Summary

Mapping the regulatory system in living organisms is a great challenge, and many methods have been created during the last 15 years to solve this problem. The biological processes are however more flexible and complex than first thought, and many of the methods lack the ability to imitate this exactly. The new method devised here is not a complete solution to this situation, but pose an innovative solution for finding approximate composite patterns in a set of sequences. Motifs are read from any third-party tool represented as either {A,C,G,T}, IUPAC or PWMs, and weighted with significance and support as an estimate to how important the patterns are. Finding combinations with both high significance and support can reveal important properties preserved in the sequences. Based on this, the algorithm use a branch-and-bound approach to traverse every combination while preserving the best solutions in this multiple object optimization problem in a Pareto front. The best patterns found, are investigated further by applying different statistical and experimental method to better support the significance of the patterns found. The three most important tests done on the TransCompel dataset, where (i) to look at the patterns predicted measured against known sites based on nucleotide correlation. (ii) Find the frequency for motifs participating in the combinations, so that the best could be studied manually. And (iii), different test where compared when the significance was based on real background sequences instead of the uniform distribution. Some of the results found where low, but still similar to the accuracy provided by other known methods that have been tested with the same methods. The test results can be biased by the parameters used, a too simple and restrictive test set or by faulty predictions done one the dataset tested. More testing and tuning of parameters might result in better predictions. However, the different tests still proved this method to be a valuable tool in composite motif discovery.

Abstract

Motif discovery in biological sequences has in recent years increased its focus on motif combinations, often referred to as modules. Modules that occur in several related sequences may be functionally important as their overrepresentation may suggest evolutionary conservation or that the same transcription factor regulates several genes in the same genome. However, all motifs contained in a set do not necessarily occur in every sequence. This can be explained by flexibility in the biological mechanism, or due to noise or inaccuracies in the discovery of single motifs.
A new algorithm to handle this is presented, and it is compared against existing methods. This algorithm is designed to discover any subset of matches between elements within a larger set, and report the approximate composite patterns discovered. Finding these patterns is a hard combinatorial optimalization problem. This method takes a list of motifs represented as {A,C,G,T}, IUPAC or PWM patterns found in a set of sequences from any third party motif discovery tool or any patterns based on known motifs. The subsets of motifs will be matched in different combinations while preserving the best values in this multiple object optimization problem based on the support and significance of the combined patterns. Different measures and constraints are added to narrow the search and a variety of statistics is calculated for better evaluation. The best patterns found can then be investigated further. And in the situations investigated here, some tests gave interesting results while other scored a little low. This method still showed promising results as a tool to discover composite motifs, and an ability to reveal pattern combinations between binding sites in DNA or active sites in proteins that might represent important biological functions.

Content

List of figures

List of tables

List of equations

1 Introduction

During the last ten years, an increasing interest for motif discovery in DNA and protein sequences has emerged. The motifs or *transcription factors* are short patterns common to a set of sequences, and they represent and match what is known as *transcription factor binding sites*, which are the actual sites recognized by the biological mechanisms during gene regulation. Thus finding these patterns is important for understanding the regulation process [15, 16]. This development have introduced a variety of different algorithms, but previous work done in computational biology focus mainly on single motifs discovery or combination of these patterns combined into complete sets as explained by Sinha [6]. Research in biology has shown that composite motifs play an important role in mapping the regulatory regions of genes [6]. The same methods can also be used to locate active sites in proteins. But since the biological mechanisms and evolution process is more complex and quite flexible, there is a need to investigate patterns where not all the elements in a combined set needs to occur in every sequence.

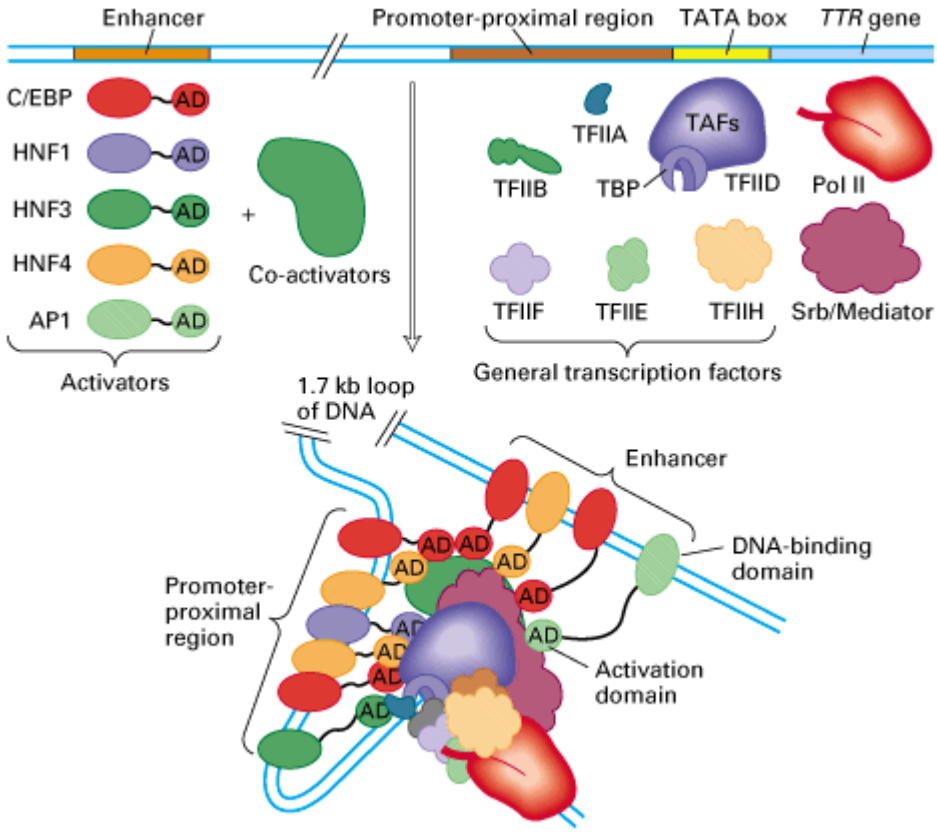


Figure 1-1:*Shows how multiple transcription factors co-regulate a complex biological process (TTR control region (promoter) example [14]).*

The method developed in my former project [1] aimed at handling incomplete sets where only a subset of m elements from the n total needed to occur, but

focused mainly on handling the special cases “3 of 4”, “4 of 5” simplified to N-1 of N, which was not a completely general method. It was therefore interesting to investigate other combinations as well, since the human regulatory system might allow these degrees of flexibility. The goal of this project was to expand and devise a completely general and flexible approach to finding true m of n composite motif, where any combination of m and n would be possible. New classes, motif representations, calculations and techniques needed to be introduced.

In addition to the deterministic motif model already supported, the new method will be extended with the probabilistic PWM model [7], thus adding the need for a variety of alternate measures for significance and support for every model. The new measures taken into account are both calculated from empirical sampling during runtime, devised by known methods in statistics or found experimentally, similar to work done by [8, 9]. This enhancement opens up the horizon regarding better comparison against earlier developed methods and the output from those as well.

Variations in significance are also explored by varying the input to the empirical models, instead of only applying new methods on the same data. In addition, using the added distance constraint can help limit hits within desired areas in the promoter region. All these additional means of measure will hopefully make it easier to validated, quality check and locate the best motif combinations, and later produce optimal results for scientist faced with similar problems. Much work is needed through experimentation on optimizing and tuning input variables when using this new method. The huge flexibility makes this method quite unique compared to the existing set models [6].

2 Biology background

In this project, I investigate composite patterns existence and how different backgrounds models affect the individual single motif significance and changes in overall results for the composite motifs. In addition to read pre calculated significance from a third party discovery tool, the method also support calculations of values based on models that better simulate a real background. Because transcription factor binding sites sometimes stand out more from the general background, three different approaches can be tested and compared. The significance can be based on the frequencies found in the target sequences used, a separate background set or randomly created sequences. For the target and random sequences a uniform or skewed probability distribution over the letters will be used. While the last method will actually align the motifs against real background material from the same domain and count hits. These variations in significance will cause alternations in which motifs becomes the most significant and then affect what is finally included in the resulting composite motifs.

Background theory is especially interesting to evaluate for promoter regions with CpG islands [10] or other un-regularities. CpG islands are regions overrepresented by the CpG pattern, and are thought to give special properties to the nearby genes. The distance to the uniform method will be most significant in cases like this.

2.1 DNA

To retrieve the information stored in DNA from a living organism, a sequencing process takes place [11]. When sequencing DNA the order of nitrogen bases are determined and form a list of letters from the amino acid alphabet “ACGT”.

CCTGACTGACCAGGTCCTAAGTCATCCTAATGGTC…

Figure 2-1*:* *Shows a sequence pattern.*

A more thorough description of the inner workings of DNA can be found in my preliminary work [1], and the Human Genome Project [11] has published a good deal about sequencing.

The interesting patterns in these sequences can be represented by a variety of models [12], but common to them is the need for a measurement, to tell how significant it is, and separate the patterns from the background, the information stored in spaces between the patterns we look for. Many regions in DNA often contain special sequences, which gives exactly this part a unique background probability. This can occur as periodic over or under representation of specific nitrogen bases [10]. And by investigating combinations of motif models with several background models, the new general method examined later can give even more exact hits, if we find that the patterns examined stand out enough so it becomes significant.

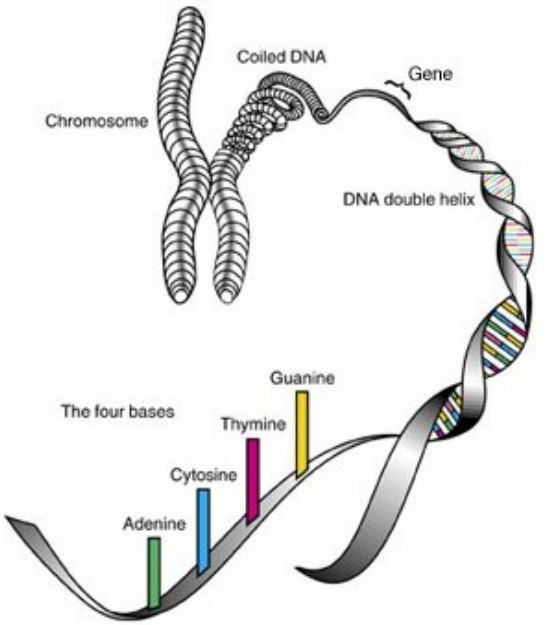


Figure 2-2:*Shows DNA uncoiling from the chromosome storing structure, overlapping into the double helix twirl, before revealing the inner building stones of DNA – the four different nitrogen bases (Adenine, Cytosine, Thymine, Guanine) [1].*

The same theory applies to protein chains build up by peptide-bindings, which are produced from genes in DNA through transcription [3]. The resulting sequences are expressed in a similar way, and pattern discovery here can reveal interesting features in active sites or domains for the proteins [1].

By being able to study more complex pattern combinations, it makes the discovery in genes regulatory regions and proteins much more flexible. These models should correspond more to the complexity and flexibility of the mechanisms we explore. The findings are presented in the result and discussion section.

2.2 Composite motifs and transcription

The transcription factor binding sites (TFBS) are located in the promoter region of the genes they regulate, and they are the target for proteins (Transcription Factors or motifs), that connects to these areas before starting the transcription process. Mapping and understanding how co-regulated sites behave is quite difficult due to the great complexity, and this process can start in several ways [24]. Composite patterns can act *synergistic* (co-operatively) or *antagonistic* (motif interference) [24]. Variations in the TBFSs occurring across sequences are discovered as composite patterns by this method and approximate composite patterns when subsets of all TFBSs are located

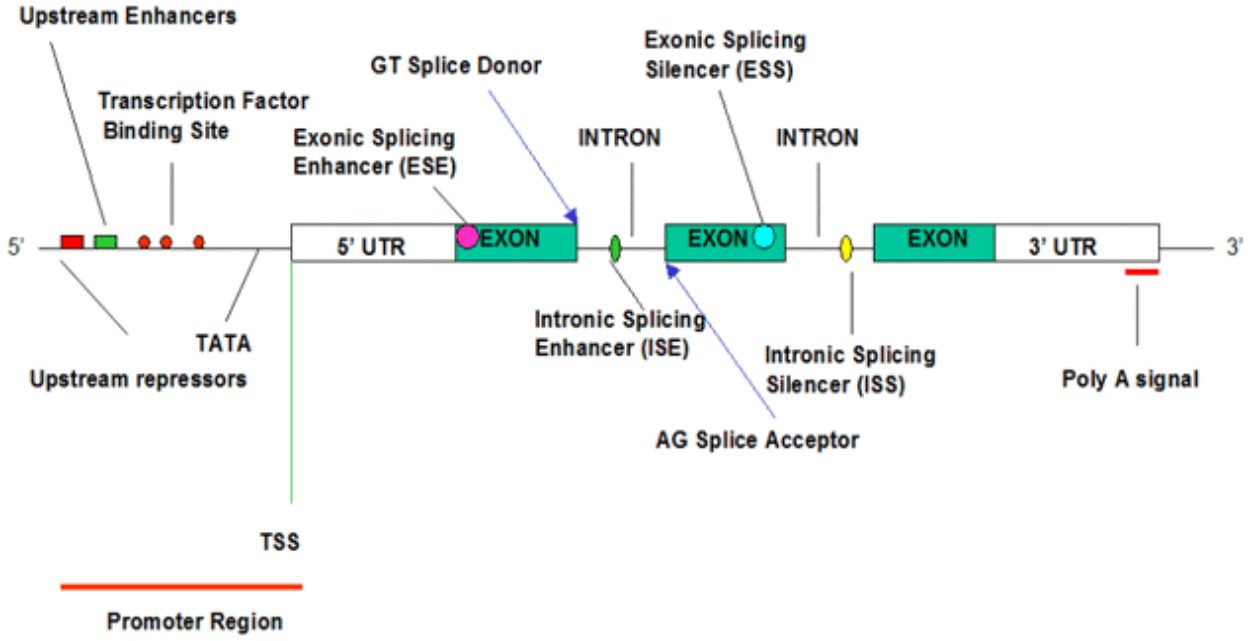


Figure 2-3: *Shows the structure of a gene, with the Transcription Factor Binding Sites located in the promoter region [25].*

2.2.1 Co-regulated example

The two main types of modules, *synergistic* and *antagonistic* co-regulate genes by simultaneous interaction or mutually excluding actions. [24]

2.2.2 Background variation example

Some regions in yeast are proved to have special properties for the nucleotide distribution [27, 28]. Spingjola et. al, located patterns mainly consisting of polyT islands, which are long sequences of T, occasional separated by an A or C. Such sequences are found to affect *splicing* [27]

3 Methods and algorithm

This method which is devised to find approximate composite motifs is based on motifs from third-party motif discovery tools or sets of known motifs. The motifs are then combined as building blocks through an exhaustive combinatorial process to find the best possible composite patterns. To be able to evaluate the modules in combination (motif variants), all the single motifs and motif variants have in addition to their pattern or PWM representation a significance value to estimate its importance and a support value which represent the number of sequences where this motif is found.

3.1 Pareto front

The best combinations found, is preserved in this multiple object optimization problem as a Pareto front based on this significance and support. Pareto front representation is a trade off between the two values.

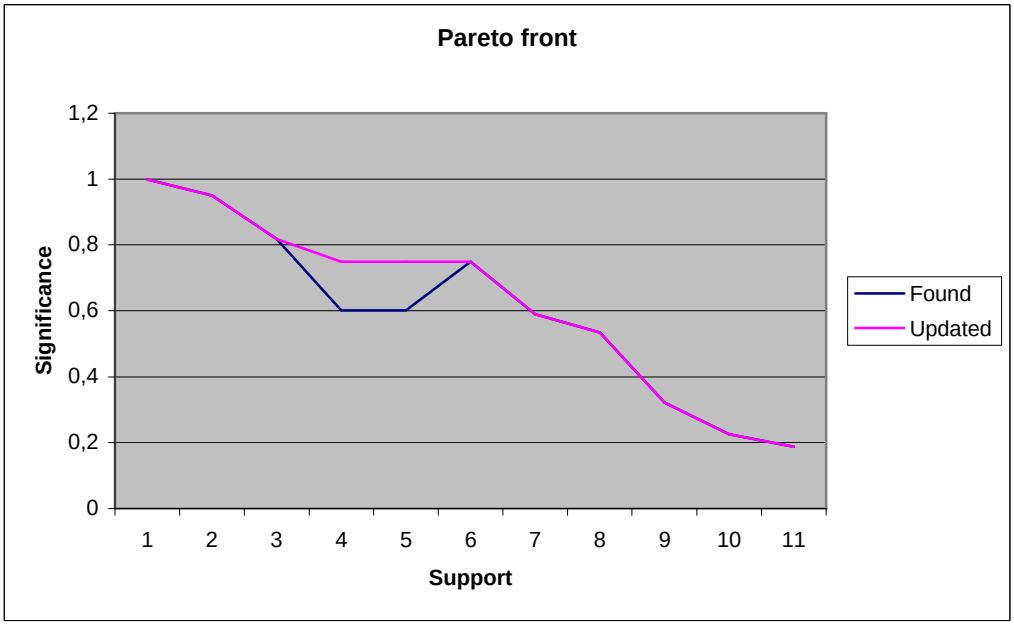


Figure 3-1: Shows the Pareto list first found and the Pareto front after dominated values has been updated. The graph after update also shows an ideal and optimal Pareto front [29].

The first implementation kept parts of the Pareto which was not really interesting or a part of the Pareto front, since many of the modules where dominated by other elements in the Pareto front [1]. To optimize running time for the algorithm, a Pareto-flatting was introduced. This update on existing Pareto front entries helps the pruning to become more effective, since a bigger subset in the enumeration process can be cut-off during runtime.

3.2 Motif representation

The motifs can be any of three different representations, either a sequence of letters from the amino acid characters which match against the input sequence directly as a hit or no hit.

ACCTGTTCATGT

Figure 3-2*: Standard amino acid characters.*

Second, the more complex IUPAC representation of a pattern [App B], which contains additional letters and signs that can accommodate for only partial hits.

ACCNRYYMTGT

Figure 3-3*: A IUPAC representation of a pattern. See [App A] for more information.*

Or last, the pattern can be represented by a PWM (**P**osition **W**eight **M**atrix), where different probabilities for hitting the sequence alphabet at different positions are used instead.

A	C	G	T
0,2	0,4	0,1	0,3
0,1	0,1	0,3	0,5
0,3	0,4	0,2	0,1
0,7	0,1	0,1	0,1

Figure 3-4*: Shows a PWM matrix with probabilities for a motif with length 4.*

Regardless of which representation is used, the motifs have different measures and properties that need to be handled. The two most important are significance and support, and they are calculated in various ways for the different representations.

3.3 Significance

Significance tells us how likely it is for a pattern to occur in a random sequence, and therefore also how special the patterns is estimated to be. This likelihood is based on Equation (3.1) [5], and lies in the closed interval 0.0 to 1.0, where values close to 0.0 are the most significant. One of the highest objective in this methods is to minimize this number for a given pattern combination. Thus maximize the chance for finding special patterns.

$$P = \frac{valid}{total} \tag{3.1}$$

The significance value used in this method is based on the likelihood for hit in a sequence, not at a special position within the sequences. The probability for observing a hit in one or more sequences can be calculated in several ways.

One way to find **sign_seq** is to start by finding the probability for a hit at every position within a sequence, referred to as **sign_pos** here, and then find sign_seq based on that. Each letter observed in the pattern is multiplied with the co-responding probability distribution for that letter based on the frequency for when it occur in the sequences.

Sign_pos, the likelihood for a hit at given position in the sequence:

$$Sign\_pos = \prod_{x=1}^{length(L)} L(x)P(L(x)) \tag{3.2}$$

Sign_seq 1, transition from sign_pos to sign_seq:

$$\text{Sign\_seq} = (1.0 - (1.0 - P(pattern)^{avg(|sequence|)} \tag{3.3}$$

Sign_seq 2, found directly from Equation 3.1.

$$Sign\_seq = \frac{(sequences\ with\ hits > 1)}{(Total\ number\ of\ sequences)} \tag{3.4}$$

The patterns and their hits at different position are treated independently.
Long or to rare patterns may cause numerical problems in Java for to small significance values, and log-representation should be used instead.

Due to complexities in the combinatorial process, this method operates with a different set of calculations and values for significance and support for single motifs, the motif variants and finally the composite motifs.

3.3.1 Single motif significance

All single motifs contain a significance value, and this value is in some experiments predetermined by a third-party tool or scientist, calculated based on statistics or measurements. To optimize the results, different approaches for significance calculations are implemented. Three methods supported are:

1. The uniform distribution, every letter with P = 0,25.
2. Probability based on actual frequencies in background sequences.
3. Aligned against background sequences, and measured.

Methods 1 and 2 above are most sensitive to the numerical problem mentioned in the previous section, while the 3rd might return 0.0 if no occurrence is found. The last method use real background sequences and should give the best results. However, using option 3 for the PWM representation dos not create a hit, but a list of calculated values. A threshold is used to count values over this limit as a hit and discard the rest.

3.3.2 Motif variants significance

Some of the design and ideas presented here are equal or similar to what I did in my preliminary work, since this project continues in the same path. However, this project demanded some big changes to make all the new requirements work.

To evaluate the different and possible routes through the search space in a general way, a common approach to loop through the steps and calculate the composite motifs where devised. Instead of the two-step solution for M=N-1 of N and M=N of N that where previously used, the new method can now handle any M of N. This was done by representing every combination in Table 3.1 as a motif variant, and constructing sets of default motif variants outside the upper border and as the initial step for each column in Table 3.1. With this in place, all partial solution would be handled perfect.

4						
3			D	3/3	3/4	3/5
2		D	2/2	2/3	2/4	2/5
1	D	1/1	1/2	1/3	1/4	1/5
0	I	I/1	I/2	I/3	I/4	I/5
m/n	0	1	2	3	4	5

Table 3.1*: Shows the different motif variants that are included and used during a 3 of 5 example in this new model. The blue "D's" are dummy objects created at runtime, while both the red and green "I's" are added as initial starting points for each column in the calculations.*

The motif variants introduced, are the most important part of the new algorithm. Table 3.2 shows the order in which the variants from Table 3.1 are created. Number six (blue) depends on the two previously calculated variants. Most of the significance and support calculations take place here.

4						
3			**7**	**10**	**12**	**13**
2		**3**	**6**	**9**	**11**	
1	**0**	**2**	**5**	**8**		
0	**0**	**1**	**4**			
m/n	0	1	2	3	4	5

Table 3.2*: Shows the order in witch the motif variants are calculated through the enumeration. 6 e.g. (marked blue) depends on the variants 2 and 3 (red).*

From the motif variant matrix in Table 3.1, if we let **r** and **c** respectively represent rows and columns and **m** a new single motif, the significance for the new motif variant C is always calculated based on the two previous motif variants A=(**r**,**c**-1) or B=(**r**-1,**c**) and **m**, which can be translated into.

(3.5) $$Sign(C) = P(A\&B|d) = P(!(!(A\&B)\&!d))$$

(3.6) $$Sign(C) = 1-(1-sign(A)*sign(B))*(1-sign(d))$$

However, since not all the motif variant exists, e.g. outside the borders, these locations are filled with *dummy* motif variants with settings that do not affect the calculations when those outer limits are reached. Since the significance for a dummy variant is set to 0.0, the last link in Equation (5.6) above becomes 1, and only the part calculating the transition from motif variant (**r**-1,**c**-1) and **m** has any effect.

At the other limit, the beginning of a column, a *first* motif variant is introduced with 1.0 as its significance, and it is used in calculations by future successors.

3.3.3 Composite motif significance

To evaluate how well composite motifs are formed, the significance between the single motifs participating in the module is calculated during runtime as motif variants. The last motif variant calculated represents a possible composite motif solution for this pattern, and the significance and support represent the likelihood for observing this pattern again. Instead of maximizing either support or significance, the combination is a trade-off which represents all the best solutions between the two extremities, this form Pareto optimality [29], which other modules not part of the solution is dominated by.

In the final stage, the best composite motif found is selected, and tested with other significance models as well. However, those calculations are also based on support, so they will be handled after the section on support.

