School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Bioinformatics of the Immune System

by

Lasath Fernando

Thesis submitted as a requirement for the degree of

Bachelor of Engineering in Software Engineering

Submitted:	May 31, 2015	Student ID:	z3373562
Supervisor:	Dr. Bruno Gaëta	Topic ID:	0278

Abstract

This document describes the requirements to theses submitted for the Bachelor of Engineering in Software Engineering degree at the School of Computer Science and Engineering. Requirements described are that of both of context and layout of the theses. The document is written using the LaTeX template provided by the school.

Acknowledgements

This work has been inspired by the labours of numerous academics in the Faculty of Engineering at UNSW who have endeavoured, over the years, to encourage students to present beautiful concepts using beautiful typography.

Further inspiration has come from Donald Knuth who designed TeX, for typesetting technical (and non-technical) material with elegance and clarity; and from Leslie Lamport who contributed LaTeX, which makes TeX usable by mortal engineers.

John Zaitseff, an honours student in CSE at the time, created the first version of the UNSW Thesis LaTeX class and the author of the current version is indebted to his work.

Abbreviations

HMM Hidden Markov Model

BLAST Basic Local Alignment Search Tool

XML Extensible Mark-up Language

Contents

List of Figures

Chapter 1

Introduction

1.1 Immunoglobulin recombination

The human immune system faces many challenges when dealing with foreign pathogens invading the body. One of them is identifying and classifying antigens that are not only vastly diverse, but also constantly mutating and evolving.

It does so by creating immunoglobulins (also called antibodies) whose job is to identify and bind to certain antigens, effectively marking them for destruction by the rest of the immune system. The process, by necessity is highly selective - a given antibody will only bind to a very specific antigen.

However, to deal with the highly diverse and evolving nature of foreign antigens, antibodies must mutate very rapidly. This allows them to be as diverse as the antigens they face, and adapt to bind to newly encountered pathogens.

The process (essentially randomly) selects 3 germline genes from a repertoire in the person's DNA potentially consisting of between 38 and 45 IGHV genes [13, 14], 23 IGHD genes [12] and 6 IGHJ [21] genes. These selected genes are then mutated (adding, removing or changing nucleotides) to generate further diversity. The antibodies that successfully bind to the antigen emit a chemical signal, which then encourages the
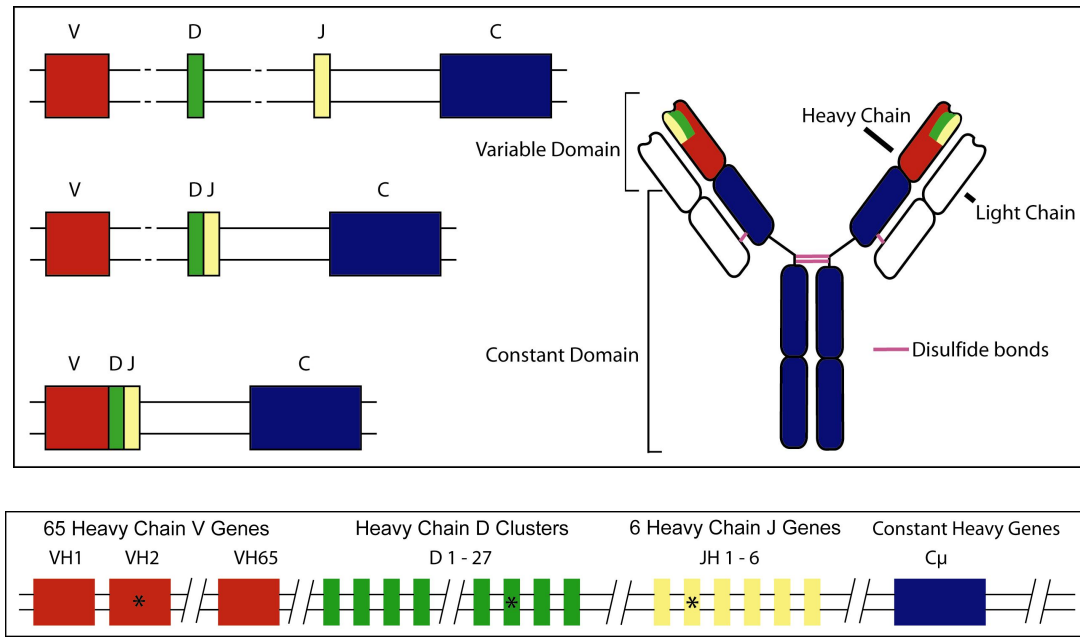
Katherine Jackson

Figure 1.1: The IGHV, IGHD and IGHJ gene recombination process
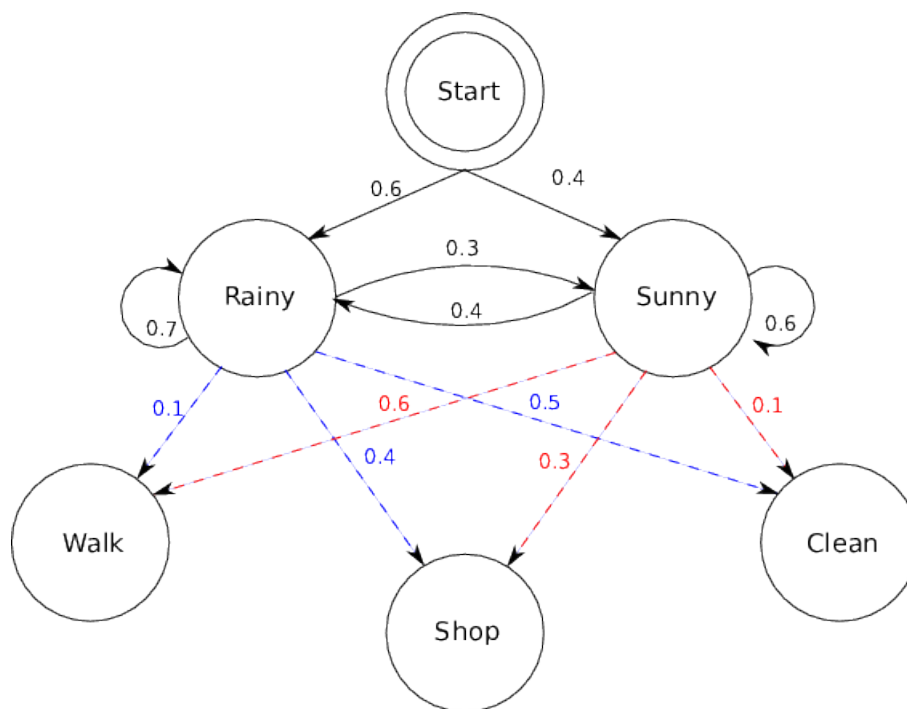
Figure 1.2: An example Hidden Markov Model [28]

