C); *YOR206W* (iYOR205C); *YOR224C* (iYOR224C); *YOR310C* (iYOR310C); *YPL012W* (iYPL013C); *YPL146C* (iYPL146C); *YPL160W* (iYPL161C); *YPR137W* (iYPR135W); *YPR187W* (iYPR186C); *YPR190C* (iYPR190C); *YGR183C* (iYGR183C); *YHR051W* (iYHR050W); *YKL016C* (iYKL016C); *YLR295C* (iYLR295C); *YPR191W* (iYPR190C); *YBR029C* (iYBR029C); *YBR030W* (iYBR029C); *YDR312W* (iYDR311W); *YDR385W* (iYDR384C); *YJR105W* (iYJR104C); *YKL182W* (iYKL183W)

ACE2: *YER126C* (iYER126C); *YER127W* (iYER126C); *YGR283C* (iYGR283C); *YHL013C* (iYHL013C); *YJR145C* (iYJR145C); *YFL022C* (iYFL022C); *YGL008C* (iYGL008C); *YHR193C* (iYHR193C); *YJL080C* (iYJL080C); *YKR042W* (iYKR041W); *YNR018W* (iYNR017W); *YOR315W* (iYOR314W); *YER079W* (iYER078C); *YFR017C* (iYFR017C); *YGL227W* (iYGL228W); *YHR194W* (iYHR193C); *YML100W* (iYML101C); *YBR158W* (iYBR157C-1, iYBR157C-0); *YGL028C* (iYGL028C); *YJL078C* (iYJL078C); *YLR286C* (iYLR286C); *YNL327W* (iYNL328C); *YMR262W* (iYMR261C); *YNL241C* (iYNL241C); *YOR138C* (iYOR138C); *YJR044C* (iYJR044C); *YKL150W* (iYKL151C); *YKL151C* (iYKL151C)

ARG80: *YER069W* (iYER068W); *YJL088W* (iYJL089W); *YOL058W* (iYOL059W); *YOR302W* (iYOR301W); *YOR303W* (iYOR301W)

ARG81: *YER069W* (iYER068W); *YJL088W* (iYJL089W); *YOL058W* (iYOL059W); *YOR302W* (iYOR301W); *YOR303W* (iYOR301W)

ARO80: *YDL091C* (iYDL091C); *YGR053C* (iYGR053C); *YHR138C* (iYHR138C); *YHR139C* (iYHR139C); *YHR140W* (iYHR139C-A); *YNL117W* (iYNL118C); *YNL125C* (iYNL125C); *YOR005C* (iYOR005C)

AZF1: *YDL037C* (iYDL037C-1, iYDL037C-0); *YDR144C* (iYDR144C); *YER007C-A* (iYER007C-A); *YJL148W* (iSNR190); *YKR024C* (iYKR024C); *YKR025W* (iYKR024C); *YLR175W* (iYLR174W); *YLR367W* (iYLR366W); *YLR413W* (iYLR412W-1, iYLR412W-0); *YNR053C* (iYNR053C); *YOL121C* (iYOL121C); *YOR210W* (iYOR209C)

BAS1: *YCL030C* (iYCL030C); *YGL234W* (iYGL235W); *YGR061C* (iYGR061C); *YKR080W* (iYKR079C); *YLR058C* (iYLR058C); *YLR359W* (iYLR358C); *YMR120C* (iYMR120C); *YMR300C* (iYMR300C); *YNL220W* (iYNL221C); *YOR128C* (iYOR128C)

CAD1: *YKL103C* (iYKL103C); *YLL060C* (iYLL060C); *YML116W* (iYML117W-A); *YNL134C* (iYNL134C); *YOL119C* (iYOL119C)

CBF1: *YHR089C* (iYHR089C); *YIL127C* (iYIL127C); *YKL191W* (iYKL192C); *YLL008W* (iYLL009C); *YNL282W* (iYNL283C); *YPL093W* (iYPL094C); *YFR030W* (iYFR029W); *YGR204W* (iYGR203W); *YIL074C* (iYIL074C); *YJR010W* (iYJR009C-1, iYJR009C-0); *YLR092W* (iYLR091W); *YNL277W* (iYNL278W)

CIN5: *YJR145C* (iYJR145C); *YLR075W* (iYLR074C); *YLR413W* (iYLR412W-1, iYLR412W-0); *YNL087W* (iYNL088W); *YNL178W* (iYNL179C-1, iYNL179C-0); *YOR315W* (iYOR314W)

CUP9: *YDR064W* (iYDR063W); *YEL040W* (iYEL041W); *YFR005C* (iYFR005C); *YGL077C* (iYGL077C); *YGR027C* (iYGR027C, YGR-Cdelta12); *YJL148W* (iSNR190); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YKL156W* (iYKL158W, iYKL157W); *YKR092C* (iYKR092C-0, iYKR092C-1); *YLR367W* (iYLR366W); *YMR014W* (iYMR013C-1); *YOR315W* (iYOR314W); *YPL198W* (iYPL199C)

DAL81: *YBR069C* (iYBR069C); *YDR046C* (iYDR046C); *YDR508C* (iYDR509W); *YEL046C* (iYEL045C); *YGR055W* (iYGR054W); *YOL020W* (iYOL021C)

DIG1: *YAR009C* (iYAR009C); *YCL019W* (iYCL022C); *YCL020W* (iYCL022C); *YER138C* (iYERCdelta20); *YER160C* (iYERCdelta24); *YCL027W* (iYCL028W); *YCL055W* (iYCL056C); *YCR089W* (iYCR088W); *YIL037C* (iYIL037C); *YIL082W* (iYIL083C, YILWTy3-1A); *YKL189W* (itL(UAA)K); *YMR198W* (iYMR197C); *YNL279W* (iYNL280C); *YOR343C* (iYORWdelta22, iYOR343C)

FHL1: *YDR385W* (iYDR384C); *YGR027C* (iYGR027C, YGRCdelta12); *YGR085C* (iYGR085C); *YHL033C* (iYHL033C); *YJR123W* (iYJR122W); *YKR057W* (iYKR056W); *YLR029C* (iYLR029C); *YLR075W* (iYLR074C); *YLR150W* (iYLR149C); *YLR264W* (iYLR263W); *YLR367W* (iYLR366W); *YOR369C* (iYOR369C); *YBL072C* (iYBL072C); *YBL092W* (iYBL093C); *YBR118W* (iYBR117C); *YBR189W* (iYBR188C); *YDL061C* (iYDL061C); *YDL136W* (iYDL137W); *YDL184C* (iYDL184C); *YDR447C* (iYDR447C); *YER102W* (iYER101C); *YER131W* (iYER130C); *YGL123W* (iYGL124C); *YGL147C* (iYGL147C); *YGR214W* (iYGR213C); *YHL001W* (iYHL002W); *YJR145C* (iYJR145C); *YKR094C* (iYKR094C); *YLL045C* (iYLL044W); *YLR325C* (iYLR325C); *YLR333C* (iYLR333C); *YLR340W* (iYLR338W); *YLR344W* (itR(CCG)L); *YLR388W* (iYLR387C); *YMR116C* (iYMR116C); *YMR229C* (iYMR229C); *YNL067W* (iYNL068C); *YNL178W* (iYNL179C-1, iYNL179C-0); *YNL302C* (iYNL302C); *YOR182C* (iYOR182C); *YPL079W* (iYPL080C); *YPR043W* (iYPR042C); *YDR500C* (iYDR500C); *YML063W* (iYML064C); *YML064C* (iYML064C); *YOR095C* (iYOR095C); *YOR096W* (iYOR095C); *YOR312C* (iYOR312C); *YBR084C-A* (iYBR084C-A); *YBR181C* (iYBR181C); *YBR191W* (iYBR189W); *YDL075W* (iYDL076C); *YDL082W* (iYDL083C); *YDL083C* (iYDL083C); *YDL191W* (iYDL192W); *YDR025W* (iYDR023W); *YDR418W* (iYDR416W); *YDR449C* (iYDR449C); *YDR450W* (iYDR449C); *YDR471W* (iYDR470C); *YEL054C* (iYEL054C); *YER074W* (iYER073W); *YER117W* (iYER116C); *YFR031C-A* (iYFR031C-A); *YGL030W* (iYGL031C); *YGL031C* (iYGL031C); *YGL189C* (iYGL189C); *YGR034W* (iYGR033C); *YGR148C* (iYGR148C); *YHR021C* (iYHR021C); *YHR141C* (iYHR141C); *YHR203C* (iYHR203C); *YIL018W* (iYIL019W); *YIL133C* (iYIL133C); *YIL148W* (iYIL149C); *YJL136C* (iYJL136C); *YJL177W* (iYJL178C); *YJL189W* (iYJL190C); *YJL190C* (iYJL190C); *YKL006W* (iYKL006C-A); *YKL180W* (iYKL181W); *YLR048W* (iYLR047C); *YLR441C* (iYLR441C); *YLR448W* (iYLR447C); *YML026C* (iYML026C); *YML073C* (iYML073C); *YMR242C* (iYMR242C); *YNL069C* (iYNL069C); *YNL096C* (iYNL096C); *YOL039W* (iYOL040C); *YOL040C* (iYOL040C); *YOL127W* (iYOL128C); *YOR234C* (iYOR234C); *YOR293W* (iYOR292C); *YPL090C* (iYPL090C); *YPL131W* (iYPL132W); *YPR102C* (iYPR102C); *YPR132W* (iYPR131C); *YDR064W* (iYDR063W); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YKL156W* (iYKL158W, iYKL157W); *YLR167W* (iYLR166C); *YOL121C* (iYOL121C); *YBL087C* (iYBL087C); *YER056C-A* (iYER056C-A); *YGL103W* (iYGL104C); *YLR061W* (iYLR060W); *YPL143W* (iYPL144W); *YPR080W* (iYPR079W); *YGL135W* (iYGL136C); *YGR118W* (iYGR117C); *YLR185W* (iYLR184W); *YNL162W* (iYNL163C); *YOL120C* (iYOL120C); *YPR131C* (iYPR131C)

