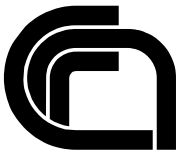

*Consiglio Nazionale delle Ricerche*

# *CE*$^3$: Customizable and Easily Extensible Ensemble Tool for Motif Discovery

K.Tillán, M. Leoncini, M. Montangero

IIT TR-16/2012

**Technical report**

**Ottobre 2012**

**Istituto di Informatica e Telematica**

# $CE^3$: Customizable and Easily Extensible Ensemble Tool for Motif Discovery

Karina Panucia Tillán
Dip. di Scienze e
Metodi dell'Ingegneria
Univ. di Modena e
Reggio Emilia, Italy
83672@studenti.unimore.it

Mauro Leoncini
Dip. di Ingegneria
dell'Informazione
Univ. di Modena e
Reggio Emilia, Italy
CNR, Istituto di Informatica e
Telematica, Pisa, Italy
leoncini@unimore.it

Manuela Montangero
Dip. di Ingegneria
dell'Informazione
Univ. di Modena e
Reggio Emilia, Italy
CNR, Istituto di Informatica e
Telematica, Pisa, Italy
manuela.montangero@unimore.it

## ABSTRACT

Ensemble methods (or simply *ensembles*) for motif discovery represent a relatively new approach to improve the accuracy of stand-alone motif finders. The performance of an ensemble is clearly determined by the included finders as well as the strategy to combine the results returned by the latter (the so called *learning rule*). A potential obstacle to a widespread adoption of ensembles is that the choice of the particular finders included is closed. Although possible in principle, the addition to an ensemble of a new "promising" tool requires knowledge of the internals of the ensemble and usually non trivial programming skills.

In this research we propose a general architecture for ensembles and a prototype called $CE^3$: *Customizable and Easily Extensible Ensemble*, which is meant to be extensible and customizable at the level of the two key components modules namely *external tools finding* and *learning rule*. In this way the user will be able to essentially "simulate" any existing ensemble, create his/her own ensemble according to his/her preferences on finding tools and learning functions and, finally, keep it up to date when new tools and new ideas for learning functions are proposed in literature. These features also make $CE^3$ a suitable tool to perform experiments that may lead to a proper configuration of ensembles in the research of novel motifs.

## Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain-specific architectures; J.3 [**Life and Medical Sciences**]: Biology and genetics

## Keywords

ensemble methods, motif discovery, software architecture

## 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 typically lengthy and costly, the problem has been tackled also from a computational perspective. Mathematical models of TFBSs have been proposed [17, 5], often termed *motifs*, and many algorithms have been designed and implemented in the last thirty years (see, e.g., [1, 18, 16, 15], and [3, 4] for many other references).

Despite such impressive efforts, the prediction accuracy remains low. A relatively recent assessment of thirteen popular algorithms performed by Martin Tompa and co-authors has made it clear that no single method (among the ones tested) performs well on different datasets, and that it is by no means easy to characterize the inputs for which a method may give good performances [7].

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 that follow this paradigm are known as *ensemble methods* (or simply *ensembles*) [2, 10, 11, 20], but sometimes also termed meta-predictors [21].

The main supporting argument behind ensemble design is embodied in the following observation: it is unlikely that algorithms that use different computational strategies, and possibly different TFBSs models, may agree on common stretches of DNA being functional sites by chance only. According to this intuition, virtually all proposed ensembles tend to rank the highest those common predictions made by more finders. The actual procedures adopted to "combine" the finders' results, often referred to as the *learning rules*, may vary a lot across different ensembles. Together with the choice of the actual finders used (typically third-party, external software tools), the learning rule is the feature that most affects the performance of an ensemble.

In this paper, we define the structure of a general ensemble

architecture for motif discovery, called $CE^3$, which is *customizable* and *extensible* with respect the two key features mentioned above. We also present a first prototype implementation of $CE^3$. The key design goal is to make $CE^3$ able to simulate almost all the available ensembles, also giving end users the possibility to create their own tool through the choice of specific finders (and or the addition of new ones) and of the specific learning rule to adopt.

The addition of a new finder is a semi-automatic process in $CE^3$, which is guided by the system. During this interaction stage, $CE^3$ gathers (through a question-answering procedure) the information needed to run the new finder and to parse its results, storing such information in appropriate XML files. Currently to be included in the ensemble, a finder must run as a command line utility under Unix/Linux operating systems.