production of similar antibodies. This entire process is illustrated in 1.1 and explained in more detail by Gaëta *et al.*[7].

1.2   iHMMuneAlign

As a result of this gene recombination and mutation process, the germline genes often don't provide much information on rearranged genes found in the B-Cells that create the antibodies themselves.

Developed in 2007 by a team of researchers at UNSW, iHMMuneAlign attempts to simulate the process of immunoglobulin gene rearrangement.[7].It represents the recombination process of immunoglobulin heavy chain genes as a hidden Markov model, and then uses it to find the most probable set of germline genes that created the rearranged gene.

First, it selects the most likely IGHV germline gene from its repertoire of possible V genes, using the NCBI's Basic Local Alignment Search Tool (BLAST) [1]. It then constructs a Hidden Markov Model with that IGHV gene, and all the possible IGHD and IGHJ genes, together with other states representing the somatic mutation.

It then uses the Viterbi algorithm to determine the most likely sequence of states (genes and their mutations) that caused the observed emissions (the rearranged sequence given as input).

1.3   Hidden Markov Models

A Hidden Markov Model is effectively a Markov Chain, where the states cannot be observed directly. They are used in a wide variety of machine learning applications, including speech recognition [6].

An example of a HMM is shown in 1.2. In this example, if the sequence of emissions

`Walk` was observed, the probability of it being `Sunny` is

$$I\,(\text{Sunny}) \times E(\text{Sunny}, \text{Walk}) = 0.4 \times 0.6 = 0.24$$

and the probability of it being Rainy is

$$I(\text{Rainy}) \times E(\text{Rainy}, \text{Walk}) = 0.6 \times 0.1 = 0.06$$

where $I(X)$ is the initial probability of X and $E(X, Y)$ is the emission probability of state Y in state X.

If `Walk, Clean` were observed. The probability of it having been Rainy then Sunny is

$$I(\text{Rainy}) \times E(\text{Rainy}, \text{Walk}) \times T(\text{Rainy, Sunny}) \times E(\text{Sunny, Clean}) = 0.6 \times 0.1 \times 0.3 \times 0.1 = 0.0018$$

where $T(X, Y)$ is the probability of transitioning from state X to state Y.

The Viterbi algorithm uses dynamic programming to recursively explore these probabilities, and selects the sequence of states with the highest probability at the end.

Chapter 2

Analysis

Since the high level aim of this project is to improve the performance of iHMMuneAlign, it was necessary to analyse the existing implementation in order to gain insight into its structure, as well as to create realistic aims.

2.1   Run Time

One of the primary attributes of performance, is run time. We begin by measuring the run time of the existing implementation and -more importantly- how it grows with workload.

The current implementation consists of a Java executable and a shell script, to allow running in batches. The Java executable (which does the actual processing) takes a range of input from the script and attempts to process that many in parallel. It then invokes Java program after breaking up the entire range into batches of a given size.

