Transcription Factor Footprint Analysis

Sal Wolffs (s4064542)

April 4, 2017

Contents

Chapter 1

Acknowledgements

I would like to thank my thesis supervisors, Wieb Bosma of the Radboud University FNWI department for mathematics, and Colin Logie of the RIMLS FNW-Molecular biology department, for their advice, expertise, feedback, and general guidance throughout the writing of this thesis.

I would like to thank Sadia Saeed, for her stellar work at Colin Logie's group producing the dataset used in this thesis, and for very generously sharing said dataset. Since this dataset is based on the analysis of white blood cells, I would also like to thank the anonymous blood donors who consented for their blood to be analysed for this purpose.

I would like to thank Kees Albers and Wout Megchelenbrink, for sharing their expertise and providing valuable feedback.

I would also like to thank the various people I have discussed this thesis with in passing, many of them at the FNWI department for computer science, most of whose names I've never even known, but whose perspectives have proven valuable in writing this thesis.

Finally, I would like to thank the Genome Reference Consortium, for publicly and freely providing a human reference genome, which, while less personal, has also been indispensable in writing this thesis.

Chapter 2

Introduction

In this thesis, we will be looking at the mathematical formalisation of a biological problem concerning DNA regulation, and attempt a solution. As this is a multidisciplinary thesis, we will pay rather more attention than is customary to basic background of the subjects involved, both in the main text and in appendices.

2.1 Structure of this text

We will first formulate our research question in terms which are intuitive for biologists. Before trying to solve the problem, we will establish some biological background and characterize the content and format of our dataset. Based on this, we will construct a suitable model, and reformulate the research question in terms of the formalism of our model, accessible for data scientists. The answer to the formal question should translate trivially to an answer to the original, biological phrasing of the question. Finally, we will describe an attempted solution, and discuss the results.

Chapter 3

Research Problem for Biologists

The goal of this thesis is as follows: Write a program that, given a dataset such as the one we have, produces a list of likely common motifs explaining the dataset, with the implication that these motifs should match the motifs recognized by the actual transcription factors active in the cell from which the dataset arises.

Chapter 4

Biological background

This section will only cover common knowledge:

Biologists and those having completed any introductory course in DNA biochemistry should be able to skip ahead to the subsection Transcription factors (and likely skip most of that).

Molecular biologists should skip ahead to [Dataset], the next major section.

4.1 Caveat

When a mathematician says "for all" or "there is no" ("not exists"), they mean it, much more so than is usual in common parlance. Biologists, on the other hand, do not; they mean it as much as anyone describing the real world means it. We will follow this latter convention in this chapter, except where clearly purely mathematical statements are concerned.

4.2 Prior knowledge

We will assume the reader is familiar with biology up to the existence of cells, DNA, chromosomes, genes and proteins. No intimate familiarity with their properties is required; such as is needed will be noted here.

4.2.1 Cells

Cells form more or less self-contained units: they can be isolated for study without completely disrupting their behaviour (though some disruption will occur). Our model will cover a part of the processes in one cell, averaged over all cells in a sample.

4.2.2 DNA

4.2.2.1 Nucleotides

DNA is a bioheteropolymer composed of nucleotides. There are 4 nucleotides used in the construction of DNA: dAMP, dCMP, dGMP and dTMP, generally referred to as A, C, T, G. A strand of DNA is

generally represented by stringing the letters of the involved nucleotides together in order: ACATGG refers to a single strand of DNA containing every nucleotide once, and A and G twice.

4.2.2.2 Directionality

Nucleotides are asymmetrical, having distinct ends called 5' and 3' (referring to the chemical numbering of those positions in the molecule), and polymerization matches 5' to 3' ends, giving DNA a persistent direction which is thought of as from 5' to 3', in which direction sequences are always reported. So, ACTG is a strand with A at the 5' end, and different from GTCA.

4.2.2.3 Complements

Each nucleotide has a complement: A complements T, C complements G. Besides polymerizing "front" to "back", nucleotides can weakly connect "side" to "side", but only to their complement. This also gives rise to complements of sequences, being the pointwise complements of their nucleotides, with one catch: the complement has reversed orientation. Illustration: ACTG is complemented by CAGT (not TGAC):

```
5'-A-C-T-G-3'
   : : : :
3'-T-G-A-C-5'
```

In nature, all DNA strands are almost always paired up in complements, forming "double stranded" DNA.

