Laboratory of Julie D. Forman-Kay

User Manual

Dr. Mickaël Krzeminski
November 2012

E N S E M B L E

Version 2.1

Copyright © The Hospital for Sick Children, 2012

Distribution of substantively modified versions of any module of this software package is prohibited without the explicit permission of the copyright holder.

Any use of this work or derivative works in whole or in part for any commercial purpose of for monetary gain is prohibited.

NO WARRANTY
This software package is provided 'as is' without warranty of any kind, expressed or implied.

Inside this manual, you will find some light bulbs. The yellow bulb indicates critically important information that should be included in the first reading, while the purple light bulb highlights notes that are less critical and can be read at a later time.

INTRODUCTION

1. Historical development

ENSEMBLE was created to describe the structural ensembles of disordered protein states, including unfolded and intrinsically disordered states. The development of ENSEMBLE was designed to meet the need to incorporate multiple experimental data into computational calculations of these states, which are important for understanding protein stability and aggregation in the case of unfolded states and for understanding biological function and protein recognition in the case of intrinsically disordered states. The ENSEMBLE concept first emerged in 2000 from the work of James Choy *et al.*[1], who wrote an algorithm to optimize the weights for each conformer in a set of pre-generated structures in order to satisfy the available experimental data. This original version was optimized later on by Chris Neal, who considerably increased the speed of the program by implementing new pseudo-energy minimization algorithms in C language, and Joe Marsh, who significantly extended the capacities of the program including integrating an iterative PERL routine that leads to better conformational sampling[2-4]. This latter version has been recently modified and enhanced by Mickaël Krzeminski and includes an easy user interface, as well as new approaches to treat data and analyze results.

The many experiments that yield information about disordered states of proteins, together with the computational algorithms for predicting the same type of information from a structural model, are utilized within ENSEMBLE to enable determination of an ensemble of conformations that represent the disordered state. ENSEMBLE has primarily been developed using the unfolded state of the N-terminal SH3 domain of Drk (downstream of receptor kinase) from *Drosophila* for which significant experimental data has been obtained. It has more recently been successfully applied to intrinsically disordered proteins to gain insight into their structural propensities and make structure-function correlations, including the cyclin dependent kinase inhibitor Sic1 both free and in complex with Cdc4[5] and regulators of protein phosphatase 1 (PP1)[6,7].

The current version of ENSEMBLE is maintained by Mickaël Krzeminski. Please, report any bug to mickael.krzeminski@sickkids.ca

2. The main core of ENSEMBLE

ENSEMBLE is a program written in C language, which aims to choose, from a large set of conformations called the *initial pool*, an ensemble that fits the available experimental data. To achieve this goal, the program makes use of a switching Monte-Carlo process embedded in a simulated annealing protocol. From the initial pool, ENSEMBLE starts by randomly choosing a user-defined number of conformers which constitute the first ensemble Ω_0. The agreement between the experimental data and the ones back-calculated for each conformer of Ω_0 is reflected through a scoring function: the lower, the better. Then, one structure from Ω_0 is swapped with one structure from the initial pool that does not belong to Ω_0, giving a new ensemble Ω_1. ENSEMBLE compares Ω_0 and Ω_1 and accepts or rejects the newly defined ensemble according to a Metropolis criterion that depends on the current temperature of the system. The process is repeated a user-defined number of times and the temperature is geometrically decreased at each step so that it becomes very close to zero at the end of the process.

The scoring function is the sum of the weighted pseudo-energy terms attributed to each data type, which we will refer to as *modules*. The weight attached to each energy term is called the *faith* factor and allows giving more or less importance to a given module.

Module	**Chemical Shift**	**RDC**	**NOE**	**PRE**	**PRE ratio**
Target Energy	$\sum_N \frac{\sigma_n^2}{16}$	$\frac{1}{16}N$	$\frac{1}{16}N$	$\frac{1}{16}N$	$\frac{1}{16}N(N-1)$
Module	**R2**	**J Coupling**	**Solvent Accessibility**	**R_H**	**SAXS**
Target Energy	$e^{K\times(1-c)}-1$	$\frac{4}{25}N$	$1\%\sum_N(low^2+up$	1% (up^2+low^2)	$10^{-4}\sum_N i_n^2$

Table 1 Modules utilized within ENSEMBLE and their target energy values. *N* is the number of data points, *n* is one specific data value. For chemical shifts, σ_n is the average standard deviation of the atom type *n* (data retrieved from the Biological Magnetic Resonnance Data Bank), and, for R2, *K* is a user defined constant and *c* is the target correction.

ENSEMBLE currently includes ten modules and, for each of them, a *target* energy is set below which the back-calculated data are considered to agree with the experimental data. The target

energy is always compared to the **non-weighted** energy calculated for the modules. Table 1 describes the different modules as well as their target energy.

The target energy is based on the equation that governs the energy calculation of its corresponding module:

- In the case of chemical shifts, one fourth of the average standard deviation for a given atom type is tolerated. The calculation of the energy for this module is based on a harmonic equation, giving one sixteenth of the square of the average standard deviation.

- For RDC data, a value of a quarter is tolerated per restraint. As the equation is harmonic, the target energy ends up with a value of one sixteenth of the total number of restraints.

