The `NestedMICA` motif finder

Thomas A. Down
Wellcome Trust Sanger Institute, Hinxton, Cambridge CB10 1SA
td2@sanger.ac.uk

Version 0.01 [20041006]

Abstract

NestedMICA is a new, scalable, pattern-discovery system, aimed at finding transcription factor binding sites and similar motifs in biological sequence. This document describes a preliminary version of NestedMICA – it needs more testing and polishing before public release. Any text in [square brackets] is a note by the author, and probably ought to be removed or replaced...

If you have any problems or questions about the system, please contact Thomas Down.

1 Theory

Motif-finding is a long standing problem in sequence bioinformatics. A typical statement of the problem would be "given a set of sequences, which motifs are significantly over-represented with respect to a given background model." In fact, background models are often ignored but they are very important for real motif-finding applications, as we will discuss later.

The classical use for motif-finding software is the detection of transcription factor binding sites in promoter regions, but there are other interesting functional elements in biological sequences and elsewhere which can be found by motif-discovery methods.

Motif-finding strategies can be broadly divided into two classes: those which rely on exhaustively enumerating a set of motifs (generally using an optimized data structure such as a suffix tree) then reporting the most frequent, and those which find significant motifs by optimizing or drawing samples from a probabilistic model. Exhaustive enumeration is a good strategy for finding perfectly conserved motifs (*i.e.* every instance is identical), but for typical transcription factor binding sites, which often have several weakly constrained positions, exhaustive enumeration becomes problematic and the results usually have to be post-processed with some kind of clustering system. We will not consider exhaustive enumeration further.

Probabilistic motif-finders treat the supplied sequences as a mixture of 'interesting' motifs and 'non-interesting' background sequences. This problem has classically been reduced to a simple case: considering just one motif at a time, model each

sequence with a random background model which may, or may not, contain a single instance of the motif under consideration. This is the zero-or-one occurance per sequence (ZOOPS) model. It can be easily represented as a hidden Markov model (figure XX), and standard techniques such as expectation maximization or Gibbs sampling can be used to find good sets of model parameters (corresponding to good motif models).

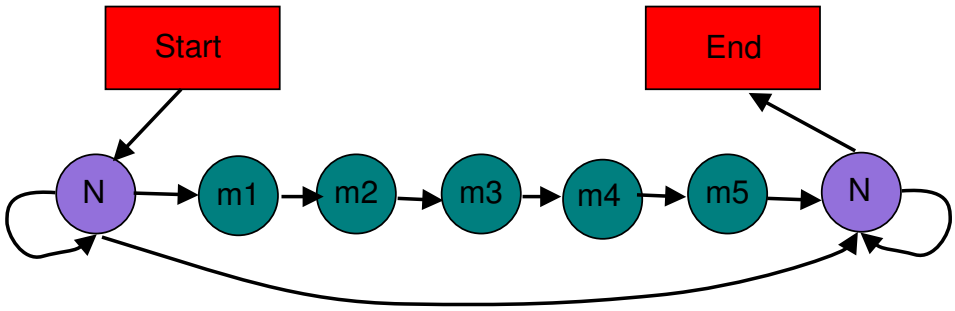

Figure 1: The ZOOPS sequence mixture model

There are two significant concerns about this strategy which NestedMICA tries to address

- Real regulatory regions, and most other contexts where interesting motifs can be found, don't really contain just a single instance of a single motif. Programs which use the ZOOPS model work around this by finding the strongest motif in a set, then scanning for all its instances, masking them out, and re-running the process on the remaining sequence. This strategy is greedy and it is by no means clear that its behaviour will be optimal, especially when working on a system where there is a set of closely related, yet still distinct, motif types. In an environment where novel transcription factors are created by gene duplication then diverge to perform a new function, such situations seem quite probable, and have not yet been well investigated.

- Existing techniques for optimizing or exploring sequence models of this type tend to be strongly local in nature – they concentrate on regions of the probability landscape close to their starting point. This is particularly clear for expectation-maximization methods, which always move in a direction which increases the likelihood of the model. Sampling strategies don't strictly have this limitation, but in practice, crossing the low-likelihood valley between two high-likelihood peaks is unlikely, often to the point where it becomes vanishingly rare.

1.1 Motif ICA

We treat finding multiple motifs in a set of sequences as a form of independent component analysis (ICA) problem. In linear ICA, a matrix of observations, X is approximated as a linear mixture, A of some sources, s:

$$x = As + \nu \tag{1}$$

(where ν is a noise matrix). A classical example is the "cocktail party problem" where a set of M microphones record different mixtures of the voices of N speakers. Given samples from these microphones at t timepoints, ICA attempts to factorize the Mxt observation matrix into a Nxt source matrix and a MxN mixing matrix.

