Identification of large nucleotide inversions in *Lactobacillus delbrueckii* ssp. *bulgaricus* strains using novel Python script

Megan L. Mair

Introduction

Lactobacillus delbrueckii ssp. *bulgaricus* serves an important role in food processing. As a producer of lactic acid, the bacterium is used alongside *Streptococcus thermophilus* for the production and fermentation of yogurt, among other dairy products (Courtin *et al.*, 2003). As such, *Lactobacillus delbrueckii* is an important probiotic; the health benefits associated with the bacteria are numerous, including particular benefits for those with lactose intolerance or inflammatory bowel disease (Sheil & O'Mahony, 2007).

The commercial and medicinal importance of this bacterium has generated interest in understanding its genome. Van de Guchte *et al.* have conducted one such investigation into the genome of one strain, *Lactobacillus delbrueckii* ssp. *bulgaricus* strain ATTC 11842 (2006). Alongside a publication of the full genome, Van de Guchte *et al.* characterized several unique features within the DNA. These included the identification of a CRISPR region (clustered regularly interspaced short palindromic repeats), annotation of a high number of pseudogenes, and, of particular interest, the presence of a large, 47 kbp region featuring an inverted repeat.

Figure 1. Example of an inverted repeat. An inverted repeat is when the reverse complement of a given nucleotide sequence appears downstream the forward sequence.

Inverted repeats are DNA elements where the reverse complement of a given region occurs further downstream the original sequence. This repeat can occur immediately after the original forward sequence, as in a palindromic sequence, or as a repeat separated by a unique nucleotide spacer.

Originally believed to be rare within prokaryotic organisms, more recent investigation has shown inverted repeats in prokaryotes may be quite common in noncoding regions (Treangen *et al.*, 2009). However, inverted repeats of the size reported in the ATTC 11842 genome are virtually unheard of. Formerly, an inverted repeat of comparable size (24 kbp) has only been reported in *Streptomyces griseus*, the result of an induced mutation by UV damage (Goshi et al., 2002).

Single primer PCR targeting the inverted repeat found in *L. bulgaricus* strain ATTC 11842 showed the inverted repeat may be conserved within 30 different strains of *L. bulgaricus*, albeit in a wide variation of sizes (Van de Guchte *et al.*, 2006). While these results can be further investigated in the wet lab, the publication of full genomes for eight *L. bulgaricus* strains enable confirmation of these results by computational means as well. Additionally, bioinformatics methods allow for further investigation into the content of these repeats, including finding their exact sizes, spacer sequences, and identifying the genes encoded in the region.

Several programs that can find nucleotide repeats within a DNA sequence are available. Previous nucleotide repeat finding programs include Tandem Repeats Finder, which, in spite of the title's suggestion, can also find repeats separated by a spacer sequence (Benson, 1999). However, Tandem Repeats Finder can only find repeats up to 500 bp in length, and therefore is not compatible with finding repeats the size of the one within *L. bulgaricus* ATTCC 11842 (>20 kbp). Additional repeat finding tools include Blaster Suite and Recon, both which focus on building libraries of frequently occurring repeats within eukaryotic organisms (Lerat, 2010). Another available program is RepSeek (Achaz *et al.*, 2007). While RepSeek is compatible with finding singular repeats of the target size, it lacks compatibility with Windows operating system.

Ultimately, it was decided to build a new repeat finding program because it enabled additional practice with Python programming and a deeper understanding of the program algorithm than would be offered by using a program previously built. In addition, this allowed for the development of original parameters that could be manipulated to suit the needs of this research or future research.

A novel Python program was thereby built in order to identify large, inverted repeats within the genomes of eight *Lactobacillus delbrueckii* ssp. *bulgaricus* strains. The results from this program were utilized to further characterize the repeat events within *L. bulgaricus*. These inverted repeats lend additional insight into the genome of *L. bulgaricus*, including details about the evolutionary relationship of the different strains.

Methods

Construction of novel nucleotide repeat finder program

The overall goal of the repeat finding program was to find exact nucleotide repeats (either direct or inverted) of a variable window size within a single or multi-line FASTA file. It was designed to create an output file with a report (hereafter referred to "repeat report") containing the following information:

1. Size of the motif sequence
2. Index or indices at which the motif occurs
3. Sequence of the motif
4. Index or indices at which the reverse complement sequence of the motif occurs, if it occurs
5. Sequence of reverse complement motif, if applicable.

Additionally, the program was designed to output a FASTA file containing the sequences of all detected repeats, within the context of filler 'N' nucleotides to denote sequences not repeated within the genome. This was done to allow visualization of sequence motifs against the original sequence using simple text viewers, without the need for sequence alignment.

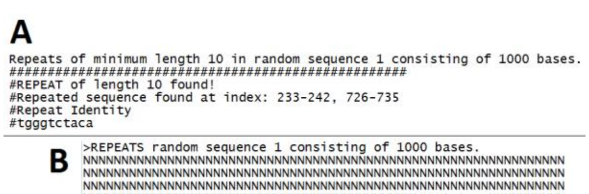


Figure 2. Example program output. A) Example of repeat report indicating detection of a 10-mer repeat at the given indices. B) Example of FASTA output, showing detected repeat in context with non-repeating nucleotides.

It is worth noting this program is not designed to only return maximal repeats. Maximal repeats are the maximum possible string size that has a repeat event; for example, the sequence

AATTCGGGAATTGAATTC contains the repeat of substring "AATTC" twice and substring "AATT" three times. A program that only provides the maximal repeat would report the "AATTC" repeat, but not the "AATT" motif. This program returns maximal repeats, in addition to their repeated substrings. This was done in order for the program to report the full array of possible repeat elements. Whilst this substring data is superfluous in the context of finding a single, large, inverted repeat within this research, with modification this technique could make the program useful for finding CRISPRs or smaller repeat elements in the future.

To eliminate dependency on pre-built algorithms, packages such as Biopython, which could be used to manage input sequences, and Numpy, which can assist in creating suffix trees to find motif elements in a sequence string, were not used. This, in addition to the program returning smaller repeated elements of maximal repeats, predictably results in a slower program, and the resultant code may not be the most computationally efficient way to accomplish the given task. This means the program is optimal for finding smaller, frequent repeat elements within a nucleotide sequence rather than larger repeat elements; however, it is still able to detect repeats *de novo* in any given genome.

To explain, the code was designed to find repeats by turning the input FASTA file sequence into a string, and then generate a sliding window of set minimum size that parses through the sequence string via a *for* loop. The regular expression *finditer* is used to count the number of times the sequence in the window occurs. If the sequence occurs more than once, it is appended to a dictionary of motif values. This motif dictionary is checked before identifying repeats within a window to ensure a repeat element is only reported once.

If the sequence within a window occurs more than once, and the sequence has not already been appended to the motif dictionary, then the window begins expanding by an interval of one nucleotide. The sequence string is then checked with the *finditer* expression to see if the sequence within the expanding window appears more than once. If the repeat is found, then it is recorded in the repeat report and the window continues expanding until it finds the maximal repeat. If the expanded window does not contain a repeat event, the window reverts back to its minimum size and continues to parse the sequence. A diagram of this expanding window process can be found in Figure 3.

Inverted repeats are found by first determining the reverse complement of the sequence in the window. A dictionary of complementary nucleotides is declared, and the corresponding nucleotide for each character in the window substring is found. Then, now-complementary window sequence is reversed. The full sequence string is then checked to see if it contains the reverse complement. If the reverse complement is in the sequence, the index value of the first occurring reverse complement is determined.