- In the case of NOE and PRE restraints, the equation that gives the penalty energy is also harmonic. The program tolerates up to a fourth of Angstrom per restraint, giving a target energy of one sixteenth per restraint.

- For PRE ratio (rPRE) data, the number of data points is equal to $N(N-1)$ and the energy function is harmonic. Based on the NOE energy function, we obtain the value displayed in Table 1.

- The R2 energy is computed as $e^{K\times(1-r)}-1$. In this equation, K is a user defined constant and r is the correlation coefficient between back-calculated data and experimental data. Hence, for the target energy, the user can choose the factor *c*, which represents the minimum correlation above which a selected ensemble fits the experimental data.

- For scalar coupling J, we merely considered a reasonable error of 0.4 Hz. As the pseudo-energy function is harmonic, then the target score has the value displayed in Table 1.

- In the cases of Hydrodynamic radius (R_H) and Solvent accessibility, we arbitrarily define a target energy based on the specified error range.

- Finally, for SAXS data, the tolerated value corresponds to 10^{-4} times the sum of all squared intensities.

3. Running ENSEMBLE iteratively

The C-core of ENSEMBLE is more efficient in terms of time when the number of structures in the initial pool is rather low (up to 5000). Nevertheless, ENSEMBLE has been designed specifically for disordered proteins and obtaining a reasonable sampling of conformational space for these is not possible with such a low number of structures. Hence, we developed an approach to run ENSEMBLE iteratively after modifying the initial pool by adding new structures and rejecting irrelevant ones, so its size is kept reasonable. This is performed within what is called the *wrapper*, which consists of code for managing the initial pool, preparing all files required to properly run ENSEMBLE and retrieving the crucial information (selected ensemble, energies…). The wrapper has additional function in adjusting the faith factor and attributing more weight to modules for which the back-calculated values do not fit experimental data yet and less weight to modules for which the back-calculated values do fit. The following list summarizes the sequential steps of the wrapper:

1. Compaction of the initial pool if too large (> 5000 conformers)
2. Generation of new structures added to the initial pool (at regular steps)
3. Preparation of files necessary to run ENSEMBLE
4. Running ENSEMBLE (C-core)
5. Adjusting parameters (faith) or terminating the process (if back-calculated values fit experimental data)

An interesting aspect of this wrapper concerns the integration of restraints during the selection process. Each module is attributed a *rank* value and its restraints are integrated into the calculations only when the modules with lower rank (i.e. higher priority) have back-calculated values that fit the experimental data.

4. Requirements

To run properly, ENSEMBLE needs a main parameter file, as well as one restraint file per selected module. The pathway of these latter is specified in the parameter file, which also

contains the name of all the options related to the environment (results directory, number of structures in the selected ensemble…) and the protocols (inclusion of modules, minimization…). A template parameter file is provided in the installation directory of ENSEMBLE. Another example of a parameter file can also be found in the example directory; this one has to be used with the provided example in order to verify the correct installation of the program. A full description of the parameter file and the restraint files is given later in this document.

The C-core of ENSEMBLE requires three parameter files for the protocols, and one parameter file and one constraint file per module. The use of a wrapper enables all these files to be grouped into a single file and the wrapper creates all original parameters before launching the calculations.

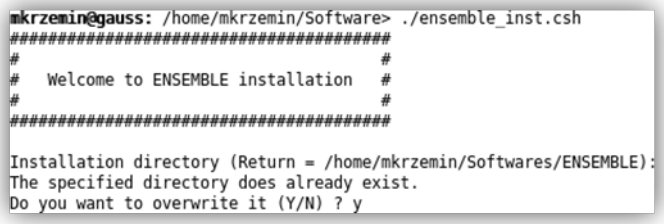
INSTALLATION

The ensemble.tar file that can be downloaded from http://pound.med.utoronto.ca/~JFKlab/ contains all the installation processes and is compatible with all Unix-based platforms. First, untar the file with the following statement:

```
tar –xf ./ensemble.tar
```

This command extracts two files in the current directory: ensemble.tar and ensemble_inst.csh. The latter is a C-Shell script that is run by simply typing ./ensemble_inst.csh at the prompt. The installation process will lead you through the following steps:

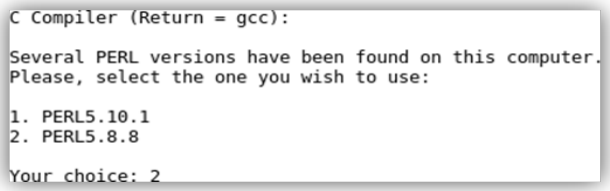
1. *The location in which the program will be installed.* By default, the location in which ENSEMBLE is installed is the home directory of the current user followed by Softwares/ENSEMBLE. This can be easily modified by entering another pathway. If a previous installation has already been done in the specified directory, you are warned and it is then proposed either to choose another directory or to overwrite the existing one. If the computer cannot create or overwrite the specified directory (permission issue), you will be invited to choose another one.



2. *The C compiler*. ENSEMBLE is written in C and a compiler is required to install it. The proposed compiler by default is gcc, already efficiently tested and installed on most Linux distributions. The installation program tests whether the proposed compiler comprises all the basic C libraries required by ENSEMBLE. If the test succeeds, the installation process goes directly to the next step; otherwise, another compiler is requested.

