Investigating Power and Limitations of Ensemble Motif Finders using CE^3 Metapredictor*

Mauro Leoncini[1,2], Manuela Montangero[1,2], and *Karina Panucia Tillán[3]

[1]Dept. of Physics, Computer Science, and Mathematics, Univ. of Modena and Reggio Emilia, Italy. Email: name.surname@unimore.it
[2]IIT-CNR, Pisa, Italy
[3]Dept. of Science and Methods for Engineering, Univ. of Modena and Reggio Emilia, Italy. Email: karina.panuciatillan@unimore.it

Abstract

Ensemble methods represent a relatively new approach to motif discovery that combines the results returned by "third-party" finders with the aim of achieving a better accuracy than that obtained by the single tools. Besides the choice of the external finders, another crucial element for the success of an ensemble method is the particular strategy adopted to combine the finders' results, a.k.a. *learning function*.

Results appeared in the literature seem to suggest that ensemble methods can indeed provide noticeable improvements over the quality of the most popular tools available for motif discovery.

With the goal of better understanding potentials and limitations of ensemble methods, we developed a general software architecture whose major feature is the flexibility with respect the crucial aspects of ensemble methods mentioned above. The architecture provides facilities for the easy addition of virtually any third-party tool for motif discovery whose code is publicly available, and for the definition of new learning functions. We present a prototype implementation of our architecture, called CE^3 (*Customizable and Easily Extensible Ensemble*).

Using CE^3, and available ensemble methods, we performed experiments with three well-known datasets. The results presented here are varied. From the one hand, they confirm that ensemble methods cannot be just considered as the universal remedy to the difficulty of "in-silico" motif discovery. On the other hand, we found some encouraging regularities that may help to find a general set up for CE^3 (and other ensemble methods as well) able to guarantee substantial improvements over single finders in a systematic way.

Keywords: DNA binding, ensemble methods, motif discovery, software tool, transcription factor, XML.

1 Introduction

The discovery of Transcription Factor Binding Sites (TFBSs), *i.e.*, functional DNA sequences involved in gene expression, is an important and challenging problem in molecular biology. As the experimental protocols available for TFBS discovery are lengthy and costly, the problem has been tackled also from a computational perspective. Mathematical models of TFBSs have been proposed [1, 2], often termed *motifs*, and many algorithms designed and implemented in the last thirty years (see, *e.g.*, [3, 4, 5] and [6] for further references).

Despite such impressive efforts, the prediction accuracy remains low. A relatively recent assessment of thirteen popular algorithms performed by Tompa et al. [7] has made it clear that no single method performs well (*i.e.*, gives accurate results) on different datasets, and that it is by no means easy to characterize the inputs for which a method may give good performances.

*A preliminary version of this paper appeared in Proc. IWBBIO 2013, Granada (Spain), 18-20 Mar. 2013

In relatively recent times, a new approach has been pursued with the aim of overcoming the limitations of existing motif discovery algorithms (here also termed *finders*). This is based on the idea that accurately combining the results returned by different finders can lead to better TFBSs predictions than using each finder alone. The tools following this paradigm are known as *ensemble methods* (hereafter simply *ensembles*) [8, 9, 10, 11], or also meta-predictors [12].

A popular reasoning that supports the design of ensembles is a more or less sophisticated *voting argument*. The idea is that the likelihood of a DNA stretch being a functional site grows with the number of different motif discovery algorithms that report that stretch among their findings. The actual procedures adopted to "combine" the finders' results, often referred to as the *learning functions*, may vary a lot across different ensembles. Together with the choice of the used finders (typically third-party, external software tools), the learning functions is the feature that mostly affects the performance of an ensemble.

All the above cited studies that propose ensemble methods also report the results of a number of experiments performed of benchmark data. Indeed, the results seem to support the idea that, even putting together low performance finders, the overall accuracy of an ensemble can be cast to an acceptable level, well above those of the single finders.

We are much more cautious about the power of ensembles. First of all, the observed performance of an ensemble cannot but strictly depend on the finders it is based on, with different "blends" likely leading to very different results. If all the component finders fail heavily on a given dataset, it is conceivably hard for the ensemble to come up with acceptable results. Even assuming that some finders give good results will not guarantee that the poor ones do not prevail in the ensemble's eventual answers. Also, though the possibility of deciding the configuration of the ensemble (which is granted by some, but not all, the available meta-predictors) seems a potential point of strength to exploit, no general rules have been suggested yet that can be turned into easy set up guidelines for the end-users. A final observation is that, to the best of our knowledge, no existing ensemble considers the possibility of easily extending the set of available finders; this prevents the adoption of any new powerful tool that might be possibly proposed in the literature.

In order to validate our beliefs and to understand more in depth the power and limitations of ensembles, we first defined and implemented a general ensemble architecture, which we called CE^3. CE^3 is *customizable* and *extensible* to a previously unpaired extent, allowing for easy addition of external finders and the definition of new learning functions. Our opinion is that, since a positive correlation must exist among the accuracy of the finders used by an ensemble and that of the ensemble itself, a clever design should allow for the latter to include any accurate tool that will be possibly available in the future. At the time of writing, the inclusion of a new finder in CE^3, though not completely automatic, is already quite an easy task, which do not require any programming skill.

We performed a large number of experiments using CE^3 as well as other available meta-predictors and standalone finders. We found striking evidences that a naive use of ensembles do not help to improve the quality of the results returned by the most accurate standalone finders. Hard datasets for single tools (such as the one proposed by Tompa and coauthors [7]) remain hard for ensembles as well, even though some small improvements may be observed. On the positive side, we propose a strategy for finding a subset of finders, among the ones available, which depends on the dataset under consideration and that often drive CE^3 to solutions that are very close to the best achievable ones.

The rest of this paper is organized as follows. In Section 2 we give a description of general ensembles. In Section 3 we describe the current CE^3 implementation, with particular emphasis on the tool's capability of being extensible. In Section 4 we present and analyze the results obtained in a number of experiments performed with various configurations of CE^3; finally, in Section 5 we offer some concluding remarks and discuss future work scheduled on CE^3 project.

2 Ensembles General Architecture

Ensembles used for the *Motif Discovery Problem* (*MDP*) orchestrate the execution of different *de-novo* motif finders. Each finder returns a set of motifs that potentially describe biologically active sites. These are analyzed by the ensemble with the aim of increasing the accuracy of predictions. The general structure of such systems is made of the four main components listed below and illustrated in Figure 1.

1. *External algorithms integration module*: ensembles integrate possibly many different (third-party) *de-novo* finders.

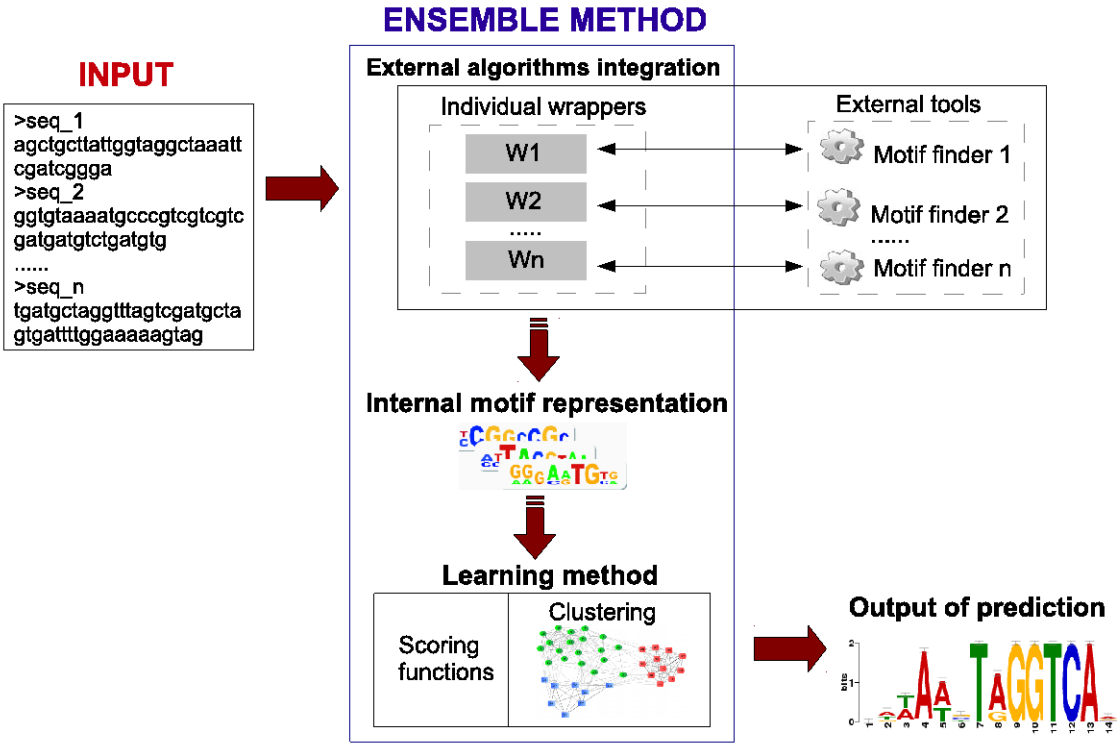

Figure 1: General architecture of an ensemble.