4.2.2.4 Pyramidines and Purines

Nucleotides vary in structure and size. A and G are "purines", similar in structure and size and larger than the "pyramidines" T and C, similarly similar to each other. This is relevant to note, since some of the processes we will want to model might e.g. only react to a rising edge (pyramidine to purine) or falling edge in a sequence, rather than to any specific nucleotide.

4.2.3 Chromosomes

Human DNA is ordered into chromosomes, 23 pairs of extremely long double DNA strands. Each cell has a copy, and distributed over these chromosomes are all human genes (excepting mitochondrial DNA which we won't get into).

4.2.4 Genes

Genes are sequences, located on chromosomes, which encode the sequence of a specific protein. The exact nature of the encoding is not relevant here, but the 1-on-1 nature is (although some genes apply tricks to encode multiple related proteins): one gene being "active" corresponds to one protein being mass-produced.

4.2.5 Proteins

Proteins are a different kind of bioheteropolymer, forming the workhorses of living cells. The sequence of a protein (along with some post-processing) gives rise to its functionality.

4.3 Transcription factors

Since every cell contains all genes, but no cell needs every gene, it needs to be possible to toggle genes on and off. This is accomplished by means of *Transcription Factors*, proteins which bind DNA and signal "somewhere relatively near this place is a gene which should/shouldn't be active".

These transcription factors attach to DNA in the right locations by recognizing (binding) specific sequences. However, they do not bind one specific sequence. Rather, they bind any sequence which conforms to specific properties, generally being "close enough" to some ideal recognized sequence and identical in the most important positions.

The characterization of these families of recognized sequences, based on the dataset described below, is the focus of this thesis.

Chapter 5

Input Data

The data we will be using have been acquired using a rather blunt approach: Rather than trying to track the behaviour of each individual species of TF, a near impossible task, the behaviour of all TFs in a living cell is analysed at once, using the procedure sketched below.

5.1 Process

First, chemically "freeze" a cell with formaldehyde, instantly killing it but preserving various aspects of its state at the moment of death, among them what DNA is bound by what proteins. However, preserving state and observing it are not the same; the best known approach to the latter is then using DNAseI to cut all unbound DNA into small pieces, then reversing the freezing process and seeing what parts remain uncut. Each such uncut part represents a "footprint" of a TF bound at the moment of death.

5.2 Limitations

Unfortunately, this does not reveal anything about what protein did the binding: all it reveals is what DNA was bound. This means protein-specific information is lost, and any analysis, including this one, can only draw conclusions "up to protein equivalence", which does not yield protein identities.

5.3 Caveat: Noise

More unfortunately, during the writing of this thesis, a source of noise was discovered: DNAseI does not cut all DNA sequences equally. This means that some sequences, while unbound, will still remain uncut due to being ignored by DNAseI, giving the false impression of being bound. It has been decided to accept this noise, since no dataset free of it is available or likely to become available.

5.4 Noise mitigation

In various places, treshold values are used in attempt to limit the effect of this noise. While the noise might surpass this treshold in some instances, it is hoped those instances will form clusters distinct from legitimate clusters, possibly recognisably so, but at worst yielding only false clusters, rather than fouling legitimate clusters. This hope is founded in some biological properties of false and true positives, which won't be discussed further here. (* Possibly create an appendix B and refer to it here *)

5.5 Data format

The data has been supplied as a single file per dataset, with on each line one footprint, formatted as such: `[chromosome ID] [start index] [end index]` We cross-reference this with publicly available reference genomes, formatted according to the fasta standard, to produce the associated sequence for each footprint.

5.6 Acknowledgement

The dataset we have used has been produced by the stellar work of (*Insert credits for dataset here*), who has/have our deep gratitude and without whose generosity in sharing their test results this thesis would have been impossible.

Chapter 6

Model

6.1 DNA sequences

We will represent DNA sequences the usual way, noting only one side of a double strand.

Because the interactions we are interested in involve double strands, we will identify each sequence with its complement (since the double strand they form is the same). This is non-trivial in implementation, and so should be explicitly addressed in implementing any sub-algorithm which considers sequences directly.

The distinction between pyramidines and purines, while interesting, has not been modeled due to time constraints and difficulty using this information in algorithms not built around it. This is an area for future work.