3.4 Support

Support is a representation of the number of distinct sequences a single motif or composite motif occurs in. A single motif can have several hit positions within a sequence, but only one contributes to the support. For deterministic motif representations, a hit is either a match or mismatch in the sequence which is easy to determine. For PWMs however, a value representing the hits needs to be calculated for every position within the sequence, and based on a selected threshold, a hit will either be kept or discarded. Deciding on a to strict value may discard significant motifs, while a to low threshold might give a search dominated by less significant motifs [2]. The support can then be found by registering all matrix scores above the selected threshold as an hit.

A **hit list** is generated for each motif, and contains every sequence number and sequence hit position found. This list can contain many hits in the same sequence, and is the same for all motif representations.

1	117	1	683	2	24	3	844	8	937	11	1002

Figure 3-5*: Shows a motif hit list with alternating sequence and position.*

Bit set is an easy way to represent if a motif or composite motif hits in a given sequence one or more times. By using true/false or 1/0 to represent a hit or miss, we can in an efficient way calculate the differences or similarities by using operations like AND and OR on the sets. The bit set cardinality equals the support.

1	1	1	0	0	0	0	…	1

Figure 3-6*: Shows a bit set representation of the hit list.*

The logic AND and OR operations behave as follows:

AND	**OR**
1 AND 1 = 1	1 OR 1 = 1
1 AND 0 = 0	1 OR 0 = 1
0 AND 1 = 0	0 OR 1 = 1
0 AND 0 = 0	0 OR 0 = 0

Figure 3-7*: Shows the logic operators AND and OR.*

3.5 Mofn algorithm

To solve this combinatorial problem, a new approach had to be taken since the existing Mofn algorithm [1] still had some weaknesses compared to what the goal was to begin with. This type of hard combinatorial problem applies a branch-and-bound [4] technique to explore and the entire solution-space. All the possible combinations are tried out through a recursive enumeration tree, where each sub-problem shares the same constrains and objectives, except at the leaf-nodes, which contains the values we are looking for [1, 4]. To create this tree, the motifs are sorted after descending significance, before the algorithm start to explore the permutations that make up the search space in a Depth First Search [4] as shown in Figure 3.8. In this new and more flexible method for finding m of n combinations, the design incorporating the motif variants mentioned in section 3.3 together with belonging calculations had to be applied to the enumeration process. Thus allowing for any combination of m and n, and giving more accurate pruning while exploring the tree. All the most significant motifs are explored first, and the pruning used will be based on the significance and support values found. The pruning cuts down on the search-time, by exiting the search loop when the best possible leaf-node so far is already located. There will be more information about pruning in a later chapter.

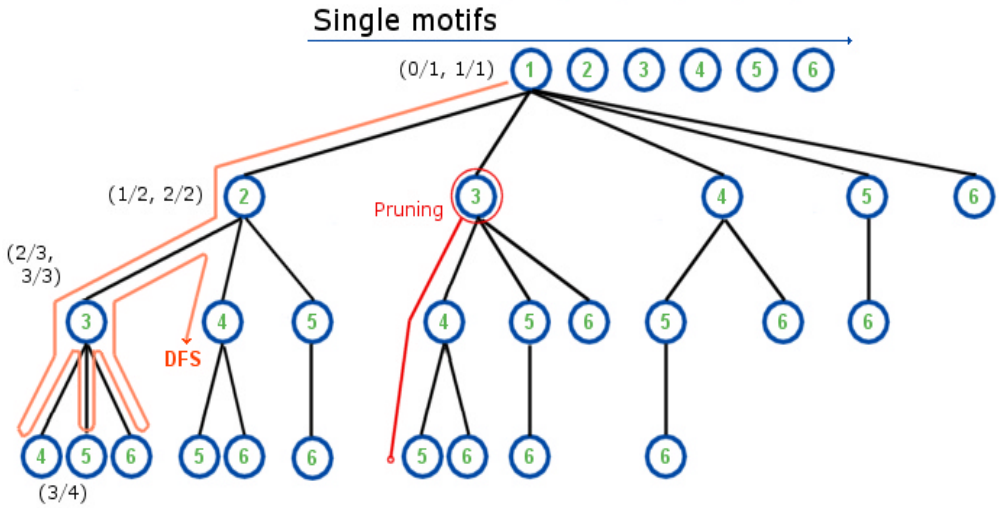


Figure 3-8: *Shows the tree structure for a 3 of 4 search with 6 single motifs. Each single motif added to the tree participate in forming a composite motif (at every blue node) with the different motif variants calculated at the current level showed in parantheses next to the node. The red circle shows how pruning is performed by constructing and expanding a fictive composite motif (red line), and checking the possibilities before exploring the entire sub-tree.*

When the list of solution is computed, the initial solution is presented in a Pareto front, which is a representation of the multiple object optimization problem based on support and significance. The composite patterns in the Pareto front are listed, and the single motifs are sorted after occurrences in the modules they participate in.

As a final first step, the results calculated will be used to evaluate the quality of the composite motifs found by applying new methods and further calculations. Based on both statistics and experiments, a set of new values that can help us to evaluate the composite motifs are reported.

3.6 Pruning

When exploring the branch-and-bound-tree, subsets in the tree will often produce combinations of values lower than results from previously examined regions of the tree, and the subsets can be discarded. This process is known as pruning [30], and in the previous method [1], the upper bound for a subset was based on theory similar to that of GCMD [2], and that upper limit was only an approximation which gave a heuristic improvement while pruning. To get a more correct pruning the new design use an ideal[1] single motif in combination with a fictive clone of the current real composite module. This fictive test is based on real motif variant calculations, equal to those used in the real expand method, and this will reveal if better possibilities not already covered by the DFS exploration so far can occur. If pruning at the leaf-node returns a value less than what is located in the Pareto, the current real composite motif expansion for this subset is aborted. The search space is reduced by a factor k [30]. A small (2 of 3) test on 15 sequences/817 motifs showed that the optimal solution could be found in approximately 1 minute instead of 20.

1: A motif with significance equal to the previous added, and the entire bit set = true.

3.7 Search tree

To find the size of the DFS tree spanned by the enumeration process without pruning, the binomial coefficient shown in Equation (3.7) needs to be summed up for every value of N, as shown in Equation (3.8).

$$\binom{I}{N} = \frac{I!}{((I-N)!N!} \tag{3.7}$$

$$S = \left[\sum_{m=1}^{N} \frac{I!}{((I-m)m!)} \right] - N \tag{3.8}$$

3.8 Runtime

The runtime of this algorithm is mostly dominated by the motif set size and motif variant to find (m of n). While supporting code run in linear time as a function off the number of sequences read in and used for support and significance estimations.

Based on the runtime theory in Cormen [4], this algorithm (without pruning) is estimated have an upper bound on runtime given by:

$$O(I^N) \tag{3.9}$$

where I is the size of the motif set, and N the number of motif permutations checked (m of n). N is usually selected to be in the area around 2-6.

However, pruning will cut down the branch-and-bound-tree dramatically by a factor k, depending on how good the pruning works.

3.9 Post processing

To further investigate the quality on composite motifs found so far, a variety of other significance measures are introduced.

3.9.1 Alternate support score

In addition to the already calculated composite motif support, another support model where also added. Instead of using intersecting values among the single motifs, a total hit count is computed.

Sm1	1	1	0	0	1	1	4
Sm2	0	1	0	1	1	0	3
Sm3	1	1	0	0	1	0	3
Total	2	3	0	1	3	1	**10**

Figure 3-9: *Shows how the alternate support is added up.*

This gives us another score for how good the motifs hit, and this is the support value used when selecting the composite pattern to investigate with additional statistical and experimental models, since this is the single motifs actual support observed and therefore give a good likelihood of observing the same results in random sequences in these new models.

3.9.2 Weighted support

To highlight tendencies in the alternate support, a weighted version was created. This simply raised the initial value in the power of 2. Other weights could be used, depending on what properties that should be highlighted.

3.9.3 Binomial significance

For the module selected, the calculations are chosen to be based on the support found during the enumeration, which represent the shared hits for this composite motif. These data then build the foundation for the binomial [5] trails that calculate the probability for observing the composite motif with the highest found value for total support.

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} \tag{3.10}$$

This binomial equation is then used to find the probability of getting the support value observed or higher:

$$P(S \geq support) = \sum_{n=support}^{L} \binom{L}{n} p^n (1-p)^{L-n} \tag{3.11}$$

where L is the number of sequences involved in the trail, and p the modules significance.

3.9.4 Total significance by trials

This method test the single motif patterns against different background sequences over a very large number of trials, in the range 1000 – 100 000, with many test sequences, and try and reproduce the hits found by the total support, and reports the empirical found significance based on the number of hits found equal to or above the support.

In statistics, the same test could be calculated by using equation 1 together with all the valid permutations divided by all the possible permutations [31], but for most input this would be an enormous number, so finding the possible permutations is not feasible during every run. A normal distribution is used instead.

3.9.5 Normal distribution

In theory, the total significance above could be found by using the support values to form the basis for expected mean μ and variance s^2, $S = N(\mu, s^2)$. And by combining different motif support, we look for a probability for the interval equal or greater to the support found. The problem in practice are often either to few samples to form the distribution with, or a number to great to calculate efficiently. Assuming we have a proper amount of samples, the normal distribution is and idealization of our independent trials.

(3.12) $$m = E(X) = \sum_x xf(x)$$

(3.13) $$s^2 = E(X - m)^2 = \sum_x (x - m)^2$$

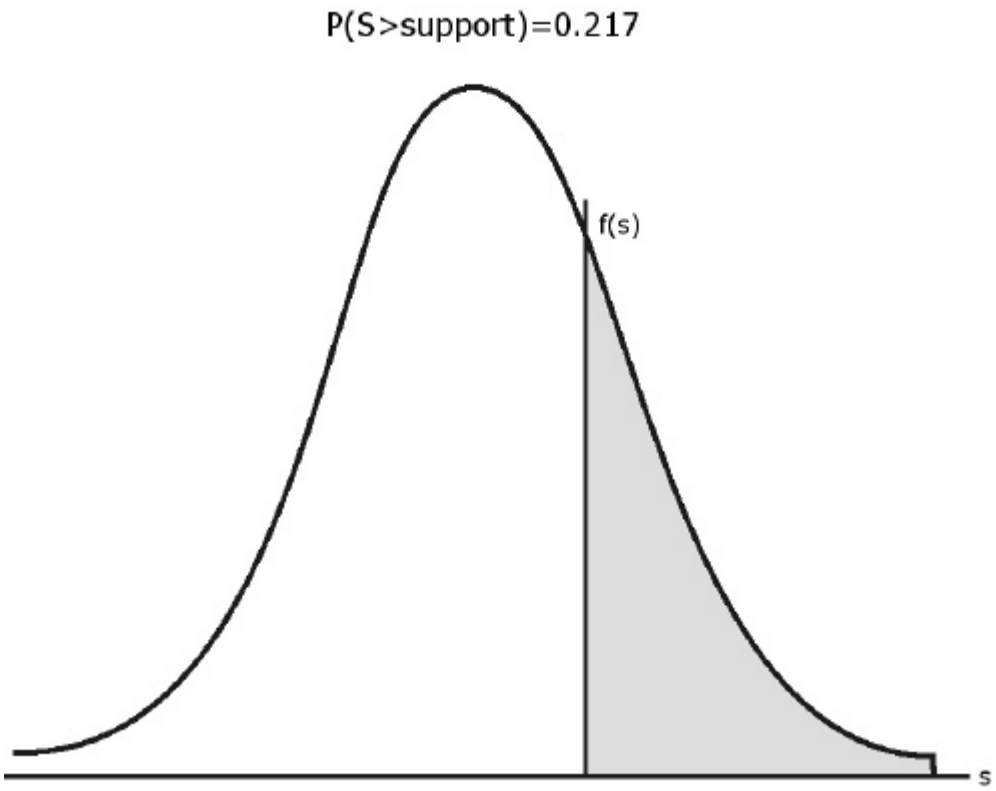


Figure 3-10: *Normal distribution curve. The gray area equals the probability for a support greater than the value tested for.*

$$N(x;m,s) = \frac{1}{\sqrt{2p}\,s} e^{-(1/2)[(x-m)/s]^2} \quad ,0 < x < |\text{sequences}| \tag{3.14}$$

$$P(S > support) = \sum_{x=s}^{|sequences|} \frac{1}{\sqrt{2p}\,s} e^{-(1/2)[(x-m)/s]^2} \tag{3.15}$$

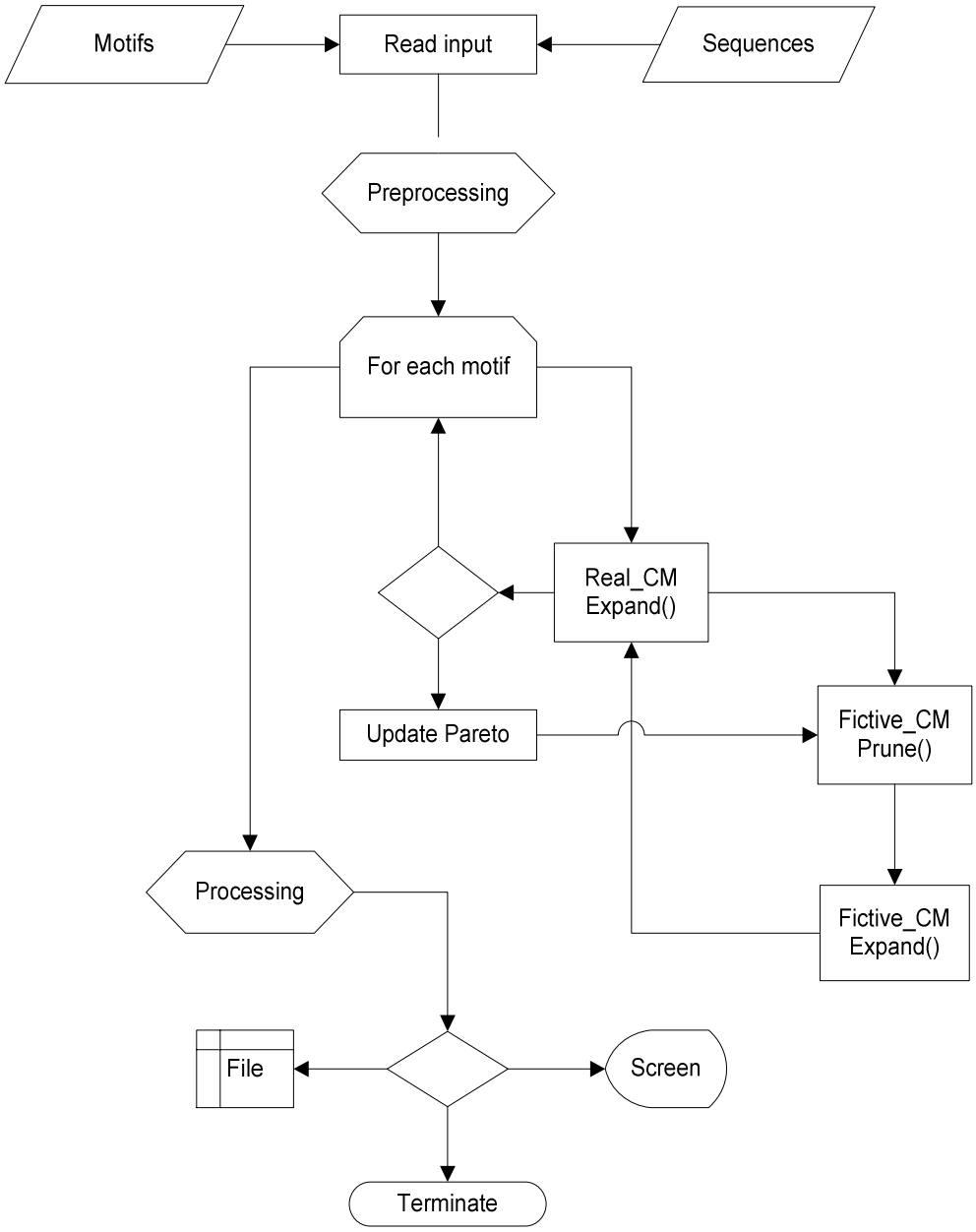
3.9.6 Distance constraints

Distance constraints can make it easier to locate patterns in special regions of the sequences. Start and end position can be added to limit output to this region. This operation takes place post a Mofn run, and recalculates the support for single motifs to take account for the new region, only hits within this is now a positive hit. All the output is then based on this new support. To implement distance constraint during runtime and to prune on it is a great challenge, and another student is currently working on a project handling this for the GCMD method [3].

3.10 System design

This new method is designed with many additional classed compared to the old method [1]. The diagram below show how the different objects work together. The UML-diagram of the classes can be seen in App. E.

Flowchart



4 Implementation

Since the method is implemented in Java [27], it can be used on any system running the Java Virtual Machine Interpreter. Appendix G contains a UML-diagram of the implementation, Appendix H contains the documentation and appendix I contains the entire source code.

4.1 System input

The method supports three different running modes: STANDALONE, SERVER and CLIENT.

For all the modes, a variety of parameters can be added to control the run.

4.1.1 List of all input parameters

-mf <motiffile>
-sf <sequencefile>
-bf <backgroundfile>, used to alter significance for single motifs
-of <outputfile>, default='output.txt'
-m <m> m<=n, default=2
-n <n>, default=3
-type <mot/pvm>, default=pvm
-mode <STANDALONE/CLIENT/SERVER>,
default=STANDALONE
-ip <xxx.xxx.xxx.xxx> Server IP address
-port <XXXX>, default=7777
-cutoff <percent>, default=20
-print <111111111> 1 prints, 0 does not
-display <111111111> 1 prints, 0 does not.
-start <distance start position (integer)>
-end <distance end position (integer)>
-prune, turns on pruning
-align, use alignment instead of frequencies for motifs (not PVM)

4.1.2 Motif files

Three different motif representations are supported, and the type used must be specified with the –type parameter. PWM is default.

ACGT

```
#Comments
5    5    gggccag 0 291 3 755 4 2669 6 335 7 219
5    5    agcaacc 1 118 2 295 3 1074 4 6734 6 1138
5    5    ccagagg 2 1154 3 1148 4 2014 5 124 6 1151
5    5    tttttctt 1 1334 2 4567 3 1464 4 6817 6 1533
```

Figure 4-1: *Shows a list of standard motif patterns based on {A,C,G,T} together with data representing hits in sequences at given postions.*

IUPAC

```
#Comments
5       5       ggg[ct]cag 0 291 3 755 4 2669 6 335 7 219
5       5       agc_aacc 1 118 2 295 3 1074 4 6734 6 1138
5       5       ccagagg 2 1154 3 1148 4 2014 5 124 6 1151
5       5       ttt[tga]ctt 1 1334 2 4567 3 1464 4 6817 6 1533
```

Figure 4-2: *Shows a similar list as Figure 4.1, but greater flexibility in the patterns are allowed, IUPAC [Appendix B].*

PWM

```
#Comments
>M00658
8.0 0.0 1.0 5.0
0.0 3.0 11.0 0.0
9.0 2.0 3.0 0.0
2.0 0.0 12.0 0.0
0.0 0.0 14.0 0.0
14.0 0.0 0.0 0.0
14.0 0.0 0.0 0.0
1.0 3.0 10.0 0.0
>M00971
41.0 12.0 20.0 8.0
3.0 55.0 5.0 18.0
12.0 2.0 0.0 67.0
0.0 0.0 0.0 81.0
0.0 81.0 0.0 0.0
1.0 77.0 0.0 3.0
4.0 8.0 14.0 55.0
5.0 33.0 31.0 12.0
```

Figure 4-3*:* *Shows the probability distribution for patterns represented as Position Weight Matrices. The columns represent A,C,G,T and the row count the motif length.*

4.2.4 Sequence file (target/background)

```
#Comments
>name1
ctcagagtggctgcagtctcgctgctggatgtgcacatggtggtcattccctctg
>name2
aggtctgtgagagctcccctcacactcaagtctctctcacagtggccagagaatg
>name3
attaatcatttcctctgtgtatttaagagctctttgccagtgagcccagctttc
```

Figure 4-4*: Shows sequences in the Fasta-format.*

4.3 Running on a standalone system

To start the program, use this commando:

```
Java Mofn –mf <motif file> -sf <sequence file> -m <m> –n <n>
```

Figure 4-5: *Shows the standard commando used to start the program.*

4.4 Making use of clients and server for a distributed run

Some calculations run with this system might demand more computational power then what a single PC can offer in reasonably time. The previous version of Mofn, had a simplified client/server-model implemented, and it was difficult to administrate. It also lacked a common Pareto front, for more optimal pruning throughout the nodes.

4.4.1 Server

A server is first started with desired port and number of clients as arguments:

```
Java Mofn –mode SERVER –clients <X> –mf <motif file> –sf
<sequence file> –m <m> –n <n>
```

Figure 4-6: *Shows how to start a server for a distributed experiment.*

The server then prints its IP, which is used from every client when connecting to the server. Both the server and the clients contain and read the input-files to do initial calculations and find the dimensions on the data.

During the run, the server contains the most updated Pareto front, which each client continuously update and read.

After a completed run, the server receives the composite motifs from the client, and the post run calculations and printing of results are handled.

4.4.2 Client

Each client is started with the IP and port for the server. The interval assigned to the client is provide by the server together with the m and n values needed for the calculations.

```
Java Mofn –mode CLIENT –ip <XXX.XXX.XXX.XXX> –
port 7777 –mf <motif file> –sf <sequence file>
```

Figure 4-7: *Shows the code used to connect to the server already started (above).*

4.5 Program output

The program can output a variety of data either to screen or file. The default setting is to print everything, but this can easily be controlled by the parameters:

-print 010100000 and **-display** 010100000

"1" prints/displays corresponding output with position list below from 1-9:

1. Display motifs with stored information
2. Pareto front
3. Composite patters (with single motifs)
4. Most common single motifs (counted)
5. Counted support (total)
6. Weighted support (^2)
7. Binomial
8. Experimental significance
9. Final results (alignment against target sequences)

The output is spread over two files, where X is the experiment number automatically provided by "nr.txt":

output_(X).txt
<motif filename>(MN)_(X+1).txt

The details are explained in the following two sub chapters.

4.6 Outputfile (X)

The first file contains the output from all the basic calculations, together with the motifs and parameters used. By including all the parameters used in the output file, the tests are very easy to reproduce.