The reverse complement string should not overlap with the window string. For example, when the sequence "GGGGGCCCCC" with window size 5 is input, the algorithm should only count "GGGGG" and "CCCCC" as reverse complement sequences, but not "GGGGC" and "GCCCC". Repeated window strings that might overlap are already ruled out as repeats by the regular expression *finditer*, which automatically precludes overlapping repeats. However, this is not applied to the reverse complement

sequences and their forward counterparts. In order to rule out these overlapping complementary sequences, the index ranges of the first sequence and its reverse complement are compared. If there is any overlap in these indices, then this complementary pair is skipped over. If there is no overlap, the starting indices of each reverse complement substring are found through *finditer*. If there is already data printed for a repeated forward sequence, the data on the reverse complement is simply added to the file. If the forward sequence only occurs once, the information on the forward sequence and the index or indices of its reverse complement are written into the repeat report (Figure 2A).

While recording repeat elements, the program additionally records the index at which each nucleotide in a repeated sequence occurs. Finally, once the *for* loop reaches the end of a sequence string, a FASTA file is created using this data. A new list is generated and assigned the same length as the original sequence, with each index filled with a filler 'N' nucleotide. These 'N's are then replaced with nucleotides if they correspond with an index site of a repeated sequence. This list is joined into a new, continuous sequence string, which is then output to a FASTA formatted file (Figure 2B).

The full Python script can be found in Appendix A. This script was written and run using Python version 2.7.3. The program was additionally tested with and supported by Python 3.5.

Figure 3. Program logic for finding repeat and inverse sequences.
1: Sliding window of given size parses the sequence.
2: Upon finding repeats or inverse repeats of the sequence in the window, the indices of these repeat events are recorded and window starts expanding.
3: Window expands by intervals of one nucleotide and identifies repeats and inverse repeats of the window sequence
4 and 5: If expanding window does not identify repeat event, then it reverts back to original size and continues parsing through the sequence.
6: As sliding window continues downstream, repeat events already identified are not recorded

Decreasing Program Runtime

One major disadvantage of this program in comparison to other repeat finding tools is that Python is notoriously slow compared to other programming languages; that this program was designed to return more than just maximal repeat events only serves to elongate the program runtime. Therefore, this program is primarily optimal for small input sequences, or sequences with small or frequent repeats. For example, a FASTA file with 1.5 Mbp consisting entirely of 5-mer repeats took approximately 30 seconds to process; in contrast, it took on the same machine over 24 hours of continuous searching to identify the 23 Kbp repeat within the approximately 2 Mbp genome for ATCC 11842.

To assist with this disadvantage, the program run time was drastically reduced by applying the concept behind a frequently used molecular biology technique. Van de Guchte *et al*. originally found the presence of large inverted repeats in other *L. bulgaricus* strains through one primer PCR (2006). In PCR, forward and reverse primers are designed to anneal to the beginning and end of a target DNA sequence, enabling replication and subsequent amplification of the captured product by DNA polymerase. Computationally, the sequence of a designated forward "primer", like the one originally used by Van de Guchte *et al.*, can be utilized to determine the index near the start site of the conserved inverted repeat. This index site can be used to limit the *for* loop in the original program to parse only the sequence region thought to contain a repeat event.

Van de Guchte *et al.* did not publish the original PCR data, so a new primer sequence needed to be determined. The repeat sequence file output by the Python program was used as a guide, and the primer sequence "**AAGTTGACCCCGG**" was determined based on 13 nucleotides near the beginning of the repeat found in *L. bulgaricus* strain ATCC 11842. Of course, in the wet lab, a primer that that is not a 100% perfect match with its complementary target DNA sequence can still be chemically favorable for binding in the right conditions. This luxury is not a given when using a Python script that finds perfect matches, so it was important to ensure each FASTA file contained a perfect match for the computational primer sequence, and count the number of times the primer sequence appeared to ensure the primer would not detect other regions in the DNA, indicating a "false positive" targeting site.

To accomplish both this, and the retrieval of the index of the putative repeat region, a short Python script was devised. The script is able to find an input target sequence in a larger FASTA file, count the number of times the sequence occurs, determine the indices at which these sequences start, and finally output this information to a tab separated format file. This script used many of the same structures of the original repeat finding program, utilizing the regular expression *finditer* in a similar way.

Once the index at which the primer sequence occurred in each genome was obtained, it was used to alter the format of the *for* loop that parses the sequence files in the main body of the repeat finding program. The loop began parsing the sequence 10000 nucleotides upstream of the determined index, using a window size minimum of 5000 bp. The program was additionally modified to exit the loop when its window stopped expanding, indicating the entirety of the large repeat was found.

Both the code used to find the index of the primer sequence, and the output used for this step, can be found in Appendix B.

Alignment of Repeat Sequences

The repeated sequences within the different strains were compared through nucleotide alignment to determine similarities within the repeat regions, and thereby identify whether the repeats arose independently or were conserved among the *Lactobacillus delbrueckii* ssp. *bulgaricus* strains. A local BLASTN was used for the sequence alignment (Altschul et al., 1990). A FASTA file consisting of all the forward repeat sequences was compiled using the sequence files output by the repeat finding program. A database consisting of the repeated sequences was constructed using this file. Once the database was

built, the compiled FASTA file was aligned to the database. The parameters of word-size>100 and e-value<0.0001 were used. Results from this BLAST analysis can be found in Appendix C.

Retrieval of Gene Annotations

Annotations of the genes encoded by the inverted repeat were retrieved through Genbank (Benson *et al.*, 2005). The indices of the largest repeat output by the repeat finding program were used, and corresponding annotations for all genes that fell in the reported range of the indices were copied to a file. A short Python script was used to obtain all gene functions, including reported pseudogenes, from this file. The Python *Counter* function was used to organize and count the annotated genes based on function. The script is viewable in Appendix D, and the output from this script is summarized in Table 2.

Construction of Phylogenetic Trees

A phylogenetic tree of the gene encoding the RecA protein of all *L. bulgaricus* strains was constructed. The RecA protein is responsible for assisting in DNA repair in prokaryotic organisms. Highly conserved among bacteria species, it was previously identified by Huang *et al.* as a target gene ideal for differentiating and calculating phylogenetic trees for *Lactobacillus* bacteria (2012).

First, a nucleotide alignment file was generated. The sequences for the RecA genes within the *Lactobacillus delbrueckii* ssp. *bulgaricus* were identified and downloaded through the gene annotation records on Genbank. The RecA nucleotide sequence for a close relative of *L. bulgaricus*, *Lactobacillus delbrueckii* spp. *lactis* strain KCTC-3034 LA1, was also retrieved, in addition to the RecA gene sequence of more distantly related *Lactobacillus acidophilus* strain LA1 (Yu et al., 2011). These sequences were compiled into a single FASTA file, which was then aligned using MUSCLE into the Phylip file format (Edgar, 2004). The output alignment was used for construction of the phylogenetic trees.

The generated alignment file was then used to construct a maximum likelihood phylogenetic tree using RaXML, via the T-Rex Online web portal (Stamatakis, 2006 & Boc *et al.*, 2012). Trees were constructed using both the default and bootstrapping options, with *Lactobacillus acidophilus* LA1 set as the outgroup. Bootstrapping enabled the reporting of confidence values. The trees were visualized using T-Rex Online.

Results

The constructed program was able to find a large, inverted repeat in *Lactobacillus delbrueckii* ssp. *bulgaricus* strain ATCC 11842 *de novo* when the minimum repeat size was set to 20 Kbp. Using the index limiting steps described above and a repeat size minimum of 5000 bp, the program was able to locate large, inverted repeats within five of the remaining seven *L. bulgaricus* strains.

