From genomes to metagenomes: Development of a rapid-aligner for genome assembly and application of macroecological models to microbiology

by

Angus S. Hilts

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Science
in
Biology

Waterloo, Ontario, Canada, 2019

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Since the development of the modern computer, many scientific fields have undergone paradigm shifts due to an increasing facility in data collection and analysis. Microbiology has been impacted by computational advances, especially in DNA sequencing applications, and this has led to an interesting problem: there is too much raw data for any person to understand. It is important to have tools that are able to process and analyze these vast amounts of data, so that microbiologists can robustly test hypotheses and predict patterns.

Long-read sequencers are capable of sequencing entire genomes with very few reads, but exhibit much higher error rates compared to short-read sequencing platforms. Most current genome assemblers were developed for highly accurate short-read data, and so there is a need to build new tools that can handle these long, error-filled reads. Here, we developed an alignment algorithm in the C programming language for error-prone long reads, as part of a larger genome assembler. This alignment algorithm creates a profile of ordered kmers representing all of the reads, then clusters these kmers to generate a consensus sequence. We show that the alignment algorithm can handle long-read error rates and produce useful results. Using a low-coverage test data set, the algorithm was able to produce a consensus sequence with 85.3% identity to a reference sequence built with extremely high coverage data. Future work will aim to improve this accuracy by error correcting kmers and identifying close repeats of kmers.

The field of metagenomics is entering a new state of maturation. Isolation of total community DNA, shotgun sequencing, and assembly of draft genomes for populations has become standard practice in many microbial ecology labs, and many pipelines for manipulating metagenomic sequence data exist. What is not as well understood, however, is how to analyze the growing databases of metagenomic datasets with statistical rigour. To examine the relationships and interactions of different groups of microorganisms across the planet requires strong statistical models that can be used to assess hypotheses. We borrowed occupancy modelling from the macroecological toolbox, and adapted it to microbial metagenomic datasets. Occupancy models are designed to assess the occupancy states of sample sites, while accounting for possible missed detections by re-sampling these sites. We emulate re-sampling by searching for multiple genes associated with functions of interest, where each gene is considered an independent sampling event. We use detection of these genes as proxies for presence of functional potential within environments, and can assess occurrence and, importantly, co-occurrence patterns. We applied this method to nearly 10,000 metagenomes to assess global occupancy patterns for methanogens

and methanotrophs, key contributors to the methane cycle. To assess the occupancy patterns of methane cyclers, we looked for genes encoding the subunits for the methyl coenzyme M reductase complex (MCR) and the methane monooxygenases (MMO), biological markers of methanogenesis and methanotrophy, respectively. Our models predicted that occupancy probabilities for both functional groups changed with ecosystem type, latitude, and the date that the data were deposited to the database. The explanatory power of the models was relatively low, which is likely due to a lack of metadata that could be used to better inform models. Occupancy models have the potential to be powerful tools, but microbial ecologists will need to embrace better standards for metadata collection and reporting for metagenomes. This metadata could include the collection of data such as pH, temperature, and other key environmental factors. Future work should focus on establishing and enforcing these metadata requirements to enable statistical assessment of functionally important groups across environments.

Acknowledgements

It is difficult to acknowledge all of the many people who have been a help to me throughout this project, but what can certainly be said is that without their support, both professionally and emotionally, I could not have finished it. Thank you to everyone who has contributed in any way, whether big or small.

I would first like to thank my committee members, Dr. Brendan McConkey who I have worked closely with to develop the software as part of this project, and Dr. Kirsten Müller, for her continued guidance since I first took her class during my undergraduate degree.

I would also like to thank my lab members, past and present: Veronica Viljakainen, Alex Sauk, Lisa Johnson, Grant Jensen, Kira Goff, and all of our past undergraduate researchers. I would especially like to thank Rebecca Co and Nikhil George, whom I spent many late nights and early mornings in the lab with, having long scientific discussions. Thank you to Manjot Singh Hunjan for his time volunteering, your work and enthusiasm was a huge help.

Dr. Heidi Swanson and Jared Ellenor were excellent resources for suggesting and helping with my understanding of occupancy modelling. I am extremely grateful for the time you spent answering my many questions. On more than one occasion, your advice was absolutely necessary to move the project forward.

I would also like to extend my sincere thanks to the many members of the department, who constituted a wonderful community, which I have been proud to call myself a member of these past years. Your support and the many hours of discussing our shared passions was vital to my learning during this degree.

Thank you to my supervisor, Dr. Laura Hug. I am so very thankful that you welcomed me to your lab as your first student and cannot believe that three years has already gone past. You have been a wonderful and patient teacher, and I could not have asked for a better supervisor in starting out my graduate career.

Finally, I would like to thank my friends and family. All of you have been the foundation on which I have built myself. Thank you, Mom, Dad, Molly, and Emily, your love and support has been the one constant in life that I know I am always able to turn to, no matter how challenging things may seem.

Dedication

For my grandfather, Angus Hilts Sr., who taught me the value of hard work, the importance of always being our best selves, and to always share the things we have learned.

Table of Contents

List of Figures

List of Tables

Code Listings

List of Abbreviations

aa	amino acid
AIC	Akaike information criterion
AOM	anaerobic oxidation of methane
B	bytes
b	bits
BLAST	basic local alignment search tool
bp	base pairs
CSV	comma-separated values
DBG	de Bruijn graph
ddATP	dideoxyadenosine triphosphate
ddCTP	dideoxycytidine triphosphate
ddGTP	dideoxyguanosine triphosphate
ddNTP	dideoxyribonucleoside triphosphate
ddTTP	dideoxythymidine triphosphate
DNA	deoxyribonucleic acid
dNTP	deoxyribose nucleoside triphosphate
GB	gigabytes (~1 billion bytes)
GHG	greenhouse gas
%id	percent identity
IMG/M	Integrated Microbial Genomes and Metagenomes (see JGI)
indel	insertion/deletion (mutation)
JGI	Join Genome Institute
Kbp	kilobase pairs (i.e. thousand)
KEGG	Kyoto Encyclopedia of Genes and Genomes
KO	KEGG orthology
MAG	metagenome assembled genome
Mbp	megabase pairs (i.e. million)
mcr	methyl-coenzyme M reductase (referring to the gene)
MCR	methyl-coenzyme M reductase (referring to the enzyme)
mmo	(soluble) methane monooxygenase (referring to the gene)
Mt CO_2 eq.	metric tonnes of carbon dioxide equivalents.
NCBI	National Centre for Biotechnology Information
NGS	next-generation sequencing
NHGRI	National Human Genome Research Institute
OLC	overlap layout consensus
ONT	Oxford Nanopore Technologies
PacBio	Pacific Biosciences
PCR	polymerase chain reaction
pmo	particulate methane monooxygenase (referring to the gene)
pMMO	particulate methane monooxygenase (referring to the enzyme)
ppb	parts per billion
ppbv	parts per billion volume
RAM	random access memory
sMMO	soluble methane monooxygenase (referring to the enzyme)
SMRT	single-molecule real time
sqrt	Square root
ssDNA	single stranded DNA

Chapter 1 Introduction:
The unexpected problem of 'too much data'

In the more than eighty years since Alan Turing first fathered the idea of the modern computer (Turing, 1937), humankind has increased its capacity to collect and analyze data at an unprecedented rate. In 1965, Gordon E. Moore attempted to quantify this growth in computing power, stating that it would as much as double every year – a metric now known as Moore's Law – which has often been extended to decreasing cost and has generally held true since. The momentum of these computational developments has dramatically changed approaches to nearly every scientific field and has led to a problem that has, arguably, never been faced before: there is too much available data to be effectively analyzed by humans. An area that has encountered this limit, perhaps more than any other discipline, are the biological sciences.

In 1977, Sanger *et al.* (1977) pioneered DNA sequencing, bringing biology into the information era. By 1995, the first bacterial genome, that of *Haemophilus influenzae,* was sequenced (Fleischmann *et al.*, 1995), with the first eukaryotic genome, from the yeast *Saccharomyces cerevisiae*, following in 1996 (Goffeau *et al.*, 1996). By 2001, a draft human genome had been completed (The International Human Genome Mapping Consortium, 2001; Venter *et al.*, 2001). In the 18 years since, the National Centre for Biotechnology Information (NCBI) has served as a repository to more than 45,000 genomes. In addition to this explosion in the amount of sequence data, the cost of obtaining a genome sequence has decreased at an astounding rate. The National Human Genome Research Institute (NHGRI) has estimated that the cost of the first draft human genome was 300 million U.S. dollars. The NHGRI estimated that in 2016, the cost to generate a 'draft' human genome fell below $1,500. This has far outpaced the predictions of Moore's Law.

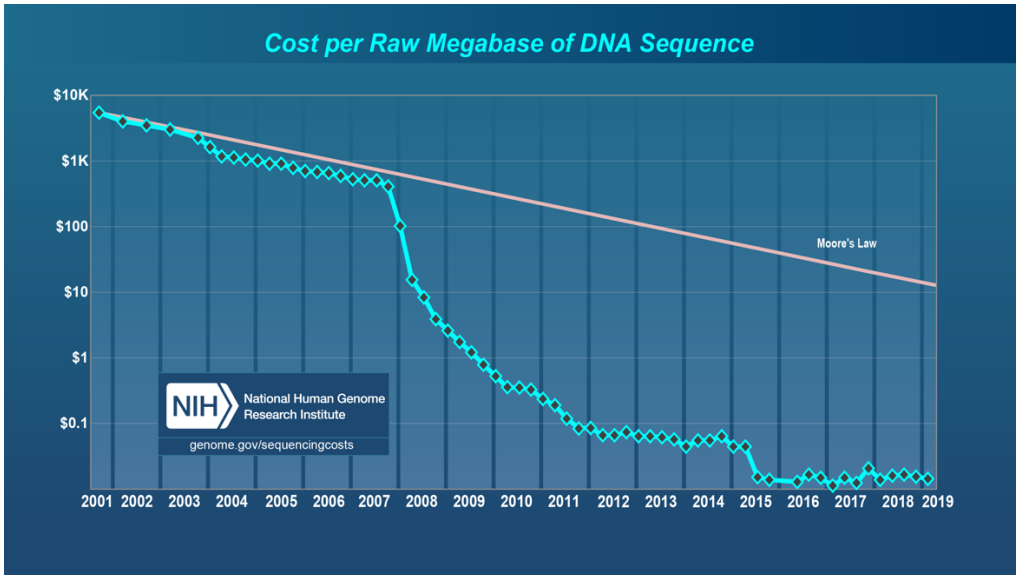

Figure 1.1: Image from the National Human Genome Research Institute (DNA Sequencing Costs: Data) showing the cost of sequencing over the past two decades in comparison to the pace predicted by Moore's Law (https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost).

One rapidly maturing field derived from recent advances in sequence data collection is metagenomics, where total DNA for entire microbial communities is retrieved from an environment, sequenced, and studied, without the need for culturing or applying other labour-intensive methods. This has led to an appreciation that the diversity of life on Earth is far greater than was previously believed (Brown *et al.*, 2015; Castelle *et al.*, 2015; Spang *et al.*, 2015; Hug *et al.*, 2016). Computational techniques (see, for example, Namiki *et al.*, 2012; Peng *et al.*, 2012; Kang *et al.*, 2015; Nissen *et al.*, 2018; Sieber *et al.*, 2018) have been developed to reconstruct high quality draft genomes from metagenomic datasets, without ever having isolates or even cultures of the organisms from which they originated (Tyson *et al.*, 2004; Parks *et al.*, 2017; Stewart *et al.*, 2018). Furthermore, metagenomic datasets allow metabolic potential to be explored across different environmental conditions, providing a more complete view of important geochemical cycles at global or local scales.

All of this leads to an important question: how can researchers derive meaning from the vast amounts of data now available? Genomes are complex and very poorly optimized for human readability, and metagenomes are orders of magnitude more complex, as assemblages of total community DNA. It is evident that we need new methods, or at least methods new to microbiology, capable of analyzing these complex, large-scale datasets.

A primary requirement arising from new sequencing technologies is the ability to manipulate and assemble raw sequence data. Current technical limitations mean that DNA sequencing platforms cannot produce full contiguous genome sequences on their own. Sequencing end products are, instead, reads: short fragmented sequences representing parts of the input DNA. These reads then need to be ordered and overlapped in order to generate assembled sequences, representative of the entire genome or genomes from which the initial DNA was derived. This assembly step typically involves complex algorithms with clever heuristics, which have largely been developed for short reads with low error rates. As sequencing hardware evolves, so too must the algorithms that handle the raw data. The development of long-read sequencing platforms, primarily by Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT), has necessitated a revisiting of assembly algorithms. These new sequencing platforms are remarkable, in that they can produce incredibly long reads (*e.g.,* on the order of Kbp as opposed to the 150-300 bp read lengths from the popular Illumina platform; Sequencing Read Length: How to calculate NGS read length). However, these long-read sequencing platforms suffer from high error rates, which older assembly algorithms are not designed to handle (Laver *et al.*, 2015; Rhoads and Au, 2015).

Beyond the ability to process and assemble genomic data, there remains the question of how to derive meaning from this information. Computer scientists and biologists have developed many tools to answer biologically relevant questions, such as predicting the locations of genes within a genome using Prodigal (Hyatt *et al.*, 2010), or identifying conserved sequences with multiple sequence alignments or hidden Markov models (Eddy, 1998). In metagenomics, binning algorithms have been used with great success to reconstruct genomes for organisms that have not been cultured (Wu *et al.*, 2014; Kang *et al.*, 2015; Lu *et al.*, 2016; Parks *et al.*, 2017). These tools allow important insights to be derived from genomic and metagenomic datasets, but there remain important questions that need to be addressed. In particular, methods for statistically assessing patterns across multiple large sequence datasets are lacking (Calle, 2019). There is a need for new ideas, or for adaptation of existing statistical tools to these data, especially for analyzing the patterns and relationships of different functionally important groups across many metagenomic samples.

Macroecologists have worked with a variety of questions concerning the relationships of organisms, or populations thereof, to one another over the past century. Through directed study, this has led to a wealth of models ranging from simple to complex (the founding of the journal Ecological Modelling in 1975 is a testament to this focus). In addition, since the 1960's there has been a movement in macroecology to develop and enforce standards for metadata collection and quality assurance (Michener, 2015). This

means that much of the work of developing ecological models is already in place, but must be adapted to fit microbial datasets.

The research presented in this thesis addresses two current themes in microbiology and bioinformatics: (1) working with current and new sequencing technologies' outputs and (2) developing new tools to answer biologically important questions using large metagenomic datasets. The thesis is organized into two data chapters, which each examine one of the aforementioned bioinformatic problems.

In chapter 2, I present a new approach to genome assembly targeting long read sequence data. The goal in developing this assembler was to enable good quality assemblies, in spite of relatively erroneous reads. While the assembler was developed to be sequencing-platform agnostic, it has been specifically designed to address the challenges associated with long-read data. The genome assembler works in two steps: the reads are ordered and assigned an overlap (this part was designed and built by Dr. Brendan McConkey), and the second step, designed by myself, uses this mapping information to align and build a final consensus sequence. One of the key advantages to our approach is the ability to generate a consensus sequence in spite of the error rate associated with current long read technologies, a problem that current assemblers, built for high-quality short-reads, specifically struggle with. Most current assemblers build a mathematical structure called a de Bruijn graph, and then find an Eulerian cycle through the graph (Compeau *et al.*, 2011). Put more plainly, this means that each read is broken into a series of "kmers" (stretches within the read that are k letters long), and kmers are connected if they overlap one another. Then a path that visits every connection exactly once is found, and traced, producing a final assembly that represents all of the reads (Li *et al.*, 2012). The problem is that errors in the sequence reads can yield incorrect connections, which results in multiple Eulerian circuits, of which only one is correct. This issue means that the platform producing the raw reads needs to be highly accurate (Brown and Morgenstern, 2014). Current popular sequencing platforms, namely the Illumina series of sequencers, are able to achieve the required accuracy for de Bruijn graph assemblers to work (Brown and Morgenstern, 2014, suggest an error rate <2%). In contrast, sequencing platforms produced by Oxford Nanopore Technologies are capable of very long reads (up to millions of base pairs; Payne *et al.*, 2018), but their high error rates (>10%) mean that de Bruijn graph assembly is no longer a well-tailored solution. Our goal was therefore to produce an assembler that could work on any data, including a combination of long and short read data, and still yield high quality assemblies. This assembler would offer many advantages. The strengths of different sequencing platforms could be integrated. In addition, cheaper long read

platforms could be used in place of Illumina sequencing and still produce good quality genomes, at significant cost savings for the researcher.

In chapter 3, I present an adaptation of occupancy models from their use in macroecology (MacKenzie *et al.*, 2002), where they are used to assess which sites species occupy, to use metagenome-derived data as input and demonstrate the ability to assess occurrence patterns of ecologically important functions. In this work, I sought to address an important question: how can we assess the occurrence patterns of organisms, functions, or groups using metagenomic data, while accounting for our detection ability? This is an issue especially important for metagenome datasets, where sequencing errors, incorrect annotations, or a failure to fully isolate all DNA from a sample could lead to missing data. In macroecology, occupancy models are often used to simultaneously assess detection probabilities and occurrence patterns. Occupancy models rely on re-sampling a site on multiple occasions for the organism(s) of interest, which in turn informs detection probabilities. In metagenomics, physically resampling a site is often not a viable option. We wondered if we could leverage the large amount of data contained within each individual metagenome to emulate site resampling. Here, I present a method for applying occupancy modeling to microbial metagenomics, with a specific focus on the co-occurrence patterns of organisms contributing to the global methane cycle. Briefly, my approach uses functional components (*e.g.*, enzyme subunits) as independent measurements of the function of interest. Each of these components represents a sampling of the dataset, akin to repeated site sampling in macroecological studies. This approach has the potential to be a valuable tool for microbial ecologists, helping to assess detection of important functions or organisms, identifying occurrence patterns, and studying co-occurrence.

Chapter 2

Development of an algorithm for rapid alignment of long read sequence data

2.1 Introduction

The ability to sequence DNA is arguably the most important recent advance in biology. To date, however, sequencing platforms are generally not capable of producing full length genomes. This leads to a requirement for assembly algorithms to reconstruct fragmented genomes based on the sequencer output.

2.1.1 Sanger Sequencing

The first major method used to sequence DNA was published in the 1970's by Sanger, *et al.* (1977). The idea was an elegant solution built on relatively simple experimental procedures. Copies of the DNA fragment of interest were synthesized in four pools. Each of the pools contained the four normal deoxynucleotides (dNTPs), but in addition to this, each pool also contained one of four dideoxynucleotides (ddATP, ddGTP, ddTTP, or ddCTP) that would terminate synthesis of the new strand. This synthesis generated fragments of varying lengths where the terminal nucleotide of each sequence was known. The four sets of fragments could then be run on four adjacent lanes of a polyacrylamide gel to separate by length. By "reading" the bands in order from shortest to longest, the sequence could be determined (*i.e.,* if the shortest band came from the ddATP pool, then the first nucleotide must be A).

The use of fluorophores was pioneered by Smith *et al.* in 1986, which allowed sequencing to be automated, greatly increasing the throughput, while still predicated on the same sequencing technology. By the early 2000's, automated sequencers could produce reads of about 400 bp with average accuracies over 99% (Paegel *et al.*, 2002). While Sanger-based methods provided good accuracy and fair read lengths, the development of the so-called "next generation sequencers" (NGS) made Sanger sequencing comparatively slow and expensive.

2.1.2 Next generation sequencing

For the past decade, NGS platforms have dominated sequencing applications, with the Illumina platforms being the most widely adopted method for sequencing at this time. NGS platforms largely eclipsed Sanger sequencers because of their ability to produce far more data per unit of time invested, while also being much cheaper per kilobase of DNA sequenced. The first innovation was sequencing by synthesis, first described in 1993 as pyrosequencing (Nyren *et al.*, 1993). 454 Life Sciences developed a pyrosequencing platform in 2005, the first commercially available next generation sequencer (Margulies

et al., 2005; Shendure *et al.*, 2017). On average, reads were between 80 and 120 bp, with an average accuracy of 96% (Margulies *et al.*, 2005). Around this time, Solexa sequencing was developed at Cambridge, which could produce paired 35 bp reads, with the first Solexa sequencer being released in 2006 (History of Illumina Sequencing and Solexa Technology; illumina.com). This method was shown to be highly accurate when used for detection of single nucleotide polymorphisms (SNPs; Bentley *et al.*, 2008) and would eventually become what is now colloquially known as Illumina sequencing.

All of the NGS methods, which here will refer to sequencing by synthesis, take a similar approach to sequencing. The input DNA is fragmented and mounted with an adapter to a 2D surface. Typically, the DNA is then amplified, using various methods (Adams and Kron; Mitra, 1999), although there are technologies that are capable of performing NGS with single molecules, forgoing the need for amplification (Braslavsky *et al.*, 2003). At this point, DNA is synthesized against the mounted DNA template strands by incorporating modified nucleotides, which either causes a fluorescent pulse directly, or has fluorescence induced by some other means, as each individual nucleotide is incorporated into the growing strand (Ronaghi *et al.*, 1996; Braslavsky *et al.*, 2003; Mitra *et al.*, 2003; Shendure, 2005). The fluorescent signal can be measured and interpreted computationally to identify each base in the sequence (base calling). Reads produced by NGS platforms have a high accuracy (upward of 96-99.9%; see Table 2.1), but the fluorescent signal quickly accumulates noise. This ultimately limits the lengths of reads (*e.g.,* to under 300 bp for most modern Illumina platforms). In addition to the limitation in read length, the amplification step introduces bias in the coverage of different regions of the input DNA, largely driven by GC content (Chen *et al.*, 2013). In practice, downstream assembly of the resultant reads is impacted more heavily by the short read lengths, so this problem will be the focus here.

2.1.3 Long read sequencing

Long read sequencing (sometimes referred to as 3rd generation sequencing) has been an enduring goal in sequencing technology research. In theory, the best sequencer would be capable of sequencing single DNA molecules in their entirety, eliminating the need for genome assemblers. Technologies have made great strides with respect to read length and, in the past decade, several long read platforms have been launched commercially, and have rapidly grown in popularity.

During the early 2000's, tools were developed that could be used to look at the dynamics of single molecules, including DNA polymerases (Levine, 2003; Eid *et al.*, 2009). These tools gave rise to PacBio's Single Molecule Real-Time sequencing (SMRT). Polymerases are loaded into small detection chambers, called Zero Mode Waveguides, which act as a sort of microscope. The polymerases are loaded

along with a primer and the DNA to be sequenced. The polymerase incorporates fluorescently labelled nucleotides one at a time. The resulting fluorescence is measured and interpreted as a sequence (Ardui *et al.*, 2018; see Video: Introduction to SMRT Sequencing). The throughput is much lower than Illumina platforms, and the error rates are estimated to be about 10% (although randomly distributed, a useful property for assembly; Ross *et al.*, 2013), but reads associated with this SMRT technology have reached lengths of hundreds of kilobases (Shendure *et al.*, 2017).

The other major long read technology that was developed, and which will be the focus here, is nanopore sequencing. This technology uses protein nanopores embedded in a membrane. Single molecules are fed through the pore, which causes a voltage change across the membrane. This can be measured and interpreted as a kmer (typically a stretch of 6 nucleotides; Branton *et al.*, 2008; Deamer *et al.*, 2016; Shendure *et al.*, 2017). Nanopore technology is capable of incredibly long reads. Jain *et al.* (2018) achieved reads around 900 Kbp in length, and reads on the order of millions of base pairs have also been reported (Payne *et al.*, 2018). In theory, the MinION, the most common nanopore sequencer, sold by Oxford Nanopore Technologies (ONT), has no limit on read length, other than the quality of the input DNA. This means that improved DNA extraction protocols may further increase read lengths beyond the already impressive current results. In addition to the advantages already described, the MinION is extremely small, weighing only 87g. This makes it an excellent option for both in-lab sequencing projects and, uniquely, field projects. The MinION has even been used to sequence and assemble a genome on the International Space Station (Castro-Wallace *et al.*, 2017). The throughput is also relatively high, with each flow cell, the disposable component needed for each sequencing run, being capable of 10-30 Gbp of sequence data. Finally, it is inexpensive, costing only about $1,000 to purchase the platform with the materials needed to sequence several runs (at the time of writing; see https://nanoporetech.com/products/minion).

Nanopore sequencing does have several disadvantages. Like the SMRT platforms developed by PacBio, the MinION suffers from much higher error rates than NGS platforms. These error rates tend to be on the order of 10%, and are non-random, which can make identifying them much harder (Rang *et al.*, 2018), since there is no way to distinguish a systematic error from a correct call. The majority of these errors are indels, which is due to the static signals produced by homopolymers (O'Donnell *et al.*, 2013). These non-random errors derive from the statistical models used to determine which kmer the signal matches. With long homopolymers, the signal does not change as the DNA moves through the pore, and the models may not be able to determine the length of the homopolymer. Two-dimensional (2D)

sequencing has been employed to reduce errors, which works by attaching a hairpin adapter and sequencing both forward and reverse strands. This technique has been used to obtain accuracies of 97% (Tyler *et al.*, 2018). Better base calling software has also greatly improved read accuracy (Wick *et al.*, 2019). Nonetheless, nanopore error rates are still relatively high for the more common assembly algorithms to handle, and it is not clear if these systematic errors will prove correctable (Rang *et al.*, 2018). Even with relatively high error rates, the long reads from nanopore sequencing have the potential to greatly reduce the computational power needed to assemble genomes, as they can greatly simplify the process of ordering and orienting genome fragments during genome assembly.

Table 2.1: Summary of statistics for various modern sequencing platforms' output, including the NGS Illumina platform and the two major long read platforms, the MinION and the PacBio RSII. Where not denoted, information was retrieved from the manufacturer's website (www.thermofisher.com; www.illumina.com; www.pacb.com; nanoporetech.com)

Sequencing Platform	Sequencing yield (per run/flow cell)	Time for run	Estimated error rate[1]	Read length	Platform release
Sanger Sequencing[2]	<2.8 Mbp[3]	< 1 day[3]	<0.001%[4]	<1,000bp[4]	1988
Illumina MiSeq	<15 Gbp	4-55 hours	<1%	<2x300bp	2011
PacBio RSII (per SMRT cell)	<160 Gbp	0.5-6 hours	13-15%[5]	10-16 Kbp	2013
MinION Mk1B (per flow cell)	30 Gbp	1 min – 48 hours	~15%[6]	> 2 Mbp[7]	2015

[1] Error rates should be interpreted with some caution, since sequencers produce some signal that must be programmatically analyzed and converted to a nucleotide sequence. This means that improvements in base calling software could improve accuracy without actually altering the sequencing platform

[2] Numbers reported are for current Sanger platforms

[3] Numbers for Thermo-Fisher Scientific's 3730 Series Genetic Analyzer

[4] Victoria *et al.*, 2012

[5] Ardui *et al.*, 2018

[6] Rang *et al.*, 2018

[7] There is no theoretical limit to MinION read length, see Payne *et al.* (2018) for the current record

2.1.4 Genome assembly

Genome assembly is a key step in the reconstruction of genomes from raw sequence reads. Without it, we would have only short, unordered, fragments of the genomes being sequenced, and the overarching architecture of an organism's genetic material could not be studied. Genome assembly leverages a fundamental property of DNA sequencing; many copies of the genome are sheared at random locations and then sequenced as part of the normal protocol. From this, common regions of reads can be overlapped

to determine the underlying order of sequence and architecture of the genome (Kent, 2001; Myers Jr, 2016).

In the 1980s, the first assembly algorithms were developed, which were based mostly on sequence alignment, and required human curation (Sanger *et al.*, 1982). In 2000, the Celera assembler was developed and used to reconstruct the genome of *Drosophila melanogaster* (Adams, 2000). This paved the way for future assemblers, capable of assembling other large genomes. These assemblers fell, largely, into two classes: de Bruijn graph (DBG) assemblers and overlap layout consensus (OLC) assemblers (Staden, 1979; Miller *et al.*, 2010; Li *et al.*, 2012), of which the Celera assembler was one of the earliest. OLC assembly was developed with long reads in mind, before the development of NGS platforms, while DBG assemblers were tailored for NGS platforms. OLC assemblers required too much memory to effectively assemble a genome given the number of reads required from NGS platforms, and so DBG assembly became the dominant algorithm for most genomics applications once NGS sequencers became the standard (Miller *et al.*, 2010; Compeau *et al.*, 2011).

At the heart of OLC assembly is the identification of overlapping reads. Overlaps are added to a mathematical structure called a graph. A graph consists of vertices (sometimes called nodes) connected by edges. These edges can either be directed (*i.e.,* the edge leaves one node and goes to another), or undirected (*i.e.,* the edge can be traced in either direction). For genome assembly, each vertex represents a read from the sequencer. Two vertices are connected if their associated reads overlap. In practice, this step is computationally challenging, since all pairs of reads need to be compared. The number of comparisons for n reads is n^2, which becomes quite large, quite quickly (Compeau *et al.*, 2011). In addition to this problem, the size of the graph can be extremely large if there are many reads. This is the key reason that OLC assembly is problematic with short reads; a genome requires far more reads for a complete assembly if the reads are shorter. Once the graph has been built, a Hamiltonian path is identified. A Hamiltonian path is one which walks along the edges of the graph and visits each node exactly once. If a path cannot be found, then a best effort is made, and the graph is refined by combining reads into larger contigs. This problem is NP-complete, where NP stands for “nonpolynomial deterministic”, meaning that an efficient algorithm to find a Hamiltonian path in a graph likely does not exist (Korte and Vygen, 2008). This means that OLC algorithms must either be used for small datasets, or use heuristics to solve the problem. Instead of identifying a Hamiltonian path, an Eulerian path can be identified, a problem which is much easier to solve (Pevzner *et al.*, 2001). This is a similar, except that the path must instead visit each edge exactly once, as opposed to each vertex (Euler, 1741). Adaptation of the

Eulerian path problem led directly to the de Bruijn graph assemblers described below. The final step in OLC assembly involves building a consensus sequence from the final path.

Assembling genomes with de Bruijn graph assembly also involves building a graph, but each node represents a prefix (*i.e.,* the first *k*-1 letters) or a suffix (*i.e.,* the last *k*-1 letters) of a kmer. Then an edge from a vertex representing the prefix to a vertex representing the suffix can be thought of as a directed edge which represents a kmer (*e.g.,* the prefix ATGC and the suffix TGCT would represent two vertices and their adjoining edge would represent the 5-mer ATGCT). These graphs, in which vertices represent overlaps of sequences of symbols, are called de Bruijn graphs, named after N. G. de Bruijn, who proposed the idea (de Bruijn, 1946). Euler proved various properties of Eulerian cycles, and methods to identify them that laid the foundation for efficient algorithms to find such cycles in graphs, which would include de Bruijn graphs, when they were developed (Euler, 1741; Pevzner *et al.*, 2001; Compeau *et al.*, 2011). For DBG assembly, the raw sequence data are divided into kmers and converted to a de Bruijn graph and a path through the graph is identified, which represents the sequence (Medvedev, 2018), and from this the consensus is derived. Compared to OLC graphs, de Bruijn graphs have a smaller memory footprint and thus can be applied to assemblies with higher numbers of reads. This is a key advantage over OLC assemblers, especially for NGS short reads.

Both OLC and DBG algorithms fail to solve several key issues with short read data, including (1) a single read overlapping with multiple other, non-identical reads, giving multiple possible paths forward; and (2) a read failing to span a repeat region, and making resolution of placement or length of tandem repeats challenging. These problems largely stem from the fact that genomes often contain tandem repeats (Epplen *et al.*, 1993; Mojica *et al.*, 1995; Usdin, 2008; Zhou *et al.*, 2014) or interspersed repeats (Lupski and Weinstock, 1992; Koeuth *et al.*, 1995; Smit, 1996; Achaz, 2002; Jansen *et al.*, 2002). In the case of tandem repeats, it is difficult to determine how many times the repetitive element occurred in the original genome without the use of extra information. Interspersed repeats pose problems since they may be collapsed into a single vertex of the assembly graph, which could fragment the assembly or cause an incorrect path to be identified as the consensus sequence through the graph. With short-read technology, extra information, such as coverage, is usually required to resolve these segments of the assembly (Treangen and Salzberg, 2012). These solutions, however, are not perfect. For example, the use of coverage may be biased, as the amplification step can cause biases related to the GC content of different regions of the genome, meaning that the necessary assumption that the genome is equally covered may not be true (Chen *et al.*, 2013).

Many of the problems associated with repeat regions and genome assembly are solved, or much less impactful, when long reads are used. Consider OLC assembly, for example: the longer overlaps made possible by long reads are much less likely to occur by chance. While repeat regions are not always perfectly solved by long-read sequencing, long-read data is certainly capable of improving repeat resolution. The *Caenorhabditis elegans* genome, for example, was expanded by more than 2 Mbp when re-sequenced using long reads, largely due to the ability of long reads to resolve repeat regions (Tyson *et al.*, 2018). Furthermore, a genome assembly required lower coverage, and thus fewer reads, when the reads are very long, which simplifies the graphs involved in assembly.

The main disadvantage of long reads is their significantly higher error rate. This makes building assembly graphs much more challenging, since overlaps cannot be precisely determined. In addition, a high error rate could introduce more, erroneous, kmers to a de Bruijn graph, which may greatly complicate the task of identifying the correct path through the graph (Lin *et al.*, 2016; Kamath *et al.*, 2017). These issues do exist when assembling data from NGS platforms, but between baseline accuracy and error correction tools, current assemblers are still able to produce high quality assemblies (Heydari *et al.*, 2017). OLC graph assembly of long reads can also be great complicated, with sequence errors inducing missing edges in some cases, and extra edges in others (Kamath *et al.*, 2017). Furthermore, homopolymer errors are quite common with long read sequencing, particularly the MinION platform (Tyler *et al.*, 2018), and with different homopolymer lengths being base called, it can be challenging to determine which is correct.

With the increasing popularity and availability of long reads, assembly methods have been developed to account for higher error rates. These fall into two major classes: hybrid and pure long read *de novo* assemblers. The first class, hybrid assemblers, use a combination of short read data and long read data. For example, Unicycler uses the SPAdes assembler to generate an initial assembly graph from short read data (Bankevich *et al.*, 2012; Wick *et al.*, 2017). Once this step is complete, the assembly graph is refined using both the long and short read data, and finally, the consensus is built (Bankevich *et al.*, 2012; Wick *et al.*, 2017). This approach effectively uses the long reads to scaffold contigs built by the short reads, allowing long reads to inform the genome architecture without contributing erroneous base calls to the consensus. Another hybrid approach works in the opposite fashion; the long-read data is used to build the assembly, and the short reads are then aligned to it and used to correct the errors (Koren *et al.*, 2012). This approach has been used to achieve genome assemblies with greater than 99.9% accuracy, with respect to the bases called (Koren *et al.*, 2012). However, both hybrid assembly approaches require enough starting

DNA for both NGS and long read platforms, and are more expensive than conducting one or the other type of sequencing. The second class of long-read assemblers, the *de novo* assemblers, work exclusively with long reads and have also been used to produce accurate assemblies. One of the best-known assemblers in this class is the Canu assembler, which is based on the old Celera assembler (Koren *et al.*, 2017). Canu applies a hierarchical approach, in which reads are mapped to the longest set of reads, then the assembly is built from these corrected reads, using an OLC assembler (Koren *et al.*, 2017). The pure long-read algorithms work in a similar fashion to hybrid algorithms, but rather than using a secondary set of reads (*i.e.,* short reads), the long reads are used to correct one another. Usually, this means that, within the long-read data, the shorter reads are aligned to the longest reads, which are in turn corrected and used to build the final consensus sequence (Chaisson and Tesler, 2012; Berlin *et al.*, 2015). The pure long read *de novo* approaches rely on having a high enough coverage in order to have sufficient information for correcting the errors (Koren and Phillippy, 2015). Systematic errors may still be challenging to correct, since they are more likely to show up in each read. The issue of systematic errors has less of an impact on PacBio SMRT sequence data, which has a highly random error profile (Ferrarini *et al.*, 2013), but is of concern with nanopore-based sequencing (Krishnakumar *et al.*, 2018).

A focus in recent years has been on improving the efficiency of assembly algorithms, which is especially important if a higher depth of coverage is required to produce accurate assemblies (Koren and Phillippy, 2015). Canu, for example, weights kmers based on their frequency (Koren *et al.*, 2017). This is because kmers are used to identify candidate overlaps, but highly repetitive kmers can vastly increase the number of reads that match one another. By down-weighting these common kmers, more emphasis can be placed on the kmers that are unique to certain parts of the genome (Koren *et al.*, 2017). Another example of an efficiency improvement is trying to decrease the space complexity of various algorithms. For example, Wtdbg2 bins the reads by grouping 256 bp segments. Then they treat a group of k consecutive bins as a sort of kmer, or a k-bin, which greatly reduces the solution space of the problem (Ruan and Li, 2019).

Current long-read assemblers are powerful tools but, while they can produce high quality solutions, they are still computationally intensive and typically rely on specific types or combinations of data. Here, we develop and describe an assembly algorithm designed to assemble reads, specifically targeted at handling high error rates. The algorithm is sequencer agnostic, and can work on subsets of read data in order to make assemblies more scalable. The algorithm works in two stages: it first orders reads and designates approximate overlaps on the basis of a kmer spectrum, and then it generates an alignment and

consensus sequence from these reads. This allows reads to be overlapped, in spite of errors, because it can identify regions with a high number of matching kmers, rather than requiring a perfect match. The second component then uses a similar kmer spectrum to determine how the reads need to be adjusted in order to account for the indels common to long read platforms. Once the alignments are complete, a final consensus sequence is determined, representative of the original, input genome.

2.2 Objective and algorithm description

The objective of this research was to develop an assembly algorithm capable of handling data with high error rates. To achieve this, we have taken an approach in which the distributions of kmers is used to determine the assembly. This project was divided into two parts, which were developed independently by Dr. Brendan McConkey (the first part) and myself (the second part).

The full genome assembler has two core algorithms, which make up the aforementioned parts of the project. The first algorithm involves the ordering and approximate overlapping of reads, hereafter referred to as the ordering algorithm (built by Dr. Brendan McConkey). Once this step has been completed, the reads then need to be aligned over these overlapping regions, in order to generate a final consensus, a step referred to as the alignment algorithm (built by myself). The alignment algorithm works from input provided by the ordering algorithm, which supplies tabulated data about the orderings of reads with the approximate amount of read overlap. The alignment algorithm identifies gaps in reads as well as determines the correct nucleotides at each position of the final consensus. The alignment algorithm is the focus of this thesis chapter, and is what is referred to by "the algorithm" from here onward.

An initial algorithm prototype was built in Python (version 3.5) with the final version being written in C. The C programming language is extremely fast and capable of important "low-level" operations, calculations that can be run on the individual bits of the data. The Python prototype will be touched on in brief here, as it served as a proof of principle, but the focus will be on the overall algorithm design and its implementation in C.

Briefly, the algorithm is built on the idea that reads from the same region of a genome should have the same kmers, short segments of DNA, occurring in the same order. However, as discussed above, long-read sequencers are error prone, and this must be accounted for. Any missing kmers, incorrect kmers, or kmers that are not in the correct position (*i.e.,* due to indels) need to be addressed and resolved. Consider two properly aligned reads; at each position within each of the reads, one would expect to find the same kmers, barring any error. With errors though, then the reads' may become slightly off-set from each other.

If identical kmers are identified at different, but proximal, positions across all the aligned reads, the positions can be averaged using some statistic (*e.g.,* the median position) to determine the best location for the kmer in the final consensus. For example: if the kmer ATGCC occurs at positions 233, 234, 234, 234, and 235 from five different aligned reads, then it is likely that the true position in the genome is position 234. This solves the issue that kmers could potentially be shifted by indels, given enough coverage. This idea can be extended further; if there is a disagreement at some point as to which kmer should be placed at the given position, then the number of reads that contained each kmer can be used to determine which is the best option. For example, if 80% of the reads have the kmer ATGTC at some position, and the other 20% have different kmers, then the ATGTC kmer is most likely to be correct. These ideas underlie how the algorithm is able to use "fuzzy" data to predict a final consensus sequence.

2.2.1 The prototype

The prototype algorithm was developed in Python to serve as a proof of concept. The key question here was whether or not the patterns predicted to be inherent to the processed data (*e.g.,* coverage behaviour around repeat kmers, *etc.*) would be produced by the algorithm in practice. This served to confirm the soundness of the logic behind the algorithm.

This algorithm took a grid as input. Each row of the grid represented a single read of DNA, and each column represented a position within the predicted final consensus. It should be noted that this position means that the rows had been adjusted to approximately align, so the first column was not necessarily the first position of a read, nor did a read necessarily start in the first column. The algorithm then worked by looking down each column, and across each row. Each kmer was added to a kmer spectrum: a series of lists, each list corresponding to a one of the 4^k unique kmers of length k. These lists each contain a list of pairs, consisting of a read number and a position within that read, identifying where the kmers originated from within the set of raw reads. Because these lists were built by looking at all first positions, then all second positions, and so on, they are sorted according to the position in which they occur, not by the reads in which they occur.

The next step involves clustering the kmers. Each kmer list was iterated over and the kmers grouped if they all occurred within some range of one another (this range could vary, values of 3-10 were used here). An average of the position was used in each cluster to determine where the kmer should occur in the final sequence (*e.g.,* the median or mode). In the prototype, both the median and mode were tried, while the final algorithm uses the median. After this step was completed for every cluster, what remains is a series of ordered kmer clusters, each with a number representing the size of the cluster from which they were

identified. By reading the kmers in this series of ordered clusters, a final consensus sequence can be constructed.

A Python script was used to generate near-perfect simulated data by creating a random string of the letters A, T, C, and G. These were then manually mutated at random, as well as artificial repeats introduced. The prototype worked on the simulated data. While the simulated dataset was unrealistic, largely because it was near perfect with respect to error rate, it was still an important milestone. Achieving this milestone meant that the algorithm, at least in principle, could work. From this prototype, the algorithm was refactored to run in C and several changes were made to the workflow, which are described in the following sections.

2.2.2 Data input

In any computer science question, the programmer is faced with the tedious (but most certainly necessary!) problem of having to get data into the program. In bioinformatics programming, the scale of the data compounds this issue, because compression becomes an absolute necessity. Here, we use a 2-bit compression algorithm in order to decrease the size of the data by a factor of approximately four. This is accomplished using a set of extremely fast operators in C, called bitwise operators. To understand how the data compression works, a brief understanding of sequence formats is required.

Sequence data is often stored in either FASTA or FASTQ format. These formats are similar in their required handling practices, and so without loss of generality, FASTA will be discussed here. Each FASTA file is stored as plain text (Figure 2.1). Sequences are identified with a particular symbol, typically '>'. This line of data is called the header. The sequence data begins on a new line of the file, below the header, and can continue, sometimes over multiple lines, until a new line followed by the header symbol is identified. In computers, plain text, which is used by FASTA files, refers to a file where each letter is encoded by a single byte, using the ASCII standard (Information technology -- ISO 7-bit coded character set for information interchange) in most cases. A byte consists of eight 0's or 1's, which are called bits. Each symbol is encoded by some unique byte. For example, the letter 'A' is 01000001. The reason for needing 8 bits to each byte, is that this allows for up to 256 different symbols, meaning that most major symbols, including both the upper- and lower-case Latin alphabets, as well as

```
>sequence_1
ATGCCGCTGTGACTGACTACGTA
GCTGATCGTGCAGTACTGACGGA
CTGATGATGCTATATAAAAGCTA
CGGCTGA
>sequence_2
ATTATCCCGAGCCATTTAGCGCG
CATCGACTGCGATCGACGGTCAG
TCGCAATA
```

Figure 2.1: Sample FASTA format file with two sequences.

numerals, and punctuation, all have a unique encoding. However, DNA has only four possible symbols, being abstractions of the nitrogenous bases. These are ‘A’, ‘T’, ‘C’, and ‘G’. Practically, this means that 252 of the total 256 possible values a byte can hold are not useful when encoding DNA. While this makes FASTA files easily readable by various computer platforms, and readily translatable to a human readable format, it has come at the cost of files being much larger than necessary. Since DNA has an alphabet of only four symbols, a much smaller encoding can be used than the standard byte.