It's possible to generalize ICA to *any* mixing situtation...

In motif ICA, the sources are short sequence motifs (currently, but not necessarily, modeled as position-weight matrices [1]), while the observations are larger sequences. There are a number of interpretations of the mixing matrix. Currently, we use a boolean mixing matrix (all coefficients are either 0 or 1), and a given sequence is expected to contain a given motif if the relevent mixing coefficient is 1. The 'noise' part of the ICA model represents all the sequence which isn't modelled by one of the motifs.

1.2 Nested sampling

Nested sampling is an alternative way of performing probabilistic inferene in a Bayesian framework, proposed recently by John Skilling [2]. It can be considered to be a Monte Carlo method, since the process is driven forward by a series of randomly chosen events, but it is quite distinct from the classic Metropolis-Hastings method, and methods such as Gibbs sampling and slice samplings, which can all be considered as optimized implementation strategies for Metropolis-Hastings.

[probably ought to write some sort of description of NS].

In this context, the most exciting propertly of nested sampling is that, given a reasonably large ensemble, the final sample drawn from a converged nested sampler can be expected to reflect the *global* optimum of hte likelihood landscape. Moreover, in cases where more than one globally significant optimum exists, these should be represented in the sample set in direct proportion to the amount of posterior mass they represent.

[NestedMICA 1 only makes full use of the first of these properties. Proper interpretation of multiple samples is difficult because of the inherent symmetry in our mixture models. NestedMICA 2 will offer faster convergence by taking advatage of multiple samples]

1.3 Background models

The background model is an important component of the SMM framework – after all, it will usually be responsible for modeling the majority of the input sequence! The simplest strategy – and still a common one – is to treat all non-motif sites as

independent and identically distributed. In HMM terms, this makes the background model a zeroth-order Markov chain. However experience shows that genomic DNA sequence, even when apparently totally non-functional, is not a good fit to the i.i.d. model. The best known deviation is perhaps the dramatic under-representation of CpG dinucleotides in most parts of vertebrate genomes, but other significant effects are known. In any case, practical experience shows that motif finders equipped with naive background models tend to report low-complexity elements rather than interesting binding sites.

The first obvious improvement is to replace the zeroth order Markov chain with a first order chain (*i.e.* the probability of observing a particular symbol at position n depends on the symbol at position $n-1$). This model is good at capturing anomalies like the CpG underrepresentation. The success of first order background models has led some researchers to investigate higher order models. One investigation of Markov chain backgrounds can be found in [ref]: this concludes that pentanucleotide frequency tables (*i.e.* fourth-order background models) are optimal. However, there are two concerns about this result: firstly, it leaves an open question about what these high-order correlations in background sequence mean (and why fourth-order models appear to outperform fifth-order). Also, training a background model generally requires sequence propoortional to the number of free parameters in the model. Fifth order models, with 768 parameters, therefore require large amounts of sequence. Moreover, it is desirable to train the background model on sequence which does not contain target motifs, since a fifth order model could easily capture some information about this motifs, thereby reducing the sensitivity of the motif-finding process. But it is hard to find large amounts of representative background training sequence which doesn't contain interesting motifs.

[Maybe this explains why fifth order is optimal – higher order models always capture more motif information than you gain by improving the background fit].

A different way to generalize the naive background model is to allow several different classes of sequence, each with its own particular base distribution (which could be zeroth-order or higher-order). We call these *mosaic models*, since their underlying assumption is that genome evolution includes some set of constraints which act non-uniformly, even on background sequence.

To investigate the benefit of mosaic models, we took a set of human upstream flanking regions, and split the set in half, using one subset to optimize the background model parameters and the second subset for testing. Test likelihoods for a variety of class numbers and Markov chain orders are shown in figure 2. Considering one class 'mosaics' (which are equivalent to uniform background models), we repeat the previously reported observation that higher order Markov chains are better models of genomic DNA. However, we also see large increases in likelihood when moving to larger numbers of mosaic classes. Interestingly, the lines for zeroth-order and first-order models run almost paralell: this suggests that the benefits of mosaic models are almost orthogonal to the benefits of first-order models. However, this is not true when moving beyond first-order models.