4.6.1 Display motifs with stored information

After data is read into the program, a list of all the motifs can be printed to either screen or file with the associated values. This makes it easy to control the data ready for processing.

```
806   -  Supp: 6   -  Sign: 0.4958583819811141   -  Pattern: cagagga
{0, 2, 3, 4, 5, 6}
807   -  Supp: 6   -  Sign: 0.4958583819811141   -  Pattern: ctgctca
{0, 1, 2, 3, 4, 6}
```

Figure 4-8: *Shows motif ID, support, significance, pattern and the hit list indicating which sequences the motif occur in.*

4.6.2 Pareto front

The Pareto front list, is printed after first applying (1-significance), this gives the most important motives the highest values.

```
Pareto[0]: 1,0
Pareto[1]: 1,0
Pareto[2]: 0,9973768658189076
Pareto[3]: 0,9948203878210522
Pareto[4]: 0,9307878926097515
Pareto[5]: 0,9307878926097515
Pareto[6]: 0,6656482801289907
Pareto[7]: 0,0
Pareto[8]: 0,0
```

Figure 4-9: *Shows the Pareto front for a test with 8 sequences. The significance has a value between 0.0 and 1.0.*

4.6.3 Composite patters (with single motifs)

Each composite motif represented in the Pareto front is then listed, so that further investigation of interesting patterns can take place.

```
Pattern[0]: aaaaatg   tcctcttcct    ttccttctag
Pattern[1]: aaaaatg   ttccttct             ttccttct
Pattern[2]: ccttctcct tcctcctcc     tcctcttcct
Pattern[3]: ccttctcct ctcctcctc     cctcctcct
Pattern[4]: aaccaacc  tcttcctt             ttcctcct
Pattern[5]: aaccaacc  tttttctt             ttttcttt
Pattern[6]: aaaaatg   tcttttt       ttttcttt
```

Figure 4-10: *Shows the single motifs participating in the composite module for each support in the Pareto Front.*

4.6.4 Most common single motifs (counted)

Since it can be interesting to see which single motifs participate most frequently, a list is built and the occurrences are counted and sorted in descending order.

```
aaaaatg: 3
ttttcttt: 2
tcctcttcct: 2
ccttctcct: 2
ttccttct: 2
aaccaacc: 2
tttttctt: 1
ttcctcct: 1
tcttttt: 1
tcttcctt: 1
tcctcctcc: 1
ctcctcctc: 1
cctcctcct: 1
ttccttctag: 1
```

Figure 4-11: *Shows the frequencies based on appearance in a module for each single motifs.*

4.6.5 Counted support (total)

To get a better estimate on the support for each module, another calculation for the support is presented.

```
CS[0]: 4
CS[1]: 3
CS[2]: 6
CS[3]: 8
CS[4]: 9
CS[5]: 12
CS[6]: 13
CS[7]: 0
CS[8]: 0
```

Figure 4-12: *Shows support based the sum of every support instead of using the logic operands for the intersection.*

4.6.6 Weighted support (^2)

Since there can be special support values that are extra interesting, a weighted addition to the previous model presented was implemented. It currently use the 2-power function, but can easily be switch with other models if needed.

```
WS(CS^2)[0]: 16,0
WS(CS^2)[1]: 9,0
WS(CS^2)[2]: 36,0
WS(CS^2)[3]: 64,0
WS(CS^2)[4]: 81,0
WS(CS^2)[5]: 144,0
WS(CS^2)[6]: 169,0
WS(CS^2)[7]: 0,0
WS(CS^2)[8]: 0,0
```

Figure 4-13*: Shows the weighted support values.*

4.6.7 Binomial

The binomial function is used in experiments where we try to reproduce the support found. The significance provided here is a good estimate for the composite motifs importance.

```
SB[0]: 1,0
SB[1]: 0,0
SB[2]: 1,906517099708663E-4
SB[3]: 7,631891688705638E-6
SB[4]: 0,0012801176037734584
SB[5]: 7,444544090325077E-5
SB[6]: 0,019976473968959022
```

Figure 4-14: *Shows the output from the binomial calculations.*

4.6.8 Experimental significance

In addition to the binomial trials, an experiment with up to 100 000 cycles randomly samples the motifs significance in a set of 1000 sequences. An occurrence over the random significance value is counted. ST is then calculated as (every observed count)/(possible occurances).

```
ST[0]: 0,75609
ST[1]: 0,90623
ST[2]: 1,2E-4
ST[3]: 2,0E-5
ST[4]: 0,01834
ST[5]: 3,8E-4
ST[6]: 0,18308
ST[7]: 0,0
ST[8]: 0,0
```

Figure 4-15: *Shows the significance for composite modules based on experimentation.*

4.7 Outputfile (X+1)

Contain alignment data against the sequences for a chosen composite motif.
This file is used as input to programs that evaluate the quality of the predictions done.

4.7.1 Final results (alignment against target sequences)

In the final stage of the program, the best composite motif found with the binomial test is selected and matched against the target sequences, this produce a list like the one showed here:

```
Module 1:
D10051, 211, 896,aaatgcattattattattatg…cagaggtaaag, M00040, M00981, M00925
AF039399, 1656, 1663, gcaggga, M00040, M00981, M00925
A28223, 526, 1318, caggccaccactgcagctggaggtac…cagacag, M00040, M00981, M00925
V01523, 381, 530, cagctggcaggaagcaggtg…gcaaaacac, M00217, M00973, M00971
```

The file contains the following information:

- Module name
  - The sequence name
  - Start position (first motif hit)
  - End position (last motif hit)
  - Pattern located between start and end position
  - The single motifs which build up the composite pattern

5 Testing

5.1 Evaluation criteria

TransCompel [24] is the dataset used in all the tests, which is one of many well studied sets used by the bioinformatics group here at NTNU. The subset used is obtained from the TransCompel database consisting of composite regulatory elements affecting gene transcription [24] with associated sequences.

Finding good test criteria for a new method like this is a challenge, but by using test sets with known binding sites for the sequences, it is possible to some extent evaluate how well this method performs. By alternating input parameters and settings for all the different tests, they should match different characteristics in the sequences and produce variations in the output that will be evaluated on criteria comparable to assessments done for other common motif discovery tools, by Tompa et. al. [23]. This measures allow the newly predicted sites to be compared to what is already known, and we are especially looking at the correlation coefficient (nCC) calculated.

nCC is based on:

- TP: The number of positions in both predicted and known sites.
- FN: The number of positions in known, but not predicted sites.
- FP: The number of positions in predicted, but not known sites.
- TN: The number of positions in neither predicted and known sites.

And is calculated:

$$nCC = \frac{TP \cdot TN - FN \cdot FP}{\sqrt{(TP + FN)(TN + FP)(TP + FP)(TN + FN)}}$$

nCC is in the range -1 to +1, where +1 equals perfect alignment, and a lower value poor alignment.

Figure 5-1: *Shows the values and calculation used by the nCC value under investigation [23].*

In other tests, new deterministic motifs where generated from the sequences by Teiresias [18], with parameters as shown in [App. A]. These motifs provided me with the possibility to compare results against the probabilistic PWMs, and to test the significance values based on alignment in real background material instead of the uniform distribution. Here the comparison needed to be done more manually, to check the number of composite matches, the region they hit in and common scores for significance, support and the region they hit are checked.

Tests like these are easily biased by human factors or properties for the test set, so I run many subtest for each test with different parameters without any

special prediction on the data, and use this to choose the best subtest for further testing.

The tests are done both with and without pruning to compare running time and results.

Two separate tools will be used to run all the tests. A *run centre* and script will produce results with desired parameters, and a python script provided by the bioinformatics group will test for correlation with the TransCompel set.

5.2 Test centre used to run tests

To run the number of tests needed, a program was written in Visual Basic to manage input, parameters and variants included in the test.

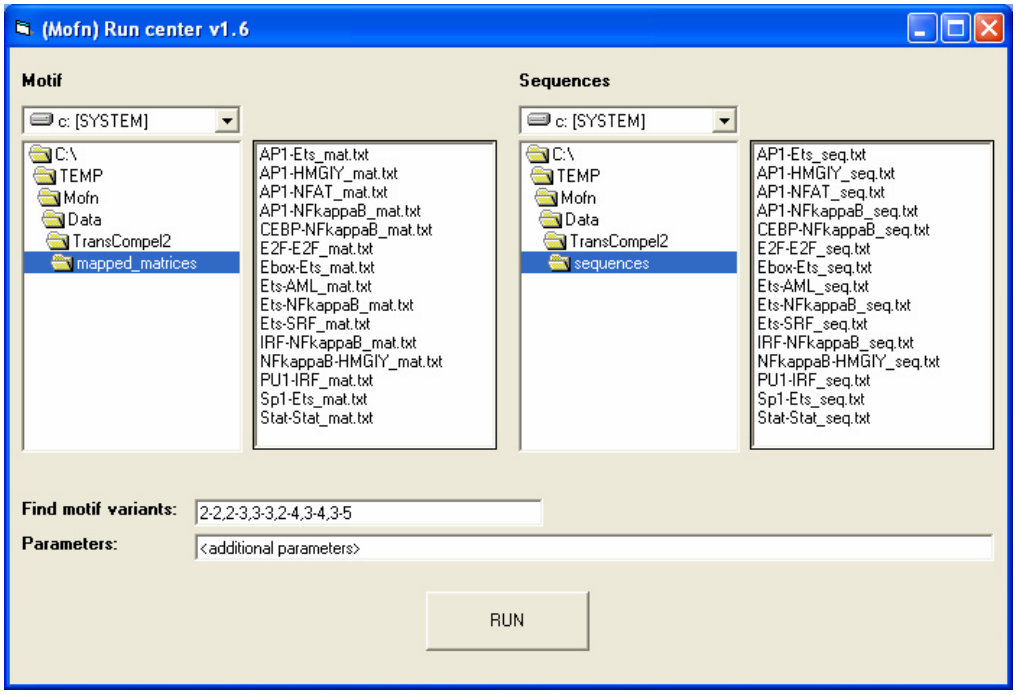


Figure 5-2*: Shows the Run centre used to run tests with complete dataset for many motif variants with the possibility for additional parameters.*

The interface needed between the *run centre* written in Visual Basic and Mofn in Java, is a script batch file created for the right number of parameters N.

```
Java Mofn %1 %2 %3 %4 … %8
```

Figure 5-3*: Shows the content of script.bat.*

For some small and standalone tests, where output is not tested as sets, a lighter *test* script was used instead.

```
java Mofn -mf Sp1-Ets_mat.txt -sf Sp1-Ets_seq.txt -bf Unique_5000.fas -m 2 -n 3
```

Figure 5-4*: Shows the content of test.bat*

.

5.3 Nucleotide-level scores

This test-script to measure the nucleotide correlation coefficient and other values is provided by the bioinformatics group at NTNU. It contains 3 Pythons files:

- Bulk_Score.py
- GraphGen.py
- ComputeScores.py

Python, Scipy, pylab and Graphgen are packages that needs to be installed to run this script.

```
Python BulkScore.py testsetfile outputfile
```

Figure 5-5*: Shows how to use the script.*

The testset file contains links to all the datasets needed to calculated nCC.

```
<predicted data> <known data> <matrix used> <related sequence>
<predicted data> <known data> <matrix used> <related sequence>
<predicted data> <known data> <matrix used> <related sequence>
<predicted data> <known data> <matrix used> <related sequence>
```

Figure 5-6*: Shows how the testset files is used to link the datasets used by the script. One line is used for each test set in a test.*

5.3.1 Script output

The script produces 3 different outputs. (i) The first output shows a graph (Figure 5.7) for different nucleotide-level scores, where nCC is one of the value most important for us to test. (ii) Shows similar values calculated for the motif level. (iii) At the end, a text file with all the background calculations is printed. This make it easy to further evaluate the values found..

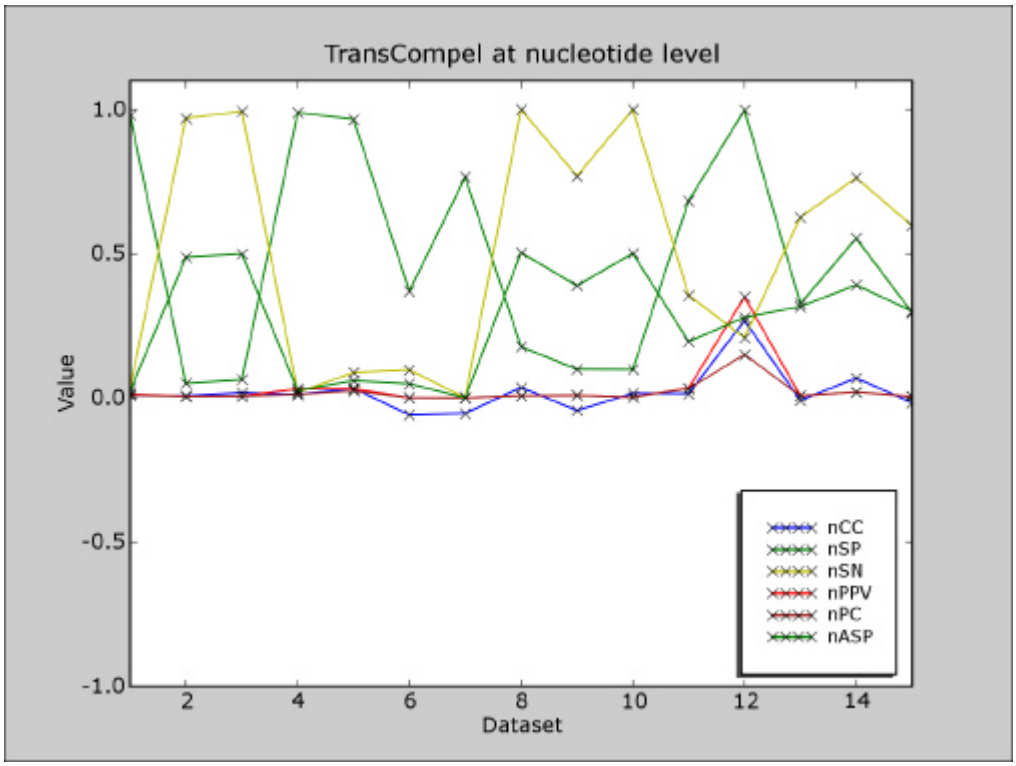


Figure 5-7: *Shows different nucleotide measures plotted for every sequence. The nCC curve (blue) is the one we focus on.*

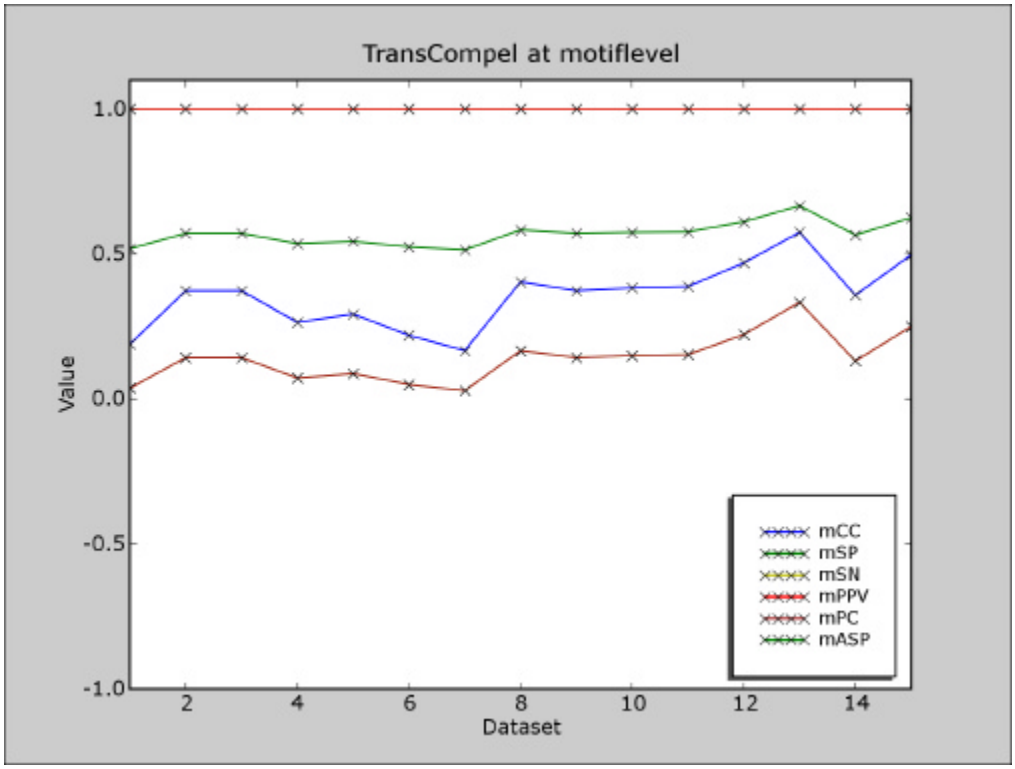


Figure 5-8: *Represents the same quality on the motif level.*

6 Results

This chapter presents a diverse selection of tests. The most important is the nucleotide-level score. The most important elements observed are shown here, while the complete datasets used and corresponding results can be found in the attach ZIP file [App. G].

6.1 Nucleotide-level scores

To best evaluate the nCC found, we calculate the observed values order by the motif variants, background model, input parameters used and corresponding sequences. This can then be compared against what other motif discovery tools achieved [32].

nCC	2 of 2	2 of 3	3 of 3	2 of 4	3 of 4	3 of 5
UF	0,042238	0,021718	0,047943	-0,023871	0,004129	-0,049502
BG	0,032191	0,028200	0,045786	-0,027099	0,016000	-0,045525

Table 6.1: *Shows how the nCC is distributed over the different motif variants (1. row) and the background model used (UniForm or BackGround), with strict parameter settings. The 4 best in every sequence is chosen before a cut-off at 20 percent decide the threshold used.*

nCC	2 of 2	2 of 3	3 of 3	2 of 4	3 of 4	3 of 5
UF	0,033575	0,177916	0,032377	0,016488	0,023416	-0,023696
BG	0,044112	0,299830	0,035947	0,016377	0,015724	0,011981

Table 6.2: *Shows how the nCC is distributed over the different motif variants (1. row) and the background model used (UniForm or BackGround), based on less strict parameters. The 10 best in every sequence is chosen before a cut-off at 30 percent decide the threshold used.*

These observations give an average score for the entire test with this composite motif discovery tool:

Mean nCC = 0,032344

Table 6.3: *Mean nCC for this Mofn method.*

6.2 Common single motifs

All the single motifs that occurred in a composite motif discovered in the TransCompel dataset are listed below together with the total number of times they contributed. The count is over all the various settings tested.

41 M00040	13 M00174	6 M00430
31 M00926	11 M00173	6 M00931
31 M00981	11 M00939	5 M00738
30 M00041	10 M00051	5 M00751
29 M00801	10 M00736	4 M00726
25 M00054	8 M00050	3 M00217
23 M00925	8 M00739	3 M00223
23 M00971	8 M00973	3 M00492
18 M00208	7 M00731	2 M00737

16 M00052 16 M00743	7 M00933 6 M00053	2 M01029 1 M00658

Table 6.4: *Show all the single motifs participating in the composite modules. And the number of occurrences found.*

6.3 PWM compared to standard motifs

The motif representation selected, should fit the problem we are trying to solve. I have done several tests, neither dominates the other clearly.

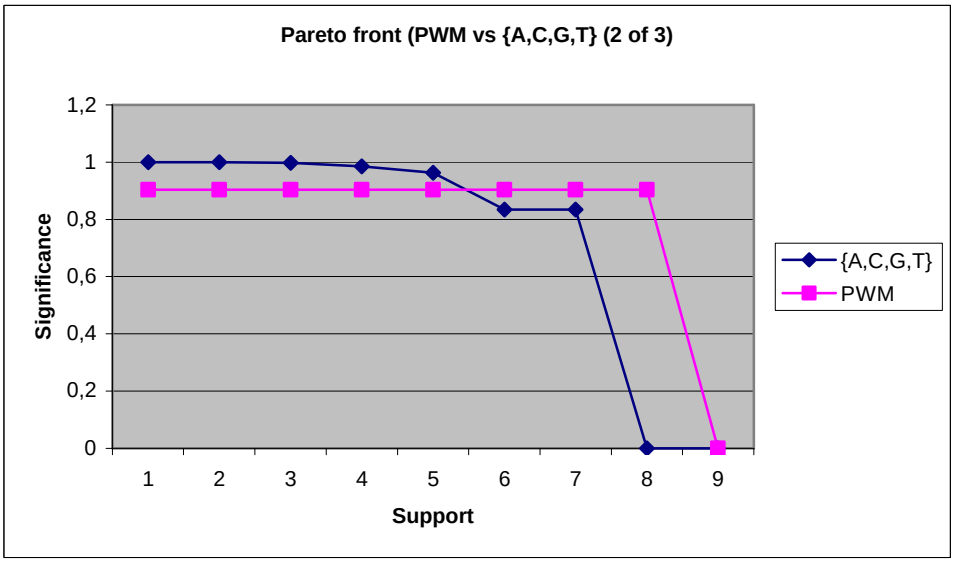


Figure 6-1: *Compares the PWM representation against {A,C,G,T}.*

6.4 Background test

Testing and comparing the significance for motifs (not PWM) find by the uniform distribution and alignment against real background sequences.

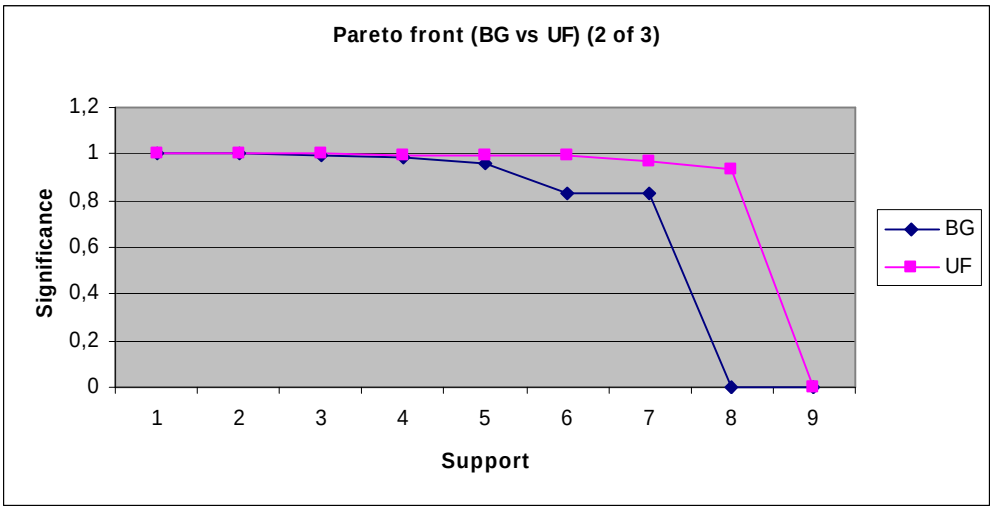


Figure 6-2: *Compare a 2 of 3 run based on {A,C,G,T} based on background distribution (UniForm or BackGround).*

6.5 Manual alignment

If additional investigation is needed, alignment against sequences can be used to verify hits in different regions.

Module 1: D87541, 565, 879, ccttctcctcttcc ..... gaagtcgccgg, cacacacac, tcctcttcct, ccttctcct M60058, 242, 250, cacacacacctcctcc, cacacaca, tcctcttcct, ccttctcct D13784, 6723, 6956, ccttctcctgc....ctgcgccgtgccttc, cacacacacaca, tcctcttcct, ccttctcct

Table 6.5: *Shows 3 motifs combined with the sequence they occur in. The sequences are cut of due to space limitations.*

6.6 Different cut-off value

After the significance for every motif is calculated, the support is found by keeping every hit position in every sequence over a given threshold set by the cut off value in percent. Through test, I have found that by using 10 percent, gave a more exact composite pattern then 20 and 30.

```
Threshold: 0.029999999999999995 Percent: 10
Pareto[2]: 0,99130279314752
Pareto[3]: 0,99130279314752
Pareto[4]: 0,0
Pattern[3]: M00933          M00971 M00743
CS[2]: 0
CS[3]: 8
CS[4]: 0
SB[3]: 3,565572059147113E-5
ST[3]: 1,9E-4
```

Figure 6-3: *Shows part of output from a run with 10 percent cut-off.*

```
Threshold: 0.004409265108473676 Percent: 20
Pareto[4]: 0,9809432758528
Pareto[5]: 0,965007864704
Pattern[4]: M00933          M00931 M00743
Pattern[5]: M00933          M00931 M00971
CS[4]: 11
CS[5]: 13
SB[4]: 8.682188650633713E-6
SB[5]: 2.6885414338786874E-6
ST[5]: 3.0E-5
```

Figure 6-4: *Shows part of output from a run with 20 percent cut-off.*

6.7 Pruning

Pruning showed to be very effective. On several test, the algorithm used about 5-10 % of the original time estimated. Some motifs with significance 0.0, can however dominate the search and affect the pruning negatively.

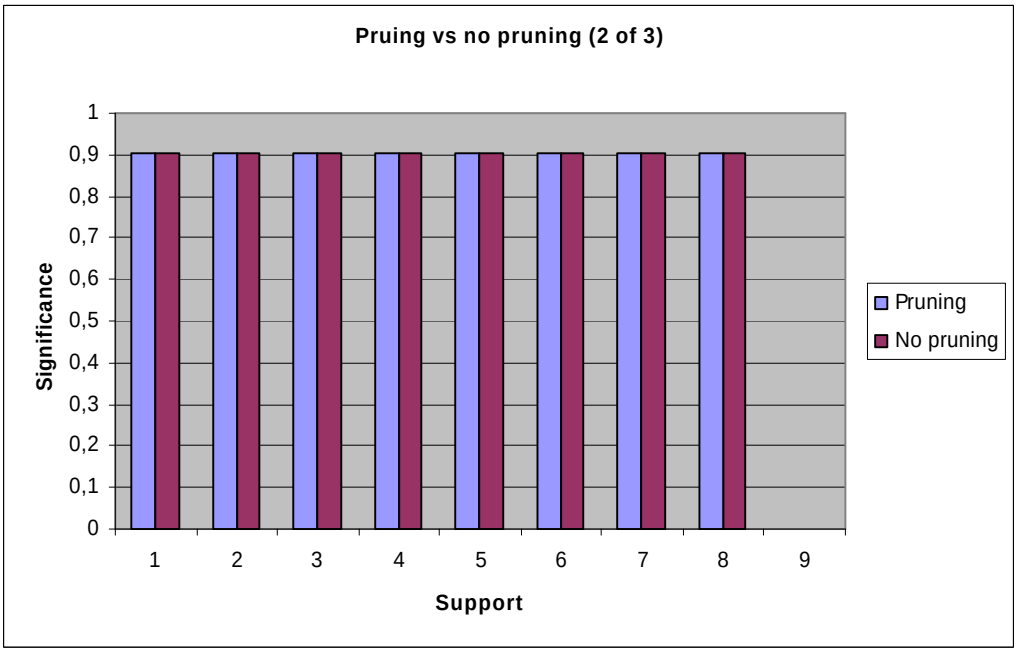


Figure 6-5:*Shows how 2 different run with and without pruning end up with the same solution. The only difference is in the time used, calculation time was drastically reduced.*

6.8 Results from 3 of 5

Since the method now supports any combination of m and n, it is possible to investigate motif variants like 3 of 5. The result can be seen below.