FKH1: *YBL064C* (iYBL064C); *YBR139W* (iYBR138C); *YKL157W* (iYKL159C); *YLL006W* (iYLL007C); *YML100W* (iYML101C); *YMR261C* (iYMR261C); *YMR262W* (iYMR261C); *YBL081W* (iYBL082C); *YDR324C* (iYDR324C); *YDR500C* (iYDR500C); *YFL022C* (iYFL022C); *YIL158W* (iYIL159W); *YKR059W* (iYKR058W); *YMR215W* (iYMR214W); *YPR119W* (iYPRWsigma2-1, iYPRWsigma2-0)

FKH2: *YBL081W* (iYBL082C); *YDR324C* (iYDR324C); *YDR500C* (iYDR500C); *YFL022C* (iYFL022C); *YIL158W* (iYIL159W); *YKR059W* (iYKR058W); *YMR215W* (iYMR214W); *YPR119W* (iYPRWsigma2-1, iYPRWsigma2-0); *YBR078W* (iYBR077C); *YLR300W* (iYLR299W); *YOR247W* (iYOR246C); *YOR248W* (iYOR246C); *YBR139W* (iYBR138C); *YDR148C* (iYDR148C); *YER079W* (iYER078C); *YFR017C* (iYFR017C); *YPR149W* (iYPR148C-1, iYPR148C-0); *YDR261C* (iYDR261C-0, iYDR261C-1); *YIL123W* (iYIL124W); *YLR084C* (iYLR084C); *YOR315W* (iYOR314W); *YML063W* (iYML064C); *YML064C* (iYML064C); *YOR095C* (iYOR095C); *YOR096W* (iYOR095C); *YOR312C* (iYOR312C); *YDR146C* (iYDR146C); *YGL021W* (iYGL022W); *YJL051W* (iYJL052W); *YJR092W* (iYJR091C); *YLR190W* (iYLR189C); *YGL008C* (iYGL008C); *YJL158C* (iYJL158C)

FZF1: *YBR143C* (iYBR143C); *YBR154C* (iYBR154C); *YBR155W* (iYBR154C); *YBR189W* (iYBR188C); *YDL192W* (iYDL193W); *YDR012W* (iYDR011W); *YDR064W* (iYDR063W); *YDR245W* (iYDR244W); *YER049W* (iYER048W-A); *YHR089C* (iYHR089C); *YJL148W* (iSNR190); *YKL172W* (iYKL173W); *YLR083C* (iYLR083C); *YLR367W* (iYLR366W); *YMR131C* (iYMR131C); *YOR004W* (iYOR003W); *YPL217C* (iYPL217C); *YPL235W* (iYPL236C)

GAL4: *YBL027W* (iYBL028C); *YBL028C* (iYBL028C); *YJR123W* (iYJR122W); *YLR117C* (iYLR117C); *YML022W* (iYML023C); *YBL087C* (iYBL087C); *YBR048W* (iYBR047W); *YBR252W* (iYBR251W); *YDL143W* (iYDL144C); *YHR193C* (iYHR193C); *YIL018W* (iYIL019W); *YIL133C* (iYIL133C); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YLR061W* (iYLR060W); *YOR369C* (iYOR369C); *YPR132W* (iYPR131C); *YBR084C-A* (iYBR084C-A); *YBR181C* (iYBR181C); *YDL075W* (iYDL076C); *YDL082W* (iYDL083C); *YDL083C* (iYDL083C); *YDR418W* (iYDR416W); *YDR471W* (iYDR470C); *YEL054C* (iYEL054C); *YER056C-A* (iYER056C-A); *YER074W* (iYER073W); *YHR141C* (iYHR141C); *YIL148W* (iYIL149C); *YML026C* (iYML026C); *YOL039W* (iYOL040C); *YOL040C* (iYOL040C); *YOL120C* (iYOL120C); *YOR312C* (iYOR312C); *YPR080W* (iYPR079W); *YPR131C* (iYPR131C); *YBR018C* (iYBR018C); *YBR019C* (iYBR019C); *YBR020W* (iYBR019C); *YDR009W* (iYDR008C); *YLR081W* (iYLR080W)

GAT3: *YBR084C-A* (iYBR084C-A); *YBR181C* (iYBR181C); *YBR191W* (iYBR189W); *YDL075W* (iYDL076C); *YDL082W* (iYDL083C); *YDL083C* (iYDL083C); *YDL191W* (iYDL192W); *YDR025W* (iYDR023W); *YDR418W* (iYDR416W); *YDR449C* (iYDR449C); *YDR450W* (iYDR449C); *YDR471W* (iYDR470C); *YEL054C* (iYEL054C); *YER074W* (iYER073W); *YER117W* (iYER116C); *YFR031C-A* (iYFR031C-A); *YGL030W* (iYGL031C); *YGL031C* (iYGL031C); *YGL189C* (iYGL189C); *YGR034W* (iYGR033C); *YGR148C* (iYGR148C); *YHR021C* (iYHR021C); *YHR141C* (iYHR141C); *YHR203C* (iYHR203C); *YIL018W* (iYIL019W); *YIL133C* (iYIL133C); *YIL148W* (iYIL149C); *YJL136C* (iYJL136C); *YJL177W* (iYJL178C); *YJL189W* (iYJL190C); *YJL190C* (iYJL190C); *YKL006W* (iYKL006C-A); *YKL180W* (iYKL181W); *YLR048W* (iYLR047C); *YLR441C* (iYLR441C); *YLR448W* (iYLR447C); *YML026C* (iYML026C); *YML063W* (iYML064C); *YML073C* (iYML073C); *YMR242C* (iYMR242C); *YNL069C* (iYNL069C); *YNL096C* (iYNL096C); *YOL039W* (iYOL040C); *YOL040C* (iYOL040C); *YOL127W* (iYOL128C); *YOR234C* (iYOR234C); *YOR293W* (iYOR292C); *YOR312C* (iYOR312C); *YPL090C* (iYPL090C); *YPL131W* (iYPL132W); *YPR102C* (iYPR102C); *YPR132W* (iYPR131C); *YHR091C* (iYHR091C); *YEL076C* (iYEL075C); *YHR204W* (iYHR203C)