Based on these results, we recommend the use of a four class, first order, mo-

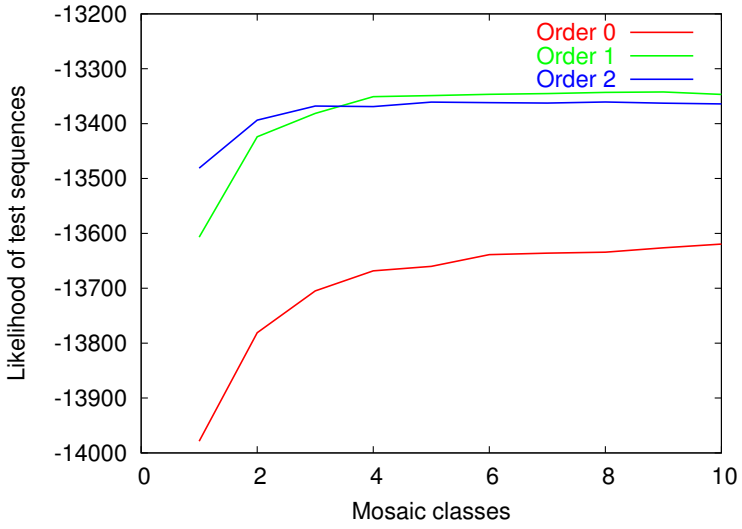

Figure 2: Comparison of mosaic backgrounds

saic background model for most motif-finding applications on mammalian genomic sequence. In practice, the four classes appear to be:

- A relatively neutral class
- A C+G rich class (CpG islands?)
- Purine rich
- Pyrimidine rich

[Probably the observation that > first order Markov chains appear to be helpful is artifactual – if several of the previous bases were pyrimidines, it's likely that you are in a pyrimidine-rich block, and therefore the probability of observing another pyrimidine increases. It ought to be possible to test this on synthetic data sampled from a mosaic model.]

There are still open questions about the biological significance of the mosaic classes we have observed...

2 Running NestedMICA

NestedMICA is open source software, released under the GNU Lesser GPL. The main program is written in Java, but there are a few small pieces of native C code which are required. The C code is simple and (hopefully) portable, so it should be possible to run NestedMICA on most modern computer platforms.

Currently, the best way to obtain NestedMICA is direct from the development source code repository, using Subversion (http://subversion.tigris.org/). To check out the latest version:

```
svn checkout http://www.derkholm.net/svn/repos/nmica/trunk nmica
```

You will also need up-to-date versions of `biojava.jar` and `bytecode.jar`, both available from http://www.biojava.org/. Finally, it is recommended that you use `changeless.jar`, a small patch to BioJava which disables the ChangeEvent infrastructure. If you are using `changeless.jar`, it *must* be added to the `CLASSPATH` before `biojava.jar`. To compile the Java code, you'll need a recent version of the ANT java build tool, which can be downloaded from `http://ant.apache.org/`.

2.1 Compilation

The main body of the NestedMICA code is written in Java. To compile this, copy the dependancy JAR files into the top level of the source tree, then type `ant`. This will compile all the Java code and package it as a JAR file, located in `ant-build/nmica.jar`.

For performance reasons, there is also a small amount of C code, which must be compiled separately. To compile it, change into the `native` directory and type:

```
make -f Makefile.PLATFORM
```

Currently supported platforms are `OSX` (Mac OS X), `gcc` (most Intel platforms which use the GCC compiler), and `icc` (Linux with the proprietary Intel ICC compiler). The Linux/Unix makefiles require that you have the `JAVA_HOME` environment variable set to point to your JDK installation. The native code is generally called something like `libnmica.so`, but this can vary a bit depending on platform, *e.g.* `libnmica.jnilib`, `nmica.dll`.

NestedMICA currently doesn't require any special installation of the native code. However, you need to se the `java.library.path` system property to point to the directory containing the compiled native library. On most platforms, the easiest way to do this is to invoke the NestedMICA commands like:

```
java -Djava.library.path=/path/to/native/code CommandClass
```

Where `CommandClass` is the NestedMICA command you want to run. It's also possible to write some wrapper scripts to do this automatically – future versions of NestedMICA may have an installer to generate a set of wrapper scripts.

2.2 Basic operation

This section describes a (very) small set of test sequences named `micatest100.fa`. This consists of 100 synthetic sequences, each of 50 bases long. The set contains two spiked-in motifs.

Given a new dataset, the first step is to obtain or train a suitable background model. In this case, a simple model can be trained using:

```
java net.derkholm.nmica.apps.MakeMosaicBackground -seqs micatest100.fa
     -mosaicClasses 1 -mosaicOrder 1 -out mt100.sbg
```

Note that, despite the name of the program, this won't be a true mosaic background, since it is limited to a single class. If the `-mosaicClasses` paramater is increased, mosaic models with arbitary numbers of sequence types can be trained using this program.