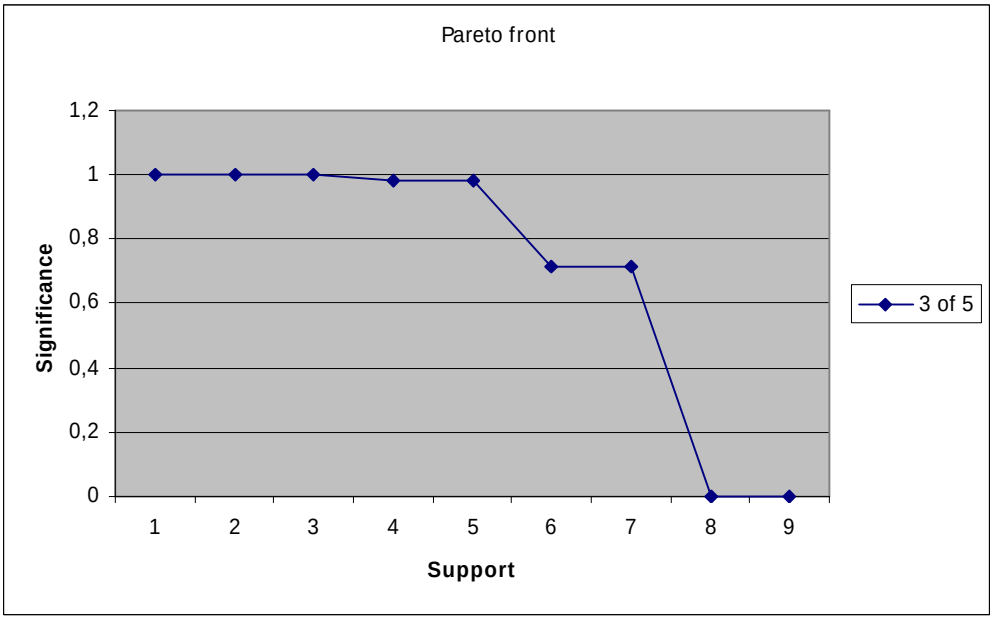


Figure 6-6: *Shows the different solution found in a 3 of 5 run.*

To find out more about this motif variant, the other support model is presented.

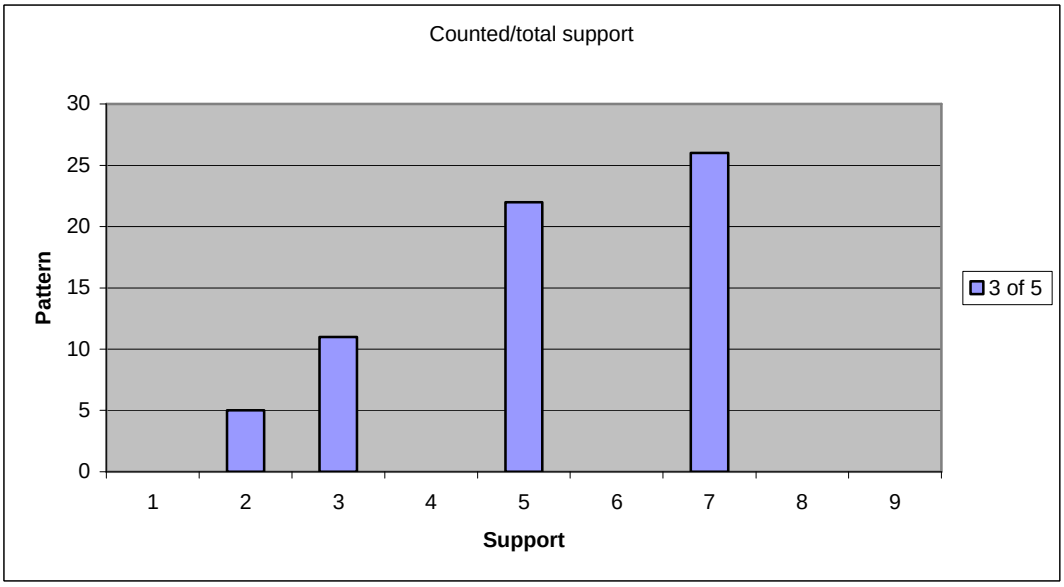


Figure 6-7: *Shows how great support all the single motifs participating in the module actually have.*

7 Discussion

After testing this algorithm, the values found can be separated into two different categories. (i) Either testing the computational parts of the algorithm, and we found the different parts here to work quite well. (ii) How this method performs on real biological data is however what a future scientist is concerned about. By testing the method on the TransCompel data set, we proved that this algorithm could discovery approximate composite motifs. Most of the results observed, where quite similar to what Tompa et. al. found [23].

The different motif variants we looked for, seemed to have a greater impact on the result, then if the motifs where based on the uniform or real background distribution or if the motifs where represented by PWM, IUPAC or {A,C,G,T}.The test results could be biased by human factors, or special properties in the dataset.

Pruning worked as expected. While other ways to increase the power, is to utilize the client-server model implemented. Even if some of the tests scored a little under expected estimations, the ideas are proven and the algorithm is found to works. Which the great flexibility this method offer in both pattern combination and output, I hopefully it will become a well tested and applied method.

8 Further work

This project has come a long way now, but there might and will almost always emerge new needs during its use, so that further development of this m of n method will probably occur.

8.1 Adding additional models to the search

The enumeration that takes place during DFS can be modelled to handle support, various weighting of motifs and constraints in different ways. Starting on a task like that can be to undertake a lot of work, but would surely be something interesting to apply here.

I know that a project to add distance constraints for GCMD [2] where done by another student.

8.2 More testing

Evaluating tools like this is a hard problem, and further work should be done to check if the results found in TransCompel really where representative for this method.

8.3 What the future brings

The ongoing work of unrevealing the information hidden in DNA and proteins is really exciting, so I hope both IDI and other scientist around the world working with similar problems will find great use for this method and application, and that they might continue to develop it.

Acknowledgements

I would like to thank my coach Geir Kjetil Sandve for his help and guidance throughout this project. Jostein Johansen for providing a test script for my results, it was greatly appreciated. Finn Drabløs for giving me more insight into the underlying biology and Magnus Lie Hetland for advices in statistics.

The last and kind regards go to my fellow students for their participation and feedback during meetings organized for our research group.

And thanks to any future user of this method, I hope it helps you in your work.

References

[1] V. Valebjørg; “A method for discovery of M-of-N de novo Motifs”, 2005

[2] G. K. Sandve, F. Drabløs: Generalized composite motif discovery

[3] J. G. Bjålie, E. Haug, O. Sand, Ø. V. Sjaastad, Med. Ill. K. C. Toverud. ”Menneskekroppen – fysiologi og anatomi”. Gyldendal, 2003

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, “DFS, search tree, divide and conquer, running time of an algorithm.”, Introduction to Algorithms 2nd edition,, MIT Press, 2001.

[5] R. E. WalPole, R. H. Myers, S. L. Myers: “Probability and statistics”, 6th edition, Prentice-Hall 1998.

[6] S. Sinha, “Composite motifs in promoter regions of genes: models and algorithms”, Generals Report, University of Washington.

[7] PWM, http://en.wikipedia.org/wiki/Position_weight_matrix

[8] A. E. Kel, E. Gößling, I. Reuter, E. Cheremushkin, O.V. Kel-Margoulis and E. Wingender: *”Match^{tm}: a tool for searching transcription factor binding sites in DNA sequences”*, 2003.

[9] R. Elkon, C. Linhart, R. Sharan, R. Shamir and Y. Shiloh: *“Genome-Wide In Silico Identification of Transcriptional Regulators Controlling the Cell Cycle in Human Cells”*, CSHLP, 2003.

[10] CpG islands, http://en.wikipedia.org/wiki/CpG_island

[11] Human Genome Project, Sequencing. http://www.ornl.gov/sci/techresources/Human_Genome/faq/seqfacts.shtml

[12] Baldi and Brunak; Bioinformatics, 2nd edition, MIT press 2001.

[13] Gary D. Stormo, “DNA binding sites: representation and discovery”

[14] RNA polymerase II example for the TTR control region (promoter) http://departments.oxy.edu/biology/Stillman/bi221/110300/rna_polymerases.htm

[15] Transcripton Process, http://en.wikipedia.org/wiki/Transcriptional_regulation

[16] G.K. Sandve, F. Drabløs; “*A survey of motif discovery methods in an integrated framework*”, 2006.

[17] Rigoutsos, I. and Floatos A., “Combinatorial Pattern Discovery in Biological Sequences: The TEIRESIAS Algorithm”, Bioinformatics, 14(1):55–67, 1998.2

[18] Bioinformatics & Pattern Discovery at IBM, The Home of Teiresias: http://cbcsrv.watson.ibm.com/Tspd.html

[19] IUPAC table http://prodoric.tu-bs.de/vfp/vfp_help.php

[20] Platform independent programming language: Java http://java.sun.com/

[21] Petroutsos, Hough: “Visual Basic 5 – Developer’s handbook”, 1996

[22] M. Fowler, K. Scott: “UML Distilled”, 2nd Edition, 1999

[23] M. Tompa, N. Li, B. L. Baily, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M.C.Frith, Y. Fu, W. J. Kent, V.J. Makeev, A. A. Mironov, W.S. Noble, G. Pavesi, G. Pesole, M. Régnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, X. Weng, C. Workman, C. Ye and X. Xhu. “Assessing computational tools for the discovery of transcription factor binding sites”. Nature Biotechnology, 23:137-144, 2005.

[24] TransCompel database, http://www.gene-regulation.com/pub/databases/transcompel/compel.html

[25] Gene structure, “NCBI Advanced Workshop for Bioinformatics Information Specialists”, http://www.ncbi.nlm.nih.gov/Class/NAWBIS/

[26] G. Stolovitzky, A. Califano; “Statistical Significance of Patterns in Biosequences”, 1998

[27] M. Spingola, L. Grate, D. Haussler, M. Jr. Ares; “Genome-wide bioinformatic and molecular analysis of introns in Saccharomyces cerevisiae.”

[28] Background distribution, Munich information center for protein sequences, http://mips.gsf.de/proj/yeast/reviews/intron/F4.html

[29] Information about the Pareto front usage and Pareto optimality. http://en.wikipedia.org/wiki/Pareto_optimality

[30] Branch-and-bound and pruning; http://en.wikipedia.org/wiki/Branch_and_bound

[31] K. H. Rosen; “Discrete Mathematics and Its Applications” 4th Edition, 1999

Appendix A – Output from Teiresias

Output from Teiresias or any other third-party discovery tool can be a list of motifs as shown below.

1. column: Total hits in every sequence
2. column: Total sequences with hits in them. (Distinct)
3. column: Motif name - IUPAC representing the pattern.
4. column: Alternating sequence number and hit position

```
############################################################
#                                                          #
#                      FINAL RESULTS                       #
#                                                          #
############################################################
7         5         gcctcct 1 1562 2 2405 3 127 4 444 4 6214 4 7996 6 471
7         5         cagcctg 0 370 1 657 2 735 4 2800 4 7953 4 7981 5 92
7         5         ctcagct 0 2136 1 1701 2 2837 3 1685 4 3622 4 4561 4 7493
7         5         tcttctt 0 1930 1 363 1 546 3 479 3 1322 4 7328 6 1261
7         5         ctgtgct 0 611 1 402 1 1125 1 1602 2 1909 3 795 4 2365
7         5         aaccaaa 0 74 1 853 2 835 2 1267 4 1821 7 27 7 132
7         5         accagcc 0 2078 0 2335 2 1456 2 2805 3 1078 4 2208 6 1142
7         5         cctggcc 0 373 1 651 1 1537 3 1005 4 6149 4 7789 6 1067
7         5         gaagctg 0 2527 2 1589 2 2147 2 3690 3 363 4 5045 6 1235
7         5         gcaacca 1 119 2 296 2 5069 3 1075 4 4361 4 6666 6 1139
7         5         ttctttc 0 1366 0 2497 1 548 2 209 2 4570 3 1324 4 6820
7         5         atttttt 0 1078 1 232 2 888 2 3530 4 6815 4 7812 6 1529
6         5         gttccac 1 1099 2 4053 3 1600 4 624 4 8006 6 1287
6         5         agaaaaa 0 411 1 968 2 1294 4 5161 4 7871 7 104
6         5         gagccag 0 1667 1 1642 2 3197 3 1622 4 702 4 2011
6         5         tgctcca 0 300 0 1892 2 4244 3 514 4 4611 6 124
6         5         ttctttttt 0 2065 1 1331 2 861 3 1461 3 1475 6 1428
6         5         gacctgg 0 109 2 2075 3 220 3 679 5 10 6 1065
6         5         ccaaaac 0 70 0 2199 1 855 2 1269 3 558 4 7718
6         4         ccaaaag 0 1789 2 3510 2 3858 3 1542 4 1018 4 4570
6         4         gaagcca 0 1572 2 3828 2 3897 3 1528 3 1812 4 199
6         4         gaaacag 0 2158 1 958 4 67 4 7260 4 8346 6 223
6         4         cttttttt 2 2618 2 4891 3 1451 3 1477 4 7014 6 1430
6         4         tttctttt 0 2064 1 1336 3 1460 3 1474 6 1427 6 1535
6         4         cagaagt 0 2357 1 1239 1 1291 3 71 4 71 4 8350
6         4         cttcctt 0 675 1 583 3 483 4 6960 4 6964 4 8197
6         4         ctgtttt 1 433 2 1440 2 2365 4 3908 4 6380 6 262
6         4         ctctttc 0 686 0 1217 1 1515 4 3873 4 7000 6 1446
6         4         ccacagg 0 1302 0 1675 1 123 2 3589 4 5027 4 7900
6         4         ctcttct 0 1929 3 573 3 1321 4 7028 4 7126 6 2043
6         4         ctctcca 0 628 1 49 3 940 4 543 4 729 4 4043
6         4         ctattat 0 2038 2 1476 4 3498 4 4793 4 5474 6 1396
6         4         cgccccgc 1 1609 1 1722 1 1836 3 974 4 1090 6 871
6         4         cctggagg 1 1299 2 2871 4 1608 4 2638 4 7091 6 622
6         4         acaggca 0 777 0 873 0 1304 2 1466 4 7247 6 1988
6         4         aaccaga 0 545 2 3782 2 5034 4 2006 4 2595 7 81
6         4         aacagcc 0 591 0 2160 2 733 2 2436 4 7262 6 409
6         4         tttttgc 2 1356 2 3434 3 1371 3 1453 4 7814 6 1440
6         4         agcccac 0 693 3 1081 4 1991 4 2818 4 7301 6 1145
6         4         acagcct 0 2161 1 1435 2 734 2 2437 2 3662 4 7980
```

Appendix B - IUPAC

The IUPAC Code is a set of symbols encoding each ambiguous subset of the four nucleotides “ACGT“ in DNA code. Below is the complete list of these symbols together with the subset they encode [19]. All motifs with patterns represented with IUPAC codes, will be translated into the corresponding class in the program.

Code	Corresponding class	Description
A	A	Adenine
C	C	Cytosine
G	G	Guanine
T (or U)	T	Thymine (or Uracil)
R	[A,G]	Purine
Y	[C,T or U]	Pyrimidin
K	[G,T or U]	Keto
M	[A,C]	Amino
S	[G,C]	Strong (3 H-bonds)
W	[A,T or U]	Weak (2 H-bonds)
B	[G,T or U,C]	not A
D	[G,A,T or U]	not C
H	[A,C,T or U]	not G
V	[GCA]	not T, not U
N	[ACGT or U]	Any
. or -	“ “	Gap

Appendix C – Glossary from various chapters

Amino acid	Biochemical building blocks. Some are made in us, others are added through food sources.
Antagonistic	Category on how compsite patterns affect each other (interference).
Binding site	Pattern in the regulatory region of a gene
DFS	Depth First Search – a way of traversing a tree-structure, where each leaf-node from left to right are visited first
DNA	Deoxyribonucleic acid
Dummy motif variant	Fake element created outside limits to aid the enumeration process.
First motif variant	Fake element created outside limits to aid the enumeration process.
Gene	A template for constructing a protein chain.
Motif	A string with letters from the alphabet used.
Pareto front	Representation of solution in a problem based on two variables.
Promoter	A DNA location used under transcription. RNA polymerase connects to the promoters before starting the transcription process.
Promoter region	See promoter region.
Protein	A peptide-bond of amino acids used for different functions in organisms (hormones/enzymes etc.).
PWM	Position Weight Matrix, used to represent motifs.
Significance	Likelihood for a motif to occur in a sequence. Log values are often used, this fix variable scale problems, outliers, negative values etc.
Support	A count of an objects frequency in a set.
Synergistic	Category on how some compsite patterns affect each other (co-operatively).
TF (Transcription Factor)	See motif.

Appendix D – Sequences from TransCompel

Below is a subset of sequences taken from the TransCompel dataset. The sequences are from the Sp1-Ets_seq.txt file [App. I]. Every sequence is represented in the FASTA format.

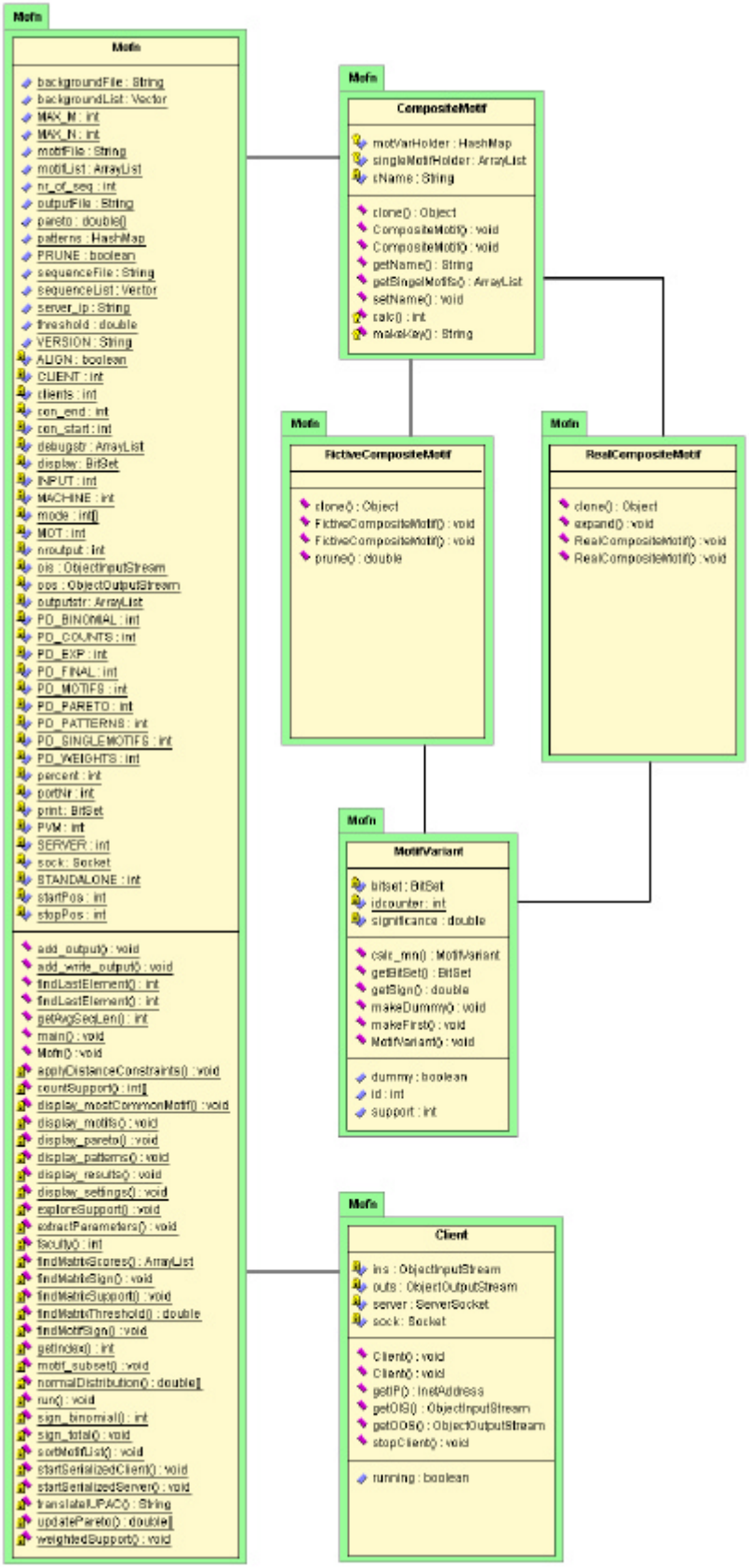
The sequences in TransCompel where provided with corresponding PWM motifs, but an alternate motif representation where made by using Teiresias on the same dataset for comparison between the different representations. The following settings where used in Teiresias:

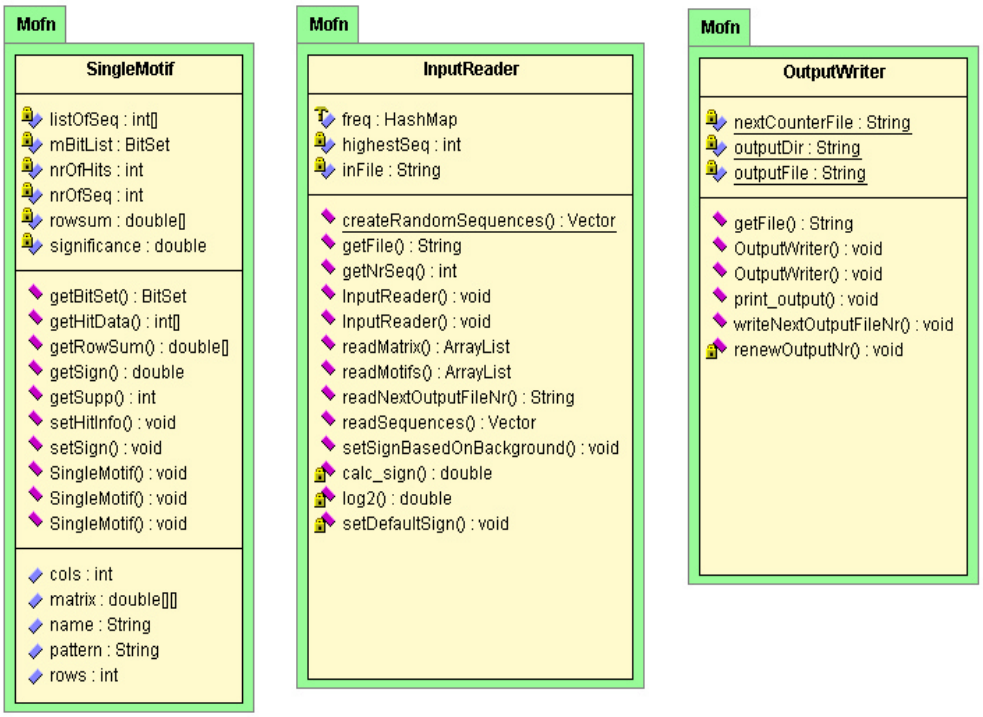
Exact Discovery: true
Only amino acid characters: true
Max Brackets: 0
L: 7
W: 7
K: 5
Q: 25

```
>M84757
taagatgagttggaaaataatgagggagggaccatcccgaccctggatttga
aggatagtttggattttcccaaaaccaaacttgccaacaggttgtgtactgc
aggtagacctggattagtgacctaagtttcatcttatcttgaggagagtgaa
ctttgatataagggctcagggaatcccagctcctttgttccttcccatgttg
gctggagagaagggattccccatgatagccagggtcttcagggcctactgtc
acaatgtacagaataaatggatggactaaaagggccaggttgctccaaggcc
gaatgtgaccattcttcccagagttcactgctccgaactctccttcacagac
tgaggtcagcctggcccctatgttgtttaccctgagttcctccttggagaaa

>D87541
ttttcagaaagagatgcagagaacacagggagtgcacagaacattcttgctc
tccacgaatcattgtctgcctcttaaaccccacattctgatcccaaatggtg
acaaggaatgaaagagcaaccacagggtcttgtgtataagtatccacacagg
atttcctgtctctgaggtggcttttacataggtattagcctgtggctgataa
gatctcccaacagttttgctgcttatttttacattttaaaaatcaacttgc
ttaacccaactaaggctatccacacccccacacatgattacttatgttatga
gatgaatgcatgctgacactgaggtgtttttctgctgtgtgagacagggat
cttcttatgttgactgggctcaggggggcctactgcctctgtgcttagtctg

>J02275
atttttagaactgaccaaccatgttcacgtaagtgacgtgatgacgcgcgct
gcgcgcgcgccttcggacgtcacacgtcacttacgtttcacatggttggtca
gttctaaaaatgataagcggttcagggagtttaaaccaaggcgcgaaaagga
agtgggcgtggtttaaagtatataagcaactactgaagtcagttacttatct
tttctttcattctgtgagtcgagacgcacagaaagagagtaaccaactaacc
atggctggaaatgcttactctgatgaagttttgggagcaaccaactggttaa
aggaaaaaagtaaccaggaagtgttctcatttgtttttaaaaatgaaaatgt
```

Appendix E – UML

UML diagrams for the different classes in the Mofn-package:




Mofn
SingleMotif
listOfSeq : int[]
mBitList : BitSet
nrOfHits : int
nrOfSeq : int
rowsum : double[]
significance : double
getBitSet() : BitSet
getHitData() : int[]
getRowSum() : double[]
getSign() : double
getSupp() : int
setHitInfo() : void
setSign() : void
SingleMotif() : void
SingleMotif() : void
SingleMotif() : void
cols : int
matrix : double[][]
name : String
pattern : String
rows : int
Mofn
InputReader
freq : HashMap
highestSeq : int
inFile : String
createRandomSequences() : Vector
getFile() : String
getNrSeq() : int
InputReader() : void
InputReader() : void
readMatrix() : ArrayList
readMotifs() : ArrayList
readNextOutputFileNr() : String
readSequences() : Vector
setSignBasedOnBackground() : void
calc_sign() : double
log2() : double
setDefaultSign() : void
Mofn
OutputWriter
nextCounterFile : String
outputDir : String
outputFile : String
getFile() : String
OutputWriter() : void
OutputWriter() : void
print_output() : void
writeNextOutputFileNr() : void
renewOutputNr() : void

Appendix F - Source code

Shows source code for all classes in the Mofn package.