Since we are interested in the rate the total runtime changes with workload as well with increasing parallelism, both were measured. It was run with a total workload ranging from 1 sequences to 100 sequences, with batch size ranging from 1 to 8. The results are shown in 2.1.
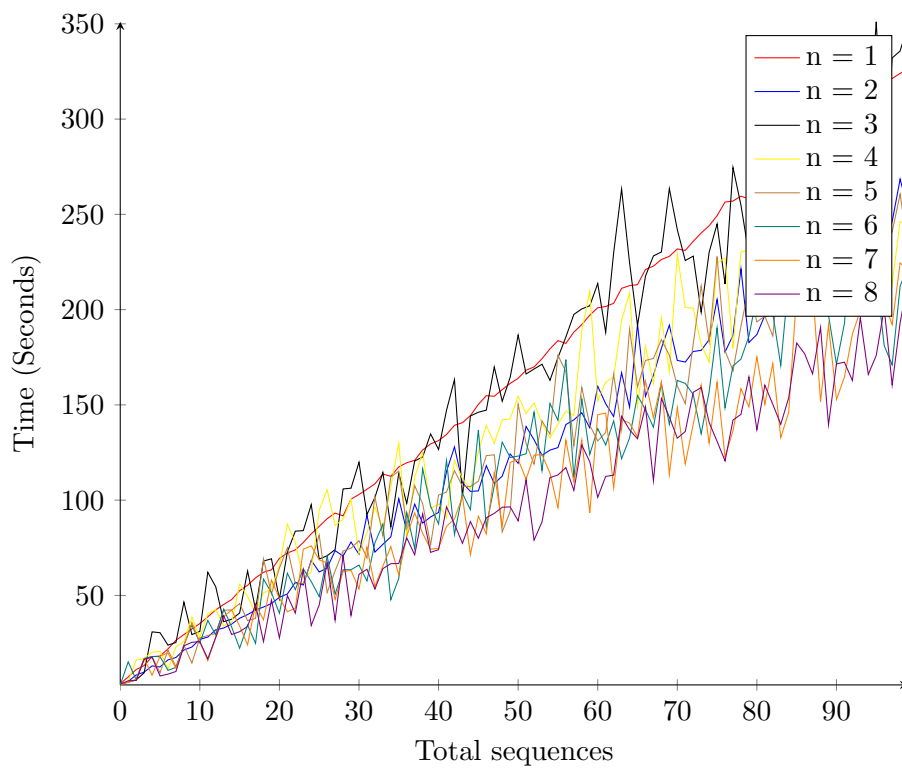
Figure 2.1: Total time taken (in seconds) to process the number of sequences (Wall-clock time). n is the number of sequences per invocation of the program
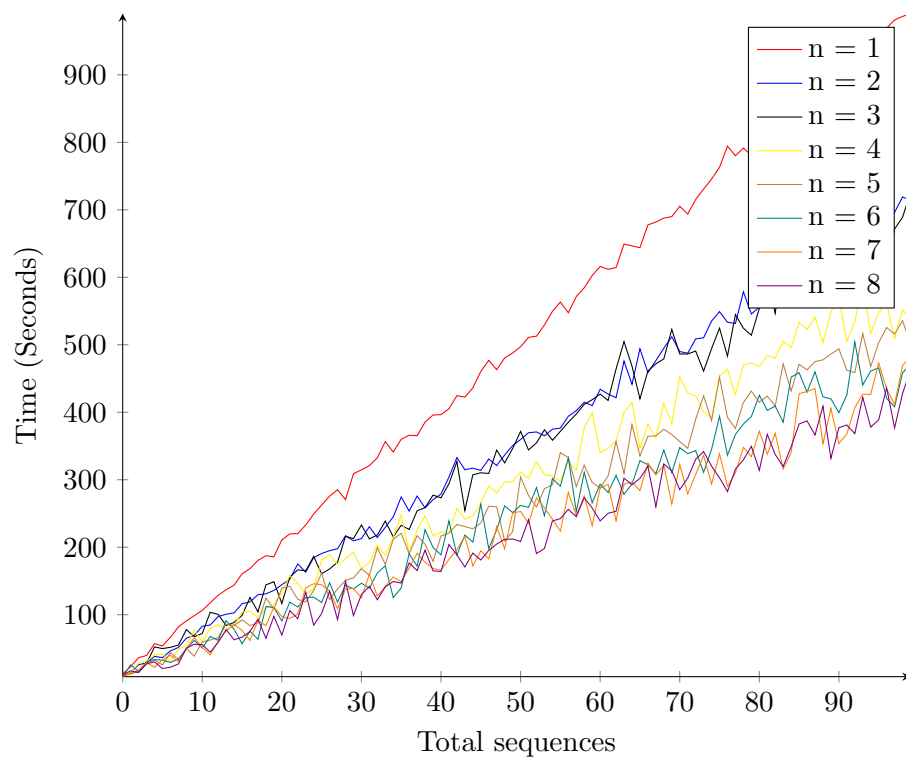
Figure 2.2: Total time spent doing work by each CPU on behalf of the process during execution (CPU Time). n is the number of sequences per invocation of the program

Note that this was performed on a machine with an AMD Phenom II 1090T (6-core) processor, 8GB of RAM and a solid state hard drive.

2.1.1   Interpretation of Results

With a batch size of 1, the expected linear increase was present. However, increasing the batch size shows a more sporadic decrease in runtime. There could be a number of causes for this, including the threads being assigned different CPUs by the scheduler on some runs but not others, increasing cache misses. Nonetheless, it wasn't investigated further, due to time constraints, and the fact that we didn't need such high accuracy for this test. However, there is an overall decrease in total execution time, as the batch size increases.

This suggests that the existing implementation is concurrent, and will utilize all the available processing power on the CPU. However, that should have diminishing returns once the amount of threads exceeds the total number of CPU cores on the machine, which is not the case here - execution time continues to decrease at roughly the same pace as n increases.

Looking at the total CPU time spent on the process, rather than total execution time provided a theory (which was later confirmed in Section 2.2.2, on page 11) to explain this behaviour. For each invocation of the Java program, there is a significant amount of overhead. For example, the Java Virtual Machine (JVM) needs to be initialized, it needs to load the program, analyse and perform any JIT compilation (Just-in-Time compilation, to improve execution time [23]). The program also needs to read in the repertoire data files for each gene and process them. On top of all that, any data that are being cached by the operating system on behalf of the program will be invalidated. Since this only needs to be done once per batch, as the batch sizes increases (and the number of batches decrease), this overhead also decreases.

This is clearly apparent in 2.2, which shows that the total work done by the program to process all the sequences is reduced significantly as the n increases.

Comparing the total executable time (wall clock time) and the total CPU time of each run shows, that there was roughly double the CPU time as execution time. This indicates that the program has 2 threads of execution, but that it doesn't increase threads with workload.

2.2 Profiling

In order to determine the most appropriate approaches for improving performance and overcoming the limitations of the original implementation, further analysis is required. Specifically, we are interested in which parts of the program consume the most CPU time and memory, as well as how they increase with the workload.

The method for collecting this data is called profiling. Since the original implementation of iHMMuneAlign was written in Java, we can use various interfaces in the Java Virtual Machine (JVM) to gather this data. The JVM has an interface called Java Native Interface (JNI)[18], that allows external programs and native libraries (usually written in C/C++) to interact with Java objects. Alongside that, from Java version 1.5 onwards, the JVM also provides another interface, called Java Virtual Machine Tool Interface (JVMTI), which allows external tools to be notified as certain events (such as object allocation, method call) occur during execution.

In 2.4, the data for a run of iHMMuneAlign with 6 sequences, instrumented by jProfiler [22] (a profiler implemented with the above method) is shown.

2.2.1 CPU Call Graph

Figure 2.4 shows the time, and the percentage of total execution time spent in each method invocation of a single execution of iHMMuneAlign.