Even new learning rules may be added to $CE^3$. This, however, is a somewhat more complicated task. In fact, while many motif finders exist as "stand-alone" software components, learning functions of interest are usually part of larger software modules, from which they cannot (often) be factored out as easily. To include new functionalities, though, $CE^3$ requires that they be coded with well-defined application programming interfaces. When this is the case for a learning rule, the addition process is again a semi-automatic procedure guided by the system.

Altogether, extending $CE^3$ with new functionalities (new motif finders and/or new learning rules) is by far much easier than developing a new tool from scratch. In spite of our efforts, however, we cannot really maintain that extensibility be directly exploitable by end users (say, in biomedical environments). However, only little bioinformatic support should suffice. To the end user, though, mature $CE^3$ (which actually means "when Web access will be made available") will appear as a customizable, highly flexible tool, giving the possibility to perform experiments under many different (but predefined) configurations.

$CE^3$ includes other software components as well, which are clearly important but that we do not regard as the ones that mostly characterize our proposal. For this reason, $CE^3$ currently implements the corresponding functions using already well-established solutions. In particular, for the internal motif representation and manipulation, $CE^3$ builds on the TAMO package [8], while for the statistical evaluations it uses the frequency files available with the RSAT tool [19]. $CE^3$ is written in Python and is available from the authors upon request.

A detailed description of $CE^3$ architecture is reported in Section 2, together with a summary of the steps required to add new motif finders and new learning functions. Note that here we do not describe any experiments performed with $CE^3$ on biological data. From the one hand, possible good results obtained using $CE^3$ can not be ascribed to $CE^3$ itself, but rather to the particular configuration with which it is used. On the other hand, to find such good configurations requires performing a great number of experiments on a blend of different datasets, which is matter of ongoing work.

# 2. ENSEMBLE ARCHITECTURE

Ensembles used for the *Motif Discovery Problem* (*MDP*) integrate 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.

- *External algorithms integration module*: ensembles integrate possibly many different (third-party) *de-novo* finders.
- *Internal motif representation*: the motifs returned by the finders are represented in a uniform way using appropriate data structures and internal motif handling software.
- *Learning rule*: one or more learning techniques are used to discover the most promising motifs. This is the component that mostly characterizes (and distinguishes) today's available ensembles.
- *Output module*: the predictions 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 list of the sites found in the input sequences.

The crucial components are clearly the first and third ones.

The choice of the included finders may be 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 [20]) allow the user to select the particular finders s/he wishes to use in a particular run; however these are 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 with a new algorithm (say one that adopts a new powerful search strategy), s/hef has to do some non trivial programming work (whenever 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.

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 tools authors and hence changes are difficult to implement. Moreover, even greater efforts are requested if one wishes to add a completely new learning method to the ensemble.

## 2.1 Architecture of $CE^3$

$CE^3$'s key features of easy customization and functionality extension depend on a conceptually simple modification of the generic ensemble's structure as shown in Figure 2.