GCN4: *YBL076C* (iYBL076C); *YBR249C* (iYBR249C); *YDR341C* (iYDR341C); *YER052C* (iYER052C); *YER055C* (iYER055C); *YER090W* (iYER089C); *YHR019C* (iYHR019C); *YHR020W* (iYHR019C); *YJL200C* (iYJL200C); *YLR355C* (iYLR355C); *YMR260C* (iYMR260C); *YOR108W* (iYOR107W); *YPR110C* (iYPR110C); *YPR145W* (iSNR45); *YER069W* (iYER068W); *YJL088W* (iYJL089W); *YOL058W* (iYOL059W); *YOR302W* (iYOR301W); *YOR303W* (iYOR301W)

GCR1: *YAL038W* (iYAL039C-1, iYAL039C-0); *YBR196C* (iYBR196C); *YDR050C* (iYDR050C); *YER177W* (iYER176W); *YGR192C* (iYGR192C); *YKL152C* (iYKL152C); *YLR044C* (iYLR044C)

GCR2: *YDL192W* (iYDL193W); *YDR245W* (iYDR244W); *YER007C-A* (iYER007C-A); *YJL148W* (iSNR190); *YLR196W* (iYLR195C); *YLR264W* (iYLR263W); *YMR259C* (iYMR259C); *YOL121C* (iYOL121C); *YOR021C* (iYOR021C)

GRF10: *YDL124W* (iYDL125C); *YDL215C* (iYDL215C); *YDR223W* (iYDR222W); *YGR205W* (iYGR204W); *YIL077C* (iYIL077C); *YIL166C* (iYIL166C); *YJL161W* (itE(UUC)J); *YLR152C* (iYLR152C); *YMR170C* (iYMR170C); *YPR149W* (iYPR148C-1, iYPR148C-0)

HAP2: *YBR025C* (iYBR025C); *YDL063C* (iYDL062W); *YGR234W* (iYGR233C-0, iYGR233C-1); *YIL096C* (iYIL096C); *YJL148W* (iSNR190); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YPL207W* (iYPL208W); *YBR039W* (iYBR038W); *YDR377W* (iYDR376W); *YDR529C* (iYDR529C); *YEL024W* (iYEL025C); *YGL187C* (iYGL187C); *YGR183C* (iYGR183C); *YHR001W-A* (iYHR001W); *YHR051W* (iYHR050W); *YJR077C* (iYJR077C); *YML120C* (iYML120C); *YLL027W* (iYLL028W); *YLR220W* (iYLR219W)

HAP3: *YBR039W* (iYBR038W); *YDR377W* (iYDR376W); *YGL187C* (iYGL187C); *YHR001W-A* (iYHR001W); *YHR051W* (iYHR050W); *YJR077C* (iYJR077C); *YML120C* (iYML120C); *YLL027W* (iYLL028W); *YLR220W* (iYLR219W)

HAP4: *YBL030C* (iYBL030C-0); *YBL045C* (iYBL045C); *YBL099W* (itF(GAA)B); *YDL004W* (iYDL005C); *YDL181W* (iYDL182W); *YDR148C* (iYDR148C); *YDR298C* (iYDR298C); *YFR033C* (iYFR033C); *YGL191W* (iYGL192W); *YKL148C* (iYKL148C); *YLR038C* (iYLR038C); *YLR395C* (iYLR395C); *YMR256C* (iYMR256C); *YNL052W* (iYNL053W); *YOR065W* (iYOR064C); *YGR183C* (iYGR183C); *YHR051W* (iYHR050W); *YKL016C* (iYKL016C); *YLR295C* (iYLR295C); *YPR191W* (iYPR190C); *YBR039W* (iYBR038W); *YDR377W* (iYDR376W); *YDR529C* (iYDR529C); *YEL024W* (iYEL025C); *YGL187C* (iYGL187C); *YHR001W-A* (iYHR001W); *YJR077C* (iYJR077C); *YML120C* (iYML120C)

HAP5: *YBR039W* (iYBR038W); *YGL187C* (iYGL187C); *YHR051W* (iYHR050W); *YLL027W* (iYLL028W); *YLR220W* (iYLR219W)

HIR1: *YBR009C* (iYBR009C); *YBR010W* (iYBR009C); *YDR224C* (iYDR224C); *YDR225W* (iYDR224C); *YNL030W* (iYNL031C); *YNL031C* (iYNL031C); *YDR064W* (iYDR063W); *YFR001W* (iCEN6); *YGR187C* (iYGR187C); *YJR007W* (iYJR006W); *YKR026C* (iYKR026C, iYKRWdelta10); *YLR002C* (iYLR002C); *YLR061W* (iYLR060W); *YLR167W* (iYLR166C); *YOL144W* (iYOL145C); *YPL019C* (iYPL019C); *YPR137W* (iYPR135W)

HIR2: *YBR009C* (iYBR009C); *YBR010W* (iYBR009C); *YDR224C* (iYDR224C); *YDR225W* (iYDR224C); *YNL030W* (iYNL031C); *YNL031C* (iYNL031C)

HSF1: *YBR072W* (iYBR071W); *YDR171W* (iYDR170W-A-1, iYDR170W-A-0); *YDR214W* (iYDR213W); *YER037W* (iYER036C); *YER103W* (iYER102W); *YFL014W* (iYFL015C); *YFL016C* (iYFL016C); *YGL037C* (iYGL037C); *YGR142W* (iYGR141W); *YGR194C* (iYGR194C); *YLL023C* (iYLL023C); *YLL026W* (iYLL027W); *YLL039C* (iYLL039C); *YLR216C* (iYLR217W); *YLR327C* (iYLR327C); *YML100W* (iYML101C); *YNL006W* (iYNL007C); *YNL007C* (iYNL007C); *YOR027W* (iYOR026W); *YOR317W* (iYOR316C-0, iYOR316C-1); *YPR154W* (iYPR153W); *YPR158W* (iYPR157W)

IME4: *YDL223C* (iYDL223C); *YFR017C* (iYFR017C); *YGL227W* (iYGL228W); *YHR138C* (iYHR138C); *YJR044C* (iYJR044C)

INO2: *YBR029C* (iYBR029C); *YBR030W* (iYBR029C); *YCR034W* (iYCR033W, iYCR034W); *YEL040W* (iYEL041W); *YER043C* (iYER043C); *YGR155W* (iYGR154C); *YJR105W* (iYJR104C); *YKL182W* (iYKL183W); *YLR249W* (iYLR248W); *YNL175C* (iYNL174W); *YNR017W* (iYNR016C-1, iYNR016C-0); *YPL231W* (iYPL232W)

INO4: *YBR029C* (iYBR029C); *YBR030W* (iYBR029C); *YDR312W* (iYDR311W); *YDR385W* (iYDR384C); *YJR105W* (iYJR104C); *YKL182W* (iYKL183W); *YCR034W* (iYCR033W, iYCR034W); *YEL040W* (iYEL041W); *YER043C* (iYER043C); *YGR155W* (iYGR154C); *YLR249W* (iYLR248W); *YNL175C* (iYNL174W); *YNR017W* (iYNR016C-1, iYNR016C-0); *YPL231W* (iYPL232W)

IXR1: *YBL087C* (iYBL087C); *YBR061C* (iYBR061C); *YDR086C* (iYDR086C); *YDR101C* (iYDR101C); *YDR361C* (iYDR361C); *YKR026C* (iYKR026C, iYKRWdelta10); *YLL034C* (iYLL034C); *YLR002C* (iYLR002C); *YLR167W* (iYLR166C); *YMR260C* (iYMR260C); *YNR054C* (iYNR054C-

0, iYNR054C-1); *YOR252W* (iYOR251C); *YOR253W* (iYOR252W); *YOR315W* (iYOR314W)

LEU3: *YBR048W* (iYBR047W); *YGL009C* (iYGL009C); *YHR208W* (iYHR207C); *YKL120W* (iYKL121W); *YMR108W* (iYMR107W); *YOR108W* (iYOR107W); *YOR271C* (iYOR271C)