Mofn.java

```
import java.util.Vector;
import java.util.ArrayList;
import java.util.BitSet;
import java.util.StringTokenizer;
import java.util.HashMap;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.util.Arrays;
import java.util.TreeMap;

import java.io.*;
import java.net.*;

/**
 * This class controlles the M of N application.
 * Starts standalone, server or client process.
 * Handles input (motifs, sequences), output (display, files)
 * @author Vetle Valebjørg
 * @version 1.0
 * @see CompositeMotif, RealCompositeMotif, FictiveCompositeMotif,
SingleMotif, MotifVariant, Client, InputReader, OutputWriter
 */
public class Mofn{
	//CONSTANTS
	public static final String VERSION = "2.07.2006";
	private static final int MACHINE = 0;
	private static final int INPUT = 1;
	private static final int PVM = 1;
	private static final int MOT = 0;

	private static final int STANDALONE = 0;
	private static final int CLIENT = 1;
	private static final int SERVER = 2;

	private static final int PD_MOTIFS = 0;
	private static final int PD_PARETO = 1;
	private static final int PD_PATTERNS = 2;
	private static final int PD_SINGLEMOTIFS = 3;
	private static final int PD_COUNTS = 4;
	private static final int PD_WEIGHTS = 5;
	private static final int PD_BINOMIAL = 6;
	private static final int PD_EXP = 7;
	private static final int PD_FINAL = 8;
	private static int nroutput = 9; //change if list above
increases

	private static BitSet display = new BitSet(nroutput);
	private static BitSet print = new BitSet(nroutput);

	//GLOBAL VARIABLES SET BY PROGRAM AND PARAMETERS
	private static Socket sock;
	private static ObjectInputStream ois;
	private static ObjectOutputStream oos;
	private static int portNr = 7777;
	private static int[] mode = {STANDALONE,PVM};
	private static int clients = 0;

	private static ArrayList outputstr = new ArrayList();
	private static ArrayList debugstr = new ArrayList();
```

```
	//list of all motifs from input-file, sorted after significance

	public static ArrayList motifList;
	public static Vector sequenceList;
	public static Vector backgroundList;

	public static boolean PRUNE = false;
	private static boolean ALIGN = false;
//default number of sequences the input motif are retrieved from
	public static int nr_of_seq;;    public static double[] pareto;
//list of significance for given support-index
	public static HashMap patterns = new HashMap();
	public static double threshold = 0.0;
	private static int startPos = 0; //startposition in the
motifList
	private static int stopPos = 0;  //stopposition in the
motifList
	private static int con_start = -1;
	private static int con_end = -1;

	//VALUES THAT CAN BE CHANGED
	public static int MAX_N = 2; //max combinations N = 3 finds m/3
(0<=m<=3)
	public static int MAX_M = 2; //number of m=goal, 0 = N 1 = N-
1/N and so on.
	private static int percent = 30;
	public static String server_ip = "10.0.0.11";

	public static String sequenceFile =
"data//TransCompel2//sequences//ap1-ets_seq.txt";
	public static String motifFile =
"data//TransCompel2//mapped_matrices//ap1-ets_mat.txt";
	public static String backgroundFile =
"";//data//Unique_5000_1000.fas";
	public static String outputFile = "output"; //default
outputfile


	/**
	* Constructor with GUI if wanted
	*/
	public Mofn() {
	/*	addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				dispose();
				System.exit(0);
			}
		});
		*/
	}

	/**
	* Parses the arguments provieded to Mofn and sets variables
accordingly
	* @param args parameters used to control the program
	*/
	private static void extractParameters(String[] args){
		int machinemode = STANDALONE;
		int inputmode = PVM;
		print.set(0,nroutput);
		display.set(0,nroutput);
		int parameter = 0;

		while(parameter<args.length){
			if(args[parameter].equals("-mf")){
				parameter++;
				motifFile = args[parameter];
```

```
                    }else if(args[parameter].equals("-type")){
                           parameter++;
                           if(args[parameter].equals("pwm")){
                                  inputmode=PVM;
                           }else{
                                  inputmode=MOT;
                           }
                    }else if(args[parameter].equals("-sf")){
                           parameter++;
                           sequenceFile = args[parameter];

                    }else if(args[parameter].equals("-of")){
                           parameter++;
                           if(args[parameter].indexOf(".")!=-1){
                                  outputFile =
args[parameter].substring(0,args[parameter].indexOf("."));
                           }else{
                                  outputFile = args[parameter];
                           }

                    }else if(args[parameter].equals("-bf")){
                           parameter++;
                           backgroundFile = args[parameter];

                    }else if(args[parameter].equals("-m")){
                           parameter++;
                           MAX_M = Integer.parseInt(args[parameter]);

                    }else if(args[parameter].equals("-n")){
                           parameter++;
                           MAX_N = Integer.parseInt(args[parameter]);

                    }else if(args[parameter].equals("-mode")){
                           parameter++;

                           if(args[parameter].equals("SERVER")){
                                  machinemode = SERVER;
                           }else if(args[parameter].equals("CLIENT")){
                                  machinemode = CLIENT;
                           }else
if(args[parameter].equals("STANDALONE")){
                                  machinemode = STANDALONE;
                           }

                    }else if(args[parameter].equals("-cutoff")){
                           parameter++;
                           percent =
Integer.parseInt(args[parameter]);

                    }else if(args[parameter].equals("-ip")){
                           parameter++;
                           server_ip = args[parameter];

                    }else if(args[parameter].equals("-port")){
                           parameter++;
                           portNr = Integer.parseInt(args[parameter]);

                    }else if(args[parameter].equals("-print")){
                           parameter++;
                           for(int
i=0;i<args[parameter].length();i++){

      if((args[parameter].substring(i,i+1)).equals("0")){
                                        print.clear(i);
                                  }
                           }
                    }else if(args[parameter].equals("-display")){
                           parameter++;
```

```
                        for(int
i=0;i<args[parameter].length();i++){

      if((args[parameter].substring(i,i+1)).equals("0")){
                                    display.clear(i);
                              }
                        }
                  }else if(args[parameter].equals("-clients")){
                        parameter++;
                        clients =
Integer.parseInt(args[parameter]);
                  }else if(args[parameter].equals("-start")){
                        parameter++;
                        con_start =
Integer.parseInt(args[parameter]);
                  }else if(args[parameter].equals("-end")){
                        parameter++;
                        con_end =
Integer.parseInt(args[parameter]);
                  }else if(args[parameter].equals("-prune")){
                        PRUNE = true;
                  }else if(args[parameter].equals("-align")){
                        ALIGN = true;
                  }

                  parameter++;

            }//while parameters left

            mode[MACHINE] = machinemode;
            mode[INPUT] = inputmode;
      }

      /**
      * This method is started at default by java
      * and analyses input parameters before it start run().
      * @param args list of parameters used to control the program.
      */
      public static void main(String args[]) {

            //DEFAULT RUN
            if(args.length!=1){

                  //PARAMETER RUN  NORMAL/CLIENT/SERVER

                  extractParameters(args);

                  //READING PARAMETERS
                  System.out.println("Reading in data...");

                  //PVM
                  if(mode[INPUT]==PVM){
                        InputReader ir = new
InputReader(sequenceFile);
                        sequenceList =
ir.readSequences(sequenceFile);

                        if(backgroundFile.equals("")){
                        }else{
                              backgroundList =
ir.readSequences(backgroundFile);
                        //
      ir.setSignBasedOnBackground(backgroundList);
                        }

                        motifList = ir.readMatrix(motifFile);

                        nr_of_seq = sequenceList.size()/2;
```

```
                        ArrayList scores =
findMatrixScores(sequenceList);
                        threshold = findMatrixThreshold(scores);

                        findMatrixSupport(scores,threshold);
                        if(backgroundFile.equals("")){
                              findMatrixSign(threshold,null);
                        }else{

      findMatrixSign(threshold,backgroundList);
                        }
                  }

                  //MOTIFS
                  if(mode[INPUT]==MOT){
                        InputReader myReader = new
InputReader(motifFile);
                        sequenceList =
myReader.readSequences(sequenceFile);

                        nr_of_seq = myReader.getNrSeq();

                        if(backgroundFile.equals("")){
                              motifList =
myReader.readMotifs(nr_of_seq);
                        }else{
                              backgroundList =
myReader.readSequences(backgroundFile);
                              if(ALIGN){
                                    motifList =
myReader.readMotifs(nr_of_seq);
                                    findMotifSign(backgroundList);
                              }else{

      myReader.setSignBasedOnBackground(backgroundList);
                                    motifList =
myReader.readMotifs(nr_of_seq);
                              }
                        }


                  }

                  //removing 0-supp motifs
                  for(int i=0;i<motifList.size();i++){
                        SingleMotif sm =
(SingleMotif)motifList.get(i);
                        if(sm.getSupp()==0){
                              motifList.remove(i);
                              i--;
                        }
                  }

                  //CLIENT/SERVER
                  if(mode[MACHINE]==CLIENT){

                        try{
                              startSerializedClient(server_ip);
                        }catch(Exception e){
                              e.printStackTrace();

                        }
                  }else if(mode[MACHINE]==SERVER){

                        try{
                              startSerializedServer(clients);
                        }catch(Exception e){
```

```
                        e.printStackTrace();

                }
        }else{
                //STANDALONE
        }

        if(mode[MACHINE]!=CLIENT){

                System.out.println("");
                add_write_output("");

                //display_motifs();
                if(mode[MACHINE]!=SERVER){
                        run();
                }

                //DO CHANGE TS FOR DISTANCE CONST)
                if(con_start!=-1 && con_end!=-1){

applyDistanceConstraints(con_start,con_end);
                }

                System.out.println();
                add_write_output("");

                //PRINT
                display_settings(args);

                System.out.println();
                add_write_output("");

                display_pareto();
                System.out.println();
                add_write_output("");

                display_patterns();
                System.out.println();
                add_write_output("");

                display_mostCommonMotif();
                System.out.println();
                add_write_output("");

                int[] cSupp = countSupport();
                System.out.println();
                add_write_output("");

                weightedSupport(cSupp);
                System.out.println();
                add_write_output("");

                int bestNr = sign_binomial(cSupp);
                System.out.println();
                add_write_output("");

                sign_total(cSupp);
                System.out.println();
                add_write_output("");

                OutputWriter ow = new OutputWriter();
                ow.print_output(outputstr);

                outputstr.clear();

                if(sequenceList!=null){

//display_results(sequenceList,bestNr);
```

```
                        }

                        int cutpos = 0;
                        if(motifFile.lastIndexOf("//")!=-1){
                              cutpos = motifFile.lastIndexOf("//");
                        }else{
                              cutpos = motifFile.lastIndexOf("\\");
                        }

                        OutputWriter ow2 = new
OutputWriter(motifFile.substring(cutpos+1,motifFile.length()-4)+"_" +
MAX_M + MAX_N+"_");
                        ow2.print_output(outputstr);


                        //ow2.print_output(debugstr);
                  }



            }else if(args.length==1){
                  //PRINT COMPLETE ARGUMENTLIST
                  System.out.println("Version: " + VERSION);
                  System.out.println("3 different running modes:
STANDALONE, SERVER and CLIENT");
                  System.out.println("Standard use: (-sf and -bf are
optional but recommended)");
                  System.out.println("STANDALONE: java Mofn -mf
<motiffile> -type <mot/pvm> -sf <sequencefile> -bf <backgroundfile> -
m <m> -n <n>");
                  System.out.println("SERVER: java Mofn -mf
<motiffile> -type <mot/pwm> -sf <sequencefile> -m <m> -n <n> -mode
SERVER -clients <nr>");
                  System.out.println("CLIENT: java Mofn -mf
<motiffile> -type <mot/pwm> -sf <sequencefile> -bf <backgroundfile> -
mode CLIENT -ip <xxx.xxx.xxx.xxx> -port 7777");
                  System.out.println("");
                  System.out.println("Parameterlist:");
                  System.out.println("-mf <motiffile>");
                  System.out.println("-sf <sequencefile>");
                  System.out.println("-bf <backgroundfile>, used to
alter significance for single motifs");
                  System.out.println("-of <outputfile>,
default='output.txt'");
                  System.out.println("-m <m> m<=n, default=2");
                  System.out.println("-n <n>, default=3");
                  System.out.println("-type <mot/pvm>,
default=pvm");
                  System.out.println("-mode
<STANDALONE/CLIENT/SERVER>, default=STANDALONE");
                  System.out.println("-ip <xxx.xxx.xxx.xxx> Server
IP address :");
                  System.out.println("-port 7777");
                  System.out.println("-cutoff <percent>,
default=20");
                  System.out.println("-print <111111111> 1 prints, 0
does not.");
                  System.out.println("-display <111111111> 1 prints,
0 does not.");
                  System.out.println("-start <distance start
position (integer)>");
                  System.out.println("-end <distance end position
(integer)>");
                  System.out.println("-prune, turns on pruning");
                  System.out.println("-align, use alignment instead
of frequencies for motifs (not PVM)");
```

```
		}

	}


	/**
	* Starts the DFS enumeration process.
	*/
	private static void run(){

		//PRINT ARRAY SIZE OF INPUT
		//System.out.println("Size input data: "+
motifList.size());

		System.out.println("Starting " + (MAX_M) + " of "+ MAX_N
+ " search...");
		//DEFAULT PARAMETERS, USE DEFAULT IF NOT SET
		//PRINT ARRAY OBJECTS

		//motif_subset(500);
		display_motifs();
		System.out.println();
		add_write_output("");

		//used by run()
		if(startPos==0){
			stopPos = motifList.size();
		}

		//Creating pareto list with size equal nr of seq
		pareto = new double[nr_of_seq+1];

		for(int i=0;i<=nr_of_seq;i++){
			pareto[i]=1;
		}


		//START COMPUTATION
		for(int i=startPos;i<stopPos;i++){

			long starttime = System.currentTimeMillis();

			//START DFS FOR EACH ROOT MOTIF
			RealCompositeMotif cm = new RealCompositeMotif();
			cm.expand(i);

			if(mode[MACHINE]==CLIENT){
				//UPDATE SERVER

				try{
					String command = "update";
					oos.writeObject(command);
					for(int p=0;p<pareto.length;p++){
						oos.writeObject(new
Double(pareto[p]));
					}

	if(ois.readObject().equals("update")){
						for(int
p=0;p<pareto.length;p++){
							pareto[p] =
((Double)ois.readObject()).doubleValue();
						}
					}
				}catch(Exception e){
					e.printStackTrace();
				}
			}
```

```
                    long elapsed = (System.currentTimeMillis()-
starttime)/1000;

                    //add_output(i + " Time used: " + (elapsed) + "
Time left (approx): " +(motifList.size()*elapsed)/120 + " min");//-
starttime/1000));
                    System.out.println(i + " Time used: " + (elapsed)
+ "    Time left (approx): " +(motifList.size()*elapsed)/120 + "
min");//-starttime/1000));

             }

      }

      /**
      * Adds debug-strings to list that can be printed
      * @param output string that should be added to the debug
output.
      */
      public static void add_output(String output){
            debugstr.add(output);
      }

      /**
      * Adds output for printing to file to an ArrayList
      * @param output string that should be added to the program
output.
      * @see OutputWriter
      */
      public static void add_write_output(String output){
            outputstr.add(output);
      }

      /**
      * Returns avg sequence length calculated from current sequence
list
      * Used by inputreader.
      * @return average the average length on sequence strings.
      * @see InputReader
      */
      public static int getAvgSeqLen(){
            int lensum = 0;

            for(int i=1;i<sequenceList.size();i+=2){
                  lensum+=((String)sequenceList.get(i)).length();
            }

            return lensum/(sequenceList.size()/2);
      }

      /**
      * Calculates and print a list of counted support added for all
single motifs in the Pareto.
      * @return list of support values found.
      */
      private static int[] countSupport(){
            //New pareto list based on support
            int[] alt_support = new int[nr_of_seq+1];

            for(int i=0; i<pareto.length;i++){

                  int alt_tot = 0;

                  if(patterns.containsKey(new Integer(i))){

                        //Get all patterns
```

```
                        ArrayList tmpList =
(ArrayList)patterns.get(new Integer(i));

                        //for the bitset-length
                        for(int k=0;k<=nr_of_seq;k++){

                              //support counter
                              int alt_supp = 0;

                              //count total on each position
                              for(int j=0;j<tmpList.size();j++){


      if(((SingleMotif)tmpList.get(j)).getBitSet().get(k)){
                                          alt_supp++;
                                    }

      //System.out.print(((SingleMotif)tmpList.get(j)).getPattern()+
"     ");
                              }
                              alt_tot=alt_tot + alt_supp;
                        }//for
                  }//if
                  alt_support[i] = alt_tot;
            }//for

            //Skriv ut den nye lista
            for(int i=0;i<alt_support.length;i++){

                  if(display.get(PD_COUNTS)){
                        System.out.println("CS["+i+ "]: " +
Integer.toString(alt_support[i]).replace('.',','));
                  }
                  if(print.get(PD_COUNTS)){
                        add_write_output("CS["+i+"]: " +
Integer.toString(alt_support[i]).replace('.',','));
                  }
            }

            return alt_support;
      }

      /**
      * Weights the output from count support to highlight
interesting values.
      * @param sList of weighted support vales.
      */
      private static void weightedSupport(int[] sList){
            double[] wSupport = new double[sList.length];

            for(int i=0;i<pareto.length;i++){
                  wSupport[i]= Math.pow(sList[i],2.0);

                  if(display.get(PD_WEIGHTS)){
                        System.out.println("WS(CS^2)["+i+"]: " +
Double.toString(wSupport[i]).replace('.',','));
                  }
                  if(print.get(PD_WEIGHTS)){
                        add_write_output("WS(CS^2)["+i+"]: " +
Double.toString(wSupport[i]).replace('.',','));
                  }
            }
      }

      /**
      * Binomail significance for the motifs counted support.
      * @param supportList list of support values used in the
calculation.
```

```
	* @return the probability to observe the support used.
	*/
	private static int sign_binomial(int[] supportList){
		int I = nr_of_seq;
		long faci = faculty(I);

		double bestSum = 0;
		int bestIndex = 0;

		for(int x=0;x<=I;x++){
			if(patterns.containsKey(new Integer(x))){
			//System.out.println(x);
			double P = pareto[x];

			double sum=0;

			for(int n=x;n<=I;n++){
				//System.out.println("kjører" +
Math.pow((1-P),(1-x)));
				sum+=faci/(faculty(I-
n)*faculty(n))*Math.pow(P,n)*Math.pow((1-P),(I-n));
			}
			if(sum>=bestSum && sum<1.0){
				bestSum = sum;
				bestIndex = x;
				//System.out.println(sum +" - "
+bestIndex);
			}
			if(display.get(PD_BINOMIAL)){
				System.out.println("SB["+x+"]: " +
Double.toString(sum).replace('.',','));
			}
			if(print.get(PD_BINOMIAL)){
				add_write_output("SB["+x+"]: " +
Double.toString(sum).replace('.',','));
			}
		}
		}

		return bestIndex;
	}

	/**
	* Calculates the facutly of an int
	* @param mul integer to calculate.
	* @return faculty calculation
	*/
	private static long faculty(int mul){
		long answ = 1;
		for(int i=mul;i>1;i--){
			answ*=i;
		}
		return answ;

		/*
		if(mul==0){
			return 1;
		}else{
			return mul * faculty(mul-1);
		}
		*/
	}

	/**
	* Finding likelyhood of observing support over support-total
after many trails
	* @param supportList list of support values.
	*/
```

```
	private static void sign_total(int[] supportList){

		//sum sign over supp in all experiments
		int[] overSupp = new int[nr_of_seq+1];

		for(int p=0;p<pareto.length;p++){
			if(patterns.containsKey(new Integer(p))){

				ArrayList tmpList =
(ArrayList)patterns.get(new Integer(p));

				//number of trials
				for(int i = 1;i<=100000;i++){
					//for each sequence
					int poeng = 0;

					for(int b=0;b<Mofn.MAX_N;b++){

						SingleMotif sm =
(SingleMotif)tmpList.get(b);

						for(int s=0;s<=nr_of_seq;s++){
							//sample support for
given sign (random)

							double tall =
Math.random();

							/*try{

	System.out.println(tall + " vs " + sm.getSign());

	Thread.sleep(500);
							}catch(Exception e){

	System.out.println("Søvn avbrutt");
							}
							*/

							if(tall<sm.getSign()){

	poeng++;//=sm.getSign();
							}
						}

					}//seq
					if(poeng>=supportList[p]){
					//	System.out.println("Wee");
						overSupp[p]++;
					}//if
				}//1000
			}//contains
		}//patterns
		for(int i=0;i<overSupp.length;i++){
			//System.out.println(overSupp[i]);
			if(display.get(PD_EXP)){
				System.out.println("ST["+i+"]: " +
Double.toString((double)overSupp[i]/100000).replace('.',','));
			}
			if(print.get(PD_EXP)){
				add_write_output("ST["+i+"]: " +
Double.toString((double)overSupp[i]/100000).replace('.',','));
			}
		}
	}

	/**
	* Applies distance constraints on patterns found after run.
```

```
	* @param start start position in sequences
	* @param stop end position in sequences
	*/
	private static void applyDistanceConstraints(int start, int
stop){
		//New pareto list based on support

		//for each composite motif found
		for(int i=0;i<pareto.length;i++){

			//if valid, supp >0
			if(patterns.containsKey(new Integer(i))){

				//Get all patterns in CM
				ArrayList tmpList =
(ArrayList)patterns.get(new Integer(i));

				//for each pattern
				for(int j=0;j<tmpList.size();j++){
					SingleMotif testMotif =
((SingleMotif)tmpList.get(j));
					int[] testData =
testMotif.getHitData();
					BitSet testBitSet =
testMotif.getBitSet();
					int removed = 0;

					//for each hit/pos data
					for(int k=1;k<testData.length;k+=2){

						if(testData[k]<start ||
testData[k]>stop){
							testBitSet.clear(k-1);
							testData[k-1]=-1;
							testData[k]=-1;
							removed++;
						}

	//System.out.print(((SingleMotif)tmpList.get(j)).getPattern()+
"	");
					}

					int[] newData = new
int[testData.length-removed*2];
					int newIndex = 0;
					for(int k=1;k<testData.length;k++){
						if(testData[k]!=-1){
							newData[newIndex] =
testData[k];
							newIndex++;
						}
					}


	testMotif.setHitInfo(testBitSet,newData,newData.length/2,testBi
tSet.cardinality());
				}//for
			}//if
		}//for

	}

	/**
	* Calulates significance based up on normal distribution.
	* Finds mean, variance and probability. ONLY WORKS ON SMALL
DATASETS.
	* @param meas list of support to check the probability for.
	* @return a list of probabilities found.
```

```
	*/
	private static double[] normalDistribution(int[] meas){
		int listSize = 0;

		for(int i=0;i<MAX_N;i++){

	listSize+=Math.floor(faculty(motifList.size())/(faculty(motifLi
st.size()-i)*faculty(i)));
		}
		listSize+=MAX_N;

		//finding normal distribution for the motifs
		int[] motsupport = new int[listSize];
		int motindex = 0;
		for(int i=0;i<motifList.size();i++){
			exploreSupport(motsupport,motindex,i,0);
		}

		Arrays.sort(motsupport);


		//x p distribution
		int countmeas[]= new int[meas.length];
		for(int a=0;a<meas.length;a++){
			for(int i=0;i<motsupport.length;i++){
				if(motsupport[i]==meas[a]){
					countmeas[a]++;
				}
			}
			countmeas[a]=countmeas[a]/motsupport.length;
		}

		//Mean
		double mean = 0.0;
		for(int a=0;a<meas.length;a++){
			mean+= meas[a]*countmeas[a];
		}

		//Variance
		double variance = 0.0;
		for(int a=0;a<meas.length;a++){
			variance+=Math.pow(meas[a]-mean,2);
		}

		//Probability
		double[] calc_ans = new double[meas.length];
		for(int a=0;a<meas.length;a++){
			for(int b=meas[a];b<nr_of_seq;b++){
				calc_ans[a] +=
1/(Math.sqrt(2*Math.PI)*variance)*Math.pow(Math.E,-
(1/2)*Math.pow(((meas[a]-mean)/variance),2));
			}
		}

		//System.out.println();
		//add_write_output();
		return calc_ans;
	}

	/**
	* Used to find the normal distribution of possible support
values.
	* @param motsupport empty list, placeholder for values
calculated.
	* @param motindex incrementing counter.
	* @param p motif index to explore.
	* @param t number of motifs calculated t<=MAX_M.
	*/
```

```
	private static void exploreSupport(int[] motsupport,int
motindex, int p, int t){
		p++;
		int partsupp = 0;

		while(p<motifList.size() && t<MAX_N){
			SingleMotif sm = (SingleMotif)motifList.get(p);
			partsupp += sm.getSupp();
			t++;
			exploreSupport(motsupport, motindex, p,t);
		}

		if(p==MAX_N){
			motsupport[motindex] = partsupp;
			motindex++;
		}
	}

	/**
	* This method is used to print or display the list of motifs
read in before starting a run.
	* @see InputReader
	*/
	private static void display_motifs(){
		//double sign = 0.0;
		//int counter = 0;

		for(int c=0;c<motifList.size();c++){
			SingleMotif mym = (SingleMotif)motifList.get(c);

			if(print.get(PD_MOTIFS)){
				add_write_output(c + "   -  Supp: " +
mym.getSupp() + "   -  Sign: " + mym.getSign() + "   -  Pattern: "+
					mym.getPattern());

	add_write_output(mym.getBitSet().toString());
			}
			if(display.get(PD_MOTIFS)){
				System.out.println(c + "   -  Supp: " +
mym.getSupp() + "   -  Sign: " + mym.getSign() + "   -  Pattern: "+
					mym.getPattern());

	System.out.println(mym.getBitSet().toString());
			}
			//int[] moreData = mym.getHitData();
			//for(int i=0;i<moreData.length;i++){
				//System.out.println(moreData[i]);
			//}

		}

	}

	/**
	* Creates a subset of the motiflist with size provided.
	* @param size Number of elements in the random subset
	*/
	private static void motif_subset(int size){

		while(motifList.size()> size){
			int remove = (int)Math.random()*motifList.size();

			Object garbage = motifList.remove(remove);
		}
	}

	/**
	* Print or displays settings used during a run
```

```
	* @param args Arguments used to run program.
	*/
	private static void display_settings(String[] args){
		String param = "";
		for(int i=0;i<args.length;i++){
			param = param + " " + args[i];
		}

		System.out.println("Version: " + Mofn.VERSION);
		System.out.println("Parameters: " + param);
		//System.out.println("Motifs: " + motifFile + "  Seqs: "
+ sequenceFile);
		System.out.println("Start: " + startPos + " - Stop: " +
stopPos + " PL: " + pareto.length + " - N: " + (Mofn.MAX_M) + "/" +
Mofn.MAX_N);
		System.out.println("Motif count: " + motifList.size());
		if(mode[INPUT]==PVM){
			System.out.println("Threshold: " + threshold + "
Percent: " + percent);
		}

		add_write_output("Version: " + Mofn.VERSION);
		//add_write_output("Motifs: " + motifFile + "  Seqs: " +
sequenceFile);
		add_write_output("Parameters: " + param);
		add_write_output("Start: " + startPos + " - Stop: " +
stopPos + " PL: " + pareto.length + " - N: " + (Mofn.MAX_M) + "/" +
Mofn.MAX_N);
		add_write_output("Motif count: " + motifList.size());
		if(mode[INPUT]==PVM){
			add_write_output("Threshold: " + threshold + "
Percent: " + percent);
		}
	}

	/**
	* This method is used to print or display the Pareto-front list
	*/
	private static void display_pareto(){
		for(int x=0;x<pareto.length;x++){
			double inv_pareto = 1-pareto[x];
			if(display.get(PD_PARETO)){
				System.out.println("Pareto["+x+"]: " +
(Double.toString(inv_pareto)).replace('.',','));
			}
			if(print.get(PD_PARETO)){
				add_write_output("Pareto["+x+"]: " +
(Double.toString(inv_pareto)).replace('.',','));
			}
			//System.out.print(x + "  " + pareto[x]);
		}
	}


	/**
	* This method is used to print or display the patterns found
	*/
	private static void display_patterns(){
		for(int i=0; i<pareto.length;i++){
			if(patterns.containsKey(new Integer(i))){
				ArrayList tmpList =
(ArrayList)patterns.get(new Integer(i));
				String outStr = "Pattern[" + i+ "]: ";

				for(int j=0;j<tmpList.size();j++){
					outStr = outStr +
((SingleMotif)tmpList.get(j)).getPattern()+ " ";
				}
```

```
                        if(display.get(PD_PATTERNS)){
                              System.out.println(outStr);
                        }
                        if(print.get(PD_PATTERNS)){
                              add_write_output(outStr);
                        }
                  }
            }
      }

      /**
      * Prints or displays the most common single motif from the
composite elements.
      */
      private static void display_mostCommonMotif(){
            TreeMap mcm = new TreeMap();

            for(int i=0; i<pareto.length;i++){
                  if(patterns.containsKey(new Integer(i))){
                        ArrayList tmpList =
(ArrayList)patterns.get(new Integer(i));
                        //String outStr = Pattern[" + i+ "]: ";

                        for(int j=0;j<tmpList.size();j++){

      if(mcm.containsKey(((SingleMotif)tmpList.get(j)).getName())){
                                    int val =
((Integer)mcm.get(((SingleMotif)tmpList.get(j)).getName())).intValue(
);

      mcm.put(((SingleMotif)tmpList.get(j)).getName(),new
Integer(val+1));
                                    }else{

      mcm.put(((SingleMotif)tmpList.get(j)).getName(),new
Integer(1));
                                    }
                        }//for
                  }//if
            }//for

            String[] mcm_str = new String[mcm.size()];
            int count = 0;

            for(int i=0;i<motifList.size();i++){
                  SingleMotif sm = (SingleMotif)motifList.get(i);
                  if(mcm.containsKey(sm.getName())){

                        mcm_str[count] =
((Integer)mcm.get(sm.getName())).toString() + "_" +
sm.getName();//System.out.println(sm.getName() + " - " +
((Integer)mcm.get(sm.getName())).intValue());
                        count++;

                        //System.out.println(sm.getName() + " - " +
((Integer)mcm.get(sm.getName())).intValue());
                  }
            }

            Arrays.sort(mcm_str);
            for(int i=mcm_str.length-1;i>=0;i--){
                  StringTokenizer strt = new
StringTokenizer(mcm_str[i],"_");
                  String cc = strt.nextToken();
                  String mm = strt.nextToken();

                  if(display.get(PD_SINGLEMOTIFS)){
                        System.out.println(mm + ": " + cc);
```

```
                  }
                  if(print.get(PD_SINGLEMOTIFS)){
                        add_write_output(mm + ": " + cc);
                  }

            }

      }

      /**
      * Prints or displays the list of motif hits combined with
sequences
      * @param sequences list of sequences with alternating (name,
pattern)
      * @param bestIndex best target pattern found.
      */
      private static void display_results(Vector sequence, int
bestIndex){

            //System.out.println("PS: " + patterns.size());
            //System.out.println("BI: "+ bestIndex);
            //System.out.println("SS: " + sequence.size());
            if(display.get(PD_FINAL)){
                  System.out.println("Module 1:");// + bestIndex +
":");
            }
            if(print.get(PD_FINAL)){
                  add_write_output("Module 1:");//"+bestIndex +
":");
            }
            if(patterns.size()!=0){
                  ArrayList bestMotifs = (ArrayList)patterns.get(new
Integer(bestIndex));
                  /*
                  for(int p=0;p<patterns.size();p++){
                        if(patterns.containsKey(new Integer(p))){
                              System.out.println(p);
                        }
                  }
                  System.out.println("BI:" + bestIndex);
                  System.out.println(bestMotifs.size());
                  */

                  //if(bestMotifs!=null){
                  int seqCounter=0;
                  for(int s=0;s<sequence.size();s+=2){
                        String seqName =
(String)sequence.elementAt(s);
                        String seqStr =
(String)sequence.elementAt(s+1);
                        String motNameStr = "";

                        int firstHit = seqStr.length()+1;
                        int lastHit = -1;

                        for(int m=0;m<bestMotifs.size();m++){
                              SingleMotif sm =
(SingleMotif)bestMotifs.get(m);
                              motNameStr=motNameStr + ", " +
sm.getName();

                              int[] hitData = sm.getHitData();

                              for(int h=0;h<hitData.length;h+=2){

      //System.out.println(hitData[h]);
                                    if(hitData[h]==seqCounter){
```

```
	if(hitData[h+1]<firstHit){
                                                        firstHit =
hitData[h+1];


	if(firstHit+sm.getRows()>lastHit){

	lastHit=firstHit+sm.getRows()-1;

	//System.out.println("LH: " + lastHit);

	//System.out.println("SL: " + seqStr.length());
                                                        }
                                                  }

	if(hitData[h+1]>lastHit){
                                                        lastHit =
hitData[h+1];
                                                  }
                                            }//if
                                      }//for listdata

                               }//for mot
                        //     outputstr= outputstr +
((String)sequence.elementAt(s+1)).substring(0,1);
                        //     outputstr= outputstr + sm.getName();
                               if(firstHit!=seqStr.length()+1 &&
lastHit!=-1){
                                      if(display.get(PD_FINAL)){
                                            System.out.println(seqName +
", " + firstHit +", " + lastHit +", "+
seqStr.substring(firstHit,lastHit)+motNameStr);
                                      }
                                      if(print.get(PD_FINAL)){
                                            add_write_output(seqName + ",
" + firstHit +", " + lastHit +", "+
seqStr.substring(firstHit,lastHit)+motNameStr);
                                      }
                               }else{
                                      //System.out.println(seqName + "," +
firstHit +"," + lastHit +","+
seqStr.substring(firstHit,lastHit)+","+motNameStr);
                               }
                               seqCounter++;
                        }//for seq
                }else{
                        if(display.get(PD_FINAL)){
                               System.out.println("Empty...");
                        }
                        if(print.get(PD_FINAL)){
                               add_write_output("Empty...");
                        }
                }//if null
        }


        /**
        * Find last non-zero element in a list, can be used to cut of a
long empty list.
        * @Param pList list to find the last element in.
        * @returns Last elements index.
        */
        public static int findLastElement(double[] pList){
                int i;
                for(i=pList.length-1;i>=0;i--){
                        if(pList[i]==0){
                        }
```

```
		}
		return i;
	}

	/**
	* Find last non-zero element in a list, can be used to cut of a
long empty list.
	* @Param pList list to find the last element in
	* @returns Last elements index
	*/
	public static int findLastElement(int[] pList){
		int i;
		for(i=pList.length-1;i>=0;i--){
			if(pList[i]==0){
			}
		}
		return i;
	}


	/**
	* Finds scores for PWMs for every position in every sequence.
	* @param sequences a list of alternating sequence name and
pattern.
	* @return A double[] with scores found for each sequence
	*/
	private static ArrayList findMatrixScores(Vector sequences){
		ArrayList allSums = new ArrayList();

		for(int m=0;m<motifList.size();m++){
			SingleMotif sm = (SingleMotif)motifList.get(m);
			double[][] mat = sm.getMatrix();
			double[] rowsum = sm.getRowSum();

			for(int i=1;i<sequences.size();i+=2){
				//System.out.println(i);

				String seqstr =
(String)sequences.elementAt(i);

				double partSums[] = new
double[seqstr.length()];

				for(int p=0;p<seqstr.length()-
sm.getRows();p++){
					double partSum = 1.0;

					for(int c=0;c<sm.getRows();c++){

	partSum*=mat[c][getIndex(seqstr.charAt(p+c))]/rowsum[c];

					}
					partSums[p] = partSum;

				}//single seq

				allSums.add(partSums);
			}//tot seqs
		}//motif
		return allSums;
	}//function

	/**
	* Translate ACGT letters to indexs 0,1,2,3
	* @return int index
	*/
	private static int getIndex(char c){
		int index = 0;
```

```
		c = Character.toUpperCase(c);
		Character ch = new Character(c);

		if(ch.equals(new Character('A'))){
			index= 0;
		}else if(ch.equals(new Character('C'))){
			index= 1;
		}else if(ch.equals(new Character('G'))){
			index= 2;
		}else if(ch.equals(new Character('T'))){
			index= 3;
		}

		return index;
	}

	/**
	* Finds a threshold value for PWMs based on percent included
	* @param allsums Contain scores as double[] for every position
in every sequence
	* @return Threshold to include the amount decided on as cutoff-
value.
	*/
	private static double findMatrixThreshold(ArrayList allsums){
			int extra = 10;

			double topScores[] = new
double[allsums.size()*extra];

			for(int s=0;s<allsums.size();s++){
				double tmpScore[] =
(double[])allsums.get(s);
				double score[] = new
double[tmpScore.length];

				for(int t=0;t<tmpScore.length;t++){
					score[t]=tmpScore[t];
				}

				Arrays.sort(score);
				int bound = extra;
				if(extra>score.length){
					bound=score.length;
				}
				for(int t=0;t<bound;t++){

	topScores[s*extra+t]=score[score.length-1];
				}
				/*for(int i=0;i<score.length;i++){
					if(score[i]!=0.0){
						System.out.println(score[i]);
					}
				}*/

			}



			Arrays.sort(topScores);

			/*for(int i=topScores.length-1;i>=0;i--){
				//if(topScores[i]!=0.0){
				System.out.println(topScores[i]);
				//}
			}*/

			int thresholdIndex =
(int)topScores.length*percent/100;
```

```
                    //System.out.println("Thre: " +
topScores[topScores.length-thresholdIndex]);
                    return topScores[topScores.length-thresholdIndex];
                    //System.out
      }

      /**
      * Finds PWM support for all the sequences based on threshold
      * @param allScores A list for every hit in every sequence
      * @param thres used during cut-off.
      */
      private static void findMatrixSupport(ArrayList allScores,
double thres){

                //System.out.println("nrofseq:" + nr_of_seq);
                //System.out.println("seq: " + allScores.size() + "
mot: " + motifList.size());
                    //double seqScores[] = new
double[allScores.size()];
                    for(int m=0;m<motifList.size();m++){
                          SingleMotif sm =
(SingleMotif)motifList.get(m);

                        Vector hitInfo = new Vector(); //pos/seq

                          for(int
s=m*(nr_of_seq);s<(m*(nr_of_seq)+(nr_of_seq));s++){
                                double seqScore[] =
(double[])allScores.get(s);

                                for(int i=0;i<seqScore.length;i++){
                                      if(seqScore[i]>=thres){
                                            //System.out.println(s-
(m*(nr_of_seq)));
                                            hitInfo.add(new
Integer(s-(m*(nr_of_seq))));
                                            hitInfo.add(new
Integer(i));
                                      }
                                }//Seq
                          }//allseq

                          BitSet bs = new BitSet(nr_of_seq+1);
                          int hitData[] = new int[hitInfo.size()];
                        int htot = hitInfo.size()/2;
                        int hseq = 0;

                        int lastSeqNr = -1;

                          for(int j=0;j<hitInfo.size();j+=2){


      bs.set(((Integer)hitInfo.elementAt(j)).intValue());

      hitData[j]=((Integer)hitInfo.elementAt(j)).intValue();

      hitData[j+1]=((Integer)hitInfo.elementAt(j+1)).intValue();

      if(((Integer)hitInfo.elementAt(j)).intValue()!=lastSeqNr){
                                      hseq++;
                                      lastSeqNr =
((Integer)hitInfo.elementAt(j)).intValue();
                                }

                          }

                          sm.setHitInfo(bs,hitData,hseq,htot);
                    }//mot
```

```
	}

	/**
	* Finds PWM significance based on random sequences or real
background
	* @param thres Value useded to cut-off matches to find
significance
	* @param backseqs List of background sequences (optional)
null/!null
	*/
	private static void findMatrixSign(double thres, Vector
backseqs){
		Vector seqs;

		if(backseqs==null){
			InputReader ir = new InputReader();
			seqs = ir.createRandomSequences(1000,1000);
		}else{
			seqs = backseqs;
		}

		for(int m=0;m<motifList.size();m++){
			SingleMotif sm = (SingleMotif)motifList.get(m);
			double[][] mat = sm.getMatrix();
			double[] rowsum = sm.getRowSum();
			int countHits = 0;
			double totSum = 0.0;

			for(int i=1;i<seqs.size();i+=2){
				//System.out.println(i);

				String seqstr = (String)seqs.elementAt(i);

				for(int p=0;p<seqstr.length()-
sm.getRows();p++){
					double partSum = 1.0;

					for(int c=0;c<sm.getRows();c++){

	partSum*=mat[c][getIndex(seqstr.charAt(p+c))]/rowsum[c];

					}

					if(partSum>=thres){
						countHits++;
						break;
					}

				}//single seq

				//totSum +=
(double)countHits/((double)seqstr.length()-(double)sm.getRows());
				//countHits = 0;
			}//tot seqs


			//double signFound =
(double)countHits/((double)(seqs.size()/2*1000));
			double signFound =
(double)countHits/((double)(seqs.size()/2));
			sm.setSign(signFound);

		}//motif

		sortMotifList();
		//motifList = sortArrayList();
	}
```

```
	/**
	* Sorts the arraylist of motifs after new significance values
are applied.
	*/
	private static void sortMotifList(){
		//SORT LIST AFTER DESCENDING SIGNIFICANCE FOUND

		ArrayList newMotifList = new ArrayList();

		for(int i=0;i<motifList.size();i++){
			SingleMotif sortMotif =
(SingleMotif)motifList.remove(0);

			if(newMotifList.size()==0){
				//System.out.println(mSign);
				newMotifList.add(sortMotif);
			}else{
				//ACENDING
				for(int s=0;s<newMotifList.size();s++){
					SingleMotif inListMotif =
(SingleMotif)newMotifList.get(s);


	if(sortMotif.getSign()<=inListMotif.getSign()){
							newMotifList.add(s,sortMotif);
							sortMotif=null;
							break;
					}
				}
				if(sortMotif!=null){
					newMotifList.add(sortMotif);
					sortMotif=null;
				}

				/*
				//DESCENDING
				int ms = 0;

				do{
					SingleMotif inListMotif =
(SingleMotif)newMotifList.get(ms);

					if (ms>=newMotifList.size()-1){
						newMotifList.add(sortMotif);
						sortMotif=null;
					}else
if(sortMotif.getSign()>=inListMotif.getSign()){

	newMotifList.add(ms,sortMotif);
							sortMotif=null;
					}
					ms++;
				}while (sortMotif!=null);//(ms<=
listOfMotifs.size() &&
				*/

			}//if
		}
		motifList = newMotifList;
	}

	/**
	* Finds motif significance (ACGT/IUPAC) based on background
	* @param backseq List of sequences with alternating (name,
pattern)
	*/
```

```
	private static void findMotifSign(Vector backseq){

		for(int t=0;t<motifList.size();t++){
			SingleMotif sm = (SingleMotif)motifList.get(t);
			int hits = 0;

			//PATTERN TO CHECK
			//String motstr = mot.getPattern();
			//TRANSLATE IUPAC INTO ACID...
			String motstr = sm.getPattern();
			motstr = motstr.toUpperCase();
			motstr = translateIUPAC(motstr);


			//FIND ACTUAL PATTERN LENGTH
			int motlen = 0;
			boolean between = false;
			int betweenloop = 0;

			for(int l=0;l<motstr.length();l++){
				if(between){

	if(motstr.substring(l,l+1).equals("]")){
						if(betweenloop==1){
							between = false;
						}
						betweenloop--;
					}else
if(motstr.substring(l,l+1).equals("[")){
						betweenloop++;
					}else{
					}

				}else{

	if(motstr.substring(l,l+1).equals("[")){
						if(betweenloop==0){
							motlen++;
							between=true;
						}
						betweenloop++;
					}else{
						motlen++;
					}
				}
			}

			//System.out.println("Motlen: " + motlen);

			//FOR EACH BACKGROUND SEQ
			for(int i=1;i<backseq.size();i+=2){
				String seqstr = (String)backseq.get(i);
				seqstr = seqstr.toUpperCase();

				for(int p=0;p<seqstr.length()-motlen;p++){
					//System.out.println("P: " + p);

					boolean mismatch = false;
					boolean inside = false;
					int insideloop = 0;

					boolean foundletter = false;
					int rp = 0;

					//while(m<motlen && !mismatch){
					for(int m=0;m<motstr.length();m++){

						if(inside){
```

```
	if(motstr.substring(m,m+1).equals("[")){
                                                        insideloop++;

                                                }else
if(motstr.substring(m,m+1).equals("]")){


	if(insideloop==1){

	inside=false;

	if(!foundletter){

	mismatch=true;

	break;
                                                                }


	insideloop--;
                                                                rp++;
                                                        }
                                                }else
if(motstr.substring(m,m+1).equals(seqstr.substring(p+rp,p+rp+1))){
                                                        foundletter=true;
                                                }
                                        }else
if(motstr.substring(m,m+1).equals("[")){
                                                inside=true;
                                                foundletter = false;
                                                insideloop++;
                                        }else
if(motstr.substring(m,m+1).equals(seqstr.substring(p+rp,p+rp+1))){
                                                rp++;
                                        }else{
                                                rp++;
                                                mismatch = true;
                                                break;
                                        }

                                }//for mot
                                if(!mismatch){
                                        hits++;
                                        break;
                                }
                        }//seqpos
                }//for seg
                sm.setSign((double)hits/(double)backseq.size()/2);
        }//for mot

        sortMotifList();
    }//function

    /**
    * Translate IUPAC characters into patterns with []
    * @param transstr Pattern to translate.
    * @return The translated pattern.
    */
    private static String translateIUPAC(String transstr){
        transstr.toUpperCase();
        transstr = transstr.replaceAll("R","[AG]");
        transstr = transstr.replaceAll("Y","[CTU]");
        transstr = transstr.replaceAll("K","[GTU]");
        transstr = transstr.replaceAll("M","[AC]");

        transstr = transstr.replaceAll("S","[GC]");
        transstr = transstr.replaceAll("W","[ATU]");
        transstr = transstr.replaceAll("B","[GTCU]");
```

```
		transstr = transstr.replaceAll("D","[GATU]");
		transstr = transstr.replaceAll("H","[ACTU]");
		transstr = transstr.replaceAll("V","[GCA]");
		transstr = transstr.replaceAll("N","[ACGTU]");
		transstr = transstr.replaceAll("T","[TU]");

		return transstr;
	}


	/**
	* Starts the server mode for a given number of clients
	* @param nrClients The number of Clients that will connect to
the server.
	* @throws Exception If IO error occurs.
	*/
	private static void startSerializedServer(int nrClients) throws
Exception{
		//while running
		ArrayList sockets = new ArrayList();

		//CALCULATE SUBWORK
		int subWork =
(int)Math.ceil(motifList.size()/nrClients);
		System.out.println("Ready for " + (MAX_M) + " of "+
MAX_N + " search...");
		System.out.println("Waiting for clients...");

		for(int c=0;c<nrClients;c++){
			Client myClient;
			if(c<(nrClients-1)){
				myClient = new
Client(portNr,c,c*subWork,c*subWork-1+subWork);
			}else{
				myClient = new
Client(portNr,c,c*subWork,motifList.size());
			}

	System.out.println(((InetAddress)myClient.getIP()).getHostAddre
ss());
			System.out.println("Can handle client: " + c + "
Port: " + (portNr+c));

			sockets.add(myClient);
		}

		int runningClients = nrClients;

		while(runningClients>0){
			for(int c=0;c<nrClients;c++){
				Client client = (Client)sockets.get(c);

				while(client.isRunning()){
					try{
						ObjectInputStream ois =
client.getOIS();
						ObjectOutputStream oos =
client.getOOS();

						String command =
(String)ois.readObject();

						if(command.equals("update")){
							for(int
p=0;p<pareto.length;p++){

	pareto[p]=((Double)ois.readObject()).doubleValue();
						}
```

```
                                                pareto =
updatePareto(pareto);
       oos.writeObject("update");
                                                for(int
p=0;p<pareto.length;p++){
       oos.writeObject(new Double(pareto[p]));
                                                }
                                                break;
                                        }else
if(command.equals("finish")){
       System.out.println(command);
                                                double[] clientPareto =
new double[pareto.length];
                                                for(int
p=0;p<pareto.length;p++){
       clientPareto[p]=((Double)ois.readObject()).doubleValue();
                                                }
                                                clientPareto =
updatePareto(clientPareto);
                                                ArrayList smList = new
ArrayList();
                                                int nrpatterns =
((Integer)ois.readObject()).intValue();
                                                int p=0;
                                                while(p<nrpatterns){
                                                        int
psupp=p;
                                                        try{
                                                               int
nrsm = ((Integer)ois.readObject()).intValue();
       psupp = ((Integer)ois.readObject()).intValue();
                                                               int
h=0;
       while(h<nrsm){
       try{
              double hsign= ((Double)ois.readObject()).doubleValue();
              int hsupp = ((Integer)ois.readObject()).intValue();
              String hpattern= (String)ois.readObject();
              int hsize =((Integer)ois.readObject()).intValue();
              int[] hlist = new int[hsize];

              int u=0;
              while(u<hsize){
                     try{
```

```
                        hlist[u]=
((Integer)ois.readObject()).intValue();

                        u++;

                  }catch(Exception e){

                  }

            }

            BitSet hbs = new BitSet();

            for(int d=0;d<hlist.length;d+=2){

                  hbs.set(hlist[d]);

            }

            SingleMotif addsm = new
SingleMotif(0,hsupp,hsign,hpattern,hlist,hbs);

            if(addsm.getName().equals("MotifX")){

                  addsm.setName(hpattern);

            }

            smList.add(addsm);



            h++;

      }catch(Exception e){

            e.printStackTrace();

            break;

      }
                                                                  }


      if(clientPareto[p]>=pareto[p]){

      patterns.put(new Integer(psupp),smList);

      }else{

      smList.clear();
                                                                  }
                                                                  p++;

      }catch(Exception e){

      e.printStackTrace();

      break;
                                                        }
                                                  //}
                                          }
                                          //ArrayList patterns =
(ArrayList)ois.readObject();
                                          client.stopClient();
                                          runningClients--;
                                          break;
```

```
                                        }else
if(command.equals("range")){

      System.out.println("range");
                                              oos.writeObject(new
Integer(client.start));
                                              oos.writeObject(new
Integer(client.stop));
                                        }else
if(command.equals("mofn")){

      System.out.println("range");
                                              oos.writeObject(new
Integer(MAX_M));
                                              oos.writeObject(new
Integer(MAX_N));
                                        }else{
                                              break;
                                        }
                                  }catch(Exception e){
                                        e.printStackTrace();
                                  }
                            }//while client
                      }//for clients
                }//while running


                //print results
          }

          /**
          * Method used by the server when pareto-lists are recieved from
clients
          * @param clientPareto Pareto received from clients
          * @return Updated Pareto list
          */
          private static double[] updatePareto(double[] clientPareto){

                for(int i=clientPareto.length-1;i>=0;i--){
                      if(clientPareto[i]>pareto[i]){
                            for(int u=i;u>=0;u--){
                                  pareto[u]=clientPareto[i];
                            }
                      }
                }

                return pareto;
          }

          /**
          * Starts the client mode and connects to the server provided
          * @param serverIP IP-adress <xxx.xxx.xxx.xxx>
          */
          private static void startSerializedClient(String serverIP)
throws Exception{
                //STARTUP

          System.out.println(InetAddress.getByName(serverIP).getHostAddre
ss());
                System.out.println(portNr);

                sock = new
Socket(InetAddress.getByName(serverIP),portNr);
                //sock = new Socket("VELTEV",portNr);

                oos = new ObjectOutputStream(sock.getOutputStream());
                ois = new  ObjectInputStream(sock.getInputStream());
```

```
		stopPos=0;

		try{
			while(stopPos==0){
				oos.writeObject("range");
				startPos =
((Integer)ois.readObject()).intValue();
				stopPos =
((Integer)ois.readObject()).intValue();
			}
		}catch(Exception e){
			e.printStackTrace();
		}

		String command = "";
		boolean running = true;
		MAX_N=0;
		try{
			while(MAX_N==0){
				oos.writeObject("mofn");
				MAX_M =
((Integer)ois.readObject()).intValue();
				MAX_N =
((Integer)ois.readObject()).intValue();
			}

		}catch(Exception e){
			e.printStackTrace();
		}

		//RUN()

		run();

		//FINISH
		//command="update";


		command="finish";
		oos.writeObject(command);

		for(int p=0;p<pareto.length;p++){
			oos.writeObject(new Double(pareto[p]));
		}

		//display_pareto();
		//System.out.println(patterns.size());

		oos.writeObject(new Integer(patterns.size()));

		for(int p=0;p<pareto.length;p++){
			if(patterns.containsKey(new Integer(p))){

				ArrayList smarr =
(ArrayList)patterns.get(new Integer(p));
				//System.out.println(smarr.size());

				oos.writeObject(new Integer(smarr.size()));
				oos.writeObject(new Integer(p));

				for(int h=0;h<smarr.size();h++){
					SingleMotif sm =
(SingleMotif)smarr.get(h);

					oos.writeObject(new
Double(sm.getSign()));
```

```
                                    oos.writeObject(new
Integer(sm.getSupp()));
                                    oos.writeObject(sm.getPattern());

                                    int[] hdl = sm.getHitData();
                                    oos.writeObject(new
Integer(hdl.length));

                                    for(int u=0;u<hdl.length;u++){
                                          oos.writeObject(new
Integer(hdl[u]));
                                    }
                                    //oos.writeObject(sm.getBitSet());

                              }

                        }
                }
                sock.close();
                System.out.println("Client done...");
        }

}
```