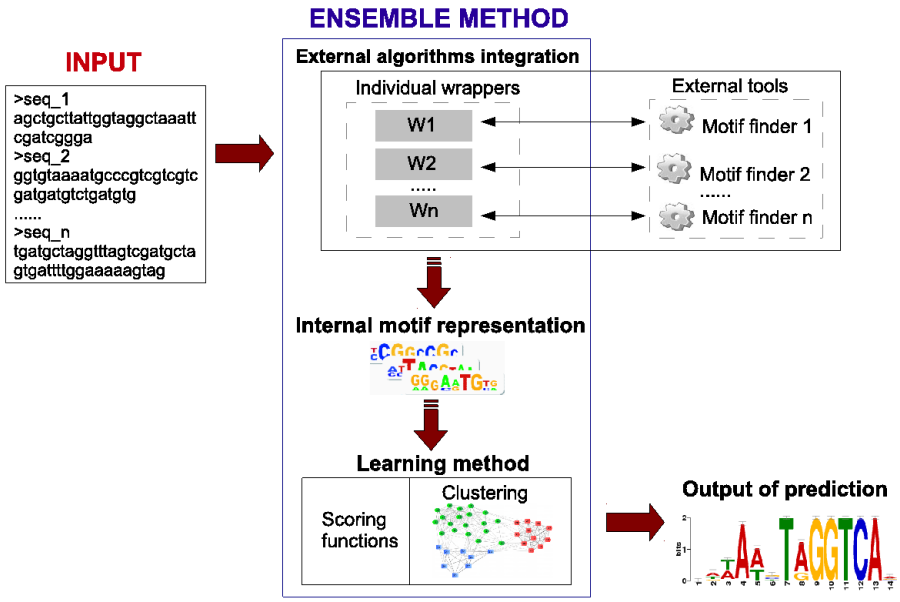


**Figure 1: General architecture of an ensemble. Each external tool is interfaced through a specific wrapper included in the ensemble. The motifs predicted by the external tools are converted to a unique internal representation and "combined" by the learning methods. The most promising combination is reported as the output of the ensemble.**

*Motif finder extensibility.* There is a unique general wrapper that specialize to the various external finders thanks to the finder descriptions available in XML configuration files. In order to add a new tool, only two files must be provided, either directly or indirectly by means of a semi-automatic process guided by the system (which does not require programming abilities).

There are some limitations, of course. The finders that can be included must satisfy the constraints listed below. Luckily, this happens for many of the most popular finders proposed so far.

1. The tool must search the motifs in a set of sequences stored in a fasta formatted file.
2. The tool itself 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*, *PWMs*.

Currently, $CE^3$ includes eight motif finders, namely: Aglam [12], AlignAce [6], BioProspector [13], MotifSampler, [18], MEME [1], RSAT (`oligo-analysis` and `pattern-assembly`) [19], MDscan [14] and Weeder [15].

*Learning function customization.* There is a module in $CE^3$ that handles the available learning functions of the ensemble. Each learning function is stored in its own folder, that contains a standard Python interface (easily derivable by a template) and an XML configuration file used by the module to recognize and include the function in the ensemble.

Currently $CE^3$ provides a learning function (customizable by means of a set of options), based on the idea of clustering similar motifs, that will be later described in details.

## 2.2 The General Wrapper

The design of a 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.

*Tool Running.* Each motif finder developed so far defines its own set of parameters and command line syntax. In order to masquerade the 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 a general wrapper able 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

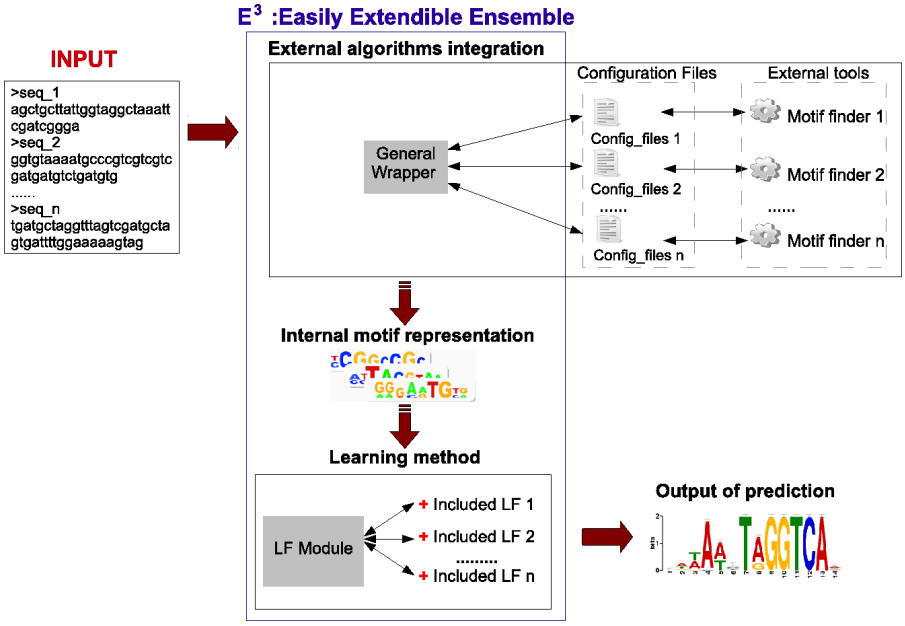


**Figure 2: $CE^3$ architecture. Differently from other ensembles, the integration of external algorithms depends on a general wrapper and configuration files for each external tool included. Distinct learning functions (LF) may be included and executed in the ensemble by means of the Learning Function Module.**

*nuisance parameters* [9]). 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 rule 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,** which is clearly mandatory;