2. *Internal motif representation*: motifs returned by the finders are represented uniformly using appropriate data structures and internal motif handling software.

3. *Learning rule*: one or more learning techniques are used to discover the most promising motifs among all those predicted by single finders. This is the component that mostly characterizes (and distinguishes) today's available ensembles.

4. *Output module*: the prediction made by the ensemble is returned to the user in one of the commonly adopted "external" motif representations (*e.g.*, weight matrices and text logos), possibly with the explicit sites list.

Basically, ensemble methods interface each external tool through a specific wrapper (Figure 1). The motifs predicted by the external tools are then converted to a unique internal representation defined by the ensemble. The motifs in this new representation are then analyzed by the learning method(s) with the aim to discover the most promising results to later report as the output of the ensemble.

It easy to see that the crucial components in the design of ensembles are the included third-party methods and the adopted learning function. On the one hand, the third-party tools determine the set of motifs that will be predicted by the ensemble. So, making a proper selection of the finders to be used is a point of strength to exploit, since a careful combinations of different algorithms may provide substantial improvements in motif predictions. However, ensembles implemented so far are characterized by a fixed set of finders. Some of them (e.g. MotifVoter [11]) allow the user to select the particular finders s/he wishes to use in a particular run, however these are again taken from a predefined and fixed set. While extensions are possible, they are nonetheless difficult to implement by the end user. In fact, if one wishes to extend an existing ensemble by adding a new motif finding algorithm (say one that adopts a new powerful search strategy), s/he has to do some non trivial programming work (assuming the source code is available). At the very least, one has to write code to interact with the new method, *i.e.*, to wrap its execution and parse the returned results. This clearly needs knowledge of the ensemble internals and some programming skills.

On the other hand, the choice of the learning function influences the output quality, affecting the prediction of relevant motifs. Thus, it has to be carefully designed to be able to discover relevant information among all motifs returned by the selected finders. Ensembles implemented so far are characterized by a specific learning function devised by the tool's authors and hence changes are difficult to implement. Here, even greater efforts are requested if one wishes to add a completely new learning method to the ensemble.

3 CE^3 Architecture

Our customizable and extensible ensemble method, called CE^3, is obtained by a conceptually simple modification of the generic ensemble's structure, as shown in Figure 2. The changes are focused on the two key features of a generic ensemble, consisting in the introduction of two modules (called wrappers) that allow the inclusion of new motif finding tools and new learning functions by using quite simple procedures (we are not taking into consideration here the task of implementing the finding algorithms and learning functions, but only their inclusion in the ensemble). CE^3 is written in Python and is available for dowload at `algogroup.unimo.it/software/CE3`, while a Web interface is still work in progress.

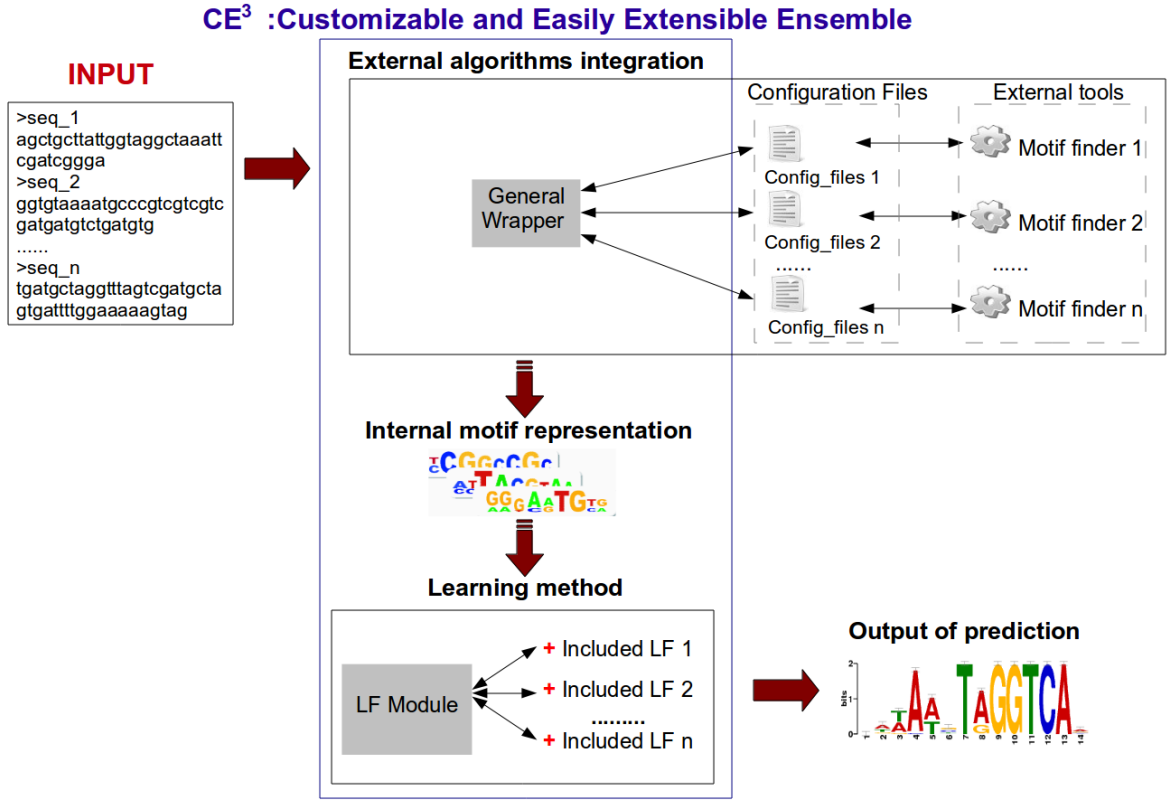

Figure 2: CE^3 architecture. External algorithms are integrated thanks to a general wrapper and two XML configuration files. Several (new) learning functions (LF) may be included in the ensemble by means of the Learning Function Module.

In the following of this section we will first describe the actual CE^3 configuration, then we will explain how a user might extend CE^3 with the addition of new finders and/or new learning functions.

3.1 CE^3 Standard Configuration

CE^3 comes with a standard configuration that allow users to directly run the ensemble "as it is". In the following, we describe CE^3 general ensemble architecture referring to the four main components depicted in Section 2.

3.1.1 External Finding Algorithms

CE^3 includes eleven motif finders, namely: Aglam[13], AlignAce [14], BioProspector[15], MotifSampler, [4], MEME [3],RSAT (`oligo-analysis` and `pattern-assembly`)[16], MDscan [17], Weeder[5], Improbizer[18], SPACE[19] and SeSiMCMC[20]. CE^3 allows the user to run any subset of the included finders.

3.1.2 Internal Motif Representation

The execution of the finders provides a set of putative motifs which CE^3 internally "rebuilds" in a way that makes it easy to perform a number of useful operations. In the current CE^3 implementation, we use a refined version of the TAMO MOTIFTOOLS package [21], that provides functions for motif creation (using either sets of sequences or matrices) as well as methods for scanning sequences using PWMs. The TAMO framework is implemented in Python, which makes the integration with CE^3 very easy.

3.1.3 CE^3 Learning Functions

At present, CE^3 implements two new learning functions: a voting method (originally inspired by the MotifVoter learning method) that is CE^3 default choice, and a customizable clustering method. Moreover, CE^3 contains a re-implementation of the MotifVoter learning function. To date, learning functions implemented in other existing ensembles are not included. This is mainly due to the fact that source code is not always available and/or full details are not always given in papers, making re-implementation a hard task and one with the outcomes likely to be criticized. Even when source code is available, it is not always trivial to isolate the learning method code inside the tool code.

The input to the learning functions is a set of motifs (given in CE^3 internal representation) obtained as the union of the motifs found by running each finder separately on input the set of sequences.

Voting Method Under the voting method, we rank the highest as putative binding sites those DNA stretches that have been predicted by the largest number of different tools. The method consists of two main steps:

- *Site Voting Procedure:* Each motif predicted by a finder defines a set of "matches" (*i.e.*, potential binding sites) in the input sequences. We say that any two sites overlap if their starting positions are shifted by at most 4 bps. Initially, each site gets at least one vote (by itself), and the vote count is incremented by the total number of sites belonging to different motifs that overlap it. For each sequence, we only retain the first k most voted sites, where k is the maximum number of putative TFBS per sequence requested by the user.

- *Nucleotide Voting Procedure.* For a given site X resulting from the Site Voting step described above, let support(X) be the set of motif matches that contributed to X's vote counter; also, let span(X) refer to the sequence of bps covered by X's support (*i.e.*, the union, position-by-position, of the base pairs in support(X)). To each nucleotide b in span(X), the Nucleotide Voting procedure assign a value that equals the number of sites in support(X) that include b. If M is the maximum value assigned by this procedure, then the final putative TFBS returned, corresponding to the original site X, is the sequence of consecutive bps in span(X) with value greater than or equal to $\lfloor M/2 \rfloor + 1$. An example is given in Figures 3.