It begins with the equivalent of the `main()` function for a thread using Java's built in threading library - `Thread.run()`, which in turn calls `startAlignment()`. However, it

Figure 2.3: Time spent by each thread; running (▬) and waiting (▬)

Figure 2.4: The call graph generated by jProfiler, for a single run of iHMMuneAlign
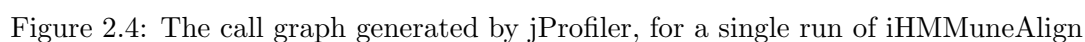
becomes immediately apparent that the majority of execution time from there is spent in the BioJava library.

68% of total execution time is spent constructing a `org.biojava.bio.dp.SingleDP` object, which will allow various algorithms (including Viterbi) to be performed. It takes a `org.biojava.bio.dp.MarkovModel` object, which represents the Markov Model, and converts it to its own internal format. This conversion involves sorting all the states, which is what 53.9% of the whole program's execution time is spent on.

For our purposes, this conversion is unnecessary, as we can generate the states ourselves, and can easily do so in the order necessary to run Viterbi on them. The actual call to the `DP.viterbi()` method, which actually runs the Viterbi algorithm to determine the most likely sequence of states, takes only 5.0% of the total execution time.

2.2.2 Threads

An important aspect of the existing implementation is its threading model - i.e. how it distributes the workload between multiple threads of execution (which can then run on multiple CPUs or CPU cores). Since we already determined in Section 2.1.1 that it does indeed use multiple threads, we need to look more closely at exactly how it does so.

For this, we have our profiler record events in the `java.util.concurrent` package, which include thread creation, destruction and blocking (waiting for another thread to complete). The results of this, shown in 2.3 confirm our earlier hypothesis. The program has two threads active most of the time; one processing the gene, and one (main thread) waiting for it to complete, before starting the next one.

The fact that they use a threading library indicates that the original authors intended to make iHMMuneAlign a concurrent program, processing multiple gene sequences in parallel. While their architecture would have been less performance efficient than using a thread pool (as they have to pay the cost of thread creation and destruction for each

sequence), it would still have caused a notable improvement if they allowed the threads to execute in parallel.

Nonetheless, the fact that the main thread doesn't create each thread until its predecessor terminates indicates said efforts of parallelism were abandoned. It is widely regarded in industry that concurrent programming is rather difficult and the original authors most likely deemed completing the project was more prudent than making it concurrent.

No further investigation was done to determine what would need to be done in order to make the current implementation concurrent, as it will most likely be rewritten for this project.

Chapter 3

Proposed Solution

3.1 Re-implementing From Scratch

Based on the above analysis, it is clear that the original codebase has many pitfalls, and limitations that make it difficult to proceed further.

3.1.1 Code Maintainability

Most of the logic for processing a gene sequence is contained in the

```
AlignmentThread.startAlignment()
```

method, which over 2000 lines long. The project doesn't use any revision control systems, and the fact it is littered with commented out code most likely indicates that it didn't do so at any point in the past either.

There are no unit tests that cover most of the program logic, and it would be difficult to write any, since the logic hasn't been split up. This makes it rather difficult to make changes, without introducing any new subtle bugs, or undesired behaviour.

The original authors are no longer available to provide assistance for understanding their codebase, and the style guide they followed (or lack thereof) makes it difficult to read and follow.

It is understandable that the original authors did not prioritize long term code maintainability, as they could have been rushed for time, or they may have deemed it unlikely their could would be used and maintained 7 years later.

3.1.2 Concurrency / Thread Safety

Concurrency must be considered from the very early design decisions onwards, for a program to be efficient and parallel. The fact the original authors abandoned their efforts for parallelism indicates that there may have been concurrency bugs, like race conditions [17] that are very difficult to trace.

Access to shared data must be carefully controlled, and synchronized to avoid data corruption (and race conditions). The potentially high overhead of synchronization should lead to program structure designs where shared data between threads is minimal. In addition, the thread safety of any libraries used must be carefully considered - if a class is not thread safe, it shouldn't be shared between threads; and if a class is not re-entrant, there shouldn't even be separate instances in different threads.

3.1.3 Libraries and Language

Some of the libraries that the original authors used have since been deprecated. The specific libraries are mentioned in more detail below. Considering that some libraries account for most of the performance inefficiencies of the program (as shown in Section 2.2.1), they will certainly need to be replaced. Since the libraries' APIs heavily influenced the structure of the code, replacing them while maintaining the existing code structure will be a difficult task.

There is also the language itself to consider: Java programs tend to be 2-3 times slower

than their equivalent C++ counterparts, and even more so in CPU bound tasks like this [3]. Reimplementing from scratch will allow us to do so in a different language. C++ is a good candidate - it is quite popular for constructing large scale, high performance applications [8]; it contains high concepts from high level languages, while still allowing for low level optimizations, including memory management.

3.1.4   Implementation Stages

The initial version of the new implementation will be loosely based off the already existing implementation. Once the code is rewritten and structured in a manner that lends itself to easy modification, the real optimization (such as threading, and caching computation) can begin.

3.2   Dependencies of iHMMuneAlign

Apart from the codebase, the current implementation has several dependencies that perform significant amounts of computation. This section outlines how the new implementation will replicate their features.

3.2.1   BioJava

BioJava [11]is a general purpose Bioinformatics library, that contains several useful features for iHMMuneAlign. However, since iHMMuneAlign was written, BioJava has undergone a complete restructuring [20], and most -if not all- the modules the original implementation uses have been deprecated.

Fasta Reader

FASTA was originally the format used by a program that bore its name to store databases of DNA sequences [19]. These days, it is commonly used in Bioinformat-

ics to represent large quantities of gene sequences within a file. BioJava could parse files FASTA format and convert them to its in-memory representation, which were then used by iHMMuneAlign. Since we no longer intend to use BioJava, an alternate method was required to parse the inputs and repertoire files for each genes was required. Since the FASTA format is a very simple format, a custom parser could be built quite easily. In addition, the NCBI C++ Toolkit can also be used to trivially parse FASTA files.