6.2 Transcription factors

Since we do not have any data about the individual transcription factors themselves, we will identify only the families of recognized sequences. To do so, some notation for such a family is needed.

6.2.1 Motifs

One way to denote such a family is a motif: a sequence of probabilities of each nucleotide appearing. For mathematicians and programmers:

$$\text{Sequence} : \{\mathrm{A}, \mathrm{C}, \mathrm{T}, \mathrm{G}\}^*$$

$$\text{Motif} : \left([0,1]^4\right)^*$$

$$t : \{\mathrm{A}, \mathrm{C}, \mathrm{T}, \mathrm{G}\}[0,1], \mathrm{A}(1,0,0,0), \mathrm{C}(0,1,0,0), \text{etc.}$$

$$i : \text{SequenceMotif}, ()(), cwt(c)i(w)$$

This is the notation preferred by biologists, but it strips some information (like implications "A in position 2 requires that C be in location 5"), and implies more knowledge about the family than I expect can be gained from this dataset.

6.2.2 Wildcard patterns

A simpler, cruder representation can be gained by replacing the $[0, 1]$ in the motifs above with $\{0, 1\}$, yielding values of "may appear" and "can't appear". Since this representation conveys less information, it is easier to construct from indirect observation, and will be the goal of initial efforts.

6.2.3 Advanced Motifs

Noting implications such as suggested for motifs might yield very interesting results, but is a subject for future research.

Chapter 7

Formalised Research Problem

Given the dataset and model as above, find Wildcard Patterns such that, with high likelyhood, each Transcription Factor active in the cells giving rise to the dataset matches every sequence within one of those Wildcard Patterns, and none outside it. Additionally, no spurious wildcard patterns must be introduced: each must correspond to at least one TF. So, there must be a surjective function f from TFs to Wildcard Patterns, and with

matches : Wildcard$P$(Sequence)

recognizes : TFP(Sequence)

with the obvious interpretations, we want matches$f$ = recognizes.

Note: f should exist, but doesn't have to be known (nor could it be found in this dataset): The set of wildcards produced should merely be such that it *can* exist.

7.1 Minimal spanning tree-based

7.1.1 Justification

One approach is to use the fact that sequences within a motif can't be too different; therefore, all sequences matching a motif must fall within a fairly close edit distance of each other. Conversely, sequences with small edit distances to each other are likely to belong to the same motif. Since MSTs form a grouped measure of closeness, they can be used to search for a cluster.

7.1.2 Outline

The algorithm is rather straightforward: Convert all footprints to sequences, yielding a multiset. Take the weighted complete graph on these sequences, with weights equal to the distance between sequences. Construct the MST and remove the longest edges from the MST, yielding a forest. Each connected component is now expected to correspond to a motif, which can be found or approximated as the "join" of the sequences in the component.

7.1.3 Tuning factors

Decisions which need to be made in this algorithm are:

- What metric to apply to the sequences.
- How to decide how many edges to take away. This can be some constant, or based runtime on the dataset.
- What join to use.

7.1.4 Strengths

- Easy to implement
- Low computational complexity

7.1.5 Weaknesses

- Likely bad resolution
- Cannot tell motifs apart whose matching sequences are too close together. Actual overlap cannot be compensated for by any amount of tuning.

7.1.6 Detailed description

7.1.6.1 Reading the data

7.1.6.1.1 Data format

The data consists of a list of all locations of footprints, in the format

`<chromosome name> <start index> <end index>`

as well as, for each chromosome name, a file containing the sequence of that chromosome in FASTA-format, named `<chromosome name>.fa`

7.1.6.1.2 Reading Algorithm

Since the human genome is rather large (on the order of 3GB), we don't want to open all files at once. Therefore, we first parse the locations file into a mapping from chromosomes to lists of locations on that chromosome. Then, for each entry in this mapping, we open the corresponding file, look up every footprint on that chromosome, and note their sequence together with their location, before closing the file and moving on to the next chromosome. This yields a list of all footprints with both location and sequence known.