Customizable Clustering Method This learning method processes the motifs predicted by motif finders based on the idea of motifs clustering, placing motifs (possibly predicted by different finders) that are similar enough in the same cluster, and considering each cluster as a motif ensemble prediction. We allow the users to choose among several clustering options, making this method highly customizable. In its basic implementation, CE^3 selects the motif(s) to be returned according to the following procedure in which, at each step, a few options are available to the user:

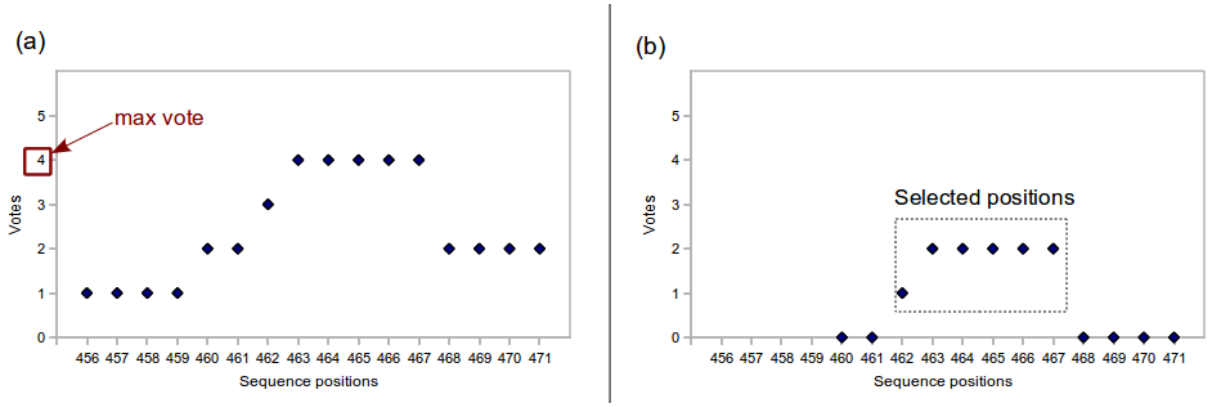

Figure 3: Nucleotide Voting procedure for a site X with span 456-471 and support including at least 4 sites. (a) Number of votes received by nucleotides from position 456 to position 471. The maximum number of votes received by nucleotides is $M = 4$. (b) Nucleotide votes scaled by $\lfloor M/2 \rfloor = 2$ (only non negative votes are shown). The rectangle highlights the putative TFBS returned.

1. Compute the similarities of each pair of motifs. Three methods are currently available:
   (a) PWM similarity implemented using RSAT's utility COMPARE-MATRICES [16];
   (b) Site overlapping, computed as follows: given motifs M_1 and M_2, and input sequence set $\mathcal{S}$, let $N_1(\mathcal{S})$ and $N_2(\mathcal{S})$ denote the sets of nucleotides in $\mathcal{S}$ predicted by M_1 and M_2, respectively. Similarity is then defined as
   $$I_{\mathcal{S}}(M_1, M_2) = \frac{|N_1(\mathcal{S}) \cap N_2(\mathcal{S})|}{\min\{|N_1(\mathcal{S})|, |N_2(\mathcal{S})|\}}$$
   Note that $0 \leq I_{\mathcal{S}}(M_1, M_2) \leq 1$ and that $I_{\mathcal{S}}(M_1, M_2) = 1$ if $M_1 \subseteq M_2$ or $M_2 \subseteq M_1$; *i.e.*, M_1 and M_2 are highly similar when the sites of one include those of the others.
   (c) Use our re-implementation[5] of a more recent PWM fuzzy integral similarity measure called FiSIM, introduced in [22].
2. Compute motif clusters using the chosen similarity measure. Here again two options (*i.e.*, clustering methods) are available:
   (a) single-linkage clustering, where at each step the closest clusters are merged, starting from singletons and stopping when the maximum among the desired number of clusters and the number of connected components is reached;
   (b) single-linkage followed by a refinement based on the detection of a dense core within each cluster.
3. Discard the clusters that do not include motifs determined by at least two different finders, and rank the remaining clusters according to one of the following measures:
   (a) average pairwise similarity of cluster members,
   (b) number of contributing finders to motifs in cluster,
   (c) cluster cardinality.
4. For each cluster, determine the representative motif and the set of binding sites to be given as output. The user has the possibility to select the representative motif as:

[5]Original code was apparently not available at the time of our CE^3 implementation.

(a) the one having largest intersection with the other motifs in the cluster;

(b) the one exhibiting the maximum similarity to all the other motifs in the cluster.

The set of output sites is given by the set of binding sites of the representative motif that (partially) overlap a site of a motif in the same cluster predicted by a different tool (with respect to the one predicting the representative motif).

MotiveVoter Learning Function We implemented MotifVoter learning function as described in details in [11].

3.1.4 Output Module

CE^3 allows the user to choose the number of top scoring motifs to be returned. Given this number, the ensemble output is a set of putative motifs, given as a set of binding sites or a set of PWMs (optionally supplemented with a text logos). CE^3 also provide optional output information (*e.g.*, statistics).

3.2 CE^3 Extensibility

The nice fact about CE^3 is that, if the user does not like the choices we made in the standard configuration, s/he can configure its own version of the ensemble. The user can intervene in the first and third components of the ensemble, adding new finders and new learning functions in a semiautomatic way, by correctly filling in predefined XML configuration file templates. Extensibility makes CE^3 some kind of *meta*-ensemble that (properly configured) is able to simulate the behavior of any existing ensemble. Moreover, the user is able to implement variations to existing ensemble or create her/is own, exploiting her/is ideas. Last but not the least, CE^3 will never become obsolete because of the design of new (better performing) finders and/or learning functions, as these can be included in CE^3 and keep it up to date.

In the rest of this section we will explain architectural extensibility design and the procedures to extend CE^3.

3.2.1 Motif Finder Inclusion

CE^3 interacts with new external motif finders by means of a general wrapper, in order to reduce the implementation of a dedicated interface to the completion of two XML configuration files. In this way, the addition of the new algorithm is a much easier semi-automatic process guided by the system which does not require programming abilities.

This wrapper specializes to each external tool using the proper description provided in two predefined XML configuration files, made to capture the necessary information to properly run such algorithms and to read their output. A finders may be included if it satisfies the following (quite common) constraints:

1. The tool input must be a set of sequences stored as a FASTA file.
2. The tool must run as a command line utility under Unix/Linux operating systems.
3. The tool must produce the output (also) in textual form (to a file or standard output), with well identifiable "blocks" describing the motifs.
4. The motifs must be described, in the output text, either as sets of sequences (the putative binding sites) or as *Position Weight/Frequency Matrices* (often referred to as *PWMs*).

In general, the design of any wrapper interacting with external algorithms must include code for the sequential execution of two key functions: (i) *Tool running*. The external command line tool must be run with the appropriate parameters using its expected syntax. (ii) *Output parsing*: the tool's output must be intercepted and parsed in order to extract the information required and to build the internal motif representation. CE^3 implements these two functions by means of two XML configuration files templates instantiated by the user and interpreted by the wrapper. An example is give in Figure 4.

(i) Tool Running Each motif finder developed so far defines its own set of parameters and command line syntax. In order to masquerade these diversities, we prepared a template *XML configuration file*, instances of which describe the "syntax" required by specific finders. The XML files will then be read and interpreted by the wrapper in order to synthesize syntactically correct command lines and launch the finders' execution via standard operating system calls.

The tags defined in the XML configuration file correspond to the fundamental parameters that one can find in virtually all the available finders. While an accurate choice of such parameters may greatly help in driving the finder to discover biologically active sites, it is nonetheless true that many users perceive the task of parameter setting as annoying (actually, they have been sometimes referred to as *nuisance parameters* [23]). Indeed, one of the possible advantages of using ensembles consists precisely in changing the way a better accuracy of prediction can be possibly achieved: from the "fine-grained" tuning of parameters as a function of the particular dataset, to a clever, but fixed learning function that combines the results of many finders run on an essentially default set up.

The *core* parameters provided in our XML configuration files are listed below.

input file (in FASTA format), which is clearly mandatory;

motif length (or width) given either as a single value or as a pair of values (an interval);

background information, which can be given as the name of a supported organism, as a set of nucleotide frequencies, or as a set of probe sequences;

topmotifs, namely the maximum number of motifs to ultimately report to the user, taken among the ones with highest score;

strand to search for possible sites (positive, negative or both).

Each parameter listed above corresponds to a easily detectable tag in the XML file listed below. The only task left to the final user is to associate the appropriate values to the right tags and attributes (such information should be easily retrieved in the tool's READ ME files).