[Future versions may come with a tool for automatically optimizing the background model architecture].

Once the background model is trained, you can start the motif finder using a command like:

```
java net.derkholm.nmica.apps.MotifFinder -seqs micatest100.fa
      -backgroundModel mt100.sbg -numMotifs 2 -sampleFile mt100-motifs
      -sampleInterval 1000 -numCycles 10000
```

This runs the nested sampler for 10000 cycles, keeping a sample every 1000 cycles. Sample files have names like `mt100-motifs.1000.jos`, so you can watch the progress of the trainer.

[There currently isn't a proper termination condition implemented, so you *must* specify a sensible value for the `-maxCycles` option. Future versions will have automatic termination, probably by monitoring the interquartile range of the nested ensemble.]

To view a sample file:

```
java net.derkholm.nmica.apps.MotifViewer mt100-motifs.10000.jos
```

This runs a simple viewer application, as shown in figure 3. If your test run doesn't find the two motifs shown here, please contact the author. MotifViewer has some simple capabilities for arranging and annotating motif sets [more about this once I've sorted out the UI...]

The sample files contain motif data in a NestedMICA-specific binary format (actually serialized instances of BioJava `WeightMatrix` objects).

2.3 Motif-finding options

The `MotifFinder` program has many options. A complete list can be seen by viewing the Java source code: whenever you see a method named `setFoo`, it means that `-foo` can be used as a switch on the command line. These option are, however, gradually being rationalized. Undocumented options may disappear in the future!

All boolean options can be negated by prepending `no`, but this follows Java capitalization conventions *e.g.* the negation of `-revComp` is `-noRevComp`.

Figure 3: Motifs discovered from the demonstration set.

2.4 Everyday options

These are options that you'll use all the time, and are generally well-behaved.

`-numMotifs` *n* specifies the number of motifs you want to find (default=1).

`-targetLength` *n* the maximum length of motifs to find (default=10).

`-seqs` *filename* a FASTA-formated database of sequences to analyse.

`-backgroundModel` *filename* the background model to use, normally generated by running the `MakeMosaicBackground` program.

`-ensembleSize` *n* the size of the nested ensemble used to explore the probability landscape. Larger values improve reproducibility at the expense of speed (default=200, which is probably a good choice for mid-sized problems).

`-revComp` allow motifs to occur in either orientation (off by default, slows the program down by a factor of about 2.

`-cluster` prefer motifs which occur in clusters. Causes a small slowdown in training. Currently not sure how helpful this is.

2.5 Scary options

More complex options, that We'd prefer weren't there. If you're interested in these, it's probably best to discuss them with Thomas Down.

`-mixtureType (binary|flat|logit|weighted)` Wierd switch which effects the interpretation of the mixing matrix. Default is `binary`, leave it that way unless you know what you're doing.

`-mixtureUpdate (resample|weakResample|max|queue|random)` Strategy for updating the mixing matrix during training. `resample` is the best in theory and works well for most problems. `weakResample` gives better results on large sets of human promoters. Don't use any of the others.

`-expectedUsageFraction` *d* number between 0.0 and 1.0 giving the prior belief of motif frequency. Default is 0.5. Try 0.1 if you think motifs are rare, 0.9 is you think almost all the sequences are likely to contain an instance.

`-crossOverProb` *d* controls how rapidly information is exchanged through the nested ensemble. Values > 0.5 may increase convergence speed, but increase the risk of learning duplicate motifs.

`-replaceComponentProb` *d* controls how often the trainer tries to completely resample a model component. Default is 0.2 which is probably good for most purposes.

2.6 Broken options

Please don't touch these!

`-uncounted` Counted models are currently badly supported – they'll either be re-implemented or ditched completely in the next release.

`-simplex, -native, -clip` These are now all on by default, can't think of any reason to turn them off. Switches may be removed soon.

2.7 Checkpointing

To minimize the risk of data loss during long runs, NestedMICA can generate checkpoint files at regular intervals during the training process. If you want to keep checkpoint files, add the following arguments to your `MotifFinder` command line

```
-checkpoint mycp -checkpointInterval 1000
```

Like sample files, checkpoints are automatically numbered with the cycle number at which they are taken. Unlike samples they files are rather large (typically several megabytes, increasing rapidly with dataset and nested ensemble size). By default, `MotifFinder` deletes old checkpoints: in the standard configuration only two checkpoints are kept on disk, so after the third is written, the first will be deleted. This behaviour can be changed using the `-keepCheckpoints` option.