CompositeMotif.java

```
import java.util.ArrayList;
import java.util.BitSet;
import java.util.HashMap;

 /**
 * This class holds the list of Single Motifs, all motif variants
 * @author Vetle Valebjørg
 * @version 1.0
 * @see SingleMotif, RealCompositeMotif, FictiveCompositeMotif,
MotifVariant
 */
public class CompositeMotif implements Cloneable{

       protected HashMap motVarHolder;
       protected ArrayList singleMotifHolder;

       private String cName = "";

       /**
       * Default constructor
       */
       public CompositeMotif(){
             motVarHolder = new HashMap();
             singleMotifHolder = new ArrayList();
       }

       /**
       * Constructor used during cloning
       * @param cm Original composite motif to copy.
       */
       public CompositeMotif(CompositeMotif cm){
             motVarHolder = (HashMap)cm.motVarHolder.clone();
             singleMotifHolder =
(ArrayList)cm.singleMotifHolder.clone();
       }

       /**
       * Sets a new name
       * @param str The new name.
       */
       public void setName(String str){
             cName = str;
       }
```

```
	/**
	* Returns the Composite Motifs name
	* @return Name
	*/
	public String getName(){
		return cName;
	}

	/**
	* Makes keys used to identify motif variants Nr-Nr
	* @param mt Part of value to make a key for the motif variant m
of n
	* @param nt Part of value to make a key for the motif variant m
of n
	*/
	protected String makeKey(int mt, int nt){
		return Integer.toString(mt) +"-"+ Integer.toString(nt);
	}

	/**
	* Handles calculations of motif variants
	* @param nt Current n calculated
	* @param sm Single motif added during expansion
	* @param expType Type of expansion (expand/prune) - used for
debug
	* @return Higest m-value calculated (used by pruning).
	*/
	protected int calc(int nt, SingleMotif sm, String expType){

		int lowl = nt-Mofn.MAX_M;
		if(lowl <0){
			lowl=0;
		}

		int highl = Mofn.MAX_M;
		if(highl>nt){
			highl=nt;
		}

		//Iterating thorugh each column for current N
		int t=0;
		for(t=lowl;t<=highl;t++){
			//Mofn.add_output("Action: " + expType);

			//Creating MV for gray area above N/N combinations
			if(t==nt && nt!=0){
				MotifVariant outOfBoundsMV = new
MotifVariant();
				outOfBoundsMV.makeDummy();

	motVarHolder.put(makeKey(nt+1,nt),outOfBoundsMV);

//				Mofn.add_output("Expand: id:" +
outOfBoundsMV.getId() + ": " + (nt + 1) + " - " + nt);
				//Mofn.add_output("Sign: " +
outOfBoundsMV.getSign());
				//Mofn.add_output("Supp: " +
outOfBoundsMV.getBitSet().toString());
			}

			if(t==0 && nt>0){
				//STARTUP
				MotifVariant firstMV = new MotifVariant();
				firstMV.makeFirst();
				motVarHolder.put(makeKey(t,nt),firstMV);
```

```
//                          Mofn.add_output("Expand: id: " +
firstMV.getId() + ": " + t + " - " + nt);
                            //Mofn.add_output("Sign: " +
firstMV.getSign());
                            //Mofn.add_output("Supp: " +
firstMV.getBitSet().toString());

                    }else{
                            //EXPAND
                            MotifVariant newMV = new MotifVariant();

                            MotifVariant prevMV
=(MotifVariant)motVarHolder.get(makeKey(t-1,nt-1));
                            MotifVariant dummyMV
=(MotifVariant)motVarHolder.get(makeKey(t,nt-1));

//                          Mofn.add_output("Expand: id: " +
newMV.getId() +":"+prevMV.getId()+":"+dummyMV.getId() + ": " + t + "
- " + nt);

                            newMV = newMV.calc_mn(prevMV,dummyMV,sm);

                            //System.out.println("RES: " +
newMV.getBitSet().toString());
                            motVarHolder.put(makeKey(t,nt),newMV);

                    }//else
             }//for
             return t-1;
      }//calc


      /**
      * This method returns the list of motifs held in object
      * @return List of motifs participating in the module.
      */
      public ArrayList getSingelMotifs(){
            return singleMotifHolder;
      }

      /**
      * Method added to override object clone
      * @return A copy of the module.
      * @see Object
      */
      public Object clone(){
            try {
                  return super.clone();
      }catch (CloneNotSupportedException e) { // Dire trouble!!!
            throw new InternalError("But we are Cloneable!!!");
      }
      }

}
```