Any alphabet that has only four symbols can be represented by two bits. These are the binary numbers from zero through three (00, 01, 10, 11). Then in the case of DNA, each of the four symbols can be assigned to one of these four binary numbers (Table 2.2). This means that every byte of data, rather than representing a single nitrogenous base in the sequence, can represent four (8 bits per byte, 2 bits per nucleotide). This decrease in size is extremely important when developing fast algorithms that work on large amounts of data, as it means more of the data can be loaded into the limited fast memory (RAM) of a computer at any given time. For example, if a program has 8 GB of free space available to it, then only 8 GB of sequence data can be loaded using a standard FASTA file. However, with the compression outlined above, this becomes 32 GB of sequence data – approximately 128 billion base pairs (Gbp) of data.

Table 2.2: Encoding of the four nucleotides used for data compression during the data input stage of the algorithm. Notice that complementary nucleotides have inverted bits from one another, which allows sequences to be readily reverse complemented programmatically.

Nucleotide	Plain text symbol	Encoding
Adenine	A	00
Cytosine	C	01
Guanine	G	10
Thymine	T	11

To achieve this compression, the algorithm uses a library called kseq (Klib — a generic library in C). This is a publicly available library of C code for rapidly reading sequences from a FASTA or FASTQ file. The kseq interface opens the FASTA file for reading, then loads only a single sequence into RAM at a time, along with a number representing its length. This is important, since it means only a small amount of memory is required at any given time. Once a sequence is loaded, a buffer in memory (a contiguous block of memory, often called an array) that is one quarter of the size, rounded up to the nearest byte, of the sequence is allocated (Figure 2.2). The sequence is then read one base at a time. Each base is converted to

a 2-bit number using a lookup table. Since all data on computers are stored as 8-bit bytes, these 2-bit numbers will have leading zeroes. This means that the 2-bits actually encoding the nucleotide need to be moved to the correct position, then combined with the other nucleotides in sets of four (Figure 2.2). This is accomplished by pulling the correct byte from the buffer, shifting the new encoding to the correct position within that byte, then combining the two. The resulting combined byte is then placed back into the buffer (Figure 2.2). Once a byte has been filled up, the next byte in the buffer is used. Bytes must be used here, since they are the smallest units in computer memory that have an address, which is required to access data.

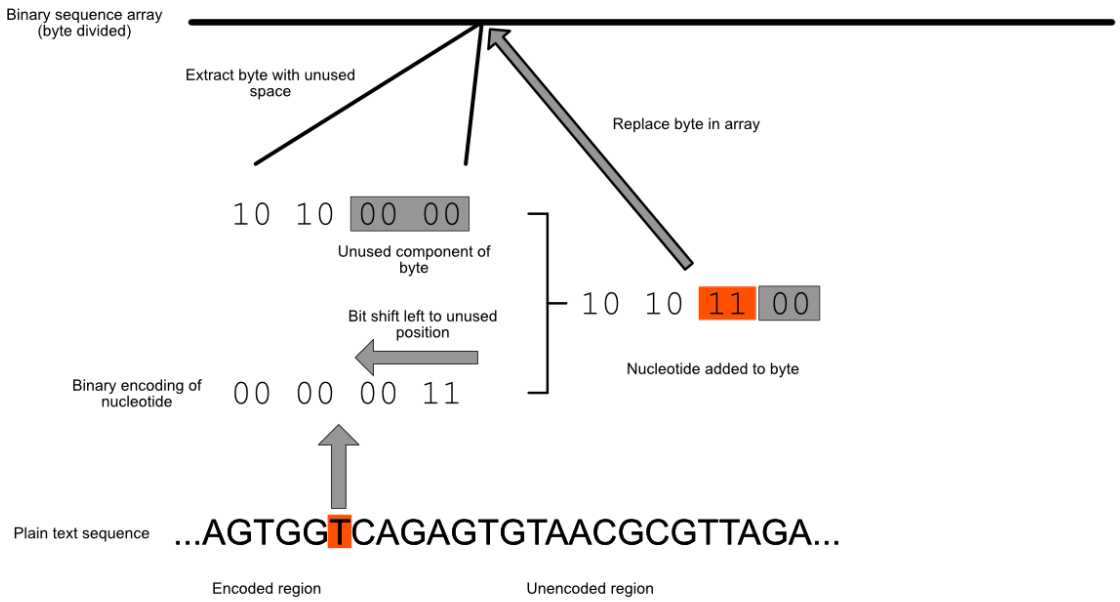

Figure 2.2: Illustration of the process for encoding a plain text DNA sequence to a binary array.

The interface for storing the compressed sequence files involves an array, or table, where the n^{th} item in the table matches the n^{th} sequence in the FASTA file. Because sequences are stored and accessed by their numerical index (*e.g.,* first sequence, second sequence, etc.), there is no need to keep the headers that are usually used to identify them. These are discarded to further save space (*N.B.* – the header can still be retrieved by finding the n^{th} header in the original FASTA file). The other advantage of using an index to store the sequences is that any sequence can be immediately accessed, without the need to look for it. This is referred to as *O(1)* lookup time (see Box 1 for a brief description of big "O" notation in computer

science; Bachmann, 1894; Landau, 1909). In a plain text file, unless some indexing is performed, this requires linear time, or *O(n)*, which would make the program significantly slower.

The implementation of this compression involves the use of two wrapper structures. The first wrapper is for individual sequences and stores the actual sequence as a byte array, as well as some metadata pertaining to the sequence. Specifically, this metadata contains the length, which is required to determine how much of the last byte should be used. The second wrapper stores the entire sequence set. This is an array of pointers, each pointing to one of the aforementioned sequence structures. It also contains some metadata, such as the number of sequences stored in the structure. The main function that interfaces with the sequence storage module of code is for extracting kmers. The sequence number and the position are passed to the kmer extraction function, which then returns a kmer of set length (*k=5* was used here) from the given read at the given position.

Box 1: Big "O" notation is a way of analyzing the running time required by a program. The idea is that it provides an approximate mathematical function that can describe the running time relative to the size of the input. If the size of the input is *n*, then this is described as a function of *n*. For example, if a program requires one calculation for each part of *n*, then this would be called linear and denoted *O(n)*. If the program always takes the same amount of time regardless of *n*, then this would be called constant and denoted *O(1)*. If the program has some step that takes *O(n)*, and this must be performed for every part of the input, then this would be *O(n*n)*, or $O(n^2)$, and called quadratic. Smaller orders of functions are desirable, since they do not grow as much with the size of the input. If the above represent three algorithms for the same task, *O(1)* would be the fastest, then *O(n)*, and finally, $O(n^2)$ would be the slowest.

The second component of data input involves the set of read overlaps generated by the ordering algorithm developed by Dr. McConkey. The input file passed to the alignment algorithm has a series of records that each describe how two different reads overlap. Each row of this "record" file has six fields. The first three correspond to what is called the query read, and the second three correspond to what is called the match read. The query read can be thought of as the read that occurs first in the ordering, whereas the match is the read that occurs second. The three fields describing each read are the read number (used to index the sequence set), the orientation (which describes whether to use the forward "+" or reverse complement "-" strand of the read), and the position within the read that matches to the other (Figure 2.3).

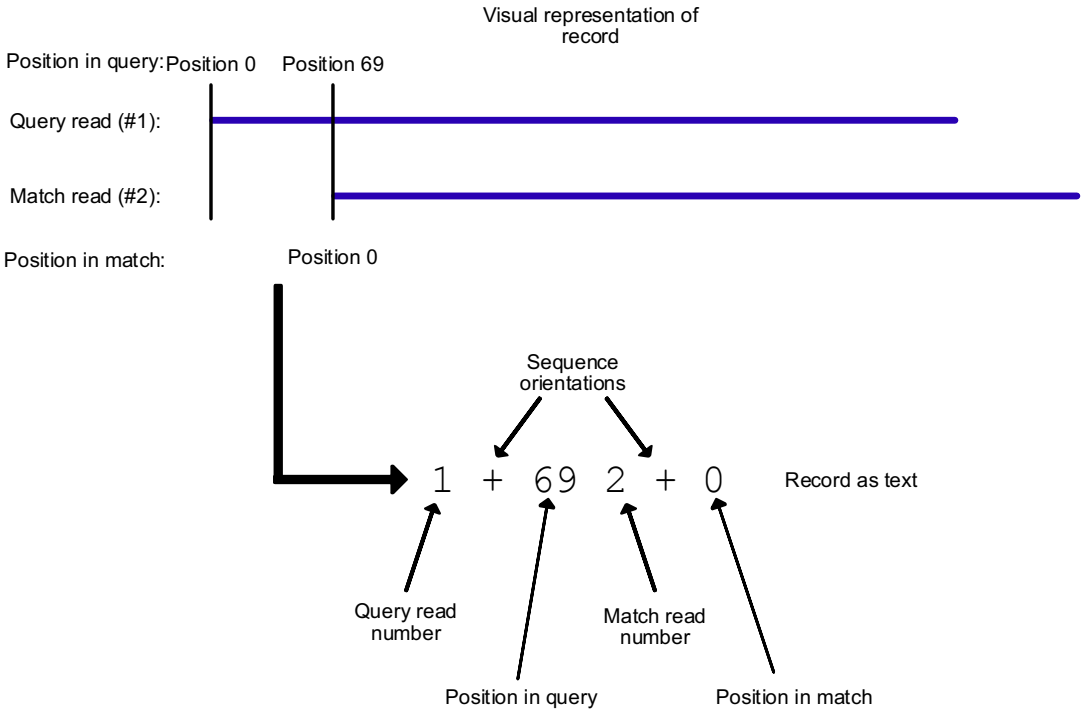

Figure 2.3: Visualization of an ordering algorithm-generated overlap record and its corresponding text format.

The record files are read one line at a time, and each record is stored as part of a linked list, representing the order that records should be used in, matching the order that the records occur in the file. The records must then be converted to an alignment window, which is conceptually the same as the grid described for the prototype.

2.2.3 Building the alignment window

The alignment window represents a grid of nucleotides. Each row represents a read, and each column represents a position within the final consensus, with the reads positioned to match this. Depending on how reads overlap, some number of cells at the start or end of a row may be empty (Figure 2.4).

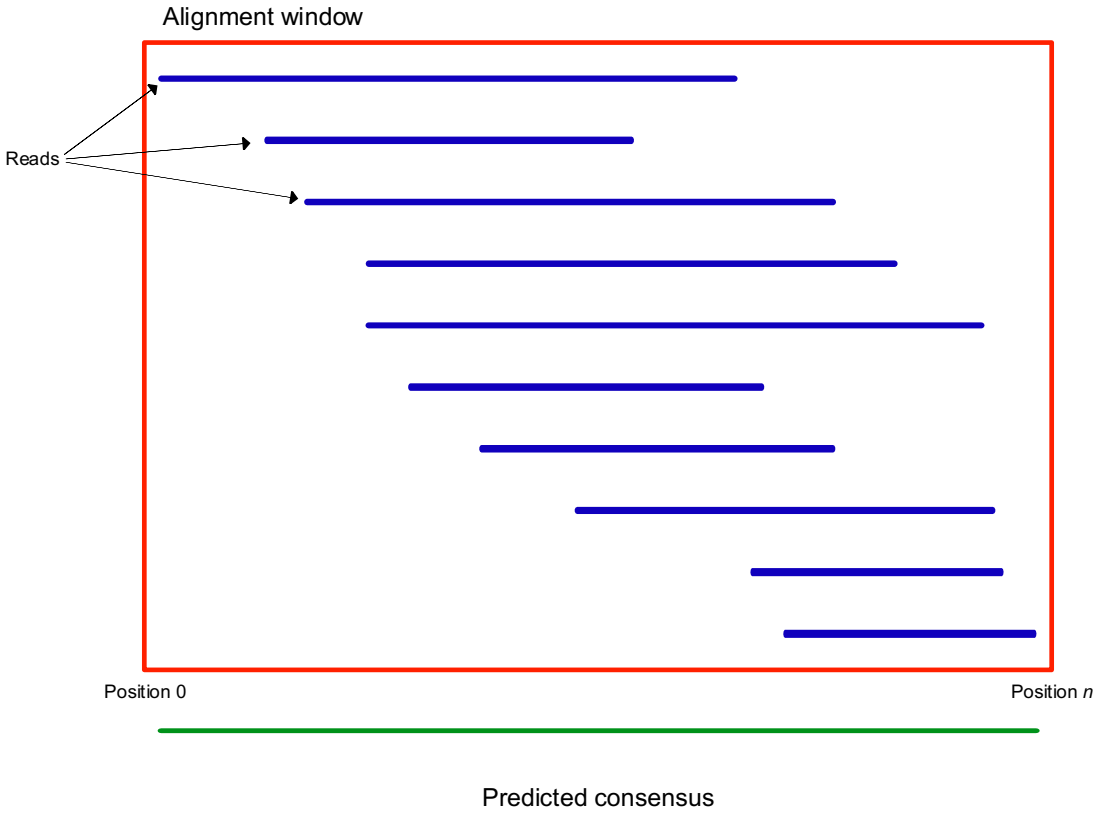

Figure 2.4: Graphical representation of an alignment window. Each read is aligned to its approximate global position within the window.

The window is built from two structures. Each row of the window is represented by a structure called an entry (*i.e.,* an entry in the table). Each entry contains a read number, a direction (representing the strand to be used), and an absolute position within the final consensus. This number is calculated from overlap records.

The second structure is the actual window. This contains an array of the entries described above, as well as metadata. These metadata are a pointer to a binary sequence object (called the FASTA handler here), a pointer to a record list structure (described above), and the number of entries in the array, which is used to prevent access beyond the last entry in the array (an out-of-bounds access causes the program to crash). The interface to this structure is a single function which takes the filename of a record file and the filename of a FASTA file as input. When this function is called, it attempts to open these input files, and

it then builds an alignment window from their information, using the input interfaces described in section 2.2.2.

Building the window requires some arithmetic in order to convert the data in the records into absolute positions for every read. To do this, the first query read in the records file is assigned an absolute position of 0. This is stored as the first entry in the window array. The match read must then be adjusted to an absolute position, rather than a position relative to the previous entry (*i.e.,* the query). This is calculated with the following formula:

Equation 2.1

$$m\_absolute = q\_absolute + q\_position - m\_position$$

Where *m_absolute* is the predicted absolute position of the match read in the final array, *q_absolute* is the absolute position of the previous read in the window (*i.e.,* the query), and *q_position* and *m_position* are the query and match positions in the record, respectively. Once this has been calculated, the entry is added to the array, and the correct orientation noted. This process is repeated for all records. When all records have been processed, the alignment window is complete. The window can then be used to build the kmer spectrum, which represents all kmers across all reads within the window.

2.2.4 Generating the kmer spectrum and clustering

The heart of the alignment algorithm is the kmer spectrum for a given set of ordered reads. The kmer spectrum is an array with each row representing a unique kmer. These rows hold a linked list of ordered read-position pairs describing where that kmer occurred across the window. These can then be used to estimate the best position, within the final consensus, at which to place a kmer, by clustering groups of kmers that occurred at similar absolute positions (according to Equation 2.1).

The main structure in the kmer spectrum is an array, where the index of each element corresponds to a unique kmer and each element is a linked list of read and position pairs. To index each row, a kmer is converted to its binary representation, and this number is used as the index within the kmer spectrum array. For example, using the encoding described in section 2.2.2, ATTGC can be represented as an integer (integers are typically 32 bits internally, allowing up to *k=16*) with the binary value of 00 11 11 10 01, or 249 in decimal. This means row 249 (which is the 250th row, including the 0th row) describes all positions in all reads where the kmer ATTGC occurred. Note the use of *k=5*, which was used as a default value, giving $4^5 = 1{,}024$ rows in the kmer spectrum. If we assume an error rate of ~10%, then we would

expect an error at approximately 1 in every 10 positions, so choosing *k=5* means that the majority of kmers (~59%) will not span errors.

To generate the kmer spectrum, each position of the window is iterated, from top to bottom, left to right. This order of scanning the kmers is important, since it means that the entries into the kmer spectrum will occur in order with respect to their position in the final consensus. This is accomplished by storing a number representing the column to look at in the window (*i.e.,* the absolute position), which starts with a value of zero, referred to here as the position. Each read is taken in turn and, if the cell is not empty, the kmer is extracted from this position. Once this has been done for all reads, the position is incremented to one, and the process is repeated. The position is then incremented to two and repeated for all reads again. This continues until the position has exceeded the ends of all reads. The algorithm is described in pseudocode below.

```
01    for position=0..MAX_POSITION; do
02          for read in window; do
03                kmer = extract_kmer(read, position)
04                kmer_spectrum[kmer].append(read, position)
```

Listing 2.1: Pseudocode describing the procedure to build the kmer spectrum for a set of reads and overlap records.

Here, the variable *position* represents the position within the overall consensus, *read* represents the read to extract the kmer from, *kmer* is the binary representation of the kmer extracted from the read, and *kmer_spectrum[]* represents the array for the kmer spectrum (Figure 2.5). Note that this algorithm is linear (*i.e., O(n)*) with respect to the total length of all reads, since each position in each read is examined exactly once. By building the array in this manner, each read and position pair will occur in order with respect to the absolute position. That is, within a given kmer list, a higher position will never occur before a lower one. This sorting allows inferences to be made about where the kmers ought to be placed in the final consensus sequence, without the need for any sorting. As will be seen, this enables the algorithm to perform in linear time. Consider the case of perfect data (*e.g.,* there are no errors in the reads and all overlap records are correct). If the first read has the kmer ATGCC at the absolute position 346, then all other reads will have the same kmer at the same position. The kmer list corresponding to ATGCC will then have a cluster with all read numbers, each having position 346, somewhere within the ATGCC list. This, then, will be taken as the correct kmer for position 346 in the final consensus. It would also stand to reason that somewhere else in the kmer spectrum, there will be a cluster that begins with TGCC with the

same number of read-position pairs, all at position 347. By identifying all of these clusters, in which a group of the same kmer occurs in close proximity, positions can be determined to build an ordering of kmers. These groups of proximal kmers will henceforth be referred to as clusters.

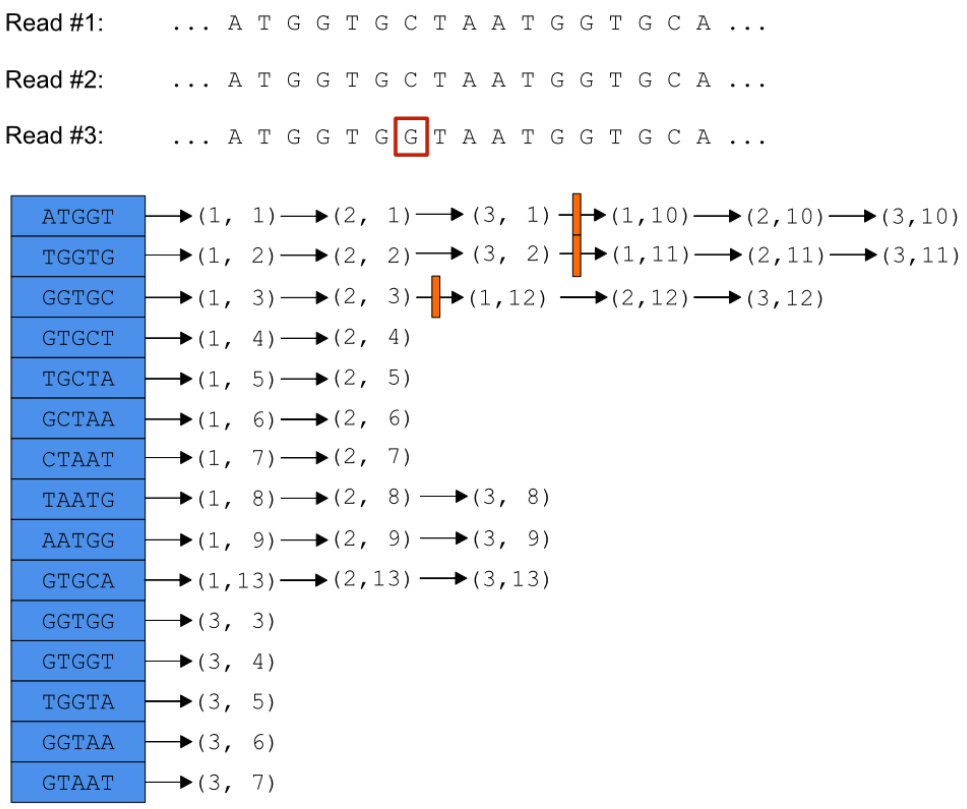

Figure 2.5: A sample kmer spectrum for three aligned reads. Orange bars demarcate clusters and each bracketed number pair is in the format (read, position). Notice that read 3 has a point mutation (red box).

In practice, data is not perfect. Because of errors, a cluster may be missing a read or, in the case of the indel mutations common with long read sequencers, the position of the kmer may be shifted. In these cases, a criterion is required to determine which kmers in the array belong to the same cluster. To solve this problem in the alignment algorithm, lookahead was used. A pointer iterates over the kmer lists, one list at a time. The first kmer is selected, then all subsequent kmers that occur within some number of positions are clustered to this first kmer. Once the last kmer within the lookahead has been identified, the iterator moves past and repeats the process to cluster the rest of the kmers in the list. To decide which

position within the final consensus sequence this cluster represents, the median position of the cluster is chosen. For example, if the kmer ATATA has a cluster containing positions 345, 345, 346, 346, 347, then the cluster would be assigned an absolute position of 346. This would mean the algorithm's prediction for position 346 is that it contains the kmer ATATA.

The consensus structure used to store these clustered kmers consists of an array the length of the predicted consensus sequence (or longer). Each position within the array contains a pointer to a linked list of entries. With perfect data, each list should have only a single entry. In practice, multiple clusters may be erroneously assigned to the same position, and so a linked list is used to prevent clusters from being ignored. Each cluster is stored in a structure called a contig entry. This consists of a count, representing the number of kmers that were clustered to that position, and an integer representing the actual kmer.

```
01    for kmer=0..4^k; do
02          cluster_start = kmer_spectrum[kmer].next.position
03          cluster_size = 1
04          while ( current_kmer = kmer_spectrum[kmer].next ); do
05               if ( in_range(current_kmer, cluster_start) ); do
06                    cluster_size = cluster_size + 1
07               else; do
08                    consensus[cluster_median] =
                           (kmer, cluster_size)
09                    cluster_start = current_kmer
10                    cluster_size = 1
```

Listing 2.2: Pseudocode describing the procedure to populate the array of kmer clusters.

Described in plain English, this algorithm (Listing 2.2) iterates over all kmer lists in the kmer spectrum. For each list, it gets the first kmer, then iterates down the list until it finds a kmer that is out of range (*i.e.,* more than some number of positions downstream from the starting kmer. A range of values, of 3-10 were tested for this work). Once an out-of-range kmer is identified, the median position of the cluster is used to index into the consensus sequence structure's array. The number of kmers in the cluster, as well as the numerical representation (*i.e.,* its binary encoding, stored as an integer [16 or 32 bits]) of the kmer sequence itself is stored at this spot in the array. If the current kmer did not exceed the range, then the size of the cluster is simply incremented. Note that the median is calculated as the cluster is gathered by moving the pointer to the current median kmer forward by one for every two kmers counted. This means that determining the median only adds a negligible amount of time to running the program.

The other thing to note with this part of the program, is that it scales linearly with respect to the total size of the reads. To prove this, consider that the inner loop (Listing 2.2, line 4 *ff.*) looks at every kmer in the particular kmer list exactly once. The outer loop (Listing 2.2, line 1 *ff.*) looks at each kmer list exactly once. This means that every cluster in the entire array is considered once, and as demonstrated above (see section 2.2.3), there will be exactly one kmer for every position of every read, up to the $(k\text{-}1)^{st}$ position, which is approximately equal to the sum total of all read lengths. From this we see that the algorithm described above is linear with respect to the total size of the input reads (*i.e., O(n)*). Thus, the algorithm up to this point is still a linear time algorithm (*i.e.,* the overall run time is *O(n)*).

With the kmer spectrum built and clustered into the consensus array, the final consensus can be built. This is done by converting the data to a graph, a series of vertices connected by edges.

2.2.5 Alignment graph construction and important properties

With the kmer spectrum built and clustered into the consensus array, the final consensus can be built. This is done by converting the data to a graph, a series of vertices connected by edges. Two issues arise that need to be addressed. The first issue is that ambiguous cases need a decision made. The second issue is that gaps in the array need to be filled. This becomes a natural problem for graph theory, where points of data are connected and paths over these connections are identified. A graph can be used to identify optimal choices where ambiguity exists using different weight metrics. Furthermore, breaks can still be connected with edges in a graph so that gaps in the original array do not interfere with the assembly, by joining vertices that are separated by some distance. In the case of this algorithm, the array already resembles a directed acyclic graph. These ideas will be discussed in further detail below.

There are two major questions that first need to be addressed when deciding to use a graph to solve a computational problem or not: is graph theory appropriate for the question being posed? If graph theory is appropriate, what should the vertices and edges be defined as? In the case of genome assembly, graph theory has been shown time and again to be an excellent tool (*e.g.,* de Bruijn graphs and overlap graphs; Compeau *et al.*, 2011; Medvedev, 2017), and our case is no exception. Graphs lend naturally to determining the best paths through a series of connected objects. Sequenced reads have the property that the underlying data is a series of connected sequences. By using graph theory, a path through a network of connected sequences can be used to determine a final consensus sequence. The second question is how to define the vertices and the edges. The consensus graph for the alignment algorithm has a natural definition for a vertex: each cluster in the graph, which contains coverage information as well as defines a kmer for a position, can be a vertex. If this is the case, then edges can be defined as overlaps between the

kmers corresponding to each cluster. Furthermore, these edges can be directed, since the order of the overlap is known. This was the basis for defining edges in the alignment algorithm as follows: two vertices are joined by a directed edge from the vertex occurring earlier in the consensus array to the vertex occurring later in the consensus array, if and only if the kmers overlap by *k-1* or *k-2* positions, and the second occurs within some predefined range of the first. This decision process was so defined because a value of *k=5* was used, meaning that *k-1* and *k-2* consider overlaps of three or four nucleotides. This minimizes potential kmer overlap errors. Overlaps of only one or two shared nucleotides between the kmers would create too many random edges, leading to a higher probability that the final consensus would be incorrect. By allowing some flexibility in the overlaps between kmers, edges can still be identified if clusters were placed at the wrong position. The reason this is important is that if a cluster is shifted earlier, then a blank space may follow in the consensus array. The space after the blank, however, is likely to have the correct kmer, but it may only overlap by *k-2* positions. In order to build edges, the consensus is iterated over from the first position to the last. At each position, all clusters are iterated over. For each of these clusters, all clusters that occur downstream within a predefined range are considered. If the associated kmers overlap, as described above, a directed edge is added to the starting cluster, going toward the overlapping cluster. The directionality is important here.

The time complexity of building the graph is somewhat challenging to analyze, but an attempt will be made here, in order to demonstrate that in the average case, this should be approximately linear. If we let *n* be the number of vertices, or kmer clusters, then the maximum number of edges the first vertex can have will be *n-1*. This is because this vertex cannot have an edge going to itself. Since these vertices are directed downstream, the next vertex can only have *n-2* possible edges, since it cannot point back to the vertex before it, nor to itself. This means that there are two less than the total number of vertices that it could point to. By similar reasoning, the next can have *n-3*, and so on until the last node, which will have zero. Then the highest possible number of edges will be the sum from zero to *n-1*. This sum is given by the formula below.

Equation 2.2

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

This equation for the sum of numbers from 0 to n-1 is well established in mathematics and was demonstrated by Gauss (1777-1855). In terms of running time, the number of edges that need to be

considered would represent a polynomial algorithm: it is no longer linear, but quadratic (note that *n* is multiplied by itself in the equation, meaning the runtime would be $O(n^2)$). However, the implemented lookahead puts an upper bound on this number. If we let *c* represent the lookahead that is used, then the worst-case runtime complexity is substantially simplified. The worst-case runtime would become the following:

Equation 2.3

$$\sum_{i=n-c}^{n-1} i = \frac{[(n-1)-(n-c)+1][(n-c)+(n-1)\,]}{2}$$

Equation 2.4

$$\sum_{i=n-c}^{n-1} i = \frac{c(2n-c-1)}{2}$$

Equation 2.5

$$\sum_{i=n-c}^{n-1} i = \frac{2nc-c^2-c}{2}$$

Note that *n* will be much larger than *c*. The current version of the code uses $c=8$, whereas *n* is on the order of thousands or more. In addition, *c* is a defined constant, whereas *n* varies with the amount of data. This means that the second and third term can be considered constant terms. We can collectively refer to these as k_2 and we can refer to the constant coefficient $2c$ as k_1. This yields a running time of $O(k_1n + k_2)$. This equation is linear with respect to the size of the input, *n*, and so the final running time is $O(n)$, and the algorithm remains linear.

To further prove that the algorithm is linear, we can consider a more realistic case, where the clusters are equally spread out across the entire array. The expected number of clusters will be approximately the length of the predicted consensus. If they are equally spread out then, on average, the worst-case number of outgoing edges for each vertex will be bounded by *c*. Over all ~*n* positions in the consensus array, this gives $O(nc)$ edges to be considered. Again, since *c* is constant, this gives an overall running time of $O(n)$.

Two different approaches to estimating the running time have shown that it is linear. Taken even further, if we assume uniform coverage of the consensus sequence, then clustering will reduce the amount of data to consider, since kmers at the same position will be condensed into a single cluster. This means

that the size of *n* for this part of the program will not be equal to the total size of the input reads, but will instead be closer to the total size of the input reads divided by the average coverage of the consensus.

```
01    for i=0..MAX_POSITION; do
02          for v in consensus[i]; do
03                for u in.range(v); do
04                      if overlap(v.kmer, u.kmer)
05                            add_edge(v, v->u)
```

Listing 2.3: Code for building the assembly graph from the consensus array.

The construction of this graph in order across the consensus array gives it several important properties. First of all, it is guaranteed to never have a cycle. That means, when a path is made by following edges, the path will always tend toward the end of the consensus array and never loop back to an earlier position in the graph. This means the graph is acyclic. The other useful property is that the graph is topologically sorted. A topological sort has to do with how a graph is stored as a data structure. One way to think about it is like a list of tasks that must be done, where one task depends on some number of previous tasks having been completed. If these tasks are topologically sorted, then their order will ensure that for any task that depends on another, that other task will already have been completed. More precisely, this can be described as follows: for any two vertices *v* and *u*, if there is a path from *v* to *u*, then *v* is guaranteed to occur at an earlier position in the consensus array (Figure 2.6). Taken together, these two properties mean that the assembly graph is a topologically sorted directed acyclic graph.

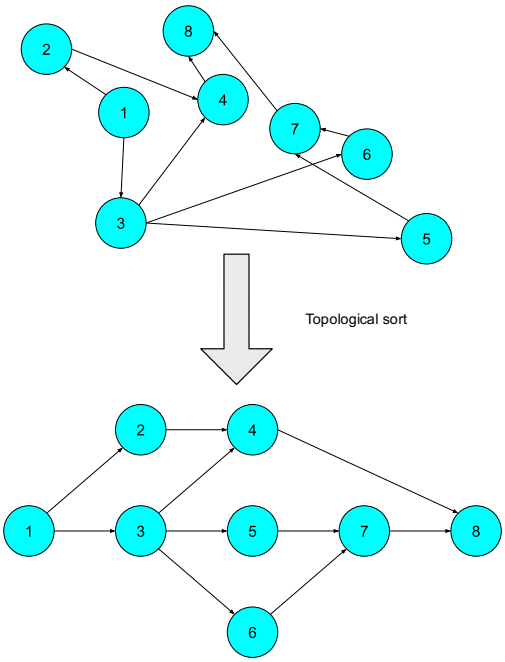

Figure 2.6: Example of a graph that is unsorted (top) and topologically sorted, from left to right (bottom). Note that in the sorted graph, an edge never points backward.

2.2.6 The longest path problem and generation of a consensus sequence

In computer science, the longest path problem has received some attention and is closely related to what is often called the Critical Path Method. The Critical Path Method and longest path algorithm for a directed acyclic graph has a special case for which a linear time algorithm exists (Ammeraal, 1996). This is not true of general graphs, for which the longest path problem is NP-hard, which motivates the importance of the properties described in section 2.2.5.

The linear longest path algorithm involves first defining the distance to all vertices from the desired start point as negative infinity (or some representation thereof). The desired starting point is then assigned a distance of zero. Each of the neighbours are then checked to see if the distance to them from the current vertex plus the distance between them is larger than the best identified candidate so far. This is given in pseudocode below. Note that we will assume that all vertices have already been assigned a distance of negative infinity, and that the vertex at the start of the consensus array has been initialized to zero, since this is where the best contig should start.

```
01    for v in consensus_array; do
02          for u in v.neighbours; do
03                if u.distance < v.distance + distance(v->u)
04                      u.distance = v.distance + distance(v->u)
05                      u.source = v
```

Listing 2.4: Pseudocode describing the procedure to build the longest path through a directed acyclic graph.

This part of the algorithm works by starting at the first entry in the consensus array, which is assigned a distance of 0. It then looks at the "distance" from the first vertex to all of its neighbours. If this distance is better than the neighbour's longest distance calculated so far, then the neighbour is updated. Once all neighbours are updated, the algorithm goes to the next vertex in the topological sort and repeats the process. This continues until all vertices have been visited and had their best distance calculated. This is the reason for starting with negative infinity, any distance will always be better (*i.e.,* longer) than that, so long as a path to the vertex exists.

Note that the algorithm has two loops. The outer loop looks at every vertex (which is proportional to the total sequence input size), and the inner loop looks at every edge of each vertex (which, as shown above, is also proportional). This can be denoted as $O(|V| + |E|)$, where V and E are the sets that have all vertices and edges, respectively, and $|V|$ and $|E|$ denote the number of items in each set. Since both of these are proportional to the total input size n (described above), then the total running time is approximated by $O(2n)$, which again, is a linear time algorithm and can be reduced to $O(n)$, confirming that the algorithm continues to be linear with respect to running time.

The longest path problem works well here since it will naturally approximate the size of the predicted contig when the weights (*i.e.,* distances) associated with edges are all assigned a value of one. The reason for this is that, in most cases, each edge will represent a single step forward in the array, which corresponds to moving forward by one position. In some cases, this may represent several steps, but this will only be the case when some number of clusters were mis-assigned. These cases will not have any practical impact on the final result, since the blank spaces do not represent anything in the final sequence, so can be effectively dropped. Furthermore, the number of edges passed over will also approximate the size of the predicted contig, regardless of the weight function used, since the edges always point forward. The other reason this algorithm works well is that it will attempt to incorporate as many useful vertices as it can, while skipping over dead ends. These dead ends occur when kmers do not properly match due to

an error. These features of the design ultimately mean that the algorithm will tend toward better assemblies, without the need for any extra correction steps.

As with any algorithm, this one is not without its flaws. First, the case where one read has a large insertion that re-converges to the main path may be preferred. The correct path could potentially skip over this region to the point of re-convergence, which would only increase the distance by one step. However, if the read with the insert has some numbers of kmers occurring between these two points, then this could artificially inflate the distance from the point where the divergence started to the point where the convergence occurred (Figure 2.7).

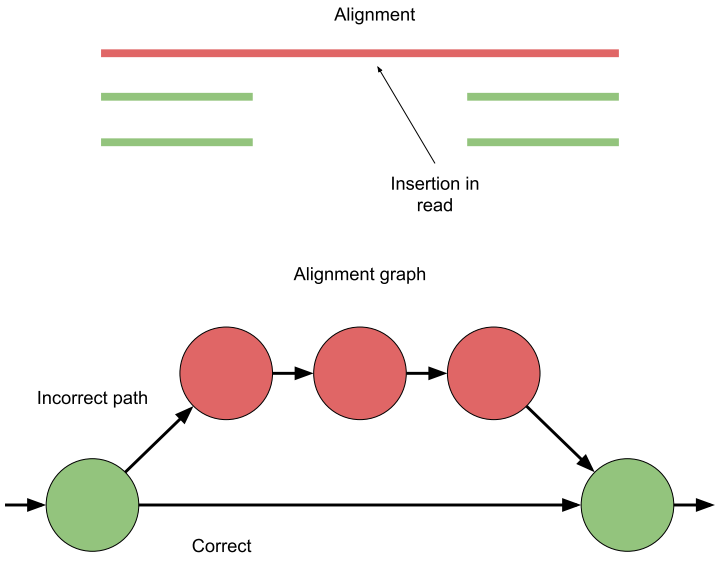

Figure 2.7: Illustration of an alignment where one read has an insertion, giving rise to the divergent path (red), away from the correct path (green, sequence encoded by the other reads).

Several small improvements to the algorithm were made to generate a better assembly. The main refinement was to fix cases where two options were equally good choices. These cases typically represent a substitution, which will force the algorithm to choose between two different nucleotides at one position, which presents as two different paths through the graph with the same distances. To make this choice, ties in distance were resolved by picking the kmer cluster that has the most support (*i.e.,* was the largest cluster, which implied the majority of reads supported that particular nucleotide). On a test dataset of 10

reads, the largest of which was 29,371 bp, this drastically increased the average predicted support of the longest contig (without the refinement the contig length was 10,930 bp with an average support of 1.68 clusters, while the refined version produced a contig of 10,510 bp with an average support of 2.27 clusters). This result is important for two reasons: first, it shows the algorithm works and can produce contigs, and secondly, small refinements have the potential to improve support from the information inherently generated by the alignment graph. Further refinements are discussed in sections 2.2.7 and 2.2.8.

2.2.7 Comparison to other assembly methods and algorithm refinement

To assess the algorithm, we compared an assembly generated by our algorithm to one generated by another established assembler, Unicycler. This assembly was for the genome of an unnamed strain of *Bacillus thuringiensis*. The reference assembly was made with Unicycler (v0.4.7; Wick *et al.*, 2017), and provided by the McConkey Lab. This was a hybrid assembly using Illumina HiSeq reads, which were corrected with Pollux (Marinier *et al.*, 2015) at a coverage of ~400x, and MinION long reads which were base called with Oxford Nanopore Technology's software, Guppy (v2.3.5) at a coverage of ~100x. This assembly was used as a reference. The assembly we generated used 10 of the long reads from this data set, which represented a predicted 29,906 bp region of the original genome. The 10 reads selected ranged in size from 5,014 bp to 29,371 bp. The performance accuracy, and strength of our assembler were assessed using a suite of statistical measures. Our algorithm was applied to the 10 selected reads, representing a low coverage assembly, and the generated consensus sequence was aligned to the Unicycler reference assembly. The main statistics used to assess the quality were the average cluster size, which approximates the average coverage; length of the best contig; number of edges in the assembly graph; the number of clusters used to build the assembly; and percent identity over the length of the sequence compared to the Unicycler reference assembly (Table 2.3). When considering these statistics, some caution is required. While they do provide useful insights, the possibility for contradictory conclusions may arise. For this reason, an attempt will be made to consider each statistic from the angle of improving assembly, as well as how they may actually mean a decrease in the quality of the assembly. In all cases, the Unicycler assembly (Figure 2.8 and Figure 2.9) was used as a standard to benchmark our algorithm's performance. Note that the test dataset emulates a low coverage genome sequence with nanopore reads, which is a case where assembly algorithms typically exhibit below-average performance.

The assembly quality was assessed while varying three key parameters: cluster range, lookahead range, and weight function. The cluster range defined the maximum distance that could occur between two

kmers in the kmer spectrum in order for them to be considered as belonging to the same cluster. We varied this parameter from 3 to 10. The lookahead value was used when building the alignment graph. This would define how far ahead the algorithm would look, in terms of position, when identifying edges (*e.g.,* if a cluster for ATGTT was at position 6 and cluster for TGTTC was at position 9, should they be joined by an edge?) Here, we tested values from 3 to 10, as well. Finally, the weight function defined the "distance" that each edge represented. For varying the weight function, three functions were used: assigning a value of one to all edges, assigning the coverage of the first kmer (called back-weighting), and assigning the coverage of the second kmer (called front-weighting; Table 2.3).

Table 2.3: A summary of key assembly statistics under different parameters for computing the best contig from the test data set. Parameters varied were cluster range, which was the distance used to determine if kmers belonged to the same cluster; lookahead, which determined how far ahead an edge could point; and weight function, which was one of three functions for determining the "distance" an edge represented. Contig length represents the longest contig from a given assembly. Average cluster size is the average size of cluster incorporated into the assembly across its length, which approximates mean read depth of the longest contig. The % identity (id) is calculated over the length of the longest contig for that assembly compared to the homologous stretch in the Unicycler reference assembly.

Cluster range	Lookahead	Weight function	Number of edges	Contig length	Average cluster size	% id to reference
5	3	1	148,158	8,622	1.73	65.0
5	5	1	167,681	10,930	1.68	68.8
5	8	1	198,858	13,264	1.66	70.2
5	10	1	229,412	13,264	1.66	70.2
5	3	Back-weighting	148,158	8,377	2.27	79.9
5	5	Back-weighting	167,681	10,510	2.27	82.4
5	8	Back-weighting	198,851	12,721	2.21	83.6
5	10	Back-weighting	229,412	12,721	2.21	83.6
5	3	Front-weighting	148,158	8,377	2.27	79.9
5	5	Front-weighting	167,681	10,510	2.27	82.4
5	8	Front-weighting	198,858	12,721	2.21	83.6
5	10	Front-weighting	229,412	12,721	2.21	83.6
3	5	Front-weighting	193,545	12,936	1.92	82.8
8	5	Front-weighting	141,808	7,450	2.47	76.8
10	5	Front-weighting	130,781	6,337	2.65	73.4
6	8	Front-weighting	182,979	12,585	2.30	83.3

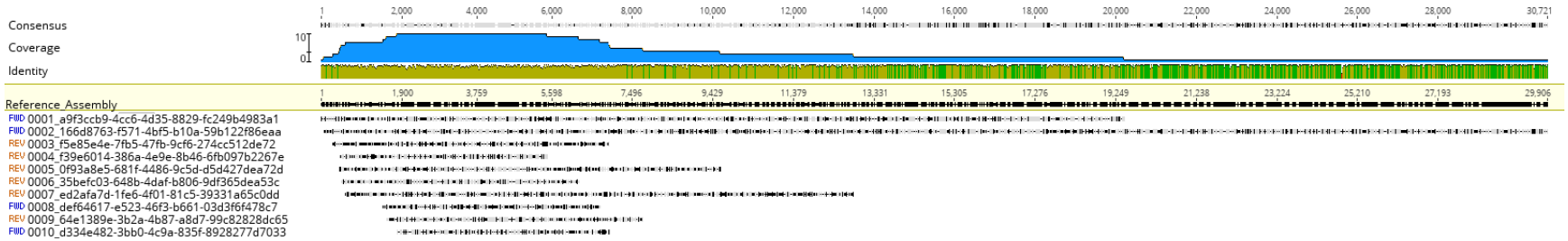

Figure 2.8: Test dataset of 10 reads aligned to the Unicycler assembly. Coverage (top plot, blue) and identity (middle plot, green indicates 100% identity, height of yellow bars indicates identities <100%) of reads (listed on the left and depicted below the identity plot as an alignment) against the Unicycler reference assembly (top sequence in the alignment, highlighted in yellow and numbered by position).

