Identification of Transcriptionally Quiescent Regions in the Neurospora Crassa Genome

Katie Marie Groskreutz
Illinois State University, katie.groskreutz@gmail.com

IDENTIFICATION OF TRANSCRIPTIONALLY QUIESCENT REGIONS OF THE *NEUROSPORA CRASSA* GENOME

Katie M. Groskreutz

62 Pages May 2014

Sexual reproduction and genetic exchange via meiosis are important and highly conserved processes in many living organisms. Occasionally, complications occur during meiosis that can result in chromosome abnormalities. In humans, improper chromosome development can cause life altering disorders such as Down Syndrome, Edwards Syndrome, and Patau Syndrome. Unfortunately, despite its importance, gaps remain in our knowledge of how this process works. For instance, little is known about how homolog identification occurs and what proteins identify matching chromosomes during pairing. This fundamental process occurs early during meiosis and ensures proper development of gametes.

Understanding the proteins involved during homolog pairing may be possible by studying a process called meiotic silencing by unpaired DNA (MSUD) in the eukaryotic fungus, *Neurospora crassa*. During MSUD, unpaired regions (or regions that do not match during homolog identification) are thought to produce special RNA molecules. Discovery of these molecules should help elucidate how unpaired DNA is identified.

This is because it is possible that the proteins involved in identifying unpaired regions in MSUD are the same proteins that identify homologs in meiosis. Furthermore, these proteins could contribute to homology searches required for DNA repair, which could contribute in the development of cancer research.

To gain a complete understanding of unpaired DNA detection, the *Neurospora crassa* transcriptome must be identified. The transcriptome represents all the RNA molecules found within an organism at a certain point in time or stage of development. Knowledge of the transcriptome can be used in efforts towards identifying the theoretical RNA molecules of MSUD. The meiotic transcriptome can be determined by performing an RNA-seq analysis on all the RNA transcripts produced during meiosis. These RNA are then aligned to the *N. crassa* genome. Then, a special algorithm is used to identify key regions of the genome that may prove particularly useful in MSUD research (i.e. transcriptionally quiescent regions). Given the sheer size of the data sets required for identifying these regions, the algorithm must be time and memory efficient due to computational constraints.

IDENTIFICATION OF TRANSCRIPTIONALLY QUIESCENT REGIONS OF THE *NEUROSPORA CRASSA* GENOME

KATIE M. GROSKREUTZ

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Mathematics

ILLINOIS STATE UNIVERSITY

2014

IDENTIFICATION OF TRANSCRIPTIONALLY QUIESCENT REGIONS OF THE *NEUROSPORA CRASSA* GENOME

KATIE M. GROSKREUTZ

COMMITTEE MEMBERS:

Thomas Hammond, Chair

Papa Sissokho, Co-Chair

Michael Plantholt

ACKNOWLEDGEMENTS

I would like to send a special thanks out to Tom Hammond. He has trusted me to help with his research and been more than patient in helping me learn the biology and programming surrounding this project. I would also like to thank Papa Sissokho and Mike Plantholt for serving on my thesis committee. Tyler Malone assisted me with performing the tedious BLAST analysis. I also feel it's very necessary to thank Scott Vossen for being my emotional rock through the roller coaster that has been graduate school. Both of my parents, Jeff and Linda Groskreutz, have supported me financially throughout my entire education, and I am eternally grateful to them as well. Last but not least, I cannot forget my amazing friends, who were always there to listen and support me.

K.M.G.

CONTENTS

TABLES

FIGURES

CHAPTER I

BIOLOGICAL BACKGROUND

1.1: Biological Background

This section is intended to give a brief description of the biological processes that are relevant to this project.

1.1.1: Meiosis

Meiosis is a specific type of cell division used for the production of sex cells (spores in the case of fungi) in eukaryotes. The steps involved in meiotic cell division are similar to those for other somatic cells, which divide through a well-known process called mitosis. Meiosis, shown in Figure 1, begins with a duplication of chromosomes (Figure 1a). Next, each chromosome pairs with its homolog (Figure 1b). This pairing allows for the exchange of genetic material, which contributes to genetic variation, through a process called crossing over. Homologous chromosomes are moved to opposite poles of the cell and the original cell is then split into two daughter cells (Figure 1c). This process repeats again, this time without initial duplication of the chromosomes (Figure 1d). This results in each of the two daughter cells dividing to produce four cells with half the genetic information (Figure 1e).

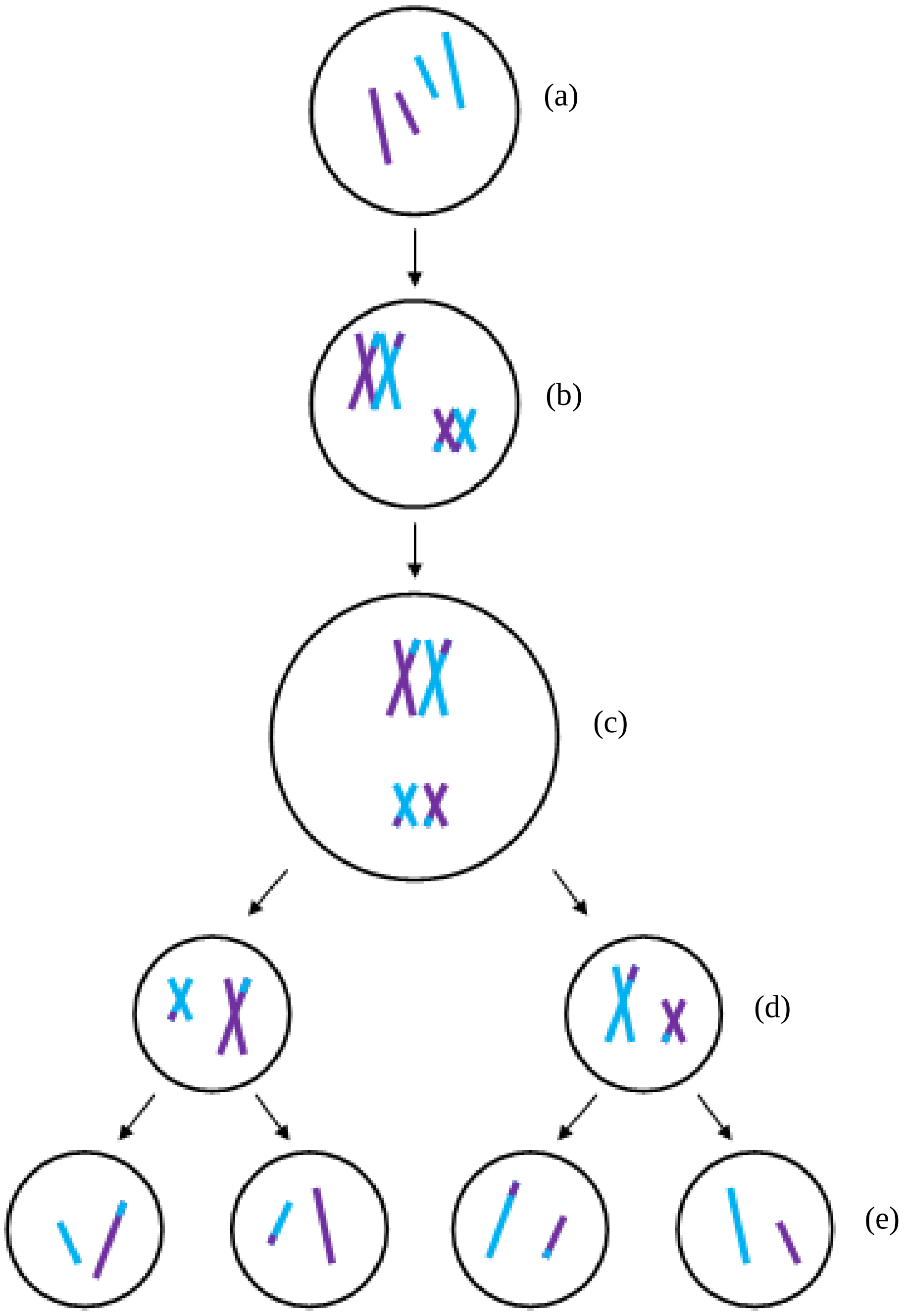


Figure 1. Description of Meiosis.
Chromosomes in a single cell duplicate, align, exchange genetic material, then segregate to form two new cells that are genetically different than the parent cell (a-d). Haploid cells are produced in the final stage of meiosis (e).

1.1.2: DNA, RNA, Transcription & Translation

DNA (deoxyribonucleic acid) is a molecule found in all living organisms that contains the genetic instructions for protein synthesis. DNA is made up of four different nucleotide bases: adenine (A), thymine (T), guanine (G) and cytosine (C). The central dogma of molecular biology explains the process of how DNA's genetic information is used to create functional proteins. Such proteins are responsible for representing the physical traits or the phenotype that we can see, i.e. height or hair color.

Transcription is the first step in this process. Double stranded DNA is transcribed into single stranded RNA (ribonucleic acid) via transcription. RNA is similar to DNA in that it has four nucleotide bases, except instead of thymine it has uracil (U). The newly created RNA strand is known as the "complement" to its parent DNA strand. In the case of DNA, a complement means that adenine pairs with thymine (uracil in RNA) and guanine pairs with cytosine.

Once the DNA has been transcribed, the RNA molecules can take on many different varieties each of which is responsible for a special role, such as protein production or gene regulation. There are a few types of RNA that are important to know for this project. The first and most common type is messenger RNA or mRNA. Each mRNA contains a specific sequence that codes for an explicit amino acid chain. This amino acid chain forms a functional protein product that gives rise to a phenotype. The process by which proteins are produced from mRNA is called translation. Other types of RNA, namely, aRNA, dsRNA, and masiRNA, are also important to know for the background of this project. However, they will be introduced and discussed in section 1.2.2.

1.1.3: An Introduction to *Neurospora crassa*

Neurospora crassa is a type of bread mold belonging to the phylum Ascomycota. *N. crassa* is a model organism for genetics research projects because it is easily maintained and cultured and has a rich history that makes use of well established protocols for genetic and biochemical techniques (Davis 2000). In particular, *N. crassa* has played a crucial role in the understanding of genome defense systems and gene silencing mechanisms, making it an important organism in which to better understand these processes (Davis 2000).

1.1.4: Genomic Invaders

Transposable elements are mobile sequences found within the genomes of most organisms. They often replicate, and then move to new locations in their host genomes. Transposable elements can be dangerous because they can insert themselves into coding regions or regions that regulate gene expression. This process is shown in Figure 2. The structure of most eukaryotic genomes suggests that they contain a relatively large proportion of transposable elements (Slotkin and Martienssen 2007). *N. crassa*, on the other hand, shows little evidence of these genome manipulators (Cambareri et al. 1991).

Given the potentially harmful nature of genomic invaders, such as transposable elements and viruses, it is not surprising that organisms have evolved different approaches to handle them. *N. crassa* utilizes many different methods in an attempt to keep these elements out of their genome (such methods are discussed in detail below). Plants, on the other hand, appear to have embraced the dynamic genome, as over half of

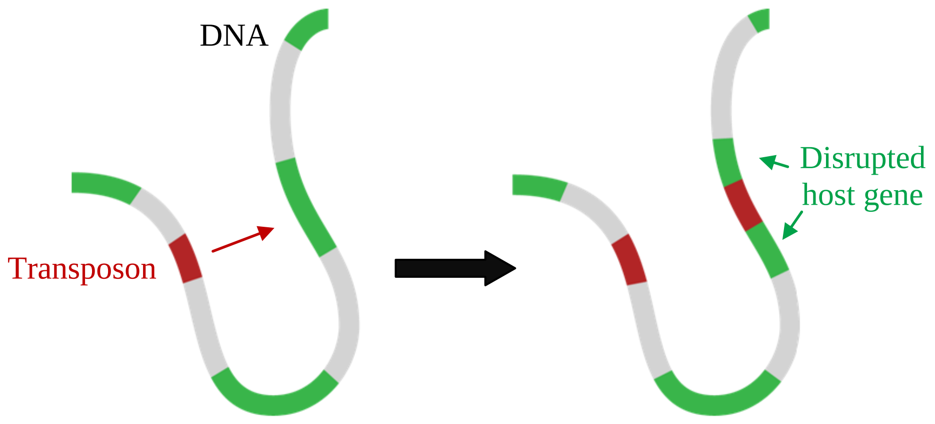


Figure 2. Depiction of Transposable Elements.
Transposable elements are mobile sequences that often replicate and move to a new location in their host genomes. Transposable elements can be dangerous because they can insert themselves into coding regions or regions that regulate gene expression. Figure adapted from Norris 2013.

their genome can be comprised of transposable elements (Feschotte et al. 2002).

1.2: Genomic Defenses in *Neurospora crassa*

Within the genome, there can be potentially harmful hitchhikers, like viruses or transposable elements, which can insert themselves into new locations the genomic code. There are many different ways organisms have evolved to defend their genomes against these 'selfish genetic elements'. *N. crassa*, in particular, has developed several different methods. They include DNA methylation, repeat point mutation (RIP), and meiotic silencing by unpaired DNA (MSUD) (Galagan et al. 2003, Borkovich et al. 2004).