MAC1: *YCL031C* (iYCL031C); *YDL184C* (iYDL184C); *YDR324C* (iYDR324C); *YER007C-A* (iYER007C-A); *YGL135W* (iYGL136C); *YHR193C* (iYHR193C); *YJL148W* (iSNR190); *YKL080W* (iYKL081W); *YNL016W* (itI(AAU)N2); *YNL255C* (iYNL255C); *YOL022C* (iYOL022C); *YOL080C* (iYOL079W); *YOR271C* (iYOR271C); *YPL183C* (iYPL183C); *YPR110C* (iYPR110C)

MBP1: *YBR191W* (iYBR189W); *YDR324C* (iYDR324C); *YDR500C* (iYDR500C); *YGL147C* (iYGL147C); *YHL013C* (iYHL013C); *YLR397C* (iYLR397C); *YNL313C* (iYNL313C); *YPL050C* (iYPL050C); *YPL081W* (iYPL082C); *YPL126W* (iYPL127C); *YBR078W* (iYBR077C); *YLR300W* (iYLR299W); *YMR215W* (iYMR214W); *YOR247W* (iYOR246C); *YOR248W* (iYOR246C); *YJL080C* (iYJL080C); *YLR367W* (iYLR366W); *YDR309C* (iYDR309C); *YGR189C* (iYGR189C); *YHR193C* (iYHR193C); *YJL148W* (iSNR190); *YLR342W* (iYLR341W); *YMR305C* (iYMR305C); *YDL003W* (iYDL004W); *YDR097C* (iYDR097C); *YDR507C* (iYDR507C); *YIL026C* (iYIL026C); *YJL074C* (iYJL074C); *YJR030C* (iYJR030C); *YKL113C* (iYKL113C); *YLR103C* (iYLR103C); *YMR076C* (iYMR076C); *YNL273W* (iYNL274C); *YOR074C* (iYOR074C); *YER070W* (iYER069W); *YJL187C* (iYJL187C); *YML027W* (iYML028W); *YMR144W* (iYMR143W); *YMR179W* (iYMR178W); *YMR199W* (iYMR198W); *YPL267W* (iYPL268W); *YPR120C* (iYPR120C); *YER079W* (iYER078C); *YFR017C* (iYFR017C); *YJR044C* (iYJR044C); *YKL150W* (iYKL151C); *YKL151C* (iYKL151C); *YML100W* (iYML101C); *YNL241C* (iYNL241C)

MCM1: *YDR461W* (iYDR460W); *YFL026W* (iYFL027C); *YGL032C* (iYGL032C); *YKL209C* (iYKL209C); *YNL145W* (iYNL146W); *YEL040W* (iYEL041W); *YIL123W* (iYIL124W); *YLR084C* (iYLR084C); *YOR315W* (iYOR314W); *YPR119W* (iYPRWsigma2-1, iYPRWsigma2-0); *YDR146C* (iYDR146C); *YGL021W* (iYGL022W); *YJL051W* (iYJL052W); *YJR092W* (iYJR091C); *YLR190W* (iYLR189C)

MET31: *YDL059C* (iYDL059C); *YEL072W* (iYEL073C-4, iYEL073C-1, iYEL073C-0, iYEL073C-3, iYEL073C-2); *YFR030W* (iYFR029W); *YGL184C* (iYGL184C); *YNL277W* (iYNL278W); *YPR167C* (iYPR167C); *YDR064W* (iYDR063W); *YDR500C* (iYDR500C); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YKL156W* (iYKL158W, iYKL157W); *YLR167W* (iYLR166C); *YOL121C* (iYOL121C)

MET4: *YAL012W* (iYAL013W); *YBR078W* (iYBR077C); *YDL060W* (iYDL061C); *YDL061C* (iYDL061C); *YGR155W* (iYGR154C); *YKR092C* (iYKR092C-0, iYKR092C-1); *YLR179C* (iYLR179C); *YLR180W* (iYLR179C); *YLR355C* (iYLR355C); *YLR432W* (iYLR431C); *YNL016W* (itI(AAU)N2); *YOL039W* (iYOL040C); *YOL040C* (iYOL040C); *YPL143W* (iYPL144W); *YPL198W* (iYPL199C); *YPR102C* (iYPR102C); *YFR030W* (iYFR029W); *YGR204W* (iYGR203W); *YIL074C* (iYIL074C); *YJR010W* (iYJR009C-1, iYJR009C-0); *YLR092W* (iYLR091W); *YNL277W* (iYNL278W)

MSN4: *YBR072W* (iYBR071W); *YER103W* (iYER102W); *YGR142W* (iYGR141W); *YHR087W* (iYHR086W); *YLL026W* (iYLL027W); *YNL077W* (iYNL078W); *YOR292C* (iYOR292C); *YPR154W* (iYPR153W)

MSS11: *YBL087C* (iYBL087C); *YBR121C* (iYBR121C); *YBR189W* (iYBR188C); *YDL136W* (iYDL137W); *YDL184C* (iYDL184C); *YDR119W* (iYDR118W); *YDR365C* (iYDR365C); *YEL040W* (iYEL041W); *YER009W* (iYER008C); *YFR005C* (iYFR005C); *YGL070C* (iYGL070C); *YGR264C* (iYGR265W); *YHR216W* (iYHRCdelta16-0); *YJL014W* (iYJL017W); *YJR041C* (iYJR041C); *YLR065C* (iYLR065C); *YLR243W* (iYLR242C); *YLR291C* (iYLR291C); *YOR078W* (iYOR077W); *YOR276W* (iYOR275C); *YPL198W* (iYPL199C); *YPR033C* (iYPR033C); *YPR069C* (iYPR069C)

MTH1: *YBL054W* (iYBL055C); *YDL038C* (iYDL038C); *YDL039C* (iYDL038C); *YDR299W* (iYDR298C); *YER056C-A* (iYER056C-A); *YIL019W* (iYIL020C); *YIL064W* (iYIL065C); *YLR406C* (iYLR406C); *YOR101W* (iYOR100C); *YOR271C* (iYOR271C)

NDD1: *YGL116W* (iYGL117W); *YGR229C* (iYGR229C); *YML052W* (iYML053C); *YMR001C* (iYMR001C); *YOR025W* (iYOR023C); *YFL022C* (iYFL022C); *YGL008C* (iYGL008C); *YHR193C* (iYHR193C); *YJL080C* (iYJL080C); *YKR042W* (iYKR041W); *YNR018W* (iYNR017W); *YOR315W* (iYOR314W); *YBR139W* (iYBR138C); *YDR148C* (iYDR148C); *YER079W* (iYER078C); *YFR017C* (iYFR017C); *YPR149W* (iYPR148C-1, iYPR148C-0); *YBR078W* (iYBR077C); *YLR300W* (iYLR299W); *YLR367W* (iYLR366W); *YOR247W* (iYOR246C); *YOR248W* (iYOR246C); *YJL100W* (iYJL101C); *YML100W* (iYML101C); *YMR135C* (iYMR135W-A); *YMR136W* (iYMR135W-A); *YPR151C* (iYPR151C); *YHR194W* (iYHR193C); *YDR146C* (iYDR146C); *YGL021W* (iYGL022W); *YJL051W* (iYJL052W); *YJR092W* (iYJR091C); *YLR190W* (iYLR189C); *YPR119W* (iYPRWsigma2-1, iYPRWsigma2-0); *YJL158C* (iYJL158C); *YJR044C* (iYJR044C); *YKL150W* (iYKL151C); *YKL151C* (iYKL151C); *YNL241C* (iYNL241C)