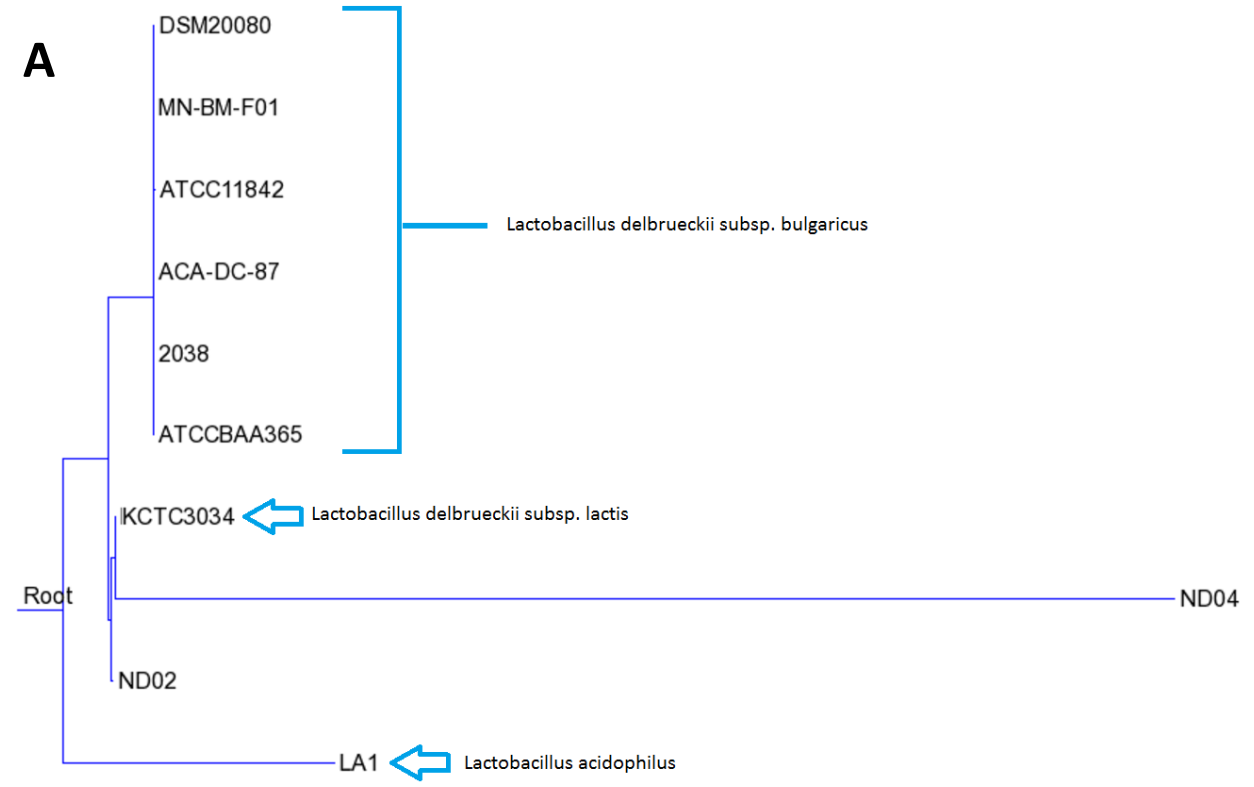
The repeat located in *L. bulgaricus* strain ATCC 11842 was approximately 23 Kbp long, with a spacer of size 1358 bp. The smallest detected repeat was within *L. bulgaricus* DSM20080, which was found to be 8541 bp long and separated by an approximately 38 Kbp spacer sequence. The largest repeat was within strain ATCC BAA-265, at around 37 Kbp long and a spacer of 1014 bp. Repeats of varying sizes were also

found in *L. bulgaricus* strains ACA-DC87, 2038, MN-BM-F01. A summary of repeats found within all the *L. bulgaricus* strains can be found in Table 1.

Table 1. Summary of inverted repeats found within *L. bulgaricus* strains.

Strain ID	Repeat Size	Sequence Range	Spacer size	IR range
ATCC 11842	23088	918952-942039	1358	943397-966484
2038	20053	933573-953625	14426	968051-988103
ACA-DC87	25130	897120-922249	8830	931079-956208
ATCC-BAA-265	37663	891540-929202	1014	930216-967878
DSM-20080	8541	901094-909634	37869	947503-956043
MN-BM-F01	15964	915960-931923	23004	954927-970890
ND02	None found	-	-	-
ND04	None found	-	-	-

Notably*, L. bulgaricus* strains ND02 and ND01 did not contain repeats over the given threshold of 5000 bp. Acknowledging this lack of conserved repeat, a phylogenetic tree was constructed to test the evolutionary relationship between strains ND02 and ND04 and the other *L. bulgaricus* strains using the RecA gene, which has been previously used to determine evolutionary relationships among *Lactobacillus* species (Huang *et al.*, 2012). RaXML-generated phylogenetic trees visualized through the T-REX portal determined ND02 and ND04 were more closely related to *Lactobacillus delbrueckii* ssp. *lactis* than the other *L. bulgaricus* strains. However, confidence values for the phylogenetic tree were markedly low (Figure 4B).



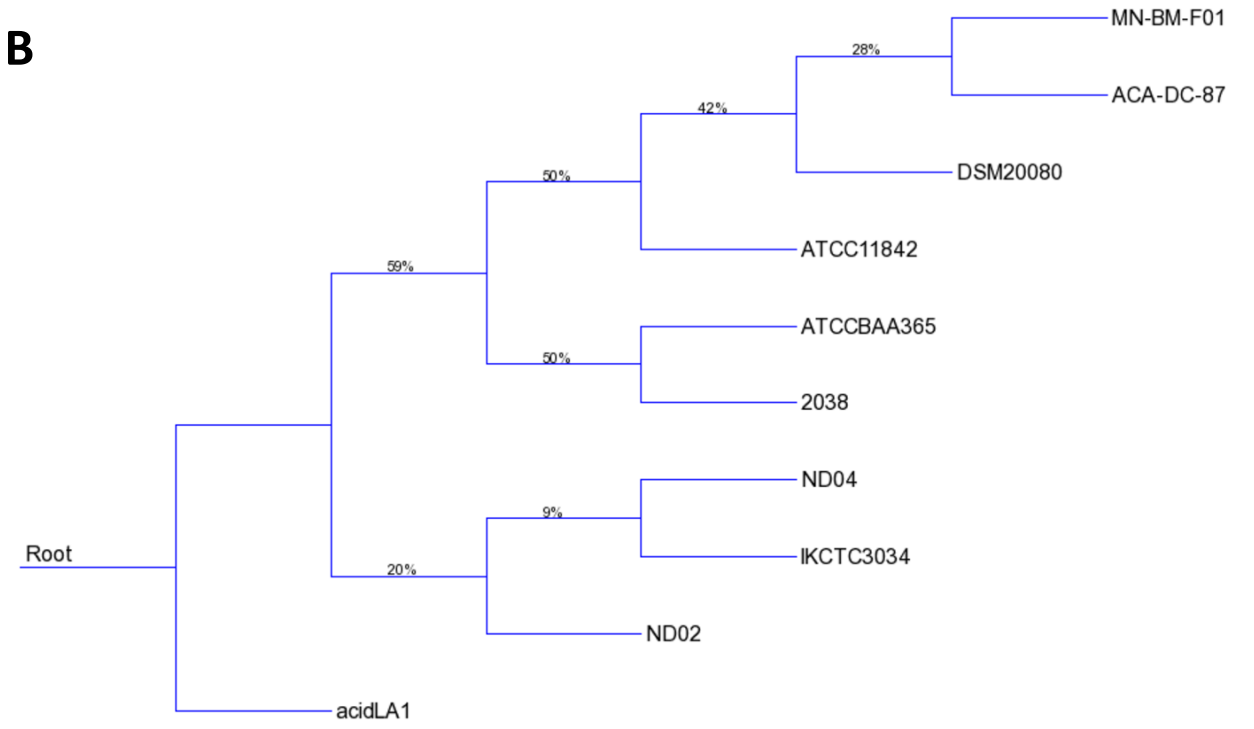


Figure 4. Phylogenetic trees comparing RecA genes of *L. bulgaricus* strains. L. acidophilus LA1 is considered the outgroup. A) Phylogenetic tree without bootstrapping. Branch length corresponds with

number of nucleotide changes. *L. bulgaricus* strains ND04 and ND02, which lack a large, inverted repeat, are not grouped with the other *L. bulgaricus* strains, but rather with *L. lactis* strain KCTC3034. B) Phylogenetic tree with bootstrapping. Tree is marked by confidence scores for the clade groupings. Note strains ND04 and ND02 continue to be grouped with the *L. lactis* strain rather than with the other *L. bulgaricus* strains.