RealCompositeMotif.java

```
import java.util.ArrayList;
import java.util.BitSet;

/**
 * This class represents real composite motifs during a run
 * @author Vetle Valebjørg
 * @version 1.0
 * @see CompositeMotif, FictiveCompositeMotif, MotifVariant,
SingleMotif
 */
public class RealCompositeMotif extends CompositeMotif implements
Cloneable{


	/**
	* Default constructor used by subclass Motifs
	* @see Motif
	*/
	public RealCompositeMotif(){
		super();

		MotifVariant firstMV = new MotifVariant();
		firstMV.makeFirst();
		motVarHolder.put(makeKey(0,0),firstMV);

		MotifVariant dummyMV = new MotifVariant();
		dummyMV.makeDummy();
		motVarHolder.put(makeKey(1,0),dummyMV);
	}

	/**
	* Constructor used during cloning of composite elements
	*/
	public RealCompositeMotif(CompositeMotif cm){
		super(cm);
	}


	/**
	* Method used to explore the DFS from Mofn
	* @param n Current position in motif list as int.
	* @see Mofn
	*/
	public void expand(int n){

		SingleMotif latestMotif =
(SingleMotif)Mofn.motifList.get(n);
		singleMotifHolder.add(latestMotif);

		int nt = singleMotifHolder.size();
		int t = calc(nt,latestMotif,"expanding");

		if(Mofn.PRUNE){
			while(n<Mofn.motifList.size()-1 && nt<Mofn.MAX_N){

				n++;
				//PRUNE, CHECK TO SEE IF SEARCH CAN BE
STOPPED
				FictiveCompositeMotif fcm = new
FictiveCompositeMotif((CompositeMotif)this.clone());
				fcm.setName("prune");
				MotifVariant chkMV =
(MotifVariant)motVarHolder.get(makeKey(t,nt));


	if(fcm.prune(n,latestMotif.getSign(),Mofn.MAX_N-
nt)<Mofn.pareto[chkMV.getSupport()]){
```

```
                                RealCompositeMotif rcm = new
RealCompositeMotif((CompositeMotif)this.clone());
                                rcm.setName("expand");
                                rcm.expand(n);
                        }else{
                                //System.out.println("pruned");
                                continue;

                        }
                }
        }else{
                while(n<Mofn.motifList.size()-1 && nt<Mofn.MAX_N){

                        n++;
                        RealCompositeMotif rcm = new
RealCompositeMotif((CompositeMotif)this.clone());
                        rcm.setName("expand");
                        rcm.expand(n);
                }
        }

        //CHECK RESULT,UPDATE AND FINISH
        if(singleMotifHolder.size()==Mofn.MAX_N){
                MotifVariant chkMV =
(MotifVariant)motVarHolder.get(makeKey(Mofn.MAX_M,Mofn.MAX_N));


    //Mofn.add_output(Integer.toString(chkMV.getSupport()));

    //Mofn.add_output(Double.toString(chkMV.getSign()));


    if(Mofn.pareto[chkMV.getSupport()]>chkMV.getSign()){

    Mofn.pareto[chkMV.getSupport()]=chkMV.getSign();

                        Mofn.patterns.put(new
Integer(chkMV.getSupport()),this.singleMotifHolder.clone());

                        for(int i=chkMV.getSupport();i>=0;i--){
                                if(chkMV.getSign()<Mofn.pareto[i]){
                                        Mofn.pareto[i] =
chkMV.getSign();

                                }//if
                        }//for
                }//if
        }//if
    }//function



    /**
    * Method added to override object clone
    * @return Copy of this module
    * @see Object
    */
    public Object clone(){

        try {
                return super.clone();
    }catch (Exception e) { // Dire trouble!!!
        throw new InternalError("But we are Cloneable!!!");
    }

    }
    }
```

FictiveCompositeMotif.java

```
import java.util.ArrayList;
import java.util.BitSet;

/**
 * This class is used for pruning, which narrow the DFS enumeration
 * @author Vetle Valebjørg
 * @version 1.0
 * @see CompositeMofif, RealCompositeMotif, SingleMotif
 */
public class FictiveCompositeMotif extends CompositeMotif implements
Cloneable{


	/**
	* Default constructor used by subclass Motifs
	* @see Motif
	*/
	public FictiveCompositeMotif(){
	}

	/**
	* Constructor used during cloning of composite elements
	*/
	public FictiveCompositeMotif(CompositeMotif cm){
		super(cm);
	}

	/**
	* Method used to prune and narrow the enumeration during DFS
	* @param n Position in motiflist
	* @param lsm_sign Last single motif significance.
	* @param left Distance left before MAX_N.
	* @return Best possible significance found during prune.
	*/
	public double prune(int n,double lsm_sign, int left){

		double bestres = lsm_sign;

		SingleMotif fsm = new SingleMotif(Mofn.nr_of_seq,
Mofn.nr_of_seq,lsm_sign);
		singleMotifHolder.add(fsm);

		int nt = singleMotifHolder.size();
		int t = calc(nt,fsm,"pruning");

		while(n<Mofn.motifList.size()-1 && nt<Mofn.MAX_N &&
left>0){

			n++;
			left--;
			//PRUNE, CHECK TO SEE IF SEARCH CAN BE STOPPED
			FictiveCompositeMotif fcm = new
FictiveCompositeMotif((CompositeMotif)this.clone());
			bestres = fcm.prune(n,lsm_sign,left);
			//if(nt==Mofn.Max_N){
			//	bestres = fcm.getSigngetSign();
			//}
			//if(singleMotifHolder.size()<Mofn.MAX_N){
			//	bestres = this.prune(n,lsm_sign);

			//}//if

		}//while
		//HashMap hm = fcm.getMotVarHolder();
		if(nt==Mofn.MAX_N){
			MotifVariant chkMV =
(MotifVariant)motVarHolder.get(makeKey(Mofn.MAX_M,Mofn.MAX_N));
```

```
                        bestres = chkMV.getSign();
                }
                return bestres;
        }//prune



        /**
        * Method added to override object clone, which then enables
cloning of CMs
        * @return A copy of this module
        * @see Object
        */
        public Object clone(){
                try {
                        return super.clone();
        }catch (Exception e) { // Dire trouble!!!
                throw new InternalError("But we are Cloneable!!!");
        }
        }
}
```

Client.java

```
import java.io.*;
import java.net.*;


/**
* Classed used buy the server to handle connections to clients
* @author Vetle Valebjørg
* @version 1.0
* @see Mofn
*/
class Client{
                private      Socket sock;
                private ObjectOutputStream outs;
                private ObjectInputStream ins;
                private ServerSocket server;
                private boolean running;
                public int start;
                public int stop;

                /**
                * Default constructor
                */
                public Client(){
                }

                /**
                * Starts a client connection with portnr, startpos and
stoppos in motiflist
                * @param portNr startnr.
                * @param c Offset for client to portnr.
                * @param start start position in motif group to handle.
                * @param stop End position in motif group to handle.
                */
                public Client(int portNr, int c, int start, int stop){
                        try{
                                server = new ServerSocket(portNr
+c,0,InetAddress.getLocalHost());

                                sock = server.accept();

                                outs = new
ObjectOutputStream(sock.getOutputStream());
                                ins = new
ObjectInputStream(sock.getInputStream());
```

```
                this.start = start;
                this.stop = stop;

                running = true;

        }catch(IOException e){
                System.out.println(e.getMessage());
        }
    }

    /**
    * Returns the objectoutputstream for the client handler
    * @return Link to stream used.
    */
    public ObjectOutputStream getOOS(){
        return outs;
    }

    /**
    * Returns the objectinputstream for the client handler
    * @return Link to stream used.
    */
    public ObjectInputStream getOIS(){
        return ins;
    }

    /**
    * Returns the connected clients status
    * @return Returns true if client is active
    */
    public boolean isRunning(){
        return running;
    }

    /**
    * Stops a client if needed
    * @throws IOException
    */
    public void stopClient() throws Exception{
        running = false;
        try{
                sock.close();
        }catch(Exception e){
                e.printStackTrace();
        }
    }

    /**
    * Returns the connections IP address
    * @return IP address used by the socket.
    */
    public InetAddress getIP(){
        return sock.getInetAddress();
    }
}
```

MotifVariant.java

```
import java.util.BitSet;

/**
* Class used to represent the motif variants calculated during
runtime
* @author Vetle Valebjørg
* @version 1.0
* @see CompositeMotif, RealCompositeMotif, FictiveCompositeMotif,
SingleMotif
*/
public class MotifVariant{

       private double significance;
       private int support;
       private BitSet bitset;
       private boolean dummy = false;
       private static int idcounter=0;
       private int id;

       /**
       * Default Constructor
       */
    public MotifVariant(){
       bitset = new BitSet();
       id = idcounter+1;
       idcounter++;
    }

    /**
    * Makes the motif variant a dummy variant which is used for
    * calculataions outside the top motif variant border
    */
    public void makeDummy(){
       significance = 0.0;
       bitset = new BitSet(Mofn.nr_of_seq+1);
       dummy = true;
       bitset.set(0,Mofn.nr_of_seq+1);
       support = bitset.cardinality();
    }

    /**
    * Makes the motif variant a first variant which is used for
    * calcualtions in the beginning of every motif variant column
    */
    public void makeFirst(){
             significance = 1.0;
             bitset = new BitSet(Mofn.nr_of_seq+1);
       //set all true
       bitset.set(0,Mofn.nr_of_seq+1);
       support = bitset.cardinality();
       }

    /**
    * Returns the unique ID for this motif variant
    * @return ID of the motif variant
    */
       public int getId(){
             return id;
       }

       /**
       * Returns true if motif variants is a dummy variant
       * @return True if isDummy true, otherwise false.
       */
       public boolean isDummy(){
             return dummy;
       }
```

```
       /**
       * Method used to calculate a new motif variant based on
predecessors and new single motif
       * @param mv1 Predecessor Motif variant diagonal
       * @param mv2 Predecessor Motif variant horizontal
       * @param sm1 New single motif
       * @return The new motifvariant
       * @see SingleMotif
       */
    public MotifVariant calc_mn(MotifVariant mv1, MotifVariant mv2,
SingleMotif sm1){
       //sign_c = P(a&b | d) = P( !( !(a&b) & !d) )
       //sign_c = 1 - (1 - sign_a * sign_b) * (1 - sign_d)
       significance  = 1-(1 - mv1.getSign()*sm1.getSign())*(1-
mv2.getSign());
  /*
       Mofn.add_output("MV1: " + mv1.getSign() + " id: " +
mv1.getId());
       Mofn.add_output("SM1: " + sm1.getSign());
       Mofn.add_output("MV2: " + mv2.getSign() + " id: " +
mv2.getId());

       Mofn.add_output("MV1b: " + mv1.getBitSet().toString());
       Mofn.add_output("SM1b: " + sm1.getBitSet().toString());
       Mofn.add_output("MV2b: " + mv2.getBitSet().toString());
  */
       //if(Double.isInfinite(log(significance))){
             //    significance =
Math.min(log(mv1.getSign())+sm1.getSign(),mv2.getSign());
             //}

       //bitstr_c = (bitstr_a AND bitstr_b) OR bitstr_d
       bitset.clear();

       bitset = (BitSet)mv1.getBitSet().clone();
       bitset.and(sm1.getBitSet());

       if(!mv2.isDummy()){
             bitset.or(mv2.getBitSet());
       }
       support = bitset.cardinality();

 //    Mofn.add_output("RES: " + bitset.toString() + "  SIGN:" +
getSign() + "  SUPP:" + getSupport() + " id: " + this.getId());
 //    Mofn.add_output("");

       return this;
    }

    /**
    * Returns significance calculated motif variant
    * @return Significance
    */
    public double getSign(){
       return significance;
    }

    /**
    * Returns support calculated in motif variant
    * @return The support
    */
    public int getSupport(){
       return support;
    }

       /**
       * Returns bitset calculated in motif variant
```

```
       * @return Bitset
       */
       public BitSet getBitSet(){
              return bitset;
       }

       /*
    public double log(double innum){
       return Math.log(innum)/Math.log(2);
    }
       */
}
```