NRG1: *YEL040W* (iYEL041W); *YER049W* (iYER048W-A); *YIL020C* (iYIL020C); *YJL145W* (iYJL146W); *YLL034C* (iYLL034C); *YLR413W* (iYLR412W-1, iYLR412W-0); *YOR315W* (iYOR314W); *YPR009W* (iYPR008W); *YDR043C* (iYDR043C-1, iYDR043C-0, iYDR043C-2); *YFR017C* (iYFR017C); *YGL227W* (iYGL228W); *YHR138C* (iYHR138C); *YIL097W* (iYIL098C); *YIL099W* (iYIL100W); *YMR017W* (iYMR016C-1, iYMR016C-0); *YMR135C* (iYMR135W-A); *YOR273C* (iYOR273C); *YPL230W* (iYPL231W); *YDL245C* (iYDL245C-0, iYDL245C-1); *YJL221C* (iYJL220W); *YJR158W* (iYJR157W-1, iYJR157W-0); *YEL069C* (iYEL069C-1, iYEL069C-0); *YEL070W* (iYEL071W-1, iYEL071W-0)

PDR1: *YDL148C* (iYDL148C); *YHR193C* (iYHR193C); *YHR197W* (iYHR196W); *YJL148W* (iSNR190); *YMR241W* (iYMR240C); *YOL080C* (iYOL079W); *YOR369C* (iYOR369C); *YPL050C* (iYPL050C); *YBL087C* (iYBL087C); *YBR191W* (iYBR189W); *YDL075W* (iYDL076C); *YDL082W* (iYDL083C); *YDL083C* (iYDL083C); *YDR064W* (iYDR063W); *YDR418W* (iYDR416W); *YDR471W* (iYDR470C); *YEL054C* (iYEL054C); *YER056C-A* (iYER056C-A); *YER074W* (iYER073W); *YER117W* (iYER116C); *YGL103W* (iYGL104C); *YGL189C* (iYGL189C); *YHR021C* (iYHR021C); *YHR141C* (iYHR141C); *YIL018W* (iYIL019W); *YIL133C* (iYIL133C); *YIL148W* (iYIL149C); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YLR061W* (iYLR060W); *YNL069C* (iYNL069C); *YOL121C* (iYOL121C); *YPL143W* (iYPL144W); *YPR080W* (iYPR079W); *YPR102C* (iYPR102C); *YHR091C* (iYHR091C)

PHD1: *YAL002W* (iYAL003W); *YER033C* (iYER033C); *YHL024W* (iYHL025W); *YJL219W* (iYJL220W); *YJR127C* (iYJR128W); *YLR369W* (iYLR368W); *YBR162C* (iYBR162C); *YDR261C* (iYDR261C-0, iYDR261C-1); *YDR309C* (iYDR309C); *YEL040W* (iYEL041W); *YJL158C* (iYJL158C); *YNL178W* (iYNL179C-1, iYNL179C-0); *YDL245C* (iYDL245C-0, iYDL245C-1); *YIL099W* (iYIL100W); *YJL221C* (iYJL220W); *YJR158W* (iYJR157W-1, iYJR157W-0); *YMR017W* (iYMR016C-1, iYMR016C-0); *YMR135C* (iYMR135W-A); *YEL069C* (iYEL069C-1, iYEL069C-0); *YEL070W* (iYEL071W-1, iYEL071W-0)

PHO4: *YCL035C* (iYCL035C); *YDR474C* (iYDR474C); *YER033C* (iYER033C); *YER079W* (iYER078C); *YFR045W* (iYFR044C); *YIL136W* (iYIL137C); *YKL142W* (iYKL143W); *YLR267W* (iYLR266C); *YPR026W* (iYPR025C)

PUT3: *YGL103W* (iYGL104C); *YIL133C* (iYIL133C); *YLL044W* (iYLL046C); *YNL016W* (itI(AAU)N2); *YOR272W* (iSNR8); *YOR306C* (iYOR306C)

RAP1: *YBL048W* (iYBL050W); *YBL049W* (iYBL050W); *YBR117C* (iYBR117C); *YER035W* (iYER034W); *YER101C* (iYER101C); *YFL014W* (iYFL015C); *YGL104C* (iYGL104C); *YGR149W* (iYGR148C); *YHR096C* (iYHR096C); *YIR016W* (iYIR015W); *YJR073C* (iYJR073C); *YKL150W* (iYKL151C); *YKL151C* (iYKL151C); *YLL039C* (iYLL039C); *YLR174W* (iYLR173W); *YLR311C* (iYLR312C); *YLR312C* (iYLR312C); *YMR206W* (iYMR205C); *YOL082W* (iYOL083W-1, iYOL083W-0); *YOR100C* (iYOR100C); *YOR161C* (iYOR161C); *YOR374W* (iYOR373W); *YPL054W* (iYPL055C); *YPR030W* (iYPR029C); *YBL072C* (iYBL072C); *YBL092W* (iYBL093C); *YBR118W* (iYBR117C); *YBR189W* (iYBR188C); *YDL061C* (iYDL061C); *YDL136W* (iYDL137W); *YDL184C* (iYDL184C); *YDR447C* (iYDR447C); *YER102W* (iYER101C); *YER131W* (iYER130C); *YGL123W* (iYGL124C); *YGL147C* (iYGL147C); *YGR214W* (iYGR213C); *YHL001W* (iYHL002W); *YJR145C* (iYJR145C); *YKR094C* (iYKR094C); *YLL045C* (iYLL044W); *YLR325C* (iYLR325C); *YLR333C* (iYLR333C); *YLR340W* (iYLR338W); *YLR344W* (itR(CCG)L); *YLR388W* (iYLR387C); *YMR116C* (iYMR116C); *YMR229C* (iYMR229C); *YNL067W* (iYNL068C); *YNL178W* (iYNL179C-1, iYNL179C-0); *YNL302C* (iYNL302C); *YOR182C* (iYOR182C); *YPL079W* (iYPL080C); *YPR043W* (iYPR042C); *YDR500C* (iYDR500C); *YML063W* (iYML064C); *YML064C* (iYML064C); *YOR095C* (iYOR095C); *YOR096W* (iYOR095C); *YOR312C* (iYOR312C); *YBR084C-A* (iYBR084C-A); *YBR181C* (iYBR181C); *YBR191W* (iYBR189W); *YDL075W* (iYDL076C); *YDL082W* (iYDL083C); *YDL083C* (iYDL083C); *YDL191W* (iYDL192W); *YDR025W* (iYDR023W); *YDR418W* (iYDR416W); *YDR449C* (iYDR449C); *YDR450W* (iYDR449C); *YDR471W* (iYDR470C); *YEL054C* (iYEL054C); *YER074W* (iYER073W); *YER117W* (iYER116C); *YFR031C-A* (iYFR031C-A); *YGL030W* (iYGL031C); *YGL031C* (iYGL031C); *YGL189C* (iYGL189C); *YGR034W* (iYGR033C); *YGR148C* (iYGR148C); *YHR021C* (iYHR021C); *YHR141C* (iYHR141C); *YHR203C* (iYHR203C); *YIL018W* (iYIL019W); *YIL133C* (iYIL133C); *YIL148W* (iYIL149C); *YJL136C* (iYJL136C); *YJL177W* (iYJL178C); *YJL189W* (iYJL190C); *YJL190C* (iYJL190C); *YKL006W* (iYKL006C-A); *YKL180W* (iYKL181W); *YLR048W* (iYLR047C); *YLR441C* (iYLR441C); *YLR448W* (iYLR447C); *YML026C* (iYML026C); *YML073C* (iYML073C); *YMR242C* (iYMR242C); *YNL069C* (iYNL069C); *YNL096C* (iYNL096C); *YOL039W* (iYOL040C); *YOL040C* (iYOL040C); *YOL127W* (iYOL128C); *YOR234C* (iYOR234C); *YOR293W* (iYOR292C); *YPL090C* (iYPL090C); *YPL131W* (iYPL132W); *YPR102C* (iYPR102C); *YPR132W* (iYPR131C); *YDR064W* (iYDR063W); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YKL156W* (iYKL158W, iYKL157W); *YLR167W* (iYLR166C); *YOL121C* (iYOL121C); *YBL087C* (iYBL087C); *YER056C-A* (iYER056C-A); *YGL103W* (iYGL104C); *YLR061W* (iYLR060W); *YPL143W* (iYPL144W); *YPR080W* (iYPR079W); *YGL135W* (iYGL136C); *YGR118W* (iYGR117C); *YLR185W* (iYLR184W); *YNL162W* (iYNL163C); *YOL120C* (iYOL120C); *YEL076C* (iYEL075C); *YHR204W* (iYHR203C); *YPR131C* (iYPR131C)