1.2.1: DNA Methylation & RIP

The process of DNA methylation serves many purposes. It silences genes by the addition of a methyl group to a nucleotide base. Also, during gamete development, DNA methylation plays an important role for ensuring that embryonic stem cells differentiate into other specific cell types. This is a permanent process that prevents a cell from changing to another type (Lister et al. 2009). DNA methylation has also been found to play a critical role in the development of cancerous tumors (Foss et al. 1993).

Occurring during the premeiotic phase in certain fungal species, RIP (repeat induced point mutation) acts on duplicated sequences within DNA, such as transposable elements that insert themselves in the genome at multiple locations, by inducing C-to-T and G-to-A mutations (Kelly & Aramayo 2007, Freitag et al 2002, Hood 2005). This process silences these sequences to protect the native genome against foreign invaders. It is believed that such alterations by RIP may trigger cytosine DNA methylation (Singer et al. 1995). It is also hypothesized to be the reason *N. crassa* has so few duplicated genes and almost no transposable elements found within its genome (Galagan et al. 2004). Essentially, the only transposable elements that exist are possible relics of old transposons that have been highly mutated by RIP (Singer et al. 1995).

1.2.2: MSUD

A process called meiotic silencing by unpaired DNA (MSUD) in *N. crassa* may better help us understand the largely obscure process of homolog pairing. MSUD occurs when a mechanism identifies regions of the chromosome that do not "pair" during homolog identification in meiosis (Figure 3 (1)) (Shiu et al. 2001). Currently, little is known about the proteins that work to identify homologous regions within chromosomes. Evidence suggests that *N. crassa,* more so than other organisms, requires a higher proportion of matches for a chromosome to be considered a homologous pair (Pratt et al. 2004).

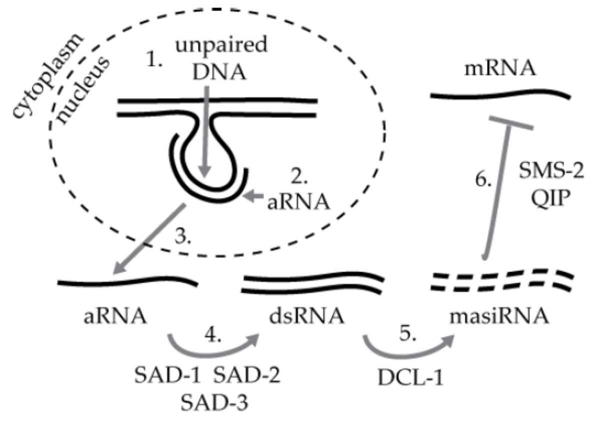


Figure 3. *Neurospora crassa* Meiotic Silencing Model.
The first three steps of this process are performed by unknown proteins. Aberrant RNA may be created from unpaired regions of DNA that do not have a homologous pair. It would then be converted to dsRNA, then masiRNA (steps 4 and 5). These masiRNA molecules silence complimentary mRNA (step 6). The dashed ellipse represents the nuclear membrane. Figure adapted from Hammond et al. 2011.

The current working model of MSUD is as follows: once an unpaired region is found, a theoretical aberrant RNA (aRNA) molecule is synthesized from that region's DNA (Figure 3 (2)). The aRNA may then be exported out of the nucleus (Figure 3 (3)), before it is converted to dsRNA (double stranded RNA) by RdRP (RNA-dependent RNA polymerase). Now, Dicer, a protein that cleaves dsRNA, can convert the dsRNA into siRNAs (short interfering RNA) (Figure 3 (5)), which are 20-25 base pairs in length (Galagan et al. 2003, Lee et al. 2003, Borkovich et al. 2004, Catalanotto et al. 2004). When these siRNAs are a part of the meiotic silencing process, they are called meiotic-silencing associated short interfering RNA (masiRNA) (Hammond et al. 2013). These masiRNA are likely incorporated into an RNA-induced silencing complex (RISC), which uses them as templates for identifying complementary mRNA for destruction or translational suppression (Figure 3 (6)). Thus, complementary mRNA is degraded no matter if is from an unpaired or paired sequence. Additional information about the proteins involved in MSUD can be found in Table 1.

Since little is known of the proteins that identify homologs, it is possible that the proteins involved in identifying unpaired regions in MSUD could also be the same proteins that identify homologs in meiosis.

Table 1. Summary of Proteins Involved in MSUD.

Protein	Location	Function
SAD-5	nuclear	unknown function [a]
SAD-1	perinuclear	turns aRNA into dsRNA [b]
SAD-2	perinuclear	serves as a scaffold for other MSUD proteins [c]
SAD-3	perinuclear	helps SAD-1 [d]
SAD-4	perinuclear	unknown function [a]
DCL-1	perinuclear	a Dicer protein that cleaves dsRNA into siRNAs [e]
SMS-2	perinuclear	uses siRNA to target complementary mRNAs [f]
QIP	perinuclear	processes siRNAs into single strands [g,h]

Sources: Hammond et al. 2013[a], Shui and Metzenberg 2002[b], Shiu et al. 2006[c], Hammond et al. 2011[d], Alexander et al. 2008[e], Lee et al. 2003[f], Lee et al. 2010[g], Xiao et al. 2010[h]

1.3: Importance of Quiescent Regions

Aberrant RNA or aRNA that is created from unpaired regions is currently theoretical. One reason why it had not been detected may be that it is difficult to distinguish it from other forms of RNA. Some attempts to do so have been made without success using standard molecular biology techniques. Here we will try to identify the most transcriptionally quiescent regions of the genome with the use of next-generation sequencing. Once the regions have been identified, we will try to force the creation of aRNA in these regions by unpairing them during meiosis. This novel technique for finding aRNA has never been attempted before. If this is successful, all the transcripts from the unpaired loci should be aRNA. Some of the proteins involved in aRNA generation may be involved in the recognition of unpaired DNA, others may be involved in aRNA generation, and others may be involved in later stages of the process.

To gain a complete understanding of unpaired DNA detection and the proteins that drive the mechanism, all of the regions that are unpaired during MSUD must be identified throughout the *N. crassa* genome.

CHAPTER II

METHODS

2.1: The Data Sets

This section outlines the data sets and methods used for gathering and analyzing data.

2.1.1: RNA-Seq

A transcriptome is known as the set of all RNA molecules that are produced in a cell. RNA-Seq uses deep-sequencing technologies to give us a precise measurement of the level of transcripts in an individual transcriptome. In general, this information is important for determining the functional elements in the genome at any given point in time. RNA-Seq is often used for determining the structure of genes in terms of location or splicing patterns and quantifying changes in expression levels at different time periods. This method is emerging as the dominant form for measuring transcriptomes, as opposed to using microarrays, since sequencing technologies have become cheaper and more accurate (Wang et al. 2009).

2.1.2: Description of Data Sets

The data sets used for this project come from four *N. crassa* crosses and represent their meiotic transcriptomes obtained via RNA-Seq. There are two types of data sets: one that contains only large RNA's (>30 nucleotides) and one that only contains small RNA's (<30 nucleotides). Data sets F201 (SRR957218) and 1005 (SRR957223) are the large RNA data sets and SRR751454 and SRR755946 are the small RNA data sets. The goal of this project is to analyze the transcriptome of *Neurospora* and to map its quiescent regions. It is possible that regions producing large RNA are different from those that produce small RNA. Since we are looking for regions that have no detectable RNA at all, it is important to look at both types of data sets.

- <u>Data Set F201 (SRR957218):</u> This data set represents an RNA-seq analysis of all RNA from the fruiting bodies and associated vegetative tissue from a cross between strains P9-42 (Oak Ridge WT *a*) and F201 (*fl A*).
- <u>Data Set 1005 (SRR957223):</u> This data set represents an RNA-seq analysis of all RNA from the fruiting bodies and associated vegetative tissue from a cross between strains P6-07 (*rid A*) and F2-26 (*rid; fl a*). This cross is theoretically the same as the one performed for data set F201 (*fl A*), except that both strains used in the cross are mutated in a gene known as *rid*.
- <u>Data Set SRR751454:</u> This data set represents an RNA-seq analysis of the small RNA from the fruiting bodies and associated vegetative tissue of a cross between strains P3-08 (Oak Ridge WT *a*) and F201 (*fl A*). It was downloaded as a small RNA data set from NCBI.

- Data Set SRR755946: This data set represents an RNA-seq analysis of the small RNA from the fruiting bodies and associated vegetative tissue of a cross between P3-08 (Oak Ridge WT *a*) and F5-39 (*r*$^{\Delta}$; *fl A*). It was downloaded as a small RNA data set from NCBI.

2.2: Bowtie

This section is intended to describe how Bowtie, a free, open-source alignment program, works and is used for identifying the quiescent regions or regions of no transcription within the *Neurospora crassa* genome.

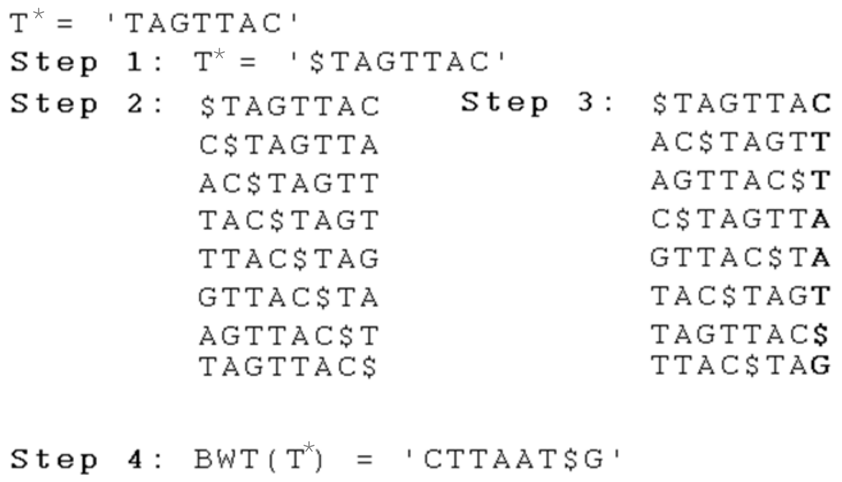
2.2.1: Introduction to Bowtie

Bowtie is a fast, yet memory efficient alignment program that can be run on a typical desktop computer. It works to align short sequences, such as 'reads' from an RNA-seq analysis, to a reference genome. Such efficiency is achieved by use of a novel indexing strategy called a Burrows-Wheeler index along with a Burrows-Wheeler transformation (BWT) (Langmead et al. 2009).

2.2.2: Burrows-Wheeler Transformation & EXACTMATCH

The BWT is a simple permutation of all the characters in a string. For instance, let T* = 'TAGTTAC' be a string of text. **Step 1** in BWT is to append a $ to the front of T* (See Figure 4). '$' is set to be lexicographically less than all other characters in T*. **Step 2** is to create a Burrows-Wheeler matrix. The rows contained in the matrix are comprised of all the cyclic rotations of T*. The next step (**Step 3**) is to sort the matrix lexicographically by the first character in each row. The Burrows-Wheeler transformation of T* (BWT(T*)) is the rightmost column of the matrix (**Step 4**).

The Burrows-Wheeler matrix has the property of last-first (LF) mapping. That is, the ith occurrence of a character X in the last column (or BWT(T*)) corresponds to the ith occurrence of X in the first column (or the lexicographically sorted T*). The last-first mapping property is necessary for the algorithms that use the BWT to search a text for an alignment.



```
T* = 'TAGTTAC'
Step 1: T* = '$TAGTTAC'
Step 2: $TAGTTAC    Step 3: $TAGTTAC
        C$TAGTTA            AC$TAGTT
        AC$TAGTT            AGTTAC$T
        TAC$TAGT            C$TAGTTA
        TTAC$TAG            GTTAC$TA
        GTTAC$TA            TAC$TAGT
        AGTTAC$T            TAGTTAC$
        TAGTTAC$            TTAC$TAG

Step 4: BWT(T*) = 'CTTAAT$G'
```

Figure 4. Burrows-Wheeler Matrix Example.
This figure shows how a Burrows-Wheeler matrix is created in order to apply the Burrows-Wheeler transformation to a string of text, T*. Figure adapted from Langmead et al. 2009.

This LF mapping is used in Bowtie's EXACTMATCH algorithm (Figure 5) to find where a short read matches a reference sequence. The example in Figure 5 uses the sequence 'TTA' as a read sequence to show how the EXACTMATCH algorithm works. The LF mapping method requires us to start with the last letter of the read, 'A', as seen in Figure 5 (a). Next, we identify the range of the rows that start with the letter A. Follow those rows that start with A to the last column of the matrix. Of these rows, the ones that contain T's (the next letter moving backwards through the read) correspond to the first and second occurrences of T in the last column. As shown in Figure 5 (b), the process starts over. We come back to the first column of the matrix to find the first and second occurrences of T's and again follow them to the rightmost column of the matrix. The next letter moving backwards through the read is a 'T'. In the rightmost column this is the third occurrence of T. This is continued for each of the characters in the read until the range equals one. Then the EXACTMATCH algorithm is done and the alignment of the read to the reference is complete.