A BLASTN alignment was used to compare all six discovered repeats against each other. Results from the alignment showed several regions of the repeat that were highly conserved among the strains. Within conserved regions, sequence identity matched nearly perfectly, ranging from 95.7% to 100% identity (Appendix C). The smallest conserved aligned region was 114 nucleotides, and the largest conserved region was over 15 Kbp.

Genbank annotations for the genes encoded by the repeated region showed several conserved gene products (Table 2). Many of the proteins within the repeats were associated with membrane transport, including an efflux pump driven by sodium ions and a number of gene products associated with ABC transport. Conserved genes also included a transcriptional regulator related to the MerR family. There were also several proteins involved in amino acid synthesis. A full report of genes within the repeats by strain can be found in Table 2.

Table 2. Annotations of genes encoded in the large, inverted repeat. Numbers next to the gene function indicate the number of genes with the given annotation. Note that pseudogenes were not annotated within the genome for *L. bulgaricus* 2038.

ATCC-BAA-265:
'hypothetical protein': 20, 'pseudogene': 8, 'Predicted transcriptional regulator': 2, 'ABC-type antimicrobial peptide transport system,': 2, 'Peptide methionine sulfoxide reductase': 1, 'Na+-driven multidrug efflux pump': 1, 'Phosphoribosylanthranilate isomerase': 1, 'Carbamoylphosphate synthase small subunit': 1, 'Metal-dependent amidase/aminoacylase/carboxypeptidase': 1, 'Saccharopine dehydrogenase related protein': 1, 'Integrase': 1, 'Protein-tyrosine-phosphatase': 1
ACA-DC-97:
'pseudogene': 18, 'hypothetical protein': 12, 'Arsenate reductase': 1, 'Transcriptional regulator, MerR family': 1, 'Oligopeptide ABC transporter, periplasmic': 1, 'ATP-dependent DNA helicase': 1, 'ABC-type antimicrobial peptide transport system,': 1, 'Ribosomal-protein-L7p-serine acetyltransferase': 1, 'Mobile element protein': 1, 'ABC transporter, ATP-binding protein': 1, 'Carbamoyl-phosphate synthase small chain': 1
ATCC 11842
'pseudogene': 12, 'hypothetical protein': 8, 'ABC transporter permease': 3, 'N-acetyltransferase': 2, 'toxin-antitoxin system, antitoxin component, Xre': 1, 'MerR family transcriptional regulator': 1, 'arsenate reductase ArsC': 1, 'ABC transporter ATP-binding protein': 1, 'FAD-binding protein': 1, 'SAM-dependent methyltransferase': 1, 'N-acetylmuramoyl-L-alanine amidase': 1, 'DUF2798 domain-containing protein': 1, 'carbamoyl phosphate synthase small subunit': 1, 'LysR family transcriptional regulator': 1, 'carbamoyl phosphate synthase large subunit': 1, 'multidrug transporter MatE': 1,

'integrase': 1, 'hydrolase': 1
2038:
'Hypothetical protein': 5, 'Hypothetical conserved protein': 4, 'ABC transporter, permease protein': 1, 'Carbamoyl-phosphate synthase, small chain': 1, 'ABC transporter ATP-binding protein': 1, 'site-specific recombinase, phage integrase': 1, 'Carbamoyl-phosphate synthase, small subunit': 1, 'Arsenate reductase, arsC,': 1, 'Methyltransferase': 1, 'Carbamoyl-phosphate synthase, large subunit,': 1, 'Putative acetyltransferase': 1, 'ABC transporter, transmembrane region:ABC': 1, 'Putative flavoprotein subunit of a reductase': 1, 'Na+ driven multidrug efflux pump': 1, 'ABC transporter, intermembrane subunit': 1 *Note: pseudogenes not annotated
MN-BM-F01:
'pseudogene': 18, 'hypothetical protein': 4, 'ABC transporter permease': 2, 'MerR family transcriptional regulator': 1, 'MATE family efflux transporter': 1, 'ABC transporter ATP-binding protein': 1, 'SAM-dependent methyltransferase': 1, 'ABC transporter': 1, 'GNAT family N-acetyltransferase': 1, 'DUF2798 domain-containing protein': 1, 'arsenate reductase ArsC': 1, 'carbamoyl phosphate synthase small subunit': 1, 'carbamoyl phosphate synthase large subunit': 1, 'integrase': 1, 'hydrolase': 1
DSM-20080:
'pseudogene': 8, 'ABC transporter permease': 1, 'MerR family transcriptional regulator': 1, 'MATE family efflux transporter': 1, 'carbamoyl-phosphate synthase large subunit': 1, 'SAM-dependent methyltransferase': 1, 'carbamoyl phosphate synthase small subunit': 1, 'hypothetical protein': 1

Discussion

The program was able to independently locate a large, 23088 bp inverted repeat within the genome for *Lactobacillus delbrueckii* ssp. *bulgaricus* ATTC 11842. This data collaborates with results originally reported by Van de Gutche *et al*., indicating the designed program can successfully locate exact nucleotide inversions and repeats *de novo*. The program was additionally able to locate large inverted repeats of varying sizes within *L. bulgaricus* strains ACA-DC 87, ATCC BAA-265, DSM-20080, MN-BM-F01, and 2038. This supports the original PCR data that suggested the inverted repeat varied in size within other strains of the bacteria.

Interestingly, there appears to be a relationship between the size of the repeated sequence and the length of the spacer sequence between the repeats in the different *L. bulgaricus* strains. The relationships between repeat size and space sizer is generally inversely proportional; that is, a larger repeat would be expected to have a smaller spacer, and a smaller repeat would be expected to have a larger spacer (Figure 5A). This information was collaborated mathematically (Figure 5B); analysis by logarithmic regression showed repeat size can explain approximately 90% of variation in spacer size within the dataset (R^2=0.8964). The spacer size for *L. bulgaricus* strain ATCC 11842 was markedly the only value that did not strongly fit this model. Eliminating strain ATCC 11842 as an outlier, the logarithmic regression analysis showed repeat size can explain over 99% of variation of spacer size within the dataset (data not pictured).

A

Strain ID	Repeat Size	Spacer size
ATCC 11842	23088	1358
2038	20053	14426
ACA-DC87	25130	8830
ATCC-BAA-265	37663	1014
DSM-20080	8541	37869
MN-BM-F01	15964	23004

B

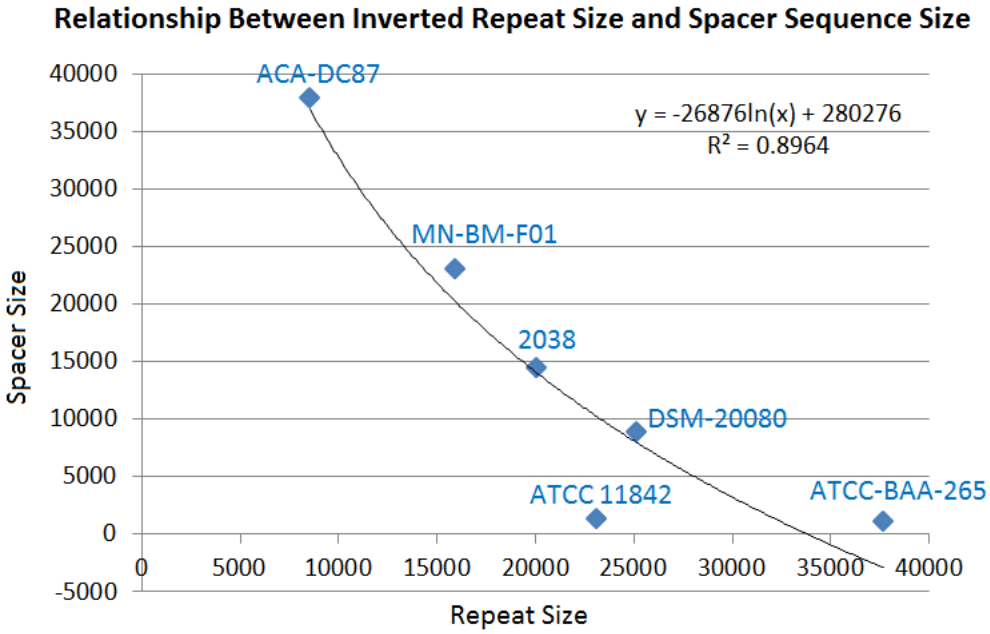


Figure 5. Comparison of size of the inverted repeat relative to size of the spacer sequence. A) Table indicating the size of the inverted repeat sequence and size of the nucleotide spacer separating the repeats, organized by strain. B) Logarithmic regression graph of repeat vs. spacer size. Plot points correspond with labeled strain IDs. The regression model shows repeat size can explain approximately 89.64% of variation in spacer size.