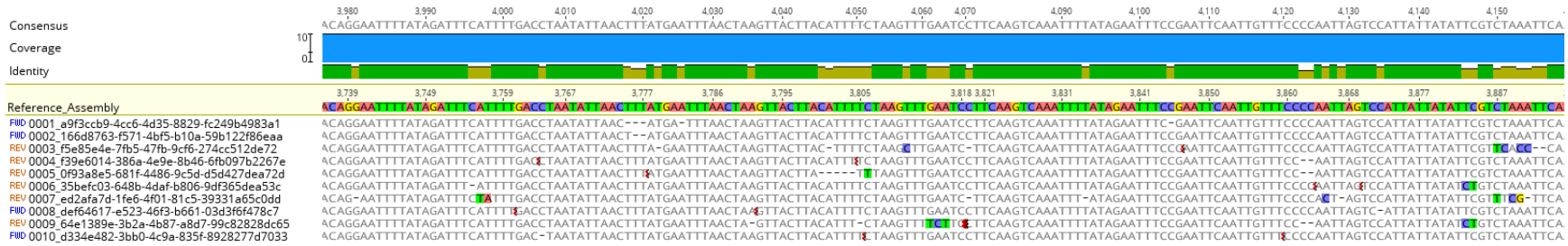

Figure 2.9: Zoomed-in regions of the 10 reads mapped to the assembly produced by Unicycler. The top plot (blue) shows the coverage, the identity plot shows column-wise identity, with green bars representing 100% identity and the height of yellow bars indicating identities <100%. The reference assembly is highlighted in yellow at the top. Nucleotide mismatches are highlighted in colour within the alignment and red squiggles indicate gaps.

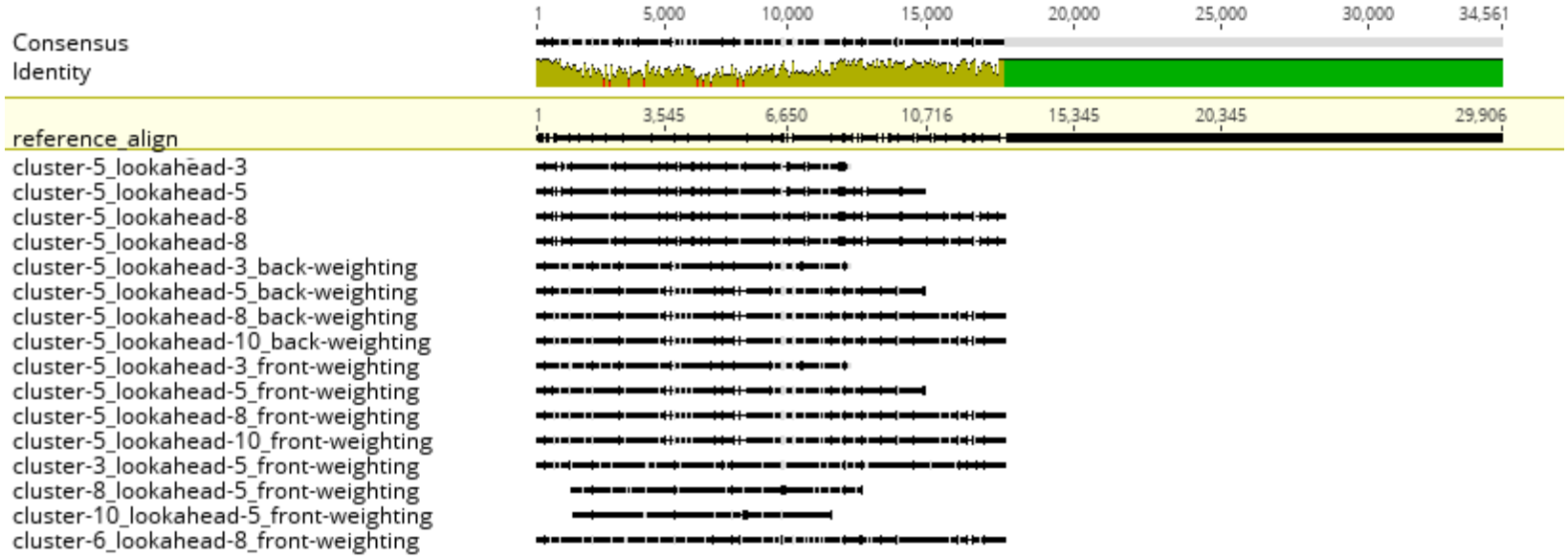

Figure 2.10: Assembly consensus sequences generated by the alignment algorithm under different parameters aligned to the Unicycler reference assembly (highlighted in yellow). Green bars represent 100% identity, yellow represents identity proportional to the height of the bar <100%. The assemblies are given as an alignment below the reference assembly.

2.2.7.1 General algorithm performance

The algorithm generally performed well, although there are several key aspects that should be addressed in future versions to improve the assembly. In general, a coverage of 3x and higher gave enough resolving power to create a relatively long contig, that had long, error free stretches. However, the algorithm performed poorly in regions where coverage was lower. With the sample dataset discussed above, the ~19 Kbp region that had low coverage (position ~12,000 onward in Figure 2.8) produced approximately 200 contigs. This was largely caused by errors in the reads, causing kmer clusters to be pulled apart farther down the consensus array, and therefore preventing edges from being formed. A simple pass over the array to move these clusters together would fix this, but the quality and confidence in the region would be extremely low. Ideally, the algorithm would produce a contig that matches, at a minimum, the length of the longest input read. As it stands however, this was not the case. With error rates potentially upward of 10%, it becomes difficult to resolve assemblies without sufficient underlying data to support decisions.

The main, high quality contig for each of the assemblies was ~10 Kbp, depending on the specific parameters used (Table 2.3). This covered the region of the test data set that represented a higher coverage (3-10x, position ~300 to ~10,000 in Figure 2.8; assemblies shown in Figure 2.10). This demonstrates that, with sufficient data, large contigs can be generated. From the data, it seems that this contig might have been greatly extended by simply increasing coverage over the last 20 Kbp region.

The algorithm is capable of assembling sequence data. However, there is still room for improvement, and error rates with the current version are too high. Small improvements have the potential to greatly improve results. These improvements would primarily include polishing the data at various steps to resolve ambiguous cases before they interfere with the overall algorithm. Strategies that may solve some of the issues with the assembly are discussed below.

2.2.7.2 The impact of the lookahead parameter

The main statistic that was predicted to change by increasing lookahead was the number of edges in each alignment graph. The reason for this is that because the algorithm looks farther ahead for candidate neighbours (*i.e.,* kmers that could potentially be connected via an edge), it was more likely to identify new edges by chance. Furthermore, in cases where the clustering algorithm caused a large separation, due to the median positions of adjacent clusters being shifted apart, a smaller lookahead would be more likely to miss this connection as an edge. A larger lookahead, on the other hand, would be more likely to make such a connection. This predicted trend was indeed observed (Table 2.3). When lookahead was increased from 3 to 8, the number of edges increased by 35.4%.

The connections being made between clusters that were pushed apart had a second key impact; the length of the best contig could be substantially increased. Using a weight function of one and clustering value of 5, varying the lookahead from a value of 3 to 10 caused the contig length to increase by a factor of 1.55x (Table 2.3). Similarly, when back- and front-weighting were used, varying the lookahead increased the longest contig length by a factor of 1.52x, in both cases (Table 2.3). This represented an average increase of 4,493 bp (averaged between a weight function of one and forward weighting; Table 2.3). This matches our prediction, as it is possible that two adjacent regions in the graph are connected by only a single edge, and if the lookahead is not long enough to identify this edge, then the contig will be broken. In addition to the improvement in contig length, improvements in identity when compared to the Unicycler assembly were observed. Varying lookahead from 3 to 8 yielded an average increase in identity of 4.45% (average between a weight function of one and forward weighting). Neither of these statistics improved by increasing the lookahead from 8 to 10, suggesting that a limit to the improvement due to lookahead is ~8 bp.

While the contig length increased with increased lookahead values, the average cluster size (~coverage) decreased slightly when all other parameters were fixed (Table 2.3). This is due to the contig extending into regions of lower coverage in the test dataset. The region of the assembly that is covered by all reads is only ~4 Kbp long (from position ~1,800 to 5,900 in Figure 2.8). As assemblies extend beyond this region, the average coverage by reads decreases, in turn limiting the maximum cluster size. For this reason, the use of cluster size (and by extension, coverage) as a quality metric needs to be approached with some caution. This higher-coverage region of the contigs generated by different parameter sets are highly similar, but the lookahead parameter caused a significant change in the length of the contigs (Figure 2.11).

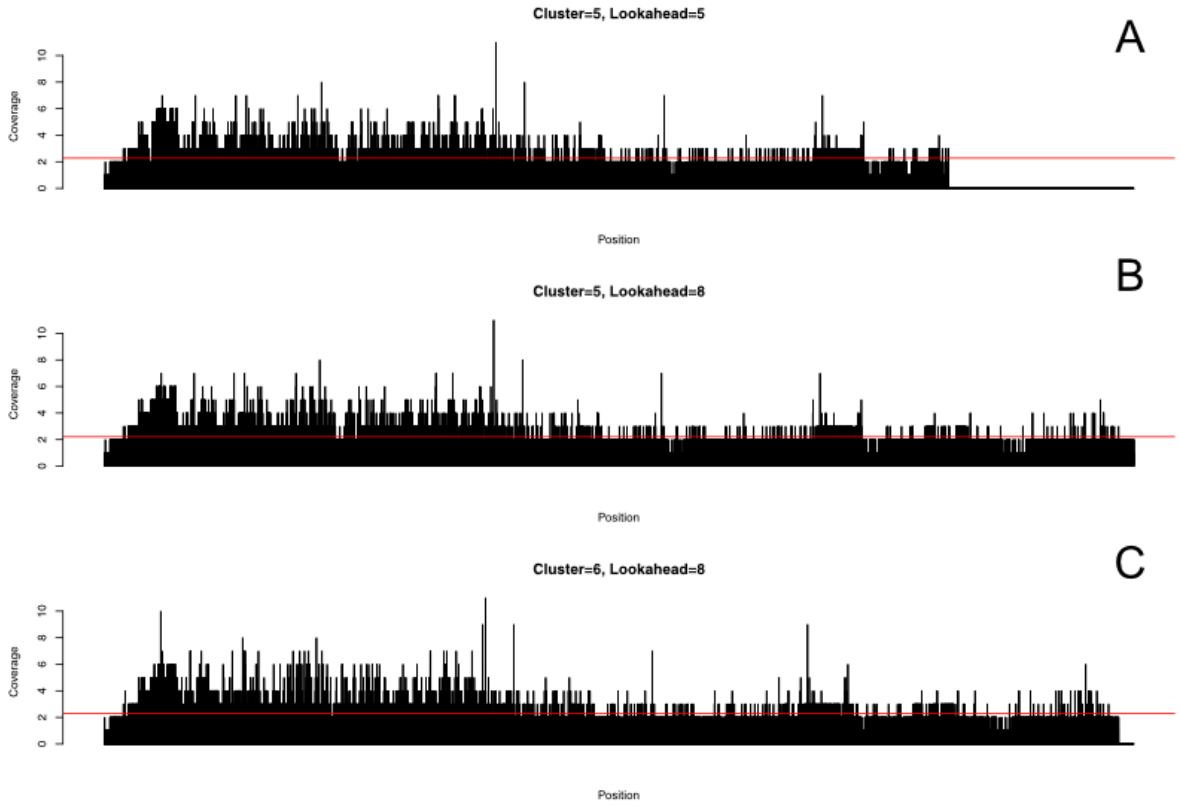

Figure 2.11: Cluster size map for three different parameter sets, using forward weighting for all three maps. Panel A to B illustrates the difference when lookahead is varied, while panel B to C illustrates the difference when cluster range is varied. The red line on each plot indicates the mean cluster size, with the means being 2.27 (A), 2.21 (B), and 2.30 (C).

2.2.7.3 The impact of the cluster parameter

The clustering parameter was predicted to have the biggest impact on the approximated contig coverage (*i.e.* average cluster size). This prediction was largely due to the fact that cluster size was used to directly approximate the coverage at each position. For a perfect dataset, the cluster for every position should have as many kmers represented within it as there are reads spanning that section of the consensus sequence. In practice, this is unlikely to be the case, due to the various errors that reads introduce to the kmer clustering. The most common errors in long read sequencing are insertions and deletions. In practice, this means that, when an error is present, kmers will occur at positions that are slightly offset from where the true consensus actually occurs. By increasing the cluster size, it is more likely that these shifted kmers will be captured within the correct cluster. However, increasing the cluster size parameter can introduce other problems. As the cluster size increases, it also becomes more likely that a repeated kmer will be

captured in a given cluster, when it should instead be kept separate. This would result in an assembly with more deletions. For example, when the clustering value was set to 6, with a lookahead of 8 and forward weighting, there were deleted regions with lengths of up to 42 bp. This leaves two questions: what cluster size minimizes these sorts of errors without lowering assembly statistics, and are these deletions reparable (and if so, how?) Over-clustering like this will cause certain clusters to become larger (due to their representing two points of the consensus) and leave gaps in the consensus array where the cluster that should have been kept separate would have appeared in the consensus sequence. This differential cluster size could be used to identify where over-clustering was occurring and could be used to repair these spots, by splitting the larger cluster to fill the gaps that were introduced in the array. The specifics of this solution, and others like it, will be discussed in detail in section 2.2.8.

2.2.7.4 The impact of the weight function

The weight function had the biggest impact on the quality of assembly, and this makes sense. The naïve implementation, in which all edges were given a weight of one, meant that in the case of a split in the graph, where one read diverged from the rest, there was no reason for the algorithm to prefer the correct path, and so consensus sequence decisions were entirely arbitrary in these cases. In certain cases, however, a weight of one would mean that large insertions in a single read would be specifically preferred, since these would maximize the path length. On the other hand, in cases where one read has a deletion, the deletion will be ignored in favour of the longer path generated by the correct reads. Assemblies made without using a weight function (assembly_cluster-5_lookahead-8, assembly_cluster-8_lookahead-8, assembly_cluster-10_lookahead-8, and assembly_cluster-4_lookahead-5) generally have more errors (see % identity in Table 2.3), as well as spurious insertions. The weighting function improved the identity to the Unicycler reference assembly and removed many of the insertions (Figure 2.12). The weight function may represent the single most important improvement to the algorithm, with a difference in identity of >10% between the assemblies with and without the weight function (Table 2.3 and Figure 2.13).

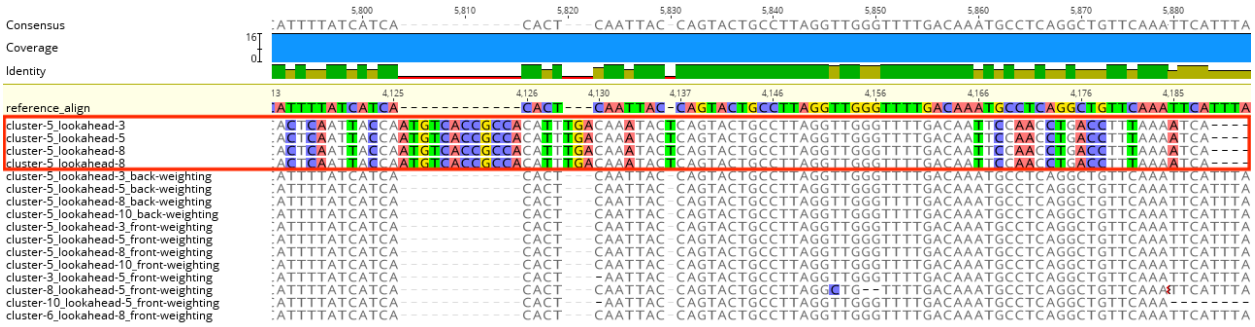

Figure 2.12: Sample region of the test assemblies aligned to the Unicycler reference. Parameters for each assembly are given on the left. The red box indicates the assemblies that used no weight function (*i.e.,* a value of one was assigned to the weight of each edge). Notice the quality of the assemblies with no weight function compared to those with, relative to the Unicycler reference assembly (highlighted in yellow at the top).

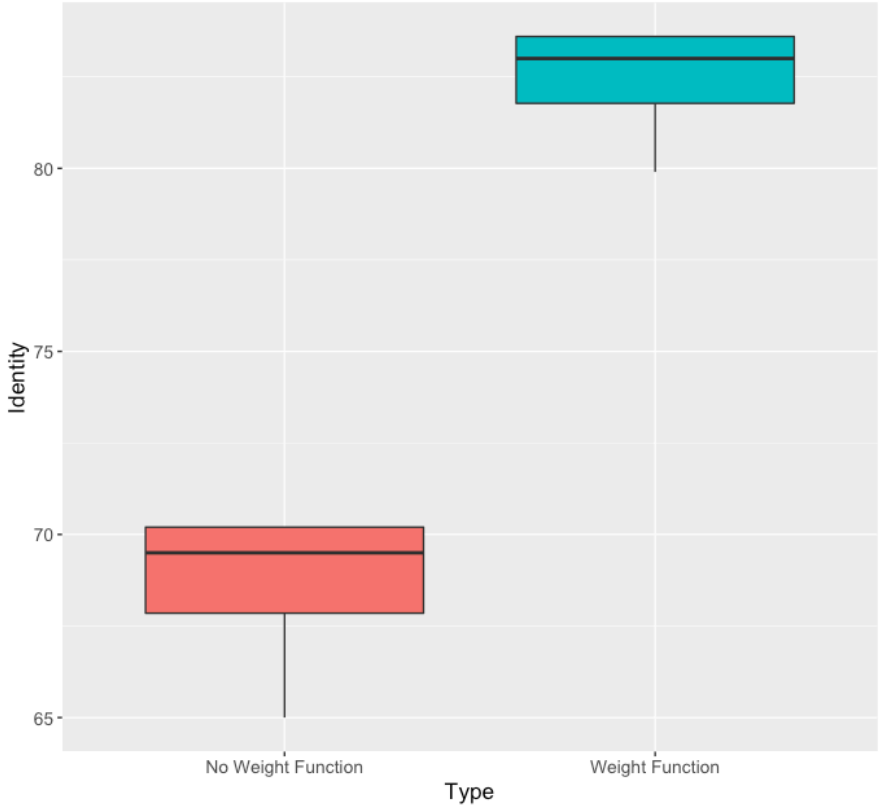

Figure 2.13: Boxplot comparing the percent identity to the Unicycler reference for the assembly algorithm using no weight function (salmon) and using a weight function (teal). Identities used for the weight function were the same for both back- and front-weighting, so the front-weighting data were used here.

Two choices for weight function, other than the naïve approach of assigning a weight of one to all edges, were explored in our testing: back- and front-weighting. Both follow similar concepts, and produced very similar results. For an edge that goes from vertex *u* to vertex *v* in the alignment graph, the weight can be defined by the coverage that the associated vertices represent. Here, a large value means that more of the reads contained exactly the kmer in question, for that position. There are two choices that were explored when deciding how to apply this coverage value. First, the support that vertex *u* represents (*i.e.,* the source vertex) is a possible weight or second, the support that vertex *v* represents (*i.e.,* the destination vertex) is another possible weight. These are called back- and front-weighting, respectively. Both of these weightings should produce identical assembly results in most cases as, if a particular vertex is well supported, then it will likely be chosen in both cases. When all other parameters were fixed, back- and front-weighting produced the exact same assembly statistics (Table 2.3). For this reason, in our assembly testing front-weighting was selected as the default weight function. We compared a proper weight function (front- or back-weighting) with assigning a value of one to all edges: the average size of the contig clusters increased by an average 1.33x when all other parameters were fixed. The average longest contig length, on the other hand, decreased by about 4%. The decrease in contig length was likely due to selecting fewer insertions from single reads, favouring the shorter, but more supported, path. Overall, the algorithm was able to follow a better-supported path through the graph when provided coverage weights for the edges. No other parameter had as significant an impact on the coverage as the weight function did. One final point to note is that changing the weight function did not change the number of edges. This is because the weight function has no bearing on the construction of the graph itself – it is only used when determining the path through the graph.

2.2.7.5 Types of errors

To profile errors, the assembly with the highest identity score was aligned to the reference. This was the assembly with a cluster parameter of 5, a lookahead of 8, and with front-weighting. MUSCLE (v3.8.425; Edgar, 2004) was used to locally align the sequence within the Unicycler reference assembly, and then Geneious' annotation tools (Kearse *et al.*, 2012) were used to identify variants between the two sequences (Table 2.4). The overall identity of this pairwise alignment improved, from 83.6% to 85.3%, an artefact of different alignment algorithms. The original alignment mapped all assemblies using the Geneious mapper, whereas the alignment between the reference assembly and the best assembly from our algorithm was produced with MUSCLE (in Geneious). The MUSCLE alignment using the Geneious predicted 571 variants in our assembly. Of these, 341 (59.7%) were indel errors. The remaining 230 (40.3%) were

substitutions of one or more base pairs. Of the insertions, 15 were tandem repeats, 13 of which were extensions of homopolymers by a single nucleotide. The remaining two were dinucleotides that were repeated. There were 112 other insertions of varying length (1 to 26 bp). 214 of the remaining indel errors were deletions. Of these, 78 were deletions in tandem repeats, 64 of which were shortenings of homopolymers. There were a further 136 deletions of varying length (1 to 33 bp). Base pair changes were mostly single nucleotide changes (130), with a further 100 substitutions ranging from 2 bp to 7 bp.

Table 2.4: Counts of error types in the best assembly (cluster=5, lookahead=8, front-weighting) compared to the Unicycler reference assembly. Errors were predicted using Geneious.

Error type	**Frequency of occurrence**
Substitutions (1 or more bp)	230
Deletions	214
Insertions	127
Total	571

2.2.7.6 Assembler-induced duplication errors

Of the detected errors, 22.2% were insertions. Many of these insertions represented tandem duplications. While it is difficult to pinpoint the exact cause for these errors, the most likely issue has to do with incorrectly separating a cluster because one or more of the reads was shifted outside of the clustering range of one or more of the other reads. This would result in two nearby clusters with the same kmer. When the longest path was traced, it would trace through the correct placement of the kmer via the correctly placed cluster, and then trace through the second misplaced cluster, causing the kmer to be repeated in the sequence (see Figure 2.14). Adjusting parameters to target these errors may prevent them, but other strategies, including data polishing steps, may be more effective, and will be discussed in section 2.2.8.

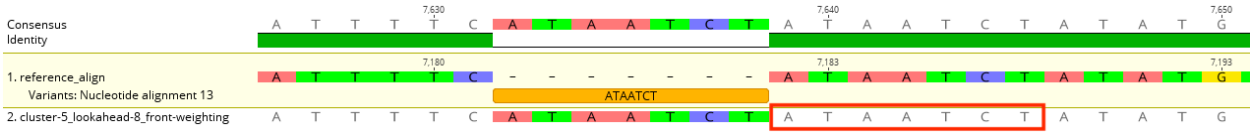

Figure 2.14: Example of a tandem repeat error. The 7 bp region occurs twice: first at the position highlighted with the orange bar, and again at the correct position, highlighted with a red box.

2.2.7.7 Small deletion errors

The second type of error that was common were small (1-33 bp) deletions. These would often occur when a kmer appeared twice in close proximity. In this case, both instances of the kmer would be clustered together and placed at a single position, rather than the two separate positions at which they should occur (Figure 2.15). These types of errors will cause the cluster size to be much higher than the cluster sizes of immediate neighbouring regions. This is because the cluster has incorporated twice (or more) as many kmers as it should have. This cluster size signal means that these deletions can be readily identified and corrected.

Adjustments to the available parameters could fix these types of deletion errors, but may create other problems. The main parameter that had an impact on these sorts of deletion errors was the cluster size. By reducing cluster size, it is less likely that kmers occurring in the same local stretch of reads as one another would be clustered together incorrectly. However, a reduction in the cluster range means that kmers that should be clustered together are less likely to be when small indel errors are present in the reads. This could generate more spurious edges in the graph, which would increase the likelihood that incorrect paths in the assembly graph are chosen.

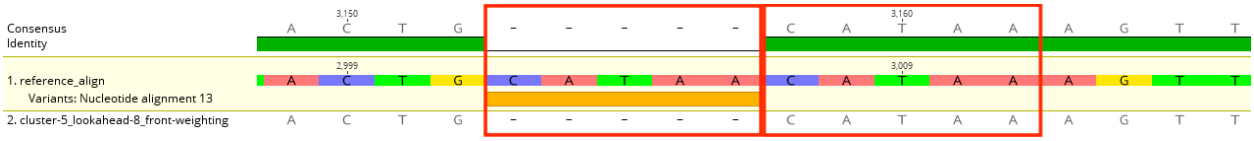

Figure 2.15: Example of a small deletion caused by a tandem repeat. The kmer CATAA shows up twice (highlighted in the red box), but the assembly has deleted the first instance of it.

2.2.7.8 A note on bug induced errors

With any software, it is also important to keep in mind that programmer error will likely exist somewhere in the code. This is inevitable in nearly any sizeable code base. Testing certainly reduces these sorts of errors, but test cases never represent all possible inputs. As a result, there are always going to be situations that may cause strange behaviour and introduce errors. It is important that this idea is discussed, at least in brief, in the context of this assembler. It is likely that there are at least some errors due to a

calculation, or incorrect logic in part of the code which has caused the algorithm to produce an erroneous result. For this reason, continued testing as the algorithm is refined will be required.

To minimize bug-induced errors, a variety of checks and tests were implemented. First, many functions in the code have checks that prevent unexpected parameters from being passed to them. This helps to define the scope of testing required by limiting the number, or range, of possible inputs that can be received by the different parts of the code. In addition to this, a unit testing suite was developed as part of this project. This was used to quickly add new tests that targeted small parts of the code to help guarantee that the individual components of the overall codebase (*i.e.,* the units) were functioning correctly. The current version of the code has a further 204 unit tests that demonstrate how to use the code and are directly embedded into it. Finally, large-scale testing through comparison to the Unicycler assembly was used to assess the overall performance of the algorithm. In spite of this depth of testing, programming errors are still possible. Overall, however, the code is performing as expected and producing assemblies that reasonably match established assemblers.

2.2.8 Outlook and future directions

Currently, the algorithm works; it is capable of producing contigs that are, mostly, reflective of the DNA from which the input reads were derived. In particular, the alignment algorithm is able to resolve contigs spanning well-covered regions. However, two open questions remain: 1) how capable is the algorithm of handling more varied data sets, especially much larger data sets? and 2) how can the tandem repeat and small deletion errors be reduced? It is likely that the answer to the first question is that the algorithm will behave similarly to the test set, resolving contigs with read depths greater than 2x and with error rates decreasing below 15% as the coverage increases. A concern is that the algorithm may take significantly longer with larger datasets. While the algorithm is mathematically linear, practical coding decisions can sometimes lead to larger increases in running time than expected. A potential remedy to longer run times, should they arise, would be to make small efficiency improvements. These would involve optimizing data structures to free up more memory (filling memory can lead to something known as thrashing, which can cause a program to slow down significantly), as well as finding pieces of code that are either not necessary, can be sped up with more efficient functions, or can be parallelized. The second question will be the focus here, as it is the aspect that is more relevant to the biology of the problem.

There were several sources of error in the program that lead to the observed error profile. In most cases, at least one of the kmer clusters available for the algorithm to select was incorrect, so error minimization largely becomes a problem of avoiding the clusters that are either incorrectly ordered due to a

substitution, or are incorrectly ordered due to misplacement. Both of these issues can be minimized with different data cleaning techniques.

In general, the most common kmer (*i.e.,* the one with which the most reads agree) at a given position will be the correct one. However, errors still arose in the regions with higher coverage, meaning that additional coverage may not be a sufficient solution. A potential solution to minimize errors like this would be singleton scanning, or identifying clusters supported by only a single read. Once a singleton is identified, neighbouring clusters can be searched to see if any of their kmers differ by only one nucleotide or position, indicating that the singleton may have had a point mutation or single base insertion. These erroneous singletons can then be corrected and merged into the correct cluster, in order to match the better-supported kmer. This technique would ultimately lead to a reduction in the complexity of the assembly graph, making it less likely that incorrect paths are chosen by mistake. Singleton scanning disambiguates the decision between two paths that differ at a single position. This usually will not be an issue, since the use of weight functions should promote accurate decisions between these paths. However, in regions of low coverage, multiple paths could be problematic in the absence of singleton scanning.

Another method to polish the data would be to perform coverage analysis. Over-clustering will cause kmers to be incorrectly grouped to one position if identical kmers occur in close proximity to each other. This sort of error leaves a distinct signal. Peaks and nearby valleys in cluster size (an approximation of coverage) would be a strong indication that over-clustering is occurring. By splitting the erroneous cluster and identifying the position that is missing a cluster, the cluster sizes would return to average levels. This would potentially fix gaps in the consensus array, accurately place identical kmers in close proximity, and improve the likelihood that the correct path is chosen. In addition to solving the tandem repeat issue, this data refining would prevent small deletions that result from an edge passing the position where the duplicated kmer should have occurred.

The final major data polishing that could lead to better assembly results, and specifically longer contigs, would be to scan the consensus array for positions that are either missing clusters, or have multiple different clusters. In the test data set, these positions led to many breaks when the coverage was low (<3x). To refine breaks in the consensus array, they would first have to be identified, along with the misplaced cluster, then this cluster must be shifted into the blank position. Alternatively, if there are no extra clusters, then the downstream clusters would have to be shifted backward to close the gap. Once the consensus array has been refined, the path could be identified in the normal way, and the final consensus produced.

The above refinements are the major algorithmic improvements that could be made, but certainly there are other improvements that are potential avenues for streamlining the program. Further unit testing might reduce the number of errors being generated by mistakes in the code. Identifying regions of the code that are redundant or not necessary may reduce the number of steps taken at certain processing stages. Improving memory management by releasing unused data structures could improve efficiency when working with larger datasets. The code could be assessed for potential parallelization, enabling a speed up by using more computer cores. And finally, decisions about algorithm parameterization can be made that give better results, without the need to clean or polish the data. Our original tests suggest the weight function being used to identify the longest path may be a key parameter for optimization.

In addition to these optimizations to the algorithm itself, it will be necessary for the algorithm or a subsequent step to connect contigs so that the final consensus sequence produced by the algorithm is at minimum the length of the longest input read. While there may be cases where this is not desirable or achievable, this approach would increase assembly length. Future work should aim to ensure the contigs produced are at least as long as the longest input read, assuming that the read is correctly aligned to the other reads.

The goal of this project was to develop an algorithm capable of assembling erroneous read data. The algorithm developed works at a proof-of-principle level. The alignment algorithm was able to produce contigs which matched a reference assembly with up to 85.3% identity using low coverage, high error long reads. While this level of accuracy may be useful in certain contexts, such as trying to understand genome architecture or the order of certain known genes, it is a relatively low quality assembly and could result in erroneous conclusions. This would be especially true if research questions are directed at highly specific sequence differences, such as single nucleotide polymorphisms with implications in altered protein function. As discussed, we anticipate that this accuracy could be further increased, but the core algorithm does perform an assembly. The algorithm was carefully and consciously designed to run in linear time, meaning that the time it takes to generate a consensus is proportional to the size of the input data and will grow accordingly, a desirable feature that can be challenging to achieve with these kinds of algorithms. Further, design criteria were selected to minimize the memory footprint, specifically the use of binary representations of sequence data. This helps to reduce the computational power required to assemble genomes. This alignment algorithm, along with the read ordering algorithm, has the potential to form the basis for a powerful long read genome assembler.

Chapter 3 Lessons from macroecology: Adapting occupancy modelling to the global methane cycle

3.1 Introduction

The Paris Agreement drafted in 2015 stipulated that signatory countries should work to keep the global average temperature less than 2ºC above the average temperatures prior to the Industrial Revolution (Paris Agreement, 2015). However, recent evidence suggests that increases in atmospheric methane will pose a significant challenge to achieving this goal (Fletcher and Schaefer, 2019). In the 800,000 years before the Industrial Revolution, the time span for which we have measurements, atmospheric methane concentrations varied from about 350 to 800 parts per billion by volume (ppbv; Spahni, 2005; Loulergue *et al.*, 2008). In the 200 years since the Industrial Revolution, methane emissions have been rising, nearly without interruption (Schaefer *et al.*, 2016). By 2017, levels had reached 1,850 ppb (Nisbet *et al.*, 2019).

The underlying causes of methane emissions are not well understood. From 1999 to 2007, methane emissions plateaued, after which atmospheric methane increased by ~8.3 ppb (Dlugokencky, 2003; Rigby *et al.*, 2008; Dlugokencky *et al.*, 2009). Atmospheric chemical oxidation of methane has been proposed as the reason for the 1999-2007 plateau (Rigby *et al.*, 2017; Turner *et al.*, 2017). There are still conflicting explanations for observed changes in emission rates (Prather and Holmes, 2017), which must be better understood to allow accurate climate modelling.

It has been estimated that the global warming potential (GWP) of methane over a 100 year period is over 20, compared to the GWP of CO_2, which is defined as 1 (Lashof and Ahuja, 1990). In 2017, the Canadian Government reported an estimated 93 metric tonnes of carbon dioxide equivalents of methane emitted to the atmosphere, representing about 13% of Canadian greenhouse gas emissions (National inventory report: greenhouse gas sources and sinks in Canada, 2017). Relative to carbon dioxide, methane is a highly potent greenhouse gas, and a significant portion of Canadian emissions.

3.1.1 The microbial methane cycle

The majority of methane emissions originate from microbial processes (Schimel, 2004). The release of methane, thought to be as a result of organic decomposition, was first noted in the 18th century (Volta, 1777). In 1910, it was shown that methane was produced biologically (Söhngen, 1910). Just over two decades later, the first pure culture of a methanogen was obtained (Stephenson and Stickland, 1933).

Since this time, many other methanogens have been isolated (*e.g.,* Edwards and McBride, 1975; W. Jack Jones *et al.*, 1983; Ladapo and Barlaz, 1997; Uchiyama *et al.*, 2010; Wu and Lai, 2011).

The other major group of organisms implicated in the methane cycle are the methane oxidizers. The oxidation of methane by bacteria was first proposed in the early 20th century (Söhngen, 1906). Methane oxidizing organisms largely fall into two categories: the bacterial methanotrophs (Hanson and Hanson, 1996), and the archaeal anaerobic oxidizers of methane (AOM; Hinrichs *et al.*, 1999; Cui *et al.*, 2015). Since Söhngen proposed bacterial oxidation of methane, many methanotrophs have been isolated (Lidstrom, 1988; Wise *et al.*, 1999; Svenning *et al.*, 2003).

These two microbial groups are the main controls on biological methane cycling. Each group is important for its aspect of the cycle and in controlling the flux of methane emissions. The rates at which methanogenesis produces, and methanotrophy consumes, methane ultimately result in the net methane production of different environments. Methanogens and methane oxidizers, as well as techniques to identify each group, are described in brief below.

3.1.1.1 Methanogenesis

Methanogenesis is predominantly considered an archaeal process (Borrel *et al.*, 2019), with a small amount produced as a by-product of other microbial pathways (Zheng *et al.*, 2018). There are three major types of methanogenesis, which contribute in varying amounts to methane emissions. The most widespread, and likely ancestral form, of methanogenesis is hydrogenotrophic methanogenesis, which involves the production of methane from CO_2 and H_2 (Bapteste *et al.*, 2005; Thauer *et al.*, 2008). The second type is acetoclastic methanogenesis, which involves the activation of acetate to acetyl-CoA (Ferry, 1992). Hydrogenotrophic and acetoclastic are the dominant forms of methanogenesis (Whiticar *et al.*, 1986; Xu *et al.*, 2016) and have been found to contribute nearly a third and two thirds of global methane, respectively (Kotsyurbenko *et al.*, 2004). In addition to hydrogenotrophic and acetoclastic methanogenesis, the third type is methylotrophic methanogenesis, which transforms various methylated compounds (Hippe *et al.*, 1979; Summons *et al.*, 1998). While it is generally considered a smaller contributor to global methane emissions (Xu *et al.*, 2016), methylotrophic methanogenesis can be a significant contributor in certain environments (Summons *et al.*, 1998). For example, methylotrophic methanogenesis accounted for up to 61.1% of methane emissions from intertidal sediments at Lowes Cove, Maine (King *et al.*, 1983).

All known methanogens, irrespective of the type of methanogenesis they perform, contain the *mcr* genes in their genomes (Borrel *et al.*, 2016, 2019). These genes encode the protein complex methyl-

coenzyme M reductase (MCR), responsible for the final step in methane formation (Grabarse *et al.*, 2001; Wongnate and Ragsdale, 2015). The *mcr* operon contains a suite of genes, but the three that encode for the MCR complex are *mcrA, mcrB,* and *mcrG*, which encode the α, β, and γ subunits, respectively (Reeve *et al.*, 1997). The MCR enzyme complex is an $\alpha_2\beta_2\gamma_2$ hexamer (Ermler *et al.*, 1997). The *mcrABG* genes have been used in combination or in place of 16S rRNA genes as methanogenic markers for some time (Lueders *et al.*, 2001; Luton *et al.*, 2002; Friedrich, 2005) and can be used to infer phylogeny of methanogens (Dziewit *et al.*, 2015). In addition, another component of methanogenesis is the gene encoding the *mcr* component A2 protein (*atwA*), which is thought to be involved in the activation of the MCR protein complex (Prakash *et al.*, 2014). The *mcrABG* and *atwA* genes are excellent biomarkers for methanogenic potential in an organism or environment, as they are universally present in known methanogens, and conserved enough to allow homology searching.

Methanogenesis has been observed across a wide variety of natural environments. It has been identified in various soils, including in wetlands (Angle *et al.*, 2017) and grasslands in the Austrian alps (Hofmann *et al.*, 2016). Methanogens have also been identified in marine sediments (Marchesi *et al.*, 2001; Kendall and Boone, 2006; Kendall *et al.*, 2007), black smokers and hydrothermal vents (Jones *et al.*, 1983; Ver Eecke *et al.*, 2012), as well as freshwater sediments (Ward and Frea, 1980; Whiticar *et al.*, 1986). Since the industrial revolution, anthropogenically-associated environments have become important in methane emissions. Rice paddies in Italy, China, and Japan have all been shown to contain methanogenic communities (Lueders *et al.*, 2001; Krüger *et al.*, 2005; Sakai *et al.*, 2007; Singh *et al.*, 2012), with some estimating that 10-25% of global methane emissions originate from rice paddies (Singh *et al.*, 2012). In Canada, landfills contribute nearly a quarter of national methane emissions (23.9%; National inventory report: greenhouse gas sources and sinks in Canada, 2017). Indeed, methanogens have been identified from landfills in a variety of studies (Luton *et al.*, 2002; Laloui-Carpentier *et al.*, 2006; Yadav *et al.*, 2015; Tang *et al.*, 2016). In addition, methanogens have been identified in the rumens of animals (Janssen and Kirs, 2008; Zhou *et al.*, 2009; St-Pierre *et al.*, 2015; Wang *et al.*, 2017; Stewart *et al.*, 2018). This is particularly important, as livestock are another major anthropogenic contributor to methane emissions.

3.1.1.2 Bacterial oxidation of methane

The majority of known methane oxidizers are aerobic bacteria. The methane oxidizing bacteria were traditionally grouped into three categories: type I, II, and X, based primarily on their use of different pathways downstream of methane oxidation (Hanson and Hanson, 1996). Type I methanotrophs use the ribulose monophosphate pathway (RuMP), type II use the serine cycle, and type X use the RuMP, but are

also capable of using the serine cycle, at lower levels (Lieberman and Rosenzweig, 2004). These categorizations largely match phylogenetic placements, where type I and X methanotrophs are members of the *Gammaproteobacteria* and type II are members of the *Alphaproteobacteria* (Knief, 2015). While distinction based on pathways has held fairly well, certain other characteristics that were used to distinguish the types (*e.g.,* membrane arrangement; Hanson and Hanson, 1996) have proven to not be universal (Knief, 2015). In addition to this, the discovery of methanotrophs in the phylum *Verrucomicrobia* (Dunfield *et al.*, 2007; Pol *et al.*, 2007; Islam *et al.*, 2008) has confused the categorization, since types I and II had become largely synonymous with members of the *Proteobacteria*. In addition, the methanotrophic members of the *Verrucomicrobia* do not possess the typical membrane structures associated with the typing system, and so are sometimes referred to as type III (Op den Camp *et al.*, 2009; Knief, 2015). As a result, there is a movement in the community away from the use of the type terminology (Op den Camp *et al.*, 2009; Knief, 2015). Another common characteristic for methanotrophs is that the majority are obligate C1 users (Lieberman and Rosenzweig, 2004), although recently species in the *Methylocella, Methylocystis,* and *Methylocapsa* (all type II *Alphaproteobacteria*) have been shown to grow on other carbon sources (Dedysh and Dunfield, 2011).

The oxidation of methane by is catalyzed by a methane monooxygenase (MMO), of which there are two types: soluble and particulate (Khmelenina *et al.*, 2018). The soluble form (sMMO) is localized to the cytoplasm, while the particulate form (pMMO) is membrane bound (McDonald *et al.*, 2008). These enzyme complexes are each encoded for by multiple genes. The pMMO complex encoded for by the *pmoCAB* operon, while sMMO is encoded for by the *mmoXYBZC* operon, which typically occurs in this conserved order (Shigematsu *et al.*, 1999; Gilbert *et al.*, 2000; Iguchi *et al.*, 2010). These genes, especially the gene for the A subunit of pMMO (*pmoA*) and the gene for the alpha subunit of sMMO (*mmoX*), are used as functional markers for aerobic methanotrophy (McDonald *et al.*, 2008; Dumont, 2014; Knief, 2015). However, neither gene can detect all methanotrophs. While the *pmo* operon is present in most methantrophs, it has recently been found to be absent in some species (Chen *et al.*, 2010; Vorobev *et al.*, 2011; for a review of species containing *mmo* and *pmo* genes, see Knief, 2015).

Similar to methanogens, methanotrophs have been identified from a wide variety of environments. These environments include freshwater lake sediments (Costello *et al.*, 2002), in marine environments (Lidstrom, 1988), in landfills (Wise *et al.*, 1999), and rice field soils (Henckel *et al.*, 1999). Recently, a group of methanotrophs belonging to the *Verrucomicrobia* were found to exist in highly acidic ($pH < 1$) environments (Pol *et al.*, 2007). Environmental stimulation of methanotrophs has also been examined, to

assess their potential to mitigate methane emissions (Lizik *et al.*, 2013). An understanding of where methanotrophs are found across the planet, and in particular if they are associated with the presence of methanogens, could strengthen bioremediation strategies and methane emission models.

3.1.1.3 Archaeal oxidation of methane

Anaerobic oxidation of methane by the ANME archaea is distinct from bacterial methanotrophy (Hinrichs *et al.*, 1999; Cui *et al.*, 2015). The ANME archaea use the methanogenic pathway in reverse to catalyze the oxidation of methane (Hallam *et al.*, 2003; Hallam, 2004; Wang *et al.*, 2014). These methane-oxidizing archaea are fascinating and likely important players in methane cycling (Lloyd *et al.*, 2006), but they will only be touched on briefly here. For the work described below, our focus was on oxidation of methane performed by bacteria, and specifically, bacteria utilizing particulate methane monooxygenase. The main relevance of the ANME archaea for this research was that *mcrABG* marker genes, canonically methanogenic markers, which were found to be closely associated with the ANME *mcr* genes, were filtered from the methanogenesis datasets. This will be discussed in more detail below.

3.1.2 Occupancy modelling

In any detection-based study, there is always the possibility that a detection was missed. This poses a problem in that a recorded absence may not indicate a true absence of the subject of interest from the surveyed environment. An absence may be either a result of the subject not being present or not being detected, and there is no way to tell the difference between these two cases. In macroecology, occupancy models are used to address this issue (MacKenzie *et al.*, 2002). Occupancy models work on the assumption that detection of a species can be modelled as the result of two statistical processes: the probability that the species is present at the site and the probability that, given the species is present, it was successfully detected. MacKenzie *et al*. proposed that this process can be thought of as two probabilities (2002). The first probability they called occupancy, which denotes the proportion of sites that a species occupies, denoted Ψ. This can be thought of as the probability that the species occurs in a given site. The second probability is the probability that the species is detected, which is denoted p. For a successful detection, occupancy and detection need to be successful. In order to estimate these values samples can be repeated and the detection probability can be determined, which in turn can be used to determine the occupancy proportion. This means, then, for each site there are multiple sampling occasions. Maximum likelihood methods can be used to identify the best parameters to fit the survey dataset.