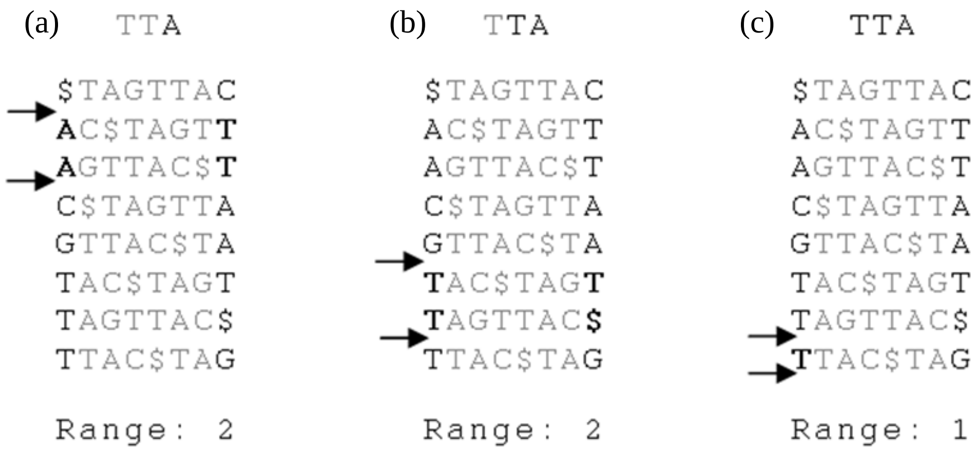


Figure 5. Last-First Mapping Example.
This figure shows how last-first mapping works to find where a read aligns to the reference sequence. Figure adapted from Langmead et al. 2009.

2.2.3: Identifying Inexact Matches

It is possible for sequencing errors or genuine differences to exist between reads and the reference sequence. Therefore, the EXACTMATCH algorithm may be insufficient in many cases. Once Bowtie has gone through the EXACTMATCH algorithm without identifying an exact match for a read, it proceeds to look for an inexact match. It does so in a similar manner to EXACTMATCH. A position that has already been matched within the read is selected and a different base is substituted in its place. This introduces a mismatch into the read. Then the EXACTMATCH algorithm continues to check for an alignment with that mismatch. If no such alignment is found, another position is randomly selected for a mismatch to be introduced.

Because Bowtie searches in this greedy, randomized, depth-first search manner, it may not find the best alignment that exists. Options such as 'best' will improve the alignment in terms of number of mismatches or quality. However, this will slow the program by as much as two or three times as much as the default mode.

2.2.4: Bowtie Compared to Other Alignment Programs

Many different topics are included as part of testing the "goodness" of an alignment program, such as CPU time, peak memory footprint, and percentage of reads aligned. In some of these performance aspects, other popular open-source alignment programs, such as SOAP and Maq (Li et al. 2008a, Li et al. 2008b), have comparable performance statistics. For instance, all three programs can achieve similar percentage of reads aligned with nearly equivalent peak memory footprint. However, Bowtie far exceeds the other two in terms of speed. For example, when aligning a set of reads with a length of 76 base pairs Bowtie takes 19 minutes of CPU time (with memory footprint: 1,323 Mb, reads aligned: 44.5%) compared to Maq's 4 hours and 45 minutes (memory footprint: 1,155 Mb, reads aligned: 44.9%) (Langmead et al. 2009). Additionally, Bowtie has the option, unlike other programs, to achieve a higher percentage aligned at the cost of speed.

2.2.5: Bowtie Features Used

Bowtie has many options, such as 'best' described above, that give it flexibility for speed, memory usage and output formats. Below are a list of the features used in this project:

- -p: This option increases the size of bowtie's memory footprint, but increases its speed (scale: 1-8). This causes it to search with a specified number of parallel threads on different cores.
- --phred33: When sequenced, each nucleotide base is assigned a certain letter based on quality or confidence of the machine doing the sequencing. This feature tells Bowtie that Phred quality scores are provided with the read sequences.
- --local: This feature allows for some characters on each end to be omitted from the alignment (also known as soft clipping).
- -a: This feature tells bowtie to report all valid alignments.

2.3: Perl Scripts -- Identify Quiescent Regions

This section describes the Perl scripts and algorithms that were developed to identify the quiescent regions of the *Neurospora crassa* genome.

2.3.1: Identify Mapped Regions -- Convert CIGAR Scores

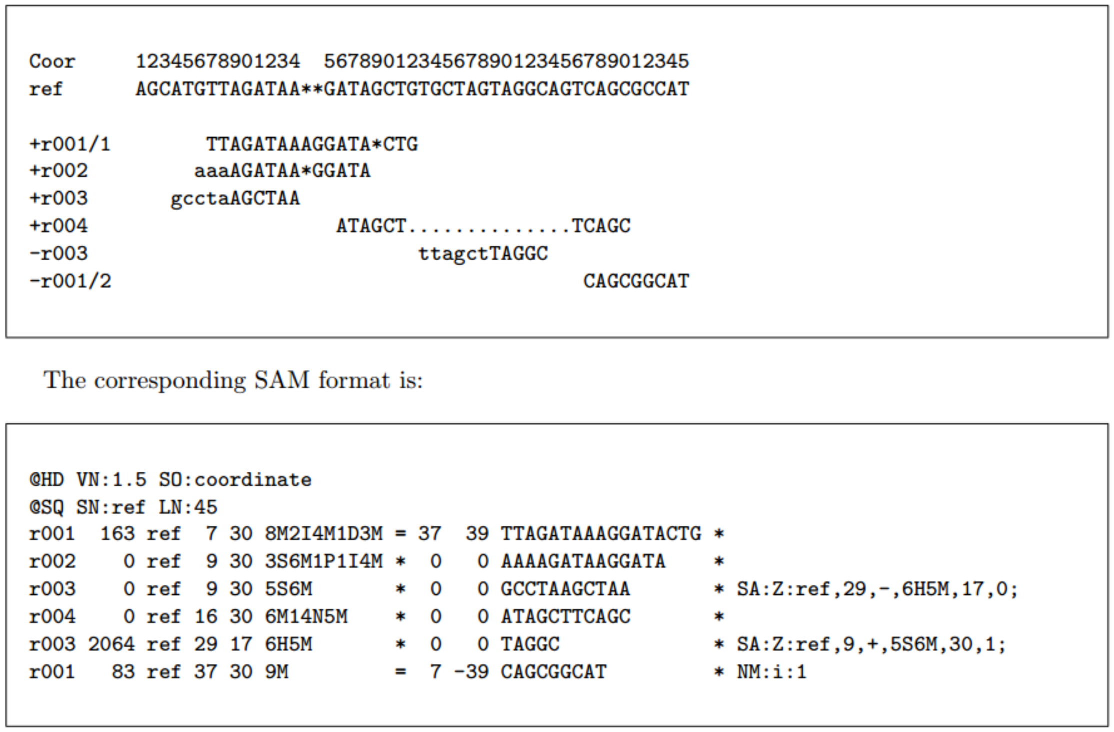
Bowtie outputs a file in SAM format (Sequence Alignment/Map Format) (Langmead and Salzberg 2012). The SAM format is tab-delimited and begins with a header section. Each line in the header begins with an '@' identifier to differentiate it from the alignment section which follows. The alignment section has different fields that give information about the alignment of each read, i.e. reference sequence name, read alignment start position, a CIGAR (Compact Idiosyncratic Gapped Alignment Report) score, etc. (Li et al. 2009).

The fields that provided the most useful information for this project were the read start position and the CIGAR score. A CIGAR string gives information about the alignment of each read. It tells where there were matches, insertions, deletions, clipped regions, etc. An example of a CIGAR string (see Figure 6) would be '8M2I4M1D3M'. Each number is followed by an operator (letter). For a list of operators see Table 2. In the example, the first 8 bases match exactly to the reference then 2 bases are inserted into the reference, and so on. Other examples of read alignments and their corresponding CIGAR scores are provided in Figure 6.

Table 2. Description of CIGAR Operators from SAM Format.

Op	Description
M	Alignment match
I	Insertion to the reference
D	Deletion from the reference
N	Skipped region from the reference
S	Soft clipping
P	Padding (silent deletion from padded reference)
X	Sequence mismatch

A Perl script was written to parse each CIGAR score in order to determine the length of each read based on this score. Once the length was determined, the start and end position of each read was calculated. Knowing the start and end positions for each read is necessary to determine which regions of the *N. crassa* genome (the reference sequence) had reads mapped to them.



```
Coor     12345678901234  5678901234567890123456789012345
ref      AGCATGTTAGATAA**GATAGCTGTGCTAGTAGGCAGTCAGCGCCAT

+r001/1        TTAGATAAAGGATA*CTG
+r002         aaaAGATAA*GGATA
+r003       gcctaAGCTAA
+r004                     ATAGCT..............TCAGC
-r003                            ttagctTAGGC
-r001/2                                        CAGCGGCAT
```

The corresponding SAM format is:

```
@HD VN:1.5 SO:coordinate
@SQ SN:ref LN:45
r001  163 ref  7 30 8M2I4M1D3M = 37  39 TTAGATAAAGGATACTG *
r002    0 ref  9 30 3S6M1P1I4M *  0   0 AAAAGATAAGGATA    *
r003    0 ref  9 30 5S6M       *  0   0 GCCTAAGCTAA       * SA:Z:ref,29,-,6H5M,17,0;
r004    0 ref 16 30 6M14N5M    *  0   0 ATAGCTTCAGC       *
r003 2064 ref 29 17 6H5M       *  0   0 TAGGC             * SA:Z:ref,9,+,5S6M,30,1;
r001   83 ref 37 30 9M         =  7 -39 CAGCGGCAT         * NM:i:1
```

Figure 6. Example Output from Bowtie in SAM Format.
This figure shows an example of multiple read alignments with their corresponding output in SAM format. Column 4 contains the start position of the read. Column 6 contains the CIGAR score. Figure adapted from Li et al 2009.

2.3.2: Identify Quiescent Regions -- Memory Efficient Algorithm

Once the start and end positions for each read have been determined, the next step is to identify which regions of the *N. crassa* genome have reads mapped to them. One of the major challenges of this task is to create an algorithm that can do this in the memory footprint allowable on a personal computer. Consider that the four data sets have a combined total size of over 200 Gb and the largest of the data sets has over 215 million lines.

This algorithm is described in detail below, but first, let a range be defined as a set of start and end points representing some superset of one or more reads.

I. Initial set up:

- Create two arrays; startArray = { }, endArray = { }. These arrays will mark the start and end points of the currently mapped regions and store only the necessary values to optimize memory usage. Throughout this algorithm, transformations will be performed on the ranges stored in the arrays to ensure that at any point throughout the application, they represent the minimal set, M, of ranges required to describe the data set as a whole.

II. First line:

- Parse or extract the start and end positions for the first read.
- Push these values to the start and end arrays, respectively. startArray = { s_1 }, endArray = { e_1 }. M = { (s_1, e_1) }.

III. Subsequent lines:

- Parse the next line as a range representing the single read; R = (r_s, r_e).
- Find the index, i, where r_s falls in the startArray such that $r_s \leq$ startArray[i]. We can use this index, i, to ensure that the ranges stored in the arrays always stay in sorted order. Let A = (startArray[i - 1], endArray[i - 1]) = (a_s, a_e) and B = (startArray[i], endArray[i]) = (b_s, b_e).

- Determine which of the following scenarios apply for the new range, R, and perform the associated action. Each scenario is shown in Figure 7.
    - Scenario 1: Composition. Composition occurs when $R \subset A$. That is, $r_e \leq a_e$.
        - **Action:** Do nothing. This read is already covered by our arrays. That is, $R \subset M$.
    - Scenario 2: Right Extension. When $r_s \leq a_e$ and $r_e < b_s$, a right extension of A occurs.
        - **Action:** Transform A into (a_s, r_e).
    - Scenario 3: Connection. When R joins A and B, then A and B form a connection by R. That is, $r_s \leq a_e$ and $r_e \geq b_s$.
        - **Action:** If $r_e < b_e$, then transform A into (a_s, b_e) and delete B, adjusting trailing indices accordingly. If $r_e > b_e$, transform A into (a_s, r_e) and delete B. Ensure r_e does not extend into any trailing ranges of M. If it does, adjust ranges of M similarly.
    - Scenario 4: Insertion. If R is mutually exclusive of A and B, then an insertion occurs. This implies that $r_s > a_e$ and $r_e < b_s$.
        - **Action:** Insert R into M at index i, incrementing the existing indices $\geq$ i by 1.
    - Scenario 5: Left Extension. If $r_s > a_e$, $r_e \geq b_s$, and $r_e \leq b_e$, there is a left extension of B.
        - **Action:** Transform B into (r_s, b_e).

- Scenario 6: Double Extension. A double extension occurs when $r_s > a_e$ and $r_e > b_e$, which means B needs to be extended to the right and the left.
    - **Action:** Transform B into (r_s, r_e). Check to make sure r_e does not extend into any trailing ranges of M. If it does, adjust ranges of M accordingly.

IV. Identify quiescent regions:

- The quiescent regions can then be described as the set Q, where Q is the inverse image of M.

V. Extract desired results:

- Determine 35 largest quiescent regions from each data set for further analysis.