We now have a `raw_dataset` $= \{(\text{lookup}(l), l) \mid l$`locationfile`$\}$Sequences»Locations where

Sequences = Nucleotides*

Locations = Strings»»

Strings = AlphaNumeric*

`locationfile` = (the locations of all footprints)Locations

lookup : LocationsSequences, locationsequence at that location in the chromosome files

7.1.6.2 Taking the multiset

This particular algorithm doesn't care about location (no useful metric considers it), but some possible refinements consider how often a particular sequence occurs (for cutoff criteria, or metric tweaking). Therefore, count occurences and discard locations, yielding a new datasetSequences»

7.1.6.3 Minimal Spanning Tree Construction

The construction of a Minimal Spanning Tree is done by Prim's Algorithm, a standard memoized greedy algorithm which takes a set of vertices V and a weight function $w : V»VW$ on these vertices. By setting

VSequences»

$w = d$ for some metric $d : V»V$

we can use Prim for clustering in a metric space. NB: while metrization is possible, V does not have any obvious concept of averages (it can, however, be embedded in a join-semilattice, which we will use later)

The idea behind Prim's algorithm is to expand the tree iteratively, calculating at each step which node not in the tree is closest to the tree (this being the minimum distance to any node in the tree), then adding that node to the tree, recalculating minima, and going to the next iteration until all nodes are in the tree, at which point our tree is both spanning and minimal.

Note that in the first iteration, there is no tree yet. However, we can simply set any vertex r to be in the tree, giving us a valid 1-vertex tree.

Pseudocode for Prim:

```
Prim (V, w, r   V):
    let W : P(V) × V →  ,
        (A,v)   min({w(a,v) | a   A})
    let MINFROM : P(V) × V → V,
        (A,v)   a   A with w(a,v) = W(A,v)
    let MINTO : P(V) → V,
        A   v   V\A with W(A,v) = min({W(A,u) | u   V\A})
    ##Note: the choices in CLOSEST and MIN are finite
    ##Note: we will explicitly calculate both W and MINFROM as part of
    ##  the algorithm: their definition above is for clarity and to show
    ##  correctness. Since this turns W into a lookup operation (O(1)),
    ##  MINTO can be calculated in O(|V|)
    Let T[0] = {r}
    We have:
        W(T[0],v) = w(r,v)
        MINFROM (T[0],v) = r
    Let E[0] =
    Now inductively define:
        u = MINTO(T[n])
        E[n+1] = E[n]   {(MINFROM(u),u)}
        T[n+1] = T[n]   {u}
        and we have:
```

```
            W(T[n+1],v) = min(W(T[n],v),w(u,v))
            MINFROM(T[n+1],v) = MINFROM(T[n],v)      | W(T[n+1],v) = W(T[n],v)
                              = u                    | otherwise
    Now T[n-1] = V, and (V,E[n-1]) forms a Minimal Spanning Tree
```

Running `Prim(dataset,d,v  dataset)` yields `MST = (V,E)`.

7.1.6.3.1 A note on complexity and choice of algorithms

An alternative approach to constructing the MST would be to use Kruskal's algorithm, which does not construct a tree but simply keeps adding the shortest edges overall without forming cycles.

However, this performs slightly worse than linear in the amount of edges, while Prim's algorithm can be made quadratic in the amount of vertices. Since we're effectively operating on a complete graph, the amount of edges is the square of the amount of vertices, making Prim's algorithm strictly linear in the amount of edges.

Several other algorithms exist, but we aren't aware of any that do better than linear in the amount of edges.

7.1.6.4 Choosing the distance metric

FIXME: For now, we have set `d = hamming o` , so a simple hamming distance on the sequence, ignoring the count.

7.1.6.5 Culling the MST

The hypothesis now is that vertices which should be clustered will be close in the MST, and so cutting longest edges will not split any true clusters. Therefore, decide on some number N of edges to cut and remove the N longest edges from E to yield E , resulting in a forest of N+1 components.

7.1.6.6 Selecting the amount of edges to cut

FIXME: Currently, this is a constant, based on a reasonable guess at the amount of clusters in the data.