RCS1: *YDR064W* (iYDR063W); *YFR001W* (iCEN6); *YGR187C* (iYGR187C); *YJR007W* (iYJR006W); *YKR026C* (iYKR026C, iYKR-Wdelta10); *YLR002C* (iYLR002C); *YLR061W* (iYLR060W); *YLR167W* (iYLR166C); *YOL144W* (iYOL145C); *YPL019C* (iYPL019C); *YPR137W* (iYPR135W)

REB1: *YDR465C* (iYDR465C); *YER049W* (iYER048W-A); *YER156C* (iYER156C); *YFL023W* (iYFL024C); *YGL120C* (iYGL120C); *YGR173W* (iYGR172C); *YGR272C* (iYGR272C); *YHR170W* (iYHR169W); *YIL148W* (iYIL149C); *YJL148W* (iSNR190); *YKL081W* (iYKL082C); *YKL082C* (iYKL082C); *YLL034C* (iYLL034C); *YLR083C* (iYLR083C); *YLR150W* (iYLR149C); *YLR367W* (iYLR366W); *YMR260C* (iYMR260C); *YNL113W* (iYNL115C); *YNR012W* (iYNR011C); *YOR116C* (iYOR116C); *YOR272W* (iSNR8); *YOR335C* (iYOR335C); *YPL211W* (iYPL212C); *YPL212C* (iYPL212C); *YPR163C* (iYPR163C)

RLM1: *YDR309C* (iYDR309C); *YGR189C* (iYGR189C); *YHR193C* (iYHR193C); *YJL148W* (iSNR190); *YLR300W* (iYLR299W); *YLR342W* (iYLR341W); *YEL040W* (iYEL041W); *YJL158C* (iYJL158C)

RME1: *YDL184C* (iYDL184C); *YDL192W* (iYDL193W); *YDR144C* (iYDR144C); *YER007C-A* (iYER007C-A); *YGL135W* (iYGL136C); *YIL052C* (iYIL052C); *YJL148W* (iSNR190); *YLR061W* (iYLR060W); *YLR065C* (iYLR065C); *YLR074C* (iYLR074C); *YLR075W* (iYLR074C); *YLR367W* (iYLR366W); *YOR236W* (iYOR235W); *YOR369C* (iYOR369C); *YPL143W* (iYPL144W); *YPL198W* (iYPL199C)

ROX1: *YAL003W* (iYAL004W); *YDL148C* (iYDL148C); *YEL040W* (iYEL041W); *YEL055C* (iYEL055C); *YER007C-A* (iYER007C-A); *YGL135W* (iYGL136C); *YHR193C* (iYHR193C); *YJL148W* (iSNR190); *YLR075W* (iYLR074C); *YLR367W* (iYLR366W); *YMR093W* (iYMR092C); *YOL121C* (iYOL121C); *YOR315W* (iYOR314W); *YPR010C* (iYPR010C); *YDL245C* (iYDL245C-0, iYDL245C-1); *YIL099W* (iYIL100W); *YJL221C* (iYJL220W); *YJR158W* (iYJR157W-1, iYJR157W-0); *YMR017W* (iYMR016C-1, iYMR016C-0); *YMR135C* (iYMR135W-A)

RPN4: *YAL003W* (iYAL004W); *YBL068W* (iYBL069W); *YBL087C* (iYBL087C); *YBR143C* (iYBR143C); *YBR189W* (iYBR188C); *YDR152W* (iYDR151C); *YFL023W* (iYFL024C); *YHR068W* (iYHR067W); *YKL009W* (iYKL010C); *YKL014C* (iYKL014C); *YKR060W* (iYKR059W); *YLL035W* (iYLL036C); *YLR167W* (iYLR166C); *YLR412W* (iYLR411W); *YMR131C* (iYMR131C); *YMR229C* (iYMR229C); *YMR239C* (iYMR239C); *YNL087W* (iYNL088W); *YNR012W* (iYNR011C); *YOL080C* (iYOL079W); *YOR236W* (iYOR235W); *YOR252W* (iYOR251C)

SFP1: *YER103W* (iYER102W); *YFR017C* (iYFR017C); *YGL104C* (iYGL104C); *YGR250C* (iYGR250C-1, iYGR250C-0); *YJL068C* (iYJL068C); *YKR009C* (iYKRCdelta8, iYKR009C); *YML070W* (iYMLCdelta1, itR(UCU)M2); *YMR304W* (iYMR303C); *YNL241C* (iYNL241C); *YNL305C* (iYNL305C); *YDL075W* (iYDL076C); *YDL082W* (iYDL083C); *YDL083C* (iYDL083C); *YDR471W* (iYDR470C); *YEL054C* (iYEL054C); *YER117W* (iYER116C); *YFR031C-A* (iYFR031C-A); *YGL030W* (iYGL031C); *YGL031C* (iYGL031C); *YGL103W* (iYGL104C); *YGL135W* (iYGL136C); *YHR021C* (iYHR021C); *YHR141C* (iYHR141C); *YIL133C* (iYIL133C); *YIL148W* (iYIL149C); *YLR048W* (iYLR047C); *YLR167W* (iYLR166C); *YNL096C* (iYNL096C); *YOL127W* (iYOL128C); *YPR102C* (iYPR102C)

SKN7: *YJL148W* (iSNR190); *YKR043C* (iYKR043C); *YMR305C* (iYMR305C); *YOL039W* (iYOL040C); *YOL040C* (iYOL040C); *YOR276W* (iYOR275C); *YOR315W* (iYOR314W); *YPL050C* (iYPL050C); *YPR009W* (iYPR008W); *YER079W* (iYER078C); *YFR017C* (iYFR017C); *YJL100W* (iYJL101C); *YML100W* (iYML101C); *YMR135C* (iYMR135W-A); *YMR136W* (iYMR135W-A); *YPR149W* (iYPR148C-1, iYPR148C-0); *YPR151C* (iYPR151C); *YDR043C* (iYDR043C-1, iYDR043C-0, iYDR043C-2); *YGL227W* (iYGL228W); *YHR138C* (iYHR138C); *YIL097W* (iYIL098C); *YIL099W* (iYIL100W); *YMR017W* (iYMR016C-1, iYMR016C-0); *YOR273C* (iYOR273C); *YPL230W* (iYPL231W); *YBR077C* (iYBR077C); *YMR262W* (iYMR261C); *YNL241C* (iYNL241C); *YOR138C* (iYOR138C)

SMP1: *YBL087C* (iYBL087C); *YBR189W* (iYBR188C); *YBR191W* (iYBR189W); *YDL060W* (iYDL061C); *YDL061C* (iYDL061C); *YDR418W* (iYDR416W); *YDR500C* (iYDR500C); *YEL040W* (iYEL041W); *YEL054C* (iYEL054C); *YER007C-A* (iYER007C-A); *YER074W* (iYER073W); *YGL135W* (iYGL136C); *YGR283C* (iYGR283C); *YHR193C* (iYHR193C); *YIL133C* (iYIL133C); *YJL191W* (iYJL192C); *YJL192C* (iYJL192C); *YMR014W*

(iYMR013C-1); *YNL113W* (iYNL115C); *YOL121C* (iYOL121C); *YOR369C* (iYOR369C); *YPL050C* (iYPL050C); *YPL198W* (iYPL199C)

SOK2: *YDL245C* (iYDL245C-0, iYDL245C-1); *YEL069C* (iYEL069C-1, iYEL069C-0); *YEL070W* (iYEL071W-1, iYEL071W-0); *YJL221C* (iYJL220W); *YJR158W* (iYJR157W-1, iYJR157W-0)

STB1: *YDR500C* (iYDR500C); *YOR246C* (iYOR246C); *YOR247W* (iYOR246C); *YOR248W* (iYOR246C); *YPL256C* (iYPL256C)