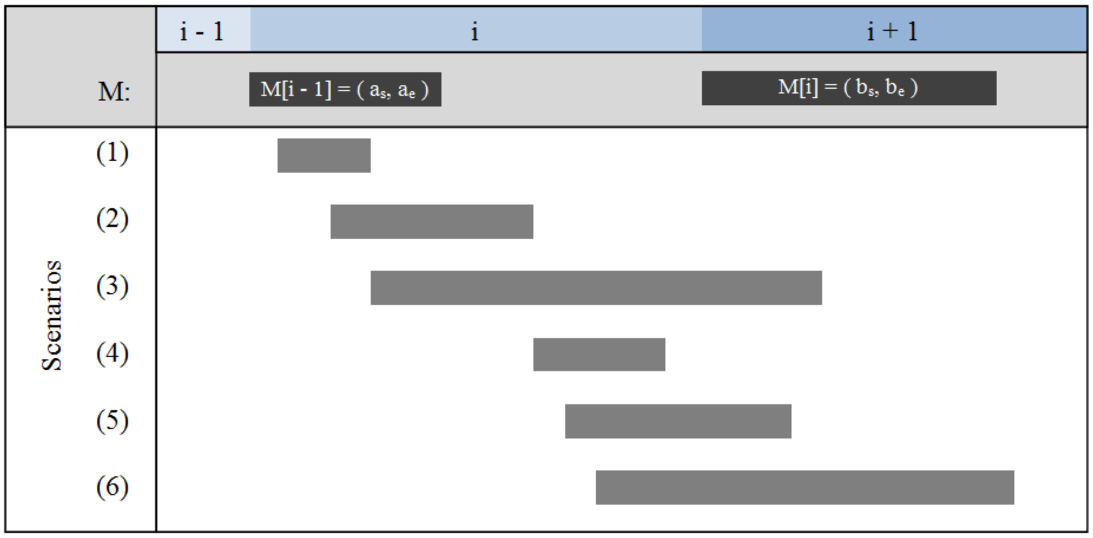


Figure 7. Scenarios for Memory Efficient Algorithm.
For any given index, i, one of six scenarios arises that the algorithm has to consider. Depending on the scenario, the algorithm has to decide how to alter the current set of ranges, M, in order to account for the newest read.

2.3.3: Identify Quiescent Regions -- Speed Efficient Algorithm

While the algorithm described above is memory efficient since it only keeps the minimum information necessary to identify the locations of the quiescent regions, it has a very slow execution speed and the logic inherent in the algorithm is highly complex and therefore prone to human error. Such slow speeds are most likely the result of the algorithm needing to find an index for each read to ensure the arrays stay in sorted order. This is a slow process since the size of the array grows as more information is processed. Also, every time a range is added or subtracted all the array indices must be adjusted up or down, respectively. This accounts for most of the CPU time that makes it so slow.

An additional algorithm was developed to try and fix these problems. The new method assigns each base in every contig or contiguous DNA segment a boolean value. The boolean value is 0 if the base has not been accounted for by a read and 1 if there has been at least one read aligned to that base. Each of these values are stored in an array (baseArray) can be used to determine where each of the quiescent regions are located. This algorithm is simple enough to describe using pseudocode (below).

```
# Determine Quiescent Regions

baseArray = [0 ,... ,0]

Foreach( read in file )
    For ( i from start pos to end pos )
        If ( baseArray[ i ] equals 0 )
            set baseArray[ i ] to 1

# Print Out Quiescent Regions

test = -1

Foreach( element in baseArray )

    If ( test does not equal -1 )
        If ( baseArray[ i ] equals 1 ) {
            set endQ to i - 1
            print out "startQ endQ"
            set test to -1
        }
        go to next element in baseArray


    If ( test equals -1 and baseArray[ i ] equals 0 )
        set startQ to i
        set test to 1
```

CHAPTER III

RESULTS

3.1: Chromosome Map

After all the quiescent regions had been identified, a chromosome map was created to see where they are located and how they are distributed throughout the genome. Figure 8 shows this map. The map was created by combining data sets 1005 and F201. A Perl script was written to determine which regions were distinct to each of the data sets and which were common between the two. Each of the colored regions represents a quiescent region, where red represents data set 1005 only, green represents data set F201 only, and blue represents quiescent regions that the two data sets share. We found that approximately 20.45% (average of 1005 and F201) of the genome is transcriptionally quiescent.

Since the large RNA data sets (1005 and F201) have theoretically identical genomes with the exception of a single point mutation, it is not surprising that we see lots of overlapping regions (shown in Figure 8 in blue). Regions that the data sets do not share appear to be smaller in size. These differences are possibly the result of errors during the read sequencing. In general, the quiescent regions do not appear to follow a general pattern, but they do seem to be slightly more common near the ends of the chromosomes.

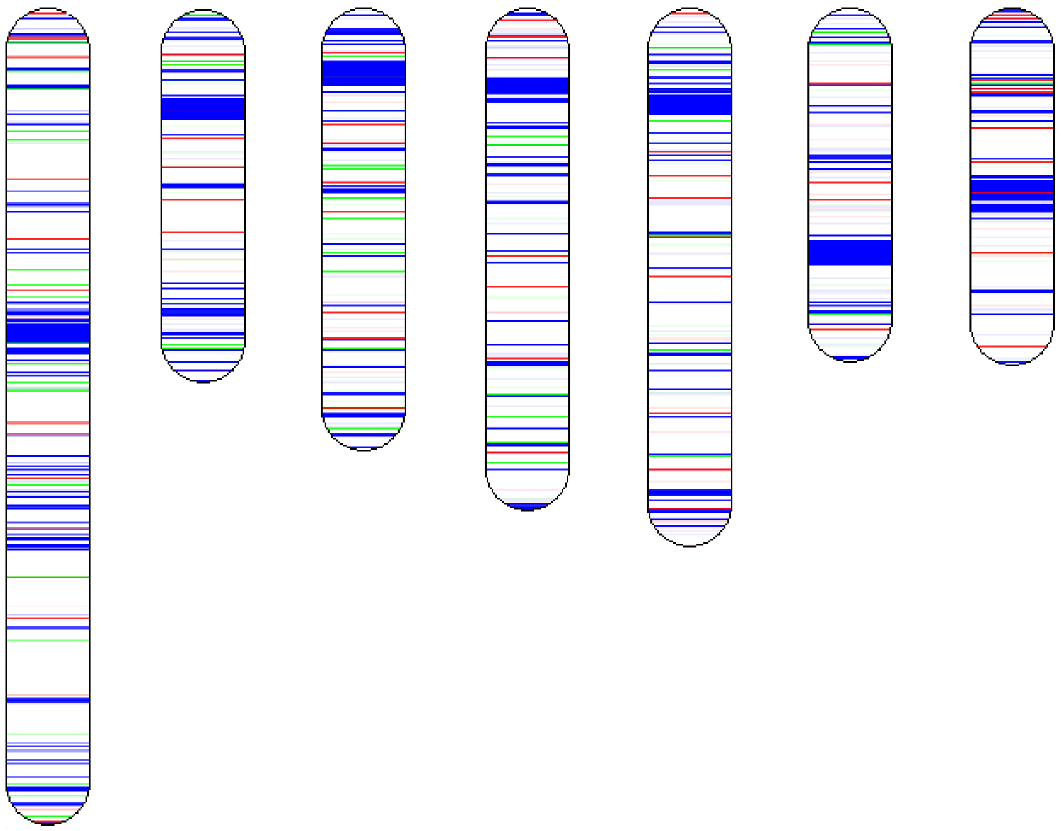


Figure 8. Chromosome Map for Combine Data Sets 1005 and F201.
The locations of the quiescent regions within the genome for the combined large RNA data sets (1005 and F201). Each colored region represents a quiescent region, where red represents data set 1005 only, green represents data set F201 only, and blue represents quiescent regions that the two data sets share. It should be noted that small regions will not show up due to the pixel limitations of the screen. Similarly, not all continuously solid colored regions are quiescent throughout. There may be small areas of transcription hidden within them. The program used for drawing the chromosome map was written in C++ by Dan Souther.

3.2: Analysis of Largest Quiescent Regions

BLAST (Basic local alignment search tool) compares a query sequence to a database of other previously published sequences to give insight about its function or relationship to other sequences (Altschul et al. 1990). For each data set, the largest 35 regions were BLASTed against NCBI's (National Center for Biotechnology Information) database to help speculate why these regions are transcriptionally quiescent.

A BLAST analysis reveals what is among the largest quiescent region in the *N. crassa* genome. The most common top hits were 'pol-like protein', ' hypothetical protein CHGG_08065', and 'hypothetical protein CHGG_10614'. These were the hits chosen to investigate further.

The 'pol-like protein' is related to a LINE-like retrotransposon in *N. crassa* called *Tad* (Cambareri et al. 1994). This transposon is active in *N. crassa* and capable of internuclear transfer. Among the most common classes, LINE-like transposable elements frequently appear in complex eukaryotes. *Tad* is one of the few transposable elements found to be active in the *N. crassa* genome (Cambareri et al. 1994).

Sequences that found hypothetical protein CHGG_08065 and hypothetical protein CHGG_10614 aligned to transposable elements found in other fungal and non-fungal genomes. In many cases these hypothetical proteins are thought to be reverse transcriptases responsible for the replication of retrotransposons and other mobile elements (Pérez-Alegre et al. 2005, DeMarco et al 2004, Kaneko et al. 2000). While they appear to be inactive in *N. crassa*, in some genomes these elements have high transcriptional activities (DeMarco et al. 2004).

By looking at the BLAST tables (Table 4 and 5), we can see that we have in fact indentified the centromeres on each chromosome as quiescent. Each of the centromere locations has been classified by Smith et al. 2011. In the BLAST tables for the large RNA data sets, values marked with a star (*) indicated regions that fall within the known centromere positions given by Table 3. Each chromosome's centromere was represented in the largest quiescent regions, except chromosome four and seven. It should be noted that chromosome four has the smallest sized centromere, which is possibly why it did not show up among the largest of the quiescent regions, but we still see it on the chromosome map.

Table 3. Centromere Locations.
Locations of each of the centromeres on each of the seven chromosomes found in *N. crassa* (Smith et al. 2011).

Chromosome	Total Length	Centromere Position	Centromere Size	Identified in Top Quiescent Regions
I	9,798,893	3,736,000-3,969,000	233,400	Yes
II	4,478,683	1,105,000-1,346,000	240,800	Yes
III	5,274,802	705,000-951,000	246,000	Yes
IV	6,000,761	894,000-1,068,000	174,000	No
V	643,246	932,000-1,209,000	276,800	Yes
VI	4,218,384	2,811,000-3,060,000	249,000	Yes
VII	4,255,303	1,801,000-2,089,000	287,400	No

Table 4. Top BLAST Results for Data Set 1005.
The top BLAST result from the 35 longest quiescent regions in the *N. crassa* genome according to data set 1005.

Contig	Start	End	Length	Top BLAST Result	E Value	Accession Number
12.6*	2843045	2913156	70112	pol-like protein	0	AAA21792.1
12.2*	1288997	1340422	51426	pol-like protein	9E-179	AAA21792.1
12.1	877947	924534	46588	hypothetical protein CHGG_10614	2E-110	XP_001228541.1
12.3	2192924	2236543	43620	polymerase	2E-170	AAK01619.1
12.10	60458	98020	37563	polymerase	3E-146	AAK01619.1
12.1	2295247	2331212	35966	hypothetical protein CHGG_10614	0	XP_001228541.1
12.2	3663764	3699364	35601	gag-pol polyprotein	5E-128	ACD86393.1
12.4	2345634	2378979	33346	hypothetical protein SMAC_09651	1E-102	XP_003342462.1
12.5	581326	614512	33187	hypothetical protein CHGG_00235	2E-82	XP_001219456.1
12.5*	1033170	1066033	32864	pol-like protein	9E-160	AAA21792.1
12.3*	743348	773864	30517	pol-like protein	0	AAA21792.1
12.2	2123504	2153803	30300	gag-pol polyprotein	2E-90	ACD86393.1
12.5	4442662	4472366	29705	hypothetical protein CHGG_10614	0	XP_001228541.1
12.2*	1094120	1123588	29469	hypothetical protein CHGG_10614	0	XP_001228541.1
12.1	6385143	6414537	29395	hypothetical protein CHGG_08065	6E-148	XP_001225721.1
12.1	9481922	9511253	29332	hypothetical protein CHGG_08065	4E-106	XP_001225721.1
12.6	2992687	3021711	29025	pol-like protein	0	AAA21792.1
12.5	5659952	5688415	28464	hypothetical protein CHGG_10614	1E-153	XP_001228541.1
12.6	2787934	2816281	28348	pol-like protein	0	AAA21792.1
12.2*	1164034	1192055	28022	pol-like protein	0	AAA21792.1
12.6	4181257	4209127	27871	hypothetical protein CHGG_10614	4E-164	XP_001228541.1
12.4	5218762	5246454	27693	hypothetical protein CHGG_10614	0	XP_001228541.1
12.4	3055027	3082226	27200	hypothetical protein CHGG_08065	0	XP_001225721.1
12.10	98071	124249	26179	hypothetical protein SMAC_09594	7E-100	XP_003343904.1
12.1*	3792820	3818921	26102	hypothetical protein, variant	0	ESA42110.1
12.4	3743730	3769455	25726	hypothetical protein CHGG_08792	3E-63	XP_001226719.1
12.3	350934	376626	25693	hypothetical protein SMAC_09651	6E-81	XP_003342462.1
12.3*	932157	957687	25531	pol-like protein	0	AAA21792.1
12.5	2594135	2619307	25173	polymerase	0	AAK01619.1
12.7	3678169	3703281	25113	hypothetical protein NEUTE2DRAFT_169926	5E-51	EGZ68151.1
12.2	111618	136717	25100	hypothetical protein CHGG_10614	1E-138	XP_001228541.1
12.7	3406221	3431178	24958	hypothetical protein SMAC_09594	2E-143	XP_003343904.1
12.1	4169861	4194781	24921	hypothetical protein CHGG_10614	9E-117	XP_001228541.1
12.4	1126474	1150552	24079	hypothetical protein CHGG_10614	0	XP_001228541.1
12.1	5920134	5944188	24055	hypothetical protein NEUTE2DRAFT_169926	2E-32	EGZ68151.1

Table 5. Top BLAST Results for Data Set F201.
The top BLAST result from the 35 longest quiescent regions in the N. crassa genome according to data set F201.