A theoretically perfect model would have different values of Ψ and p for each site and sampling occasion, but in order to simplify the model to a workable level, several assumptions need to be met. These assumptions are:

i) The closure assumption, which states that there is no chance of the occupancy state changing between sampling occasions for the site, within the same season;

ii) The probability of occupancy is the same across all sites, or is otherwise modelled appropriately with covariates;

iii) The probability of detection is the same across all sites, or accounted for by covariates;

iv) The detection at each site is independent of detection at other sites; and

v) There are no false positives

These assumptions are important and must be considered when applying these models to any dataset. The other key element to this occupancy framework is that it was designed for a single species. However, the models have been expanded since the initial design in 2002. Important for the work here is the expansion of occupancy models to allow multiple species to be explored (MacKenzie *et al.*, 2004). These initial multi-species models made the assumption that one of the species was "dominant", meaning that patterns could be studied in only one direction. More recent multispecies occupancy models have been developed which drop this assumption of dominance, allowing multiple species to be examined as well as their interactions in both directions (Rota *et al.*, 2016).

3.1.3 Metagenomic data analysis and the incorporation of occupancy models

Culture-independent methods in microbial ecology, including the initial development of 16S and 18S rRNA gene amplification and sequencing (Hugenholtz *et al.*, 1998), have allowed deeper exploration of microbial diversity and distributions of functions across environments. Once the value of culture-independent sequencing was understood, it was not long before the idea of metagenomics emerged (Handelsman *et al.*, 1998), where genomic profiles could be captured for whole communities by extracting and sequencing the DNA from an environmental sample (Chen and Pachter, 2005). While there has been much progress in environmental sequencing strategies over the past two decades, and metagenomics has emerged as an established method for assessing microbial communities, metagenomics is still maturing. Particular challenges are identifying and applying valid statistical tests to metagenomic datasets, and in defining a sampling unit. Often, a single metagenome is thought of as a single sample, but

the cost and labour required to generate and analyze metagenomes is a barrier to obtaining replicate samples. The question then is, in these incredibly large datasets, are there other ways to define sample units that would allow application of statistics with some rigour?

Here, we describe a method to emulate macroecological re-sampling of an environment using a metagenomic dataset, by applying different searches to a single metagenome. The core idea is that, often, several genes are necessary for a given function to occur. As an example, multiple genes may encode required subunits of an enzyme complex. By applying independent searches for each gene, a single metagenome can be re-sampled for a function of interest and statistical methods can be robustly applied.

The goal of this research was to assess if metagenomic data could be used in occupancy modeling to assess functional co-occurrence. We adapted occupancy models (MacKenzie *et al.*, 2002) from macroecology to microbial ecology. These models were designed to estimate the occurrence of species, while taking into account the fact that detection probabilities are rarely perfect. We applied this approach to statistically test whether or not methanogens and methane oxidizing bacteria co-occur across the planet. To approach this problem, we mined nearly 10,000 metagenomes for multiple markers for methanogenesis and methanotrophy. These detections were the input to occupancy models used to model the occurrence patterns of both groups of organisms.

3.2 Methods

3.2.1 Reference set retrieval and curation

3.2.1.1 The initial reference set

An initial analysis used a reference dataset for the genes *mcrA* and *pmoB*. These sequences were retrieved from the National Centre for Biotechnology Information (NCBI) using BLAST (Altschul *et al.*, 1990). Reference sequences were manually selected to represent most known lineages (see Appendix A References for the 6K dataset). Protein sequences were size filtered with pullseq, using a minimum size of 275 amino acids for McrA and a minimum size of 190 amino acids for PmoB, which represented approximately 50% the length of the genes, respectively. The sequences were then aligned using MUSCLE (v3.8.425; Edgar, 2004) and verified with a maximum likelihood tree using FastTree (v2.1.5; Price *et al.*, 2010). These reference sets were used to build Hidden Markov Models using HMMER (v3.1b2; Eddy, 1998).

3.2.1.2 The full reference set

Reference sequences for twelve methane cycle biomarker genes were retrieved from the Genome Taxonomy Database (GTDB; Parks *et al.*, 2018) using AnnoTree (Mendler *et al.*, 2019; Table 3.1, Accessed February 11th, 2019). The Kyoto Encyclopedia of Genes and Genomes (KEGG; Kanehisa, 2000) Orthology (KO) was used for the annotations. Sequences were downloaded in CSV format. Any sequences from genomes that were either metagenome-derived or not taxonomically resolved to the species level were removed, unless there was supporting literature for the gene as a true representative of the function of interest (see Appendix B
Repository of online data). The final sequence sets were imported to Geneious (v11.0.2; Kearse *et al.*, 2012), aligned with MUSCLE (v3.8.425; Edgar, 2004), and maximum likelihood trees were inferred using FastTree (v2.1.5; Price *et al.*, 2010) to assess the quality of the reference datasets.

Table 3.1: KEGG orthology (KO) identifiers for the initial genes selected as potential biomarkers for methanogenesis and methanotrophy.

Gene symbol	Gene name	KO identifier	Function
mcrA	methyl-coenzyme M reductase alpha subunit	K00399	Alpha subunit of MCR complex[1]
mcrB	methyl-coenzyme M reductase beta subunit	K00401	Beta subunit of MCR complex[1]
mcrG	methyl-coenzyme M reductase gamma subunit	K00402	Gamma subunit of MCR complex[1]
atwA	methyl-coenzyme M reductase system, component A2	K00400	ATP-binding, role in protein activation[2]
pmoA	particulate methane/ammonia monooxygenase subunit A	K10944	Transmembrane domain[3, 4]
pmoB	particulate methane/ammonia monooxygenase subunit B	K10945	Active site containing domain[3]
pmoC	particulate methane/ammonia monooxygenase subunit C	K10946	Transmembrane domain[3,4]
mmoB	soluble methane monooxygenase regulatory protein B	K16160	Regulatory; controls access of substrate to active site (MMOB)[5]
mmoC	soluble methane monooxygenase component C	K16161	Reductase, converts between MMO_{ox} and MMO_{red} (MMOR)[5,6]
mmoX	soluble methane monooxygenase component A alpha chain	K16157	Alpha chain of hydroxylase (MMOH)[6,4]
mmoY	soluble methane monooxygenase component A beta chain	K16158	Beta chain of hydroxylase (MMOH)[6,4]
mmoZ	soluble methane monooxygenase component A gamma chain	K16159	Gamma chain of hydroxylase (MMOH)[6,4]

[1] Ermler *et al.*, 1997
[2] Prakash *et al.*, 2014
[3] Culpepper and Rosenzweig, 2012
[4] Murrell *et al.*, 2000
[5] Lee *et al.*, 2013
[6] Ross and Rosenzweig, 2017

3.2.2 Data collection and curation

3.2.2.1 Initial dataset collection from IMG

An initial dataset was collected from the Joint Genome Institute's portal for Integrated Microbial Genomes and Metagenomes (JGI IMG/M) by querying 5,868 metagenomes based on gene annotations (data accessed from May 23rd to July 18th, 2018). Genes that had been annotated with Pfam as either *mcrA* (PF02249) or *pmoB* (PF04744) were retrieved as translated amino acid sequences. Sequences were removed if they were shorter than a minimum length of 275 aa for McrA and 190 aa for PmoB. Qualitative filtering was performed using the custom Hidden Markov Models (HMMs; Eddy, 1998), based on the reference sets described in section 3.2.1.1. An e-value cut-off of less than or equal to $1x10^{-100}$ was used for PmoB and the default value of 10.0 was used for McrA. E-value cut-offs were determined by identifying the e-values associated with the reference set sequences, which were seeded into the data prior to the HMM search. Sequences which passed quality filtering were imported into Geneious for alignment with MUSCLE (v3.8.425; Edgar, 2004) and verification of phylogenetic congruence with the reference set using FastTree (v2.1.5; Price *et al.*, 2010). The final gene datasets were converted to presence/absence data using R. The presence/absence matrix consisted of rows, representing sites, and two columns, one for each gene. If a gene was present in a given metagenome, then the corresponding cell in the matrix was assigned a value of 1, otherwise it was assigned a 0. In the case of the gene being identified multiple times within a metagenome, this was still only marked as a 1. This dataset will be referred to as the 6K dataset (for its origin from nearly 6,000 metagenomes).

3.2.2.2 Full dataset collection and curation

An improved protocol was developed to allow better statistical analysis. This required generation of new datasets. Sequence data for twelve genes of interest (Table 3.1) were retrieved from 9,629 metagenomes stored on the JGI IMG/M based on gene annotations (data accessed between December 14th, 2018 and February 22nd, 2019). All sequences with the correct annotations were downloaded as FASTA amino acid files, along with corresponding metadata files linking the protein sequences to metagenomes (*i.e.,* tables containing the sequence identifier and the metagenome identifier). Size filtering was performed on the gene sequences retrieved from IMG/M using a minimum size of the shortest reference sequences less 50 amino acids and a maximum size of the longest reference sequence plus 50 amino acids. These criteria were applied to the pMMO and MCR (including AtwA) proteins, with

independent minimum and maximum sizes used for each gene set. For reasons that will be discussed below, sMMO protein sequences were not included for downstream analyses.

For a quality filtration step, reference sequences, including homologs not involved in methane cycling (*e.g.*, ammonium monooxygenases, butyrate-active Mcr homologs, *etc*.), were labelled as either true positives or false positives. DIAMOND BLASTp (v0.9.24; Buchfink *et al.*, 2015) was used to identify nearest matches to the reference sequences in a local copy of the UniRef50 database (release 2019_04 ; Suzek *et al.*, 2015). Database versions of the reference sequences were removed, to prevent redundancy, and the True/False-labelled reference sequences were seeded into the database. Top matches for each of the environmentally-derived sequences retrieved from IMG/M were identified from this modified UniRef50 database using DIAMOND BLASTp. Sequences from IMG/M were only retained if their top database match was one of the true positives from the reference sets. Passing IMG/M sequences were then imported into Geneious, aligned with the reference set using MUSCLE (v3.8.425), and maximum likelihood trees built using FastTree (v2.1.5). Any sequences that did not phylogenetically cluster with true positives in the reference set were removed. Sequences that were excessively divergent or which were on excessively long branches were removed as well.

The curated sequences were converted to a matrix of presence/absence data, where each row represented a single metagenome. A column for each of *mcrABG* and *pmoABC* were included (six in total) which denoted if a gene was present (a value of 1) or absent (a value of 0) for the given metagenome. In addition to these columns, metadata columns were included, which contained the geocoordinates (longitude and latitude), the sample add date (date that the sample was uploaded to IMG/M), as well as the ecosystem type (which was either 'Environmental', 'Host-associated', or 'Engineered'). The data were then aggregated into three different data sets. The first dataset had no aggregation, with each metagenome representing a single sampling site. The second dataset aggregated all metagenomes where the geocoordinates and ecosystem type matched into a single site. The final dataset aggregated metagenomes using only the geocoordinates. All of these steps were performed using R, with the aggregation done using the 'dplyr' package.

These datasets are collectively referred to as the "10K datasets" based on their origin from nearly 10,000 metagenomes.

3.2.3 Co-occurrence modelling

A basic co-occurrence analysis between the genes used in the 6K dataset was carried out using the R package 'cooccur' (Griffith *et al.*, 2016). This assessed co-occurrence between the genes *mcrA* and *pmoB* by comparing the frequency of co-occurrences in the 6K dataset when compared to a null model. The presence/absence data for the 6K dataset were used as input to the function 'cooccur' from the package 'cooccur', under default parameters. This assessed co-occurrence using a hypergeometric distribution, as described in Veech, 2013, to determine if the occurrence of these genes were either positively, neutrally, or negatively correlated.

3.2.4 Single-species occupancy modelling

To determine the viability of applying occupancy models to the full datasets, single species occupancy models were used to estimate occupancy (Ψ) and detection (p) of the different gene sets. Here, the detection, and thus genomic potential, for methanogenesis and methanotrophy were used as the "species" in the models. The individual marker genes were used to emulate the different re-sampling occasions at each site (*i.e., mcrA* represents one sampling occasion, *mcrB* represents a second, *etc.*) The sites were defined in three different ways: each metagenome was a single site, all metagenomes that had matching geocoordinates were aggregated to represent a single site, and all metagenomes that had matching geocoordinates as well as ecosystem labels were aggregated to represent a single site (described in section 3.2.2.2). In order to ensure that covariates could be properly incorporated into the models, sites with missing data were removed from the original dataset, as this would make comparisons between models with different covariates impossible. The covariates that were used were: date that the sample was added, as a proxy for improvements in technology; the latitude the samples were taken, as a proxy for climate; and the ecosystem type, which was either 'engineered', 'environmental', or 'host-associated'. The "date added" covariate was converted to a numerical format, in days since the date 1970-01-01, and then square root transformed to improve model fits. The 1970 data was the default value for conversion to numerical format. In addition to this, the models were run using days since 2006-01-01, but this was found to have little impact on the results, and the square root transform was still needed. Models shown used the default 1970-01-01 start date. The latitude covariate was missing from 209 samples, so these were not used for subsequent analyses, as covariate data needed to be complete. In total, there were six model sets for single species: three for *mcr* (*A*, *B*, and *G*) and three for *pmo* (*A*, *B*, and *C*).

Data were separated into three data frames: the detection history for *mcr*, the detection history for *pmo*, and the covariates for each site. The detection histories were the presence/absence data for each gene set.

The separation into different data frames was done for the individual metagenomes dataset, as well as the aggregated data sets. The data were then loaded into a data object for the 'unmarked' package, using the function 'unmarkedOccuFrame'. This combined the site covariates and detection history data, and was done for both *mcr* and *pmo*. The function 'occu' was used to generate the estimates for the models, with the engine parameter set to "C" (uses faster C code for the underlying calculations, as opposed to the slower, native R code), and all other parameters set to default values. Models with different statistical parameterizations were run with these settings, then compared using the Akaike Information Criterion (AIC). The lowest AIC corresponds to the model which best explains the data (on the basis of information gain/loss). To further assess the model fits, the continuous covariate, latitude, was plotted against the occupancy probability (Ψ), with 95% confidence interval estimates. Empirical Bayes estimates were used to estimate the proportion of sites occupied for each of the models using the functions 'ranef' and 'bup' from the 'unmarked' package, with the stat parameter set to "mode".

3.2.5 Multi-species occupancy modelling

Multi-species occupancy models were used to assess co-occurrence using the function 'occuMulti' in the 'unmarked' package for R. This uses the multi-species occupancy model developed by Rota *et al.* (2016). This model is advantageous for our data in that it does not assume a dominant species. Multiple models were run using different parameterizations and compared using the Akaike Information Criterion (AIC), and used to predict the occupancy (Ψ) under different conditions (Table 3.2). To assess global co-occurrence patterns, the unparameterized model was selected to isolate co-occurrence from other effects, and to allow the model to approximate co-occurrence globally, rather than on a per-environment basis.

Table 3.2: Occupancy probability notation and interpretation for single and multi-species occupancy models.

Notation	Interpretation
$\Psi(mcr)$	Probability of occupancy of *mcr*
$\Psi(pmo)$	Probability of occupancy of *pmo*
$\Psi(mcr\|pmo)$	Probability of occupancy of *mcr*, given that *pmo* is present
$\Psi(mcr\|\text{-}pmo)$	Probability of occupancy of *mcr,* given that *pmo* is absent
$\Psi(pmo\|mcr)$	Probability of occupancy of *pmo,* given that *mcr* is present
$\Psi(pmo\|\text{-}mcr)$	Probability of occupancy of *pmo,* given that *mcr* is absent

More refined models were run with varying parameterizations using the covariates latitude (as proxy for climate), and environment type (host-associated, environmental, or engineered), as well as upload date. Both aggregated and un-aggregated data (as described above) were used. When these data were aggregated, choices had to be made for the other metadata fields. For the add.date field, the median date was used. For the environment type field, when sites were aggregated by geocoordinates alone, the decision of which environment type the aggregated site should have was ambiguous (*i.e.,* if a marine and a host-associated site were aggregated for having the same geocoordinates), and so environment types were not used in analyses with this dataset.

3.3 Results

3.3.1 Reference set curation and annotation

The reference set that was used for the 6K dataset consisted of sequences retrieved from NCBI using BLASTp. Sequences were manually selected to cover the diversity of known methanotrophic and methanogenic species as best as possible. In addition, ammonia monooxygenase (AmoA) sequences were retrieved as part of the PmoB reference set, to serve as known false positives and to help curate

environmentally-derived sequences retrieved from IMG/M. The PmoB reference set consisted of 25 PmoB sequences and 7 AmoA sequences. The McrA reference set consisted of 106 McrA sequences (see Appendix A
References for the 6K dataset).

The reference datasets from the 10K dataset were built by retrieving sequences from AnnoTree using annotation identifiers from the Kyoto Encyclopedia of Genes and Genomes (KEGG). There were relatively few sMMO reference sequences available. In addition to this, there is no straightforward way to treat *mmo* and *pmo* genes as representing the same function within occupancy modelling. For these two reasons, as well as the fact that *pmo* appears to be the more relevant methanotrophic marker gene, as it occurs in nearly all methanotrophs compared to the patchy distribution of *mmo* (Nakamura *et al.*, 2007), the *mmo* reference sequences were excluded from further analyses. Most sequences that were not resolved to the species level according to GTDB (Parks *et al.*, 2018) were removed, as well as most sequences that were metagenome-derived. Exceptions to these were sequences with methane cycling potential supported by the literature (*e.g.,* the metagenome-derived methylotrophic methanogens belonging to the *Verstraetearchaeota;* Vanwonterghem *et al.*, 2016). The final methanogenesis reference set consisted of 186 McrA sequences, 186 McrB sequences, and 185 McrG sequences, as well as 271 AtwA sequences. Because occupancy models are designed to work with the same number of sampling occasions for all species, the *atwA* gene was excluded from further analyses so that both the *mcr* and *pmo* gene sets consisted of three individual genes. The final reference set for methanotrophy consisted of 182 PmoA sequences, 181 PmoB sequences, and 240 PmoC sequences.

Both the *mcr* and *pmo* genes contain false positive clades embedded within their phylogenetic range, where homologous proteins are known to exhibit different substrate specificities. In the case of the *pmo* gene, these are the *amo* genes, involved in ammonia oxidation. For the *mcr* genes, these are ANME-associated genes, which are canonical *mcr* genes, but which act in reverse, with these organisms oxidizing methane instead of generating it. In addition to the ANME sequences, there are also divergent sequences within the *mcr* family which are thought to metabolise other short chain alkanes, such as butane and ethane (Singh *et al.*, 2017; Borrel *et al.*, 2019). Reference sequences belonging to these clades were kept in the reference sets and manually flagged as false positives. These false positives were used to screen metagenome-derived sequences during the quality filtering steps.

3.3.2 Retrieval, curation, and co-occurrence of the 6K dataset

The 5,868 metagenomes used for the 6K dataset sampled a variety of different environments, with the majority being either soil (25.5%), marine (19.7%), or freshwater (18.8%; Figure 3.1). The remaining 36% of metagenomes were from quite diverse environments, with no other environment type representing more than 5% of the total set of metagenomes, and 11.9% of the environments comprising different environments at less than 0.1% of the total dataset (amalgamated as 'Other' in Figure 3.1).

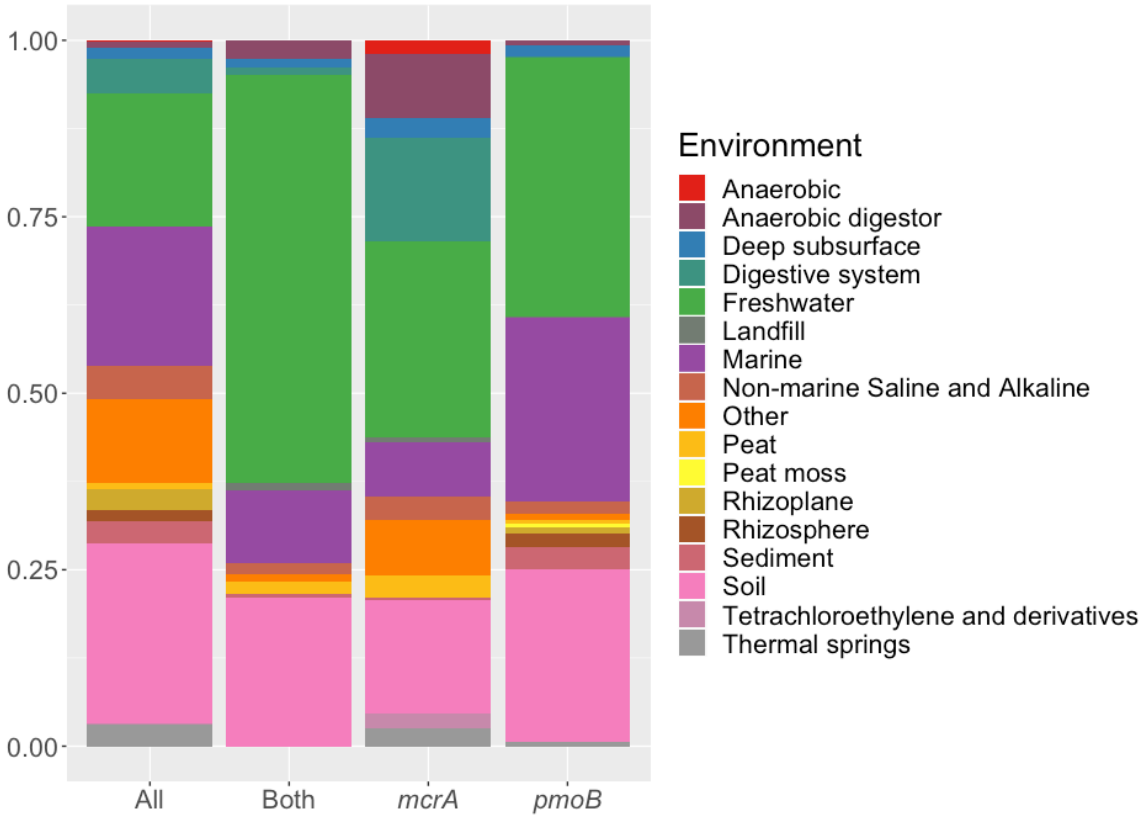

Figure 3.1: Stacked bar chart showing the distribution of environments types for the 6K dataset. From left to right, they represent all environments surveyed, environments containing both genes, and environments containing either *mcrA* or *pmoB*, respectively.

The initial search identified 3,998 putative McrA sequences and 35,480 PmoB sequences. After the sequences were filtered by size, 2,063 McrA sequences and 8,410 PmoB sequences remained. HMMER (v3.1b2) was used to automate removal of divergent sequences based on e-value scores against custom HMMs built from the reference sets. This left 2,061 McrA sequences and 2,920 PmoB sequences. Finally, phylogenetic congruence between metagenome-derived sequences and reference sequences was assessed

(Figure 3.2). From this, it was determined that 711 of the sequences in the PmoB tree were, in fact, AmoA sequences, which were discarded as false positives (blue clade in Figure 3.2).

Metagenome data, sequence data, and sequence names for the final curated sets of sequences were imported into R and used to create a presence/absence matrix. For this analysis, the sites were defined as the metagenomes, and no aggregation was performed. The ‘cooccur’ function from ‘cooccur’ showed a positive co-occurrence between these two – the model predicted that the two genes occurred together more often than expected under the assumption of the null model.

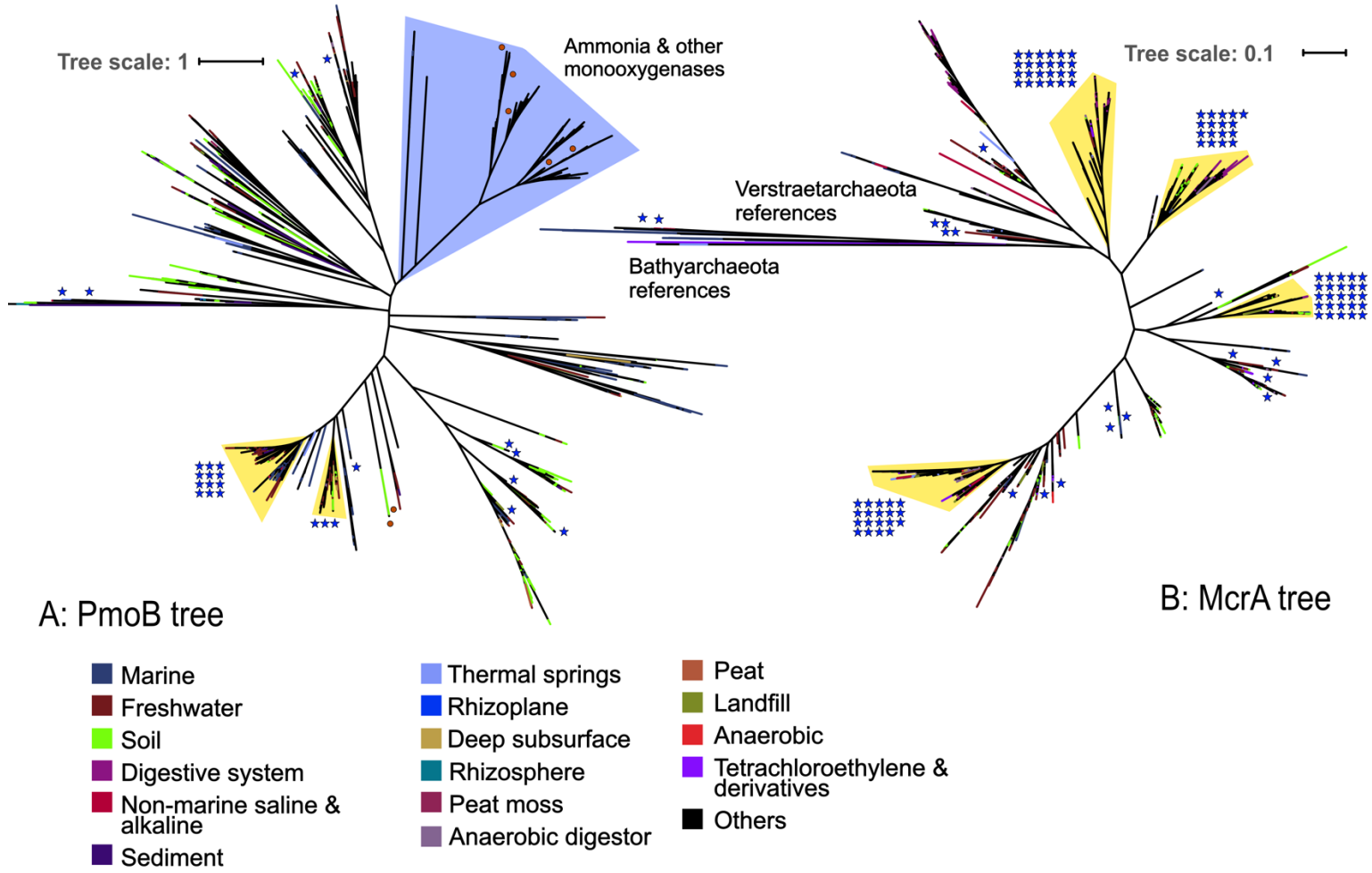

Figure 3.2: Phylogenetic trees with references and metagenome sequences used to curate the 6K dataset. Branches are coloured by environment type. Blue stars indicate the position of reference sequences. For the PmoB tree, orange circles indicate AmoA sequences, which identify false positive clades. Clades highlighted in yellow contain large numbers of references.

3.3.3 Retrieval and curation of the 10K dataset

The 10K dataset was derived from 9,629 metagenomes hosted on IMG/M. These metagenomes represented a variety of different environments distributed across the planet (Figure 3.3). A goal of the project was to test a method that was applicable on a global scale, and so it was important that the dataset represented a global sample. While regional bias certainly exists in the data, the metagenomes surveyed include samples from every continent and across the major oceans. In the Americas, Northern Canada and Greenland were under-sampled. While numerous samples have been sequenced from seas off the African coasts, inland Africa was largely unrepresented in the data. A large region in Asia, including regions of Russia, China, and India, also had relatively low representation. The strongest representation in sampling locations was for central America, the United States, and Europe. Samples across the oceans generally followed science cruise trajectories as transects, and large regions of the North and South Pacific had comparatively few samples.

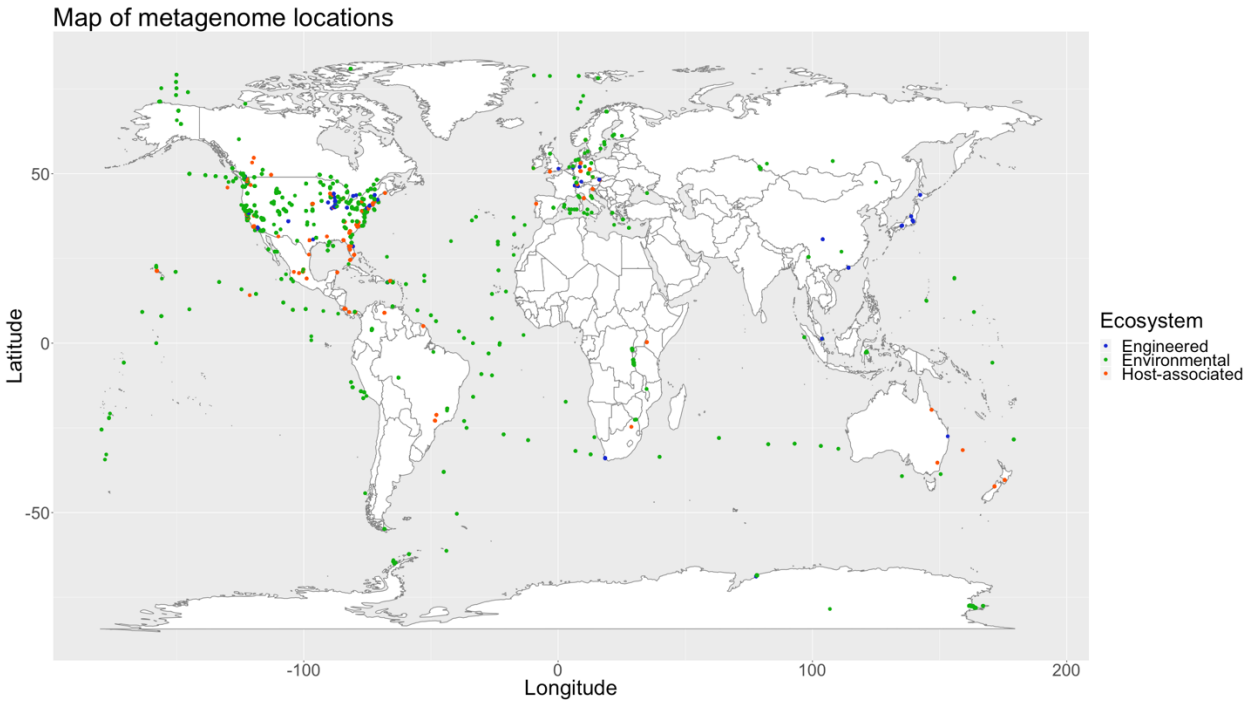

Figure 3.3: World map indicating the locations of the samples from which the metagenomes used in the analysis were sequenced.

The metagenomes were generated from samples representing many different environmental categories, which were broadly grouped by the IMG/M environment hierarchy as belonging to “environmental”, “host-associated”, or “engineered”. Within the IMG/M hierarchy, several different levels of

categorization were available (Figure 3.4). While other levels of categories would have provided more specificity for occupancy modeling, they were highly subjective and dependent on the researchers depositing the data to complete the hierarchy accurately. Given the subcategorizations were patchy and unreliable data, and that they increase the potential for errors due to over-parameterization, they were not used in downstream analyses. They are presented here to showcase the underlying data and depict potential biases due to certain environment types being over- or under-sampled.

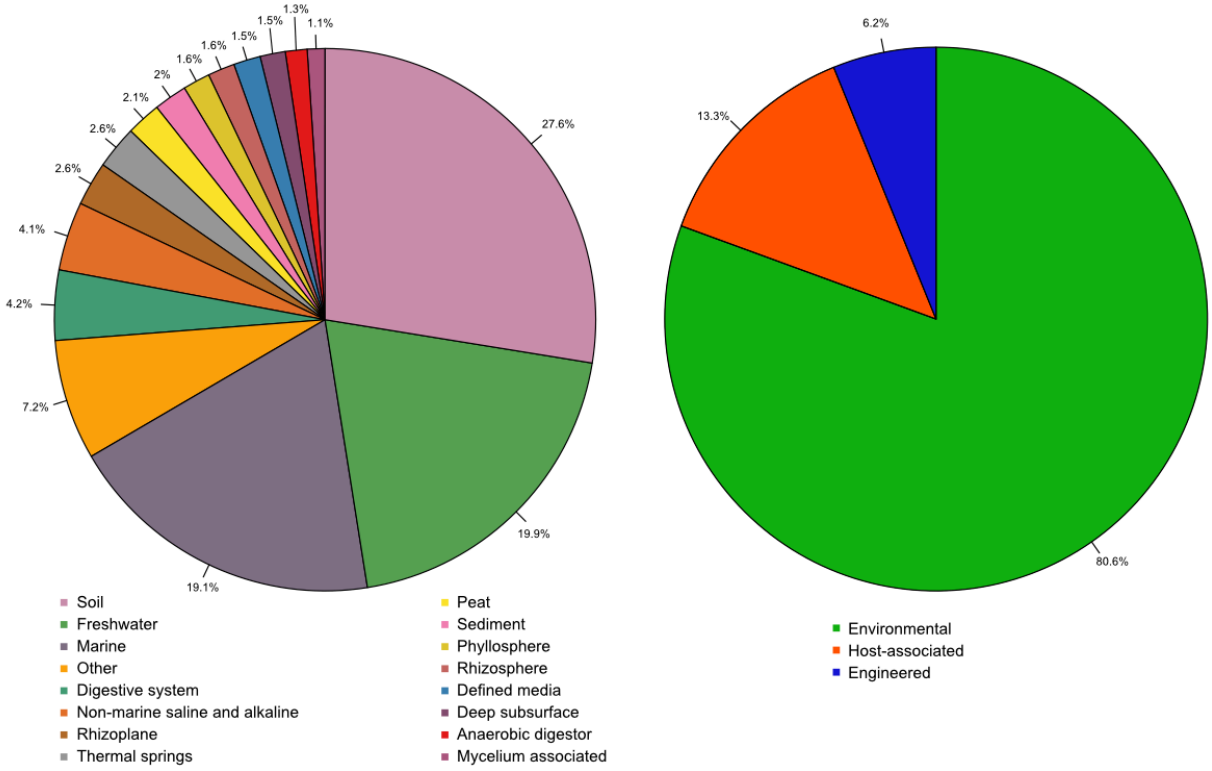

Figure 3.4: Environment sources for the 9,629 metagenomes at two levels of categorization. The more detailed (left) environment types were not used for models due to over-parameterization concerns following erratic model behaviour. The broader categories (right) were the ones used as the "Ecosystem" parameter in the actual models. The "Other" category represents metagenomes from all environments at less than 1% abundance in the full dataset.

Thirteen genes were searched for in the 9,629 metagenomes to build the 10K dataset: the *mcr* genes *mcrABG* and *atwA*, the *pmo* genes *pmoABC*, and the *mmo* genes *mmoBCDXYZ*. A total of 340,598 gene sequences were identified prior to filtering (Table 3.3). Due to the counts for the *mmo* genes being highly variable, and because *pmo* occurs in nearly all methanotrophs, whereas *mmo* does not (Nakamura *et al.*, 2007), the *mmo* sequences were not included in subsequent analyses. An added complication driving the

decision to remove sMMO was that because sMMO and pMMO do not always co-occur in the same organisms, they would have to be treated as separate functions in the models. This was possible to implement, but would conflate the question of functional co-occurrence from a modelling perspective. Because these models have not been used in this context before, it was decided that the analysis would be more robust if a simpler approach was taken. In addition to these points, the reference data for sMMO was not as complete (discussed above), so false positives were more likely to occur. The *atwA* gene was also removed from consideration to ensure methanogenesis and methanotrophy were each represented by three genes, as most occupancy models assume that all species (functions) are looked for at each sampling occasion, which means that each of the functions in the model should have the same number of sample events (here, genes). When selecting which methanogenesis marker gene to remove, *atwA* was chosen because the AtwA protein is not a component of the MCR complex, and is thus a more indirect biomarker.

Table 3.3: Unfiltered gene sequence counts for the 10K dataset.

Gene symbol	Count
mcrA	33,269
mcrB	25,977
mcrG	18,013
atwA	43,930
pmoA	51,840
pmoB	60,267
pmoC	72,503
mmoB	1,696
mmoC	3,559
mmoD	479
mmoX	15,590
mmoY	11,916
mmoZ	1,559
Total	340,598

We applied size filtering to curate the metagenome-derived sequences. This was accomplished by setting minimum and maximum length values for each of the six genes used in the final analysis. The minimum was defined as 50 amino acids less than the shortest reference sequence for the gene in question. The maximum was defined as 50 amino acids larger than the longest reference sequence for the gene in question. This was a relatively permissive threshold but helped to eliminate spurious sequences that matched to the KEGG annotations. Only sequences that were within these thresholds were maintained (Table 3.4). The size filtering greatly reduced the number of sequences. The reductions in *mcrA*, *mcrB,* and *mcrG* were 91.3%, 50.3%, and 60.0%, respectively. The reductions in *pmoA*, *pmoB*, and *pmoC* counts were 72.7%, 88.2%, and 79.1%, respectively. In total, approximately 77.3% of the initially identified sequences were removed based on size filtering.

Table 3.4: Summary of identified gene sequences before and after size filtering criteria were applied.

Gene symbol	Minimum reference length	Maximum reference length	Accepted length range	Sequences before filtering	After size filtering
mcrA	459	573	[409,623]	33,269	2,866
mcrB	331	479	[281,529]	25,977	12,898
mcrG	189	268	[139,318]	18,013	7,209
pmoA	217	294	[167,344]	51,840	14,177
pmoB	304	435	[254,485]	60,267	7,098
pmoC	189	267	[139,318]	72,503	15,142
Totals				261,869	59,390

In addition to the size filtering, DIAMOND BLASTp against reference data sets and assessment of phylogenetic congruence were applied. These filtration steps were primarily to remove genetically homologous genes which are not likely to be involved in the functions of interest. The DIAMOND BLASTp filtering step was automated, while phylogenetic congruence was assessed manually. Again, these quality filtering steps significantly reduced the number of sequences (Table 3.5). Of the *mcrA, mcrB*, and *mcrG* sequences, 7.6%, 69.5%, and 7.8% were removed during this filtering, respectively. With respect to *pmoA*, *pmoB*, and *pmoC*, 72.5%, 52.8%, and 52.0% were removed in these filtering steps.

Table 3.5: Summary of gene sequences removed during homology and phylogenetic congruence quality filtering steps.

Gene symbol	Sequences after size filtering	BLASTp filtered sequences	Phylogenetically congruent sequences
mcrA	2,866	2,695	2,648
mcrB	12,898	4,246	3,928
mcrG	7,209	6,762	6,649
pmoA	14,177	4,215	3,896
pmoB	7,098	3,512	3,347
pmoC	15,142	7,878	7,262
Totals	59,390	29,308	27,730

The final gene sequence sets were converted to presence/absence data for each metagenome. The genes covered a variety of environments, in differing proportions. Within a gene set, proportions, however, were similar. For example, *mcrA*, *mcrB*, and *mcrG*, tended to show up in roughly equal proportions within any given environmental category. It should be noted that the total number of environments containing these genes will be less than the numbers summarized above (Table 3.5), since multiple copies of the same gene occurring in a single environment were collapsed to a single "presence". The proportion of sites occupied for different environmental categories shows how the presence/absence data are distributed (Figure 3.5 and Figure 3.6). The raw presence/absence data are shown as co-occurrence matrices in Appendix D (Figures S1-3).

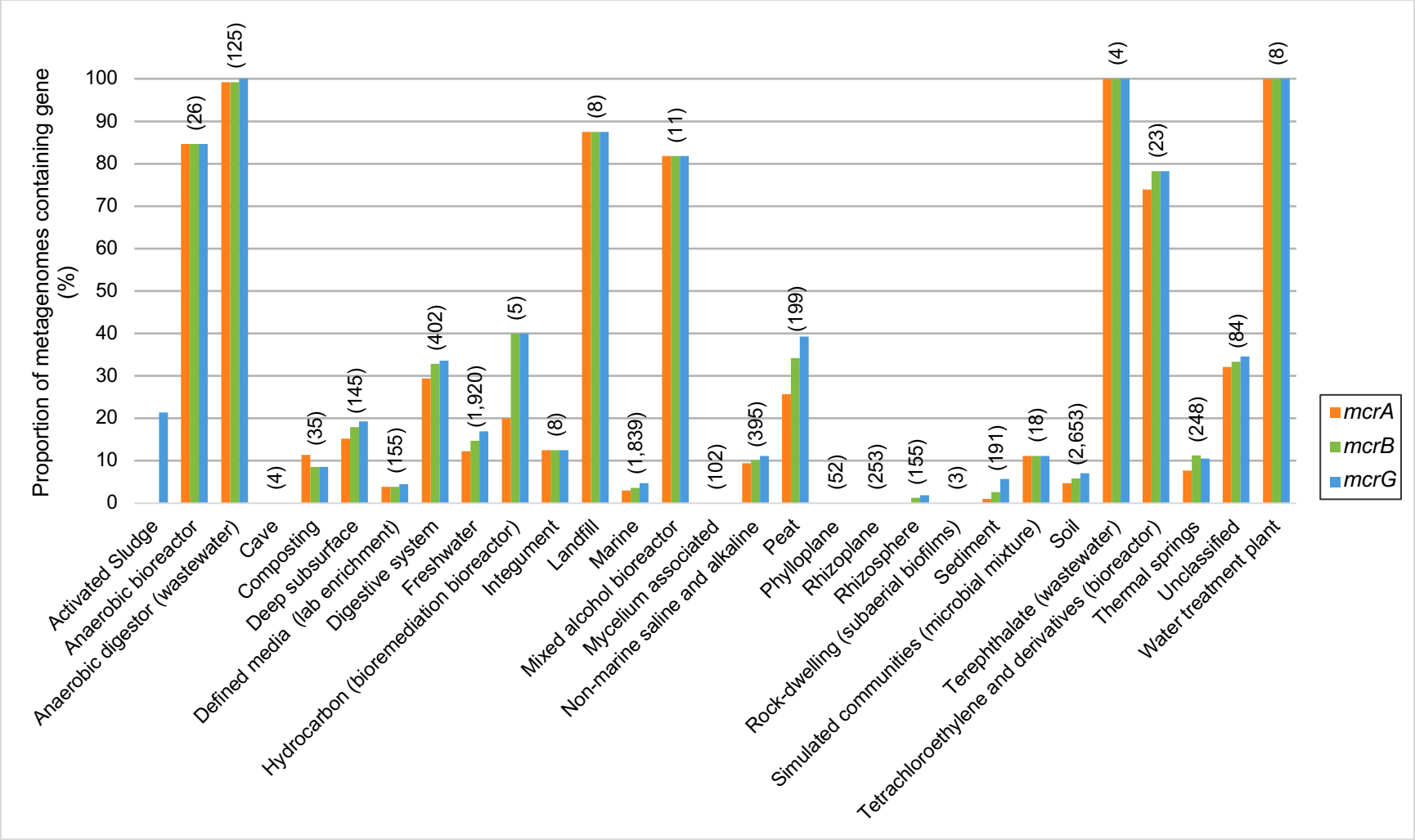

Figure 3.5: Proportions (as percentage) of environment type containing *mcr* genes. Numbers in parentheses indicate the number of metagenomes included in that category.