If you decide to cancel the installation at this point, you will have to manually remove the .test.c and .err files created for testing the C compiler.

3. *The PERL version.* The wrapper is written in PERL. Hence, the installation process searches in all pathways specified in the environment variable $PATH for all PERL versions. If several of them are found, a choice is given. If only one version is found, no choice is given, and if no PERL version is found, the installation process is aborted. The PERL version is tested, in particular the libraries crucial for the wrapper to run properly. If one is missing, the user is warned and the installation process is aborted.



To find all available PERL versions on your system, the script verifies each pathway specified in the environment variable $PATH. You can modify this variable if the version you want to use does not appear in the list proposed and before re-running the installation.

4. *Creation of the documentation directory.* In this directory this current user manual pdf file and the updates are found.
5. *Installation of the ENSEMBLE tools.* Some tools have been designed for a specific use within ENSEMBLE. After the installation is successfully completed, the environment variable $ENSEMBLE_TOOLS is created for directly accessing all the useful scripts located in the tools directory. The tools include the following programs (More details are given at the end of this document):

Managing	Filename	Purpose
CSP files	make_csp	Creates a CSP file from a list file
	extract_csp	Extracts some or all structures from a CSP file
	combine_csp	Combine two or more CSP files into one file
	combine_cdp	Combine two or more CDP files into one file
	get_info	Provides some information for any file created by ENSEMBLE
Analysis	accessurf	Calculates the solvent accessibilty of each atom of a protein
	analyzens	Executable designed to fully analyze an selected ensemble
	caca_map	Calculate the all Cα-Cα distances of an ensemble
	pdb2seq	Display the sequence of a protein from the PDB

	read_best	Display the evolution of the best ensembles found along ENSEMBLE runs
	ss_distr	Calculates the secondary structures in an ensemble using STRIDE

CSP stands for Concatenated Structure Pool. The CSP files contain the coordinates of structures in binary format.

6. *Installation of the predictors*. To back-calculate the data from a structural model, ENSEMBLE makes use of different predictors (See Table 2). The installation process indicates whether the installation of each of these predictors proceeded successfully.
7. *Installation of the main modules*. These modules encompass the main wrapper that will have to be launched, as well as all PERL libraries necessary to run it.
8. *Compilation of the C-core of ENSEMBLE.*
9. *Link to the wrapper*. If you are the administrator, a link to the wrapper is created in the /usr/bin/ directory so that all users will be able to run ENSEMBLE. Otherwise, an environment variable is added in the .cshrc and/or .bashrc files of your home directory.

Module	**Chemical Shift**	**RDC**	**NOE**	**PRE**	**rPRE**
Predictor	ShiftX[8]	Local alignment[9]	*	*	*
Module	**R2**	**J Coupling**	**Solvent accessibility**	**R_H**	**SAXS**
Predictor	*	*	*[10]	HYDROPRO[11]	CRYSOL[12]

Table 2 Predictors used within ENSEMBLE to back-calculate data. The star '*' means that back-calculation of data is internally computed by the C-core of ENSEMBLE. For more details about the methods utilized, see (1) and (2).

Installation of ENSEMBLE includes three external programs, ShiftX, HYDROPRO and CRYSOL, for incorporation of chemical shifts, hydrodynamic and SAXS restraints, respectively. Use of any of these modules requires that the appropriate literature(s) be cited. Use of CRYSOL by commercial entities requires, in addition, a commercial license from EMBLEM (http://www.embl-em.de/contact.php?lang=de&Cat=Contact).

DEEPER INTO ENSEMBLE

When running ENSEMBLE for the first time, the RUN directory specified in the main parameter file is created, as well as the subdirectories "Results", where all results (selected ensembles and related energies) are stored, "PDBs", where all the initial pools used during the ENSEMBLE runs are stored, and "Save", where the data necessary to restore the progression of the calculations in the event of a crash are stored. If ENSEMBLE is run again after a crash, it will first check in the Save directory for the presence of the file ensemble.dat, which contains all user-defined parameters. Moreover, once calculations are finished, a directory called "Analysis" is created upon launching the script *analysis.pl* found in the ENSEMBLE_TOOLS directory. The PERL script performs a couple of analyses on an ensemble that fits all experimental data.