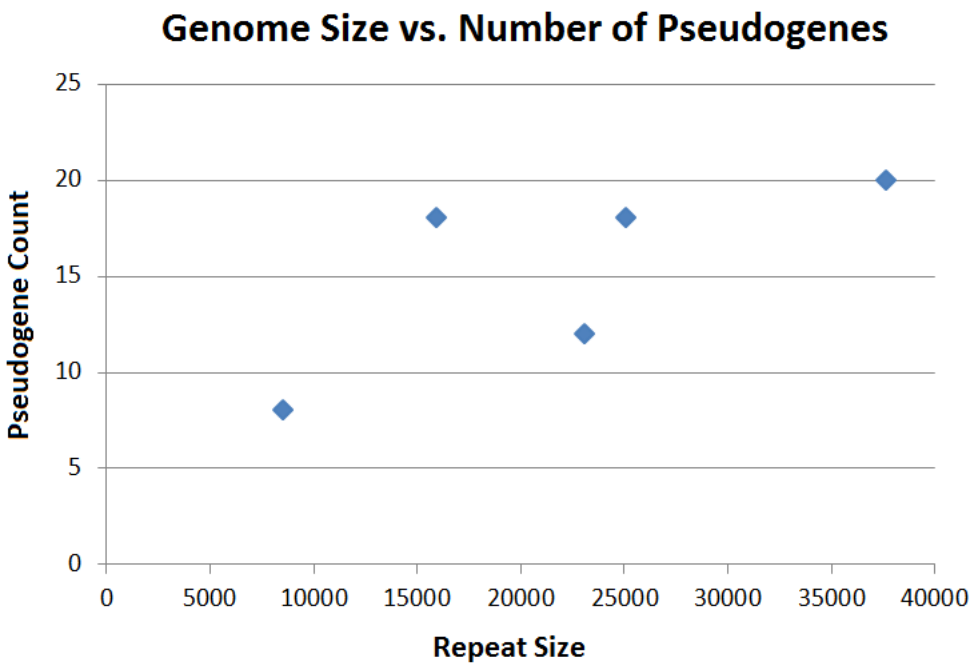
Notably, repeats within *L. bulgaricus* strains ND02 and ND04 were not identified. By varying the window size in the repeat finding program to approximately 2000 bp, the program detected an approximately 2000 bp repeat in strain ND04 near an index site reported by the primer location program. However, running the *de novo* version of the repeat finding program on the full genome for *L. bulgaricus* ND04 showed multiple, distinct repeat events of similar size. Due to this, and lack of shared homology, the approximately 2000 bp repeat event in strain ND04 cannot be reported as the same characteristic, single, large inverted repeat event found in the other *L. bulgaricus* strains.

The absence of conserved large repeats within the genomes for within *L. bulgaricus* strains ND02 and ND04 warranted further investigation. A phylogenetic tree using the RecA gene was constructed in order to test the relationship between ND02 and ND04 with the remaining strains. A second subspecies, *L.*

delbrueckii ssp. *lactis,* was included as a control for clade groupings, and a *Lactobacillus acidophilus* strain was designated as the outgroup. The phylogenetic trees constructed using the RecA sequences for all strains grouped *L. bulgaricus* strains ND02 and ND02 in a clade with *Lactobacillus delbrueckii* spp. *lactis*, separate from the *L. bularicus* strains. The confidence values are notably low (Figure 4B), but the trees constructed for all the bacteria, in addition to the lack of large characteristic inverted repeat within the genomes for ND04 and ND02, suggests the subspecies of these bacteria might be misidentified.

The results of BLASTN analysis showed the repeats within the six strains had several large, conserved regions that ranged from 95.7 to 100% identical in nucleotide content. The highly similar identity between conserved regions within the repeats implies the repeats may have come from a single ancestor shared by all *Lactobacillus delbrueckii* spp. *bulgaricus* strains. This could mean that either the inverted repeat has either grown larger within the different strains over time, or is actually shrinking in size.

Results by Van de Guchte *et al.* found the genome for *L. bulgaricus* strain ATCC 11842 consists largely of fragmented pseudogenes, implying the total genome is losing coding regions and consequentially reducing its size. This might suggest the large repeat is in fact shrinking and phasing out of the *L. bulgaricus* genome. However, there appears to be no strong relationship between repeat sizes and the number of pseudogenes within the repeat (Figure 6). While the largest repeat contained the most pseudogenes and the smallest repeat contained the fewest, the repeats between these two extremes saw wide variation in pseudogene count. This could mean the pseudogenes are eliminated from the repeats at varying rates, accounting for the variation of repeat and spacer size across the different strains.



Strain ID	Repeat Size	# Pseudogenes
ATCC 11842	23088	12
2038	20053	No data
ACA-DC87	25130	18
ATCC-BAA-265	37663	20
DSM-20080	8541	8
MN-BM-F01	15964	18

Figure 6. Relationship between sizes of inverted repeat and number of pseudogenes therein. (Top) Plotted data showing size of the repeated sequence versus the count of pseudogenes within the repeat. (Bottom) Table summarizing data within the graph.

The fact the inverted repeat sequences are likely conserved from an ancestral source would imply the genes once served an adaptive purpose. It may have been beneficial for *Lactobacillus delbrueckii* spp. *bulgaricus* to have multiple copies of the genes encoded in the repeat, either due to increased transcription or an increased quantity of protein products. Gene annotations for the genes contained in the repeat indicate a variety of known gene functions, including genes associated with transcription, membrane transport, and amino acid synthesis. Given the repeat was adaptive in the past, it might have been the genes within the repeat are metabolic in function and helped the bacteria survive and process energy within a past living environment.

In conclusion, the novel repeat finding program was able to locate large, inverted repeats in six *L. bulgaricus* strains. Analysis of the sequence content showed this repeat was a conserved feature in the L*. bulgaricus* genome. Lack of this characteristic repeat within strains ND02 and ND04 point to the

theory that these strains do not belong to the *L. bulgaricus* clade, a theory supported by phylogenetic analysis of RecA genes. Gene annotations and varying repeat sizes provide evidence that the inverted repeat is likely shrinking in size and being phased out of the genome of *L. bulgaricus*.

Bibliography

Achaz G, Boyer F, Rocha EP, Viari A, Coissac E. Repseek, a tool to retrieve approximate repeats from large DNA sequences. Bioinformatics. 2007;23(1):119-21.

Altschul, S.F., Gish, W., Miller, W., Myers, E.W. & Lipman, D.J. (1990) "Basic local alignment search tool." J. Mol. Biol. 215:403-410.

Bao Z, Eddy SR. Automated de novo identification of repeat sequence families in sequenced genomes. Genome Res. 2002;12(8):1269-76.

Benson DA, Karsch-mizrachi I, Lipman DJ, Ostell J, Wheeler DL. GenBank. Nucleic Acids Res. 2005;33(Database issue):D34-8.

Benson G. Tandem repeats finder: a program to analyze DNA sequences. Nucleic Acids Res. 1999;27(2):573-80.