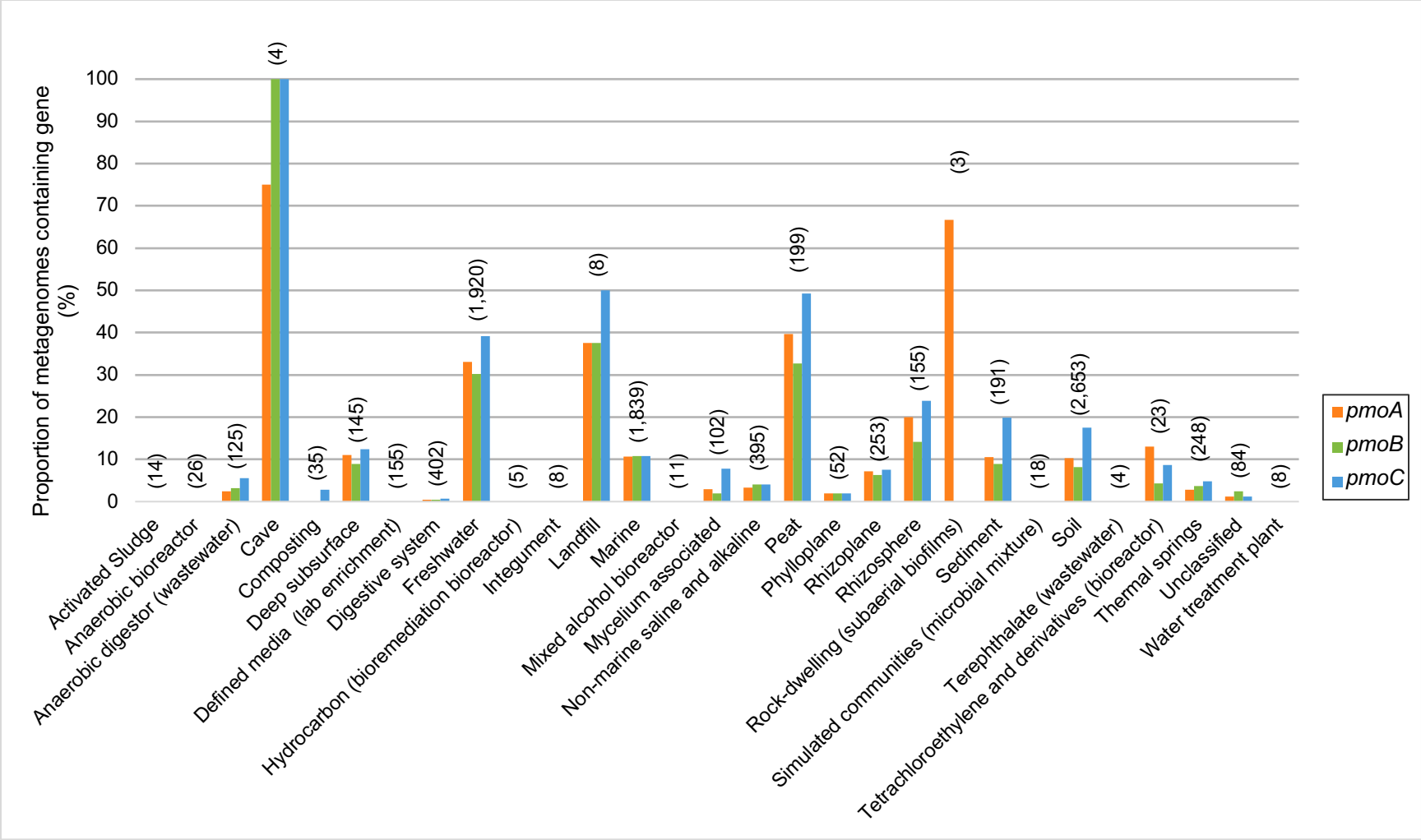

Figure 3.6: Proportions (as percentage) of environment type containing *pmo* genes. Numbers in parentheses indicate the number of metagenomes included in that category.

3.3.4 Single-species occupancy models

Three different metagenome treatment sets were used to assess the best models. The difference between these datasets was how sites were defined. For the first dataset, sites were defined as single metagenomes. The second data set aggregated metagenomes by their geocoordinates and their ecosystem type and each aggregated set of metagenomes was treated as a single site. In this case, ecosystem type could still be used as a covariate for the models. The final set aggregated metagenomes only on geocoordinates, which meant that ecosystem could not be used as a covariate for these models, as it introduced ambiguity (*i.e.,* if an environmental and a host-associated metagenome were aggregated).

For the un-aggregated dataset and the dataset aggregated by geocoordinates and ecosystem, five different parameterizations were used (Table 3.6 and Table 3.7). The parameterizations were the same for both data sets, as well as for both of the functions (methanogenesis and methanotrophy). For both *mcr* and *pmo* genes, under both the un-aggregated data set and the aggregated site sets, the best models (lowest AIC) incorporated ecosystem as a parameter for the occupancy (Ψ). All next best models incorporated latitude as a parameter for occupancy (Ψ), with the addition of sqrt(Numeric.Add.Date) as a parameter for detection (p) improving the AIC for three of four conditions (unaggregated *mcr*, both *pmo* sets).

The dataset that was aggregated using only the geocoordinates had two parameterizations: a base model with no covariates and a model with latitude as a covariate for occupancy. For both *mcr* and *pmo*, the models that incorporated latitude scored as stronger models.

The models were used to estimate the number of occupied sites. These estimates were generated using the posterior Bayes distributions and the 'ranef' and 'bup' functions included with the 'unmarked' package. Estimates were the same for all models within each dataset. For *mcr*, models for the un-aggregated data set produced an estimated 12.7% of sites occupied. The models for the data set aggregated by geocoordinate and by ecosystem estimated that *mcr* occupied 18.2% of sites. Finally, the models for the data set aggregated by geocoordinate only estimated that 18.6% of sites were occupied by *mcr*. The *pmo* models estimated that 20.1% of sites were occupied for the un-aggregated data set. The models for the *pmo* data set aggregated by geocoordinates and ecosystem had an estimated site occupancy of 24.0%, and models for the data set aggregated only by geocoordinate estimated 24.3% of sites were occupied by *pmo*.

Occupancy was estimated for both gene sets for each of the three ecosystem types (Figure 3.7). This was done for the un-aggregated data and the data aggregated by geocoordinate and ecosystem. These estimates could not be conducted for the data set aggregated only by geocoordinate, since ecosystem was

not a valid covariate for this data set. For both dataset types, *mcr* occupancy was estimated as being much higher than *pmo* for engineered systems. For environmental sites, estimates of *pmo* occupancy were higher, and in host-associated sites, the estimates for both functions were similar.

Occupancy distributions were calculated for across the range of latitudes (Figure 3.8). At more Southern latitudes (below ~50ºN), *pmo* was predicted to have a higher occupancy proportion. At more Northern latitudes (further North than ~50ºN), the occupancy for both *mcr* and *pmo* was approximately the same. This was the result of an increase in *mcr* occupancy more than a decrease in *pmo* occupancy, which exhibited relatively less change than *mcr* over the latitudinal transect. These results may be the result of sampling bias present in the data, especially due to the different geographies of the regions that were heavily sampled, versus regions that were under-sampled. These potential biases will be discussed below.

Detection probabilities increased with the square-root-transformed add date (Figure 3.9). This covariate was used to reflect advances in technology and techniques for obtaining metagenomic sequence data, as depth of sequencing and thus probabilities of detection have been steadily increasing. The *mcr* genes had a higher detection rate than *pmo* across the entire date range. Both estimates were relatively precise, based on the 95% confidence intervals, but this does not necessarily mean that the results are accurate. The detection probability for both functions seems to have increased a fair amount in the last decade. It is noteworthy that the detection curve for *mcr* is concave down, suggesting that improvements in detectability may be decreasing, as current methods are performing at, or near to, their highest capacity for *mcr* detection. Another possibility may be changes in the environments being sampled over time skewing the occupancy underlying the detection probabilities. The detection of *pmo*, on the other hand, showed rapid growth (concave up) for most of the first decade in the plot, but since about 2016, the curve has also become concave down, suggesting that detectability in *pmo* is now following a similar pattern to *mcr*. This result should be interpreted with care, however, since the detectability is near zero prior to 2008, which would suggest that sampling bias may be having a particularly profound impact on the results observed for upload dates. To test the impact that counting days from the default of 1970-01-01 may have had, the models used to analyze detection over the date range were run again using 2006-01-01 as the starting point to count days for numerical conversion of the dates. These seemed to have little impact on the results (Appendix D; Figure S4), and the AIC values were lower, indicating poorer fit of the models to the data. For the unaggregated data single species model, the AIC value for *mcr* with the modified date was 9,982.881, and for *pmo*, it was 15,807.38. These represent decreases in AIC values of

4.539 and 122.13, respectively. For *mcr*, this difference is quite small. For *pmo*, the difference is greater, but interpretation of these results is challenging, which will be discussed below.

Table 3.6: AIC values for single-species *mcr* models. Estimated proportions of sites occupied used empirical Bayes estimates. *p* is the probability a species (here, *mcr*) is detected, while Ψ is the occupancy probability of *mcr*. For both probabilities, the value following the tilde (~) is the covariate parameter applied. Best-performing models for each dataset are highlighted in grey. sqrt = square-root-transformed, dates counted from 2006-01-01.

Metagenomes as sites (9,420 sites)

Model	Number of parameters	AIC	ΔAIC	Estimated proportion of sites occupied
$p \sim 1$, $\Psi \sim$ Ecosystem	4	9691.23	0.00	1193
$p \sim$ sqrt(Numeric.Add.Date), $\Psi \sim$ Latitude	4	9934.10	242.87	1193
$p \sim 1$, $\Psi \sim$ Latitude	3	9958.31	267.08	1193
$p \sim$ sqrt(Numeric.Add.Date), $\Psi \sim 1$	3	9982.88	291.65	1193
$p \sim 1$, $\Psi \sim 1$	2	10007.15	315.91	1193

Aggregated by geocoordinates and environment (1,229 sites)

Model	Number of parameters	AIC	ΔAIC	Estimated proportion of sites occupied
$p \sim 1$, $\Psi \sim$ Ecosystem	4	1555.15	0.00	224
$p \sim 1$, $\Psi \sim$ Latitude	3	1594.58	39.43	224
$p \sim$ sqrt(Numeric.Add.Date), $\Psi \sim$ Latitude	4	1596.54	41.39	224
$p \sim 1$, $\Psi \sim 1$	2	1619.65	64.50	224
$p \sim$ sqrt(Numeric.Add.Date), $\Psi \sim 1$	3	1621.61	66.46	224

Aggregated by geocoordinates only (1,202 sites)

Model	Number of parameters	AIC	ΔAIC	Estimated proportion of sites occupied
$p \sim 1$, $\Psi \sim$ Latitude	3	1578.66	0.00	223
$p \sim 1$, $\Psi \sim 1$	2	1605.03	26.37	223

Table 3.7: AIC values for single-species *pmo* models. Estimated proportions of sites occupied used empirical Bayes estimates. p is the probability a species (here, *pmo*) is detected, while Ψ is the occupancy probability of *pmo*. For both probabilities, the value following the tilde (~) is the covariate parameter applied. Best-performing models for each dataset are highlighted in grey. sqrt = square-root-transformed, dates counted from 2006-01-01.

Metagenomes as sites (9,420 sites)

Model	Number of parameters	AIC	ΔAIC	Estimated proportion of sites occupied
p ~ 1, Ψ ~ Ecosystem	4	15669.51	0.00	1896
p ~ sqrt(Numeric.Add.Date), Ψ ~ Latitude	4	15678.85	9.35	1896
p ~ sqrt(Numeric.Add.Date), Ψ ~ 1	3	15807.38	137.87	1896
p ~ 1, Ψ ~ Latitude	3	15929.51	260.00	1896
p ~ 1, Ψ ~ 1	2	16053.68	384.17	1896

Aggregated by geocoordinates and environment (1,229 sites)

Model	Number of parameters	AIC	ΔAIC	Estimated proportion of sites occupied
p ~ 1, Ψ ~ Ecosystem	4	2284.47	0.00	295
p ~ sqrt(Numeric.Add.Date), Ψ ~ Latitude	4	2303.10	18.63	295
p ~ sqrt(Numeric.Add.Date), Ψ ~ 1	3	2305.06	20.58	295
p ~ 1, Ψ ~ Latitude	3	2329.47	45.00	295
p ~ 1, Ψ ~ 1	2	2330.83	46.36	295

Aggregated by geocoordinates only (1,202 sites)

Model	Number of parameters	AIC	ΔAIC	Predicted sites occupied
p ~ 1, Ψ ~ Latitude	3	2297.88	0.00	292
p ~1, Ψ ~ 1	2	2299.55	1.67	292

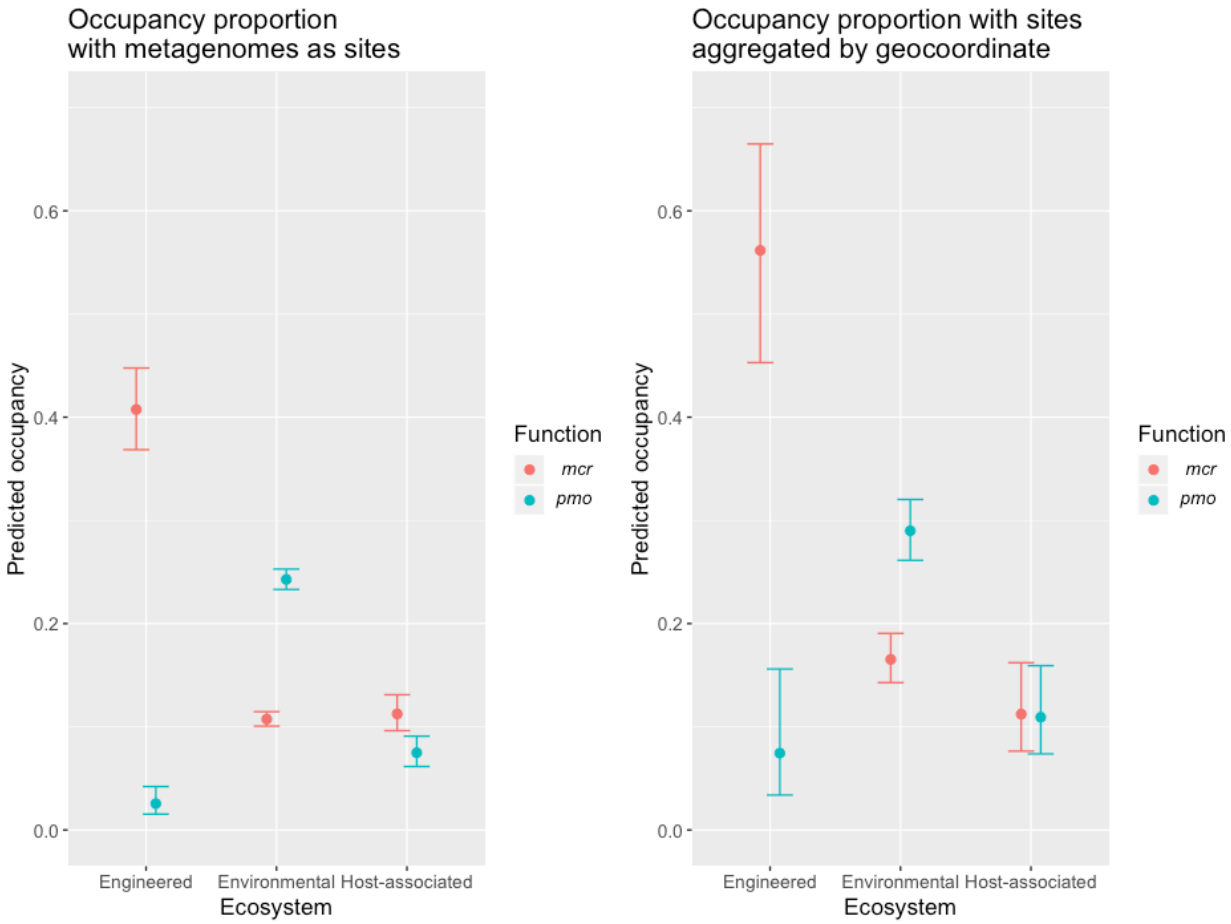

Figure 3.7: Predicted occupancy proportion for both *mcr* and *pmo* using the non-aggregated (left) and aggregated-by-geocoordinate-and-environment (right) datasets, by ecosystem type. In both cases, the model used was $p \sim 1$, $\Psi \sim$ Ecosystem. Error bars represent 95% confidence intervals for the estimates.

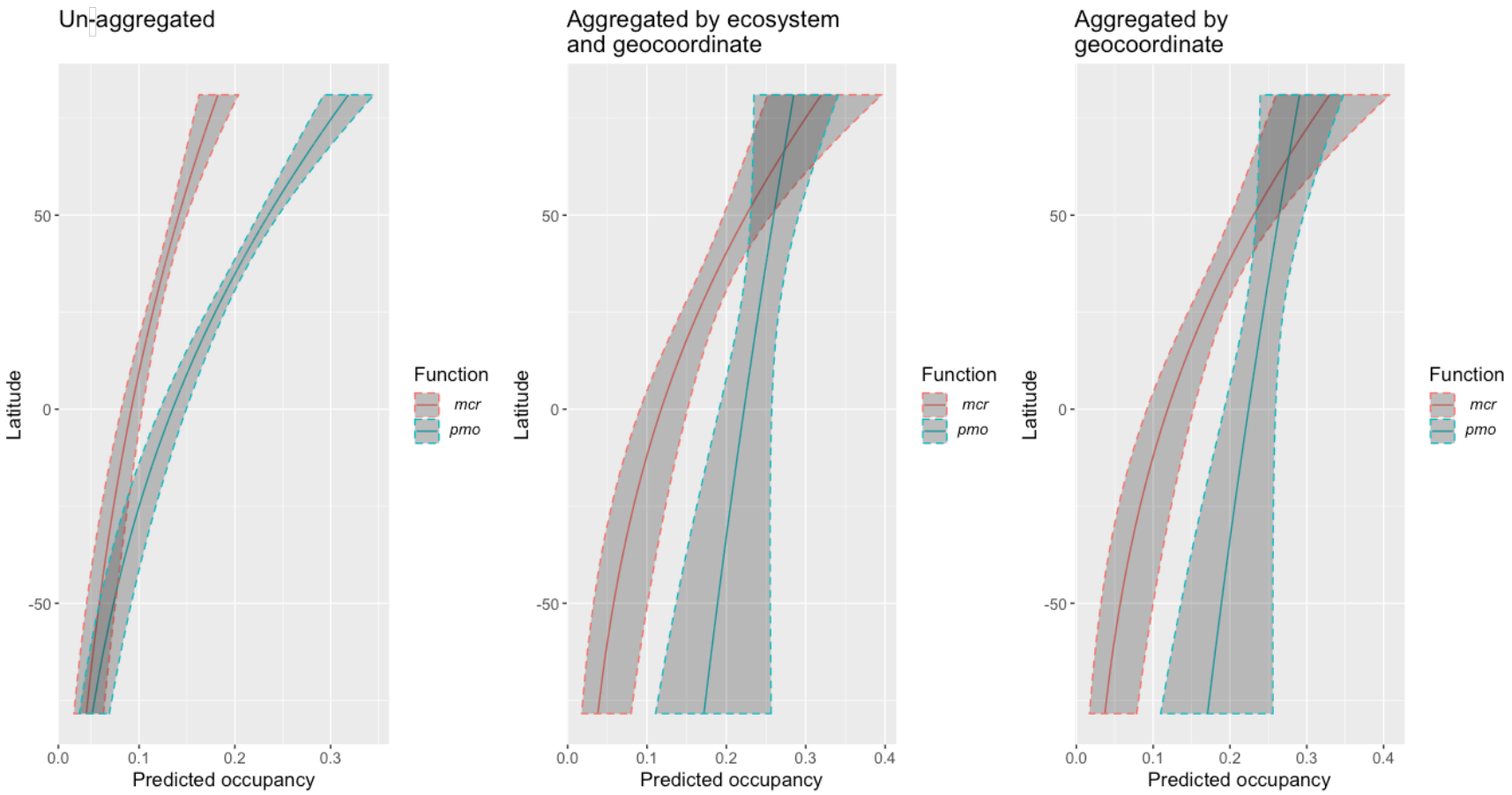

Figure 3.8: Estimated occupancy proportion versus latitude when using metagenomes as individual sites (left), aggregated data based on both geocoordinates and ecosystem type (middle), and aggregated data based only on geocoordinates (right). Darker grey windows bounded by dotted lines represent 95% confidence intervals for the estimated occupancies (solid lines).

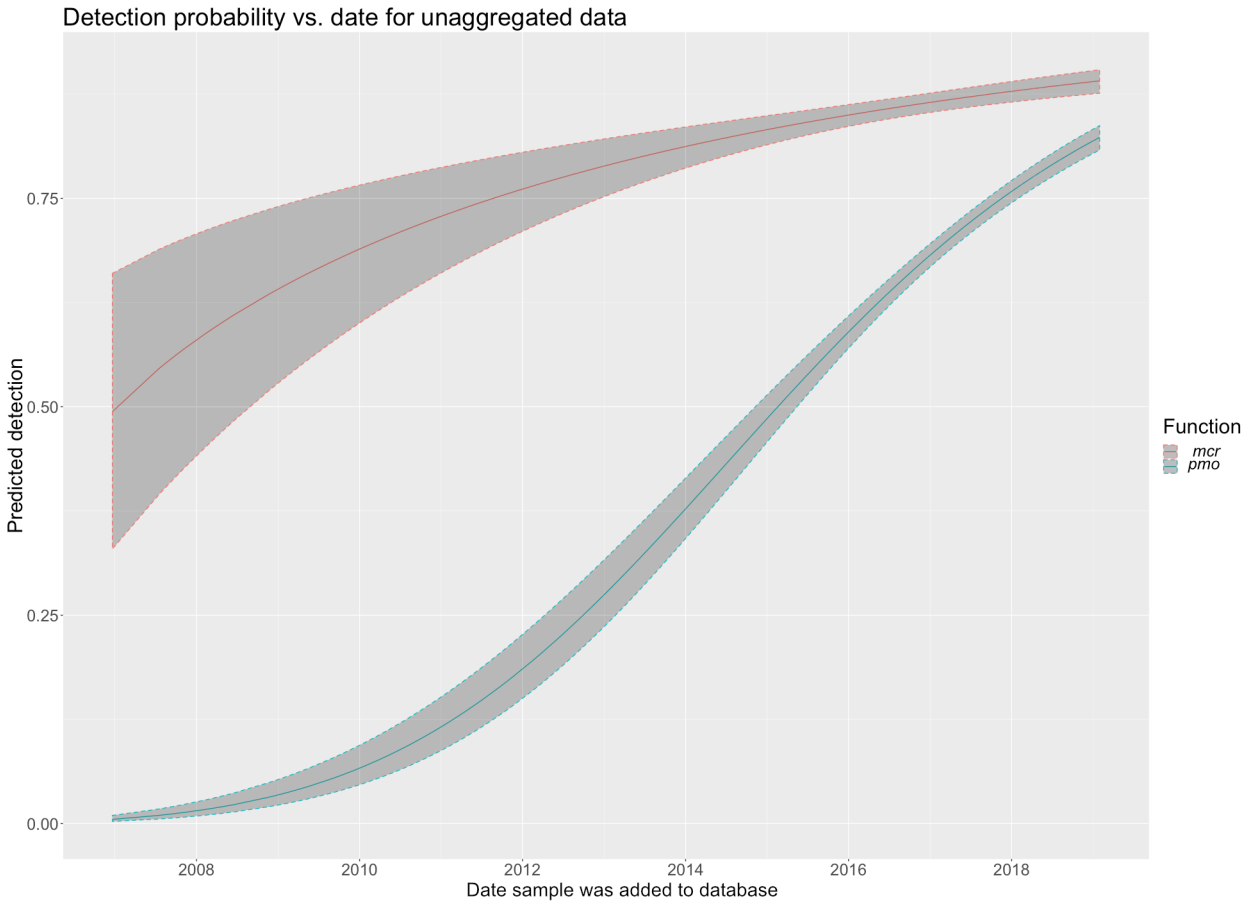

Figure 3.9: Detection probability (*p*) versus date as predicted by occupancy models using only the square-root-transformed add date for each metagenome (sqrt(Numeric.Add.Date)) as a covariate. Darker grey windows bounded by dotted lines represent 95% confidence intervals for the estimated occupancies (solid lines). Dates counted from 2006-01-01.

3.3.5 Multi-species occupancy models

To assess the co-occurrence of *mcr* and *pmo* across sites, the two datasets which included the ecosystem covariate (1: with metagenomes as sites and 2: aggregated on geocoordinates as well as ecosystems), were each modelled with occupancy models using 10 different parameterizations (Table 3.8). The order of best fitting models was the same for both datasets. In each case, the best models incorporated ecosystem as a covariate. In addition, when sqrt(Numeric.Add.Date) was incorporated for occupancy of *pmo*, the model further improved. The best model also incorporated ecosystem for the interaction term between *mcr* and *pmo*. Interestingly, when sqrt(Numeric.Add.Date) was incorporated for both *mcr* and *pmo*, the model did not perform as well as it did when this covariate was only incorporated for *pmo*. When the dates were

counted from 2006-01-01, errors were introduced, likely due to perfect detections or issues with convergence to parameters, so the default 1970-01-01 dates were used for these data.

Table 3.8: Occupancy models for the three data sets incorporating *mcr* and *pmo* as multiple species. Each model has three formulae: one for *mcr*, one for *pmo*, and one for both (denoted *mcr*:*pmo*). *p* is the probability of species detection, while *Ψ* is the occupancy probability of the given species (square brackets). For both probabilities, the value(s) following the tilde (~) is the covariate parameter(s) applied. Best-performing models for each dataset are highlighted in grey. sqrt = square-root-transformed, dates counted from 1970-01-01.

Metagenomes as sites (9,420 sites)			
Model	**Number of parameters**	**AIC**	**ΔAIC**
Ψ[mcr] ~ Ecosystem, *Ψ[pmo]* ~ Ecosystem+sqrt(Numeric.Add.Date), *Ψ[mcr*:*pmo]* ~ Ecosystem	12	24083.63	0.00
Ψ[mcr] ~ Ecosystem+Latitude, *Ψ[pmo]* ~ Ecosystem+Latitude, *Ψ[mcr*:*pmo]* ~ 1	11	24919.92	116.28
Ψ[mcr] ~ Ecosystem, *Ψ[pmo]* ~ Ecosystem, *Ψ[mc*:|*pmo]* ~ Ecosystem	11	25043.00	239.37
Ψ[mcr] ~ Ecosystem, *Ψ[pmo]* ~ Ecosystem, *Ψ[mcr*:*pmo]* ~ 1	9	25076.89	273.26
Ψ[mcr] ~ sqrt(Numeric.Add.Date), *Ψ[pmo]* ~ sqrt(Numeric.Add.Date), *Ψ[mcr*:*pmo]* ~ sqrt(Numeric.Add.Date)	8	25622.46	818.83
Ψ[mcr] ~ Latitude, *Ψ[pmo]* ~ Latitude, *Ψ[mcr*:*pmo]* ~ Latitude	8	25728.01	924.38
Ψ[mcr] ~ 1, *Ψ[pmo]* ~ 1, *Ψ[mcr*:*pmo]* ~ Ecosystem	7	25792.37	988.73
Ψ[mcr] ~ 1, *Ψ[pmo]* ~ 1, *Ψ[mcr*:*pmo]* ~ 1	5	25882.22	1078.59
Ψ[mcr] ~ Ecosystem+ sqrt(Numeric.Add.Date), *Ψ[pmo]* ~ Ecosystem+ sqrt(Numeric.Add.Date), *Ψ[mcr*:*pmo]* ~ 1	11	35276.17	10472.54
Ψ[mcr] ~ 1, *Ψ[pmo]* ~ 1 *Ψ[mcr*:*pmo]* ~ Ecosystem	13	35280.17	10476.54

Aggregated by geocoordinates and environment (1,229 sites)			
Model	**Number of parameters**	**AIC**	**ΔAIC**
Ψ[mcr] ~ Ecosystem, *Ψ[pmo]* ~ Ecosystem+sqrt(Numeric.Add.Date), *Ψ[mcr:pmo]* ~ Ecosystem	12	3745.72	0.00
Ψ[mcr] ~ Ecosystem+Latitude, *Ψ[pmo]* ~ Ecosystem+Latitude, *Ψ[mcr:pmo]* ~ 1	11	3757.98	116.28
Ψ[mcr] ~ Ecosystem, *Ψ[pmo]* ~ Ecosystem, *Ψ[mcr:pmo]* ~ Ecosystem	11	3776.41	239.37
Ψ[mcr] ~ Ecosystem, *Ψ[pmo]* ~ Ecosystem, *Ψ[mcr:pmo]* ~ 1	9	3781.61	273.26
Ψ[mcr] ~ sqrt(Numeric.Add.Date), *Ψ[pmo]* ~ sqrt(Numeric.Add.Date), *Ψ[mcr:pmo]* ~ sqrt(Numeric.Add.Date)	8	3878.95	818.83
Ψ[mcr] ~ Latitude, *Ψ[pmo]* ~ Latitude, *Ψ[mcr:pmo]* ~ Latitude	8	3885.95	924.38
Ψ[mcr] ~ 1, *Ψ[pmo]* ~ 1, *Ψ[mcr:pmo]* ~ Ecosystem	7	3888.03	988.73
Ψ[mcr] ~ 1, *Ψ[pmo]* ~ 1, *Ψ[mcr:pmo]* ~ 1	5	3908.36	1078.59
Ψ[mcr] ~ Ecosystem+ sqrt(Numeric.Add.Date), *Ψ[pmo]* ~ Ecosystem+ sqrt(Numeric.Add.Date), *Ψ[mcr:pmo]* ~ 1	11	5131.21	10472.54
Ψ[mcr] ~ 1, *Ψ[pmo]* ~ 1 *Ψ[mcr:pmo]* ~ Ecosystem	13	5135.21	10476.54

Aggregated by geocoordinates only (1,202 sites)			
Model	**Number of parameters**	**AIC**	**ΔAIC**
Ψ[mcr] ~ Latitude *Ψ[pmo]* ~ Latitude, *Ψ[mcr*:*pmo]* ~ 1	7	3839.91	0.00
Ψ[mcr] ~ Latitude, *Ψ[pmo]* ~ Latitude, *Ψ[mcr*:|*pmo]* ~ Latitude	8	3841.47	1.56
Ψ[mcr] ~1, *Ψ[pmo]* ~ 1, *Ψ[mcr*:*pmo]* ~ Latitude	6	3851.49	11.58
Ψ[mcr] ~ 1, *Ψ[pmo]* ~ 1, *Ψ[mcr*:*pmo]* ~ 1	5	3865.21	25.39

Because it can be more challenging to interpret the results of a model where *mcr* and *pmo* are modelled with different parameters, here the second-best model was used for data interpretation. This model incorporated latitude and ecosystem for both *mcr* and *pmo*, and applied no covariates for the interaction term Ψ [*mcr:pmo*]. All of the models predicted that the occupancy of a function increased if the other function was present. This held true for all three site definitions and across all latitudes (Figure 3.10, Figure 3.11, and Figure 3.12). In engineered sites, *mcr* was predicted to have a much higher occupancy than *pmo*, with *pmo* having near zero occupancy for these sites. Environmental sites were predicted to have similar occupancy rates for both groups. Environmental sites also had much less precise estimates, with 95% confidence intervals exceeding a range of predicted occupancy ±25% for the dataset aggregated by only geocoordinates. The two aggregated datasets showed a relatively small change in occupancy estimates for *pmo* over the range of latitudes, whereas all models, for all environment types, showed an increase in *mcr* occupancy as latitude increased. As with the single-species occupancy models, these results should be interpreted with caution, since sampling bias may strongly impact the results. The data show various biases in the samples that were available. For example, the varying geographies could have substantial impact on the results. This would include issues such as the distribution of deserts between about 30ºN and 30ºS. In addition, there are likely discrepancies in the freshwater and marine samples, resulting from the distributions of these types of environments. Ultimately, these are driven by the heavy bias to sampling the Northern hemisphere and the uneven distribution of different climates across the planet.

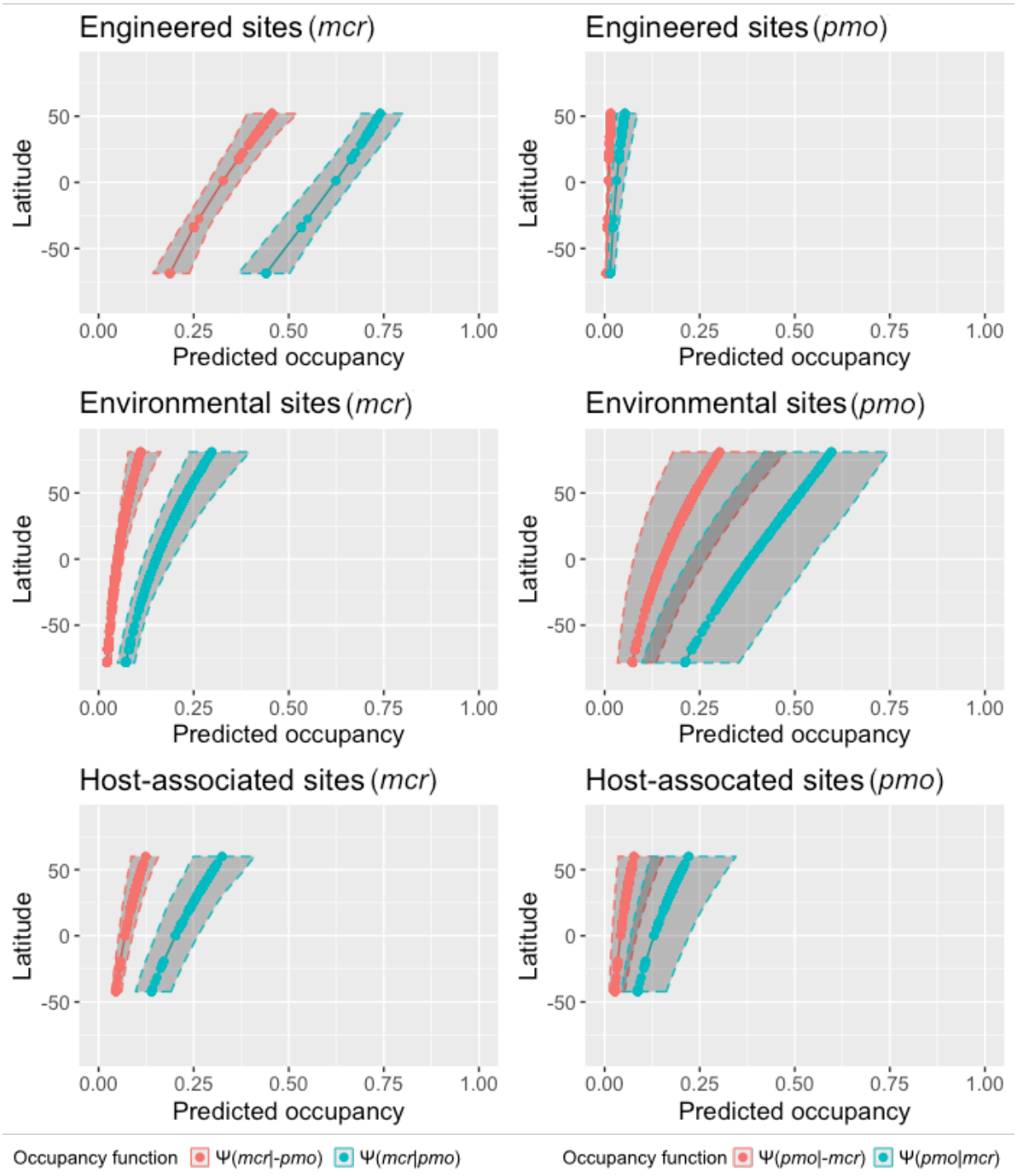

Figure 3.10: Predicted occupancy for functions of interest given the presence or absence of the other function for each environment type. Darker grey windows bounded by dotted lines represent 95% confidence intervals for the estimated occupancies (solid lines). Sites are un-aggregated metagenomes.

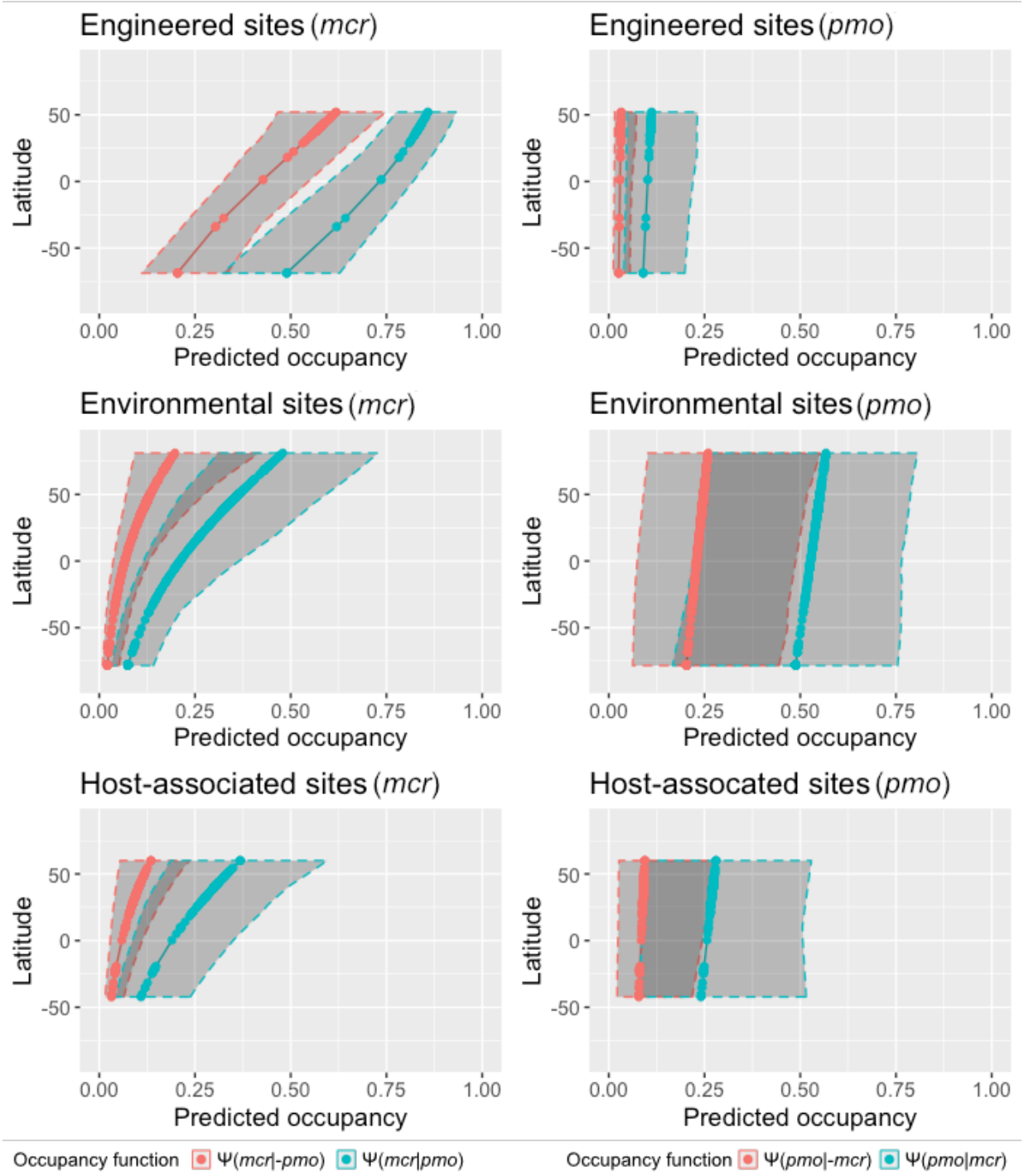

Figure 3.11: Predicted occupancy for functions of interest given the presence or absence of the other function for each environment type. Darker grey windows bounded by dotted lines represent 95% confidence intervals for the estimated occupancies (solid lines). Sites are aggregated on geocoordinates and ecosystem type.

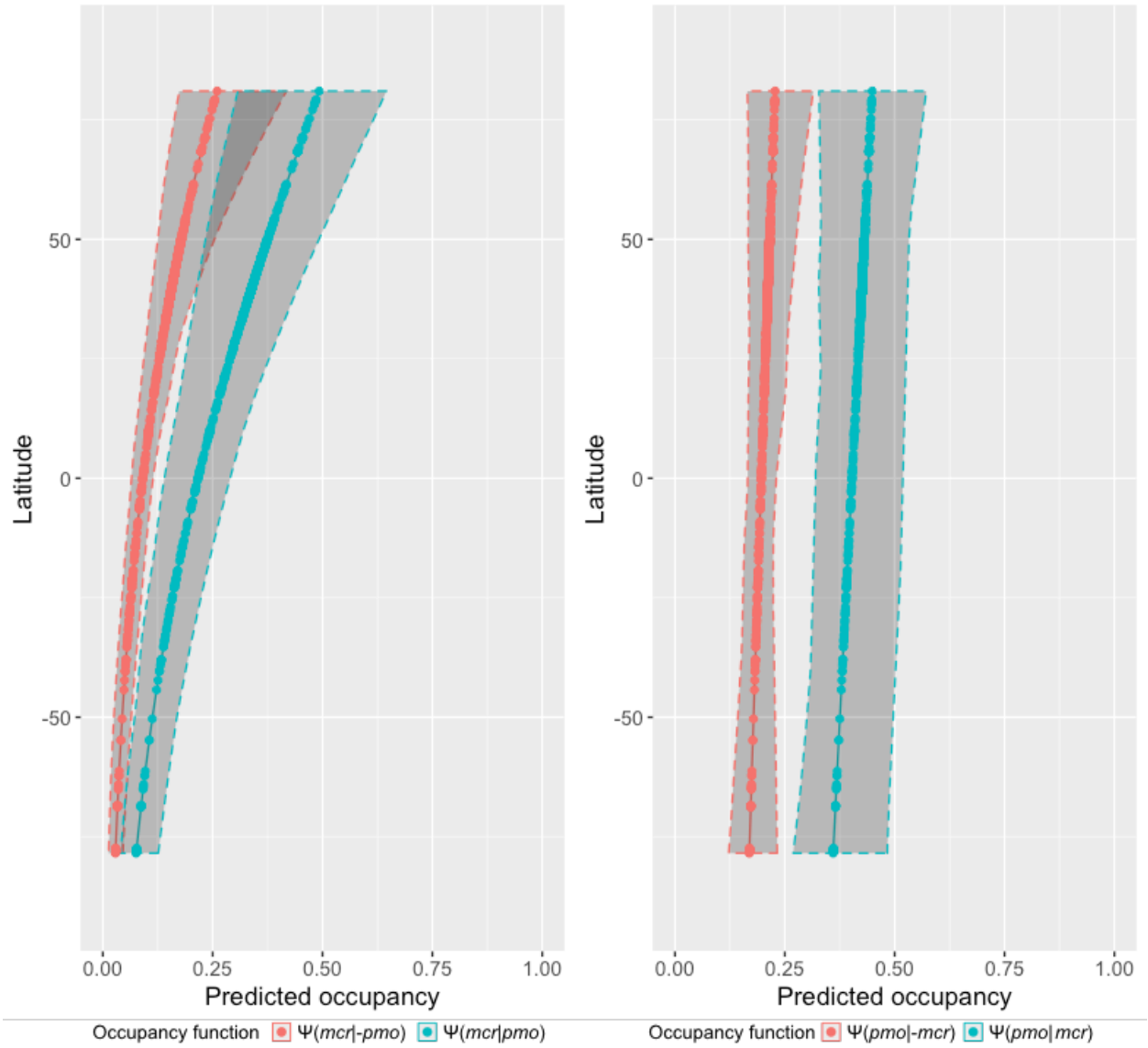

Figure 3.12: Estimated occupancy for functions of interest given the presence or absence of the other function. Darker grey windows bounded by dotted lines represent 95% confidence intervals for the estimated occupancies (solid lines). Sites were aggregated on geocoordinates only.