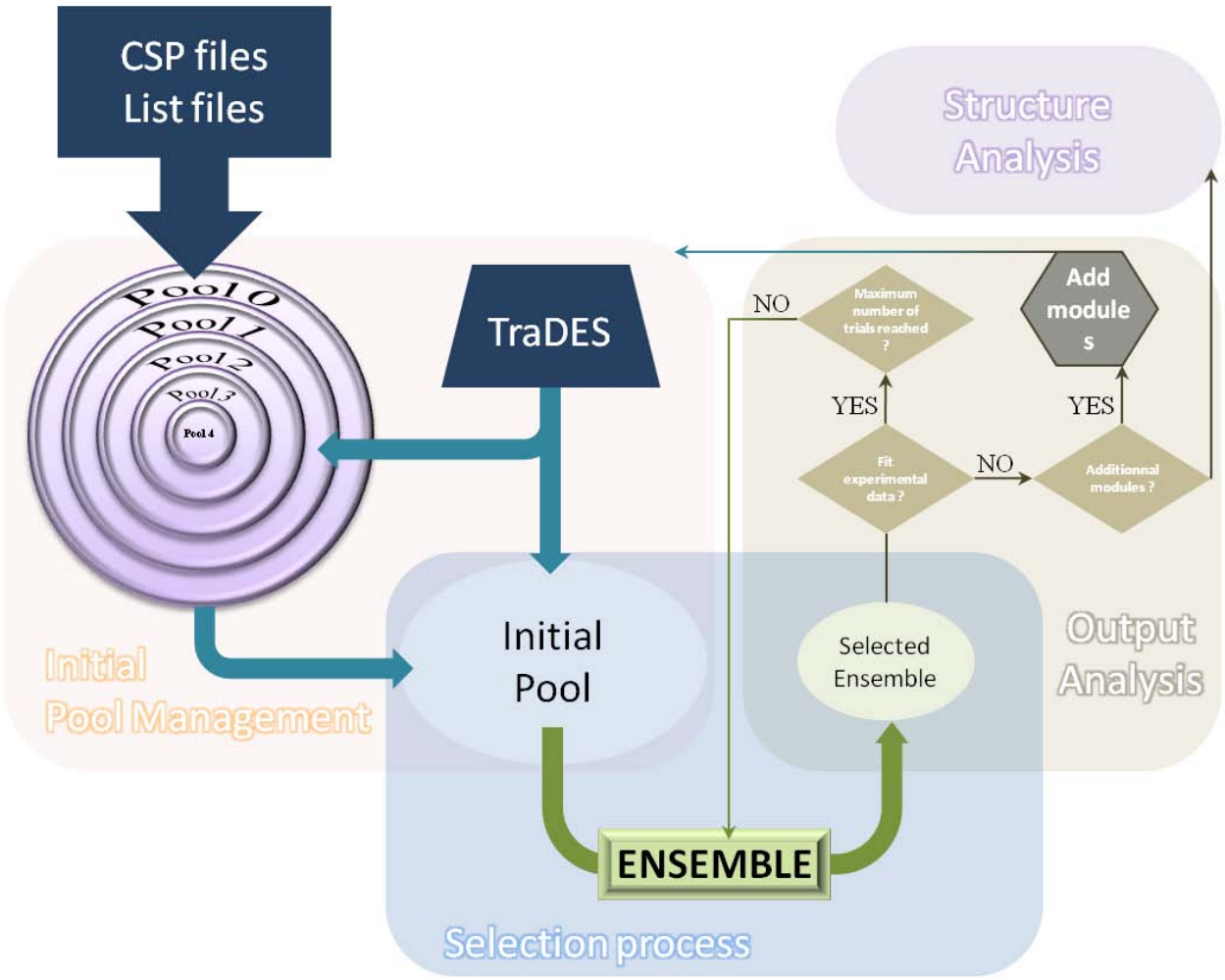


Fig. 1 The ENSEMBLE machinery. This is a schematic representation of the way the ENSEMBLE wrapper works.

The C-core of ENSEMBLE performs the selection of an ensemble from all conformers present in the initial pool. These latter are randomly selected from the initial soup, which is fed either from the structures provided before running ENSEMBLE (CSP or list files) and/or generated by TraDES package[13]. The selection process is iterated several times (the flag CHOOSE_NEW of the parameter file allows choosing the number of times) with the same initial pool, before selecting other conformers that will increase its size. Each iteration will be referred to as a *trial*, and a set of trials as a *run*. To prevent an initial pool from becoming too large, a compaction system is included, which starts first by keeping in the initial pool only structures that have been selected at least once by ENSEMBLE after any of all performed runs so far. If this number is higher than the user-defined maximum size of the initial pool (flag POOLSIZE), only some of the selected structures are chosen in a random way. On the other hand, if this number is lower than 1.5 times the size of the selected ensemble (flag NB_STRUCTURES), some other conformers are randomly selected from the initial pool

The way conformers are selected from the initial soup and included into the initial pool is random. The initial soup is split into several *selection groups*. The program randomly chooses a selection group, then a random conformer from this group. All conformers of a group have the same probability to be chosen and every time a conformer is selected, it goes to the downstream selection group. Initially, each selection group is attributed an "a priori" probability to be chosen. A probability of 1 is given to pool 0, where structures have never been selected, and 0 to the last one, where structures have been selected the user-defined maximum number of times (flag MAX_NUMBER_SELECT). Decreasing probabilities are given to the intermediate groups. The probability for a group to be chosen depends on its "a priori" probability, but also on the number of conformers it contains. This latter is taken into account to define the "a posteriori" probability, as following:

$$s_i = \frac{n_i \times p_i}{\sum_{j=0}^{N} n_j \times p_j} \qquad \textbf{Eq. (1)}$$

Where:

- s_i is the "a posteriori" probability of pool *i*,
- n_i is the number of conformers in the pool *i*,
- p_i is the "a priori" probability of pool *i*,
- N is the total number of pools.

Finally, a newly generated conformer is automatically inserted into the initial pool, as well as into pool 1 of the initial soup.