Boc A, Diallo AB, Makarenkov V. T-REX: a web server for inferring, validating and visualizing phylogenetic trees and networks. Nucleic Acids Res. 2012;40(Web Server issue):W573-9.

Courtin, Pascal & Rul, Françoise. (2003). Interactions between microorganisms in a simple ecosystem: Yogurt bacteria as a study model. http://dx.doi.org/10.1051/lait:2003031. 84. . 10.1051/lait:2003031.

Edgar, Robert C. (2004), MUSCLE: multiple sequence alignment with high accuracy and high throughput, Nucleic Acids Research 32(5), 1792-97.

Goshi K, Uchida T, Lezhava A, et al. Cloning and analysis of the telomere and terminal inverted repeat of the linear chromosome of Streptomyces griseus. J Bacteriol. 2002;184(12):3411-5.

Huang CH, Chang MT, Huang MC, Wang LT, Huang L, Lee FL. Discrimination of the Lactobacillus acidophilus group using sequencing, species-specific PCR and SNaPshot mini-sequencing technology based on the recA gene. J Sci Food Agric. 2012;92(13):2703-8.

Lerat E. Identifying repeats and transposable elements in sequenced genomes: how to find your way through the dense forest of programs. Heredity (Edinb). 2010;104(6):520-33.

Treangen TJ, Abraham AL, Touchon M, Rocha EP. Genesis, effects and fates of repeats in prokaryotic genomes. FEMS Microbiol Rev. 2009;33(3):539-71.

Sheil B, Shanahan F, O'Mahony L. Probiotic effects on inflammatory bowel disease. J Nutr. 2007;137(3 Suppl 2):819S-24S.

Stamatakis A. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics. 2006;22(21):2688-90.

Van de Guchte M, Penaud S, Grimaldi C, et al. The complete genome sequence of Lactobacillus bulgaricus reveals extensive and ongoing reductive evolution. Proc Natl Acad Sci USA. 2006;103(24):9274-9.

Yu J, Wang WH, Menghe BL, et al. Diversity of lactic acid bacteria associated with traditional fermented dairy products in Mongolia. J Dairy Sci. 2011;94(7):3229-41.

Appendix A: Repeat Finding Code

```
import sys, re #used regular expression findinter

header = [] #input FASTA file header
motifs = {} #dictionary of all repeated sequences
indices = {} #index of each base in a repeated sequence
i = 0

#Parse an input FASTA file
def read_file(input_file):
    print("Reading sequence file...")
    file1 = open(input_file, 'r')
    seq = [] #list for sequence lines, helps for reading a multi-line FASTA file
    for line in file1:
        # parsing the file
        line = line.rstrip('\n') # getting rid of new line characters
        if line.startswith('>'):
            header.append(line.strip('\n').strip('>'))
        else:
            line = line.strip('\n')
            seq.append(line)
    sequence = "".join(seq) #take all the seq lines in a multi-line FASTA file, put
them all in the same string
    seq = []#empty the list because we don't need it any longer
    return sequence

#Take index of each nucleotide in a repeat and stores them in dictionary, helps
generate alignment file.
def index(window, repeat_indices):
    for i in range(0,len(repeat_indices)):
        for x in range(0,len(window)):
            base = window[x]
            index = repeat_indices[i]+x
            indices[index]=base
    return(indices)

#Finds the indexrange of a repeated sequence and converts format for printing
def repeat_range(repeat_indices, dynamic_window):
    expect = None
    run = []
    result = [run]
    #Go through indices in list and group them if they are consecutive
    for index in repeat_indices:
        if (expect is None) or (index == expect):
            run.append(index)
        else:
            run = [index]
            result.append(run)
        expect = index + 1
    final = []
    #Format a range for printing
    for items in result:
        if len(items)>1:
            #NOTE: Need to add 1 because Python counts from 0
            first = str(items.pop(0)+1)
            last = str(items.pop(-1)+dynamic_window)
            final.append([first + "-" + last])
        elif len(items)==1:
            items = list(map(lambda x: x + 1, items))
            first = str(items.pop(0))
            last = str(int(first)-1+dynamic_window)
            final.append([first + "-" + last])
```

```
        else:
            final.append("ERROR") #Place holder that should not be used...make sure
data formatting worked.
    printout = str(final).replace('[', '').replace(']','').replace("'",'') #turn list
into easy-to-read text
    return printout

#Find repeated sequence and its index
def repeated(window, i,dynamic_window):
    if sequence.count(window) > 1:
        size = len(window)
        repeat_indices = [base.start() for base in re.finditer(window, sequence)]
#generate list of indices where each sequence occurs'
        printout = repeat_range(repeat_indices,dynamic_window)#repeat_range function
defined above
        repeat_index = []
        index(window, repeat_indices)#index function defined above
        repeat_indices= list(map(lambda x: x+1, repeat_indices))
        for num in repeat_indices:
            new = str(num) + "-" + str((num-1)+size)
            repeat_index.append(new)
        if window not in motifs:
            motifs[window] = repeat_indices

report_outfile.write("#######################################################\n" +
                                "#REPEAT of length " + str(size) + " found!" + "\n" +
                                "#Repeated sequence found at index: " + printout +
"\n" +
                                "#Repeat Identity" + "\n" +
                                "#" + window + "\n")
        return(window)

#Find and take the index of reverse complement sequences
def revComplement(window, i, window_size, dynamic_window):
    complements = {'A': 'T', 'T': 'A', 'C': 'G', 'G': 'C'}
    revComp = "".join(complements.get(base, base) for base in reversed(window))  #
create reverse complement list, turn it into a string
    if revComp in sequence:
        repeat_index = sequence.find(revComp)
        if repeat_index < i + dynamic_window and i < repeat_index + dynamic_window:  #
ensure the repeated sequence does not contain a fragment of the original window (no
mathematical overlap of index ranges)
            pass
        else:
            revComp_re = revComp
            repeat_indices = [base.start() for base in re.finditer(revComp_re,
sequence)]#Take the index for output sequence alignment
            printout = repeat_range(repeat_indices, dynamic_window) #Format for
printing
            index(revComp, repeat_indices)
            if sequence.count(window) > 1:
                if revComp not in motifs:
                    motifs[window] = repeat_indices
                    report_outfile.write("#Reverse complement sequence found at index:
" + printout + "\n"+
                                         "#Reverse Complement Sequence Identity" +
"\n" +
                                         "#" + revComp + "\n")
                    return "continue" #repeat found, keep searching for repeats in
dynamic window
            else:
                if revComp not in motifs:
                    motifs[window] = repeat_indices
```

```
                    index(window, [i])
                    length = str(len(window))
                    printout2 = repeat_range([i],dynamic_window)

report_outfile.write("#####################################################\n" +
"#REPEAT of length " + length + " found! \n" +
                                            "#Repeated sequence found at index: " +
printout2 + "\n" +
                                            "#Sequence IDENTITY" + "\n" +"#" + window +
"\n"+
                                            "#Reverse complement sequence found at index:
" + printout + "\n" +
                                            "#Reverse Complement Sequence Identity" +
"\n" +
                                            "#" + revComp + "\n")
                    return "continue" #repeat found, keep searching for repeats in
dynamic window
    else:
        if sequence.count(window) <=1:
            return "stop" #return that there's no longer a repeat, so we should stop


#Main code... parse through sequence with sliding window

print("Running the Repeat Finder")
#user-defined input file
infile = sys.argv[1]

#user-defined window-size
input_window = int(sys.argv[2])

#user-defined report file name
report_outfile = open(sys.argv[3],'w')

#user-defined alignment file name
seq_outfile = open(sys.argv[4], 'w')
sequence = read_file(infile)


print("Finding repeat sequences and creating report. This may take a while...")
window_size = input_window
report_outfile.write("Repeats of minimum length " + str(window_size) + " in " +
str(header).strip('[').strip(']').strip("'") + "\n")


def loop(sequence,window_size,):
    for i in range(0,len(sequence),+1): #generate sliding window
        window = sequence[i:i+window_size]
        #The following is an optional progress statement. Finding repeats takes some
time, so this prints out alerts you the program is still running
        print("Parsing nucleotide " + str(i) +"/" + str(len(sequence)) + "\r")
        #Find a repeat of  the window:
        if window not in motifs: #make sure the repeat has not already been identified
and documented (prevents indentical windows)
            if len(window) == window_size: #ensure the window is the right size
                #find repeats that are multiples of the window size
                window = sequence[i:i + window_size]
                repeated(window, i,window_size)
                revComplement(window, i, window_size,window_size)
                #find repeats that are not multiples of the window size (i.e.,
window_size =3, find repeats that are 4 and 5 bp long)
                for z in range(1,window_size):
                    if (window_size+z)%window_size != 0:
```

```
                        dynamic_window = window_size+z #increase window size
                        window = sequence[i:i + window_size+z]
                        repeated(window,i,dynamic_window)
                        n = revComplement(window, i, window_size,dynamic_window)
                        if n == "stop": #no point in continuing the count if repeat of
given size isn't found (i.e., window_size = 3 (AAA), no point in seeing if repeats of
length 5 (AAATT) exist if length 4 (AAAT) doesn't exist)
                            break

loop(sequence,window_size)

#Create output FASTA file to visualize repeats through sequence alignment
print("Report complete. Generating file for sequence alignment...")
new_sequence=[] #Build list of repeated sequence bases or unknown nucleotide N where
no repeated sequence occurs
for i in range(0,len(sequence)):
    if i in indices:
        new_sequence.append(indices[i])
    else:
        new_sequence.append("N")

output_sequence = "".join(new_sequence)#put the list into a string
#Print output FASTA file
seq_outfile.write(">REPEATS "+str(header).strip('[').strip(']').strip("'") +"\n")
seq_outfile.write(output_sequence)
print("Program successfully completed.")
```