3.4 Discussion

3.4.1 Analysis of co-occurrence from the 6K dataset

The original goal of this research was to assess whether methanogens and methanotrophs exhibited co-occurrence across global environments. Many methanotrophs are obligate C1 users, and so there is an expected advantage to occurring in proximity to a C1 source such as a methanogen. However, the

majority of methanogens are strictly anaerobic (there may be an exception in the species *Methanothrix paradoxum*; Angle *et al.*, 2017), while most methanotrophs are aerobic (Hanson and Hanson, 1996). As such, methanotrophs and methanogens should not occur together based on environmental limitations. In an initial trial to assess co-occurrence of these two functions, we searched through 5,868 metagenomes hosted on IMG/M. Using the marker genes *mcrA* and *pmoB* as proxies for the metabolic potential for methanogenesis and methanotrophy, respectively, environments whose microbial communities contain the genetic potential for methane cycling could be identified. The resultant dataset was analyzed using the Veech co-occurrence model (2013). This is a relatively simple model which considers the number of sites at which some gene occurs, determines how often the two genes should occur together by random chance, based on the frequency at which they each occur, and then compares this number to how often the two genes were observed together. For this initial dataset, the model predicted that the two genes were positively correlated (*i.e.,* they occur together more often than would be expected by random chance). This initial result was interesting, and led to a search for better models that could be applied to this question. The Veech co-occurrence model only allows for a single observation for each site and gene, so there is no way to account for missed detections (false negatives). We wanted to leverage the depth of information available from metagenomes, which led to our adaptation of macroecology's occupancy modelling to microbial metagenomic data.

3.4.2 Occupancy modelling and its assumptions

Perfect detection is unlikely to ever be achieved in field ecology. The need to appropriately account for false negatives in detection surveys motivated the development of the occupancy model, which is growing in popularity in macroecology. The occupancy model was originally developed by MacKenzie *et al*., and is based on the tenet that, by repeating samplings, detection probability (p) could be informed (2002). In macroecological surveys, this is accomplished by repeatedly visiting sites. In this way, a detection probability can be estimated, which in turn can be used to determine occupancy (Ψ), where occupancy is the proportion of sites thought to be physically occupied by the species of interest. Estimates for both p and Ψ parameters can be determined using maximum likelihood approaches. Our question was "how can this modelling be translated to microbiology?". There is no clear way to define a repeated sample in metagenomics. Sampling a site alters or destroys it, and the difference in scale between microorganisms and the amount of sample typically taken for DNA extractions means that it is challenging to determine an area that would constitute a single site. Metagenomes contain a large amount of information, which is generally not leveraged to assess questions with statistical rigour. Here, we adapt

occupancy modeling to metagenomic data, by using separate genes indicative of the same function as a way to emulate re-sampling. If multiple genes are necessary for some function of interest, then the function can be thought of as the "species", and searching independently for each of the genes within the metagenome can be thought of as sampling repeatedly. In this way, a detection probability for the function of interest could be estimated.

As with any statistical models, it is important to consider the underlying assumptions being made about the models used. MacKenzie *et al*. (2002) laid out five assumptions for occupancy modelling. These were:

i) The closure assumption, which states that there is no chance of the occupancy state changing between sampling occasions for the site, within the same season;

ii) The probability of occupancy is the same across all sites, or is otherwise modelled appropriately with covariates;

iii) The probability of detection is the same across all sites, or accounted for by covariates;

iv) The detection at each site is independent of detection at other sites; and

v) There are no false positives

It can be challenging to interpret the results of a model when the assumptions are violated, and so here we discuss each of these assumptions in the context of metagenomics. Several of the assumptions come "for free" given the nature of metagenomic datasets, while others likely pose problems, and future work may be required to address these potential violations. Assumption (i) certainly holds true for metagenomic datasets – all samples are taken at an instantaneous moment, since they all come from the same metagenome. The community cannot shift from one state of occupancy to another within the instantaneous window of metagenomic sampling. Assumption (iv) will generally hold true as well. The isolation and sequencing of DNA at one site will have no impact on the isolation and sequencing of DNA from another. Exceptions to this would include contamination, where the two samples have been partially mixed or the DNA extraction kit contributes confounding DNA. In general, however, it is fair to make the assumption that extractions are contaminant-free.

Assumptions (ii) and (iii) present more challenges. For (ii), it is unlikely that occupancy probability is the same across all sites (*e.g.,* probability of methanogenesis in deep sea sediments will differ from soils and waste water bioreactors). As the amount of collected data increases, however, estimates of occupancy

will likely converge to a true background occupancy probability (this can be thought of as coming closer and closer to sampling every site in existence, which would give the "true" occupancy). This background probability would match, roughly, the total proportion of global environments occupied in this case. This number is not particularly useful, beyond perhaps suggesting that a particular group is more common than another or vice versa. It would be much better if variation in occupancy probability could be appropriately modelled, since stronger predictions about specific environment(s) could be made. This is similar for assumption (iii). For (iii), the probability of detection is dependent on the community complexity at the site and the depth of sequencing conducted for the metagenome in question – these will also vary across sites. Consider, by way of example, one metagenome derived from drinking water and one from soil. It is unlikely the proportion of DNA isolated and sequenced would be the same in both cases. Furthermore, if studying multiple functions, differences in cell physiology may alter how well DNA is retrieved from relevant organisms.

The final assumption, that there are no false positives, is challenging and will not be the same for all surveys. Well-conserved functional markers with no known homologs will generally make this assumption more reasonable, but this is not often the case. The best way to work with this assumption is to rigorously analyze biomarker sequences obtained from metagenomes and apply strict thresholds of quality for inclusion in modeling datasets. Occupancy models were designed to handle false negatives, so it is best to aggressively avoid false positives. Here, we used four independent methods to filter sequences for inclusion in our datasets. First, sufficient homology to gain an automated KEGG annotation on IMG/M developed the initial, fairly permissive dataset. Next, we size filtered the data, to remove sequences that were much too long, or that were too short to have sufficient information to strongly support the annotation. DIAMOND BLASTp was used to match all sequences to a database consisting of many different protein genes, as well as a set of reference sequences. The final filtering step involved assessing phylogenetic congruence to identify potential false positives. It remains important to think about the possibility of false positives, but it is our hope that this level of curation made assumption (v) reasonable for our data.

3.4.3 Single-species occupancy models

3.4.3.1 Predicted detectability of genes of interest

When applying the date that metagenomes were uploaded as a covariate, the models predicted clear trends of increasing detectability over time (Figure 3.9). This was highly pronounced for the *pmo* gene set, but

the *mcr* gene set also underwent an increase in detectability. The curvature of the *mcr* detection over the date range, and the fact that the *pmo* curve began to approach a similar value of ~85% detection may suggest that detectability is nearing a plateau. This would further imply that current metagenomic sequencing depths and assembly techniques are capturing environmental metabolic potential in a near-complete fashion. This should not be conflated with assuming that the entire environmental community is being captured, however. It may mean that, given that a species was captured in a sample and DNA extracted, its genome will be fully represented in the metagenome, but there may still be low-abundance organisms or organisms resisting lysis which are not being captured at all.

Upload date was used as a covariate because it was one of the few complete variables available. Careful consideration of the upload date results suggests sampling bias plays an overly strong role in its effects. To ensure that there was no bias introduced due to the large date range implemented when using the default start date of 1970-01-01 (dates converted to days since this date), a second analysis was run using 2006-01-01 as the start date, but results were not markedly different (Appendix D; Figure S1), suggesting that the model was robust to this issue. In the case of *mcr*, the models were improved only by a negligible amount (on the basis of AIC). While the improvement afforded to the *pmo* models was better, the results indicate that strong sampling bias may be impacting the date-related results. Whether the start date was set to 1970 or 2006, a square root transform was required for both of these covariates to be included in the model. This indicates that interpretation of this covariate may be troublesome. To further emphasize this, the detectability of *pmo* was near zero for the earliest metagenomes (uploaded prior to 2008), which is unexpectedly low. This number may be the result of sampling biases against environments that harbour methanotrophs, which caused the models to estimate that detectability was much lower in these earlier environments than it actually was. While sample upload date may indeed be a predictor of detectability, it is very challenging to interpret and the biases may be too strong for robust conclusions to be drawn.

The ability to model detection probabilities is promising, but some caution is still advisable. It is important to consider how different variables could be impacting estimates. For example, an open question is how differences in detectability between each gene for a given function impact the overall detection estimates. Our data shows this is a significant concern, based on the observed differences in the numbers of sequences for our selected biomarkers after the various filtering steps (Table 3.5). For the most part, it would be expected that genes within a given function would be detected in roughly equal numbers, barring any possible gene duplications. However, both gene sets showed fairly divergent

numbers for individual genes. For the *mcr* set, *mcrG* had 2.51 times as many detections as *mcrA* and 1.60 times as many as *mcrB* (Table 3.5). For the *pmo* set, *pmoC* had 1.86 times as many detections as *pmoA* and 2.17 times as many as *pmoB* (Table 3.5). These numbers illustrate that detection of the marker genes differs, and the source, or sources, of these discrepancies may be different in each case. For example, *pmoC* is duplicated within some species' genomes, including *Methylococcus capsulatus* Bath (Stolyar *et al.*, 1999). While this may explain the higher detection of *pmoC* in our data, this gene duplication will impact detectability of *pmo* as a whole, and should be accounted for within the model. Future models may need to incorporate detection covariates at the level of the individual genes. These covariates could include average copy number (to address inconsistencies across "samplings", as seen for *pmoC*), and others as appropriate.

3.4.3.2 The impact of ecosystem category on occupancy

The covariate that had the most profound impact on the fit of the models was ecosystem type. Three possible values were used for ecosystem type here, from the broadest ecosystem categorization available from the metadata retrieved from IMG/M. We limited our analyses to this highest level in order to avoid inconsistencies between manual user entries, and to prevent over-parameterization of the models. We did attempt to run models with lower-levels, but found that the modelling software was producing errors, convergence of numbers, and unrealistic probabilities (*e.g.,* perfect detections in certain cases). The ecosystem covariate reduced the AIC by the largest amount for every model (including the multi-species models), but this is perhaps not surprising. The category types were quite broad, and it was likely that any background probability of occupancy would vary substantially between them. This assumption was borne out, as there are distinct observed changes in occupancy between the different environment types, seen most strongly for the *mcr* genes (Figure 3.7).

The actual differences in predicted occupancy probabilities for each category provide some interesting insight. First, it is noteworthy that methanogens are predicted to have near-equal occupancy for both host-associated and environmental sites, at about 10% occupied (Figure 3.7). It is well established that methanogens occur in the rumens of various animal hosts and so should have some expected occupancy in this category, but would not be expected in plant-associated or non-ruminant host environments. In contrast, engineered sites showed relatively high occupancy rates for *mcr* (just over 40% occupied). As discussed earlier, methanogens have been identified in a variety of engineered environments, such as landfills and bioreactors. Engineered systems are thought to be important contributors to methane emissions (*e.g.,* methanogens have been identified in landfills, rice paddies, and others.; Laloui-Carpentier

et al., 2006; Sakai *et al.*, 2007; Tang *et al.*, 2016), and so this result is consistent with what is expected based on our understanding of methane cycling. The *pmo* operon has a comparatively low occupancy for engineered environments, at less than 10%. In the environmental category, *pmo* was predicted to have a higher occupancy than *mcr* in both the aggregated and un-aggregated datasets (Figure 3.7). This may suggest that methanotrophs are more widespread than methanogens in natural environments. Some caution should be taken when interpreting these numbers, though, as *pmo* genes are closely related to the *amo* genes, so it is possible that there is a higher degree of over-detection for these genes despite our stringent filtering process. Finally, the host-associated systems showed similar numbers in the aggregated datasets. In the unaggregated data for host-associated systems, *mcr* was predicted to occupy a slightly larger portion of sites than *pmo*.

A final note for consideration is the fact that some of the user-defined categorizations could overlap, which may cause issues with interpretation of results. One of the better examples of this is metagenomes from the rhizosphere of plants, which could be categorized as either host-associated or as a soil metagenome under the environmental category. These issues will be challenging to parse out without more standardization in the way sample sites are categorized.

3.4.3.3 The impact of latitude on occupancy

The single species models for all three of the data sets showed an increase in occupancy with increasing latitude (Figure 3.8). There are various possible explanations for this trend. For example, differences in the climates of landmass in the Southern versus the Northern hemispheres may impact results. Another possible explanation relates to the geography. The ratio of land-to-sea increases as one moves north. This may be important if the two groups occur more frequently on land, although this explanation lacks any basis in known distributions of methanogens or methanotrophs. What is more likely is that biases in sampling are driving this observed trend. There are relatively few marine samples, and so there is a bias in sampling. This bias is further aggravated by the fact that the majority of sampled sites are in the Northern hemisphere. In addition to these biases, nearly all of the engineered sites sampled were in the Northern hemisphere. Given the large differences in occupancy between engineered and the other sites, this could be a significant source of bias, and it is possible that the models simply lack sufficient evidence to generate strong distinctions between occupancy states in this case. For example, large regions of the Pacific Ocean were not sampled, so the overall occupancy of the ocean may be quite different from the sites that were sampled. It is noteworthy that the confidence intervals around the latitude of 25ºN are narrower than across the rest of the curves (Figure 3.8). This was likely because of the bias toward

sampling of the United States and, Central and Western Europe at that latitude. In addition to these biases, other differences in the environments sampled and their latitudinal distribution are important to consider. For example, the freshwater locations relative to latitude, or the fact that most of Earth's deserts lie between 30ºS and 30ºN. Together, these biases could be heavily influencing the changes in occupancy over the gradient of latitude. Future efforts could work with subsets of environments in an attempt to remove such biases. It is plausible that the occupancy numbers here represent a good estimate of the overall occupancy of these two functions, which would suggest that as many as 20% of randomly sampled environments in the United States and Central and Western Europe would be expected to contain *pmo* and as many as 15% to contain *mcr*.

3.4.3.4 The impact of aggregation on occupancy

There were relatively few differences in the trends observed between the aggregated data and the unaggregated data, but there are several points that are worth noting. First, overall occupancy increased in the aggregated data sets. This matches our predictions: when aggregating data, a site which was not occupied may be combined with one that was, and the aggregated site would be occupied. The tendency would be for an increase in the proportion of occupied sites. The other notable change was in the confidence intervals for occupancy estimates. For the most part, these became larger in the aggregated data, likely a result of having fewer data points. The unaggregated models consisted of 27,720 data points for each function, while the aggregated data sets by geocoordinate+environment and by geocoordinate only consisted of 3,687 and 3,606 data points, respectively. The differences in data richness are likely driving the changes in the confidence estimates.

3.4.4 Multi-species occupancy models

Multispecies occupancy models are powerful in that they can estimate the impact that species have on the occupancy of one another, and this impact can be modelled against the impact of different covariates. Here, we used various parameterization for the two functions, applied to all three versions of the dataset. Many of the trends observed in the single-species occupancy models were also observed in the multispecies models. The single-species models tend to be simpler and more straightforward to interpret. For this reason, the focus for this section will be on how the two species interacted, rather than on the trends that the various covariates exhibited.

3.4.4.1 The impact of latitude on occupancy

While the trends here generally matched the single-species models, the aggregated data sets, in both cases, showed relatively little change in *pmo* occupancy across the range of latitudes. The difference here is interesting. This trend held true whether or not *mcr* was present, contrary to our observations on the two functions' individual occupancies (see below). These conflicting results indicate further work is required to assess the validity of observed trends and of applied covariates.

3.4.4.2 The impact of one function's presence/absence on the other

The presence or absence of one function impacted occupancy of the other in all cases. For the un-aggregated data, the occupancy in *mcr* increased if *pmo* was present for the entire range of latitudes, in all three environment types (Figure 3.10). According to the models, this prediction was most likely to be true for engineered sites. The confidence intervals for both environmental and host-associated sites were much broader, but the same trends were observed. The same trend was observed for *pmo*, which increased in occupancy if *mcr* was present. However, the occupancy of *pmo* in engineered sites was very low regardless of the occupancy of *mcr*.

The aggregated data sets were similar in the patterns that they displayed compared to the unaggregated data. However, the confidence intervals were much larger, especially for the occupancy of *pmo* at environmental sites (Figure 3.11 and Figure 3.12). The broad confidence intervals make our results challenging to interpret. It may be that the occupancy of *pmo* increased at these sites, but this cannot be stated with certainty. The data that were aggregated solely on geocoordinates more clearly showed the trend that occupancy of both functions increases when the other function is present (Figure 3.12).

The original hypothesis of this work was that methanotrophs would be more likely to occur at sites where methanogens either occurred, or occurred in close proximity to, given the differing oxygen requirements of these two groups. The opposite case, where a methanogen would be more likely to occur if a methanotroph was present, is not anchored in our understanding of the biology of these organisms. This makes our model results interesting, since both of these scenarios are predicted to be true. It may be that external variables controlling the presence or absence of each group are shared, and so the two functional groups coincide because of limitations to their distributions. It is challenging to draw strong conclusions without more covariates, particularly continuous ones, to lend better explanatory power to the models.

3.4.5 Potential for occupancy modelling and current short-comings

We successfully applied occupancy modeling to metagenomic datasets, which opens the door for deeper statistical treatment of metagenomic data. Whether or not these models produce meaningful results remains to be seen. Before the potential of occupancy modeling can be assessed, it will be necessary to test these models with datasets that have known trends. While our models predicted interesting co-occurrence patterns here, we were constrained by the lack of metadata available for the metagenomes of interest.

An ideal occupancy model would have a unique detection and occupancy parameter for every site, but this would come at the cost of predictive power, since there would be no way to know these parameters *a priori*. For this reason, occupancy and detection are assigned single parameters across all samples, as governed by the model assumptions. However, it is unlikely that detectability and occupancy are uniform across all sites around the world. To address changes in detectability and occupancy, covariates can be extremely powerful, particularly continuous covariates that can be used to assess how these values vary as some external parameter does. The issue with the metagenomic data used in this study is that numerical covariates were relatively sparse, and often were not complete for all metagenomes. This seriously restricted our ability to apply these covariates to strengthen the predictive models. It was surprising that the models showed strong trends for the available numerical covariates, particularly for the latitude of the sample location. As was discussed earlier, however, some of the trends may be driven by sampling biases. Having more covariates could conceivably help to de-conflate the underlying confounding factors that are driving these biases. For example, can the latitudinal trends observed be explained by some other covariate that has not been included in metadata entries (*e.g.*, increasing population densities with latitude, and commensurate increase in engineered environments)? With more data and more complete metadata, much stronger predictive power could come from these models.

I believe that the remedy to the current lack of covariates is to require better metadata deposition standards. This would not require new or standardized sampling protocols, which would be near-impossible to implement across environments. Instead, database administrators could make deposition of certain data mandatory to be uploaded alongside sequence datasets, to enable statistical analyses of database collections. Established and enforced metadata reporting requirements could greatly strengthen modelling practices and allow a further maturation of metagenomic analyses within microbial ecology. This will require defining specific data that must be collected alongside a metagenomic sample at the time of capture. The types of data to include, for example, could be pH, mineral content, oxygen content, and

moisture content, among others. Further to this, when categorizing environments in hierarchical classification structures like that of IMG/M, standard definitions need to be established and enforced. This would serve two purposes: first, it would reduce the number of categories employed, allowing use of categorical covariates in statistical models without over-parameterization, and secondly, it would increase confidence in conclusions, knowing that the categories to which the metagenomes were assigned were well defined without redundancies or overlaps.

Occupancy models represent a step in the direction towards better modelling of relationships between microbes from metagenomic data. Microbial ecology has a strong foundation on which to build or adapt statistical tools to assess microbial interactions on a global scale. As occupancy modeling is refined for metagenomic data, careful choices must be made in order to ensure that these models not only model the data against which they were built, but also have predictive power for new sites.

Chapter 4
Conclusions and future directions

Recent advances in sequencing technology, particularly the development of long-read sequencers, has opened new avenues for biological exploration. The throughput of sequencing platforms means that current databases house unprecedented amounts of data. These increases in data availability and sequencing techniques impact microbiological research, where metagenome sequencing allows in-depth profiling of uncultured organisms. With such exciting developments has come a need for new tools that are capable of processing this wealth of data at all levels, from the raw sequence information to higher abstractions, so that meaning can be derived from the growing databases of metagenomic sequences.

Long-read sequencers are powerful, low cost platforms that are making sequencing available to a broader audience. However, the error rates of long read platforms are relatively high, and so assembling the raw sequence read data which they produce cannot be readily accomplished with established assembly methods, such as de Bruijn graph assembly. As part of this thesis work, I developed a rapid aligner of reads as a component of a larger genome assembler (the other component is under development by Dr. Brendan McConkey). The goal of the alignment algorithm was to take ordered reads along with their approximate overlaps and generate a consensus sequence representative of the genome from which the reads originated. To accomplish this, I built a tool that uses the order of kmers within each set of reads to determine the best sequence of kmers, representative of the original genome. When applied to a sample data set of 10 reads, this algorithm was able to produce a consensus with 85.3% identity when compared to a reference sequence that was built from high coverage long- and short-read data using an established hybrid read assembler (Unicycler; Wick *et al.*, 2017). This test dataset demonstrates that the algorithm I developed can produce an assembly that is reasonably close to the established consensus sequence.

The ~15% error rate that was detected for the best performing parameters of the alignment algorithm is relatively high, and minimizing this should be the focus of future work. There are two aspects of this that would have to be considered: how well does the algorithm perform with different data sets, especially ones with higher coverage? And, how well can the errors be corrected algorithmically? The first of these questions will be answered by simply applying the algorithm to additional data sets and assessing performance. Of particular interest would be to repeat experiments with the current data set, but with added coverage. We predict that a higher depth of sequencing will improve the quality of the alignment, but this must be validated experimentally. The second question will need to be the target of future

development work. Computationally identifying misaligned regions of the reads (*i.e.,* due to indel errors along the read) could inform subsequent steps in the alignment process to improve the quality of the final consensus sequence. In addition, finding points where breaks are introduced into the consensus sequence would identify candidates for joining. More testing of the code base could also improve assemblies by reducing computational errors. While there do not seem to be any major programmatic errors causing mis-assembly, there is the potential that small errors are occurring, especially around edge cases, such as printing the first kmer of a contig.

The second half of the research presented in this thesis examines the growing need for analysis techniques to derive meaning from metagenomic datasets and databases. There is a growing toolkit available to microbial ecologists, but there has been relatively little focus on modifying or developing statistical methods to be tailored to microbial datasets. Here, we adapted occupancy modelling, borrowed from macroecology, to demonstrate its applicability to microbial ecology. Occupancy modelling requires re-sampling, which is rare in metagenomic data collection. We emulated re-sampling by searching for different genes necessary for the same function. We applied occupancy modelling to the global methane cycle. Specifically, we looked at the key enzymes involved in methanogenesis (Mcr) and methanotrophy (Pmo), and searched for three genes encoding subunits in each of these enzymes. From this, we were able to "re-sample" for a function of interest by looking for multiple biomarkers within a single metagenome. Re-sampling was used in the occupancy model to inform our detection probabilities. In turn, this allowed us to infer how methanogens and methanotrophs were distributed globally, and to assess their co-occurrence. This initial proof-of-concept demonstrated the possibility that methanogens and methanotrophs are detected differentially across different environment types (which was not particularly surprising), with a detection trend across latitudes – the reason for this latitudinal trend is not clear. Further, based on the models used here, it is unlikely that methanogens and methanotrophs co-occur in all environments, but there was some evidence of shared preferred environmental conditions.

The work presented in chapter 3 demonstrates that occupancy models can, in principle, be applied to microbial datasets, but there is much more that can be done with this approach. Future work should focus on better assessing the fit of the occupancy models and improving predictive power. The most needed future direction here is to determine goodness-of-fit statistics that are appropriate for this type of data. Further, a deeper examination of how the assumptions of occupancy modelling can be understood in a metagenomic system may allow more robust application of these statistics.

The second future direction is to improve the predictive power of the models. The specific models assessed here are unlikely to have strong predictive power, since they lacked the good covariates to support predictions, even if they did identify some trends. Increasing predictive power is not likely to be an easy task because of the general scarcity of metadata available for metagenomic databases at this time. To fully exploit occupancy models at a global scale, it will be important to collect detailed metadata for all metagenomic samples, and for that metadata to be publicly available. Metadata can be used to parameterize the occupancy models. I believe the field of microbial ecology requires standardization of data-collection protocols and metadata release. In the best case, this would mean standardizing *what* information is collected as well as *how* it is entered into databases. If a soil sample for a metagenome is reported, then what constitutes soil (versus sediment, for example) should be well defined. In addition, numerical, or continuous, metadata needs to be entered in a consistent manner, which would require standardizing options like entering ranges when precise values cannot be obtained. Finally, which data ought to be collected should be decided and enforced. There are many possibilities here, so I will only suggest a few parameters that may be of some use. One important value, which is well known to impact biological systems, would be to collect the pH for all samples. A second important variable is sample temperature. Both of these are reasonably easy to collect in the field. An additional set of more complex covariates that would be useful include non-biological content of the sample, such as the amount of carbon and nitrogen, the moisture content of the sample, and the salinity of the sample. I believe that the most important future direction for modelling systems in microbial ecology will be the standardization of data and metadata collection.

The next few years of biological research will undoubtedly reveal fascinating new things. As we move forward, it is important that we continue to develop the tools needed to properly interpret these developments. The first part of this work involved programming a rapid aligner to work as a component of a genome assembler, while the second part of the work adapted a macroecological model for understanding patterns that exist between different organisms on a global scale to metagenomic data. We then applied that model to methane cycling microorganisms, as a proof of principle, and, in doing so, identified key areas for data deposition that will need to be improved by microbial ecologists collectively. This research developed several tools that could prove useful for future work in both genomics and metagenomics.

Bibliography

Achaz, G. (2002) Origin and fate of repeats in bacteria. *Nucleic Acids Research* **30**: 2987–2994.

Adams, C.P. and Kron, S.J. (1997) Method for performing amplification of nucleic acid with two primers bound to a single solid support. Patent #US5641658A.

Adams, M.D. (2000) The genome sequence of Drosophila melanogaster. *Science* **287**: 2185–2195.

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. (1990) Basic local alignment search tool. *Journal of Molecular Biology* **215**: 403–410.

Ammeraal, L. (1996) Algorithms and data structures in C++, Chichester ; New York: Wiley.

Angle, J.C., Morin, T.H., Solden, L.M., Narrowe, A.B., Smith, G.J., Borton, M.A., et al. (2017) Methanogenesis in oxygenated soils is a substantial fraction of wetland methane emissions. *Nature Communications* **8**.

Ardui, S., Ameur, A., Vermeesch, J.R., and Hestand, M.S. (2018) Single molecule real-time (SMRT) sequencing comes of age: applications and utilities for medical diagnostics. *Nucleic Acids Research* **46**: 2159–2168.

Bachmann, P. (1894) Die analytische Zahlentheorie [Analytical number theory], Leipzig B.G. Teubner.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A.A., Dvorkin, M., Kulikov, A.S., et al. (2012) SPAdes: A New genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology* **19**: 455–477.

Bapteste, E., Brochier, C., and Boucher, Y. (2005) Higher-level classification of the Archaea: evolution of methanogenesis and methanogens. *Archaea* **1**: 353–363.

Bentley, D.R., Balasubramanian, S., Swerdlow, H.P., Smith, G.P., Milton, J., Brown, C.G., et al. (2008) Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* **456**: 53–59.

Berlin, K., Koren, S., Chin, C.-S., Drake, J.P., Landolin, J.M., and Phillippy, A.M. (2015) Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature Biotechnology* **33**: 623–630.

Borrel, G., Adam, P.S., and Gribaldo, S. (2016) Methanogenesis and the Wood–Ljungdahl pathway: an ancient, versatile, and fragile association. *Genome Biology and Evolution* **8**: 1706–1711.

Borrel, G., Adam, P.S., McKay, L.J., Chen, L.-X., Sierra-García, I.N., Sieber, C.M.K., et al. (2019) Wide diversity of methane and short-chain alkane metabolisms in uncultured archaea. *Nature Microbiology* **4**: 603–613.

Branton, D., Deamer, D.W., Marziali, A., Bayley, H., Benner, S.A., Butler, T., et al. (2008) The potential and challenges of nanopore sequencing. *Nature Biotechnology* **26**: 1146–1153.

Braslavsky, I., Hebert, B., Kartalov, E., and Quake, S.R. (2003) Sequence information can be obtained from single DNA molecules. *Proceedings of the National Academy of Sciences* **100**: 3960–3964.

Brown, C.T., Hug, L.A., Thomas, B.C., Sharon, I., Castelle, C.J., Singh, A., et al. (2015) Unusual biology across a group comprising more than 15% of domain Bacteria. *Nature* **523**: 208–211.

Brown, D. and Morgenstern, B. eds. (2014) Algorithms in Bioinformatics: 14th International Workshop, WABI 2014, Wroclaw, Poland, September 8-10, 2014. Proceedings, Berlin, Heidelberg: Springer Berlin Heidelberg.

de Bruijn, N.G. (1946) A combinatorial problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam* **49**: 758–764.

Buchfink, B., Xie, C., and Huson, D.H. (2015) Fast and sensitive protein alignment using DIAMOND. *Nature Methods* **12**: 59–60.

Calle, M.L. (2019) Statistical analysis of metagenomics data. *Genomics & Informatics* **17**: e6.

Castelle, C.J., Wrighton, K.C., Thomas, B.C., Hug, L.A., Brown, C.T., Wilkins, M.J., et al. (2015) Genomic expansion of domain Archaea highlights roles for organisms from new phyla in anaerobic carbon cycling. *Current Biology* **25**: 690–701.
Castro-Wallace, S.L., Chiu, C.Y., John, K.K., Stahl, S.E., Rubins, K.H., McIntyre, A.B.R., et al. (2017) Nanopore DNA sequencing and genome assembly on the International Space Station. *Scientific Reports* **7**.
Chaisson, M.J. and Tesler, G. (2012) Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC Bioinformatics* **13**.
Chen, K. and Pachter, L. (2005) Bioinformatics for whole-genome shotgun sequencing of microbial communities. *PLoS Computational Biology* **1**: e24.
Chen, Y., Crombie, A., Rahman, M.T., Dedysh, S.N., Liesack, W., Stott, M.B., et al. (2010) Complete genome sequence of the aerobic facultative methanotroph Methylocella silvestris BL2. *Journal of Bacteriology* **192**: 3840–3841.
Chen, Y.-C., Liu, T., Yu, C.-H., Chiang, T.-Y., and Hwang, C.-C. (2013) Effects of GC bias in Next-Generation-Sequencing data on de novo genome assembly. *PLoS ONE* **8**: e62856.
Compeau, P.E.C., Pevzner, P.A., and Tesler, G. (2011) How to apply de Bruijn graphs to genome assembly. *Nature Biotechnology* **29**: 987–991.
Costello, A.M., Auman, A.J., Macalady, J.L., Scow, K.M., and Lidstrom, M.E. (2002) Estimation of methanotroph abundance in a freshwater lake sediment. *Environ Microbiol* **4**: 443–450.
Cui, M., Ma, A., Qi, H., Zhuang, X., and Zhuang, G. (2015) Anaerobic oxidation of methane: an "active" microbial process. *MicrobiologyOpen* **4**: 1–11.
Culpepper, M.A. and Rosenzweig, A.C. (2012) Architecture and active site of particulate methane monooxygenase. *Critical Reviews in Biochemistry and Molecular Biology* **47**: 483–492.
Deamer, D., Akeson, M., and Branton, D. (2016) Three decades of nanopore sequencing. *Nature Biotechnology* **34**: 518–524.
Dedysh, S.N. and Dunfield, P.F. (2011) Facultative and Obligate Methanotrophs. In, *Methods in Enzymology*. Elsevier, pp. 31–44.
Dlugokencky, E.J. (2003) Atmospheric methane levels off: Temporary pause or a new steady-state? *Geophysical Research Letters* **30**.
Dlugokencky, E.J., Bruhwiler, L., White, J.W.C., Emmons, L.K., Novelli, P.C., Montzka, S.A., et al. (2009) Observational constraints on recent increases in the atmospheric CH_4 burden. *Geophysical Research Letters* **36**.
DNA Sequencing Costs: Data *National Human Genome Research Institute (www.genome.gov)*. Accessed 2019-07-30. https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data
Dumont, M.G. (2014) Primers: Functional Marker Genes for Methylotrophs and Methanotrophs. In, McGenity,T.J., Timmis,K.N., and Nogales,B. (eds), *Hydrocarbon and Lipid Microbiology Protocols*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 57–77.
Dunfield, P.F., Yuryev, A., Senin, P., Smirnova, A.V., Stott, M.B., Hou, S., et al. (2007) Methane oxidation by an extremely acidophilic bacterium of the phylum Verrucomicrobia. *Nature* **450**: 879–882.
Dziewit, L., Pyzik, A., Romaniuk, K., Sobczak, A., Szczesny, P., Lipinski, L., et al. (2015) Novel molecular markers for the detection of methanogens and phylogenetic analyses of methanogenic communities. *Frontiers in Microbiology* **6**.
Eddy, S.R. (1998) Profile hidden Markov models. *Bioinformatics* **14**: 755–763.
Edgar, R.C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* **32**: 1792–1797.
Edwards, T. and McBride, B.C. (1975) New method for the isolation and identification of methanogenic bacteria. *Appl Microbiol* **29**: 540–545.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., et al. (2009) Real-time DNA sequencing from single polymerase molecules. *Science* **323**: 133–138.
Epplen, C., Melmer, G., Siedlaczck, I., Schwaiger, F.W., Mäueler, W., and Epplen, J.T. (1993) On the essence of "meaningless" simple repetitive DNA in eukaryote genomes. *EXS* **67**: 29–45.
Ermler, U., Grabarse, W., Shima, S., Goubeaud, M., and Thauer, R.K. (1997) Crystal structure of methyl-coenzyme M reductase: the key enzyme of biological methane formation. *Science* **278**: 1457–1462.
Euler, L. (1741) Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae* **8**: 128–140.
Ferrarini, M., Moretto, M., Ward, J.A., Šurbanovski, N., Stevanović, V., Giongo, L., et al. (2013) An evaluation of the PacBio RS platform for sequencing and de novo assembly of a chloroplast genome. *BMC Genomics* **14**: 670.
Ferry, J.G. (1992) Methane from acetate. *Journal of Bacteriology* **174**: 5489–5495.
Fleischmann, R.D., Adams, M.D., White, O., Clayton, R.A., Kirkness, E.F., Kerlavage, A.R., et al. (1995) Whole-genome random sequencing and assembly of Haemophilus influenzae Rd. *Science* **269**: 496–512.
Fletcher, S.E.M. and Schaefer, H. (2019) Rising methane: A new climate challenge. *Science* **364**: 932–933.
Friedrich, M.W. (2005) Methyl-Coenzyme M Reductase Genes: Unique Functional Markers for Methanogenic and Anaerobic Methane-Oxidizing Archaea. In, *Methods in Enzymology*. Elsevier, pp. 428–442.
Gilbert, B., McDonald, I.R., Finch, R., Stafford, G.P., Nielsen, A.K., and Murrell, J.C. (2000) Molecular analysis of the pmo (particulate methane monooxygenase) operons from two type II methanotrophs. *Applied and Environmental Microbiology* **66**: 966–975.
Goffeau, A., Barrell, B.G., Bussey, H., Davis, R.W., Dujon, B., Feldmann, H., et al. (1996) Life with 6000 Genes. *Science* **274**: 546–567.
Grabarse, W., Mahlert, F., Duin, E.C., Goubeaud, M., Shima, S., Thauer, R.K., et al. (2001) On the mechanism of biological methane formation: structural evidence for conformational changes in methyl-coenzyme M reductase upon substrate binding. *Journal of Molecular Biology* **309**: 315–330.
Griffith, D.M., Veech, J.A., and Marsh, C.J. (2016) **cooccur** : Probabilistic species co-occurrence analysis in *R*. *Journal of Statistical Software* **69**.
Hallam, S.J. (2004) Reverse methanogenesis: Testing the hypothesis with environmental genomics. *Science* **305**: 1457–1462.
Hallam, S.J., Girguis, P.R., Preston, C.M., Richardson, P.M., and DeLong, E.F. (2003) Identification of methyl coenzyme M reductase A (mcrA) genes associated with methane-oxidizing archaea. *Applied and Environmental Microbiology* **69**: 5483–5491.
Handelsman, J., Rondon, M.R., Brady, S.F., Clardy, J., and Goodman, R.M. (1998) Molecular biological access to the chemistry of unknown soil microbes: a new frontier for natural products. *Chemistry & Biology* **5**: R245–R249.
Hanson, R.S. and Hanson, T.E. (1996) Methanotrophic bacteria. *Microbiol Rev* **60**: 439–471.
Henckel, T., Friedrich, M., and Conrad, R. (1999) Molecular analyses of the methane-oxidizing microbial community in rice field soil by targeting the genes of the 16S rRNA, particulate methane monooxygenase, and methanol dehydrogenase. *Appl Environ Microbiol* **65**: 1980–1990.
Heydari, M., Miclotte, G., Demeester, P., Van de Peer, Y., and Fostier, J. (2017) Evaluation of the impact of Illumina error correction tools on de novo genome assembly. *BMC Bioinformatics* **18**.
Hinrichs, K.-U., Hayes, J.M., Sylva, S.P., Brewer, P.G., and DeLong, E.F. (1999) Methane-consuming archaebacteria in marine sediments. *Nature* **398**: 802–805.

Hippe, H., Caspari, D., Fiebig, K., and Gottschalk, G. (1979) Utilization of trimethylamine and other N-methyl compounds for growth and methane formation by Methanosarcina barkeri. *Proceedings of the National Academy of Sciences* **76**: 494–498.
History of Illumina Sequencing and Solexa Technology *(www.illumina.com)*. Accessed 2019-07-30. https://www.illumina.com/science/technology/next-generation-sequencing/illumina-sequencing-history.html
Hofmann, K., Praeg, N., Mutschlechner, M., Wagner, A.O., and Illmer, P. (2016) Abundance and potential metabolic activity of methanogens in well-aerated forest and grassland soils of an alpine region. *FEMS Microbiology Ecology* **92**: fiv171.
Hug, L.A., Baker, B.J., Anantharaman, K., Brown, C.T., Probst, A.J., Castelle, C.J., et al. (2016) A new view of the tree of life. *Nature Microbiology* **1**.
Hugenholtz, P., Goebel, B.M., and Pace, N.R. (1998) Impact of culture-independent studies on the emerging phylogenetic view of bacterial diversity. *Journal of Bacteriology* **180**: 4765–4774.
Hyatt, D., Chen, G.-L., LoCascio, P.F., Land, M.L., Larimer, F.W., and Hauser, L.J. (2010) Prodigal: prokaryotic gene recognition and translation initiation site identification. *BMC Bioinformatics* **11**.
Iguchi, H., Yurimoto, H., and Sakai, Y. (2010) Soluble and particulate methane monooxygenase gene clusters of the type I methanotroph Methylovulum miyakonense HT12: sMMO and pMMO gene clusters of Methylovulum miyakonense HT12. *FEMS Microbiology Letters* **312**: 71–76.
Information technology -- ISO 7-bit coded character set for information interchange International Organization for Standardization. Report #ISO/IEC 646:1991.
Introduction to SMRT Sequencing *PacBio (www.pacb.com)*. Accessed 2019-07-30. https://www.pacb.com/videos/video-introduction-to-smrt-sequencing/
Islam, T., Jensen, S., Reigstad, L.J., Larsen, O., and Birkeland, N.-K. (2008) Methane oxidation at 55 C and pH 2 by a thermoacidophilic bacterium belonging to the Verrucomicrobia phylum. *Proceedings of the National Academy of Sciences* **105**: 300–304.
Jain, M., Koren, S., Miga, K.H., Quick, J., Rand, A.C., Sasani, T.A., et al. (2018) Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*.
Jansen, R., van Embden, J.D.A., Gaastra, W., and Schouls, L.M. (2002) Identification of a novel family of sequence repeats among prokaryotes. *OMICS: A Journal of Integrative Biology* **6**: 23–33.
Janssen, P.H. and Kirs, M. (2008) Structure of the archaeal community of the rumen. *Applied and Environmental Microbiology* **74**: 3619–3625.
Jones, W.J., Leigh, J.A., Mayer, F., Woese, C.R., and Wolfe, R.S. (1983) Methanococcus jannaschii sp. nov., an extremely thermophilic methanogen from a submarine hydrothermal vent. *Archives of Microbiology* **136**: 254–261.
Jones, W.J., Paynter, M.J.B., and Gupta, R. (1983) Characterization of Methanococcus maripaludis sp. nov., a new methanogen isolated from salt marsh sediment. *Archives of Microbiology* **135**: 91–97.
Kamath, G.M., Shomorony, I., Xia, F., Courtade, T.A., and Tse, D.N. (2017) HINGE: long-read assembly achieves optimal repeat resolution. *Genome Research* **27**: 747–756.
Kanehisa, M. (2000) KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Research* **28**: 27–30.
Kang, D.D., Froula, J., Egan, R., and Wang, Z. (2015) MetaBAT, an efficient tool for accurately reconstructing single genomes from complex microbial communities. *PeerJ* **3**: e1165.
Kearse, M., Moir, R., Wilson, A., Stones-Havas, S., Cheung, M., Sturrock, S., et al. (2012) Geneious Basic: An integrated and extendable desktop software platform for the organization and analysis of sequence data. *Bioinformatics* **28**: 1647–1649.
Kendall, M.M. and Boone, D.R. (2006) Cultivation of methanogens from shallow marine sediments at Hydrate Ridge, Oregon. *Archaea* **2**: 31–38.

Kendall, M.M., Wardlaw, G.D., Tang, C.F., Bonin, A.S., Liu, Y., and Valentine, D.L. (2007) Diversity of archaea in marine sediments from Skan Bay, Alaska, including cultivated methanogens, and description of Methanogenium boonei sp. nov. *Applied and Environmental Microbiology* **73**: 407–414.

Kent, W.J. (2001) Assembly of the working draft of the human genome with GigAssembler. *Genome Research* **11**: 1541–1548.

Khmelenina, V.N., Colin Murrell, J., Smith, T.J., and Trotsenko, Y.A. (2018) Physiology and Biochemistry of the Aerobic Methanotrophs. In, Rojo,F. (ed), *Aerobic Utilization of Hydrocarbons, Oils and Lipids*. Cham: Springer International Publishing, pp. 1–25.

King, G.M., Klug, M.J., and Lovley, D.R. (1983) Metabolism of acetate, methanol, and methylated amines in intertidal sediments of Lowes Cove, Maine. *Appl Environ Microbiol* **45**: 1848–1853.

Klib — a generic library in C. Accessed 2019-07-31. http://attractivechaos.github.io/klib/#About

Knief, C. (2015) Diversity and habitat preferences of cultivated and uncultivated aerobic methanotrophic bacteria evaluated based on pmoA as molecular marker. *Frontiers in Microbiology* **6**.

Koeuth, T., Versalovic, J., and Lupski, J.R. (1995) Differential subsequence conservation of interspersed repetitive Streptococcus pneumoniae BOX elements in diverse bacteria. *Genome Research* **5**: 408–418.

Koren, S. and Phillippy, A.M. (2015) One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. *Current Opinion in Microbiology* **23**: 110–120.

Koren, S., Schatz, M.C., Walenz, B.P., Martin, J., Howard, J.T., Ganapathy, G., et al. (2012) Hybrid error correction and de novo assembly of single-molecule sequencing reads. *Nature Biotechnology* **30**: 693–700.

Koren, S., Walenz, B.P., Berlin, K., Miller, J.R., Bergman, N.H., and Phillippy, A.M. (2017) Canu: scalable and accurate long-read assembly via adaptive *k* -mer weighting and repeat separation. *Genome Research* **27**: 722–736.

Korte, B. and Vygen, J. (2008) NP-Completeness. In, *Combinatorial Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 359–392.

Kotsyurbenko, O.R., Chin, K.-J., Glagolev, M.V., Stubner, S., Simankova, M.V., Nozhevnikova, A.N., and Conrad, R. (2004) Acetoclastic and hydrogenotrophic methane production and methanogenic populations in an acidic West-Siberian peat bog. *Environmental Microbiology* **6**: 1159–1173.