If some list files are provided (flag TRAJECTORY_FILE_X), they are first transformed into CSP files, which are directly saved in the directory of the trajectory file, if permissions allow it, otherwise in the Save directory, with the name of the trajectory file followed by '.csp'. From all CSP files, the program back-calculates the data for the selected modules and keeps them in memory. These data are also immediately saved into the directory specified in the parameter file (flag STORE_DATA) as CDP files (Concatenated Data Pool). If there is write permission in the directory where the CSP file is located, then the STORE_DATA directory is created there. Otherwise, it is created in the Save directory. The name of the CDP file is the same as the CSP file followed by the name of the module. For instance, from file1.csp, the CDP file that contains the back-calculated chemical shifts is called file1.csp.SHIFTX.

It is important to note that only the data back-calculated from the conformers initially provided are immediately saved, while new conformers generated with TraDES along the ENSEMBLE runs are only recorded in memory and will be saved only at a regular time (flag FREQ_SAVE).

ENSEMBLE enables one to choose the number of structures in the initial pool for the first run (flags NB_PICK_START and NB_GEN_START) and the number of structures to add to the initial pool after each run. There are three ways to add structures in the initial pool:

1. Random selection from the initial soup (flags NB_PICK_NEW),
2. Generate with TraDES (note that TraDES generated conformers are also inserted into pool 1 of the initial soup, so that they can be selected again for future runs (flags NB_GEN_NEW),
3. Generate an unfolded state starting from one random structure of the best ensemble found so far (flag NB_UNFOLDED). In this case, the process of unfolding the protein makes use of simulations that depend on the temperature of the system (flag UNFOLD_TEMP) and the timestep (flag UNFOLD_STEP). ENSEMBLE modifies the value of the temperature to get even more unfolded/extended structures, by adding to the specified value a random value so that the final

temperature can range between the specified temperature and twice its value. The same procedure is used for the timestep.

If the generation of structures by TraDES is activated, then the program will first start initiating four trajectory files, one based on alpha helices, one on beta sheets, one on coils and the last one on alpha helices, beta sheets and coils simultaneously. These files are crucial to generate conformers and are immediately stored in the Save directory. The choice of the trajectory file to generate new conformers is random.

Saving the data directly before running the selection process is done in the event of a crash that happens before the very first save. In such a case, the program will be much faster to initialize the next time it is started.

Finally, before running the calculation loop, the target energy of the selected modules is estimated (Table 1) and only the modules with the highest priority (lowest rank) are activated.

The first step of the loop consists of saving the user-defined data parameters, the initial pools utilized, the results (selected conformers, energies…) and the state of the different modules. Then, the files required by the C-core of ENSEMBLE are written. These latter encompass one main parameter file, a starting weighting file, three protocol files, and one parameter file per activated module. For some modules, ENSEMBLE needs the structural file of the conformers of the initial pool. In this case, these latter are extracted from the CSP files and stored locally in the pdbs subdirectory of the WORK directory. The C-core is then able to perform calculations properly. The output provides some information about the weighted energies of the different activated modules, as well as a weighting file that contains the proportion of each conformer of the initial pool in the selected ensemble. All this information is retrieved by the wrapper.

In the C-core, the energy of module ε_w^m is weighted by a faith factor. The wrapper removes this faith factor to obtain the real, or effective energy ε_e^m for each module . This effective energy is compared to the target energy ε_t^m for the same module (Table 1). The module is

considered to fit (i.e. the back-calculated values fit the experimental data) if ε_e^m is lower than ε_t^m.

The MAX_RELATIVE_WEIGHT flag of the parameter file defines the final ratio between the faith factor of a module that would always fit and the faith factor of a module that would never fit over all performed trials. Before the first trial of each run, the faith factor of all modules is set to 1. After each trial, the modules that do not fit the experimental data have their faith factor increased by a factor which corresponds to a linear progression of the value specified by MAX_RELATIVE_WEIGHT, while the faith factor of modules that fit remains the same. Then, the faith factor of all activated modules is decreased so that the lowest one is always 1. Finally, for ENSEMBLE efficiency, the faith factors of all activated modules are scaled so that the sum of the weighted energies is equal to a pre-defined energy (flag TARGET_ENERGY).

MRW = 10	Actual Faith Factor		ENSEMBLE Energy		Effective Energy		Fit or not		Updated Faith Factor	
Module	A	B	A	B	A	B	A (15.0)	B (7.5)	A	B
Trial 1	1.000	1.000	4.050	11.052	4.050	11.052	X		1.000	3.250
Trial 2	2.502	8.131	12.516	86.906	5.002	10.688	X		1.000	5.500
Trial 3	1.568	8.622	21.162	66.017	13.496	7.657	X		1.000	7.750
Trial 4	1.373	10.640	12.352	100.731	8.996	9.467	X		1.000	10.000
Trial 5	0.965	9.647	11.558	64.642	11.977	6.701	X			

MRW = 5	Actual Faith Factor		ENSEMBLE Energy		Effective Energy		Fit or not		Updated Faith Factor	
Module	A	B	A	B	A	B	A (12.5)	B (3.5)	A	B
Trial 1	1.000	1.000	14.551	4.940	14.551	4.940			1.000	1.000
Trial 2	5.131	5.131	70.322	17.390	13.705	3.389		X	2.000	1.000
Trial 3	4.635	9.271	63.148	25.107	13.624	2.708		X	3.000	1.000
Trial 4	6.884	2.295	84.258	14.021	12.240	6.109	X		1.000	1.000
Trial 5	5.450	5.450	78.291	20.108	14.365	3.690				

Table 3 Faith factor variation along one ENSEMBLE run made of 5 trials and restrained by 2 modules A and B. MRW indicates the maximum relative weight. The target energy is specified between brackets in the "Fit or not" column. The effective energy corresponds to the ratio between the ENSEMBLE energy and the actual faith factor.