Appendix B: Index Start Site Program

```
#Find a target sequence in a larger file
import re
def counter(id):
    find = "AAGTTGACCCCGG"
    file1 = open(id,'r')
    lines = []
    for line in file1:
        if line.startswith(">"):
            header = line.strip("\n")
        else:
            line = line.strip("\n")
            lines.append(line)
    #convert sequence to string
    sequence = "".join(lines)
    #find index start site of target sequence
    index = [base.start() for base in re.finditer(find, sequence)]
    #count number of times target sequence occurs (based on length of list of all
indices returned)
    count= len(index)
    #print the data
    print id + "\t"+str(count)+"\t"+str(index)

print("ID" + "\t" + "Count" + "\t" + "Indices")
counter('ATCC-11842.fasta')
counter('ATCC-BAA-265.fasta')
counter('MN-BM-F01.fasta')
counter('ND04.fasta')
counter('ND02.fasta')
counter('DSM-20080.fasta')
counter('ACA-DC87.fasta')
counter('2038.fasta')
file1=open('ND02.fasta','r')
```

EXAMPLE OUTPUT: This data was used for the repeat finding program limiting runtime steps

ID	Count	Indices
ATCC-11842.fasta	1	[919063]
ATCC-BAA-265.fasta	1	[906452]
MN-BM-F01.fasta	1	[916088]
ND04.fasta	2	[961730, 1854707]
ND02.fasta	1	[884847]
DSM-20080.fasta	1	[901221]
ACA-DC87.fasta	1	[900146]
2038.fasta	1	[933969]

Appendix C: BLASTN Results

MN-BM-F01, 29481	MN-BM-F01,	100.00	15964	0	0	1	15964	1	15964	0.0
MN-BM-F01, 15189	DSM-20080	98.80	8560	52	13	3	8530	1	8541	0.0
MN-BM-F01, 19680	ATCC-BAA-365,	98.53	11194	74	17	1	11134	14785	25948	0.0
MN-BM-F01, 8338	ATCC-BAA-365,	98.72	4697	56	4	11270	15964	25947	30641	0.0
MN-BM-F01, 26430	ACA-DC-87	98.50	15040	126	23	974	15964	3238	18226	0.0
MN-BM-F01, 621	ACA-DC-87	99.13	346	2	1	1	345	2899	3244	2e-179
MN-BM-F01, 11574	ATCC-11842	98.61	6561	50	13	4196	10742	3854	10387	0.0
MN-BM-F01, 8844	ATCC-11842	99.05	4933	42	5	11036	15964	10911	15842	0.0

MN-BM-F01, 6658	ATCC-11842	97.79	3892	29	6	18	3889	1	3855	0.0
MN-BM-F01,	2038,	98.72	7839	48	12	2929	10742	2661	10472	0.0 13874
MN-BM-F01,	2038,	99.68	4929	16	0	11036	15964	10996	15924	0.0 9014
MN-BM-F01,	2038,	97.27	2414	20	9	1	2377	269	2673	0.0 4050
DSM-20080 15189	MN-BM-F01,	98.80	8560	52	13	1	8541	3	8530	0.0
DSM-20080 15773	DSM-20080	100.00	8541	0	0	1	8541	1	8541	0.0
DSM-20080 15173	ATCC-BAA-365,	98.69	8579	58	11	1	8541	14787	23349	0.0
DSM-20080 13439	ACA-DC-87	98.71	7597	47	9	973	8541	3238	10811	0.0
DSM-20080 603	ACA-DC-87	98.26	345	4	2	1	344	2901	3244	4e-174
DSM-20080 7747	ATCC-11842	98.90	4351	24	3	4201	8541	3854	8190	0.0
DSM-20080 6732	ATCC-11842	98.10	3889	30	5	16	3894	1	3855	0.0
DSM-20080	2038,	98.72	5629	37	7	2934	8541	2661	8275	0.0 9963
DSM-20080	2038,	97.84	2409	19	8	1	2382	271	2673	0.0 4130
ATCC-BAA-365, 19680	MN-BM-F01,	98.53	11194	74	17	14785	25948	1	11134	0.0
ATCC-BAA-365, 8338	MN-BM-F01,	98.72	4697	56	4	25947	30641	11270	15964	0.0
ATCC-BAA-365, 15173	DSM-20080	98.69	8579	58	11	14787	23349	1	8541	0.0
ATCC-BAA-365, 69551	ATCC-BAA-365,	100.00	37663	0	0	1	37663	1	37663	0.0
ATCC-BAA-365, 18809	ACA-DC-87	99.26	10429	52	4	25947	36374	13541	23945	0.0

ATCC-BAA-365,	ACA-DC-87	98.98	10197	59	11	15768	25948	3238	13405	0.0
18212										
ATCC-BAA-365,	ACA-DC-87	95.70	3208	85	20	11952	15130	61	3244	0.0
5110										
ATCC-BAA-365,	ACA-DC-87	98.57	839	11	1	36446	37284	23946	24783	0.0
1482										
ATCC-BAA-365,	ACA-DC-87	97.33	262	3	1	37363	37624	24751	25008	4e-125
442										
ATCC-BAA-365,	ATCC-11842	99.35	9938	43	9	25947	35879	11145	21065	0.0
17972										
ATCC-BAA-365,	ATCC-11842	99.24	6544	39	6	19009	25551	3854	10387	0.0
11797										
ATCC-BAA-365,	ATCC-11842	98.41	3889	27	4	14802	18689	1	3855	0.0
6806										
ATCC-BAA-365,	ATCC-11842	98.69	839	11	0	36446	37284	21744	22582	0.0
1489										
ATCC-BAA-365,	ATCC-11842	98.09	262	1	1	37363	37624	22550	22807	2e-128
453										
ATCC-BAA-365,	ATCC-11842	100.00	114	0	0	25835	25948	10896	11009	1e-055
211										
ATCC-BAA-365,	2038,	98.94	8045	64	8	25947	33988	11230	19256	0.0	14366
ATCC-BAA-365,	2038,	98.95	7831	55	6	17729	25551	2661	10472	0.0	13982
ATCC-BAA-365,	2038,	97.91	2679	32	7	14517	17177	1	2673	0.0	4615
ATCC-BAA-365,	2038,	99.63	805	3	0	34142	34946	19249	20053	0.0	1471
ATCC-BAA-365,	2038,	100.00	114	0	0	25835	25948	10981	11094	1e-055	211
ACA-DC-87	MN-BM-F01,	98.50	15040	126	23	3238	18226	974	15964	0.0
26430										
ACA-DC-87	MN-BM-F01,	99.13	346	2	1	2899	3244	1	345	3e-179
621										
ACA-DC-87	DSM-20080	98.71	7597	47	9	3238	10811	973	8541	0.0
13439										