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

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

**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;

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

We also included one tag for providing *seeds* to probabilistic methods (to guarantee the possibility of repeating experiments) and one for all the other (non mandatory) parameters, which will be given as a single unstructured string to be inserted "as is" in the command line. The template XML configuration file is depicted in Figure 3. With the structure of the file being predefined in $CE^3$, the user has only to associate the appropriate values to the right tags and attributes as required by the system.

```
<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>
```

**Figure 3: *XML configuration file template.* The first seven tags define the typical tools' parameters. The `extra` tag gives room for tool specific parameters. 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 `cd` to the tool's directory before execution.**

*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. Also, 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 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

```
<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>
```

**Figure 4: *XML parse file template* 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.**

descriptions in the output results.

To store the information needed in the parsing phase, we have defined an *XML parse file* with predefined structures that help identifying the information of interest (Figure 4). In order to create one such file, for a given finder, the user must provide *static* information (rather than working code in a suitable programming language). The pieces of information required by $CE^3$ include the ones listed below.

**motif representation,** that is whether the motifs reported by the finder at hand are given as list of sites or as matrices;

**leading and trailing signatures** to help locate the motif description, which are well defined pieces of text "copied" by a sample output;

**modifiers,** namely those parts of the signatures that may vary, and that will be described using regular expression.

The general wrapper needs one XML configuration file and one XML parse file for each finder included in the ensemble.

### 2.2.1 Adding a New Algorithm

The process of adding a new finder is now a relatively easy process, in which $CE^3$ gathers (through question-answering) 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.

Figure 5 shows (part of) a typical output produced by Bioprospector and the corresponding XML parse file in Figure 6

## 2.3 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 prototype $CE^3$ implementation, we use a refined version of the TAMO MotifTools package [8], 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.

## 2.4 Learning Function Module

The *learning function module* handles the addition and the execution of learning functions. This module receives as input the set of motifs predicted by the motif finders executed in the previous phase and runs one learning functions among those included in the ensemble (at the user choice).

For each learning function the module requires a *learning function configuration XML file* describing the function and its parameters.

Each function included in the ensemble is stored in a dedicated directory (named after the function) that contains:

- all files implementing the function (written in any programming language);
- a Python interface with a predefined class and methods.

*Learning Function Configuration XML File.* The XML file defined for each learning function (see Figure 7) stores a description (using the tag `description`) and defines the set of parameters (in case of) needed for the execution of the new function. Each parameter is captured in the XML file by the tag `parameter`.

*Python Interface Template.* The Python interface template (see Figure 8) 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). The class contains a function (`exec_()`) used to invoke the learning function.

### 2.4.1 Adding a New Learning Function

The process of adding a new learning function (given its implementation) is now a relatively easy task. The user have to perform the following steps:

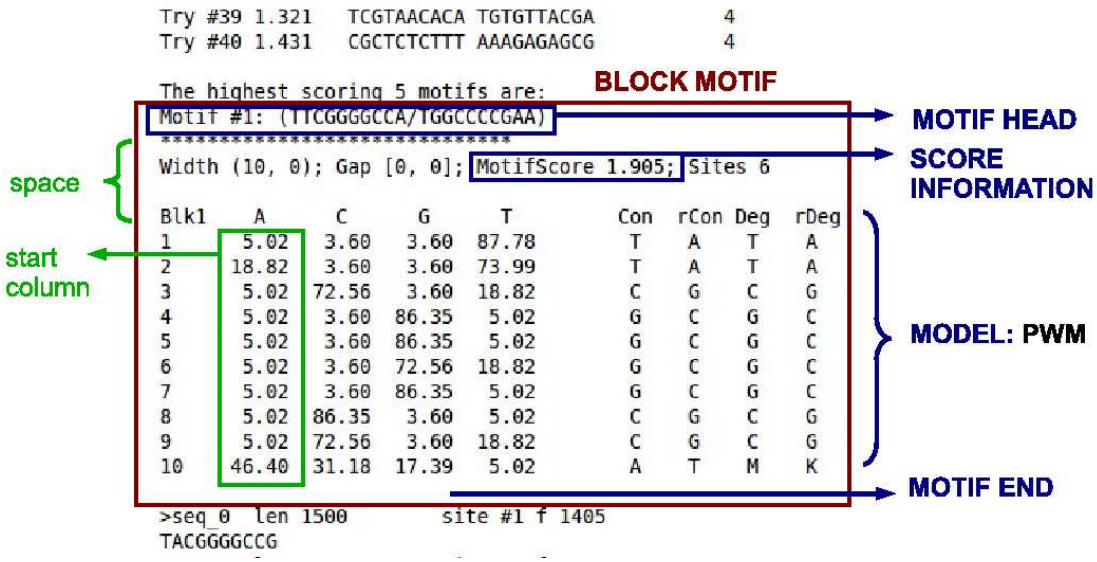


**Figure 5: Sample Bioprospector output listing: the large rectangle highlights a text block describing one motif, with all the relevant information needed to internally rebuild the motif itself.**

```
<parse>
<motif_head> Motif #'I': 'L' </motif_head>
<binding type="matrix" space="4" letters="Top" order="ACGT" start="1"></binding>
<motif_end> null </motif_end>
<score type="Evalue" key_score="MotifScore">
     Width ('I', 'I'); Gap ['I', 'I']; MotifScore 'F'; Sites 'L'
</score>
</parse>
```

**Figure 6: XML parse file corresponding to Bioprospector output listed in Figure 5: tags have been filled using the proper information identified in the output listing. In particular: 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, while the other 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). 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.**

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

**Figure 7: *Learning Function XML Configuration File.* 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.**

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

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

**Figure 8: *Python Interface Template.***

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 in the new folder) of the Python interface template: write in function `exec_()` the command line that invokes the new learning function.
4. Create the XML configuration file by specifying the name of the learning function, its description (a textual description) and, in case, define the extra parameters used for invocation.

#### 2.4.2 $CE^3$ Learning Method

At present, $CE^3$ implements its own strategy for post processing motifs predicted by motif finders (based on the idea of motifs clustering), leaving the user the possibility to chose among a certain number of options.

As for existing learning functions in other published ensembles, it is not trivial to retrieve them from tools code (*i.e.*, source code is not always available) and full details are not always given in papers, making the task of re-implementing existing functions a hard and time consuming one. In the future we plan to re-implement the most promising existing learning functions reported in the literature.

In its basic implementation, $CE^3$ select the motif(s) to be returned according to the following three step procedure in which, at each step, a few options are available to the user:

1. Compute the similarities of each pair of motifs. Two methods are currently available: (a) PWM similarity, and (b) sites overlapping. The former is implemented using RSAT's utility COMPARE-MATRICES [19]; the latter is 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.
2. Compute motif clusters using the chosen similarity measure. Here again two options (*i.e.*, clustering methods) are available: (1) 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; (2) 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: (i) average similarity of cluster members, (ii) number of contributing finders to motifs in cluster, (iii) cluster cardinality.

$CE^3$ allows the user to choose the number of top scoring clusters to be returned.

### 2.5 Output Module

The ensemble output is a set of putative motifs, given as a set of binding sites or a set of PWMs. $CE^3$ also allows learning functions to provide optional output information (*e.g.*, statistics).

#### 2.5.1 $CE^3$ Output and Site Prediction

In current $CE^3$ implementation, when using its learning method, for any cluster $C$ in the output set, $CE^3$ returns the following pieces of information: (a) PWMs of all component motifs (optionally supplemented with a text logo), some cluster statistics (minimum, maximum, and average similarity, number of contributing finders), and the indication of the cluster representative. The latter is simply the motif exhibiting maximum similarity to all the other motifs in the cluster.

The putative sites corresponding to $C$ are computed as the hits of its representative motif, but only retaining those sites that are (partially) overlapped by at least one site corresponding to a different motif. Note, however, that with the PWMs of all the motifs in the cluster, a client application using $CE^3$ might perform different scans of the input sequences.

## 3. CONCLUSIONS AND FURTHER WORK

In this paper we have presented a first prototype implementation of $CE^3$, an ensemble general architecture for motif discovery. $CE^3$'s key feature is that of being *tailorable* with respect to the crucial functionalities that characterize virtually all currently available ensembles: (1) the particular selection of external, (typically) third-party motif discovery tools, and (2) the choice of the learning rule, i.e., the strategy adopted to "combine" the tools' results in the *ensemble result* to be reported.