7.1.6.7 Collecting the MST

Now determine the connected components $\{V[n] = \{vV \mid \text{path from } v[n] \text{ to } v\}\}$ with $n\check{n}m < n, v[n]V[m]$

7.1.6.8 Taking the join of a connected component.

At this point, we want to produce a list of motifs to answer the biological question. To do this, we embed the sequence space Sequence (from chapter Model) into the wildcard space Wildcard $:= (0,1)^*$ by embedding $i : \mathrm{A}(1,0,0,0), \mathrm{C}(0,1,0,0), \mathrm{T}(0,0,1,0), \mathrm{G}(0,0,0,1)$. Now we define a join-semilattice on Wildcard, by defining the join of two motifs of the same length to be their pointwise coordinatewise disjunction:

$$\text{join} : \{0,1\} \times \{0,1\}\{0,1\}$$
$$((x,x,x,x),(y,y,y,y))(xy,xy,xy,xy)$$
$$\text{join} : (\{0,1\})^* \gg (\{0,1\})^* \ (\{0,1\})^*$$
$$(l,r) \begin{cases} () & \text{if } (l,r) = ((),()) \\ \text{join}(b,c)\,\text{join}(v,w) & \text{if } (l,r) = (bv,cw) \\ & \text{otherwise} \end{cases}$$

Having defined this semilattice, we can take the join of each cluster, translate the resulting may/may not appear-codes back into standard IUPAC DNA ambiguity codes, and deliver these to biologists as our best guess.

7.1.6.9 Formalism and types (WIP)

We'll first note some type definitions, then note the steps taken in pseudocode. Lines starting with `Let` introduce some concept not easily or concisely expressible in pseudocode, and follow mathematical conventions instead. Comments will be proceded by "Note:"

NB: We will ignore most details of the interpretation of this formalism. While systems for rigorous typing of IO actions do exist (IO monad, uniqueness typing), this is not the subject of this thesis. Instead, we'll leave this to the actual implementation, and introduce some dummy functions to mark where files are opened and closed.

```
Let Numeric, AlphaNumeric, * (Kleene Star), N have their usual meanings.

toNat : Numeric* → N,
    base 10 notation of n |→ n

Strings := AlphaNumeric*

Let Files be the set of files.

open : Strings → Files ,
       path |→ file in the location described by path

close : Files → {1}
     , file |→ 1
Note: This is a placeholder to denote the closing of a function.

Let Nucleotides be the set of nucleotides, representable by
{'a','c','t','g'}.

Sequences := (Nucleotides U {'N'})*

readFasta : Files → Sequences
```

Chapter 8

Results

[Appendix: Results] contains raw output of various runs of the program compiled from the current version of the source.

8.1 Reading the raw results

The first part of each run of output is diagnostic information. This has been introduced in the debugging phase, and gives some insight into the state of the program during execution, facilitating potential further development. Outside of development, it should be ignored.

Actual output starts, in each run, below the line "Press any key to continue." After this, every line contains one motif, formatted as

`<sequence> <#(data points in cluster)> <#(unique original sequences in cluster)>`

With sequences written using IUPAC ambiguity codes. The lists are sorted by the second column, with shorter sequences coming first in case of a tie.

All runs are performed with the same settings, except for the lower limit on the amount of identical sequence occurences in the input dataset required before a sequence gets included as a data point for the MST algorithm.

Runs are presented in nor particular order. Runs with lower limits below 5 are excluded from the print version, due to such settings creating excessive amounts of output. Digital plain text files for runs with lower limits 3 and 1 are available.

8.2 Summary

While the output of the analysis program itself is too large to include here, a summary is possible:

- The vast majority of the data is absorbed by N^* sequences of various lengths, which accept everything.
- For every lower bound, the data has a tail of sequences based on a single identical sequence.
- There are several examples of sequences which look very similar, but are of different length.

Chapter 9

Discussion and reflection

9.1 Main conclusion

Based on the summary given above, the main result is negative: No useful clusters can be found in the majority of the dataset by this algorithm. From this, we can conclude that a simple hamming metric is probably too coarse. The lowest non-zero distance it can create is 1, and this has been taken as the maximum distance to relate two otherwise unrelated sequences. No stricter criterium is possible, yet it is not strict enough to not absorb the vast majority of the data into a single cluster per sequence length.

What this says about the dataset is that, using this metric, we can walk within sequences in the dataset from almost any point to almost any other point without ever taking steps longer than 1. Intuitively, this seems to indicate that if any real clusters exist, those clusters all touch, which gives a rather "fuzzy" view.

Note that this is a property of the dataset equipped with this metric: Other metrics might produce a clearer view, and indeed this is likely if they have more possible values and are chosen sensibly.