Contig	Start	End	Length	Top BLAST Result	E Value	Accession Number
12.6*	2835356	2866221	30866	pol-like protein	7E-163	AAA21792.1
12.10	80009	109358	29350	putative retrotransposon nucleocapsid protein	1.00E-114	EMR88181.1
12.4	1881482	1910463	28982	hypothetical protein CHGG_10614	0	XP_001228541.1
12.4	5218614	5246451	27838	hypothetical protein CHGG_10614	0	XP_001228541.1
12.4	5969351	5996397	27047	hypothetical protein CHGG_08288	2E-44	XP_001225944.1
12.7	2248012	2274723	26712	hypothetical protein SMAC_09594	5E-161	XP_003343904.1
12.5*	1021689	1047668	25980	hypothetical protein, variant	1.00E-168	ESA42110.1
12.4	3743737	3769558	25822	hypothetical protein CHGG_08792	3.00E-63	XP_001226719.1
12.1	6428798	6454471	25674	polymerase	7.00E-164	AAK01619.1
12.1	9481998	9507656	25659	hypothetical protein CHGG_08065	4.00E-106	XP_001225721.1
12.2	2118897	2143472	24576	hypothetical protein CHGG_02381	2.00E-64	XP_001228897.1
12.1	5920169	5944131	23963	hypothetical protein NEUTE2DRAFT_169926	2.00E-32	EGZ68151.1
12.5	5659960	5683728	23769	hypothetical protein CHGG_10614	9E-154	XP_001228541.1
12.1*	3792803	3816294	23492	hypothetical protein, variant	0	ESA42110.1
12.1	6385143	6408213	23071	hypothetical protein CHGG_08065	4.00E-148	XP_001225721.1
12.5	591727	614523	22797	hypothetical protein CHGG_00235	1.00E-82	XP_001219456.1
12.7	2142674	2165318	22645	hypothetical protein SMAC_09651	4E-90	XP_003342462.1
12.7	2216640	2238710	22071	hypothetical protein SMAC_09651	2E-116	XP_003342462.1
12.3	1252653	1274085	21433	hypothetical protein NEUTE1DRAFT_112574	1.00E-18	EGO54005.1
12.1	9777606	9798700	21095	hypothetical protein SMAC_09594	3.00E-90	XP_003343904.1
12.3	4074450	4095395	20946	hypothetical protein NCU10906	0.009	XP_001728026.1
12.2*	1259337	1280022	20686	hypothetical protein CHGG_08393	2.00E-56	XP_001226320.1
12.1	8227340	8247725	20386	hypothetical protein NCU04703	3.00E-36	EAA30998.3
12.1	904503	924640	20138	hypothetical protein CHGG_08065	2.00E-99	XP_001225721.1
12.3	4256738	4276599	19862	hypothetical protein NCU04703	6.00E-35	EAA30998.3
12.7	1324187	1344028	19842	hypothetical protein CHGG_08065	7.00E-104	XP_001225721.1
12.5	4452868	4472371	19504	hypothetical protein CHGG_10614	0	XP_001228541.1
12.3	1702684	1722126	19443	hypothetical protein CHGG_10614	0	XP_001228541.1
12.5	5897059	5916495	19437	hypothetical protein CHGG_10614	9.00E-169	XP_001228541.1
12.1	9736539	9755560	19022	gag-pol polyprotein	1.00E-70	ACD86393.1
12.1	4075895	4094838	18944	gag-pol polyprotein	2.00E-57	ACD86393.1
12.2	3904890	3923740	18851	hypothetical protein SMAC_00575	1.00E-41	XP_003352028.1
12.7	3678160	3696954	18795	hypothetical protein NEUTE2DRAFT_169926	4.00E-51	EGZ68151.1
12.1	6299389	6318162	18774	hypothetical protein NEUTE1DRAFT_98304	1.00E-41	EGO61164.1
12.4	4641201	4659543	18343	hypothetical protein NEUTE1DRAFT_112574	3.00E-47	EGO54005.1

Table 6. Top BLAST Results for Data Set SRR751454.
The top BLAST result from the 35 longest quiescent regions in the *N. crassa* genome according to data set SRR751454.

Contig	Start	End	Length	Top BLAST Result	E Value	Accession Number
12.7	1117525	1120940	3416	hypothetical protein NC02593	0.00008	EAA35760.2
12.1	4187715	4190235	2521	hypothetical protein CHGG_08065	5E-11	XP_001225721.1
12.7	1271256	1273678	2423	No significant similarity found		
12.7	906720	909056	2337	hypothetical protein NCU16630	2E-16	ESA43159.1
12.5	192452	194581	2130	No significant similarity found		
12.1	7424339	7426439	2101	No significant similarity found		
12.1	922438	924505	2068	No significant similarity found		
12.1	3698085	3700141	2057	hypothetical protein NCU05738	0.0002	XP_962894.1
12.4	1929397	1931453	2057	No significant similarity found		
12.3	2290744	2292755	2012	hypothetical protein CHGG_04034	4E-25	XP_001223248.1
12.1	8285026	8286976	1951	hypothetical protein CHGG_04032	0.0006	XP_001223246.1
12.3	1703106	1704988	1883	No significant similarity found		
12.4	2026841	2028719	1879	Uncharacterized protein FFUJ 02954	3E-31	CCT65963.1
12.3	434434	436308	1875	No significant similarity found		
12.5	5709814	5711657	1844	Uncharacterized protein FFUJ 02954	5E-28	CCT65963.1
12.7	1281123	1282924	1802	No significant similarity found		
12.4	440594	442369	1776	No significant similarity found		
12.5	178415	180140	1726	No significant similarity found		
12.1	3482653	3484370	1718	hypothetical protein CHGG_10614	1E-12	XP_001228541.1
12.2	2906890	2908598	1709	polymerase	1E-32	AAK01619.1
12.6	3532761	3534463	1703	No significant similarity found		
12.1	5539518	5541219	1702	No significant similarity found		
12.7	3678401	3680101	1701	No significant similarity found		
12.4	4253084	4254763	1680	No significant similarity found		
12.5	2622180	2623859	1680	No significant similarity found		
12.6	1988691	1990317	1627	predicted protein	1E-25	XP_001219297.1
12.5	4245440	4247060	1621	No significant similarity found		
12.6	3238617	3240230	1614	hypothetical protein CHGG 00074	0.0004	XP 001219295.1
12.1	5956005	5957600	1596	hypothetical protein FOXB_12797	1E-15	EGU76692.1
12.5	5234556	5236150	1595	No significant similarity found		
12.6	2385076	2386660	1585	hypothetical protein FOC4 g10013526	5E-11	EMT63306.1
12.7	1335604	1337177	1574	hypothetical protein CHGG 10614	7E-16	XP 001228541.1
12.1	3747988	3749553	1566	hypothetical protein FOXB_04429	3E-12	EGU85056.1
12.1	890965	892523	1559	hypothetical protein CHGG_08065	8E-16	XP_001225721.1
12.1	3063531	3065087	1557	hypothetical protein FOC4_g10004840	2E-19	EMT68755.1

Table 7. Top BLAST Results for Data Set SRR755946.
The top BLAST result from the 35 longest quiescent regions in the *N. crassa* genome according to data set SRR755946

Contig	Start	End	Length	Top BLAST Result	E Value	Accession Number
12.5	192381	194584	2204	No significant similarity found		
12.1	3698115	3700139	2025	hypothetical protein NCU05738	2.00E-04	XP_962894.1
12.5	2204995	2206976	1982	hypothetical protein FOXB_16638	0.14	EGU72852.1
12.4	446096	447980	1885	No significant similarity found		
12.5	6109704	6111539	1836	hypothetical protein PDIG_52550	0.031	EKV11112.1
12.5	5232425	5234253	1829	hypothetical protein CHGG_10614	9.00E-10	XP_001228541.1
12.7	1271893	1273679	1787	No significant similarity found		
12.7	2153900	2155628	1729	retrovirus polyprotein, putative	1E-16	XP_002145610.1
12.7	2204108	2205827	1720	No significant similarity found		
12.1	5395490	5397196	1707	predicted protein	4.00E-42	XP_001219297.1
12.7	3700796	3702409	1614	No significant similarity found		
12.7	1119373	1120954	1582	No significant similarity found		
12.4	4482424	4483910	1487	No significant similarity found		
12.1	2384366	2385775	1410	No significant similarity found		
12.6	2827499	2828904	1406	No significant similarity found		
12.3	2882047	2883438	1392	hypothetical protein CHGG_03501	4.00E-08	XP_001230017.1
12.6	1905196	1906586	1391	hypothetical protein CHGG_00074	7.00E-07	XP_001219295.1
12.4	4277774	4279158	1385	No significant similarity found		
12.7	1269731	1271115	1385	No significant similarity found		
12.1	72579	73956	1378	No significant similarity found		
12.7	1017788	1019158	1371	hypothetical protein SMAC_09594	4.00E-67	XP_003343904.1
12.7	3679627	3680991	1365	No significant similarity found		
12.4	1910907	1912268	1362	No significant similarity found		
12.7	1281654	1283003	1350	No significant similarity found		
12.5	178860	180191	1332	No significant similarity found		
12.5	2618058	2619383	1326	No significant similarity found		
12.4	4769367	4770677	1311	hypothetical protein CHGG_00074	3.00E-11	XP_001219295.1
12.5	587701	589008	1308	No significant similarity found		
12.4	4254496	4255797	1302	No significant similarity found		
12.1	5956094	5957394	1301	hypothetical protein FOXB_12797	6.00E-16	EGU76692.1
12.3	1703527	1704821	1295	No significant similarity found		
12.6	3411349	3412589	1241	hypothetical protein NEUTE2DRAFT_114192	2.00E-27	EGZ70959.1
12.2	391164	392399	1236	No significant similarity found		
12.6	2819151	2820382	1232	hypothetical protein NCU04255	0.26	XP_961246.1
12.4	451143	452369	1227	No significant similarity found		

3.3: Quiescent Region Summary Statistics

Summary statistics (see Table 8) about the size of the quiescent regions for each of the data sets were obtained using R. 75% of the quiescent region lengths fall below 130 given by the third quartile. Hence, most of the quiescent regions are very small, so the data on the size of the quiescent regions is highly right skewed. A similar situation is true for the small RNA data sets.

Table 8. Summary Statistics: Size of Quiescent Regions.

Summary Statistics	**Data Set**			
	1005	**F201**	**SRR751454**	**SRR755946**
Min	1	1	1	1
Q1	20	20	10	8
Median	52	53	25	21
Q3	131	135	55	46
Max	70110	46520	3416	2204
Mean	366.1	344.4	46.24	37.91
SD	1959.793	1589.285	72.590	57.800

A population proportion test was performed to check if there are differences in GC content in the quiescent regions (p_1) is significantly different than what is observed in the whole genome (p). Once the locations of the quiescent regions were determined, a Perl script was written to calculate the A/T/G/C proportions. This test can be done using a single population proportion test with the following null and alternative hypotheses:

$$H_0: p_1 = p$$

$$H_A: p_1 \neq p$$

with test statistic,

$$z = \frac{\widehat{p_1} - p}{\sqrt{\frac{p(1-p)}{n}}}$$

Four tests were performed at significance level α = 0.05. For all four tests, we found a significant test statistic to reject the null hypothesis that the quiescent regions have the same GC content as the whole genome (P < 0.0001 for all tests). Results for these tests are summarized in Table 9.

Table 9. Summary of Results from Hypothesis Test for Differences in GC Content. This table compares GC and AT content in all quiescent regions from each data set to the whole genome proportions. The resulting p-values make it clear that all data sets had significantly different proportions of AT and GC than the whole genome. Significance level used for these tests was α = 0.05.

Summary Statistics	**Data Sets**				
	Whole Genome (p)	1005 ($\hat{p}_1$)	F201 ($\hat{p}_1$)	SRR751454 ($\hat{p}_1$)	SRR755946 ($\hat{p}_1$)
% GC	48.25%	33.12%	33.13%	45.19%	44.67%
% AT	51.75%	66.88%	66.87%	54.81%	55.33%
p-Value for GC Differences		<.0001	<.0001	<.0001	<.0001
p-Value for AT Differences		<.0001	<.0001	<.0001	<.0001
Critical Values, $z_{.025}$		±1.96	±1.96	±1.96	±1.96

3.4: Analysis of Algorithms

Two scripts for determining the locations of the quiescent regions were written. One was designed to be memory efficient, since we are dealing with whole genome sized data, while the other one was designed to be simpler to understand and more time efficient.