Even more than this, $CE^3$ is *extensible*, meaning that it makes it a relatively easy task the addition of new functionalities (new promising motif finders and/or learning rules), which indeed can be done with only minor user intervention

Ongoing work on $CE^3$ is following two different routes. First of all, we plan to perform a comprehensive set of experiments on a number of benchmark data with the goal of possibly finding good "standard" configurations (i.e., a base set of finders together with a specific learning rules) that guarantee good results across different datasets. These standard configurations will be those proposed to the end users as the most effective ones.

The second line of activity is essentially technical and has to do with the deployment of $CE^3$ as a Web-service. This is crucial for "real" end-user (e.g., molecular biologists) to experiment with our software.

## 4. REFERENCES


[1] T. L. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with meme. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, pages 21–29, Menlo Park, CA, 1995. AAAI Press.

[2] A. Chakravarty, J. M. Carlson, R. S. Khetani, and R. H. Gross. A novel ensemble learning method for de novo computational identification of dna binding sites. *BMC Bioinformatics*, 2007.

[3] M. K. Das and H. K. Dai. A survey of dna motif finding algorithms. *BMC Bioinformatics*, 8, 2007.

[4] P. D'haeseleer. How does dna sequence motif discovery work? *Nature Biotechnology*, 24:959–961, 2006.

[5] P. D'haeseleer. What are dna sequence motifs? *Nature Biotechnology*, 24:423–425, 2006.

[6] J. H. et al. Computational identification of cis-regulatory elements associated with groups of functionally related genes in saccharomyces cerevisiae. *J. Mol. Biol.*, (296):1205–1214, 2000.

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

[8] D. B. Gordon, L. Nekludova, S. McCallum, and E. Fraenkel. Tamo: a flexible, object-oriented framework for analyzing transcriptional regulation using dna-sequence motifs. *Bioinformatics*, 21(14):3164–3165, 2005.

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

[10] J. Hu, Y. D. Yang, and D. Kihara. Emd: an ensemble algorithm for discovering regulatory motifs in dna sequences. *BMC Bioinformatics*, 2006.

[11] B. R. Huber and M. L. Bulyk. Meta-analysis discovery of tissue-specific dna sequence motifs from mammalian gene expression data. *BMC Bioinformatics*, 2006.

[12] N. Kim, K. Tharakaraman, L. Mario-Ramrez, , and J. Spouge. Finding sequence motifs with bayesian models incorporating positional information: an application to transcription factor binding sites. *BMC Bioinformatics*, 9, 2008.

[13] X. Liu, D. L. Brutlag, and J. S. Liu. Bioprospector: discovering conserved dna motifs in upstream regulatory regions of co-expressed genes. pages 127–138, 2001.

[14] X. S. Liu, D. L. Brutlag, and J. S. Liu. An algorithm for finding protein-dna binding sites with applications to chromatin-immunoprecipitation microarray experiments. *Nat. Biotechnol.*, 8(20), 2002.

[15] G. Pavesi, G. Mauri, and G. Pesole. An algorithm for finding signals of unknown length in dna sequences. *Bioinformatics*, (17):207–214, 2001.

[16] P. A. Pevzner and S.-H. Sze. Combinatorial approaches to finding subtle signals in DNA sequences. In *Proceedings of 8th International Conference on Intelligent Systems for Molecular Biology (ISMB '00)*, pages 269–278, 2000.

[17] G. D. Stormo. Dna binding sites: representation and discovery. *Bioinformatics*, 16(1):16–23, 2000.

[18] G. Thijs, K. Marchal, M. Lescot, S. Rombauts, B. D. Moor, P. Rouze, and Y. Moreau. A gibbs sampling method to detect over-represented motifs in the upstream regions of co-expressed genes. *Computational Biology*, 2002.

[19] M. Thomas-Chollier, M. Defrance, A. Medina-Rivera, O. Sand, C. Herrmann, D. Thieffry, and J. van Helden. Rsat 2011: regulatory sequence analysis tools. *Nucleic Acids Research*, 39(suppl 2):W86–W91, 2011.

[20] E. Wijaya, S. M. Yiu, N. T. Son, R. Kanagasabai, and W. K. Sung. Motifvoter: a novel ensemble method for fine-grained integration of generic motif finders. *BIOINFORMATICS*, 24(20):2288–2295, 2008.

[21] F. Zambelli, G. Pesole, and G. Pavesi. Motif discovery and transcription factor binding sites before and after the next-generation sequencing era. 2012.