

# The NestedMICA motif finder

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

Version 0.7.3 [20061103]

## Abstract

NestedMICA is a new, scalable, pattern-discovery system, aimed at finding transcription factor binding sites and similar motifs in biological sequence. More discussion of the principles behind NestedMICA, and an evaluation of its sensitivity can be found in a recent paper [?].

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 occurrence per sequence (ZOOPS) model. It can be easily represented as a hidden Markov model (figure 1)), 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).

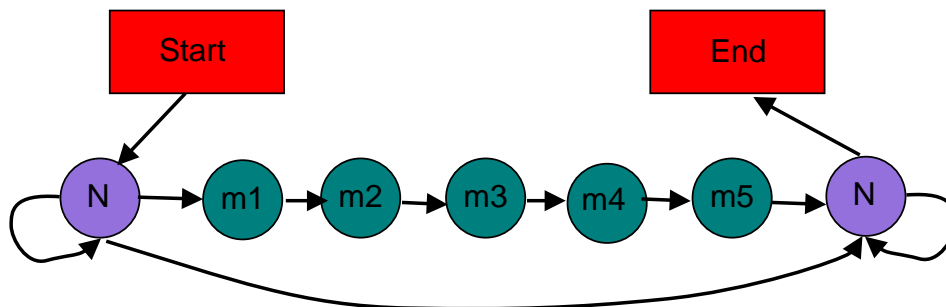


Figure 1: The ZOOPS sequence mixture model

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  $\nu$  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  $M \times t$  observation matrix into a  $N \times t$  source matrix and a  $M \times N$  mixing matrix.

It’s possible to generalize ICA to *any* mixing situation...

In motif ICA, the sources are short sequence motifs (currently, but not necessarily, modeled as position-weight matrices [?]), 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 relevant 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 inference in a Bayesian framework, proposed recently by John Skilling [?]. 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.

Nested sampling is applied to an *ensemble* of states, which represent possible solutions to the problem at hand. The ensemble is initialized by sampling uniformly from the prior distribution, then sorting the states according to their likelihoods. In nested sampling, Each state in the ensemble is considered to be a representative of the set of states with similar likelihoods. If the likelihood of each state is drawn as a contour on the likelihood distribution, we see a nested set of contour lines, converging towards the peaks of the likelihood distribution. We therefore call the ordered set of states a nested ensemble. For each cycle of nested sampling, the least likely state in the ensemble is discarded, and a new state is chosen by sampling uniformly from the prior subject to the constraint that the likelihood of the new state must be greater than or equal to the likelihood of the discarded state.

In this context, the most exciting property 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 the 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.

### 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 [?]: 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 proportional 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.

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 parallel: 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.

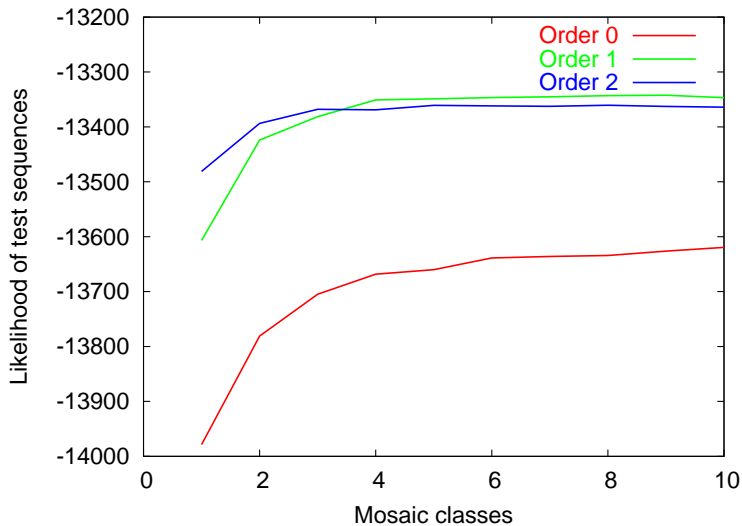


Figure 2: Comparison of mosaic backgrounds

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

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.

The NestedMICA source distribution can be downloaded from <http://www.sanger.ac.uk/Users/td2/>. This distribution includes all the required library code. 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/>.

As of NestedMICA 0.7.0, a Java 5 runtime environment is required. In general, we recommend that you use the latest available Java from Sun Microsystems. Mac OS X is a supported platform, but you'll need to run version 10.4.0 ("Tiger") or later, and separately download the Java 5 packages from Apple. You'll also need to reconfigure your system to make Java 5 the default version for command-line applications (note that Apple's Java Preferences tool doesn't do this, it only affects programs started via your web browser). The best way to do this is to add the following lines to the `.bashrc` file in your home directory:

```
export JAVA_HOME=/System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home
export PATH=${JAVA_HOME}/bin:$PATH
```

## 2.1 Compilation

The main body of the NestedMICA code is written in Java. To compile this, 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), `sunos`, and `icc` (Linux with the proprietary Intel ICC compiler). The 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`. Note that we've had reports of problems when building on Mac OS with old developer tools (before gcc version 3.1). If in doubt, please install the latest developer tools.