Table 3 shows two examples of the evolution of the faith factor for two modules (A and B) along two ENSEMBLE runs, each made of 5 trials. The upper table helps explain the notion of maximum relative weight. While module A always fits the experimental data, module B never fits them. Hence, for the last trial, the weight of module B is MWR=10 times the weight of module A.

The lower table was obtained with a MRW equal to 5, giving an increase of the faith factor of 1.0 per trial if a module does not fit. After trial 1, none of the two modules fits. Hence their faith factor is increased by 1 and becomes 2. As the lowest faith factor is then equal to 2, all faith factors are decreased by 1 so that the lowest faith factor of all modules is 1. After trials 2 and 3, only module B fits, leading to an increase of 2 of the faith factor of module A. After trial 4, module A fits but not module B. The faith factor of module A is decreased and the faith factor of module B is increased, giving in both cases a faith factor of 2, which then become 1.

PARAMETERIZATION

ENSEMBLE requires a single parameter file, which contains all information necessary for running it properly and efficiently. The name of this parameter file must be given as an argument when launching ENSEMBLE. The parameter file is made of several sections that are intended for a specific aspect of the program. Here is a summary of all options and their meanings.

1. ENSEMBLE environment

The ENSEMBLE environment section contains the main information related to the system environment of ENSEMBLE.

RUN

The RUN parameter corresponds to the directory where all results will be stored. These latter encompasses the initial pools used along all ENSEMBLE runs and the selected ensembles, as well as their energies. In the RUN directory is also found the Save directory that keeps all information necessary to re-run ENSEMBLE in the event of a crash, and the PDBlist directory that contains all initial pool the program used. Moreover, the Analysis directory is created upon launching the analysis of an ensemble.

WORK

The WORK parameter is the directory where all calculations will be performed and temporary files crucial for the C-core of the program stored. This directory is like a trash bin, which is entirely deleted at the end of calculations.

A great advantage of the WORK directories resides in the possibility of specifying the pathway of the RAM memory of your computer to significantly accelerate the calculations. On the other hand, you have to make sure the amount of memory is sufficient.

STORE_DATA

The name of this parameter corresponds to the directory where the CDP files are stored. This directory is created in the directory of the CSP or list file when permitted, otherwise it is created in the Save directory.

SEQUENCE

This is the sequence specified with one letter code. The sequence is used to generate initial trajectories with TraDES.

CSP_FILE

It is possible to include into the calculations pre-generated structures. This is done via the CSP_FILE flags. A simple example of specifying CSP and list files to include looks like the following:

```
CSP_FILE_1      /home/user/project_X/file1.csp
CSP_FILE_2      /home/user/project_X/file1.list
CSP_FILE_3      /home/user/project_X/file2.csp
```

These latter can be CSP or list files. The list files consist merely of a list of the absolute pathway of the structural files. The program first checks whether each structure exists and creates from this list a CSP file (with the same name followed by '.csp') that is directly stored in the same directory as the list file if you have write permission, otherwise in the Save directory.

Note that each CSP_FILE flag is followed by a number. It is important to keep these numbers associated with the same files in the event of a crash and re-start of the program.

FREQ_SAVE

At regular times or numbers of performed runs, ENSEMBLE saves the progression state of the calculations. This flag specifies the save frequency.

If a time is demanded, then the value must be followed by 'm' (e.g. 120m for every two hours), otherwise the value is considered as the number of runs.

2. ENSEMBLE protocol parameters

After these first parameters, the ENSEMBLE protocol parameters are specified.

POOLSIZE The maximum size of the initial pool.

NB_STRUCTURES The number of conformers in the final ensemble.

CHOOSE_NEW The number of times ENSEMBLE will attempt to find a set of structures from the same initial pool that can fit experimental data.

NB_PICK_NEW The number of structures to randomly choose from the initial soup to put into the initial pool after each run.

NB_PICK_START The number of structures to choose from the initial soup before the first run.

This option obviously makes sense only if some CSP and/or list files have been provided in the beginning.

NB_GEN_NEW The number of structures to generate with TraDES after each run. The newly produced conformers are inserted in pool 1 of the initial soup and put into the initial pool.

NB_GEN_START The number of structures to generate with TraDES before the first run. The newly produced conformers are inserted in pool 1 of the initial soup and put into the initial pool.

NB_BEST_TO_UNFOLD After each run, a conformer of the best ensemble found so far is randomly chosen. This structure is used by TraDES as a starting point in an unfolding process. This flag specifies the number of conformers to randomly pick up.

NB_UNFOLD_PER_CONF This is the number of unfolded conformers to generate from

each randomly selected conformer (flag NB_BEST_TO_UNOLD).