Both of the scripts were profiled to reveal exactly where their strengths and weakness were. Two memory tests were performed, one with a medium sized data set and another with a small data set. The first test used chromosome 7 from data set 1005, which contained approximately 18 million lines. During this test, the memory efficient algorithm used 9,962 K of memory, whereas the speed efficient algorithm used almost 15X that (152,980 K). The second test used a much smaller data set from contig 9 of data set SRR751454, which contained approximately one million lines. The memory efficient algorithm peaked at 6,652 K executing 920,262,957 statements, whereas the speed efficient algorithm peaked at 10,628 K and only executed 9,990,142 statements. We can see that as the size of the file gets smaller the difference between the memory required decreases.

We also found that this memory efficiency came with a high cost of speed. Speed efficiency was determined using a Perl profiling module called NYTProf. This module gives the total time it takes for a Perl script to run, and it also breaks down the time spent in each subroutine of the code. Profiling with the data from chromosome 7 did not finish due to hard drive limitations. This module stores information about the algorithm's profile on a per-line basis, which requires a lot of space.

Table 10. Memory Efficient Algorithm Times by Subroutine.

Calls	**Exclusive Time**	**Subroutine**
1383871	633s	findIndex
1383873	1.93s	readline
5844	1310ms	insertNewLine
10150	78.0ms	checkOverlap
1	15.6ms	writeOutput

Using the smaller data set (contig 9 from data set SRR751454) the memory efficient program took 30.26 minutes to run, whereas the speed efficient algorithm only took 50.5 seconds. This time difference increases as the data size set gets larger. In the speed efficient algorithm, almost all of the time was accounted for in reading in and writing out lines. This was not case in the memory efficient algorithm. Table 10 shows a breakdown of each of the subroutine calls and the amount of time spent in each one for the smaller data set (contig 9 from data set SRR751454). Now, we can see that finding the index for each of the reads is what slowed the algorithm down. Also, when we had to insert a new line, each of the indices for the currently stored set needed to all be adjusted accordingly to make sure the arrays stayed in sorted order.

Table 11. Time and Memory Efficiency of Algorithms.
Description of time and memory efficiency in big O notation of the algorithms that were written to determine quiescent regions.

Algorithm	Time	Memory
Memory Efficient	O(sizeArray * numAdjustments) = O(n^3)	O(min(numRanges to represent non-QRegions))
Time Efficient	O(numReads * sizeRead) = O(n^2)	O(lengthContig)

The time and memory efficiency of both algorithms is described in Table 11. Both algorithms were polynomial in terms of time. However, the time efficient algorithm is more efficient since O(n^2) < O(n^3). The memory efficient algorithm is more memory efficient since min(numRanges to represent non-QRegions) << lengthContig.

CHAPTER IV

DISCUSSION

4.1: Hypotheses About Transcriptional Quiescence

This section is intended to discuss possible hypotheses about why these regions are transcriptionally quiescent.

4.1.1: Evolutionary Origins of the Quiescent Regions

In general, many of the largest quiescent regions contained transposable elements or relics of transposable elements many times within their top BLAST results. This trend has been observed in several other works (Lewis et al. 2009, Selker et al. 2003, Rountree 2010). In *N. crassa*, RIP acts as a genomic defense mechanism by identifying duplicated sequences and introducing mutations within them. Locations where mutations have occurred are highly susceptible to silencing via methylation. Therefore, it is not surprising that we see mutated transposable elements within the quiescent regions of the genome (Galagan and Selker 2004).

Centromeric regions, like other transcriptionally quiescent regions, have been found to be largely comprised of duplicated transposable elements that have been heavily mutated by RIP (Smith et al. 2011). Additionally, some of these centromeric regions may contain predicted genes (Smith et al. 2011). This would explain why we do not see quiescent regions the entire size of the centromeres, and in some cases (chromosomes four and seven) they do not even show up as one of the largest regions.

4.1.2: GC/AT Content

A process called RIP (repeat induced point mutation) acts on duplicated sequences within DNA, such as transposable elements that insert themselves in the genome at multiple locations, by inducing C-to-T and G-to-A mutations (Kelly & Aramayo 2007, Freitag et al 2002, Hood 2005). Such mutations make these sequences highly susceptible to methylation, which causes silencing (Galagan and Selker 2004). This process protects the native genome against foreign invaders (Singer et al. 1995).

One hypothesis is that these regions are quiescent because RIP changes GC/AT content. We tested this hypothesis statistically to see if GC (or AT) content in the quiescent regions (p_1) is significantly different than what is observed in the whole genome (p). We performed this test at significance level $\alpha = 0.05$ for each of the data sets. There was a significant test statistic to reject the null hypothesis that the quiescent regions have the same GC content as the whole genome ($P < 0.0001$ for all tests).

Therefore, since the quiescent regions have statistically significant differences in GC/AT content than the whole genome, it is reasonable to hypothesize that the quiescent regions are highly mutated by RIP in *N. crassa*. This suggests that these regions are relics of transposable elements that have been heavily mutated by RIP and are now inactive.

4.2: Memory Efficiency vs. Speed Efficiency

Two algorithms were written to determine which regions of the *N. crassa* genome are quiescent. There are pros and cons to each one. The first one was written strictly to be memory efficient, which I had assumed would be a major issue given that we are dealing with genome sized data. Unfortunately, the memory efficiency came at the cost of speed. Running this script for all four data sets took two computers almost two weeks to complete the analysis. Such time requirements would decrease the burden of repeating the analysis with additional data sets. This time inefficiency was found to be caused by a few different things. Most notably, for each read in the input file the algorithm had to search for the index where that read belonged so that it could always remain in sorted order. Additionally, each time an element was added or subtracted from the array all of the indices following it needed to be adjusted.

A simpler algorithm was written to solve the time requirement issue. This that was less memory efficient, but faster and less prone to logic errors in writing the scripts. This algorithm was so time efficient that most of the total time was spent reading in and writing out lines, as opposed to processing the information. While this algorithm had a larger memory footprint, it was not so large as to prevent the program from running.

Therefore, when deciding which algorithm to use it is really only important to consider the hardware being used. I would recommend using the memory efficient algorithm when memory is an issue. Consider that the *Neurospora* genome contains 40 million bases. To run the whole genome at once would require approximately 5.35 GB of memory. This is possible on a relatively modern computer. However, this may not be realistic with the human genome, which contains over 3 billion bases. This translates to approximately 400 GB of memory.

One way to get around possible memory issues would be to split the genome into smaller pieces before performing the analysis. A logical way to do this would be to divide the data set into each of the contigs. This would make using a typical computer more plausible if we could decrease the memory footprint sufficiently. Another possibility would be to alter the memory efficient algorithm improve its speed. As it is now, there are several places where this could happen. For instance, simply changing how it checks for overlapping data could reduce its speed down to O(n^2).

CHAPTER V

CONCLUSIONS

5.1: Final Thoughts

Identifying the quiescent regions of the genome is the first step towards being able to distinguish theoretical aRNA from other RNA molecules. Since there are no other RNA transcripts present in these regions, if we see RNA molecules when we unpair them for MSUD, it's likely that they will in fact be aRNA. This is important for the study of MSUD proteins and possibly those proteins involved in meiosis that are required for identifying chromosomes as homologous pairs.

Based on an analysis of the quiescent regions, we can speculate that these regions appear to be relics of transposable elements that are highly mutated by RIP. We also found that there was a higher AT than GC content in the quiescent regions than what was present in the whole genome. This also suggests that RIP is at work since it causes C-to-T and G-to-A mutations.

5.2: Future Research

Based on results obtained from the current work, a natural extension of this is to ask how similar the quiescent regions are in terms of structure and methylation patterns. Preliminary work by Jamieson et al. (2013) on histone H3 lysine methylation

(H3K27me3) shows that H3K27me3 covers 6.8% of the *Neurospora* genome. All of these regions were found to be transcriptionally silent (Jamieson et al. 2013). Since we have found all of the transcriptionally quiescent regions, it would be appropriate to see if they all follow a similar pattern.

Additionally, theoretically the two large RNA data sets differ by only one single point mutation that causes the process of RIP to be active. It would be interesting to compare gene expression levels throughout the genome. Regions that differ most may be involved in 'ripping' duplicated regions of the genome.

REFERENCES

Alexander, W. G., Raju, N. B., Xiao, H., Hammond, T. M., Perdue, T. D., Metzenberg, R. L., ... & Shiu, P. K. (2008). DCL-1 colocalizes with other components of the MSUD machinery and is required for silencing. *Fungal Genetics and Biology*, *45*(5), 719-727.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, *215*(3), 403-410.

Borkovich K.A., Alex L.A., Yarden O. et al. (2004). Lessons from the genome sequence of *Neurospora crassa*: tracing the path from genomic blueprint to multicellular organism. *Microbiol Mol Biol Rev, 68*: 1Y108.

Cambareri, E. B., Singer, M. J., & Selker, E. U. (1991). Recurrence of repeat-induced point mutation (RIP) in *Neurospora crassa*. *Genetics, 127*(4), 699-710.

Cambareri, E. B., Helber, J., & Kinsey, J. A. (1994). Tadl-1, an active LINE-like element of *Neurospora crassa*. *Molecular and General Genetics MGG*, *242*(6), 658-665.

Catalanotto C., Pallotta M., ReFalo P. et al. (2004). Redundancy of the two dicer genes in transgene-induced posttranscriptional gene silencing in *Neurospora crassa*. *Mol Cell Biol*, *24*:2536Y2545.

Davis, R. H. (2000). *Neurospora: contributions of a model organism* (p. 333). New York: Oxford University Press.

DeMarco, R., Kowaltowski, A. T., Machado, A. A., Soares, M. B., Gargioni, C., Kawano, T., ... & Verjovski-Almeida, S. (2004). Saci-1,-2, and-3 and Perere, four novel retrotransposons with high transcriptional activities from the human parasite *Schistosoma mansoni*. *Journal of virology*, *78*(6), 2967-2978.

Feschotte, C., Jiang, N., & Wessler, S. R. (2002). Plant transposable elements: where genetics meets genomics. *Nature Reviews Genetics*, *3*(5), 329-341.

Foss H.M., Roberts C.J., Claeys K.M., Selker, E.U. (1993). Abnormal chromosome behavior in *Neurospora* mutants defective in DNA methylation. *Science,262*, 1737–1741.

Freitag, M., Williams, R. L., Kothe, G. O., & Selker, E. U. (2002). A cytosine methyltransferase homologue is essential for repeat-induced point mutation in *Neurospora crassa*. *Proceedings of the National Academy of Sciences*, *99*(13), 8802-8807.

Galagan J.E., Calvo S.E., Borkovich K.A. et al. (2003). The genome sequence of the filamentous fungus *Neurospora crassa*. *Nature*, *422*: 859Y868.

Galagan, J. E., & Selker, E. U. (2004). RIP: the evolutionary cost of genome defense. *Trends in Genetics*, *20*(9), 417-423.

Hammond, T. M., Xiao, H., Boone, E. C., Decker, L. M., Lee, S. A., Perdue, T. D., ... & Shiu, P. K. (2013). Novel proteins required for meiotic silencing by unpaired DNA and siRNA generation in *Neurospora crassa*. *Genetics*, *194*(1), 91-100.

Hammond, T. M., Xiao, H., Boone, E. C., Perdue, T. D., Pukkila, P. J., & Shiu, P. K. (2011). SAD-3, a putative helicase required for meiotic silencing by unpaired DNA, interacts with other components of the silencing machinery. *G3: Genes, Genomes, Genetics*, *1*(5), 369-376.

Hood, M. E., Katawczik, M., & Giraud, T. (2005). Repeat-induced point mutation and the population structure of transposable elements in *Microbotryum violaceum*. *Genetics*, *170*(3), 1081-1089.

Jamieson, K., Rountree, M. R., Lewis, Z. A., Stajich, J. E., & Selker, E. U. (2013). Regional control of histone H3 lysine 27 methylation in *Neurospora*. *Proceedings of the National Academy of Sciences*, *110*(15), 6027-6032.

Kaneko, I., Tanaka, A., & Tsuge, T. (2000). REAL, an LTR retrotransposon from the plant pathogenic fungus *Alternaria alternata*. *Molecular and General Genetics MGG*, *263*(4), 625-634.

Kelly, W. G., & Aramayo, R. (2007). Meiotic silencing and the epigenetics of sex. *Chromosome research*, *15*(5), 633-651.

Langmead, B., Trapnell, C., Pop, M., & Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, *10*(3), R25.

Langmead, B., & Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature methods*, *9*(4), 357-359.

Lee, D. W., Pratt, R. J., McLaughlin, M., & Aramayo, R. (2003). An argonaute-like protein is required for meiotic silencing. *Genetics*, *164*(2), 821.

Lee, D. W., Millimaki, R., & Aramayo, R. (2010). QIP, a component of the vegetative RNA silencing pathway, is essential for meiosis and suppresses meiotic silencing in *Neurospora crassa*. *Genetics*, *186*(1), 127-133.

Lewis, Z. A., Honda, S., Khlafallah, T. K., Jeffress, J. K., Freitag, M., Mohn, F., ... & Selker, E. U. (2009). Relics of repeat-induced point mutation direct heterochromatin formation in Neurospora crassa. *Genome research*, *19*(3), 427-437.

Li, H., Ruan, J., & Durbin, R. (2008a). Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome research*, *18*(11), 1851-1858.

Li, H., Handsaker, B., Wysoker, A., Fennel, T., Ruan, J., Homer, N., ...& Durbin, R. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics, 25*(16), 2078-2079.

Li, R., Li, Y., Kristiansen, K., & Wang, J. (2008b). SOAP: short oligonucleotide alignment program. *Bioinformatics*, *24*(5), 713-714.