STE12: *YFL027C* (iYFL027C); *YHR005C* (iYHR005C); *YHR084W* (iYHR083W); *YLR452C* (iYLR452C); *YML046W* (iYML047C); *YCL027W* (iYCL028W); *YCL055W* (iYCL056C); *YCR089W* (iYCR088W); *YIL037C* (iYIL037C); *YIL082W* (iYIL083C, YILWTy3-1A); *YKL189W* (itL(UAA)K); *YMR198W* (iYMR197C); *YNL279W* (iYNL280C); *YOR343C* (iYORWdelta22, iYOR343C); *YDR461W* (iYDR460W); *YFL026W* (iYFL027C); *YGL032C* (iYGL032C); *YKL209C* (iYKL209C); *YNL145W* (iYNL146W); *YGR014W* (iYGR013W); *YGR189C* (iYGR189C); *YIL123W* (iYIL124W); *YOR247W* (iYOR246C); *YOR248W* (iYOR246C)

SWI4: *YDR453C* (iYDR453C); *YDR513W* (iYDR511W); *YJR044C* (iYJR044C); *YKL007W* (iYKL008C); *YKL094W* (iYKL095W); *YKL103C* (iYKL103C); *YKR011C* (iYKR011C); *YER079W* (iYER078C); *YFR017C* (iYFR017C); *YGL227W* (iYGL228W); *YHR194W* (iYHR193C); *YML100W* (iYML101C); *YDR261C* (iYDR261C-0, iYDR261C-1); *YIL123W* (iYIL124W); *YLR084C* (iYLR084C); *YOR315W* (iYOR314W); *YPR119W* (iYPRWsigma2-1, iYPRWsigma2-0); *YBR078W* (iYBR077C); *YGR189C* (iYGR189C); *YLR300W* (iYLR299W); *YMR305C* (iYMR305C); *YOR247W* (iYOR246C); *YOR248W* (iYOR246C); *YEL040W* (iYEL041W); *YMR135C* (iYMR135W-A); *YMR136W* (iYMR135W-A); *YPR149W* (iYPR148C-1, iYPR148C-0); *YBR162C* (iYBR162C); *YDR309C* (iYDR309C); *YJL158C* (iYJL158C); *YNL178W* (iYNL179C-1, iYNL179C-0); *YLR342W* (iYLR341W); *YBR077C* (iYBR077C); *YDR500C* (iYDR500C); *YOR246C* (iYOR246C); *YPL256C* (iYPL256C); *YGR014W* (iYGR013W); *YKR013W* (iYKR011C); *YNL300W* (iYNL301C); *YPL163C* (iYPL163C); *YGL008C* (iYGL008C); *YER070W* (iYER069W); *YJL187C* (iYJL187C); *YML027W* (iYML028W); *YMR144W* (iYMR143W); *YMR179W* (iYMR178W); *YMR199W* (iYMR198W); *YPL267W* (iYPL268W); *YPR120C* (iYPR120C)

SWI5: *YDR398W* (iYDR397C); *YFL022C* (iYFL022C); *YFR031C-A* (iYFR031C-A); *YHL013C* (iYHL013C); *YIL018W* (iYIL019W); *YJL080C* (iYJL080C); *YJL148W* (iSNR190); *YLR276C* (iYLR276C); *YML125C* (iYML125C); *YPL050C* (iYPL050C); *YBR158W* (iYBR157C-1, iYBR157C-0); *YGL028C* (iYGL028C); *YJL078C* (iYJL078C); *YLR286C* (iYLR286C); *YNL327W* (iYNL328C); *YDL223C* (iYDL223C); *YFR017C* (iYFR017C); *YGL227W* (iYGL228W); *YHR138C* (iYHR138C); *YJR044C* (iYJR044C); *YER079W* (iYER078C); *YML100W* (iYML101C); *YMR262W* (iYMR261C); *YNL241C* (iYNL241C); *YOR138C* (iYOR138C); *YKL150W* (iYKL151C); *YKL151C* (iYKL151C)

SWI6: *YDR144C* (iYDR144C); *YDR500C* (iYDR500C); *YHL013C* (iYHL013C); *YLL034C* (iYLL034C); *YML082W* (iYML083C); *YNL301C* (iYNL301C); *YNL313C* (iYNL313C); *YNR054C* (iYNR054C-0, iYNR054C-1); *YOR056C* (iYOR056C); *YOR310C* (iYOR310C); *YPL126W* (iYPL127C); *YDL003W* (iYDL004W); *YDR097C* (iYDR097C); *YDR507C* (iYDR507C); *YIL026C* (iYIL026C); *YJL074C* (iYJL074C); *YJR030C* (iYJR030C); *YKL113C* (iYKL113C); *YLR103C* (iYLR103C); *YMR076C* (iYMR076C); *YNL273W* (iYNL274C); *YOR074C* (iYOR074C); *YBR078W* (iYBR077C); *YGR014W* (iYGR013W); *YIL123W* (iYIL124W); *YJL158C* (iYJL158C); *YKR013W* (iYKR011C); *YLR300W* (iYLR299W); *YMR305C* (iYMR305C); *YNL300W* (iYNL301C); *YPL163C* (iYPL163C); *YER070W* (iYER069W); *YJL187C* (iYJL187C); *YML027W* (iYML028W); *YMR144W* (iYMR143W); *YMR179W* (iYMR178W); *YMR199W* (iYMR198W); *YPL267W* (iYPL268W); *YPR120C* (iYPR120C)

YAP1: *YFL056C* (iYFL056C-0, iYFL056C-1); *YFL057C* (iYFL056C-0); *YGL114W* (iYGL115W); *YKL071W* (itW(CCA)K-1, itW(CCA)K-0); *YLR460C* (iYLR460C-0, iYLR460C-1); *YML131W* (iYML132W-0, iYML132W-1); *YKL103C* (iYKL103C); *YLL060C* (iYLL060C); *YML116W* (iYML117W-A); *YNL134C* (iYNL134C); *YOL119C* (iYOL119C)

YAP5: *YDR287W* (iYDR286C); *YDR368W* (iYDR367W); *YGL104C* (iYGL104C); *YGR149W* (iYGR148C); *YHR087W* (iYHR086W); *YJR044C* (iYJR044C); *YOR292C* (iYOR292C); *YBR191W* (iYBR189W); *YDL082W* (iYDL083C); *YDL083C* (iYDL083C); *YDL191W* (iYDL192W); *YDR449C* (iYDR449C); *YDR450W* (iYDR449C); *YDR471W* (iYDR470C); *YEL054C* (iYEL054C); *YER074W* (iYER073W); *YER117W* (iYER116C); *YGL189C* (iYGL189C); *YGR034W* (iYGR033C); *YGR148C* (iYGR148C); *YHR141C* (iYHR141C); *YHR203C* (iYHR203C); *YIL018W* (iYIL019W); *YIL133C* (iYIL133C); *YIL148W* (iYIL149C); *YJL136C* (iYJL136C); *YJL177W* (iYJL178C); *YJL189W* (iYJL190C); *YJL190C* (iYJL190C); *YLR048W* (iYLR047C); *YLR441C* (iYLR441C); *YLR448W* (iYLR447C); *YML026C* (iYML026C); *YML063W* (iYML064C); *YML073C* (iYML073C); *YNL069C* (iYNL069C); *YOL127W* (iYOL128C); *YOR234C* (iYOR234C); *YOR293W* (iYOR292C); *YPL090C* (iYPL090C); *YPR132W* (iYPR131C); *YER056C-A* (iYER056C-A); *YGL103W* (iYGL104C); *YPR080W* (iYPR079W); *YGR118W* (iYGR117C); *YLR185W* (iYLR184W); *YNL162W* (iYNL163C); *YOL120C* (iYOL120C); *YHR091C* (iYHR091C); *YEL076C* (iYEL075C); *YHR204W* (iYHR203C)