MAX_NUMBER_SELECT The number of times each conformer can be chosen from the initial soup and put into the initial pool.

PROB_PROG This parameter controls the *a priori* probability of each selection group of the initial soup. If an algorithmic progression is chosen (A), the *a priori* probability of pool n is obtained by removing from the probability of pool n-1 a certain value given that pool 1 has a probability of 1 and the last pool a probability of .0001. For instance, if MAX_NUMBER_SELECT is set to 4, then the probabilities will be 1.0, 0.75, 0.50, 0.25 and 0 for pools 1, 2, 3 and 4, respectively.

If a geometric progression is preferred (G), then the *a priori* probability of pool *n* is obtained by dividing the probability of pool *n-1* given that pool 1 has a probability of 1 and the last pool a probability of .0001. This latter is finally set to 0 for consistency.

MAX_RELATIVE_WEIGHT Final ratio between the faith factor of a module that would always fit and the faith factor of a module that would never fit once all trials have been performed.

SW_ROUNDS The number of attempted swaps in the minimization process.

This might depend on the maximum number of conformers in the initial pool (flag POOLSIZE).

3. Module parameters

Finally the parameters related to the different modules are specified. In this section, you first have to choose which modules you want to include in the calculations (i.e. for which you have experimental data). A value of 0 (zero) means no inclusion, while a value of 1 will take the

module into account. Some options are common between all modules: reference file, rank, starting faith and the scale up factor.

Most of these modules have the following common options:

FLAG_? (RANK)

The restraints can be included into the calculations in a non-simultaneous manner. Thus, it is possible to let the program try to fit some restraints before adding some others. As previously explained, the rank reflects the order of inclusion of restraints. The lower, the higher priority.

If the value of the flag is 0, then the module is not considered.

If the rank of ShiftX, SAXS and NOE are 1, 2 and 2, respectively, then ENSEMBLE will first try to fit the chemical shift data, ignoring the SAXS and NOE data. Once the fit is achieved, these latter are inserted into the restraints before starting the calculations again.

REFERENCE

This flag indicates the pathway of the file that contains the experimental data, which will be targeted by ENSEMBLE. See next section for the format of restraints.

The hydrodynamic radius does not need any restraint file. Instead, the target value is directly indicated in the parameter file with the flag HYDRO_RH.

Refer to MODULE RESTRAINTS in the next section to see how to prepare the restraint files.

EXEC

Location of the executable to predict the experimental observable based on the structural models.

This parameter is currently not used as the program makes use of only one predictor per module. However, future versions of ENSEMBLE could allow a choice between different predictors.

MODULE RESTRAINTS

ENSEMBLE can accommodate up to ten modules. The restraints attributed to each of them must follow a strict format readable by ENSEMBE. A line starting with '!' or '#' is a comment that is ignored. In the following examples, the first line must be commented if you want to include it in the restraint file.

1. Chemical Shift

Residue	Atom	Chemical Shift	Error
3	C	177.10	0.0
3	CA	52.43	0.0
3	CB	19.26	0.0
3	H	8.71	0.0
4	C	176.10	0.0

2. Residual Dipolar Coupling (RDC)

Residue 1	Atom 1	Residue 2	Atom 2	RDC
4	H	4	N	-2.0686
5	H	5	N	-0.1825
6	H	6	N	-0.3651
8	H	8	N	0.1826

3. Nuclear Overhauser effect (NOE)

The data extracted from a NOESY spectrum or PRE experiments can be interpreted in terms of inter-proton distances. In the table below, Atoms 1 of Residue 1 and Atom 2 of Residue 2 correspond to the two atoms involved in the restraint, Aver. indicates the mean distance, and Low and Up specify the distance range between below and above the average distance, respectively. The keyword 'OR' denotes ambiguous restraints and the wildcard '*' can be used.

Of note, ambiguous restraints (using the keyword 'OR') and wildcarded restraints are treated differently. With ambiguous restraints, the energy of each restraint is calculated and the lowest energy contribution is kept. When using a wildcard, the energy is calculated using the average position of all corresponding atoms.

Residue 1	Atom 1	Residue 2	Atom 2		Aver.	Low	Up
14	H	17	3HD2		8.0	8.0	0
17	1HD2	20	H	OR			
17	2HD2	20	H	OR			
17	3HD2	20	H		8.0	8.0	0
29	H	36	H		8.0	8.0	0
39	H	46	H		8.0	8.0	0

4. Paramagnetic Relaxation Enhancement (PRE)

PRE data are interpreted like NOE data, in terms of distance restraints. However, as they are issued from different experimental techniques, we keep these two modules as individual. The following table describes the way PRE restraints must be provided to ENSEMBLE (All labels are the same than for NOE data).

Residue 1	Atom 1	Residue 2	Atom 2	Aver.	Low	Up
2	CD	6	H	12.1	2.5	2.5
2	CD	8	H	14.7	2.5	2.5
59	OD1	38	H	11.2	11.2	2.5
59	OD1	43	H	13.5	2.5	2.5

5. PRE ratios (rPRE)

The ratios of R2 values yielded by PRE experiments are related to the ratios of distances (See reference [2] for more details). ENSEMBLE computes all possible ratios and this is the reason this approach can be rather time demanding. The way of providing restraints to the program is similar to the R2 restraints. The approach is quite useful when the tumbling time (τ_C) is unknown or cannot be determined.