To restart from a checkpoint:

```
java net.derkholm.nmica.apps.MotifFinder
      -restartFromCheckpoint mycp.12000.jos
      -maxCycles 100000 -sampleFile sample -sampleInterval 10000
      -checkpoint mycp2 -checkpointInterval 1000
```

Note that, while the entire trainer state is restored from the checkpoint file, currently none of the 'housekeeping' options (termination, sampling, and checkpointing) are, so it is necessary to re-specify them when restarting the checkpoint. This is potentially useful if you want to continue the training of a run which hasn't quite converged for a few extra cycles.

2.8 Alternate background models

There is an alternative background model designed specifically for finding non-coding motifs embedded within protein-coding sequence. These are trained using the `MakeCodingBackground` program. Contact Bernard Leong for more details.

To use a totally custom background model, write a new implementation of the `net.derkholm.nmica.model.motif.SequenceBackground` interface, then construct a suitably parameterized instance and write it to disk using the standard Java serialization mechanism.

2.9 Multithreaded operation (SMP machines)

If you are running on a multi-processor machine, add the option `-workerThreads N` to the `MotifFinder` command line, where `N` is the number of processors in the machine, or the number of CPUs in a large multiprocessor server that you want to devote to `MotifFinder`. This mode of operation is quite efficient, often giving better than 95% scaling from one to two processors.

2.10 Distributed operation (clusters, farms, Beowulfs, etc.)

An alternative approach to paralelizing a large `MotifFinder` run is distribute the workload over multiple nodes. Start `MotifFinder` as usual, but add the options:

```
-distributed -port XXXXX
```

You may use any port number between 1024 and 65535, but if you are running multiple `MotifFinder` jobs on one cluster, you should give each job a unique port number. This process will be your *master node*. Once it has started up, you can run any number of *worker nodes*, which will evaluate sub-problems for the master node. To run a worker node:

```
java net.derkholm.nmica.apps.MotifFinderNode -server YYYYY -port XXXXX
```

Where `YYYYY` is the name (or IP address) of the master node, and the port number matches that used to start the master. To remove a worker node from the cluster you can simply kill the process. It's fine to add and remove workers at any point during the run, although there is currently a short ($\tilde{1}5$ seconds) glitch after a worker dies during which no work will be done. If the `MotifFinder` run ends normally, all associated worker nodes will be sent a shutdown message, but under other circumstances it may be necessary to kill worker nodes manually.

The distributed training system is currently optimized for homogenous clusters. Jobs are handed to worker nodes in roughly equal-sized packages, so in a heterogenous cluster, overall throughput will be limited by that of the slowest node.

2.11 Care-free comparative genomics

If groups of orthologous regulatory regions are available, NestedMICA can take advantage of the known relationships between species. Unlike phylogenetic footprinting and other 'infinite monkeys' techniques, comparative NestedMICA does not consider the alignments of the sequences, but simply assumes that orthologous regulators are likely to contain similar complements of motifs. In practice, this is implemented by using a single row of the ICA mixing matrix to model all the orthologs.

To run NestedMICA in comparative mode, just add more than one `-seq` option to the `MotifFinder` command line. If two sequences in different files share the same name, they will be treated as orthologues.

There are two options for providing background models in comparative mode. If only one `-backgroundModel` switch is present on the command line, that single background model will be used for all species. Alternatively, you can train a separate background model for each species, and specify them with multiple `-backgroundModel` switches, e.g.

```
-seq human.fa -backgroundModel human.sbg
-seq mouse.fa -backgroundModel mouse.sbg
```

If you use multiple background models they *must* be listed in the same order as the sequence databases, and there must be exactly equal numbers of sequence databases and background models.

If ortholog information is only available for some sequence, that's fine.

There is no limit to the number of species that can be used (except for practical memory limits). We would expect to see diminishing returns beyond about 4 species, but much more testing is needed to determine the optimal configurations.

[When using more that two species, there ought to be some option to specify approximate branch lengths, and weight the contributions accordingly.]

References

[1] Bucher P: **Weight matrix descriptions of four eukaryotic RNA polymerase II promoter elements derived from 502 unrelated promoter sequences**. *Journal of Molecular Biology* 1990, **212**:563–578.

[2] Skilling J: **Nested Sampling**[[http://www.inference.phy.cam.ac.uk/bayesys/]].