Bioinformatic Data Structures

BioJava contains a large collection of pre-built data structures to represent commonly occurring biological objects and concepts (E.g. RichSequenece, FiniteAlphabet). Here we have several viable solutions.

Bio++   is a set of general purpose Bioinformatics libraries written in C++ [5]. Much like BioJava, it also provides various pre built data structures that represent various biological entities, like gene sequences. However, how well they correspond to the used data structures from BioJava and how useful they are when re-implementing is yet to be seen.

Qt   is a C++ framework, originally created by Trolltech and now maintained by Digia [29]. It has various components that assist in the rapid building of user interfaces, as well complex systems and algorithms. While they are less specific to biology, they are designed for speed and efficiency. Once the initial version is built using Bio++, if Bio++'s data structures are deemed a significant performance bottleneck, new ones will be custom built using Qt's support structure.

STL   The Standard Template Library (STL) that accompanies C++ also contains a series of very high performance, generic data structures and algorithms [16]. However, it is widely considered to be less user friendly than its alternatives, like Qt. For this

reason, Qt's data structures were selected as the primary candidate for a custom implementation, and if they cause a significant performance limitation, they will be replaced by their STL counterparts.

Boost   is another, highly comprehensive set of libraries that extend the functionality of the STL [4]. Select features from Boost often end up in subsequent revisions of the STL. Should functionality of the STL be insufficient to replace the Qt implementation once it's been deemed a performance bottleneck, Boost will be used to compensate.

3.2.2   DP Module

We have determined in Section 2.2.1 that the DP module (in the `org.biojava.bio.dp` package) accounts for more than two thirds of the total runtime, with half on a data conversion that is ultimately unnecessary.

Since the iHMMuneAlign was written, BioJava has undergone a major overhaul [20] and the entire module is now deprecated. For replacing the parts of its functionality that could have been useful moving forward (namely, the implementation of Viterbi algorithm), we have two choices:

3.2.3   External Library

Considering Hidden Markov Models are a very popular for a variety of applications [6], there are several C++ implementations available for use. For this, two of them were evaluated

MLPack   is a scalable C++ machine learning library [2], by the Georgia Institute of Technology. It has a module to deal with Hidden Markov Models, which includes an implementation of Viterbi. However, the implementation entertains the possibility of

every state transitioning to every other state, and as such performed too slow in our tests.

StochHMM   is a library and set of tools dealing with Hidden Markov Models, by the University of California [15]. Its evaluation was done by feeding in the model as a text file to its Viterbi solver, which crashed. The cause of the crash was not determined, as we moved on to other solutions.

3.2.4   Custom Implementation

After the failures of the other approaches, the possibility of implementing the algorithm ourselves was evaluated. A proof of concept implementation was created, tested with a roughly equivalent workload, and showed an order of magnitude improvement over the original solver from BioJava.

Later on in the project, specifically when the model generation code is implemented, its performance can be evaluated more accurately. The final decision regarding its use will be made at the time, depending on the performance of the alternatives.

3.3   Concurrency

Since the advent of multi-core and multi-processor machines, concurrency has become essential for high performance applications [24]. Since clock speeds have stagnated, yet the parallelism available in modern hardware continues to rise designing our implementation to be concurrent from the start will ensure that it can take advantage of the speed improvements granted by future generations processors.

The nature of our problem presents us two obvious ways of splitting it up to distribute over multiple threads of execution.

3.3.1 Per Sequence

The fundamental idea is to have each processing a different input gene sequence at the same time. This is the approach the authors of the original implementation attempted, although their implementation created a new thread for each sequence.

Creating a thread requires at least one system call (which is very expensive [26]) and destroying requires another. Additionally, the operating system has to do more book-keeping, as well as preparing a bunch of extra resources for the thread, which then have to be cleaned up - causing a significant performance cost [27].

A much better performing design would be to create a pool of worker threads at the start, and a queue of sequences for them to process. Each thread will then pop the sequence at the top of the queue, process it, write out the result, pop the sequence that is now at the head of the queue and repeat. In this case, the primary limitation for parallelism would come from the queue itself - by virtue of being a shared data structure, it would require synchronization to maintain consistency [9].

If the synchronization overhead of the queue is found to be a significant limitation on parallelism, it can be skipped altogether. Since all the inputs sequences are passed to the program in a file, the file itself can be partitioned - then each thread would read, parse and process all the sequences within a partition.

Doing so has the advantage that it will reduce the synchronization necessary between threads. On the other hand, if the sequences in some partitions require significantly less work than others, some threads would finish much quicker than others and be idling instead of doing useful work. This can be mitigated somewhat, by using a technique called work stealing [10], where threads that have finished all their jobs take pending jobs from other threads.

3.3.2   Per Processing Stage

The other way, is to consider each sequence as a series of tasks, and perform each one in a different thread. It's infeasible to speed up the processing of an individual sequence using this method, as most (if not all) of these tasks would depend on the results of the previous, and initializing a thread could take longer than some of the tasks.

Considering that the goal of the project is to scale well with increasing numbers of sequences, the following approach makes more sense; tasks can be done in batches. I.e. have n threads process the first stage of $n \times k$ sequences (where k is an integer constant), and then have all n threads process the second stage and so on. This has the primary advantage of having much greater data locality (decreasing the working set, page faults, cache misses) and increasing performance significantly [25].

Per contra, this approach will be heavily reliant on multiple work queues to distribute the tasks of different sequences between threads. As explained in the above section, the synchronization overhead of the queues may become a serious bottleneck on parallelism - even more so, due to the much heavier reliance on it.

Chapter 4

Design & Implementation

4.1   Initialization

Since iHMMuneAlign is rather configurable, it requires a large amount of input data for each run that cannot be embedded in the code directly. This data has to be read during program initialisation and stored in memory such that it can be efficiently used in their required of the algorithm.