The main NestedMICA programs can be run using simple wrapper scripts which are included in the `bin` directory of the NestedMICA distribution. You should add this directory to your `PATH` environment variable, *e.g.*:

```
export PATH=$(pwd)/bin:$PATH
```

The wrapper scripts locate all the Java and C code they need relative to the place where the scripts are located, so if you need to install NestedMICA in a new location, copy the complete directory structure (minus the `src` directory if you don't want that) to the desired location, then set your `PATH` accordingly.

Note that the wrapper scripts will only work on Unix-like systems. If you need to use the code on an exotic platform, you'll need to write your own wrapper scripts or call the Java code directly.

## 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:

```
makemosaicbg -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` parameter is increased, mosaic models with arbitrary numbers of sequence types can be trained using this program.

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

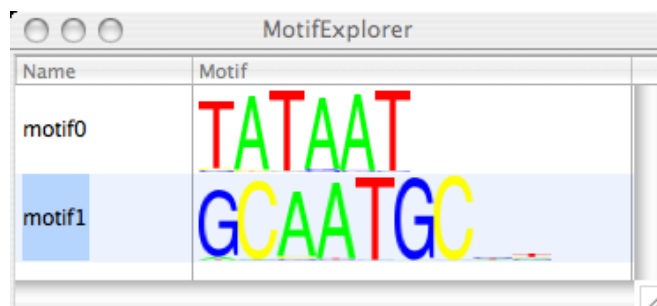
Once the background model is trained, you can start the motif finder using a command like:

```
motiffinder -seqs micatest100.fa -backgroundModel mt100.sbg
            -numMotifs 2 -ensembleSize 50
```

This runs the motif finding system until the nested sampling process is close to convergence, then writes the optimal set of motifs to a file named `motifs.xml`.

You can view your discovered motifs using the MotifExplorer tool, available from:

<http://www.sanger.ac.uk/Software/analysis/nmica/mxt.shtml>



Name	Motif
motif0	TATAAT
motif1	GCAATGC

Figure 3: Motifs discovered from the demonstration set.

One word of warning here: the `-ensembleSize 50` option is recommended here in order to get quick results on this simple test data set, but will lead to lower sensitivity than leaving `-ensembleSize` at its default value. For normal use, we do not recommend altering `-ensembleSize`.

## 2.3 Increasing the memory limits

Some Java implementations may, by default, limit NestedMICA's memory usage to an unreasonably small value. If NestedMICA is crashing with `OutOfMemoryError`, you can explicitly set a new memory limit by setting an environment variable. To set the memory limit to 500Mb, use:

```
export NMICA_JVMOPTS=-Xmx500M
```

As a rough rule-of-thumb, a good memory limit for large datasets is 1Mb of memory for every 1Kb of input sequence.

## 2.4 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 options 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`.

## 2.5 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-formatted database of sequences to analyse.
- `-backgroundModel` *filename* the background model to use, normally generated by running the `MakeMosaicBackground` program.
- `-outFile` *filename* name of a file to write the final motif set (default `motifs.xms`)
- `-sampleFile` *filename* name of a file to write periodic samples during the motif-finding process. If you run the finder with the option `-sampleFile foo`, it will write sample files `foo.1000.xms`, `foo.2000.xms`, etc.
- `-sampleInterval` *n* number of cycles between sample files (default=1000).
- `-maxCycles` *n* the maximum number of cycles to run the sampler. The program will then exit regardless of convergence status. Currently the criteria for deciding when the process has converged are quite conservative, so this option may be useful.
- `-ensembleSize` *n* the size of the nested ensemble used to explore the probability landscape. Larger values improve reproducibility at the expense of speed. Default behaviour is to set the `ensembleSize` to 200 for large problems ( $\geq 5$  motifs), but to use larger ensemble sizes when learning less motifs, up to a maximum of 1000 when learning a single motif. This should give maximum sensitivity under most circumstances. When learning one or two motifs from a very simple dataset, you may want to explicitly force a small `ensembleSize` for performance reasons.
- `-alphabet` (`dna|protein`) the alphabet of input sequences. Default is DNA.



`-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.

## 2.6 Advanced 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 re-sample a model component. Default is 0.2 which is probably good for most purposes.

## 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:

```
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.derholm.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 `-threads 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:

```
dlepnode -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 (~10 seconds) glitch after a worker dies during which no work will be done, so very rapid worker turnover can be inefficient. 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.

## 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.

## 2.12 Scanning with motifs

A simple program is supplied to scan a sequence using weight matrices learned by NestedMICA. Hits are output in GFF format. This program uses bits-sub-optimal scoring: this means that the best possible match to a given weight matrix will always score 0.0, while other sequences will receive negative scores.

Typical usage:

```
motifscanner -seqs seqfile.fa -motifs motifs.xml -scoreThreshold -5.0 -strand both
```