ACA-DC-87 603	DSM-20080	98.26	345	4	2	2901	3244	1	344	1e-173
ACA-DC-87 18809	ATCC-BAA-365,	99.26	10429	52	4	13541	23945	25947	36374	0.0
ACA-DC-87 18212	ATCC-BAA-365,	98.98	10197	59	11	3238	13405	15768	25948	0.0
ACA-DC-87 5110	ATCC-BAA-365,	95.70	3208	85	20	61	3244	11952	15130	0.0
ACA-DC-87 1482	ATCC-BAA-365,	98.57	839	11	1	23946	24783	36446	37284	0.0
ACA-DC-87 442	ATCC-BAA-365,	97.33	262	3	1	24751	25008	37363	37624	3e-125
ACA-DC-87 46407	ACA-DC-87	100.00	25130	0	0	1	25130	1	25130	0.0
ACA-DC-87 18438	ATCC-11842	99.41	10173	43	9	13292	23450	10896	21065	0.0
ACA-DC-87 11795	ATCC-11842	99.27	6539	32	5	6477	13006	3854	10385	0.0
ACA-DC-87 5016	ATCC-11842	97.78	2934	20	4	3238	6170	966	3855	0.0
ACA-DC-87 2499	ATCC-11842	99.00	1396	13	1	23736	25130	21534	22929	0.0
ACA-DC-87 595	ATCC-11842	99.39	329	1	1	2916	3244	1	328	2e-171
ACA-DC-87	2038,	99.03	8280	64	8	13292	21559	10981	19256	0.0 14833
ACA-DC-87	2038,	98.98	7814	59	7	5210	13006	2661	10470	0.0 13967
ACA-DC-87	2038,	98.37	1414	14	5	3238	4647	1265	2673	0.0 2475
ACA-DC-87	2038,	99.25	805	6	0	21713	22517	19249	20053	0.0 1454
ACA-DC-87	2038,	99.19	614	4	1	2631	3244	1	613	0.0 1105
ATCC-11842 11574	MN-BM-F01,	98.61	6561	50	13	3854	10387	4196	10742	0.0

ATCC-11842 8844	MN-BM-F01,	99.05	4933	42	5	10911	15842	11036	15964	0.0
ATCC-11842 6658	MN-BM-F01,	97.79	3892	29	6	1	3855	18	3889	0.0
ATCC-11842 7747	DSM-20080	98.90	4351	24	3	3854	8190	4201	8541	0.0
ATCC-11842 6732	DSM-20080	98.10	3889	30	5	1	3855	16	3894	0.0
ATCC-11842 17972	ATCC-BAA-365,	99.35	9938	43	9	11145	21065	25947	35879	0.0
ATCC-11842 11797	ATCC-BAA-365,	99.24	6544	39	6	3854	10387	19009	25551	0.0
ATCC-11842 6806	ATCC-BAA-365,	98.41	3889	27	4	1	3855	14802	18689	0.0
ATCC-11842 1489	ATCC-BAA-365,	98.69	839	11	0	21744	22582	36446	37284	0.0
ATCC-11842 453	ATCC-BAA-365,	98.09	262	1	1	22550	22807	37363	37624	1e-128
ATCC-11842 211	ATCC-BAA-365,	100.00	114	0	0	10896	11009	25835	25948	7e-056
ATCC-11842 18438	ACA-DC-87	99.41	10173	43	9	10896	21065	13292	23450	0.0
ATCC-11842 11795	ACA-DC-87	99.27	6539	32	5	3854	10385	6477	13006	0.0
ATCC-11842 5016	ACA-DC-87	97.78	2934	20	4	966	3855	3238	6170	0.0
ATCC-11842 2499	ACA-DC-87	99.00	1396	13	1	21534	22929	23736	25130	0.0
ATCC-11842 595	ACA-DC-87	99.39	329	1	1	1	328	2916	3244	2e-171
ATCC-11842 42636	ATCC-11842	100.00	23088	0	0	1	23088	1	23088	0.0

ATCC-11842	2038,	99.26	15334	95	11	3854	19174	3928	19256	0.0	27675
ATCC-11842	2038,	97.63	2366	33	6	1	2348	286	2646	0.0	4037
ATCC-11842	2038,	99.79	961	2	0	2895	3855	2661	3621	0.0	1764
ATCC-11842	2038,	99.75	805	2	0	19328	20132	19249	20053	0.0	1476
2038,	MN-BM-F01,	98.72	7839	48	12	2661	10472	2929	10742	0.0	13874
2038,	MN-BM-F01,	99.68	4929	16	0	10996	15924	11036	15964	0.0	9014
2038,	MN-BM-F01,	97.27	2414	20	9	269	2673	1	2377	0.0	4050
2038,	DSM-20080	98.72	5629	37	7	2661	8275	2934	8541	0.0	9963
2038,	DSM-20080	97.84	2409	19	8	271	2673	1	2382	0.0	4130
2038,	ATCC-BAA-365,	98.94	8045	64	8	11230	19256	25947	33988	0.0	14366
2038,	ATCC-BAA-365,	98.95	7831	55	6	2661	10472	17729	25551	0.0	13982
2038,	ATCC-BAA-365,	97.91	2679	32	7	1	2673	14517	17177	0.0	4615
2038,	ATCC-BAA-365,	99.63	805	3	0	19249	20053	34142	34946	0.0	1471
2038,	ATCC-BAA-365,	100.00	114	0	0	10981	11094	25835	25948	6e-056	211
2038,	ACA-DC-87	99.03	8280	64	8	10981	19256	13292	21559	0.0	14833
2038,	ACA-DC-87	98.98	7814	59	7	2661	10470	5210	13006	0.0	13967
2038,	ACA-DC-87	98.37	1414	14	5	1265	2673	3238	4647	0.0	2475
2038,	ACA-DC-87	99.25	805	6	0	19249	20053	21713	22517	0.0	1454
2038,	ACA-DC-87	99.19	614	4	1	1	613	2631	3244	0.0	1105
2038,	ATCC-11842	99.26	15334	95	11	3928	19256	3854	19174	0.0	27675
2038,	ATCC-11842	97.63	2366	33	6	286	2646	1	2348	0.0	4037
2038,	ATCC-11842	99.79	961	2	0	2661	3621	2895	3855	0.0	1764
2038,	ATCC-11842	99.75	805	2	0	19249	20053	19328	20132	0.0	1476
2038,	2038,	100.00	20053	0	0	1	20053	1	20053	0.0	37031

Appendix D: Genbank Annotation Finding Program

```
list = [] #list that contains annotations
with open("initial.txt", 'r') as infile: #infile of annotation text

    #The following startswith queries are taken from base annotation data on genebank
    for line in infile:
        if line.startswith("Query="):
            print(line)
        elif line.startswith(" Identities "):
            print(line)
        elif line.startswith(">"):
            print(line)
            print next(infile)
        elif line.startswith("                     /product"):
            line2 = line.strip('\n')
            line = line2.replace("                         /product="","")
            line2 = line.strip('"')
            if "hypothetical protein" in line2:
                list.append("hypothetical protein")
            else:
                list.append(line2)
            print line
        elif line.startswith("                     /pseudo"):#pseudogenes follow
different annotation format
            print "pseudo"
            list.append("pseudogene")


from collections import Counter #count number of times a gene appears in the list
print Counter(list) #print out the final counts
```