4.1.1   D/J Repertoires

4.1.2   V Repertoire

4.1.3   Mutation Ratios

4.2   Pipeline

The entire process of creating a HMM to represent immunoglobulin recombination and solving it using Viterbi's algorithm can be split up into several smaller problems. These

Figure 4.1: Overview of the Model Solving 'Pipeline'

Input Sequence   BLAST Hit

①

A Score

②

States   Transitions

③

Most Likely Path

④

Output
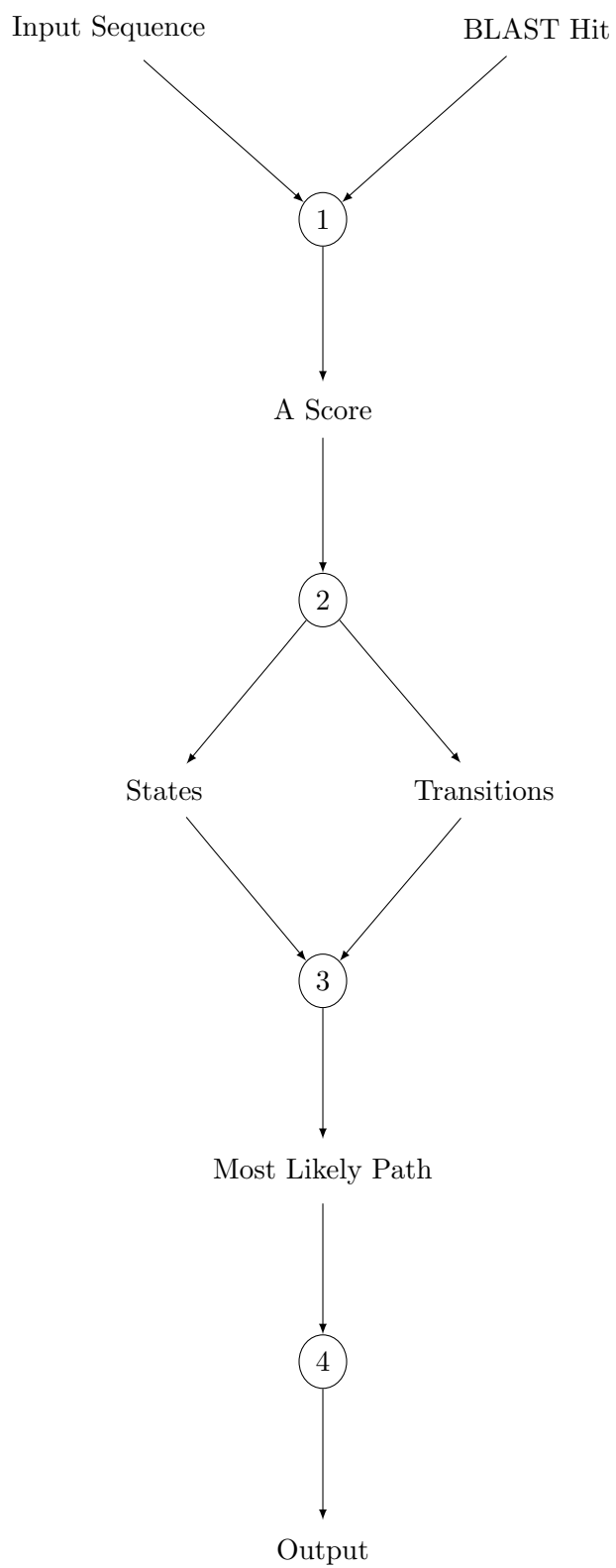
can be viewed as stages in an assembly line where the result of each one feeds into the next, terminating with the final result. In computing, this is called a pipeline.

4.2.1 BLAST

The first step is to determine the most likely germline V gene is in the target sequence. Fortunately, this can be determined quite efficiently by aligning the input sequence with all known V genes [7]. For that we opted to use the excellent alignment tool by NCBI: BLAST [1], like the original implementation.

BLAST is distributed as source code, and can be used in two ways:

Invoking Binary

There are binaries for all the common tasks, that take in sequence repertoires as files in FASTA [19] format. The previous implementation used the `blastn` executable, which takes in a sequence in FASTA format, and writes the results (matching sequences in the repertoires) out to an XML file.

The primary advantage of using blast in this manner would be the ease of development. Since the binaries come pre-packaged, very little extra code would be required to invoke the binary from within iHMMuneAlign, and parse the XML output. However, doing so incurs a (potentially significant) performance penalty at run time due to the cost of

Process Creation The operating system has to create a new process for the blast executable, perform all its bookkeeping operations, and clean up after it is completed.

Reading Input Files The executable would then have to read the database files from disk again, instead of using the copy that already in memory (residing in iHMMuneAlign's address space).

Serialising Results Once the blast executable completes its alignment search, it will then need to convert that data into a human readable, or standardised format that can be parsed by iHMMuneAlign.

Transferring Results disk, ipc

Linking Shared Library

BLAST+ can be

4.2.2 Parsing Blast Results

4.2.3 Calculating A-Score

4.2.4 Generating Model

Generating V/D/J States

Generating Transitions

4.2.5 Running Viterbi

4.2.6 Printing Results

4.3 Approach

Talk about agile here.

Chapter 5

Maintenance

Talk about how I attempted to improve maintainability/readability here

5.1   Revision Control

Explain how revision control will help future maintenance

Chapter 6

Performance

Talk about how I set off to improve performance here

6.1   Blast Lib

Chapter 7

Evaluation

This chapter is mainly provided for the purpose of showing a typical thesis structure. There are no more thesis requirements described.

7.1   Results

The result of this work is the present document, being both a LaTeX template and a thesis requirement specification.