XML configuration file template:

```
<tool executable_name="" change_dir="False">
   <fasta param=""/>
   <width param="" range="0" possible_values="1"  values=""  />
   <background param="" gen_file="0" exec_command="" value="" function=""/>
   <topmotifs param="" />
   <seed param="" value="" />
   <genome param="" value="" />
   <strand param="" value="" />
   <extra value="" />
   <order value="" />
</tool>
```

Besides the five tags whose meaning has already been described, the file contains: the `seed` tag to provide *seeds* to probabilistic methods (in order to guarantee the possibility of repeating experiments); the `genome` tag to specify the genome from which the input sequences came from, if needed; the `extra` tag gives room for (optional) tool specific parameters, which will be given as a single unstructured string to be inserted "as is" in the command line; the `order` tag is used when the tool command line requires that the input parameters must be provided using a specific order. The attribute `change_dir` in the `tool` tag tells the ensemble to move (by means of the unix/linux `cd` command) to the tool's directory before execution.

(ii) Output Parsing The output produced by different motif finder tools are clearly different from one another: information provided, adopted format, error and log messages reported, etc., are usually all given according to a format which is specific for the tool under consideration. Moreover, some tools write results to the standard output, while others write to a file. In spite of these differences, it is usually easy to detect in the output listing a block of information describing the candidate motifs. Such blocks of text are well delimited by easily detectable syntactic markers (since the output is for a human to read). Basing on this feature, we use patterns and regular expression to make it possible to locate the motif descriptions in the output results.

A predefined *XML parse file* (listed below) stores the information needed in the parsing phase. In order to fill in one such file, the user must provide *static* information rather than working code in a suitable programming language.

XML parse file template:

```
<parse>
   <motif_head> </motif_head>
   <binding type="" space="" letters="" order="" start=""></binding>
   <motif_end> </motif_end>
   <output_file name="" extension="" dir=""> </output_file>
   <score type="" key_score="None"> </score>
</parse>
```

The `motif_head` (resp., `motif_end`) tag stores the leading (resp., trailing) signature of the text block describing one predicted motif. The `binding` tag captures the model used to represent the motif in the external tool output (PWMs or list of binding sites). The `output_file` is used when the output is printed to file. The `score` tag stores the score of a specific motif.

Adding a New Finder Algorithm

The process of adding a new finder is now a relatively easy process, in which CE^3 gathers information from the user and performs the following actions:

1. Include the path to the home directory of the new tool in the CE^3 path file.
2. Create the XML configuration file: identify the general parameters for the specific tool and associate the proper parameters to the keywords defined in the XML configuration file.
3. Create the XML parse file: first produce (or otherwise gather) a sample output of the new tool. Check the format with which the motifs are presented to the user. Isolate one such motif description and the "signatures" that surround it, also detecting the variable parts. Use these information to fill in the predefined tags of the XML parse file and to define (through a guided procedure) the regular expressions needed in it.

An example is given in Figure 4, where we show how to include the finding tool Bioprospector to CE^3 by means of the previous procedure and described XML files. The upper part of the figure shows the usage of the tool from command line: parameters needed to fill the XML configuration file are highlighted and explained. Just below the instantiated XML configured template file. The lower part of the figure shows a sample from Bioprospector output listing: the outer (brown) rectangle highlights a text block describing one motif, with all the relevant information needed to internally rebuild the motif itself. Just below, the instantiated XML parse file. Tags have been filled using the proper information identified in the output listing. In particular: the `motif_head` and `motif_end` tags are associated to the delimiters identified (a blank line delimiter is represented by the *null* tag value); the `type` attribute of the binding tag tells that the motif is given as a matrix. The remaining attributes allow the correct matrix scanning: `space` is the number of lines between the motif head and the beginning of the matrix; `letters` orient the sense in which the matrix is supplied (each line represents one position of the motif); `order` gives the order in which letters `a, c, g, t` are provided; `start` gives the offset of the first column of the matrix from line beginning. Finally, Patterns `'I', 'F'` and `'L'` represent regular expressions defined in CE^3 to facilitate identification of variable values in the text block: `'I'` is used for integer values, `'F'` for float values and `'L'` for alphanumeric characters.

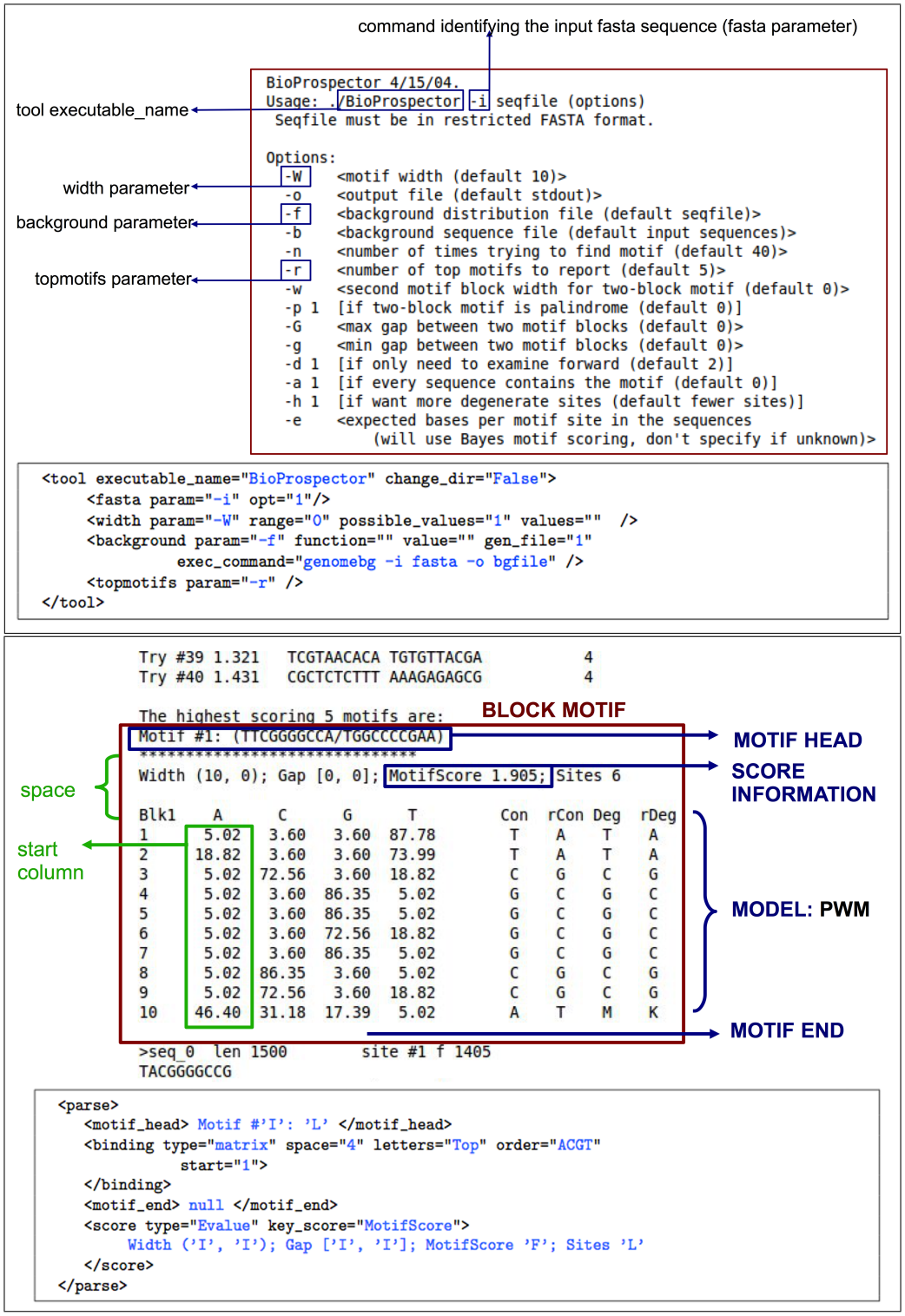

Figure 4: Example: Adding Bioprospector Tool in CE^3. Top: Bioprospector usage (command line) and corresponding XML configuration file. Bottom: Bioprospector output listing (sample) and corresponding XML parse file. The user contribution is highlighted in blue.

3.2.2 Module for Learning Function Inclusion

The extensibility of the ensemble concerns also the inclusion of new learning functions. CE^3 contains a *learning function wrapper* that handles the execution of learning functions and the addition of new ones. This module receives as input the set of motifs predicted by the motif finders in the previous phase and runs one learning function among those included in the ensemble (at the user choice).

The module assumes that each function included in the ensemble is stored in a dedicated directory (named after the function) that contains:

- the function code; *i.e.*, all files implementing the function written in any programming language. Obviously, implementing the learning function might be a demanding task and clearly requires some programming skills, but it is nevertheless mandatory if one wishes to add a completely new learning function. What we are discussing here is the fact that, once the implementation is done, the inclusion in CE^3 is quite an easy task. Observe, however, that CE^3 does not force the use of a specific programming language to implement a (new) learning function.

- a *Learning Function configuration XML file* describing the function and its parameters:

```
<learning_function name="">
<description> </description>
<parameter name="" values="" id=""/>
</learning_function>
```

  The tag `learning_function` contains the name given to the learning function, while the tag `description` stores a textual description. The tag `parameter` is used to describe parameters needed by the learning function (for each parameter, one such tag should be added): the attribute `name` records the identification name of the parameter; `values` stores the default parameter value; `id` is the parameter name that will be used by the learning function.