InputReader.java

```
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.BitSet;
import java.io.*;

import java.math.*;
import java.util.HashMap;

/**
 * This class handles the input from files and test-data.
 * @author Vetle Valebjørg
 * @version 2.0
 */
public class InputReader{

       private String inFile = "data//input.txt";
       private int highestSeq = 0;
       HashMap freq = new HashMap();

       /**
       * Default constructor
       */
       public InputReader(){
              setDefaultSign();
       }

       /**
       * Constructor that takes a filename (String) as parameter)
       * @param str Filename as string
       */
       public InputReader(String mFile){
                     inFile = mFile;
                     setDefaultSign();
       }

       private void setDefaultSign(){
              freq.put("A", new Double(0.25));
              freq.put("C", new Double(0.25));
              freq.put("G", new Double(0.25));
              freq.put("T", new Double(0.25));
              freq.put("a", new Double(0.25));
              freq.put("c", new Double(0.25));
              freq.put("g", new Double(0.25));
              freq.put("t", new Double(0.25));
              freq.put(".", new Double(1.00));
       }


       /*
       public void makeTestMotifs(int number){
              Vector motpatterns = createRandomSequences(number,7);
              for(int i=0;i<number;i++){
```

```
                    SingleMotif testMotif = new
SingleMotif(20,20,0.90,motpatterns.get(i*2+1),new int[20],new
BitSet(20));
                    testMotif.setName(Integer.toString(i));
                    motifList.add();
              }
       }
       */

       /**
       * Creates a set of random sequence patterns
       * @param nrseq Number of random sequences
       * @param lengthseq Length of random sequences
       * @return Sequence list (name, patterns)
       */
       public static Vector createRandomSequences(int nrseq, int
lengthseq){

              int[] acgt = {65,67,71,84};
              Vector seqs = new Vector();

              for(int i=0;i<nrseq;i++){
                    String seq = "";
                    seqs.add(new Integer(i));

                    for(int l=0;l<lengthseq;l++){

                           seq=seq +
(char)acgt[((int)(Math.random()*4))];
                    }

                    seqs.add(seq);
              }
              return seqs;
       }

       /**
       * Calculates the frequencies for A,C,G and T in the
       * sequences provied (name, sequence).
       * @param seq List of sequences (name, pattern)
       */
       public void setSignBasedOnBackground(Vector seqs){
              double totA = 0.0;
              double totC = 0.0;
              double totG = 0.0;
              double totT = 0.0;

              for(int i=1;i<seqs.size();i+=2){
                    String backstr = (String)seqs.get(i);
                    backstr = backstr.toUpperCase();
                    int sumA = 0;
                    int sumC = 0;
                    int sumG = 0;
                    int sumT = 0;

                    for(int c=0;c<backstr.length();c++){

                           if(backstr.substring(c,c+1).equals("A")){
                                  sumA++;
                           }else
if(backstr.substring(c,c+1).equals("C")){
                                  sumC++;
                           }else
if(backstr.substring(c,c+1).equals("G")){
                                  sumG++;
                           }else{
                                  sumT++;
                           }
```

```
                }
                totA = (double)sumA/backstr.length();
                totC = (double)sumC/backstr.length();
                totG = (double)sumG/backstr.length();
                totT = (double)sumT/backstr.length();
        }

        freq.put("A", new Double(totA));
        freq.put("C", new Double(totC));
        freq.put("G", new Double(totG));
        freq.put("T", new Double(totT));
        freq.put("a", new Double(totA));
        freq.put("c", new Double(totC));
        freq.put("g", new Double(totG));
        freq.put("t", new Double(totT));
        freq.put(".", new Double(1.00));

    }

    /**
    * Returns number of sequences found in input-files
    * @return Nr of sequences found
    */
    public int getNrSeq(){
        ArrayList trash = readMotifs(10);
        //System.out.println("Seqnr: " + (highestSeq+1));
        return highestSeq;

    }

    /**
    * This method reads the data from the filename provied
    * @return Returns a list of single motifs objects read from
file.
    * @exception IOException
    * @see IOException
    */
    public ArrayList readMotifs(int nrofseq){
        ArrayList listOfMotifs = new ArrayList();

        try{

            FileReader myFIS = new FileReader(inFile);

            BufferedReader br = new BufferedReader(myFIS);
            String str;

            while((str = br.readLine())!=null){

                //System.out.println(str);

                String mName = "";
                double mSign=0.0;
                int mTotHit=0;
                int mAntSeq=0;
                int[] mSeqHitList = new int[0];
                BitSet mBitset = new BitSet(nrofseq);


                StringTokenizer stt;
                StringTokenizer sts;

                if(str.length()!=0 &&
!(str.startsWith("#"))){
                    stt = new StringTokenizer(str,"  ");

                        int calc = stt.countTokens();
```

```
                                        //System.out.println("Normal
sign"); ==4
                                        for(int i=0; i<=calc-(5-
calc);i++){
                                              if(i == 0){
                                                    mTotHit =
Integer.parseInt(stt.nextToken());

      //System.out.println("Tot treff: " + stt.nextToken());
                                              }else if(i == 1){
                                                    mAntSeq =
Integer.parseInt(stt.nextToken());

      //System.out.println("Ant sekv: " + stt.nextToken());
                                              }else if(i == 2){ //NOT
RUN IF MANUAL stt.count==3
                                                    mSign =
Double.parseDouble(stt.nextToken());

      //System.out.println("Sign: " + mSign);
                                              }//if

                                        }//for

                                  String newStr = stt.nextToken();
                                  sts = new StringTokenizer(newStr,"
");
                                  mName = sts.nextToken();
                                  //System.out.println("Mønster: " +
mName);

                                  //MANUAL CALC OF SIGN (IF NOT IN
INPUT)
                                  if(calc==3){
                                        //System.out.println(mName);
                                        mSign = calc_sign(mName);
                                        //System.out.println(mSign);
                                  }

                                  mSeqHitList = new
int[sts.countTokens()];
                                  int teller = 0;
                                  //System.out.println("Lengde: " +
mSeqHitList.length);

                                  while(sts.hasMoreTokens()){
                                        int listSeq =
Integer.parseInt(sts.nextToken());
                                        int listPos =
Integer.parseInt(sts.nextToken());



                                        if(listSeq>highestSeq){
                                              highestSeq = listSeq;
                                              /*
      System.out.println(listSeq);
                                              try{

      Thread.sleep(1000);
                                              }catch(Exception e){

      System.out.println("Søvn avbrutt");
                                              }*/
                                        }

                                        mBitset.set(listSeq,true);
```

```
                                        mSeqHitList[teller] = listSeq;
                                        mSeqHitList[teller+1] =
listPos;
                                        teller+=2;
                                        //System.out.println(listInt);
                                  }
                                  //System.out.println("Len: "+
mSeqHitList.length);

                                  //SETT INN FØRSTE I LISTA
                                  if(listOfMotifs.size() ==0){
                                        //System.out.println(mSign);
                                        SingleMotif newMotif = new
SingleMotif(mTotHit, mAntSeq, mSign, mName, mSeqHitList,mBitset);

      newMotif.setName(mName);//"Motif");//-"+listOfMotifs.size());
                                        listOfMotifs.add(newMotif);
                                  }else{
                                  //ELLER SORTER ETTER SYNKENDE
SIGNIFICANS

                                        SingleMotif newMotif = new
SingleMotif(mTotHit, mAntSeq, mSign, mName, mSeqHitList,mBitset);

      newMotif.setName(mName);//"Motif");//-"+listOfMotifs.size());
                                        //int ms = 0;

                                        //ASC
                                        for(int
i=0;i<listOfMotifs.size();i++){
                                              SingleMotif inListMotif
= (SingleMotif)listOfMotifs.get(i);


      if(newMotif.getSign()<=inListMotif.getSign()){

      listOfMotifs.add(i,newMotif);
                                                    newMotif=null;
                                                    break;
                                              }
                                        }
                                        if(newMotif!=null){

      listOfMotifs.add(newMotif);
                                              newMotif=null;
                                        };//(ms<= listOfMotifs.size()
&&

                                        /*
                                        //DESC
                                        do{
                                              SingleMotif inListMotif
= (SingleMotif)listOfMotifs.get(ms);

                                              if
(ms>=listOfMotifs.size()-1){

      listOfMotifs.add(newMotif);
                                                    newMotif=null;
                                              }else
if(newMotif.getSign()>=inListMotif.getSign()){

      listOfMotifs.add(ms,newMotif);
                                                    newMotif=null;
                                              }
                                              ms++;
                                        }while
(newMotif!=null);//(ms<= listOfMotifs.size() &&
```

```
                                        */
                                    }//if

                              }//if
                        }//while

                        myFIS.close();

                  }catch(Exception e){
                        //ERROR READING INPUT
                        System.out.println(e);
                  }

                  return listOfMotifs;
            }//function

            /**
            * Returns the filename used.
            * @return filname
            */
            public String getFile(){
                  return inFile;
            }

            /**
            * This method takes the 2-logarithm of a double
            * @param num as double
            * @return Answer as 2-log of num
            */
            private double log2(double num){
                  return Math.log(num);
            }

            /**
            * This method calculates the significance for a pattern
            * based on the frequency of letters and sequence length. (log
removed since previous project.)
            * @param pattern String representation of pattern
            * @return The significance found.
            */
            private double calc_sign(String pattern){

                  int datalen = 1000;
                  if(Mofn.sequenceList.size()>0){
                        datalen = Mofn.getAvgSeqLen();
                  }



                  double sign = 1.0;
            double logsign = 0.0;
            double ambsign = 1.0;
            int length = 0;
            boolean inamb = false;

                for(int i=0;i<pattern.length()-1;i++){
                    String keystr = pattern.substring(i,i+1);
                    //System.out.println(keystr);

//System.out.println(((Double)freq.get(keystr)).doubleValue());

                    //A..G.AAAA
                    if(!inamb){
                        if(freq.containsKey(keystr)){
                            logsign +=
log2(((Double)freq.get(keystr)).doubleValue());
                            sign = sign *
((Double)freq.get(keystr)).doubleValue();
```

```
                    length += 1;

                }else if(keystr.equals("[")){
                    ambsign = 0.0;
                    inamb = true;
                    continue;
                }
             }else if(freq.containsKey(keystr)){
                 ambsign +=
((Double)freq.get(keystr)).doubleValue();

             }else if(keystr.equals("]")){
                 sign *= ambsign;
                 logsign += log2(ambsign);
                 length += 1;
                 inamb = false;

             }else{
                 System.out.println("unknown symbol in pattern");
             }
           }

         if(length>=datalen){
             //return logsign;
             return sign;
         //}else if(logsign < -40){
             //System.out.println("sign1");
             //return (logsign + log2(datalen - length));
           //  return (sign + datalen - length);

         }
         else{
            //System.out.println("Sign2");
             //return (log2(1.0 - Math.pow(1.0 - sign, datalen -
length)));
             return 1.0 - Math.pow(1.0 - sign, datalen - length);

         }//if
      }//for

      /**
      * Reads next number used with output files
      * @param nrFile Filename containing the next output nr
      * @return The next number that should be used.
      */
      public String readNextOutputFileNr(String nrFile){
            String retStr = "";

            try{

                  FileReader myFR = new FileReader(nrFile);

                  BufferedReader br = new BufferedReader(myFR);

                  retStr = br.readLine();

                  myFR.close();

            }catch(Exception e){
                  //ERROR READING INPUT
                  System.out.println(e);
            }

            return retStr.trim();
      }

      /**
      * Reads all PWM matrices found in file provided
```

```
	* @param filename Filename
	* @return Motiflist with PWM motifs
	*/
	public ArrayList readMatrix(String filename){
		ArrayList listOfMotifs = new ArrayList();

		try{

			FileReader myFR = new FileReader(filename);
			BufferedReader br = new BufferedReader(myFR);

			String mName = "";
			int cols = 0;
			int rows = 0;
			int countRows = 0;
			Vector matrix = new Vector();

			boolean moretoread = true;

			String str;
			while(moretoread){
				str = br.readLine();

				if(str==null){
					str =">avslutt";
					moretoread = false;
				}

				if(!str.startsWith("#")){

					if(str.startsWith(">") &&
countRows==0){
						mName = str.substring(1);
					}else if(!str.startsWith(">")){

						//System.out.println("str: " +
str);

						StringTokenizer stt = stt =
new StringTokenizer(str," ");
						//System.out.println(str);
						countRows++;

						cols = stt.countTokens();

						while(stt.hasMoreTokens()){

							//System.out.println("-"
+ sttSub.nextToken()+"-");
							matrix.add(new
Double(stt.nextToken()));
						}
					}

					if(str.startsWith(">") &&
countRows!=0){
						rows = countRows;

						double[][] mList = new
double[rows][cols];
						double[] rowsum = new
double[rows];
						//System.out.println(">" +
mName);
						for(int r=0;r<rows;r++){
							for(int c=0;c<cols;c++){
								double value =
((Double)matrix.elementAt(r*4+c)).doubleValue();
```

```
	//System.out.print(value + " ");
                                                        mList[r][c] =
value;
                                                        rowsum[r]+=value;
                                                }

	//System.out.println(rowsum[r]);
                                        }

                                        SingleMotif sm = new
SingleMotif(0,0,0.0,mName,new int[Mofn.nr_of_seq+1],new BitSet());

	sm.setMatrix(mList,rows,cols,rowsum);
                                        sm.setName(mName);
                                        listOfMotifs.add(sm);
                                        rows = 0;
                                        cols = 0;
                                        countRows=0;
                                        matrix = new Vector();
                                        mName = str.substring(1);

                                }//if
                        }//#
                }//while

                myFR.close();

        }catch(Exception e){
                e.printStackTrace();
        }//try

        return listOfMotifs;
	}//func


	/**
	* Reads all sequences found in file provided
	* @param filename Sequence filename
	* @return Sequences with alternating (name, pattern)
	*/
	public Vector readSequences(String filename){

        Vector sequences = new Vector();

        try{

                FileReader myFR = new FileReader(filename);
                BufferedReader br = new BufferedReader(myFR);
                String str;

                while((str = br.readLine())!=null){

                        String mName = "";
                        String sequence = "";

                        StringTokenizer stt;

                        if(str.length()!=0 &&
!(str.startsWith("#"))){
                                stt = new StringTokenizer(str,"  ");

                                if(str.startsWith(">")){
                                        mName=stt.nextToken();

                                        mName = mName.substring(1);
                                        sequences.add(mName);
                                }else{
```

```
        sequences.add(stt.nextToken());

                                    }//else
                              }//if
                        }//while

                        myFR.close();

                  }catch(Exception e){
                        e.printStackTrace();
                  }//try
                  highestSeq = sequences.size()/2;

                  return sequences;
            }//func

}//class
```

OutputWriter.java

```
import java.util.ArrayList;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.BitSet;
import java.io.*;

import java.math.*;
import java.util.HashMap;

/**
 * This class handles the output to files.
 * The calculations are done in Mofn and passed to functions here.
 * @author Vetle Valebjørg
 * @version 1.0
 * @see Mofn, InputReader
 */
public class OutputWriter{

	private static String nextCounterFile = "nr.txt"; //contains
next output nr
	private static String outputFile = "output"; //output filename
	private static String outputDir ="output//"; //output directory

	/**
	* Default constructor
	*/
	public OutputWriter(){
	}

	/**
	* Constructor with output file as parameter
	* @param newout Filename
	*/
	public OutputWriter(String newout){
		outputFile = newout;
	}

	/**
	* Reads, increments and writes new output file number.
	*/
	private void renewOutputNr(){
		//read number
		InputReader ir = new InputReader();
		String nrStr = ir.readNextOutputFileNr(nextCounterFile);

		outputFile = outputFile + nrStr + ".txt";

		//write new number
		int newnr = Integer.parseInt(nrStr)+1;
		writeNextOutputFileNr(String.valueOf(newnr));
	}

	/**
	* This method writes the next output file number to file
	* @exception IOException
	* @see IOException
	*/
	public void writeNextOutputFileNr(String wStr){

		try{

			FileWriter myFW = new FileWriter(nextCounterFile);
			BufferedWriter bw = new BufferedWriter(myFW);

			myFW.write(wStr);
			myFW.close();
```

```
            }catch(Exception e){
                        String errorstr = "writeNextOutputFileNr";
                        System.err.println("Error during " +
errorstr);
            }
      }//function

      /**
      * Prints information from the run to file
      * @param strings List of strings to write to file.
      */
      public void print_output(ArrayList strings){
            //PRINTING OUTPUT
            //System.out.println(outputFile);

            try{
                  renewOutputNr();
                  BufferedWriter myBW = new BufferedWriter(new
FileWriter(outputDir + outputFile));


                  for(int x=0;x<strings.size();x++){
                        myBW.write((String)strings.get(x));
                        myBW.newLine();
                  }

                  myBW.close();
            }catch(Exception e){
                  e.printStackTrace();
                  //System.out.println("Output file error");
            }
      }


      /**
      * Returns current output file
      * @return Filename used.
      */
      public String getFile(){
            return outputFile;
      }

}//class
```

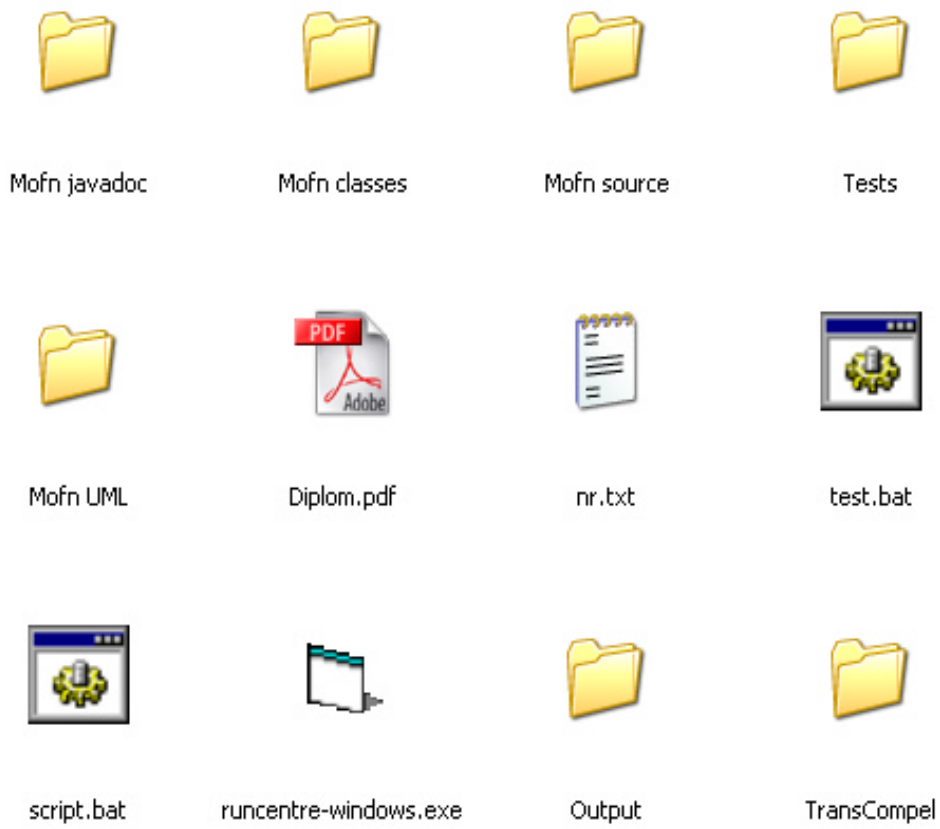
SingleMotif.java

```
import java.util.BitSet;

/**
 * This class contains the structure for motifs used by Mofn
(IUPAC/PWM)
 * @author Vetle Valebjørg
 * @version 2.0
 * @see InputReader, Mofn
 */
public class SingleMotif{
	private String name = "MotifX";
	private double significance;
	private String pattern;
	private	int nrOfHits;
	private int nrOfSeq;
	private int[] listOfSeq;
	private BitSet mBitList;

	//spesielt for matriser
	private double[][] matrix;
	private double[] rowsum;
	private int rows;
	private int cols;

	/**
	* Default constructor
	*/
	public SingleMotif(){
	}

	/**
	* Constructor taking all the values from the InputReader as
input
	* @param mHits Total hits as int., bitlist as BitSet
	* @param mSeq Hits as distinct sequences.
	* @param mSign Significance.
	* @param mMotif Pattern if(iupac, name if pwm).
	* @param mHitList list of hits and sequences, alternating seqnr
and posnr.
	* @param mbList Bitlist represention of hits, true false for 0
-> seq size.
	* @see BitSet
	*/
	public SingleMotif(int mHits, int mSeq, double mSign, String
mMotif, int[] mHitList,BitSet mbList){
		pattern = mMotif;
		significance = mSign;
		nrOfHits = mHits;
		nrOfSeq = mSeq;
		listOfSeq = mHitList;
		mBitList = mbList;
	}

	/**
	* Constructor used by the prune-method
	* Constructs best motif possible based on prev. motifs
	* @param  mhits Total hits.
	* @param mSeq Distinct hit in seq.
	* @param mSign Significance
	*/
	public SingleMotif(int mHits, int mSeq, double mSign){
		pattern = "";
		significance = mSign;
		nrOfHits = mHits;
		nrOfSeq = mSeq;
		listOfSeq = new int[0];
```

```
            mBitList = new BitSet(Mofn.nr_of_seq+1);
        //set all true
        mBitList.set(0,Mofn.nr_of_seq+1);

            //mBitList = new BitSet(mSeq+1); //in this case, since
max
            //mBitList.set(0, mSeq);
        }

        /**
        * Returns pattern representation as string.
        * @return pattern as string
        */
        public String getPattern(){
            return pattern;
        }

        /**
        * Returns matrix used to represent PWMs
        * @param Matrix values in PWM.
        */
        public double[][] getMatrix(){
            return matrix;
        }

        /**
        * Returns a vector with the sum for each row in the PWM matrix
        * @param Vector with sum for each matrix row.
        */
        public double[] getRowSum(){
            return rowsum;
        }

        /**
        * Returns the number of rows in a PWM, length of the single
motif
        * @param Number of rows in PWM matrix.
        */
        public int getRows(){
            return rows;
        }

        /**
        * Returns the number of columns in a PWM, always 4 though
        * @param Number of columns in PWM matrix.
        */
        public int getCols(){
            return cols;
        }

        /**
        * Returns the motif name. An ID is used if no name provided.
        * @param Motif name or ID created.
        */
        public String getName(){
            return name;
        }

        /**
        * Returns the motif significance based on the model used
        * @return Significance
        */
        public double getSign(){
            return significance;
        }

        /**
        * Returns the BitSet used to represent sequence hits
```

```
	* @return Bitset representation of the support.
	*/
	public BitSet getBitSet(){
		return mBitList;
	}

	/**
	* Returns the hit data provied or found as a list
	* @return List of alternating seqnr and hitpos
	*/
	public int[] getHitData(){
		return listOfSeq;
	}
	/**
	* Returns the support found, bitset.cardinality
	* @return Support
	*/
	public int getSupp(){
		return nrOfSeq;
	}

	/**
	* Sets the significance based on new calculations
	* @param newSign New significance
	*/
	public void setSign(double newSign){
		significance = newSign;
	}

	/**
	* Sets the motif name found
	* @param newName New name
	*/
	public void setName(String newName){
		name = newName;
	}

	/**
	* Sets the PWM matrix
	* @param m PWM matrix
	* @param r Rows
	* @param c columns
	* @param sum Vector with row sums
	*/
	public void setMatrix(double[][] m, int r, int c, double[]
sum){
		matrix = m;
		rows = r;
		cols = c;
		rowsum = sum;
	}

	/**
	* Sets hit data found
	* @param bs New bitset representation of hits in sequences
	* @param hitList Hit data with sequence number and position
number for hits
	* @param htot Hits total
	* @param hseq Distinct number of sequences
	*/
	public void setHitInfo(BitSet bs, int[] hitList, int htot, int
hseq){
		nrOfHits = htot;
		nrOfSeq = hseq;
		mBitList = (BitSet)bs.clone();
		listOfSeq = hitList;
	}
	}
```

Appendix G – Content of the attached ZIP-file



Source code:

- **Mofn.java**
- **CompositeMotif.java**
- **RealCompositeMotif.java**
- **FictiveCompositeMotif.java**
- **MotifVariant.java**
- **SingleMotif.java**
- **InputReader.java**
- **OutputReader.java**
- **Client.java**