7.2   Discussion

The Dual function of this document somewhat de-emphasises the primary purpose of the document, namely the thesis requirements. It would be better, if these could be stated on a few concise pages (cf Appendix 1, p32).

Chapter 8

Conclusion

A thesis requirements/template document has been created. This serves the dual purposes of giving students specific requirements to their theses — both style and content related — while providing a typical thesis structure in a LaTeX template.

8.1 Future Work

Extract the requirements from the template in order to have very concise requirements.

Bibliography

[1]   Stephen F. Altschul et al. "Basic local alignment search tool". In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410. ISSN: 0022-2836. DOI: 10.1016/S0022-2836(05)80360-2. URL: http://www.sciencedirect.com/science/article/pii/S0022283605803602.

[2]   Ryan R Curtin et al. "MLPACK: A scalable C++ machine learning library". In: *The Journal of Machine Learning Research* 14.1 (2013), pp. 801–805. URL: http://dl.acm.org/citation.cfm?id=2502606;%20http://www.mlpack.org/mlpack_jmlr.pdf.

[3]   Matthias Kalle Dalheimer. "A comparison of qt and java for large-scale, industrial-strength gui development". In: *Klarlvdalens Datakonsult AB, Tech. Rep* (2005).

[4]   Beman Dawes, David Abrahams, and Rene Rivera. "Boost C++ libraries". In: *URL http://www. boost. org* 35 (2009), p. 36.

[5]   Julien Dutheil et al. "Bio++: a set of C++ libraries for sequence analysis, phylogenetics, molecular evolution and population genetics". In: *BMC bioinformatics* 7.1 (2006), p. 188.

[6]   Robert J Elliott, Lakhdar Aggoun, and John B Moore. *Hidden Markov Models.* Springer, 1994.

[7]   Bruno A. Gaëta et al. "iHMMune-align: hidden Markov model-based alignment and identification of germline genes in rearranged immunoglobulin gene sequences". In: *Bioinformatics* 23.13 (2007), pp. 1580–1587. DOI: 10.1093/bioinformatics/btm147. eprint: http://bioinformatics.oxfordjournals.org/content/23/13/1580.full.pdf+html. URL: http://bioinformatics.oxfordjournals.org/content/23/13/1580.abstract.

[8]   Lois Goldthwaite. "Technical report on C++ performance". In: *ISO/IEC PDTR* 18015 (2006).

[9]   Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint.* Elsevier, 2012. ISBN: 978-0-12-370591-4. URL: http://books.google.com.au/books?hl=en&lr=&id=vfvPrSz7R7QC&oi=fnd&pg=PP2&dq=art+of+multiprocessor+programming+Herlihy,+Maurice&ots=e_fBsPVMeP&sig=D6u_nmnRdj6Lu5Ojg29_j8O_PtI.

[10]   Maurice Herlihy and Nir Shavit. "Work Distribution". In: *The Art of Multipro-cessor Programming, Revised Reprint.* Elsevier, 2012. Chap. 16.4. ISBN: 978-0-12-370591-4. URL: `http://books.google.com.au/books?hl=en&lr=&id=vfvPrSz7R7QC&oi=fnd&pg=PP2&dq=art+of+multiprocessor+programming+Herlihy,+Maurice&ots=e_fBsPVMeP&sig=D6u_nmnRdj6Lu5Ojg29_j8O_PtI.`

[11]   R. C. G. Holland et al. "BioJava: an open-source framework for bioinformatics". In: *Bioinformatics* 24.18 (2008), pp. 2096–2097. DOI: `10.1093/bioinformatics/btn397`. eprint: `http://bioinformatics.oxfordjournals.org/content/24/18/2096.full.pdf+html`. URL: `http://bioinformatics.oxfordjournals.org/content/24/18/2096.abstract.`

[12]   CEH Lee et al. "Reconsidering the human immunoglobulin heavy-chain locus". In: *Immunogenetics* 57.12 (2006), pp. 917–925. URL: `http://link.springer.com/article/10.1007/s00251-005-0062-5.`

[13]   Marie-Paule Lefranc et al. "IMGT, the international ImMunoGeneTics informa-tion system®". In: *Nucleic Acids Research* 33.suppl 1 (2005), pp. D593–D597. URL: `http://nar.oxfordjournals.org/content/33/suppl_1/D593.short;` `%20http://nar.oxfordjournals.org/content/33/suppl_1/D593.long.`

[14]   Honghua Li et al. "Genetic diversity of the human immunoglobulin heavy chain VH region". In: *Immunological reviews* 190.1 (2002), pp. 53–68. URL: `http://onlinelibrary.wiley.com/doi/10.1034/j.1600-065X.2002.19005.x/full.`

[15]   Paul C Lott and Ian Korf. "StochHMM: a flexible hidden Markov model tool and C++ library". In: *Bioinformatics* (2014), btu057. URL: `http://bioinformatics.oxfordjournals.org/content/early/2014/01/30/bioinformatics.btu057.short.`

[16]   David R. Musser, Gillmer J. Derge, and Atul Saini. *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library.* 3rd. Addison-Wesley Professional, 2009. ISBN: 0321702123, 9780321702128.