- a *Python interface* with a predefined class and methods used to run the function:

```
class execLF:
     def __init__(self):
             self._motifs = []
             self._seqs = []
             self._params = {}

     def exec_(self):
            # write invocation command
```

  The Python interface template defines the class `execLF` containing three basic parameters: the list of motifs (using the TAMO representation), the list of sequences given as input to the ensemble, and the list of parameters needed to invoke the learning function (these parameters will be derived by the module from the XML configuration file). Function `exec_()` is used to invoke the learning function.

Adding a New Learning Function The process of adding a new learning function, whose code is available (*e.g.*, the source code or a compiled library) is now a relatively easy 4-step task.

1. Create a new folder, named after the new function, in a specific directory;

2. store all files corresponding to the new function in the new folder;

3. create an instance (and store it in the new folder) of the Python interface template: write in function `exec_()` the command line that invokes the new learning function;

4. create an instance (and store it in the new folder) of the XML Configuration file by specifying the name of the learning function, its (textual) description and possibly define the extra parameters used for invocation.

4 Experiments

We performed a number of experiments using three different datasets. As already pointed out in the Introduction, we had several goals in mind: first we tested CE^3 against single motif finders, in order to understand if the very idea of ensembles is worth being pursued; second, we tested CE^3 against other (available) ensembles, looking for clear evidence that extensibility can indeed be exploited to achieve more accurate results; third, we wished to possibly find a standard set up for CE^3 that would guarantee sufficiently accurate results across different datasets. The results described in this section give strong enough indications to meet the first two objectives. On the other hand the third goal seems much more difficult to achieve, as expected, although some general indications have emerged which made it possible to define a first promising strategy for the CE^3 configurations.

4.1 Experimental Settings

We run CE^3 under many different configurations, by varying both the set (number and composition) of the underlying finders and the various learning function options. We run the single finders that are included in CE^3 using default parameters and without performing any pre- or post-processing. This is clearly different from what was allowed in the Tompa et al.'s assessment, which explains why some results are not consistent with those reported in [7].

We run ensembles for which code was available; in particular, we tested SCOPE [8] and MotifVoter [11]. We could not retrieve two single motif finders that were originally included in MotifVoter, namely MITRA [24] and AnneSpec [25], and this might be another reason why our results show some difference with those originally published in [11]. On the other hand, by using our implementation of MotifVoter's learning function, we were able to compare CE^3 and MotifVoter using also some finders not originally included in the latter, namely SeSiMCMC, RSAT and Aglam.

We compared the results obtained using all the main statistics commonly used at nucleotide level, namely: *Correlation Coefficient* (*nCC*), *Sensitivity* (*nSn*), *Positive Predicted Values* (*nPPV*), *Performance Coefficient* (*nPC*), computed as in see [7]. All statistics range in the interval $[0,1]$: values close to zero are negative predictions, the closer to one the better.

All experiments were performed on a Dual Core Pentium with 2.7GHz CPUs and 2GB RAM, running Ubuntu/Linux OS.

4.2 Datasets

E. Coli Dataset This dataset has been created starting from 68 Escherichia Coli regulatory proteins [26]. Data contain 34 files for each transcription factor and 390 verified bindings sites altogether. Each file includes at least 4 and at most 49 binding sites, with an average of 11 binding sites per transcription factor. The binding site length range is between 10 and 48 bps with a mean of 22 bps. Sequence lengths varies from 210 to 248 with a mean of 222 bps. Each sequence of the dataset includes one instance of the corresponding motif.

TOMPA Dataset Tompa's dataset files is the well known benchmark used in the assessment of the de-novo motif finders [7]. It is considered a very hard benchmark, mainly because of the irregularities exhibited by the binding sites of a same transcription factor, *e.g.*, site lengths ranging from 5 bps to 27 bps. The benchmark is obtained from the TRANSFAC database, where the final data come from a selection process described in [7]. The dataset contains 52 files with sequences from 4 different organisms: 6 from fly, 8 from yeast, 12 from mouse and 26 from human. Four additional sequences are included with no planted binding sites. The number of sequences per dataset range from 1 to 35 with an average of 7 sequences; sequence lengths vary from 500 bps to 3000 bps.

G7 Dataset The overall G7 dataset [27] is composed by 7 real datasets containing sequences from distinct organisms and binding sites for the following transcription factors: CREB, MEF2, MYOD, SRF, TBP, ERE, CRP and E2F family. The number of sequences per dataset varies from 17 to 95. The sequence length is 150 bps for the CRP dataset and 200 bps for the others. Motif widths range from 8 to 22 bps with an average of 10.38 bps. Data include 263 binding sites with an average of 32 binding sites per dataset.

4.3 Results and Discussion

Figures 5, 6, 7, report statistics relative to the following experiments:

- predictions made by single motif finders running in their default configurations;
- predictions made by SCOPE in its default configuration;
- predictions made by CE^3 and MotifVoter in their respective "best nCC performing" configurations[13];
- predictions made by CE^3 and MotifVoter using all eleven finders (in the case of MotifVoter, we use our re-implementation).

We could not directly compare SCOPE with the other two ensembles by varying the number of finders, because the former does not allow to select the tools to use. Simple finders and ensembles were asked to return only the top scoring motif. We also performed the same experiments asking for the five top scoring motifs. Results are not reported here because the overall picture that emerged was essentially the same.

From the analysis of the results we observe that ensembles in general obtain better accuracies on the combined statistics (nCC and nPC). On only one dataset we observed a better behavior of a single finder (namely, Improbizier on the G7 dataset). In fact, as expected, a single good result cannot but be degraded when combined with other results of lower quality. But the point is that no easily computable criterion is known that guarantee the accuracy of motif finders a priori, and indeed, essentially all the tools available to date return unsatisfactory results in the majority of cases.

As for the ensembles, CE^3 shows a moderate to important gain in performances over the competitors.

We are aware that the results presented are far from being conclusive. On different datasets the same collection of finders (both single and ensemble) might produce results leading to radically different judgments. However, the figures we have provided at least suggest that the ensembles have the capability of granting sensible improvements in the state of the art of in-silico motif prediction.

Assuming this, a major problem is left open, which has to do with the correct set up of the ensemble for it to produce at least "close to best possible" results. We stress that Figures 5-7 report the most accurate results among those obtained by running CE^3 (and MotifVoter as well) with possibly very different configurations; and clearly, when the answer is not known *a priori*, it is impossible to tell which solution is the best. To make things even more complicated, we observed large gaps between the best and the worst results obtained in any given dataset (see below in this Section).

Single finders share the problem of parameter setting, of course, which is clearly magnified on ensembles, due to the possible choices of component finders and learning function. As already pointed out in Section 4.1, we decided in the first place to run the single motif finders under default configuration (and to rule out pre- and post-processing steps), thus restricting the set up problem to the characterizing parameters of ensembles. The "instabilities" of the observed results are thus only due to variations in the latter parameters, and must be mitigated to make ensembles (and CE^3 in particular) competitive against single finders.

We now briefly summarize some facts that emerged from the experiments described above and, as the result of an in-depth analysis of these facts, in section 4.3.1 we will propose a strategy that might be useful, under certain circumstances, to help the user select a good configuration for a particular dataset.

[13] The best performing configuration has been determined *a posteriori*, running the ensembles in all possibile configurations and selecting the one resulting in the best nCC prediction