Lister, R., Pelizzola, M., Dowen, R. H., Hawkins, R. D., Hon, G., Tonti-Filippini, J., ... & Ecker, J. R. (2009). Human DNA methylomes at base resolution show widespread epigenomic differences. *Nature*, *462*(7271), 315-322.

Norris, Jeffery. (2013, February 13). "Gene Invaders Are Stymied by a Cell's Genome Defense." *University of California, San Francisco*. N.p., n.d.

Pérez-Alegre, M., Dubus, A., & Fernández, E. (2005). REM1, a new type of long terminal repeat retrotransposon in *Chlamydomonas reinhardtii*. *Molecular and cellular biology*, *25*(23), 10628-10638.

Pratt R.J., Lee D.W., Aramayo R. (2004). DNA methylation affects meiotic trans-sensing, not meiotic silencing, in *Neurospora*. *Genetics*, *168,*1925-1935.

Rountree, M. R., & Selker, E. U. (2010). DNA methylation and the formation of heterochromatin in *Neurospora crassa*. *Heredity*, *105*(1), 38-44.

SAM Format Specification Working Group. The SAM format specification (v1. 4-r985).

Selker, E. U., Tountas, N. A., Cross, S. H., Margolin, B. S., Murphy, J. G., Bird, A. P., & Freitag, M. (2003). The methylated component of the *Neurospora crassa* genome. *Nature*, *422*(6934), 893-897.

Shiu P.K., Raju B.N., Zickler D., Metzenberg R. (2001). Meiotic silencing by unpaired DNA. *Cell, 107*: 905Y916.

Shiu, P. K., & Metzenberg, R. L. (2002). Meiotic silencing by unpaired DNA: properties, regulation and suppression. *Genetics*, *161*(4), 1483-1495.

Shiu, P. K., Zickler, D., Raju, N. B., Ruprich-Robert, G., & Metzenberg, R. L. (2006). SAD-2 is required for meiotic silencing by unpaired DNA and perinuclear localization of SAD-1 RNA-directed RNA polymerase. *Proceedings of the National Academy of Sciences of the United States of America*, *103*(7), 2243-2248.

Singer, M. J., Marcotte, B. A., & Selker, E. U. (1995). DNA methylation associated with repeat-induced point mutation in *Neurospora crassa*. Molecular and cellular biology, 15(10), 5586-5597.

Slotkin, R. K., & Martienssen, R. (2007). Transposable elements and the epigenetic regulation of the genome. *Nature Reviews Genetics*, *8*(4), 272-285.

Smith, K. M., Phatale, P. A., Sullivan, C. M., Pomraning, K. R., & Freitag, M. (2011). Heterochromatin is required for normal distribution of *Neurospora crassa* CenH3. *Molecular and cellular biology*, *31*(12), 2528-2542.

Wang Z., Gerstein M., Snyder M. (2009). RNA-Seq: a revolutionary tool for transcriptomics. *Nature Reviews Genetics*, *10*,57–63.

Xiao, H., Alexander, W. G., Hammond, T. M., Boone, E. C., Perdue, T. D., Pukkila, P. J., & Shiu, P. K. (2010). QIP, a protein that converts duplex siRNA into single strands, is required for meiotic silencing by unpaired DNA. *Genetics*,*186*(1), 119-126.

APPENDIX

PERL SCRIPTS

Convert CIGAR Scores to Lengths

```
#!/perl/bin/perl

use strict;

# CONVERT CIGAR SCORES
my $lines = '';
my @columns = '';

open(IN, '1005_to_combo_9MAY13_a_with_after_local_a.sam') or die "Could
not open file\n";
open(OUT,
">1005_to_combo_9MAY13_a_with_after_local_a_convertedScores.sam");

while($lines = <IN>) {
    chomp $lines;
    if ( $lines !~ m\^@\){
        @columns = split(/\t/, $lines);
        if ( $columns[3] !~ m\^0$\){
            my $cigar = "$columns[5]";
            my @nums = '';
            my @chars = '';

            # parse $cigar
            my @nums = $cigar =~ /(\d+)/g; #extract integers
            my @chars = $cigar =~ /(\D+)/g; #extract non-digit chars

            # find indices for each cigar operation
            my @M = grep { $chars[$_] =~ /M/ } 0..$#chars;
            #locate index of M in array @chars
            my @I = grep { $chars[$_] =~ /I/ } 0..$#chars;
            my @D = grep { $chars[$_] =~ /D/ } 0..$#chars;
            my @N = grep { $chars[$_] =~ /N/ } 0..$#chars;
            my @S = grep { $chars[$_] =~ /S/ } 0..$#chars;
            my @H = grep { $chars[$_] =~ /H/ } 0..$#chars;
            my @P = grep { $chars[$_] =~ /P/ } 0..$#chars;
            my @X = grep { $chars[$_] =~ /X/ } 0..$#chars;


            # get length for each cigar operation

            my $M_length = 0;
            my $I_length = 0;
            my $D_length = 0;
            my $N_length = 0;
            my $S_length = 0;
            my $H_length = 0;
            my $P_length = 0;
            my $X_length = 0;

            for (@M) {
                $M_length += $nums[$_]
            }

            for (@I) {
                $I_length += $nums[$_]
            }

            for (@D) {
                $D_length += $nums[$_]
            }

            for (@N) {
                $N_length += $nums[$_]
            }

            for (@S) {
                $S_length += $nums[$_]
            }

            for (@H) {
                $H_length += $nums[$_]
            }

            for (@P) {
                $P_length += $nums[$_]
            }

            for (@X) {
                $X_length += $nums[$_]
            }

            # determine total length
            my $ref_len;
            my $end_pos;
            my $start_pos;

            $start_pos = $columns[3];
            $ref_len = $M_length
                      + 0*$I_length
                        # I is an insertion into the ref seq.
                      + $D_length
                      + $N_length
                      + 0*$S_length
                        # dont include soft clipping
                      + 0*$H_length
                        # dont include hard clipping.
```

```
                        #There shouldn't be any anyway.
                      + $P_length
                      + $X_length;

            $end_pos = $start_pos + $ref_len - 1;
            print OUT "$columns[2]\t$start_pos\t$end_pos\n";
        }
    }
}

close IN;
close OUT;
```

Sort By Contig

```
#!/perl/bin/perl

use strict;

my $lines;
my @columns;

for(my $i = 1; $i < 21; $i = $i + 1) {

    open(IN, '1005_to_combo_9MAY13_a_with_after_local_a_convertedScores
    .sam') or die "Could not open file\n";
    open(OUT, ">Sorted\\1005_to_combo_sorted_contig$i.txt");

    while($lines = <IN>) {
        chomp $lines;
        @columns = split(/\t/, $lines);
        if ($columns[0] =~ m/Supercontig_12.$i$/) {
            print OUT "$columns[0]\t$columns[1]\t$columns[2]\n";
        }
    }
}
```

Speed Efficient Algorithm

```
#!/perl/bin/perl

use strict;

my @bases;
my $contig;
my $dataSetName = "1005";

# ==========================================================
# -==================  Main Method   =====================-
# ==========================================================
my @files = <input/*>; #get all the items in the directory
foreach my $file (@files) {

    # only pay attention to files
    if (-f $file) {
        # initialize variables
        open(IN, $file) or die "Could not open file\n";
        my $lineNum = 0;
        print "\n\nProcessing file: $file\n";

        # for each line in the file
        while (my $line = <IN>) {
            $lineNum++;

            chomp $line;
            my @columns = split(/\t/, $line);

            # cache the contig name for later
            if ($lineNum == 1) {
                $contig = $columns[0];
            }

            my $start = $columns[1];
            my $end = $columns[2];

            for (my $i = $start; $i <= $end; $i++) {

                # count expression
                $bases[$i] = ($bases[$i]) ? $bases[$i] + 1 : 1;

            }
        }

        close IN;

        # Write results
        print "\nWriting Output... \n";
        $bases[0] = 1;
        # set index 0 as true so we don't
        # pick it up as a Q-region in the output
        outputQuiescentRegions();
        print "\nDone.\n";

        # Reset
        @bases = ();
    }
}

# ==========================================================
# -==================  SubRoutines   =====================-
# ==========================================================
sub outputQuiescentRegions() {

    open(OUT, ">qRegions/$dataSetName"."_to_combo_quiescent_regions_
    $contig.txt");

    # Identify Quiescent Regions
    my %contig_lengths = ("Supercontig_12.1", 9798893,
                          "Supercontig_12.2", 4478683,
                          "Supercontig_12.3", 5274802,
                          "Supercontig_12.4", 6000761,
                          "Supercontig_12.5", 6436246,
                          "Supercontig_12.6", 4218384,
                          "Supercontig_12.7", 4255303,
                          "Supercontig_12.8", 192308,
                          "Supercontig_12.9", 142473,
                          "Supercontig_12.10", 125404,
                          "Supercontig_12.11", 31696,
                          "Supercontig_12.12", 19714,
                          "Supercontig_12.13", 13515,
                          "Supercontig_12.14", 11565,
                          "Supercontig_12.15", 9397,
                          "Supercontig_12.16", 8983,
                          "Supercontig_12.17", 6701,
                          "Supercontig_12.18", 6309,
                          "Supercontig_12.19", 4755,
                          "Supercontig_12.20", 1646);

    my $start = -1;
    my $end = -1;

    # =======================================================
    # loop through all but the last index to find Q-regions
    # =======================================================
    for (my $i = 0; $i <= $#bases - 1; $i++) {

        #check to see if we're in a Q-region or not
        if ($start != -1) {
            # check for end of Q-region
            if ($bases[$i]) {

                # print the region
                $end = $i - 1;
                #print "print region: ($start, $end)\n";
                print OUT "$contig\t$start\t$end\n";

                #reset
                $start = -1;
                $end = -1;
            }

            next;
        }

        # check for the start of a Q-region
        if (!$bases[$i]) {
            $start = $i;
        }
    }

    # =======================================================
    # Handle last index
    # =======================================================
    if ($start == -1) { # we're not in a Q-region already
```

```
        if ($bases[$#bases]) {

        }
        else {
            # we have the start of a Q-region at the last index
            # SHOULDN'T HAPPEN
            print "!!! Suspicious start of Qregion at end of \@bases:
            ($start, $end)\n";
            print OUT "$contig\t$#bases\t$contig_lengths{$contig}\n";
        }
    }
    else { # We are in a Q-region.
           # Determine if we need to include the last index.

        if ($bases[$#bases]) {

            # print the current region
            $end = $#bases - 1;

            print OUT "$contig\t$start\t$end\n";

            #check to see if we are not at the end
            if ($#bases + 1 != $contig_lengths{$contig}) {
                # print the trailing region
                $start = $#bases + 1;
                print OUT
                "$contig\t$start\t$contig_lengths{$contig}\n";
            }
        }
        else {
            # we have a region at the end of our array
            # SHOULDN'T HAPPEN
            $end = $contig_lengths{$contig};
            print "Suspicious region at end of \@bases:
            ($start, $end)\n";
            print OUT "$contig\t$start\t$end\n";
            next;
        }
    }

    close OUT;
}

sub outputExpressionCounts() {

    open(OUT, ">exprCounts/$dataSetName"."_expression_counts_$contig
    .txt");

    for (my $i = 1; $i <= $#bases; $i++) {
        my $count = $bases[$i];
        print OUT "$contig\t$i\t$count\n";
    }

    close OUT;
}
```