[17]   Robert H. B. Netzer and Barton P. Miller. "What Are Race Conditions?: Some Issues and Formalizations". In: *ACM Lett. Program. Lang. Syst.* 1.1 (Mar. 1992), pp. 74–88. ISSN: 1057-4514. DOI: `10.1145/130616.130623`. URL: `http://doi.acm.org/10.1145/130616.130623.`

[18]   Oracle. *Java Native Interface Specification.* 2014. URL: `http://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/jniTOC.html.`

[19]   WilliamR. Pearson. "Using the FASTA Program to Search Protein and DNA Sequence Databases". English. In: *Computer Analysis of Sequence Data.* Ed. by AnnetteM. Griffin and HughG. Griffin. Vol. 24. Methods in Molecular Biology. Humana Press, 1994, pp. 307–331. ISBN: 978-0-89603-246-0. DOI: `10.1385/0-89603-246-9:307`. URL: `http://dx.doi.org/10.1385/0-89603-246-9:307.`

[20]   Andreas Prlić et al. "BioJava: an open-source framework for bioinformatics in 2012". In: *Bioinformatics* 28.20 (2012), pp. 2693–2695. DOI: `10.1093/bioinformatics/bts494`. eprint: `http://bioinformatics.oxfordjournals.org/content/28/20/2693.full.pdf+html`. URL: `http://bioinformatics.oxfordjournals.org/content/28/20/2693.abstract.`

[21]   Jeffrey V Ravetch et al. "Structure of the human immunoglobulin μ locus: characterization of embryonic and rearranged J and D genes". In: *Cell* 27.3 (1981), pp. 583–591. URL: http://www.sciencedirect.com/science/article/pii/0092867481904001.

[22]   Jack Shirazi. "Tool Report: JProfiler". In: *Java Performance Tuning, Jun* (2002).

[23]   T. Suganuma et al. "Overview of the IBM Java Just-in-Time Compiler". In: *IBM Systems Journal* 39.1 (2000), pp. 175–193. ISSN: 0018-8670. DOI: 10.1147/sj.391.0175.

[24]   Herb Sutter. "The free lunch is over: A fundamental turn toward concurrency in software". In: *Dr. Dobb's journal* 30.3 (2005), pp. 202–210.

[25]   Andrew S. Tanenbaum and Herbert Bos. *Modern Operating Systems*. 4th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. Chap. 16.4.6, 3. ISBN: 013359162X, 9780133591620.

[26]   Andrew S. Tanenbaum and Herbert Bos. "System Calls". In: *Modern Operating Systems*. 4th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. Chap. 1.6. ISBN: 013359162X, 9780133591620.

[27]   Andrew S. Tanenbaum and Herbert Bos. "Threads". In: *Modern Operating Systems*. 4th. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. Chap. 2.2. ISBN: 013359162X, 9780133591620.

[28]   Terencehonles. *English: Hidden Markov Model*. Nov. 8, 2009. URL: http://commons.wikimedia.org/wiki/File:HMMGraph.svg (visited on 10/21/2014).

[29]   Qt Trolltech. *4.1 Whitepaper*.

Appendix 1

This section contains the options for the UNSW thesis class; and layout specifications used by this thesis.

A.1   Options

The standard thesis class options provided are:

undergrad	default
hdr	
11pt	default
12pt	
oneside	default for HDR theses
twoside	default for undergraduate theses
draft	(prints DRAFT on title page and in footer and omits pictures)
final	default
doublespacing	default
singlespacing	(only for use while drafting)

A.2   Margins

The standard margins for theses in Engineering are as follows:

	U'grad	HDR
\oddsidemargin	40 mm	40 mm
\evensidemargin	25 mm	20 mm
\topmargin	25 mm	30 mm
\headheight	40 mm	40 mm
\headsep	40 mm	40 mm
\footskip	15 mm	15 mm
\botmargin	20 mm	20 mm

A.3   Page Headers

A.3.1   Undergraduate Theses

For undergraduate theses, the page header for odd numbers pages in the body of the document is:

Author's Name	*The title of the thesis*

and on even pages is:

The title of the thesis	Author's Name

These headers are printed on all mainmatter and backmatter pages, including the first page of chapters or appendices.

A.3.2   Higher Degree Research Theses

For postgraduate theses, the page header for the body of the document is:

The title of the chapter or appendix

This header is printed on all mainmatter and backmatter pages, except for the first page of chapters or appendices.

A.4   Page Footers

For all theses, the page footer consists of a centred page number. In the frontmatter, the page number is in roman numerals. In the mainmatter and backmatter sections, the page number is in arabic numerals. Page numbers restart from 1 at the start of the mainmatter section.

If the **draft** document option has been selected, then a "Draft" message is also inserted into the footer, as in:

	14	**Draft:** May 31, 2015

or, on even numbered pages in two-sided mode:

Draft: May 31, 2015	14	

A.5   Double Spacing

Double spacing (actualy 1.5 spacing) is used for the mainmatter section, except for footnotes and the text for figures and table.

Single spacing is used in the frontmatter and backmatter sections.

If it is necessary to switch between single-spacing and double-spacing, the commands `\ssp` and `\dsp` can be used; or there is a `sspacing` environment to invoke single spacing and a `spacing` environment to invoke double spacing if double spacing is used for the document (otherwise it leaves it in single spacing). Note that switching to single spacing should only be done within the spirit of this thesis class, otherwise it may breach UNSW thesis format guidelines.

A.6   Files

This description and sample of the UNSW Thesis LaTeX class consists of a number of files:

unswthesis.cls	the thesis class file itself
crest.pdf	the UNSW coat of arms, used by `pdflatex`
crest.eps	the UNSW coat of arms, used by `latex` + `dvips`
dissertation-sheet.tex	formal information required by HDR theses
pubs.bib	reference details for use in the bibliography
sample-thesis.tex	the main file for the thesis

The file sample-thesis.tex is the main file for the current document (in use, its name should be changed to something more meaningful). It presents the structure of the thesis, then includes a number of separate files for the various content sections. While including separate files is not essential (it could all be in one file), using multiple files is useful for organising complex work.

This sample thesis is typical of many theses; however, new authors should consult with their supervisors and exercise judgement.

The included files used by this sample thesis are:

definitions.tex	mywork.tex
abstract.tex	evaluation.tex
acknowledgements.tex	conclusion.tex
abbreviations.tex	appendix1.tex
introduction.tex	appendix2.tex
background.tex	

These are typical; however the concepts and names (and obviously content) of the files making up the matter of the thesis will differ between theses.

Appendix 2

This section contains scads of supplimentary data.

B.1 Data

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.

Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data. Heaps and heaps and heaps and heaps and heaps and heaps of data.