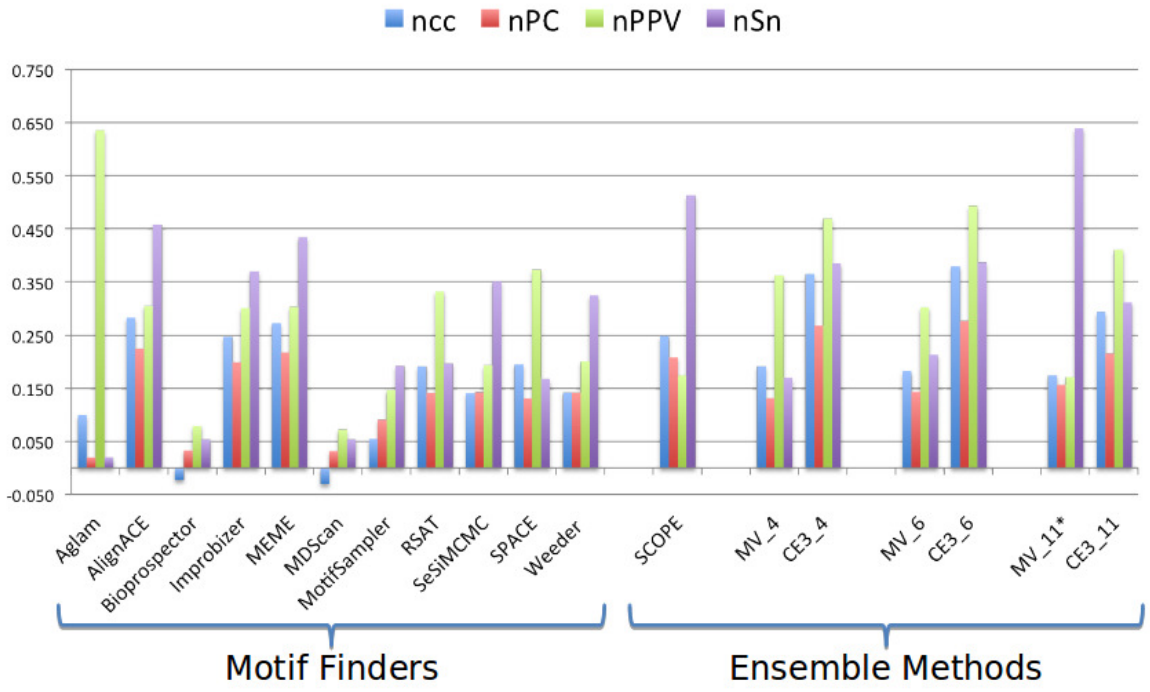

Figure 5: Experimental values of the statistics listed in Section 4.1 for the E.Coli dataset. The best nCC performing tool configuration for MotifVoter (denoted with MV_4) is given by: MEME, MDScan, MotifSampler and Weeder. CE^3_4 is the best nCC performing four tool performing. The best nCC performing tool configuration for CE^3 (denoted with CE^3_6) is given by: MEME, Alignace, SPACE, RSAT, Motif Sampler, and Weeder. MV_6 is the best nCC performing six tool configuration. MV_11* is our implementation of MotifVoter run with all eleven tools.

- No single blend of motif finders has been able to drive CE^3 to acceptable quality results in all the considered datasets, although some finders, among the most acknowledge ones in the literature, very often participate in the best performing runs (notably, MEME and Weeder).
- The above observation in turn implies that ensembles adopting a "fixed" set of external finders are likely to fail in the majority of datasets, unless very accurate fine tuning can be done at the level of the component finders themselves.
- Running an ensemble with a large set of finders is almost never a winning strategy. Devising three to six good component finders seems the right way to go. Figure 8 is representative of CE^3 behavior when the number of tools used in the prediction varies from two to eleven. Statistics are taken from the best performing solution (meaning tool combination) varying the number of executed of tools. Performance values tend to increase up to 6 (or 7) tools and then start decreasing. The already presented results (Figures 5-7) relative to ensembles running with all the eleven available tools strengthens this hypothesis.
- The voting learning strategy performed better than the clustering strategy in the large majority of tests.
- Apparently, no single property or group of properties of a given dataset (*e.g.*, site abundance) seems able to characterize the quality of the results produced by a given configuration, not even properties that can be hardly known in advance to a de-novo motif discovery software (say, average site length or degree of conservation).
- However, when considering similar datasets (*i.e.*, data coming from close-by organisms; genes with similar functionalities, possibly involving the same set of transcription factors) we observed that the best performing configurations were almost identical or varied a little.

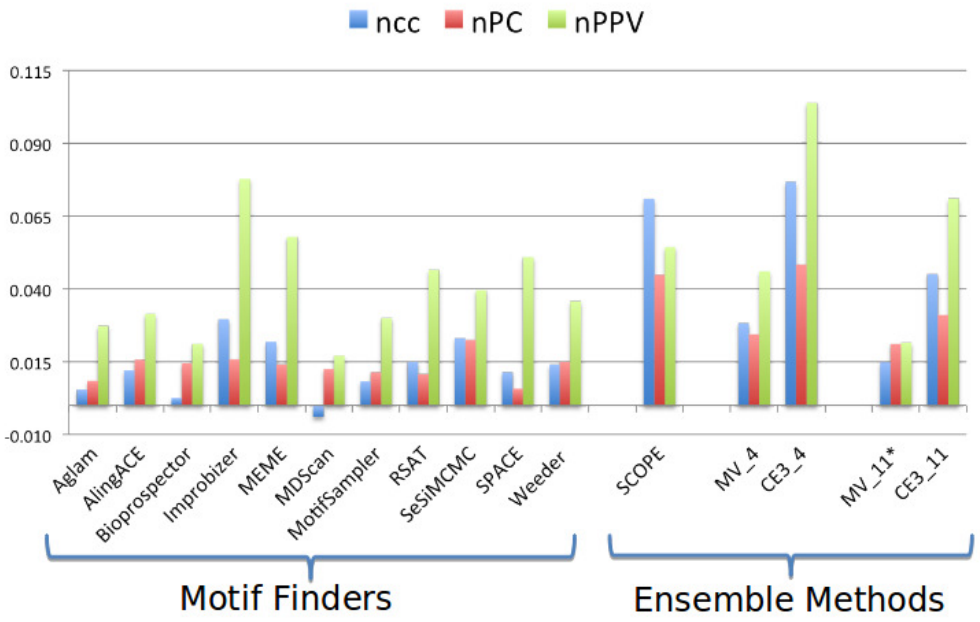

Figure 6: Experimental values of the statistics listed in Section 4.1 for the Tompa dataset. Sensitivity (not reported for graph legibility) is smaller than 0.052 for all tools, with the exception of SCOPE (0.203) and MV_11* (0.534). The best nCC performing tool configuration for MotifVoter (denoted with MV_4) is given by three configurations: MEME, Alignace, MotifSampler and one of the following: MDscan, SPACE or Weeder. The best nCC performing tool configuration for CE^3 (denoted with CE^3_4) is given by: MEME, Improbizer, RSAT and Weeder. MV_11* is our implementation of MotifVoter run with all eleven tools.

- Having more than one tool with similar optimization criteria (*e.g.*, Gibbs sampling) leads to very bad results more often than not. Instead, the voting criterion seems more suitable to a blend of finders representative of all the few good algorithmic strategies developed so far, which is another reason that supports the development of extensible ensembles.

4.3.1 CE^3 configuration

In this section we propose guidelines that, when applicable, can be used to establish a set of tools that is likely to leverage CE^3's performance close to the best possible output (given the dataset and the overall set of external finders available). As for the learning function, we always adopt the voting learning strategy, as suggested by one observation above.

As pointed out in the previous section, we observed a certain degree of stability in the configurations that guaranteed good results across similar datasets, in particular those whose sequences came from the same organism and/or shared a small set of transcription factors. Hence, as customary in learning-by-examples approaches, given a new dataset (called *test set*), we collect information coming from biologically similar datasets (called *training set*), to determine a stable CE^3 configuration to be used to make predictions on all files (collections of sequences) forming the test set.

More precisely, using the training set as input, we run CE^3 by varying the number of tools from 4 to 6 (see again one of the observations in the previous section) in all the possible ways. We compute the corresponding nCC values and use the one giving the better score(s) to make predictions for the test set.

We evaluated this strategy on two datasets showing some degree of homogeneity: the E. Coli, which includes sequences coming from the same organism, and the G7 dataset, which contains sequences sharing transcription factors

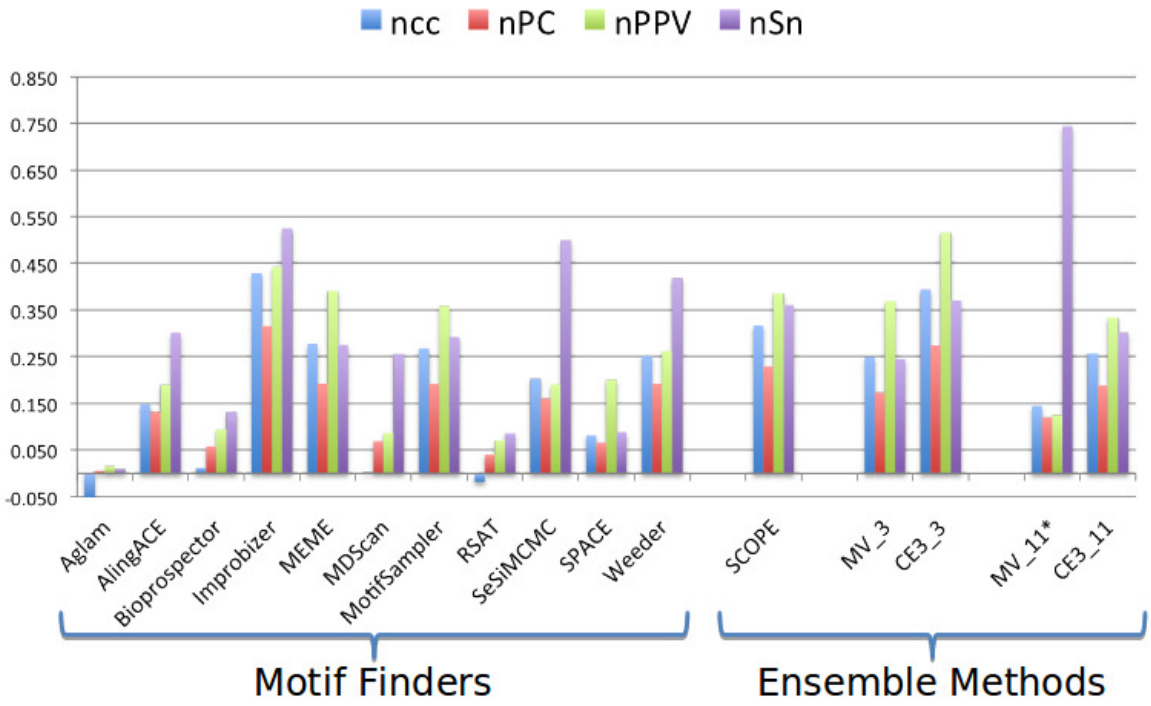

Figure 7: Experimental values of the statistics listed in Section 4.1 for the G7 dataset. The best nCC performing tool configuration for MotifVoter (denoted with MV_3) is given by three configurations: MEME, SPACE and Weeder. The best nCC performing tool configuration for CE^3 (denoted with CE^3_3) is given by: SeSiMCMC, SPACE and Weeder. MV_11* is our implementation of MotifVoter run with all eleven tools.