Krishnakumar, R., Sinha, A., Bird, S.W., Jayamohan, H., Edwards, H.S., Schoeniger, J.S., et al. (2018) Systematic and stochastic influences on the performance of the MinION nanopore sequencer across a range of nucleotide bias. *Scientific Reports* **8**.

Krüger, M., Frenzel, P., Kemnitz, D., and Conrad, R. (2005) Activity, structure and dynamics of the methanogenic archaeal community in a flooded Italian rice field. *FEMS Microbiology Ecology* **51**: 323–331.

Ladapo, J.A. and Barlaz, M.A. (1997) Isolation and characterization of refuse methanogens. *Journal of Applied Microbiology* **82**: 751–758.

Laloui-Carpentier, W., Li, T., Vigneron, V., Mazéas, L., and Bouchez, T. (2006) Methanogenic diversity and activity in municipal solid waste landfill leachates. *Antonie van Leeuwenhoek* **89**: 423–434.

Landau, E. (1909) Handbuch der Lehre von der Verteilung der Primzahlen, Leipzig B.G. Teubner.

Lashof, D.A. and Ahuja, D.R. (1990) Relative contributions of greenhouse gas emissions to global warming. *Nature* **344**: 529–531.

Laver, T., Harrison, J., O'Neill, P.A., Moore, K., Farbos, A., Paszkiewicz, K., and Studholme, D.J. (2015) Assessing the performance of the Oxford Nanopore Technologies MinION. *Biomolecular Detection and Quantification* **3**: 1–8.

Lee, S.J., McCormick, M.S., Lippard, S.J., and Cho, U.-S. (2013) Control of substrate access to the active site in methane monooxygenase. *Nature* **494**: 380–384.

Levene, M.J. (2003) Zero-mode waveguides for single-molecule analysis at high concentrations. *Science* **299**: 682–686.
Li, Z., Chen, Y., Mu, D., Yuan, J., Shi, Y., Zhang, H., et al. (2012) Comparison of the two major classes of assembly algorithms: overlap-layout-consensus and de-bruijn-graph. *Briefings in Functional Genomics* **11**: 25–37.
Lidstrom, M.E. (1988) Isolation and characterization of marine methanotrophs. *Antonie van Leeuwenhoek* **54**: 189–199.
Lieberman, R.L. and Rosenzweig, A.C. (2004) Biological methane oxidation: Regulation, biochemistry, and active site structure of particulate methane monooxygenase. *Critical Reviews in Biochemistry and Molecular Biology* **39**: 147–164.
Lin, Y., Yuan, J., Kolmogorov, M., Shen, M.W., Chaisson, M., and Pevzner, P.A. (2016) Assembly of long error-prone reads using de Bruijn graphs. *Proceedings of the National Academy of Sciences of the United States of America* **113**: E8396–E8405.
Lizik, W., Im, J., Semrau, J.D., and Barcelona, M.J. (2013) A field trial of nutrient stimulation of methanotrophs to reduce greenhouse gas emissions from landfill cover soils. *Journal of the Air & Waste Management Association* **63**: 300–309.
Lloyd, K.G., Lapham, L., and Teske, A. (2006) An anaerobic methane-oxidizing community of ANME-1b archaea in hypersaline Gulf of Mexico sediments. *Applied and Environmental Microbiology* **72**: 7218–7230.
Loulergue, L., Schilt, A., Spahni, R., Masson-Delmotte, V., Blunier, T., Lemieux, B., et al. (2008) Orbital and millennial-scale features of atmospheric CH4 over the past 800,000 years. *Nature* **453**: 383–386.
Lu, Y.Y., Chen, T., Fuhrman, J.A., and Sun, F. (2016) COCACOLA: binning metagenomic contigs using sequence COmposition, read CoverAge, CO-alignment and paired-end read LinkAge. *Bioinformatics* btw290.
Lueders, T., Chin, K.J., Conrad, R., and Friedrich, M. (2001) Molecular analyses of methyl-coenzyme M reductase alpha-subunit (mcrA) genes in rice field soil and enrichment cultures reveal the methanogenic phenotype of a novel archaeal lineage. *Environmental Microbiology* **3**: 194–204.
Lupski, J.R. and Weinstock, G.M. (1992) Short, interspersed repetitive DNA sequences in prokaryotic genomes. *Journal of Bacteriology* **174**: 4525–4529.
Luton, P.E., Wayne, J.M., Sharp, R.J., and Riley, P.W. (2002) The mcrA gene as an alternative to 16S rRNA in the phylogenetic analysis of methanogen populations in landfill. *Microbiology (Reading, Engl)* **148**: 3521–3530.
MacKenzie, D.I., Bailey, L.L., and Nichols, J.D. (2004) Investigating species co-occurrence patterns when species are detected imperfectly. *Journal of Animal Ecology* **73**: 546–555.
MacKenzie, D.I., Nichols, J.D., Lachman, G.B., Droege, S., Andrew Royle, J., and Langtimm, C.A. (2002) Estimating site occupancy rates when detection probabilities are less than one. *Ecology* **83**: 2248–2255.
Marchesi, J.R., Weightman, A.J., Cragg, B.A., Parkes, R.J., and Fry, J.C. (2001) Methanogen and bacterial diversity and distribution in deep gas hydrate sediments from the Cascadia Margin as revealed by 16S rRNA molecular analysis. *FEMS microbiology ecology* **34**: 221–228.
Margulies, M., Egholm, M., Altman, W.E., Attiya, S., Bader, J.S., Bemben, L.A., et al. (2005) Genome sequencing in microfabricated high-density picolitre reactors. *Nature* **437**: 376–380.
Marinier, E., Brown, D.G., and McConkey, B.J. (2015) Pollux: platform independent error correction of single and mixed genomes. *BMC Bioinformatics* **16**.
McDonald, I.R., Bodrossy, L., Chen, Y., and Murrell, J.C. (2008) Molecular ecology techniques for the study of aerobic methanotrophs. *Applied and Environmental Microbiology* **74**: 1305–1315.

Medvedev, P. (2018) Modeling biological problems in computer science: A case study in genome assembly. *Briefings in Bioinformatics* 11.

Mendler, K., Chen, H., Parks, D.H., Lobb, B., Hug, L.A., and Doxey, A.C. (2019) AnnoTree: visualization and exploration of a functionally annotated microbial tree of life. *Nucleic Acids Research* **47**: 4442–4448.

Michener, W.K. (2015) Ecological data sharing. *Ecological Informatics* **29**: 33–44.

Miller, J.R., Koren, S., and Sutton, G. (2010) Assembly algorithms for next-generation sequencing data. *Genomics* **95**: 315–327.

Mitra, R. (1999) In situ localized amplification and contact replication of many individual DNA molecules. *Nucleic Acids Research* **27**: 34e–334.

Mitra, R.D., Shendure, J., Olejnik, J., Edyta-Krzymanska-Olejnik, and Church, G.M. (2003) Fluorescent in situ sequencing on polymerase colonies. *Analytical Biochemistry* **320**: 55–65.

Mojica, F.J.M., Ferrer, C., Juez, G., and Rodríguez-Valera, F. (1995) Long stretches of short tandem repeats are present in the largest replicons of the Archaea Haloferax mediterranei and Haloferax volcanii and could be involved in replicon partitioning. *Molecular Microbiology* **17**: 85–93.

Murrell, J.C., Gilbert, B., and McDonald, I.R. (2000) Molecular biology and regulation of methane monooxygenase. *Archives of Microbiology* **173**: 325–332.

Myers Jr, E.W. (2016) A history of DNA sequence assembly. *it - Information Technology* **58**.

Nakamura, T., Hoaki, T., Hanada, S., Maruyama, A., Kamagata, Y., and Fuse, H. (2007) Soluble and particulate methane monooxygenase gene clusters in the marine methanotroph *Methylomicrobium* sp. strain NI. *FEMS Microbiology Letters* **277**: 157–164.

Namiki, T., Hachiya, T., Tanaka, H., and Sakakibara, Y. (2012) MetaVelvet: an extension of Velvet assembler to de novo metagenome assembly from short sequence reads. *Nucleic Acids Research* **40**: e155–e155.

National inventory report: greenhouse gas sources and sinks in Canada (2017) Canada: Environment Canada. Report #ISSN 2371-1329. Accessed 2019-07-16. http://publications.gc.ca/site/eng/9.816345/publication.html

Nisbet, E.G., Manning, M.R., Dlugokencky, E.J., Fisher, R.E., Lowry, D., Michel, S.E., et al. (2019) Very strong atmospheric methane growth in the 4 years 2014–2017: Implications for the Paris Agreement. *Global Biogeochemical Cycles* **33**: 318–342.

Nissen, J.N., Sønderby, C.K., Armenteros, J.J.A., Grønbech, C.H., Bjørn Nielsen, H., Petersen, T.N., et al. (2018) Binning microbial genomes using deep learning. *bioRxiv*.

Nyren, P., Pettersson, B., and Uhlen, M. (1993) Solid phase DNA minisequencing by an enzymatic luminometric inorganic pyrophosphate detection assay. *Analytical Biochemistry* **208**: 171–175.

O'Donnell, C.R., Wang, H., and Dunbar, W.B. (2013) Error analysis of idealized nanopore sequencing. *Electrophoresis* **34**: 2137–2144.

Op den Camp, H.J.M., Islam, T., Stott, M.B., Harhangi, H.R., Hynes, A., Schouten, S., et al. (2009) Environmental, genomic and taxonomic perspectives on methanotrophic *Verrucomicrobia*: Perspectives on methanotrophic *Verrucomicrobia*. *Environmental Microbiology Reports* **1**: 293–306.

Paegel, B.M., Emrich, C.A., Wedemayer, G.J., Scherer, J.R., and Mathies, R.A. (2002) High throughput DNA sequencing with a microfabricated 96-lane capillary array electrophoresis bioprocessor. *Proceedings of the National Academy of Sciences* **99**: 574–579.

Paris Agreement (2015) Paris, France: United Nations. Report #54113. Treaty.

Parks, D.H., Chuvochina, M., Waite, D.W., Rinke, C., Skarshewski, A., Chaumeil, P.-A., and Hugenholtz, P. (2018) A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life. *Nature Biotechnology* **36**: 996–1004.

Parks, D.H., Rinke, C., Chuvochina, M., Chaumeil, P.-A., Woodcroft, B.J., Evans, P.N., et al. (2017) Recovery of nearly 8,000 metagenome-assembled genomes substantially expands the tree of life. *Nature Microbiology* **2**: 1533–1542.
Payne, A., Holmes, N., Rakyan, V., and Loose, M. (2018) Whale watching with BulkVis: A graphical viewer for Oxford Nanopore bulk fast5 files. *bioRxiv*.
Peng, Y., Leung, H.C.M., Yiu, S.M., and Chin, F.Y.L. (2012) IDBA-UD: a de novo assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics* **28**: 1420–1428.
Pevzner, P.A., Tang, H., and Waterman, M.S. (2001) An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences* **98**: 9748–9753.
Pol, A., Heijmans, K., Harhangi, H.R., Tedesco, D., Jetten, M.S.M., and Op den Camp, H.J.M. (2007) Methanotrophy below pH 1 by a new Verrucomicrobia species. *Nature* **450**: 874–878.
Prakash, D., Wu, Y., Suh, S.-J., and Duin, E.C. (2014) Elucidating the process of activation of methyl-coenzyme M reductase. *Journal of Bacteriology* **196**: 2491–2498.
Prather, M.J. and Holmes, C.D. (2017) Overexplaining or underexplaining methane's role in climate change. *Proceedings of the National Academy of Sciences* **114**: 5324–5326.
Price, M.N., Dehal, P.S., and Arkin, A.P. (2010) FastTree 2 – Approximately maximum-likelihood trees for large alignments. *PLoS ONE* **5**: e9490.
Rang, F.J., Kloosterman, W.P., and de Ridder, J. (2018) From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. *Genome Biology* **19**.
Reeve, J.N., Nölling, J., Morgan, R.M., and Smith, D.R. (1997) Methanogenesis: genes, genomes, and who's on first? *Journal of Bacteriology* **179**: 5975–5986.
Rhoads, A. and Au, K.F. (2015) PacBio sequencing and its applications. *Genomics, Proteomics & Bioinformatics* **13**: 278–289.
Rigby, M., Montzka, S.A., Prinn, R.G., White, J.W.C., Young, D., O'Doherty, S., et al. (2017) Role of atmospheric oxidation in recent methane growth. *Proceedings of the National Academy of Sciences* **114**: 5373–5377.
Rigby, M., Prinn, R.G., Fraser, P.J., Simmonds, P.G., Langenfelds, R.L., Huang, J., et al. (2008) Renewed growth of atmospheric methane. *Geophysical Research Letters* **35**.
Ronaghi, M., Karamohamed, S., Pettersson, B., Uhlén, M., and Nyrén, P. (1996) Real-time DNA sequencing using detection of pyrophosphate release. *Analytical Biochemistry* **242**: 84–89.
Ross, M.G., Russ, C., Costello, M., Hollinger, A., Lennon, N.J., Hegarty, R., et al. (2013) Characterizing and measuring bias in sequence data. *Genome Biology* **14**: R51.
Ross, M.O. and Rosenzweig, A.C. (2017) A tale of two methane monooxygenases. *JBIC Journal of Biological Inorganic Chemistry* **22**: 307–319.
Rota, C.T., Ferreira, M.A.R., Kays, R.W., Forrester, T.D., Kalies, E.L., McShea, W.J., et al. (2016) A multispecies occupancy model for two or more interacting species. *Methods in Ecology and Evolution* **7**: 1164–1173.
Ruan, J. and Li, H. (2019) Fast and accurate long-read assembly with wtdbg2. *bioRxiv*.
Sakai, S., Imachi, H., Sekiguchi, Y., Ohashi, A., Harada, H., and Kamagata, Y. (2007) Isolation of key methanogens for global methane emission from rice paddy fields: a novel isolate affiliated with the Clone Cluster Rice Cluster I. *Applied and Environmental Microbiology* **73**: 4326–4331.
Sanger, F., Coulson, A.R., Hong, G.F., Hill, D.F., and Petersen, G.B. (1982) Nucleotide sequence of bacteriophage λ DNA. *Journal of Molecular Biology* **162**: 729–773.
Sanger, F., Nicklen, S., and Coulson, A.R. (1977) DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences* **74**: 5463–5467.
Schaefer, H., Fletcher, S.E.M., Veidt, C., Lassey, K.R., Brailsford, G.W., Bromley, T.M., et al. (2016) A 21st-century shift from fossil-fuel to biogenic methane emissions indicated by 13CH4. *Science* **352**: 80–84.

Schimel, J. (2004) Playing scales in the methane cycle: From microbial ecology to the globe. *Proceedings of the National Academy of Sciences* **101**: 12400–12401.
Sequencing Read Length: How to calculate NGS read length *(*www.*illumina.com)*. Accessed 2019-07-30. https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/read-length.html
Shendure, J. (2005) Accurate multiplex polony sequencing of an evolved bacterial genome. *Science* **309**: 1728–1732.
Shendure, J., Balasubramanian, S., Church, G.M., Gilbert, W., Rogers, J., Schloss, J.A., and Waterston, R.H. (2017) DNA sequencing at 40: past, present and future. *Nature* **550**: 345–353.
Shigematsu, T., Hanada, S., Eguchi, M., Kamagata, Y., Kanagawa, T., and Kurane, R. (1999) Soluble methane monooxygenase gene clusters from trichloroethylene-degrading Methylomonas sp. strains and detection of methanotrophs during in situ bioremediation. *Applied and Environmental Microbiology* **65**: 5198–5206.
Sieber, C.M.K., Probst, A.J., Sharrar, A., Thomas, B.C., Hess, M., Tringe, S.G., and Banfield, J.F. (2018) Recovery of genomes from metagenomes via a dereplication, aggregation and scoring strategy. *Nature Microbiology* **3**: 836–843.
Singh, A., Singh, R.S., Upadhyay, S.N., Joshi, C.G., Tripathi, A.K., and Dubey, S.K. (2012) Community structure of methanogenic archaea and methane production associated with compost-treated tropical rice-field soil. *FEMS Microbiology Ecology* **82**: 118–134.
Singh, R., Guzman, M.S., and Bose, A. (2017) Anaerobic Oxidation of Ethane, Propane, and Butane by Marine Microbes: A Mini Review. *Frontiers in Microbiology* **8**.
Smit, A.F. (1996) The origin of interspersed repeats in the human genome. *Current Opinion in Genetics & Development* **6**: 743–748.
Smith, L.M., Sanders, J.Z., Kaiser, R.J., Hughes, P., Dodd, C., Connell, C.R., et al. (1986) Fluorescence detection in automated DNA sequence analysis. *Nature* **321**: 674–679.
Söhngen, N.L. (1910) Sur le rôle du Méthane dans la vie organique. *Recueil des Travaux Chimiques des Pays-Bas et de la Belgique* **29**: 238–274.
Söhngen, N.L. (1906) Über Bakterien, welche Methan als Kohlenstoffnahrung und Energiequelle gebrauchen. *Zentrabl Bakteriol Parasitenk Infektionskr* **15**: 513–517.
Spahni, R. (2005) Atmospheric methane and nitrous oxide of the late Pleistocene from Antarctic ice cores. *Science* **310**: 1317–1321.
Spang, A., Saw, J.H., Jørgensen, S.L., Zaremba-Niedzwiedzka, K., Martijn, J., Lind, A.E., et al. (2015) Complex archaea that bridge the gap between prokaryotes and eukaryotes. *Nature* **521**: 173–179.
Staden, R. (1979) A strategy of DNA sequencing employing computer programs. *Nucleic Acids Research* **6**: 2601–2610.
Stephenson, M. and Stickland, L.H. (1933) Hydrogenase: The bacterial formation of methane by the reduction of one-carbon compounds by molecular hydrogen. *Biochemical Journal* **27**: 1517–1527.
Stewart, R.D., Auffret, M.D., Warr, A., Wiser, A.H., Press, M.O., Langford, K.W., et al. (2018) Assembly of 913 microbial genomes from metagenomic sequencing of the cow rumen. *Nature Communications* **9**.
Stolyar, S., Costello, A.M., Peeples, T.L., and Lidstrom, M.E. (1999) Role of multiple gene copies in particulate methane monooxygenase activity in the methane-oxidizing bacterium Methylococcus capsulatus Bath. *Microbiology* **145**: 1235–1244.
St-Pierre, B., Cersosimo, L.M., Ishaq, S.L., and Wright, A.-D.G. (2015) Toward the identification of methanogenic archaeal groups as targets of methane mitigation in livestock animals. *Frontiers in Microbiology* **6**.

Summons, R.E., Franzmann, P.D., and Nichols, P.D. (1998) Carbon isotopic fractionation associated with methylotrophic methanogenesis. *Organic Geochemistry* **28**: 465–475.
Suzek, B.E., Wang, Y., Huang, H., McGarvey, P.B., Wu, C.H., and the UniProt Consortium (2015) UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics* **31**: 926–932.
Svenning, M.M., Wartiainen, I., Hestnes, A.G., and Binnerup, S.J. (2003) Isolation of methane oxidising bacteria from soil by use of a soil substrate membrane system. *FEMS Microbiology Ecology* **44**: 347–354.
Tang, W., Wang, Y., Lei, Y., and Song, L. (2016) Methanogen communities in a municipal landfill complex in China. *FEMS Microbiology Letters* **363**: fnw075.
Thauer, R.K., Kaster, A.-K., Seedorf, H., Buckel, W., and Hedderich, R. (2008) Methanogenic archaea: ecologically relevant differences in energy conservation. *Nature Reviews Microbiology* **6**: 579–591.
The International Human Genome Mapping Consortium (2001) A physical map of the human genome. *Nature* **409**: 934–941.
Treangen, T.J. and Salzberg, S.L. (2012) Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature Reviews Genetics* **13**: 36–46.
Turing, A.M. (1937) On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* **s2-42**: 230–265.
Turner, A.J., Frankenberg, C., Wennberg, P.O., and Jacob, D.J. (2017) Ambiguity in the causes for decadal trends in atmospheric methane and hydroxyl. *Proceedings of the National Academy of Sciences* **114**: 5367–5372.
Tyler, A.D., Mataseje, L., Urfano, C.J., Schmidt, L., Antonation, K.S., Mulvey, M.R., and Corbett, C.R. (2018) Evaluation of Oxford Nanopore's MinION sequencing device for microbial whole genome sequencing applications. *Scientific Reports* **8**.
Tyson, G.W., Chapman, J., Hugenholtz, P., Allen, E.E., Ram, R.J., Richardson, P.M., et al. (2004) Community structure and metabolism through reconstruction of microbial genomes from the environment. *Nature* **428**: 37–43.
Tyson, J.R., O'Neil, N.J., Jain, M., Olsen, H.E., Hieter, P., and Snutch, T.P. (2018) MinION-based long-read sequencing and assembly extends the *Caenorhabditis elegans* reference genome. *Genome Research* **28**: 266–274.
Uchiyama, T., Ito, K., Mori, K., Tsurumaru, H., and Harayama, S. (2010) Iron-corroding methanogen isolated from a crude-oil storage tank. *Applied and Environmental Microbiology* **76**: 1783–1788.
Usdin, K. (2008) The biological effects of simple tandem repeats: Lessons from the repeat expansion diseases. *Genome Research* **18**: 1011–1019.
Vanwonterghem, I., Evans, P.N., Parks, D.H., Jensen, P.D., Woodcroft, B.J., Hugenholtz, P., and Tyson, G.W. (2016) Methylotrophic methanogenesis discovered in the archaeal phylum Verstraetearchaeota. *Nature Microbiology* **1**.
Veech, J.A. (2013) A probabilistic model for analysing species co-occurrence: Probabilistic model. *Global Ecology and Biogeography* **22**: 252–260.
Venter, J.C., Adams, M.D., Myers, E.W., Li, P.W., Mural, R.J., Sutton, G.G., et al. (2001) The Sequence of the Human Genome. *Science* **291**: 1304–1351.
Ver Eecke, H.C., Butterfield, D.A., Huber, J.A., Lilley, M.D., Olson, E.J., Roe, K.K., et al. (2012) Hydrogen-limited growth of hyperthermophilic methanogens at deep-sea hydrothermal vents. *Proceedings of the National Academy of Sciences* **109**: 13674–13679.
Victoria, X., Blades, N., Ding, J., Sultana, R., and Parmigiani, G. (2012) Estimation of sequencing error rates in short reads. *BMC Bioinformatics* **13**: 185.

Volta, A. (1777) Lettere dell'lllustrissimo Signor Volta Alessandro sull'aria inflammabile native dele paludi. In, *Giuseppe Marelli*. Milan, Italy.
Vorobev, A.V., Baani, M., Doronina, N.V., Brady, A.L., Liesack, W., Dunfield, P.F., and Dedysh, S.N. (2011) Methyloferula stellata gen. nov., sp. nov., an acidophilic, obligately methanotrophic bacterium that possesses only a soluble methane monooxygenase. *International Journal of Systematic and Evolutionary Microbiology* **61**: 2456–2463.
Wang, F.-P., Zhang, Y., Chen, Y., He, Y., Qi, J., Hinrichs, K.-U., et al. (2014) Methanotrophic archaea possessing diverging methane-oxidizing and electron-transporting pathways. *ISME J* **8**: 1069–1078.
Wang, Z., Elekwachi, C.O., Jiao, J., Wang, M., Tang, S., Zhou, C., et al. (2017) Investigation and manipulation of metabolically active methanogen community composition during rumen development in black goats. *Scientific Reports* **7**.
Ward, T.E. and Frea, J.I. (1980) Sediment distribution of methanogenic bacteria in lake Erie and Cleveland harbor. *Appl Environ Microbiol* **39**: 597–603.
Whiticar, M.J., Faber, E., and Schoell, M. (1986) Biogenic methane formation in marine and freshwater environments: CO2 reduction vs. acetate fermentation—Isotope evidence. *Geochimica et Cosmochimica Acta* **50**: 693–709.
Wick, R.R., Judd, L.M., Gorrie, C.L., and Holt, K.E. (2017) Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads. *PLOS Computational Biology* **13**: e1005595.
Wick, R.R., Judd, L.M., and Holt, K.E. (2019) Performance of neural network basecalling tools for Oxford Nanopore sequencing. *Genome Biology* **20**.
Wise, M.G., McArthur, J.V., and Shimkets, L.J. (1999) Methanotroph diversity in landfill soil: isolation of novel type I and type II methanotrophs whose presence was suggested by culture-independent 16S ribosomal DNA analysis. *Applied and Environmental Microbiology* **65**: 4887–4897.
Wongnate, T. and Ragsdale, S.W. (2015) The reaction mechanism of methyl-coenzyme M reductase: How an enzyme enforces strict binding order. *Journal of Biological Chemistry* **290**: 9322–9334.
Wu, S.-Y. and Lai, M.-C. (2011) Methanogenic archaea isolated from Taiwan's Chelungpu fault. *Applied and Environmental Microbiology* **77**: 830–838.
Wu, Y.-W., Tang, Y.-H., Tringe, S.G., Simmons, B.A., and Singer, S.W. (2014) MaxBin: an automated binning method to recover individual genomes from metagenomes using an expectation-maximization algorithm. *Microbiome* **2**.
Xu, X., Yuan, F., Hanson, P.J., Wullschleger, S.D., Thornton, P.E., Riley, W.J., et al. (2016) Reviews and syntheses: Four decades of modeling methane cycling in terrestrial ecosystems. *Biogeosciences* **13**: 3735–3755.
Yadav, S., Kundu, S., Ghosh, S.K., and Maitra, S.S. (2015) Molecular analysis of methanogen richness in landfill and marshland targeting 16S rDNA sequences. *Archaea* **2015**: 1–9.
Zheng, Y., Harris, D.F., Yu, Z., Fu, Y., Poudel, S., Ledbetter, R.N., et al. (2018) A pathway for biological methane production using bacterial iron-only nitrogenase. *Nature Microbiology* **3**: 281–286.
Zhou, K., Aertsen, A., and Michiels, C.W. (2014) The role of variable DNA tandem repeats in bacterial adaptation. *FEMS Microbiology Reviews* **38**: 119–141.
Zhou, M., Hernandez-Sanabria, E., and Guan, L.L. (2009) Assessment of the microbial ecology of ruminal methanogens in cattle with different feed efficiencies. *Applied and Environmental Microbiology* **75**: 6524–6533.

Appendix A

References for the 6K dataset

Species	Accession Number	Gene set
Candidatus_Bathyarchaeota_archaeon_BA1	KPV65186.1	*mcrA*
Candidatus_Bathyarchaeota_archaeon_BA2	KPV61791.1	*mcrA*
Methanomassiliicoccus_luminyensis	WP_019176774.1	*mcrA*
Verstraetearchaeota_V4	2701347163_Ga0138511_17310	*mcrA*
Verstraetearchaeota_V3	2701349801_Ga0138509_16830	*mcrA*
Verstraetearchaeota_V2	2701353610_Ga0138507_108103	*mcrA*
Verstraetearchaeota_V1	2701352036_Ga0138500_13090	*mcrA*
ANME-2_cluster_archaeon_HR1	PPA79495.1	*mcrA*
Methanosaeta_pelagica	BAL63106.1	*mcrA*
Methanosaeta_harundinacea_6Ac	AET63880.1	*mcrA*
Methanothrix_soehngenii_GP6	AEB67565.1	*mcrA*
Methanothrix_thermoacetophila_PT	ABK14360.1	*mcrA*
Methermicoccus_shengliensis	WP_084174107.1	*mcrA*
Methanosarcina_lacustris_Z-7289	AKB73417.1	*mcrA*
Methanosarcina_thermophila_TM-1	AKB13402.1	*mcrA*
Methanosarcina_flavescens	KPL45056.1	*mcrA*
Methanosarcina_vacuolata_Z-761	AKB45682.1	*mcrA*
Methanosarcina_barkeri_CM1	AKJ39604.1	*mcrA*
Methanosarcina_horonobensis_HB-1_=_JCM_15518	AKB76567.1	*mcrA*
Methanosarcina_soligelidi	WP_048050667.1	*mcrA*
Methanosarcina_mazei_S-6	AKB63269.1	*mcrA*
Methanosarcina_acetivorans_C2A	AAM07885.1	*mcrA*
Methanosarcina_siciliae_C2J	AKB38845.1	*mcrA*
Methanohalobium_evestigatum_Z-7303	ADI73798.1	*mcrA*
Methanohalophilus_euhalobius	SNY14756.1	*mcrA*
Methanohalophilus_mahii_DSM_5219	ADE36137.1	*mcrA*
Methanohalophilus_portucalensis_FDF-1	SMH31307.1	*mcrA*
Methanohalophilus_halophilus	BAL72744.1	*mcrA*
Methanosalsum_zhilinae_DSM_4017	AEH60719.1	*mcrA*
Methanolobus_vulcani	SDF34414.1	*mcrA*
Methanolobus_profundi	SFM58348.1	*mcrA*
Methanolobus_tindarius_DSM_2278	ETA68796.1	*mcrA*

Methanolobus_psychrophilus_R15	AFV23037.1	*mcrA*
Methanomethylovorans_hollandica_DSM_15978	AGB49863.1	*mcrA*
Methanococcoides_methylutens	WP_048204805.1	*mcrA*
Methanococcoides_vulcani	SET04961.1	*mcrA*
Methanococcoides_alaskense	BAL72743.1	*mcrA*
Methanococcoides_burtonii_DSM_6242	ABE53268.1	*mcrA*
Methanocella_arvoryzae_MRE50	CAJ37204.1	*mcrA*
Methanocella_conradii_HZ254	AFC99668.1	*mcrA*
Methanocella_paludicola_SANAE	BAI60588.1	*mcrA*
Methanoregula_boonei_6A8	ABS55100.1	*mcrA*
Methanoregula_formicica_SMSP	AGB01930.1	*mcrA*
Methanolinea_tarda	WP_007314361.1	*mcrA*
Methanosphaerula_palustris_E1-9c	ACL17594.1	*mcrA*
Methanocorpusculum_labreanum_Z	ABN07725.1	*mcrA*
Methanocorpusculum_parvum	PAV09008.1	*mcrA*
Methanocorpusculum_bavaricum	WP_042699351.1	*mcrA*
Methanomicrobium_mobile	WP_042705994.1	*mcrA*
Methanoplanus_limicola_DSM_2279	EHQ36916.1	*mcrA*
Methanogenium_cariaci	WP_062396498.1	*mcrA*
Methanolacinia_petrolearia_DSM_11571	ADN37161.1	*mcrA*
Methanolacinia_paynteri	WP_048153025.1	*mcrA*
Methanofollis_ethanolicus	BAL72755.1	*mcrA*
Methanofollis_liminatans_DSM_4140	EJG07654.1	*mcrA*
Methanoculleus_bourgensis_MS2	CCJ36661.1	*mcrA*
Methanoculleus_chikugoensis	BAL72746.1	*mcrA*
Methanoculleus_marisnigri_JR1	ABN56546.1	*mcrA*
Methanoculleus_sediminis	WP_048180309.1	*mcrA*
Methanospirillum_hungatei_JF-1	ABD41854.1	*mcrA*
Methanospirillum_stamsii	PWR73637.1	*mcrA*
Methanospirillum_psychrodurum	AGT97393.1	*mcrA*
Methanospirillum_lacunae	PWR69748.1	*mcrA*
Methanobrevibacter_ruminantium_M1	ADC47774.1	*mcrA*
Methanobrevibacter_millerae	ALT69783.1	*mcrA*
Methanobrevibacter_smithii_DSM_2374	EFC93211.1	*mcrA*
Methanobacterium_congolense	SCG86533.1	*mcrA*
Methanobacterium_uliginosum	BAI67104.1	*mcrA*
Methanobacterium_oryzae	BAI67101.1	*mcrA*
Methanobacterium_aarhusense	AAR27839.1	*mcrA*

Methanobacterium_palustre	BAI67102.1	*mcrA*
Methanobacterium_formicicum	AIS31752.1	*mcrA*
Methanobacterium_petrolearium	BAI67093.1	*mcrA*
Methanobacterium_ferruginis	BAI67094.1	*mcrA*
Methanobacterium_movens	ADM52195.1	*mcrA*
Methanobacterium_alcaliphilum	BAN67656.1	*mcrA*
Methanobacterium_flexile	ADM52196.1	*mcrA*
Methanothermobacter_marburgensis_str._Marburg	ADL59127.1	*mcrA*
Methanothermobacter_wolfeii	SCM58307.1	*mcrA*
Methanobrevibacter_wolinii	WP_042708239.1	*mcrA*
Methanobrevibacter_arboriphilus_JCM_13429_=_DSM_1125	OQD59595.1	*mcrA*
Methanopyrus_kandleri_AV19	AAM01870.1	*mcrA*
Methanosphaera_stadtmanae_DSM_3091	CAE48306.1	*mcrA*
Methanosphaera_cuniculi	PAV07777.1	*mcrA*
Methanobrevibacter_oralis	WP_042694806.1	*mcrA*
Methanobrevibacter_olleyae	AMK15668.1	*mcrA*
Methanococcus_voltae_A3	ADI36785.1	*mcrA*
Methanococcus_maripaludis_S2	CAF31115.1	*mcrA*
Methanococcus_vannielii	AAA72598.1	*mcrA*
Methanococcus_aeolicus	WP_011973976.1	*mcrA*
Methanothermococcus_thermolithotrophicus	WP_018153522.1	*mcrA*
Methanothermococcus_okinawensis_IH1	AEH06926.1	*mcrA*
Methanotorris_formicicus_Mc-S-70	EHP88088.1	*mcrA*
Methanocaldococcus_villosus	WP_004590263.1	*mcrA*
Methanocaldococcus_infernus	WP_013099697.1	*mcrA*
Methanocaldococcus_vulcanius_M7	ACX72017.1	*mcrA*
Methanocaldococcus_fervens_AG86	ACV24756.1	*mcrA*
Methanocaldococcus_jannaschii_DSM_2661	AAB98851.1	*mcrA*
Methanobacterium_subterraneum	AUB60672.1	*mcrA*
Methanobacterium_lacus	ADZ10495.1	*mcrA*
Methanobacterium_ivanovii	BAI67108.1	*mcrA*
Methanobacterium_espanolae	BAI67106.1	*mcrA*
Methanobacterium_bryantii	PAV05908.1	*mcrA*
Methanotorris_igneus_Kol_5	AEF96001.1	*mcrA*
Methanothermus_fervidus_DSM_2088	ADP77533.1	*mcrA*
Methanothermobacter_thermautotrophicus_str._Delta_H	AAB85618.1	*mcrA*
Methylacidiphilum_fumariolicum	WP_009060833.1	*pmoB*

Methylacidiphilum_infernorum	WP_012463842.1	*pmoB*
Methylocystis_rosea	WP_018409560.1	*pmoB*
Methylobacter_marinus	WP_020159527.1	*pmoB*
Methylohalobius_crimeensis	WP_022947316.1	*pmoB*
Methylomarinovum_caldicuralii	B9ZY20	*pmoB*
Methyloglobulus_morosus_KoM1	V5C0P9	*pmoB*
Crenothrix_polyspora	WP_087144449.1	*pmoB*
Methylosarcina_fibrata	WP_020564882.1	*pmoB*
Methylomicrobium_agile	WP_005374465.1	*pmoB*
Methylosarcina_lacus	WP_024298804.1	*pmoB*
Methylomarinum_vadi	WP_031433835.1	*pmoB*
Methylomonas_methanica	WP_013817027.1	*pmoB*
Methylomonas_koyamae	WP_064041205.1	*pmoB*
Methylomicrobium_buryatense	WP_017841994.1	*pmoB*
Methylomicrobium_japanense	BAE86886.1	*pmoB*
Methylomicrobium_alcaliphilum	WP_014147022.1	*pmoB*
Methylovulum_miyakonense	WP_019865089.1	*pmoB*
Methylovulum_psychrotolerans	A0A1Z4C2E5	*pmoB*
Methylobacter_tundripaludum	WP_006891798.1	*pmoB*
Methylobacter_luteus	WP_027159171.1	*pmoB*
Methylobacter_whittenburyi	WP_036297856.1	*pmoB*
Methylogaea_oryzae	WP_054774742.1	*pmoB*
Methylococcus_capsulatus	WP_010961049.1	*pmoB*
Methylomagnum_ishizawai	A0A1Y6D3S8	*pmoB*
Methylocaldum_szegediense	WP_026609852.1	*pmoB*
Methylocaldum_marinum	A0A250KPZ1	*pmoB*
Methylocapsa_acidiphila	CAJ01618.1	*pmoB*
Methylocapsa_aurea	WP_051953405.1	*pmoB*
Methylosinus_trichosporium_OB3b	AAF37894.1	*pmoB*
Methylosinus_sporium	WP_108917566.1	*pmoB*
Methylocystis_parvus	WP_016921577.1	*pmoB*

Appendix B

Repository of online data

Much of the data used for this thesis was too large to include here, so has been made available at the following link:

http://bit.ly/2KxOyu2

Data included are:

- Reference sequence accession numbers for the 10K dataset
- R code used for the occupancy analysis