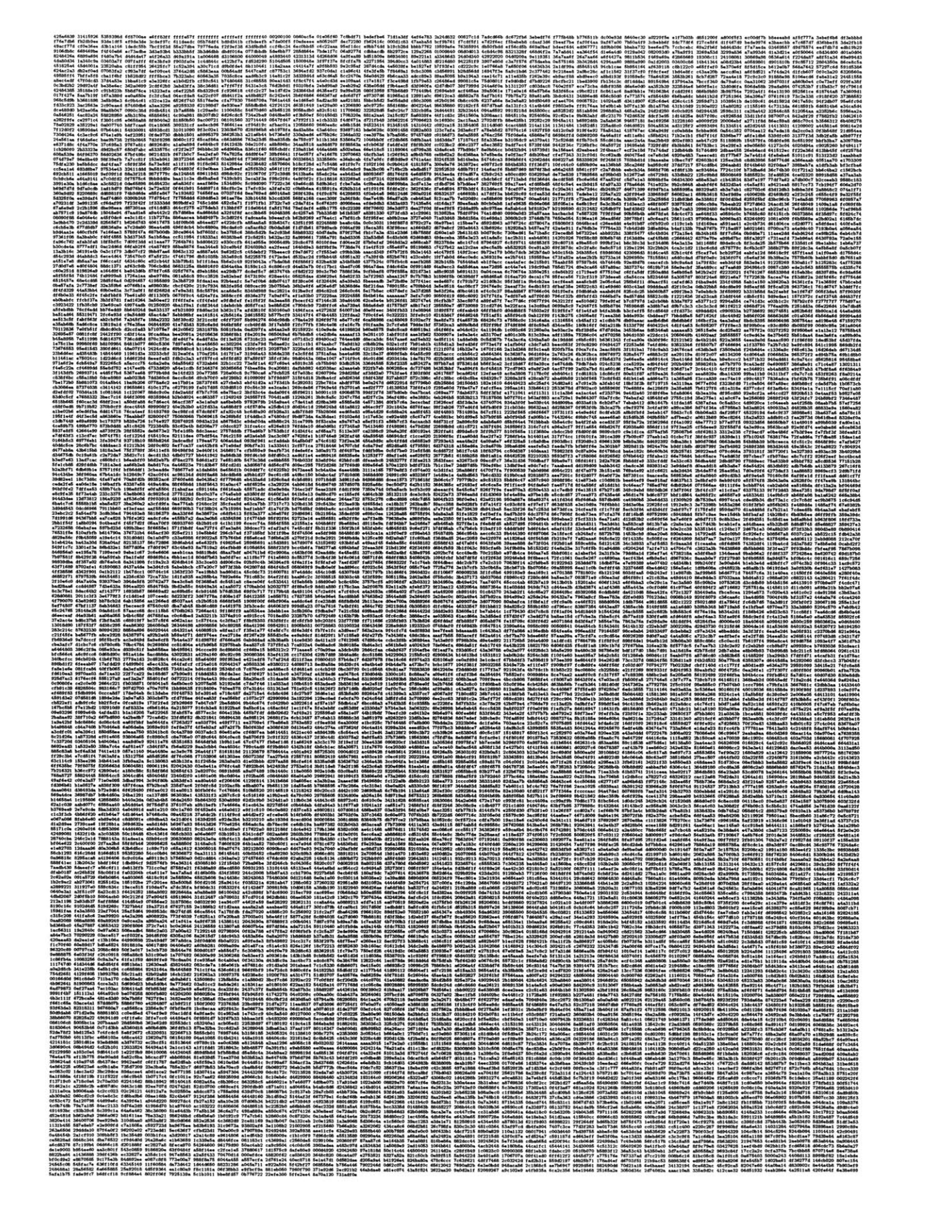
YAP6: *YBL027W* (iYBL028C); *YBL028C* (iYBL028C); *YDR144C* (iYDR144C); *YEL040W* (iYEL041W); *YER007C-A* (iYER007C-A); *YJL148W* (iSNR190); *YJL191W* (iYJL192C); *YJR094W-A* (iYJR094C-0, iYJR094C-1); *YKL110C* (iYKL110C-0, iYKL110C-1); *YLR074C* (iYLR074C); *YLR367W* (iYLR366W); *YMR014W* (iYMR013C-1); *YOL121C* (iYOL121C); *YOR369C* (iYOR369C); *YJR145C* (iYJR145C); *YLR075W* (iYLR074C); *YLR413W* (iYLR412W-1, iYLR412W-0); *YNL087W* (iYNL088W); *YNL178W* (iYNL179C-1, iYNL179C-0); *YOR315W* (iYOR314W); *YDL245C* (iYDL245C-0, iYDL245C-1); *YIL099W* (iYIL100W); *YJL221C* (iYJL220W); *YJR158W* (iYJR157W-1, iYJR157W-0); *YMR017W* (iYMR016C-1, iYMR016C-0); *YMR135C* (iYMR135W-A); *YEL069C* (iYEL069C-1, iYEL069C-0); *YEL070W* (iYEL071W-1, iYEL071W-0)

YFL044C: *YBR154C* (iYBR154C); *YBR155W* (iYBR154C); *YCR072C* (iYCR073C); *YGR083C* (iYGR083C); *YGR128C* (iYGR128C); *YLR017W* (iYLR016C); *YLR409C* (iYLR409C); *YNL141W* (iYNL142W); *YPR110C* (iYPR110C); *YPR112C* (iYPR112C)

Appendix B

Source code

This appendix contains the source code to the programs used in this thesis. The data below encodes (in hexadecimal) a `tar` file compressed by the `bzip2` program. It should be relatively easy to extract the text below from a PDF version of this document (cut and paste with Acrobat reader) or directly out of the Postscript using a text processing program like `perl`.



Bibliography

[1] Timothy L. Bailey and Charles Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pages 28–36, Menlo Park, California, 1994. AAAI Press.

[2] Ziv Bar-Joseph, Georg K. Gerber, Tong Ihn Lee, Nicola J. Rinaldi, Jane Y. Yoo, François Robert, D. Benjamin Gordon, Ernest Fraenkel, Tommi S. Jaakkola, Richard A. Young, and David K. Gifford. Computational discovery of gene modules and regulatory networks. *(Submitted for publication)*, 2003.

[3] Y. Barash, G. Bejerano, and N. Friedman. A simple hyper-geometric approach for discovering putative transcription factor binding sites. In O. Gascuel and B. M. E. Moret, editors, *Algorithms in Bioinformatics: Proc. First International Workshop, 2001*, number 2149 in Lecture Notes in Computer Science, pages 278–293. Springer-Verlag Heidelberg, 2001.

[4] Mathieu Blanchette and Saurabh Sinha. Separating real motifs from their artifacts. In *Proceedings of ISMB*, 2001.

[5] A. Brazma, I. Jonassen, J. Vilo, and E. Ukkonen. Predicting gene regulatory elements in silico on a genomic scale. *Genome Research*, 8(11):1202–1215, November 1998.

[6] J. Buhler and M. Tompa. Finding motifs using random projections. In *Proceedings of The Fifth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 2001.

[7] D. B. Gordon, 2003. personal communication.

[8] D. Benjamin Gordon, Lena Nekludova, Nicola J. Rinaldi, Craig M. Thompson, David K. Gifford, Tommi Jaakkola, Richard A. Young, and Ernest Fraenkel. A knowledge-based analysis of high-throughput data reveals the mechanisms of transcription factor specificity. *(Submitted for publication)*, 2003.

[9] X. S. Liu, D. L. Brutlag, and J. S. Liu. An algorithm for finding protein-DNA binding sites with applications to chromatin-immunoprecipitation microarray experiments. *Nature Biotechnology*, 20:835–839, August 2002.

[10] Bing Ren, François Robert, John J. Wyrick, Oscar Aparicio, Ezra G. Jennings, Itamar Simon, Julia Zeitlinger, Jörg Schreiber, Nancy Hannett, Elenita Kanin, Thomas L. Volkert, Christopher J. Wilson, Stephen P. Bell, and Richard A. Young. Genome-wide location and function of DNA binding proteins. *Science*, 290:2306–2309, 2000.

[11] S. Sinha. Discriminative motifs. In *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, 2002.

[12] J. Vilo, A. Brazma, I. Jonassen, A. Robinson, and E. Ukkonen. Mining for putative regulatory elements in the yeast genome using gene expression data. In *Proceedings International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 384–394, August 2000.