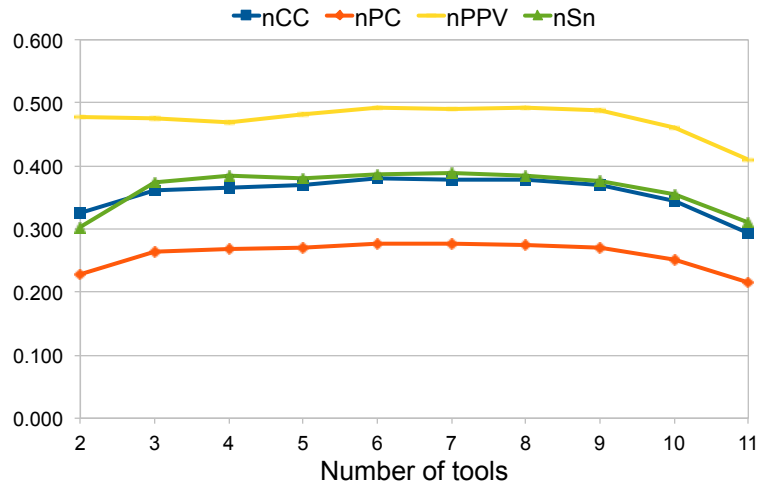

Figure 8: *Correlation Coefficient* (*nCC*), *Performance Coefficient* (*nPC*), *Positive Predicted Values* (*nPPV*) and *Sensitivity* (*nSn*) values obtained by CE^3 best configuration over the E. Coli dataset, when varying the number of tools. The maximum nCC value is reached for six and seven tools.

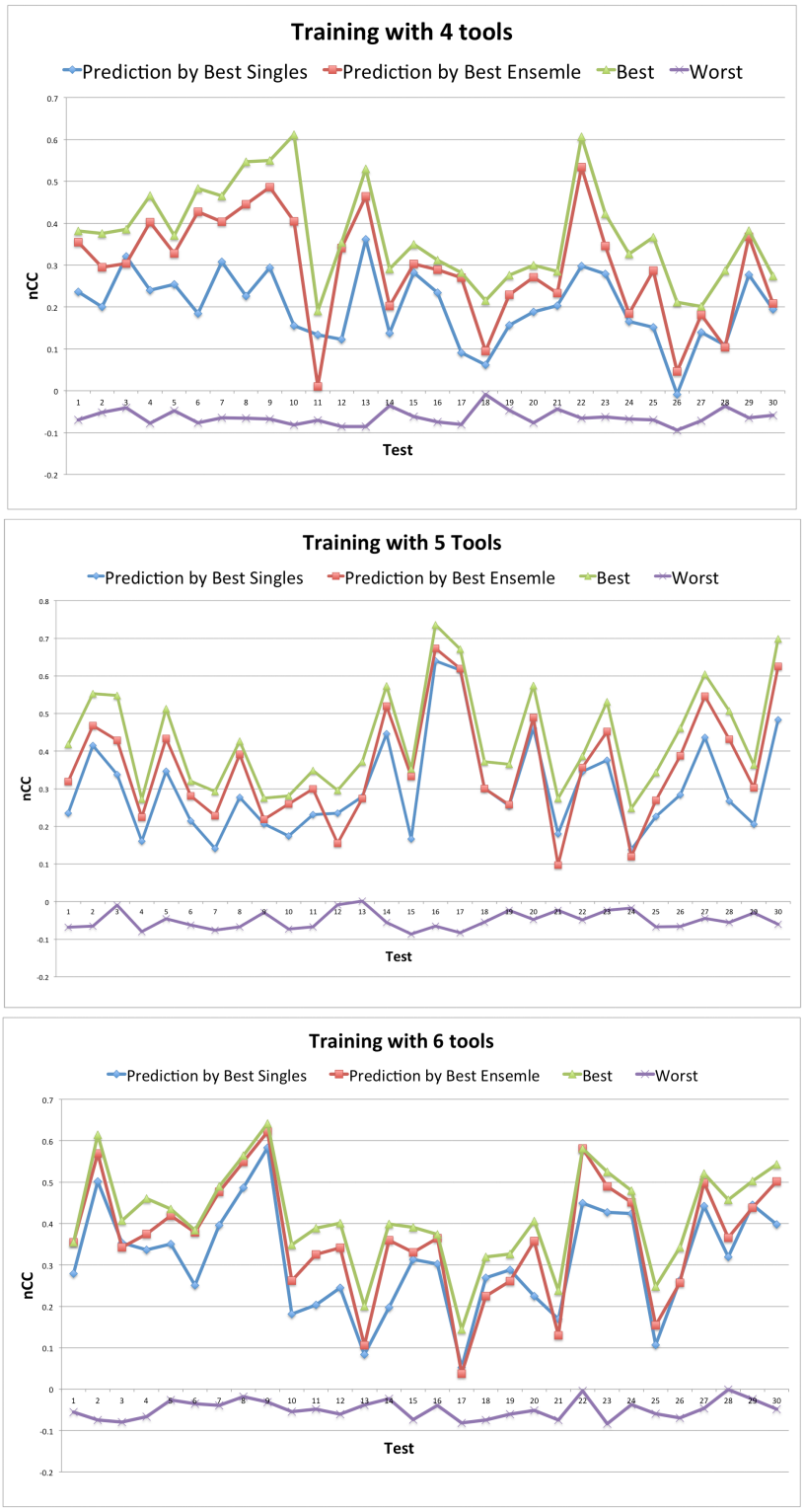

Figure 9: The training strategy tested on the E. Coli dataset. nCC values are shown for 30 independent tests, with the number of tools varying from four to six. *Best* is the nCC value obtained by the best performing configuration on the test set; *Worst* is the nCC value obtained by the worst performing configuration on the test set; *Prediction with Best Ensemble* is the nCC value obtained running CE^3 on input the test set with the best performing ensemble configuration for the training set; *Prediction with Best Singles* is the nCC value obtained running CE^3 on input the test set with the *n* best performing single finders on the training set.

from a small set of eight TFs but coming from different organisms.

By varying the number n of used tool from four to six, we repeated the following experiment K times.

1. randomly select m datasets and make them the *test set*;
2. let the remaining datasets form the *training set*;
3. for all possible combination of n tools, run CE^3 on the training set, compute the nCC values, and determine the configuration giving the best score;
4. use the best performing configuration to make prediction on the test set and compute the corresponding nCC value.

The value of K (resp. m) is set to 30 (resp. 5) for the E. Coli dataset, while for the G7 dataset K (resp. m) is set to 5 (resp. 2), being the latter dataset much smaller than the former.

The results obtained are reported in Figures 9 and 10. We compared the nCC achieved by CE^3 when run with the configurations determined by the strategy outlined above, denoted as *Prediction by Best Ensemble*, with the following values: *Best*/*Worst*, namely the best/worst nCC value obtained by CE^3 on the test set when considering "all" the possible sets of finders; *Prediction with Best Singles*, where CE^3 has been run on input the test set, using the n best performing single finders for the training set.

The first thing worth noticing is that the proposed training strategy gives results that are much closer to the best prediction achievable by CE^3 (in its default configuration and given a fixed number of finders) than to the worst one, sometimes even the best itself. Second, the results are better when sequences come from the same organism (as in the E. Coli dataset) than from different ones, meaning that the gap between prediction and best achievable solution is smaller. Nevertheless, even when the training and test sets only have the same set of transcription factors in common (as in the G7 dataset), results are still encouraging. On the contrary, the same experiment on the Tompa dataset gives very bad results, probably due to the fact that sequences come from very different organisms and there is a large heterogeneity in the transcription factors binding to the sequences (these are not well conserved and might vary a lot in length).

A final observation has to do with the *Prediction with Best Singles* strategy. Actually, running CE^3 with the combination of 4-6 tools that alone performed best on the training set achieves fairly good results as well. These however, are in general slightly worse than those returned by the strategy that makes prediction based on the best ensemble combination. The general better accuracy granted by the latter might be balanced, depending on the applications and the size of the dataset (and the number of available finders as well), with the much more cheap computational cost of the former.