The problem, however, might very well be the MST algorithm itself: it cannot distinguish overlapping clusters at all, and such overlap is not impossible or even unlikely given the subject matter. If clusters overlap, a more sophisticated algorithm will be required, likely designed specifically with this problem in mind, rather than this most generic of clustering algorithms.

9.2 Other considerations

9.2.1 Singleton clusters

The tails of singletons noted in the summary are very unlikely to actually exist as proper motifs belonging to transcription factors, especially considering their low occurences. They are almost certainly noise, and should be ignored.

However, their existence also indicates the amount of noise in the entire dataset, including where it might interfere with clustering algorithms. Any succesful clustering algorithm will have to handle such noise correctly. A filter which can remove such noise before clustering might work, but ideally, clustering algorithms should be made robust against this type of noise.

9.2.2 Similarity across lengths

In designing both the distance function and the join function, it had been assumed that all sequences within a motif should be of the same length. Several sequences in the result contradict this assumption, including some non-trivial pairs with a length difference of 1. Therefore, future algorithms should *not* make this assumption, and instead find some way to interpret and represent similarity between sequences of different length.

9.3 Potential improvements

9.3.1 Distance functions

The distance function used was a simple Hamming metric, which is relatively course. Several refinements are thinkable:

9.3.1.1 Sequence-based refinement

Not all distances between two nucleotides are equal: each nucleotide has spatial and chemical features which are closer to or further from those of each other nucleotide. Incorporating such differences into the distance function could increase point-wise resolution, if based on biochemical knowledge. This would in turn increase resolution on the sequence level.

Additionally and alternatively, some features recognized by transcription factors are known to be based not on individual nucleotides, but by the transition between them in a sequence. Thus, the metric could be enriched by adding some similarity score if certain properties on sequential pairs of nucleotides are equal, rather than the nucleotides themselves.

9.3.1.2 Frequency-based refinement

There is a possibility that motifs have properly disjoint "cores" which are most often seen, but also contain less commonly seen "peripheries", which can overlap. If so, clusters might be distinguishable by having distances between sequences seen with low frequency be penalized (increased) compared to distances between high-frequency sequences. This would, under the assumptions given, keep cluster cores together while pushing the confounding peripheries apart.

9.3.2 Alternative algorithms

Constructing an MST doesn't really leave any choices or tuning variables, so it is hard to improve it except by choosing a better distance function. Alternative algorithms are thinkable however, so long as one keeps in mind that the sequence space does not have meaningful averages, and so can't use averaging-based clustering algorithms.

9.3.2.1 Convexity-based

One very plausible theory is that true clusters are (mostly) convex. This requires a generalisation of the concept of convexity:

A subset A of a metric space X is considered convex iff, $x, yA\, zX\, d(x,z) + d(z,y) = d(x,y)zA$.

In other words, if two points are in a convex set, that set must also include all points "between" them for which the triangle inequality is an equality. This is consistent with the notion of convexity in spaces with weighted averages such as real and complex numbers and vector spaces.

Given such a notion of convexity, it might be possible and useful to define an algorithm which tries to split non-convex clusters, ideally such that as many points in the cluster but not in the dataset are excluded from the union of the two new clusters (intuitively, split along the plane of maximum missing datapoints in the cluster). Given an algorithm to perform such a step, apply this iteratively to the most non-convex cluster after each step, starting with a universal cluster.

No attempt has been made here to write such an algorithm beyond formulating the concept, but further work could include making such an attempt.

9.3.2.2 Co-occurence based

Another possibility is to look not only at what sequences are found, but also where they are found. Transcription factors form complexes, so often, the same motifs will always occur together, with similar relative positioning. It is possible to base a statistical algorithm on this, with the chance of two sequences belonging to the same motif increasing as they occur next to the same other motif. This is a very different approach, and only vaguely sketched here, but it might detect patterns which pure sequence-based algorithms would miss.

9.3.3 Dealing with different sequence lengths

While it is relatively easy to adapt distance functions to enable comparisons on sequences of unequal length, by replacing Hamming distances by edit distances, doing the same for joins is a greater challenge, and we will make no attempt at a solution here.

Chapter 10

Conclusion

The dataset has turned out to not be amenable to analysis by simple MST using a Hamming metric. Several improvements are thinkable, some of which are based on the little information which the approach used here revealed. However, implementing such improvements is beyond the scope of this thesis.