Appendix C

R code used for model building

```
library(cooccur)
library(unmarked)
library(dplyr)
library(ggplot2)
library(ggmap)
library(maptools)
library(maps)
library(gridExtra)

set.seed(1237)

# history.df <- detection histories
# siteCovs.df <- covariates for all sites
# detCov.df <- covariates and detection histories

detCov.df <- detCov.df[order(detCov.df$Latitude),]

# Make the data frame for data aggregated by geocoord
agg.df <- detCov.df %>% group_by(Longitude, Latitude, Ecosystem) %>%
  summarize(mcrA=max(mcrA), mcrB=max(mcrB), mcrG=max(mcrG),
            pmoA=max(pmoA), pmoB=max(pmoB), pmoC=max(pmoC),

Ecosystem.Category=unique(list(as.character(Ecosystem.Category))),
            Assembly.Method=unique(list(as.character(Assembly.Method))),
            Sequencing.Method=unique(list(as.character(Sequencing.Method))),
            Numeric.Add.Date=median(Numeric.Add.Date),
            Numeric.Sample.Date=median(Numeric.Sample.Date))
# Remove rows with missing latitude
agg.df <- agg.df[!is.na(agg.df$Latitude),]
agg.df <- agg.df[order(agg.df$Latitude),]

# Make the data frame for data aggrefated by geocoord
geo.df <- detCov.df %>% group_by(Longitude, Latitude) %>%
  summarize(mcrA=max(mcrA), mcrB=max(mcrB), mcrG=max(mcrG),
            pmoA=max(pmoA), pmoB=max(pmoB), pmoC=max(pmoC))
# Remove rows with missing latitude
geo.df <- geo.df[!is.na(geo.df$Latitude),]
geo.df <- geo.df[order(geo.df$Latitude),]

#####################################################################
# Single-species occupancy
#####################################################################

#####################################################################
# Un-aggregated models
#####################################################################
clean.df <- detCov.df[!is.na(detCov.df$Latitude),]
clean.df <- clean.df[!is.na(clean.df$Ecosystem),]
```

```
clean.df <- clean.df[!is.na(clean.df$Numeric.Add.Date),]
mcr.spp <- clean.df[1:3]
mcr.covs <- data.frame(clean.df[c("Latitude", "Ecosystem",
"Numeric.Add.Date")])
mcr.umf <- unmarkedFrameOccu(mcr.spp, siteCovs=mcr.covs)
pmo.spp <- clean.df[4:6]
pmo.covs <- data.frame(clean.df[c("Latitude", "Ecosystem",
"Numeric.Add.Date")])
pmo.umf <- unmarkedFrameOccu(pmo.spp, siteCovs=pmo.covs)

mcr.mod1 <- occu(~1 ~1, mcr.umf, engine="C")
mcr.mod2 <- occu(~sqrt(Numeric.Add.Date) ~1, mcr.umf, engine="C")
mcr.mod3 <- occu(~1 ~Ecosystem, mcr.umf, engine="C")
mcr.mod4 <- occu(~1 ~Latitude, mcr.umf, engine="C")
mcr.mod5 <- occu(~sqrt(Numeric.Add.Date) ~Latitude, mcr.umf, engine="C")

mcr.fl <- fitList(mcr.mod1, mcr.mod2, mcr.mod3, mcr.mod4, mcr.mod5)
mcr.ms <- modSel(mcr.fl)

pmo.mod1 <- occu(~1 ~1, pmo.umf, engine="C")
pmo.mod2 <- occu(~sqrt(Numeric.Add.Date) ~1, pmo.umf, engine="C")
pmo.mod3 <- occu(~1 ~Ecosystem, pmo.umf, engine="C")
pmo.mod4 <- occu(~1 ~Latitude, pmo.umf, engine="C")
pmo.mod5 <- occu(~sqrt(Numeric.Add.Date) ~Latitude, pmo.umf, engine="C")

pmo.fl <- fitList(pmo.mod1, pmo.mod2, pmo.mod3, pmo.mod4, pmo.mod5)
pmo.ms <- modSel(pmo.fl)

mcr.re3 <- ranef(mcr.mod3) # on latitude
sum(bup(mcr.re3, stat="mode"))

pmo.re3 <- ranef(pmo.mod3) # on latitude
sum(bup(pmo.re3, stat="mode"))

######################################################################
# Aggregated models
######################################################################

mcr.agg.spp <- agg.df[4:6]
mcr.agg.covs <-
data.frame(agg.df[c("Latitude","Ecosystem","Numeric.Add.Date")])
pmo.agg.spp <- agg.df[7:9]
pmo.agg.covs <-
data.frame(agg.df[c("Latitude","Ecosystem","Numeric.Add.Date")])

mcr.agg.umf <- unmarkedFrameOccu(mcr.agg.spp, mcr.agg.covs)
pmo.agg.umf <- unmarkedFrameOccu(pmo.agg.spp, pmo.agg.covs)

mcr.agg.mod1 <- occu(~1 ~1, mcr.agg.umf, engine="C")
mcr.agg.mod2 <- occu(~sqrt(Numeric.Add.Date) ~1, mcr.agg.umf, engine="C")
mcr.agg.mod3 <- occu(~1 ~Ecosystem, mcr.agg.umf, engine="C")
mcr.agg.mod4 <- occu(~1 ~Latitude, mcr.agg.umf, engine="C")
```

```
mcr.agg.mod5 <- occu(~sqrt(Numeric.Add.Date) ~Latitude, mcr.agg.umf,
engine="C")

mcr.agg.fl <- fitList(mcr.agg.mod1, mcr.agg.mod2, mcr.agg.mod3, mcr.agg.mod4,
mcr.agg.mod5)
mcr.agg.ms <- modSel(mcr.agg.fl)

pmo.agg.mod1 <- occu(~1 ~1, pmo.agg.umf, engine="C")
pmo.agg.mod2 <- occu(~sqrt(Numeric.Add.Date) ~1, pmo.agg.umf, engine="C")
pmo.agg.mod3 <- occu(~1 ~Ecosystem, pmo.agg.umf, engine="C")
pmo.agg.mod4 <- occu(~1 ~Latitude, pmo.agg.umf, engine="C")
pmo.agg.mod5 <- occu(~sqrt(Numeric.Add.Date) ~Latitude, pmo.agg.umf,
engine="C")

pmo.agg.fl <- fitList(pmo.agg.mod1, pmo.agg.mod2, pmo.agg.mod3, pmo.agg.mod4,
pmo.agg.mod5)
pmo.agg.ms <- modSel(pmo.agg.fl)

pmo.agg.pred <- predict(pmo.agg.mod5, type='state')
pmo.agg.pred$Latitude <- pmo.agg.covs$Latitude

mcr.agg.pred <- predict(mcr.agg.mod4, type='state')
mcr.agg.pred$Latitude <- mcr.agg.covs$Latitude

######################################################################
# Geo-aggregated (not on environment)
######################################################################
mcr.geo.spp <- geo.df[3:5]
mcr.geo.covs <- data.frame(geo.df[c("Latitude")])
pmo.geo.spp <- geo.df[6:8]
pmo.geo.covs <- data.frame(geo.df[c("Latitude")])

mcr.geo.umf <- unmarkedFrameOccu(mcr.geo.spp, mcr.geo.covs)
pmo.geo.umf <- unmarkedFrameOccu(pmo.geo.spp, pmo.geo.covs)

mcr.geo.mod1 <- occu(~1 ~1, mcr.geo.umf, engine="C")
mcr.geo.mod2 <- occu(~1 ~Latitude, mcr.geo.umf, engine="C")

mcr.geo.fl <- fitList(mcr.geo.mod1, mcr.geo.mod2)
mcr.geo.ms <- modSel(mcr.geo.fl)

pmo.geo.mod1 <- occu(~1 ~1, pmo.geo.umf, engine="C")
pmo.geo.mod2 <- occu(~1 ~Latitude, pmo.geo.umf, engine="C")

pmo.geo.fl <- fitList(pmo.geo.mod1, pmo.geo.mod2)
pmo.geo.ms <- modSel(pmo.geo.fl)

pmo.geo.pred <- predict(pmo.geo.mod2, type='state')
pmo.geo.pred$Latitude <- pmo.geo.covs$Latitude

mcr.geo.pred <- predict(mcr.geo.mod2, type='state')
mcr.geo.pred$Latitude <- mcr.geo.covs$Latitude
```

```
###########
# Plots
###########

# Occupancy by site type with metagenomes as site
newdat = data.frame(Ecosystem=c("Engineered", "Environmental", "Host-
associated"))
pred.mcr <- predict(mcr.mod3, type='state', newdata=newdat, appendData=TRUE)
pred.mcr$Gene <- "mcr"
pred.pmo <- predict(pmo.mod3, type='state', newdata=newdat, appendData=TRUE)
pred.pmo$Gene <- "pmo"

pred.df <- rbind(pred.mcr, pred.pmo)

pred.plot <- ggplot(pred.df, aes(x=Ecosystem, y=Predicted)) +
  scale_x_discrete(breaks=c("Engineerd", "Environmental", "Host-associated"),
labels=c("Engineered", "Environmental", "Host-associated")) +
  coord_cartesian(ylim=c(0.0,0.7)) +
  geom_point(aes(color=Gene), position=position_dodge(.3), size=2.2) +
  geom_errorbar(aes(ymin=lower, ymax=upper, color=Gene), width=.4,
position=position_dodge(.3), show.legend=FALSE) +
  labs(title = "Occupancy proportion\nwith metagenomes as sites", y =
"Predicted occupancy") +
  theme(text = element_text(size=13)) +
  scale_color_discrete(name="Function", breaks=c("mcr","pmo"),
                       labels = c(expression(italic("mcr")),
expression(italic("pmo"))))


# with aggregated sites on geocoordiates and ecosystem
agg.mcr <- predict(mcr.agg.mod3, type='state', newdata=newdat,
appendData=TRUE)
agg.mcr$Gene <- "mcr"
agg.pmo <- predict(pmo.agg.mod3, type='state', newdata=newdat,
appendData=TRUE)
agg.pmo$Gene <- "pmo"

agg.pred <- rbind(agg.mcr, agg.pmo)

agg.plot <- ggplot(agg.pred, aes(x=Ecosystem, y=Predicted)) +
  scale_x_discrete(breaks=c("Engineerd", "Environmental", "Host-associated"),
labels=c("Engineered", "Environmental", "Host-associated")) +
  coord_cartesian(ylim=c(0.0,0.7)) +
  geom_point(aes(color=Gene), position=position_dodge(.3), size=2.2) +
  geom_errorbar(aes(ymin=lower, ymax=upper, color=Gene), width=.4,
position=position_dodge(.3), show.legend=FALSE) +
  labs(title = "Occupancy proportion with sites\naggregated by
geocoordinate", y = "Predicted occupancy") +
  theme(text = element_text(size=13)) +
  scale_color_discrete(name="Function", breaks=c("mcr","pmo"),
                       labels = c(expression(italic("mcr")),
expression(italic("pmo"))))
```

```
#####
# Detection/Date Plots
#####

#date.pred.mcr <- predict(mcr.mod2, type='det',
newdata=data.frame(Numeric.Add.Date=mcr.covs$Numeric.Add.Date),
appendData=TRUE)
#date.pred.pmo <- predict(pmo.mod2, type='det',
newdata=data.frame(Numeric.Add.Date=pmo.covs$Numeric.Add.Date),
appendData=TRUE)

date.pred.mcr$Sqrt.Add.Date <- sqrt(mcr.covs$Numeric.Add.Date)
date.pred.mcr$Gene <- "mcr"
date.pred.pmo$Sqrt.Add.Date <- sqrt(pmo.covs$Numeric.Add.Date)
date.pred.pmo$Gene <- "pmo"

date.pred <- rbind(date.pred.mcr, date.pred.pmo)

date.pred$Date <- as.Date(date.pred$Numeric.Add.Date, origin="1970-01-01")

date.plot <- ggplot(date.pred, aes(x=Date, y=Predicted, colour=Gene)) +
  geom_line(aes(colour=Gene)) +
  geom_ribbon(aes(ymin=lower, ymax=upper, colour=Gene), linetype=2,
alpha=0.3) +
  labs(title="Detection probability vs. date for unaggregated data",
y="Predicted detection", x="Square-root-transfomred add date") +
  theme(text = element_text(size=13)) +
  scale_color_discrete(name="Function", breaks=c("mcr","pmo"),
                       labels = c(expression(italic("mcr")),
expression(italic("pmo"))))

######
# Latitude plots
######

mcr.lat <- predict(mcr.mod4, type='state',
newdata=data.frame(Latitude=mcr.covs$Latitude), appendData=TRUE)
mcr.lat$Latitude <- mcr.covs$Latitude
mcr.lat$Gene <- "mcr"

pmo.lat <- predict(pmo.mod4, type='state',
newdata=data.frame(Latitude=pmo.covs$Latitude), appendData=TRUE)
pmo.lat$Latitude <- pmo.covs$Latitude
pmo.lat$Gene <- "pmo"

lat.df <- rbind(mcr.lat, pmo.lat)

lat.plot <- ggplot(lat.df, aes(x=Latitude, y=Predicted, colour=Gene)) +
  coord_cartesian(xlim=c(0.0,0.4)) +
  geom_line() +
  geom_ribbon(aes(ymin=lower, ymax=upper, colour=Gene), linetype=2,
alpha=0.3) +
  labs(title="Unaggregated\n ", y="Predicted occupancy") +
```

```
  scale_color_discrete(name="Function", breaks=c("mcr","pmo"),
                       labels = c(expression(italic("mcr")),
expression(italic("pmo")))) +
  coord_flip() +
  theme(text = element_text(size=13))

mcr.agg.lat <- predict(mcr.agg.mod4, type='state',
newdata=data.frame(Latitude=mcr.agg.covs$Latitude), appendData=TRUE)
mcr.agg.lat$Latitude <- mcr.agg.covs$Latitude
mcr.agg.lat$Gene <- "mcr"

#pmo.agg.lat <- predict(pmo.agg.mod4, type='state',
newdata=data.frame(Latitude=pmo.agg.covs$Latitude), appendData=TRUE)
pmo.agg.lat$Latitude <- pmo.agg.covs$Latitude
pmo.agg.lat$Gene <- "pmo"

lat.agg.df <- rbind(mcr.agg.lat, pmo.agg.lat)

lat.agg.plot <- ggplot(lat.agg.df, aes(x=Latitude, y=Predicted, colour=Gene))
+
  coord_cartesian(xlim=c(0.0,0.4)) +
  geom_line() +
  geom_ribbon(aes(ymin=lower, ymax=upper, colour=Gene), linetype=2,
alpha=0.3) +
  labs(title="Aggregated by ecosystem\nand geocoordinate", y="Predicted
occupancy") +
  scale_color_discrete(name="Function", breaks=c("mcr","pmo"),
                       labels = c(expression(italic("mcr")),
expression(italic("pmo")))) +
  coord_flip() +
  theme(text = element_text(size=13))

mcr.geo.lat <- predict(mcr.geo.mod2, type='state',
newdata=data.frame(Latitude=mcr.geo.covs$Latitude), appendData=TRUE)
mcr.geo.lat$Latitude <- mcr.geo.covs$Latitude
mcr.geo.lat$Gene <- "mcr"

pmo.geo.lat <- predict(pmo.geo.mod2, type='state',
newdata=data.frame(Latitude=pmo.geo.covs$Latitude), appendData=TRUE)
pmo.geo.lat$Latitude <- pmo.geo.covs$Latitude
pmo.geo.lat$Gene <- "pmo"

lat.geo.df <- rbind(mcr.geo.lat, pmo.geo.lat)

lat.geo.plot <- ggplot(lat.geo.df, aes(x=Latitude, y=Predicted, colour=Gene))
+
  coord_cartesian(xlim=c(0.0,0.4)) +
  geom_line() +
  geom_ribbon(aes(ymin=lower, ymax=upper, colour=Gene), linetype=2,
alpha=0.3) +
  labs(title="Aggregated by \ngeocoordinate", y="Predicted occupancy") +
  scale_color_discrete(name="Function", breaks=c("mcr","pmo"),
```

```
                        labels = c(expression(italic("mcr")),
expression(italic("pmo")))) +
  coord_flip() +
  theme(text = element_text(size=13))

#########################################################################
# Multi-species occupancy
#########################################################################

#########################################################################
# Un-aggregated models
#########################################################################

mult.spp <- list ( as.matrix(mcr.spp),
                   as.matrix(pmo.spp) )
names(mult.spp) <- c("mcr", "pmo")
mult.covs <- data.frame(clean.df[c("Latitude", "Ecosystem",
"Numeric.Add.Date")])

mult.umf <- unmarkedFrameOccuMulti(mult.spp, siteCovs=mult.covs)

det1 = c("~1", "~1")
occ1 = c("~1", "~1", "~1")

det2 = c("~1", "~1")
occ2 = c("~sqrt(Numeric.Add.Date)", "~sqrt(Numeric.Add.Date)",
"~sqrt(Numeric.Add.Date)")

det3 = c("~1", "~1")
occ3 = c("~Ecosystem", "~Ecosystem", "~Ecosystem")

det4 = c("~1", "~1")
occ4 = c("~Latitude", "~Latitude", "~Latitude")

det5 = c("~1", "~1")
occ5 = c("~Ecosystem", "~Ecosystem", "~1")

det6 = c("~1", "~1")
occ6 = c("~1", "~1", "~Ecosystem")

det7 = c("~1", "~1")
occ7 = c("~Ecosystem", "~Ecosystem+sqrt(Numeric.Add.Date)", "~Ecosystem")

det8 = c("~1", "~1")
occ8 = c("~Ecosystem+(Numeric.Add.Date)",
"~Ecosystem+sqrt(Numeric.Add.Date)", "~Ecosystem")

det9 = c("~1", "~1")
occ9 = c("~Ecosystem+(Numeric.Add.Date)",
"~Ecosystem+sqrt(Numeric.Add.Date)", "~1")

det10 = c("~1", "~1")
occ10 = c("~Ecosystem+Latitude", "~Ecosystem+Latitude", "~1")
```

```
mult.mod1 = occuMulti( detformulas=det1, stateformulas=occ1, data=mult.umf,
engine="C")
mult.mod2 = occuMulti( detformulas=det2, stateformulas=occ2, data=mult.umf,
engine="C")
mult.mod3 = occuMulti( detformulas=det3, stateformulas=occ3, data=mult.umf,
engine="C")
mult.mod4 = occuMulti( detformulas=det4, stateformulas=occ4, data=mult.umf,
engine="C")
mult.mod5 = occuMulti( detformulas=det5, stateformulas=occ5, data=mult.umf,
engine="C")
mult.mod6 = occuMulti( detformulas=det6, stateformulas=occ6, data=mult.umf,
engine="C")
mult.mod7 = occuMulti( detformulas=det7, stateformulas=occ7, data=mult.umf,
engine="C")
mult.mod8 = occuMulti( detformulas=det8, stateformulas=occ8, data=mult.umf,
engine="C")
mult.mod9 = occuMulti( detformulas=det9, stateformulas=occ9, data=mult.umf,
engine="C")
mult.mod10 = occuMulti( detformulas=det10, stateformulas=occ10,
data=mult.umf, engine="C")

mult.fl <- fitList(mult.mod1, mult.mod2, mult.mod3, mult.mod4, mult.mod5,
mult.mod6, mult.mod7, mult.mod8, mult.mod9, mult.mod10)
mult.ms <- modSel(mult.fl)

######################################################################
# Aggregated models
######################################################################

agg.spp = list ( as.matrix(agg.df[4:6]),
                 as.matrix(agg.df[7:9]) )
names(agg.spp) <- c("mcr", "pmo")

agg.covs <- data.frame(agg.df[c("Latitude","Ecosystem","Numeric.Add.Date")])
agg.covs <- agg.covs[!is.na(agg.covs$Latitude),]
agg.covs$Ecosystem <- as.character(agg.covs$Ecosystem)

agg.umf = unmarkedFrameOccuMulti(agg.spp, siteCovs=agg.covs)

agg.mod1 = occuMulti( detformulas=det1, stateformulas=occ1, data=agg.umf,
engine="C")
agg.mod2 = occuMulti( detformulas=det2, stateformulas=occ2, data=agg.umf,
engine="C")
agg.mod3 = occuMulti( detformulas=det3, stateformulas=occ3, data=agg.umf,
engine="C")
agg.mod4 = occuMulti( detformulas=det4, stateformulas=occ4, data=agg.umf,
engine="C")
agg.mod5 = occuMulti( detformulas=det5, stateformulas=occ5, data=agg.umf,
engine="C")
agg.mod6 = occuMulti( detformulas=det6, stateformulas=occ6, data=agg.umf,
engine="C")
```

```
agg.mod7 = occuMulti( detformulas=det7, stateformulas=occ7, data=agg.umf,
engine="C")
agg.mod8 = occuMulti( detformulas=det8, stateformulas=occ8, data=agg.umf,
engine="C")
agg.mod9 = occuMulti( detformulas=det9, stateformulas=occ9, data=agg.umf,
engine="C")
agg.mod10 = occuMulti( detformulas=det10, stateformulas=occ10, data=agg.umf,
engine="C")

agg.fl <- fitList(agg.mod1, agg.mod2, agg.mod3, agg.mod4, agg.mod5, agg.mod6,
agg.mod7, agg.mod8, agg.mod9, agg.mod10)
agg.ms <- modSel(agg.fl)

geo.spp = list ( as.matrix(geo.df[3:5]),
                 as.matrix(geo.df[6:8]) )
names(geo.spp) <- c("mcr", "pmo")

geo.covs <- data.frame(geo.df["Latitude"])
geo.covs <- data.frame(geo.covs[!is.na(geo.covs$Latitude),])
colnames(geo.covs) <- c("Latitude")

geo.umf = unmarkedFrameOccuMulti(geo.spp, siteCovs=geo.covs)

geo.det1 = c("~1", "~1")
geo.occ1 = c("~1", "~1", "~1")

geo.det2 = c("~1", "~1")
geo.occ2 = c("~Latitude", "~Latitude", "~Latitude")

geo.det3 = c("~1", "~1")
geo.occ3 = c("~Latitude", "~Latitude", "~1")

geo.det4 = c("~1", "~1")
geo.occ4 = c("~1", "~1", "~Latitude")

geo.mod1 = occuMulti( detformulas=geo.det1, stateformulas=geo.occ1,
data=geo.umf, engine="C")
geo.mod2 = occuMulti( detformulas=geo.det2, stateformulas=geo.occ2,
data=geo.umf, engine="C")
geo.mod3 = occuMulti( detformulas=geo.det3, stateformulas=geo.occ3,
data=geo.umf, engine="C")
geo.mod4 = occuMulti( detformulas=geo.det4, stateformulas=geo.occ4,
data=geo.umf, engine="C")

geo.fl <- fitList(geo.mod1, geo.mod2, geo.mod3, geo.mod4)
geo.ms <- modSel(geo.fl)

########
# Plots
########
########
# Un-aggregated Data
########
```

```
mult.newDat = mult.covs[c("Ecosystem", "Latitude")]

mult.mcr.10 <- predict(mult.mod10, 'state', species='mcr', cond='-pmo',
newdat=mult.newDat, appendData=TRUE) # mcr | pmo absent
mult.mcr.10$Type <- "Ψ(mcr|-pmo)"
mult.mcr.10$Latitude <- mult.newDat$Latitude
mult.mcr.10$Ecosystem <- mult.newDat$Ecosystem
mult.mcr.11 <- predict(mult.mod10, 'state', species='mcr', cond='pmo',
newdat=mult.newDat, appendData=TRUE)  # mcr | pmo present
mult.mcr.11$Type <- "Ψ(mcr|pmo)"
mult.mcr.11$Latitude <- mult.newDat$Latitude
mult.mcr.11$Ecosystem <- mult.newDat$Ecosystem

mcr.mult.pred <- rbind(mult.mcr.10, mult.mcr.11)
mcr.mult.Eng = mcr.mult.pred[mcr.mult.pred$Ecosystem=="Engineered",]
mcr.mult.Env = mcr.mult.pred[mcr.mult.pred$Ecosystem=="Environmental",]
mcr.mult.Hos = mcr.mult.pred[mcr.mult.pred$Ecosystem=="Host-associated",]

lab1 = paste( "Ψ(", expression(italic("mcr")), "|-",
expression(italic("pmo")), ")")
lab2 = paste( "Ψ(", expression(italic("mcr")), "|",
expression(italic("pmo")), ")")

mcr.mult.plot.eng <- ggplot(mcr.mult.Eng, aes(x=Latitude, y=Predicted,
colour=Type)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Engineered sites (mcr)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

mcr.mult.plot.env <- ggplot(mcr.mult.Env, aes(x=Latitude, y=Predicted,
colour=Type)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Environmental sites (mcr)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

mcr.mult.plot.hos <- ggplot(mcr.mult.Hos, aes(x=Latitude, y=Predicted,
colour=Type)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Host-associated sites (mcr)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")
```

```
### PMO plots ###
pmo.10 <- predict(mult.mod10, 'state', species='pmo', cond='-mcr',
newdat=mult.newDat, appendData=TRUE) # pmo | mcr absent
pmo.10$Type <- "Ψ(pmo|-mcr)"
pmo.10$Latitude <- mult.newDat$Latitude
pmo.10$Ecosystem <- mult.newDat$Ecosystem
pmo.11 <- predict(mult.mod10, 'state', species='pmo', cond='mcr',
newdat=mult.newDat, appendData=TRUE)  # pmo | mcr present
pmo.11$Type <- "Ψ(pmo|mcr)"
pmo.11$Latitude <- mult.newDat$Latitude
pmo.11$Ecosystem <- mult.newDat$Ecosystem

pmo.mult.pred <- rbind(pmo.10, pmo.11)
pmo.mult.Eng = pmo.mult.pred[pmo.mult.pred$Ecosystem=="Engineered",]
pmo.mult.Env = pmo.mult.pred[pmo.mult.pred$Ecosystem=="Environmental",]
pmo.mult.Hos = pmo.mult.pred[pmo.mult.pred$Ecosystem=="Host-associated",]

pmo.mult.plot.eng <- ggplot(pmo.mult.Eng, aes(x=Latitude, y=Predicted,
colour=Type)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Engineered sites (pmo)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

pmo.mult.plot.env <- ggplot(pmo.mult.Env, aes(x=Latitude, y=Predicted,
colour=Type)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Environmental sites (pmo)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

pmo.mult.plot.hos <- ggplot(pmo.mult.Hos, aes(x=Latitude, y=Predicted,
colour=Type)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Host-assocated sites (pmo)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

grid.arrange(mcr.mult.plot.eng, pmo.mult.plot.eng, mcr.mult.plot.env,
pmo.mult.plot.env, mcr.mult.plot.hos, pmo.mult.plot.hos, nrow=3, ncol=2)

#########
# Aggregated Data
#########
```

```
agg.newDat = agg.covs[c("Ecosystem", "Latitude")]

mcr.10 <- predict(agg.mod10, 'state', species='mcr', cond='-pmo',
newdat=agg.newDat, appendData=TRUE) # mcr | pmo absent
mcr.10$Type <- "psi(mcr|-pmo)"
mcr.10$Latitude <- agg.newDat$Latitude
mcr.10$Ecosystem <- agg.newDat$Ecosystem
mcr.11 <- predict(agg.mod10, 'state', species='mcr', cond='pmo',
newdat=agg.newDat, appendData=TRUE)  # mcr | pmo present
mcr.11$Type <- "psi(mcr|pmo)"
mcr.11$Latitude <- agg.newDat$Latitude
mcr.11$Ecosystem <- agg.newDat$Ecosystem

mcr.agg.pred <- rbind(mcr.10, mcr.11)
mcr.agg.Eng = mcr.agg.pred[mcr.agg.pred$Ecosystem=="Engineered",]
mcr.agg.Env = mcr.agg.pred[mcr.agg.pred$Ecosystem=="Environmental",]
mcr.agg.Hos = mcr.agg.pred[mcr.agg.pred$Ecosystem=="Host-associated",]

mcr.agg.plot.eng <- ggplot(mcr.agg.Eng, aes(x=Latitude, y=Predicted,
colour=Type)) +
   coord_cartesian(ylim=c(0,1.0), xlim=c(-90,90)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Engineered sites (mcr)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

mcr.agg.plot.env <- ggplot(mcr.agg.Env, aes(x=Latitude, y=Predicted,
colour=Type)) +
   coord_cartesian(ylim=c(0,1.0), xlim=c(-90,90)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Environmental sites (mcr)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

mcr.agg.plot.hos <- ggplot(mcr.agg.Hos, aes(x=Latitude, y=Predicted,
colour=Type)) +
   coord_cartesian(ylim=c(0,1.0), xlim=c(-90,90)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Host-associated sites (mcr)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

### PMO plots ###
```

```
pmo.10 <- predict(agg.mod10, 'state', species='pmo', cond='-mcr',
newdat=agg.newDat, appendData=TRUE) # pmo | mcr absent
pmo.10$Type <- "psi(pmo|-mcr)"
pmo.10$Latitude <- agg.newDat$Latitude
pmo.10$Ecosystem <- agg.newDat$Ecosystem
pmo.11 <- predict(agg.mod10, 'state', species='pmo', cond='mcr',
newdat=agg.newDat, appendData=TRUE)  # pmo | mcr present
pmo.11$Type <- "psi(pmo|mcr)"
pmo.11$Latitude <- agg.newDat$Latitude
pmo.11$Ecosystem <- agg.newDat$Ecosystem

pmo.agg.pred <- rbind(pmo.10, pmo.11)
pmo.agg.Eng = pmo.agg.pred[pmo.agg.pred$Ecosystem=="Engineered",]
pmo.agg.Env = pmo.agg.pred[pmo.agg.pred$Ecosystem=="Environmental",]
pmo.agg.Hos = pmo.agg.pred[pmo.agg.pred$Ecosystem=="Host-associated",]

pmo.agg.plot.eng <- ggplot(pmo.agg.Eng, aes(x=Latitude, y=Predicted,
colour=Type)) +
   coord_cartesian(ylim=c(0,1.0), xlim=c(-90,90)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Engineered sites (pmo)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

pmo.agg.plot.env <- ggplot(pmo.agg.Env, aes(x=Latitude, y=Predicted,
colour=Type)) +
   coord_cartesian(ylim=c(0,1.0), xlim=c(-90,90)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Environmental sites (pmo)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

pmo.agg.plot.hos <- ggplot(pmo.agg.Hos, aes(x=Latitude, y=Predicted,
colour=Type)) +
   coord_cartesian(ylim=c(0,1.0), xlim=c(-90,90)) +
   geom_line() +
   geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
   geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
   labs(title="Host-assocated sites (pmo)", y="Predicted occupancy") +
   coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
   theme(text = element_text(size=13), legend.position = "none")

grid.arrange(mcr.agg.plot.eng, pmo.agg.plot.eng, mcr.agg.plot.env,
pmo.agg.plot.env, mcr.agg.plot.hos, pmo.agg.plot.hos, nrow=3, ncol=2)

########
```

```
# Geo-aggregated Data
########

geo.newDat = geo.covs[c("Latitude")]

mcr.geo.10 <- predict(geo.mod3, 'state', species='mcr', cond='-pmo',
newdat=geo.newDat, appendData=TRUE) # mcr | pmo absent
mcr.geo.10$Type <- "psi(mcr|-pmo)"
mcr.geo.10$Latitude = geo.newDat$Latitude
mcr.geo.11 <- predict(geo.mod3, 'state', species='mcr', cond='pmo',
newdat=geo.newDat, appendData=TRUE) # mcr | pmo present
mcr.geo.11$Type <- "psi(mcr|pmo)"
mcr.geo.11$Latitude = geo.newDat$Latitude

mcr.geo.pred <- rbind(mcr.geo.10, mcr.geo.11)

pmo.geo.10 <- predict(geo.mod3, 'state', species='pmo', cond='-mcr',
newdat=geo.newDat, appendData=TRUE) # pmo | mcr absent
pmo.geo.10$Type <- "psi(mcr|-pmo)"
pmo.geo.10$Latitude = geo.newDat$Latitude
pmo.geo.11 <- predict(geo.mod3, 'state', species='pmo', cond='mcr',
newdat=geo.newDat, appendData=TRUE) # pmo | mcr present
pmo.geo.11$Type <- "psi(mcr|pmo)"
pmo.geo.11$Latitude = geo.newDat$Latitude

pmo.geo.pred <- rbind(pmo.geo.10, pmo.geo.11)

mcr.geo.plot <- ggplot(mcr.geo.pred, aes(x=Latitude, y=Predicted,
colour=Type)) +
  coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
  geom_line() +
  geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
  geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
  labs(y="Predicted occupancy") +
  theme(text = element_text(size=13), legend.position = "none")

pmo.geo.plot <- ggplot(pmo.geo.pred, aes(x=Latitude, y=Predicted,
colour=Type)) +
  coord_flip(ylim=c(0,1.0), xlim=c(-90,90)) +
  geom_line() +
  geom_ribbon(aes(ymin=lower, ymax=upper, colour=Type), linetype=2,
alpha=0.3) +
  geom_point(aes(x=Latitude, y=Predicted, colour=Type)) +
  labs(y="Predicted occupancy") +
  theme(text = element_text(size=13), legend.position = "none")

grid.arrange(mcr.geo.plot, pmo.geo.plot, nrow=2)

#########
# World map
#########
```

```
worldMap <- borders("world", colour="gray50", fill="white")
colours <- c("#10B211","#FF5300","#1617D2")
names(colours) <- c("Environmental", "Host-associated", "Engineered")

siteMap <- ggplot(detCov.df) +
  worldMap +
  geom_point(aes(x=Longitude, y=Latitude, colour=Ecosystem), size=.6) +
  scale_color_manual(values=colours) +
  labs(title="Map of metagenome locations", y="Latitude", x="Longitude")
```

Appendix D

Supplemental figures

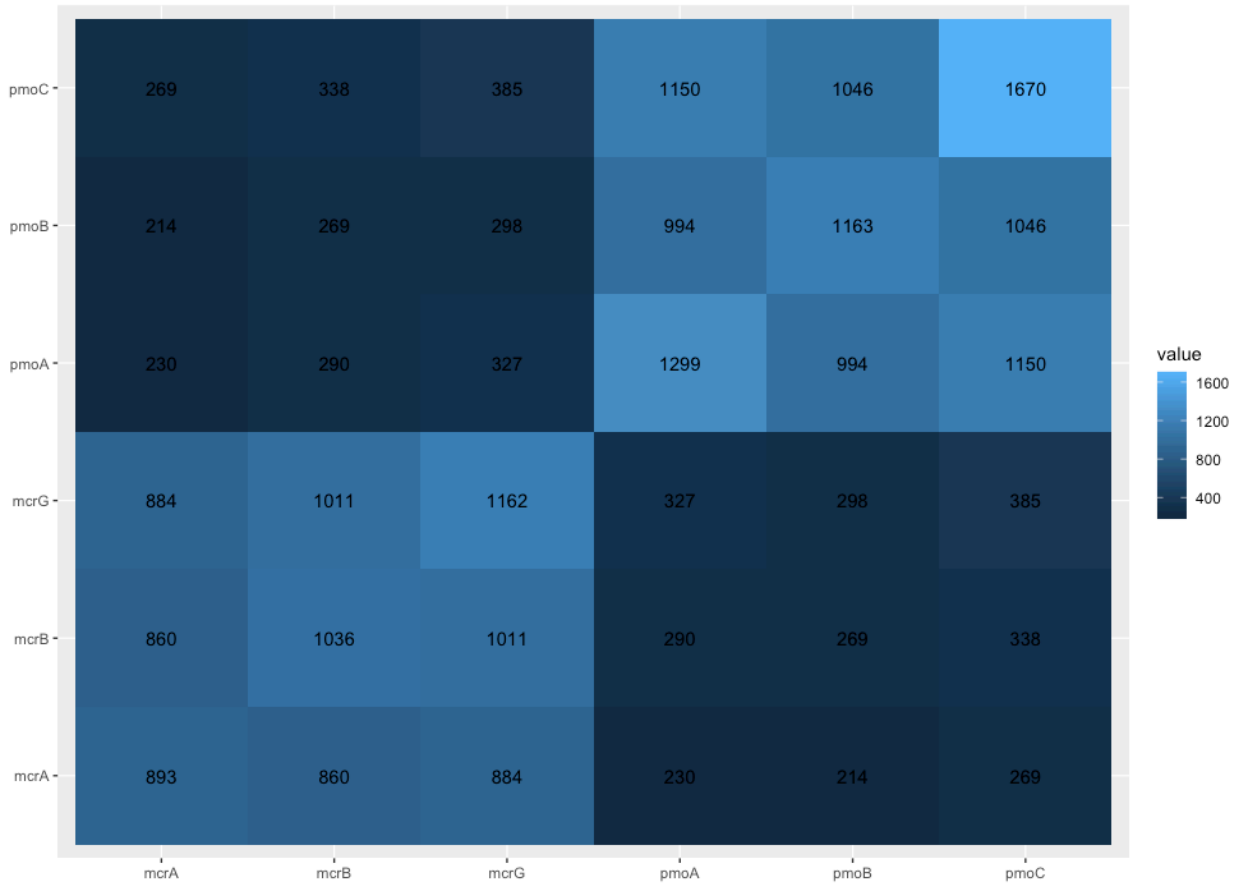

Figure S1: Co-occurrence matrix for the different genes using the un-aggregated presence/absence data. Shading indicates the strength of the co-occurrence and the numbers in each tile indicate the number of co-occurrences for the given gene pair.

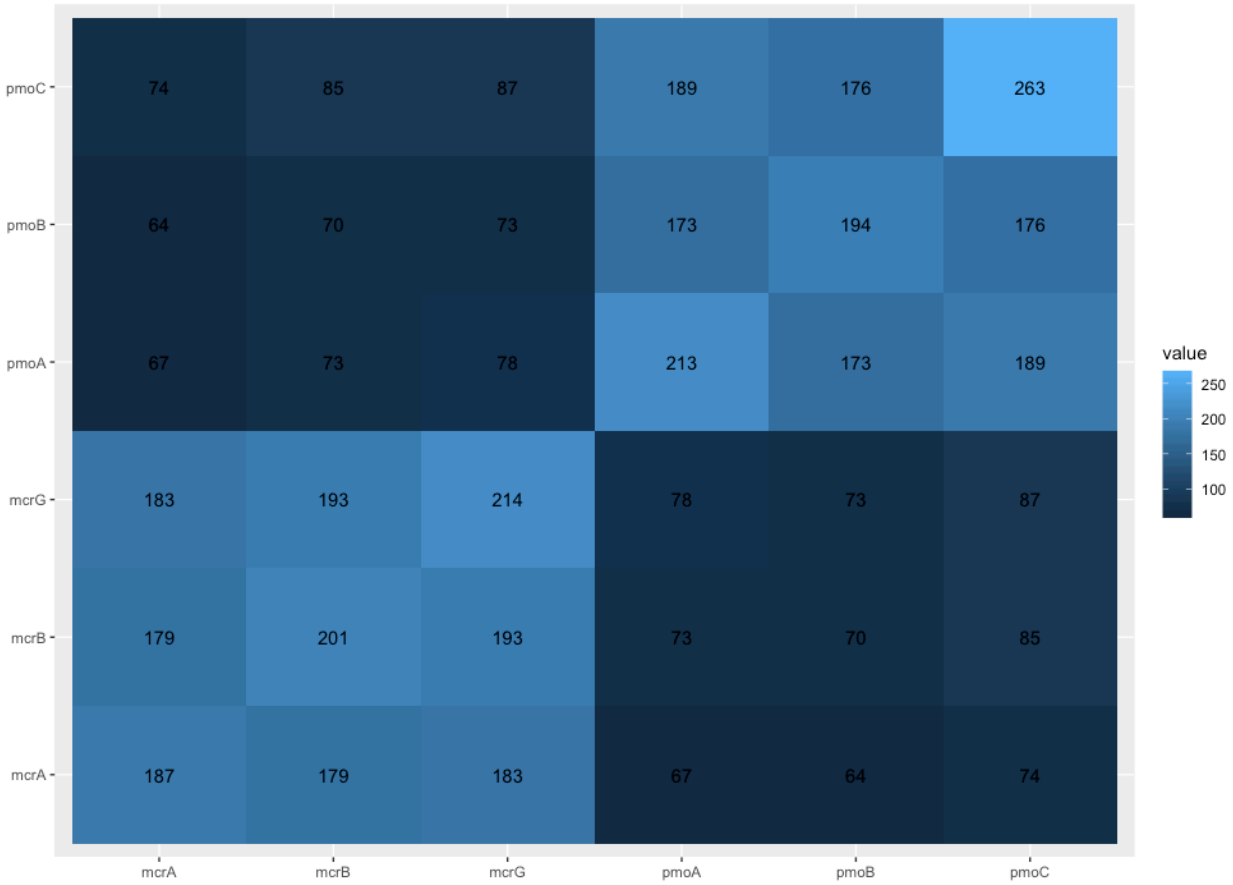

Figure S2: Co-occurrence matrix for the different genes using the data aggregated by geocoordinate and ecosystem type. Shading indicates the relative strength of the co-occurrence and the number in each tile indicates the number of co-occurrences.

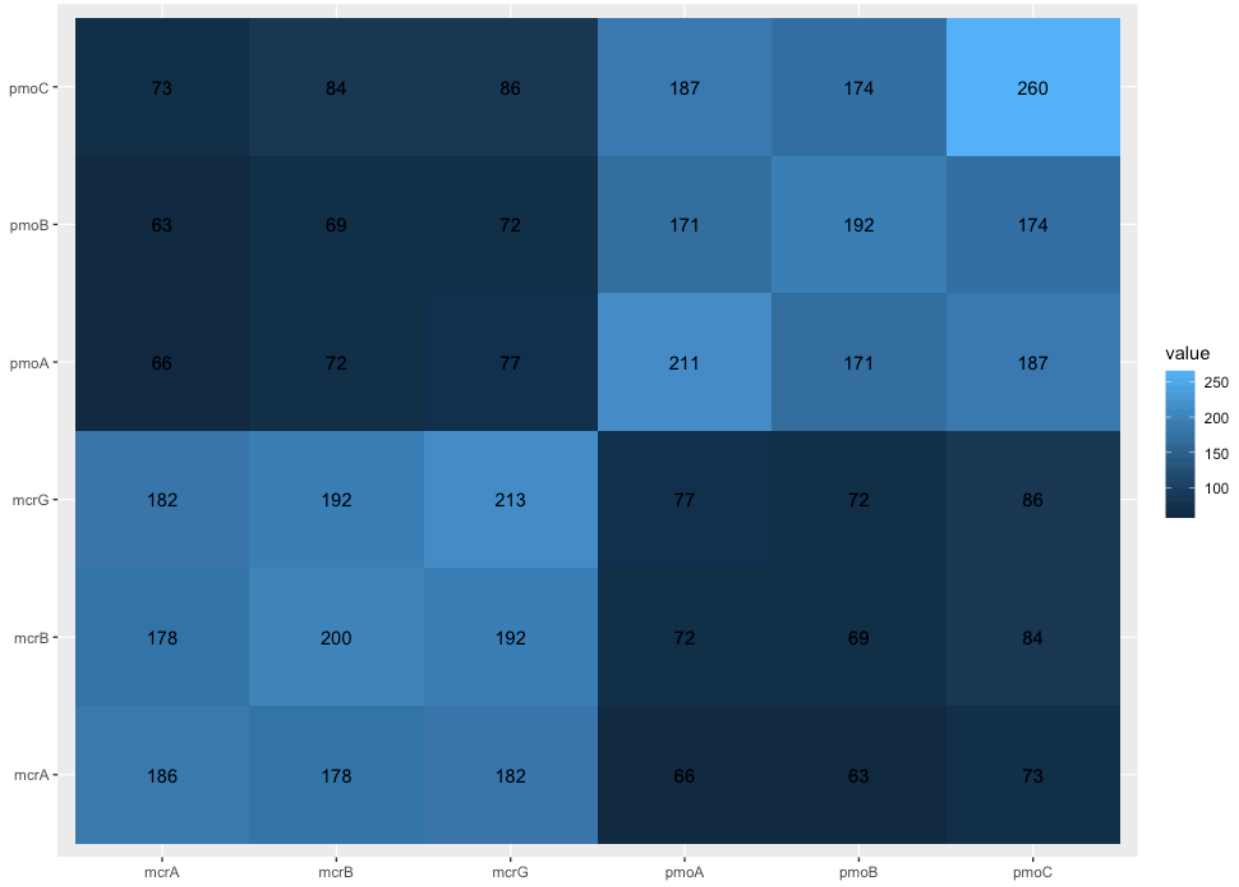

Figure S3: Co-occurrence matrix for the different genes using the presence/absence data that was aggregated by geocoordinate only. Shading indicates the strength of the co-occurrence and the number in each tile indicates the number of co-occurrences.

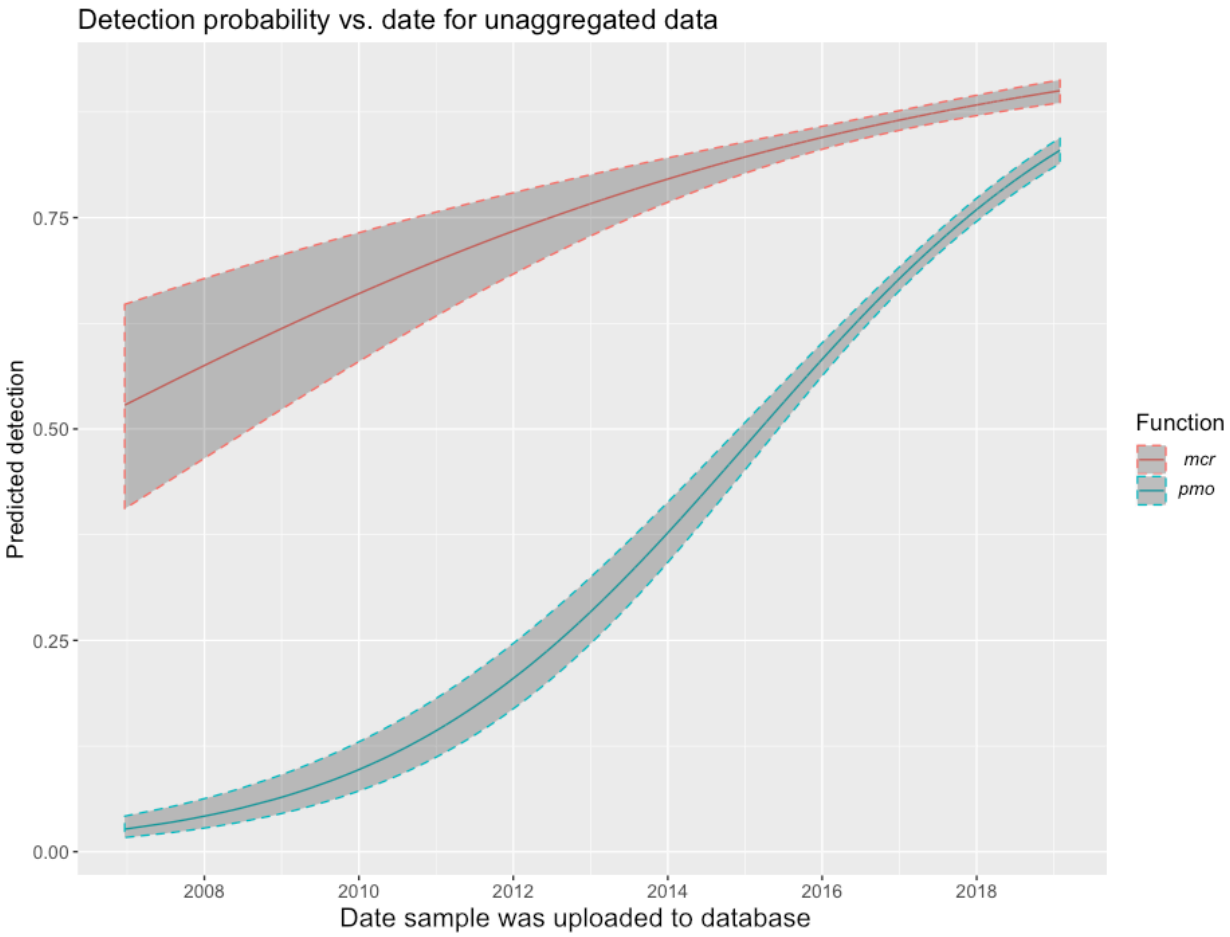

Figure S4: Detection probability versus the date that the sample was uploaded. The model used days since 2006-01-01, which were then square root transformed. There was relatively little difference from using the default 1970-01-01 start date for conversion to numerical format.

Glossary

Algorithm; a sequence of steps to produce a well-defined output, given some well-defined input.

Array; a contiguous block of memory in a computer, of some fixed size, used to store values (see buffer).

Base-calling (sequencing); The process of determining a nucleotide sequence from the signal produced by a sequencing platform.

Bit; a single binary digit (either 0 or 1).

Buffer; a continuous block of memory in a computer, of some fixed size, used to store values (see array).

Byte; a byte consists of 8 bits of data.

Contig; a contiguous, or unbroken, DNA sequence. Usually referring to the unbroken sequences an assembler produces.

Coverage (sequencing); refers to the number of raw reads that map to a specific region of an assembly, measured in times (*e.g.* 25x indicates 25 reads mapped to a region)

Depth (sequencing); synonymous to 'coverage'.

Graph (mathematics); a structure consisting of a series of vertices joined to one another by edges. Edges can either be directed (going from one vertex to another) or undirected (going in either direction between vertices).

Interface (computer science); the collective set of functions used to interact with a module of code, used to separate the internal working of a code base from the user of the code base.

Kmer; a DNA sequence of length k, where k is typically a relatively small number.

Linked-list; a series of items connected by pointers, allowing large amounts of connected data to be stored without the need to reserve large amounts of contiguous memory.

Long-read; a DNA sequencing read longer than the reads produced by NGS sequencing platforms, often exceeding one thousand base pairs.

Plain text; a format of storing data in which every symbol is represented by a byte, and there is no compression. These files are readily open in a human-readable format.

Pointer; an address in computer memory, used to refer to where some data is being stored, often a structure or list element.

Read; a contiguous, fully determined length of DNA produced by a sequencer, after base-calling.

Short-read; a DNA sequencing read from an NGS platform, typically under 300 base pairs long.

String (computer science); a series of symbols, such as a sentence or DNA sequence.

Thrashing; refers to a situation when a computer's memory is full during the runtime of a program, causing frequent accesses to the hard drive, and ultimately resulting in the program slowing down.

Throughput (sequencing); referring to the rate at which a sequencing platform can sequence DNA, often measured in base pairs per unit time.

Wrapper; in program design, a wrapper refers to something that encapsulates another thing. Most often, this refers to a structure that houses some other data structure in order to contain extra data.