5 Conclusions and Further Work

In this paper we have presented a first prototype implementation of CE^3, an ensemble tool for motif discovery whose key quality is that of being extensible and tailorable with respect to the (third-party) motif finders used and the learning strategies adopted: CE^3 has the potential to simulate almost all the currently available ensembles and to "easily" allow the user to create her/his ad-hoc ensemble, *i.e.*, without starting from scratch.

Test showed that CE^3 achieves interesting performances when compared to single finders and other ensemble methods. Moreover, we devised and tested a training strategy to help the user define a configuration to use CE^3 to make prediction on new datasets.

Work in progress includes further investigations with new datasets and the construction of a Web application that will also include a (customizable) implementation of the training strategy introduced in section 4.3.1. Future work include the investigation of the possibility to integrate CE^3 with some DNA analysis platforms.

Conflict of Interest

None declared

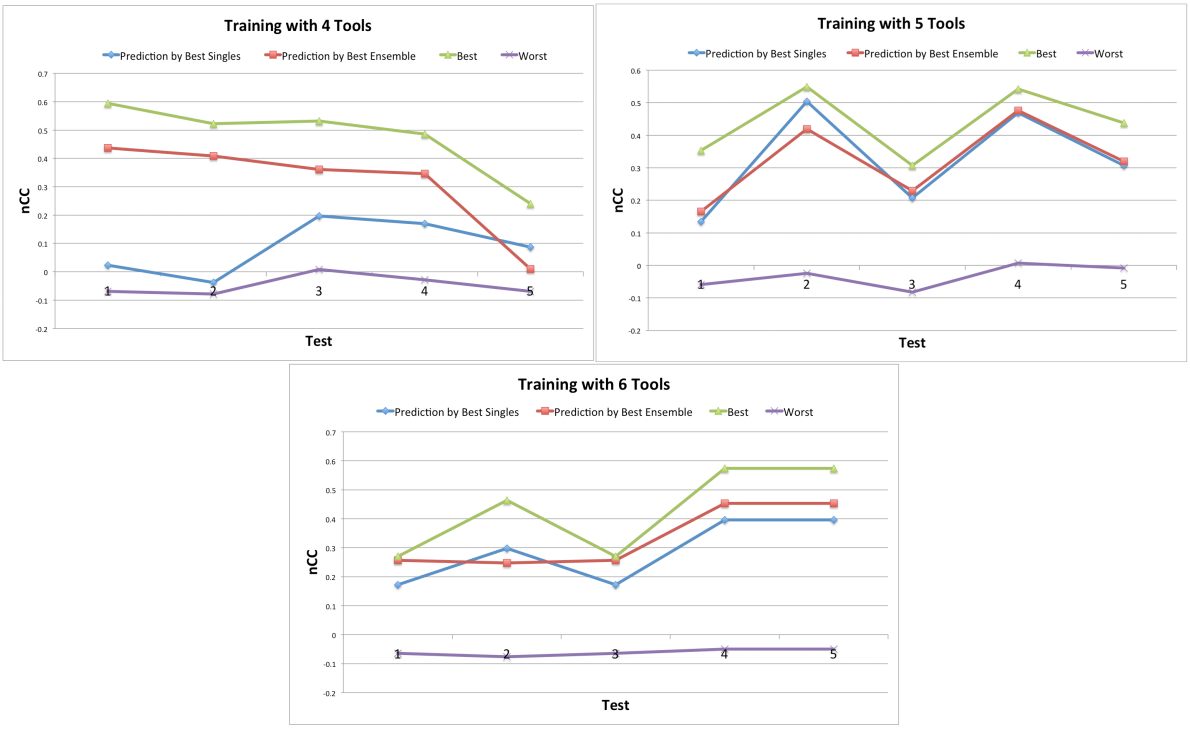

Figure 10: The configuration strategy tested on the G7 dataset (legend is explained in text and in the caption of Figure 9). nCC values are shown for 5 independent tests, with the number of tools varying from four to six.

Acknowledgments

None declared

References

[1] Stormo GD. DNA binding sites: representation and discovery. Bioinformatics. 2000;16(1):16–23.

[2] D'haeseleer P. What are DNA sequence motifs? Nature Biotech. 2006;24:423–425.

[3] Bailey TL, Williams N, Misleh C, Li WW. MEME: discovering and analyzing DNA and protein sequence motifs. Nucleic Acids Research. 2006 July;34(Web-Server-Issue):369–373.

[4] Lipman DJ, Altschul SF, Kececioglu JD. A tool for multiple sequence alignment. In: Proceedings of National Acadademy of Sciences. vol. 86. USA; 1989. p. 4412–4415.

[5] Pavesi G, Mauri G, Pesole G. An algorithm for finding signals of unknown length in DNA sequences. Bioinformatics. 2001;17(1):207–214.

[6] Das MK, Dai HK. A survey of dna motif finding algorithms. BMC Bioinformatics. 2007;8.

[7] Tompa M, et al. Assessing computational tools for the discovery of transcription factor binding sites. Nature Biotech. 2005;23(1):137–144.

[8] Chakravarty A, Carlson JM, Khetani RS, Gross RH. A novel ensemble learning method for de novo computational identification of DNA binding sites. BMC Bioinformatics. 2007;8.

[9] Hu J, Yang YD, Kihara D. EMD: an ensemble algorithm for discovering regulatory motifs in DNA sequences. BMC Bioinformatics. 2006;7.

[10] Huber BR, Bulyk ML. Meta-analysis discovery of tissue-specific DNA sequence motifs from mammalian gene expression data. BMC Bioinformatics. 2006;7.

[11] Wijaya E, Yiu SM, Son NT, Kanagasabai R, Sung WK. MotifVoter: a novel ensemble method for fine-grained integration of generic motif finders. BIOINFORMATICS. 2008;24(20):2288–2295.

[12] Zambelli F, Pesole G, Pavesi G. Motif discovery and transcription factor binding sites before and after the next-generation sequencing era. Brief in Bioinf. 2012;14:225–237.

[13] Kim N, Tharakaraman K, Mario-Ramrez L, , Spouge JL. Finding sequence motifs with Bayesian models incorporating positional information: an application to transcription factor binding sites. BMC Bioinformatics. 2008;9.

[14] Hughes JD, et al. Computational identification of Cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. J Mol Biol. 2000;296(5):1205–1214.

[15] Liu X, Brutlag DL, Liu JS. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In: Proceedings of the 7th Pacific Symposium of Biocomputing (PSB); 2001. p. 127–138.

[16] Thomas-Chollier M, Defrance M, Medina-Rivera A, Sand O, Herrmann C, Thieffry D, et al. RSAT 2011: regulatory sequence analysis tools. Nucleic Acids Research. 2011;39(suppl 2):W86–W91.

[17] Liu XS, Brutlag DL, Liu JS. An algorithm for finding protein-DNA binding sites with applications to chromatin-immunoprecipitation microarray experiments. Nat Biotechnol. 2002;8(20).

[18] Ao W, Gaudet J, Kent WJ, Muttumu S, Mango SE. Environmentally induced forgut remodelling by PHA-4/FoxA and DAF-12/NHR. Science (New York, NY). 2004;305(5691):1743–1746.

[19] Edward Wijaya SMY Kanagasabai Rajaraman, Sung WK. Detection of Generic Spaced Motifs Using Submotif Pattern Mining. Bioinformatics. 2007;23:1476–1485.

[20] Favorov A, Gelfand M, Gerasimova A, Ravcheev D, Mironov A, Makeev V. A Gibbs sampler for identification of symmetrically structured, spaced DNA motifs with improved estimation of the signal length. Bioinformatics. 2005 May;21(10):2240–2245.

[21] Gordon DB, Nekludova L, McCallum S, Fraenkel E. TAMO: a flexible, object-oriented framework for analyzing transcriptional regulation using DNA-sequence motifs. Bioinformatics. 2005;21(14):3164–3165.

[22] Garcia F, Lopez FJ, Cano C, Blanco A. FiSIM: A new similarity measure between transcription factor binding sites based on the fuzzy integral. BMC Bioinformatics. 2009;10.

[23] Hu J, Li B, Kihara D. Limitations and potentials of current motif discovery algorithms. Nucleic Acid Res. 2005;33:4899–4913.

[24] Eskin E, Pevzner PA. Finding composite regulatory patterns in DNA sequences. Bioinformatics. 2002;18:354–363.

[25] Workman CT, Stormo GD. ANN-Spec: a method for discovering transcription factor binding sites with improved specificity. Pac Symp Biocomput. 2000;5:467–478.

[26] Osada R, Zaslavsky E, Singh M. Comparative analysis of methods for representing and searching for transcription factor binding sites. Bioinformatics. 2004;20(18):3516–3525.

[27] Wei Z, Jensen ST. GAME: detecting cis-regulatory elements using a genetic algorithm. Bioinformatics. 2006 Jul;22(13):1577–1584.