Memory Efficient Algorithm

```
#!/perl/bin/perl

use strict;

my $start;
my $end;
my @start_array = '';
my @end_array = '';
my $index;
my $contig;


# ==========================================================
# -==================  Main Method   =====================-
# ==========================================================
my @files = <input/*>; # get all the items in the directory
foreach my $file (@files) {

    # only pay attention to files
    if (-f $file) {

        # initialize variables
        open(IN, $file) or die "Could not open file\n";
        my $lineNum = 0;
        @start_array = '';
        @end_array = '';
        print "\n\nProcessing file: $file\n";

        # for each line in the file
        while (my $line = <IN>) {
            $lineNum++;
            print "\rLine: $lineNum";

            chomp $line;
            my @columns = split(/\t/, $line);
            $contig = $columns[0];
            $start = $columns[1];
            $end = $columns[2];

            if ($lineNum == 1) {
                push (@start_array, $start);
                push (@end_array, $end);
                next;
            }

            $index = findIndex();
            if ($start >= $start_array[$index - 1] && $end <=
            $end_array[$index-1]) {
                next;
            }
            elsif ($start <= $end_array[$index - 1] && $end >
            $end_array[$index-1]) {
                rightMerge();
                checkOverlap1();
            }
            elsif ($start < $start_array[$index] && $end >=
            $start_array[$index] && $end <= $end_array[$index]) {
                leftMerge();
                checkOverlap2();
            }

            elsif ($start < $start_array[$index] && $end >
            $end_array[$index]) {
                largerInterval();
                checkOverlap2();
            }
            elsif (($start > $start_array[$index - 1] && $start <
            $start_array[$index] && $end > $end_array[$index - 1] &&
            $end < $end_array[$index]) || $index == @start_array) {
                @start_array = insertNewLineStart();
                @end_array = insertNewLineEnd();
            }
            else {
                print "Error - this shouldn't ever fire";
                last;
            }
        }

        # combine regions that are next to each other
        my $j = @start_array;
        while ($j != 0) {
            my $new_end = $end_array[$j] + 1;
            if ($new_end == $start_array[$j + 1]) {
                #put them together
                $end_array[$j] = $end_array[$j + 1];
                splice (@start_array, $j + 1, 1);
                splice (@end_array, $j + 1, 1);
            }
            else {
                #do nothing
                $j--;
            }
        }

        close IN;

        print "\nWriting Output... ";
        writeOutput();
        print "Done.\n";
    }
}

# ==========================================================
# -==================  SubRoutines   =====================-
# ==========================================================
sub writeOutput() {

    open(OUT, ">1005_to_combo_quiescent_regions_$contig.txt");
    # Identify Quiescent Regions
    my %contig_lengths = ("Supercontig_12.1", 9798893,
                          "Supercontig_12.2", 4478683,
                          "Supercontig_12.3", 5274802,
                          "Supercontig_12.4", 6000761,
                          "Supercontig_12.5", 6436246,
                          "Supercontig_12.6", 4218384,
                          "Supercontig_12.7", 4255303,
                          "Supercontig_12.8", 192308,
                          "Supercontig_12.9", 142473,
                          "Supercontig_12.10", 125404,
                          "Supercontig_12.11", 31696,
                          "Supercontig_12.12", 19714,
                          "Supercontig_12.13", 13515,
                          "Supercontig_12.14", 11565,
                          "Supercontig_12.15", 9397,
                          "Supercontig_12.16", 8983,
                          "Supercontig_12.17", 6701,
                          "Supercontig_12.18", 6309,
                          "Supercontig_12.19", 4755,
                          "Supercontig_12.20", 1646);

    # deal with first element
    if ($start_array[1] != 1) {
        my $first_end = $start_array[1] - 1;
        print OUT "$contig\t1\t$first_end\n";
    }

    for (my $i = 1; $i < @start_array - 1; $i++) {
        my $start_qui_reg = $end_array[$i] + 1;
        my $end_qui_reg = $start_array[$i + 1] - 1;
        print OUT "$contig\t$start_qui_reg\t$end_qui_reg\n";
    }

    # deal with last element
    if ($end_array[$#end_array] != $contig_lengths{$contig}) {
        my $last_start = $end_array[$#start_array] + 1;
        print OUT "$contig\t$last_start\t$contig_lengths{$contig}";
    }

    close OUT;
}

sub findIndex {
    my $len = @start_array;
    if ($start >= $start_array[$len - 1]) {
        $index = $len;
        return $index;
    }
    else {
        for (my $i = 0; $i <= $len; $i++) {
            if ($start < $start_array[$i]) {
                $index = $i;
                return $index;
            }
        }
    }
}

sub insertNewLineStart {
    splice @start_array, $index, 0, $start;
}

sub insertNewLineEnd {
    splice @end_array, $index, 0, $end;
}

sub leftMerge() {
    $start_array[$index] = $start;
}

sub rightMerge () {
    $end_array[$index - 1] = $end;
}

sub largerInterval() {
    $start_array[$index] = $start;
    $end_array[$index] = $end;
}

sub checkOverlap1 () {
    my $length = @start_array;

    for (my $i = 0; $i < $length; $i++) {
        if ($end_array[$index - 1] > $end_array[$index]){
            splice (@start_array, $index, 1);
            splice (@end_array, $index, 1);
            $length--;
        }
        elsif ($end_array[$index - 1] >= $start_array[$index] &&
        $end_array[$index - 1] <= $end_array[$index]) {
            $end_array[$index - 1] = $end_array[$index];
            splice (@start_array, $index, 1);
            splice (@end_array, $index, 1);
            $length--;
        }
        elsif ($end_array[$index - 1] < $start_array[$index]) {
            $length--;
        }
    }
}

sub checkOverlap2 () {
    my $length = @start_array;

    for (my $i = 0; $i < $length; $i++) {
        if ($end_array[$index] > $end_array[$index + 1]){
            splice (@start_array, $index + 1, 1);
            splice (@end_array, $index + 1, 1);
            $length--;
        }
        elsif ($end_array[$index] >= $start_array[$index + 1] &&
        $end_array[$index] <= $end_array[$index + 1]) {
            $end_array[$index] = $end_array[$index + 1];
            splice (@start_array, $index + 1, 1);
            splice (@end_array, $index + 1, 1);
            $length--;
        }
        elsif ($end_array[$index] < $start_array[$index + 1]) {
            $length--;
        }
    }
}
```

Find the 35 Largest Quiescent Regions

```
#!/perl/bin/perl

use strict;

# define local variables
my @contig_array;
my @start_array;
my @end_array;
my @length_array;
my $topN = 35;
my $logLevel = 2; # 0=none, 1=minimum, 2=detailed, 3=excessive

# ===========================================================
# -==================  Main Method   =====================-
# ===========================================================
# foreach file in the directory
my @files = <qRegions/*>; # get all the items in the directory
foreach my $file (@files) {

    Log("\nProcessing file $file...", 1);

    # only pay attention to files
    if (-f $file) {

        open my $data, $file or die "Could not open $file: $!";
        my $lineNum = 0;

        # for each line in the file
        while (my $line = <$data>) {

            SortTopN(); # LIST MUST BE SORTED!

            $lineNum++;

            chomp $line;
            my @columns = split(/\t/, $line);
            my $contig = $columns[0];
            my $start = $columns[1];
            my $end = $columns[2];

            my $contigLength = GetLength($start, $end);
            Log("\nProcessing length [$start, $end] ($contigLength)...
            ", 3);

            # Check if the list is full yet, add the values if not
            if (@length_array < $topN) {
                my $index = @length_array;
                $contig_array[$index] = $contig;
                $start_array[$index] = $start;
                $end_array[$index] = $end;
                $length_array[$index] = $contigLength;

                Log("\nAdded [$start, $end] ($contigLength) at index
                $index.", 2);

                next; # done with this contig
            }

            # Check if there is a shorter value and replace
            my $ignored = 1;
            for (my $j = 0; $j < @length_array; $j++) {

                my $val = $length_array[$j];

                if ($val == $contigLength) {
                    next;
                }

                if ($val < $contigLength) {

                    # do the insert
                    Log("\nInsert [$start, $end] ($contigLength) at
                    index $j.", 2);
                    $ignored = 0; # mark as not ignored for later
                    splice(@contig_array, $j, 0, $contig, );
                    splice(@start_array, $j, 0, $start, );
                    splice(@end_array, $j, 0, $end, );
                    splice(@length_array, $j, 0, $contigLength, );

                    # remove the last element
                    # as it's now not in the topN
                    pop @contig_array;
                    pop @start_array;
                    pop @end_array;
                    pop @length_array;

                    last;
                }
            }

            if ($ignored) {
                Log("Ignored.", 3);
```

```
                }
            }
        }
    }

    SortTopN();
    WriteOutput();



    # ===========================================================
    # -==================  SubRoutines   =====================-
    # ===========================================================
    sub GetLength() {
        my($a, $b);
        ($a, $b) = @_;

        return $b - $a + 1;
    }

    sub SortTopN() {

        # http://stackoverflow.com/questions/3382264/how-do-you-sort-
        parallel-arrays-in-perl

        my @permutation = sort { $length_array[$b] <=> $length_array[$a] }
        0..$#length_array;

        @contig_array = @contig_array[@permutation];
        @start_array = @start_array[@permutation];
        @end_array = @end_array[@permutation];
        @length_array = @length_array[@permutation];
    }

    sub WriteOutput() {
        my $output = ">top" . $topN . "_output.txt";

        Log("\n\nWriting output to $output...", 1);
        open(OUT, $output);

        for (my $i = 0; $i < @length_array; $i++) {
            print OUT "$contig_array[$i]\t$start_array[$i]\t
            $end_array[$i]\t$length_array[$i]\n";
        }

        close OUT;
        Log(" done.", 1);
    }

    sub Log() {
        my($msg, $option);
        ($msg, $option) = @_;

        if ($logLevel >= $option) {
            print $msg;
    }
}
```

Determine A/T/G/C Content in Whole Genome

```
#!/perl/bin/perl

use strict;

# Each line of neurosporaContigs.txt contains the sequence information
for each contig
open(IN, "neurosporaContigs.txt") or die "Could not open file \n";
my $count = 0;
my $DNA;

my $a;
my $c;
my $g;
my $t;
my $at;
my $gc;

my $Total;
my $aTotal;
my $cTotal;
my $gTotal;
my $tTotal;
my $gcTotal;
my $atTotal;
my $baseTotal;

while (<IN>) {
    $count = $count + 1;
    $DNA = $_;
    chomp $DNA;
    print "Count is: $count\n";

    countATGC ();
    addToTotal ();
}

calcPercent ();

# ==========================================================
# -==================  SubRoutines   =====================-
# ==========================================================

sub countATGC {

    #length of Quiescent region
    my $length = length ($DNA);

    $a = ($DNA =~ tr/A//);
    $c = ($DNA =~ tr/C//);
    $g = ($DNA =~ tr/G//);
    $t = ($DNA =~ tr/T//);
    $at = ($DNA =~ s/AT/AT/g);
    $gc = ($DNA =~ s/GC/GC/g);

    $Total = $a + $c + $g + $t;
}

sub addToTotal {

    $aTotal = $aTotal + $a;
    $cTotal = $cTotal + $c;
    $gTotal = $gTotal + $g;
    $tTotal = $tTotal + $t;
    $gcTotal = $gcTotal + $gc;
    $atTotal = $atTotal + $at;
    $baseTotal = $baseTotal + $Total;
}

sub calcPercent {

    open(OUT, ">whole_genome_ATCG_content.txt");

    my $gcBases = 2*$gcTotal;
    my $atBases = 2*$atTotal;

    my $aPercent = $aTotal/$baseTotal;
    my $cPercent = $cTotal/$baseTotal;
    my $gPercent = $gTotal/$baseTotal;
    my $tPercent = $tTotal/$baseTotal;
    my $gcPercent = $gcBases/$baseTotal;
    my $atPercent = $atBases/$baseTotal;

    print OUT "Percentage of A's = $aPercent\n";
    print OUT "Percentage of C's = $cPercent\n";
    print OUT "Percentage of G's = $gPercent\n";
    print OUT "Percentage of T's = $tPercent\n";
    print OUT "Percentage of GC dinucleotide bases = $gcPercent\n";
    print OUT "Percentage of AT dinucleotide bases = $atPercent\n";

}
```

<u>Determine A/T/G/C Content in all Quiescent Regions</u>

```
#!/perl/bin/perl

use strict;

# Each line of neurosporaContigs.txt contains the sequence information
for each contig
open(IN1, "neurosporaContigs.txt") or die "Could not open file \n";
my $count = 0;
my $DNA;
my $contig;
my $start;
my $end;

my $qreg;

my $a;
my $c;
my $g;
my $t;
my $at;
my $gc;

my $Total;
my $aTotal;
my $cTotal;
my $gTotal;
my $tTotal;
my $gcTotal;
my $atTotal;
my $baseTotal;

while (<IN1>) {
    $count = $count + 1;
    $DNA = $_;
    chomp $DNA;
    print "Count is: $count\n";
    open(IN2, "1005_to_combo_quiescent_regions_Supercontig_12." .
    "$count" . ".txt") or die "Could not open file contig\n";

        while (my $line = <IN2>) {

            chomp $line;
            my @columns = split(/\t/, $line);
            $contig = $columns[0];
            $start = $columns[1];
            $end = $columns[2];
            print "\n$start $end\n";

            $qreg = findQReg ();
            countATGC ();
            addToTotal ();
        }
}

calcPercent ();

sub findQReg {
    $start = $start - 1;
    $end = $end - 1;
    my $len = $end - $start + 1;
    my $qreg = substr $DNA, $start, $len;
    return $qreg;
}
```

```perl
sub countATGC {

    #length of Quiescent region
    my $length = length ($qreg);

    print "the length of DNA $length\n";

    $a = ($qreg =~ tr/A//);
    $c = ($qreg =~ tr/C//);
    $g = ($qreg =~ tr/G//);
    $t = ($qreg =~ tr/T//);
    $at = ($qreg =~ s/AT/AT/g);
    $gc = ($qreg =~ s/GC/GC/g);

    $Total = $a + $c + $g + $t;
}

sub addToTotal {

    $aTotal = $aTotal + $a;
    $cTotal = $cTotal + $c;
    $gTotal = $gTotal + $g;
    $tTotal = $tTotal + $t;
    $gcTotal = $gcTotal + $gc;
    $atTotal = $atTotal + $at;
    $baseTotal = $baseTotal + $Total;

}

sub calcPercent {

    my $gcBases = 2*$gcTotal;
    my $atBases = 2*$atTotal;

    my $aPercent = $aTotal/$baseTotal;
    my $cPercent = $cTotal/$baseTotal;
    my $gPercent = $gTotal/$baseTotal;
    my $tPercent = $tTotal/$baseTotal;
    my $gcPercent = $gcBases/$baseTotal;
    my $atPercent = $atBases/$baseTotal;

    print "Percentage of A's = $aPercent\n";
    print "Percentage of C's = $cPercent\n";
    print "Percentage of G's = $gPercent\n";
    print "Percentage of T's = $tPercent\n";
    print "Percentage of GC dinucleotide bases = $gcPercent\n";
    print "Percentage of AT dinucleotide bases = $atPercent\n";
}
```