Residue	Atom	R2
4	N	5.29162
5	N	6.22572
6	N	5.53679
7	N	9.53548
8	N	6.46877

6. R2

Residue	Atom	R2
4	N	5.29162
5	N	6.22572
6	N	5.53679
7	N	9.53548
8	N	6.46877

7. J-coupling

Res1	Atom1	Res2	Atom2	Res3	Atom3	Res4	Atom4	J	Low	Up
1	C	2	N	2	CA	2	C	5.97	0	0
2	C	3	N	3	CA	3	C	5.02	0	0
3	C	4	N	4	CA	4	C	7.02	0	0
4	C	5	N	5	CA	5	C	5.14	0	0

8. Solvent accessibility

Res	Atom		Group	Access	Low	Up	Asym	Weight
1	CA		1	1.178	.118	.118	1.0	1.0
2	N		1	0.000	0.00	0.00	1.0	1.0
3	N	AND						
3	HN	AND						
3	CA	AND						
3	HA1	AND						
3	HA2		2	74.021	7.402	7.402	1.0	1.0
3	RES		2	74.021	7.402	7.402	1.0	0.5

9. Hydrodynamic radius

For the hydrodynamic radius, the restraint value has to be indicated directly in the parameter file (flag HYDRO_RH).

10. Small Angle X-ray Scattering

r	Intensity	Error	Asym
0.030000	0.161292	0.0	1.0
0.035000	0.154920	0.0	1.0
:	:	:	:
0.205000	0.018079	0.0	1.0
0.210000	0.016948	0.0	1.0

RUNNING ENSEMBLE

1. Locally

Once the parameter and the module restraint files are ready, ENSEMBLE can be run. The program is accessible from any location of the system since the installation process created the link file in the /usr/bin/ensemble (if the user who installed it was the administrator) or added the pathway of the program into the PATH environment variable of the user. To run ENSEMBLE locally, simply type the following statement: ensemble parameter_file. The absolute or relative pathway of the parameter file can be specified. After each run, ENSEMBLE checks the remaining memory and will crash if there is not enough space.

2. On a cluster

ENSEMBLE has been designed to run on a single CPU. However, it is more efficient if it is launched on a cluster, as the generated structures will better sample space and the selection from the initial soup will produce different initial pools. Moreover, a computational cluster is more efficient for adjusting or testing different parameters. It is difficult to define a single way that would work with all possible clusters, as each cluster is governed by specific rules. Ask the administrator of the cluster in case you have problems. If the problem cannot be solved, please address your problem to us and we will do our best to help you.

Here, we provide a simple method you might try and adapt to your needs. First of all, make sure that the ENSEMBLE directory is accessible from each node of the cluster. Then, create a script that contains the important environment variables which have been set when installing the program on your system. Two examples are given in the ENSEMBLE directory, one in C-shell, the other one in bash. The following lines display the C-shell version:

```
#!/bin/csh

set job_nb = 0

foreach par_file (path_1/file_1.par path_2/file_2.par)
        @ job_nb += 1
        printf "#!/bin/csh\n" > submit_${job_nb}.job
        printf "setenv ENSEMBLE __ENSEMBLE_DIR__\n" >> submit_${job_nb}.job
```

```
        printf "$ENSEMBLE/ensemble $par file\n" >> submit_${job_nb}.job
        __SUBMIT_STATEMENT__ submit_${job_nb}.job
        Sleep 1
end
```

After the installation, the ENSEMBLE_DIRECTORY is automatically replaced by your ENSEMBLE directory. However, the "submit_statement" depends on the queuing system of the cluster you use and has to be manually modified.

3. After a crash

In the event of a crash, simply re-run ensemble with the same parameter file you used the first time you ran ENSEMBLE. The program will be able to retrieve the exact state since the last time it was saved since the parameter file you specify contains the pathway of the RUN directory where the Save directory is located. The program first checks whether the ensemble.dat file can be found. This latter contains all user-defined parameters and its presence means the other CSP and CDP files can be rapidly read and recorded.

RESULTS ANALYSIS

The software is provided with a number of programs to analyze the different ensembles calculated along the process, most interesting being the ensembles that fit all the available experimental data. These tools are found in the tools directory of the ensemble directory. During the installation, an environment variable has been automatically set so that you can directly access this directory from anywhere with $ENSEMBLE_TOOLS. An executable has been specifically designed in order to facilitate the full analysis of an ensemble: analyzens. This latter needs as argument the name of an ENSEMBLE file, which was saved in the Result directory. For instance, to analyze the best ensemble found so far, simply type:

$ENSEMBLE_TOOLS/analyzens /home/user/RUN_directory/Results/best.ens

Following this statement, a directory named BEST is created in the same directory as the best.ens file and all results will be stored in it. The analyzens script will output several files:

• analyze.txt	Contains information about the protein
• radiusGyration.txt	Contains the radius of gyration of each conformer of the ensemble as well as the average
• ss_distr.txt	Contains the secondary structure distribution of the ensemble
• caca_map.txt	Contains all averaged Cα-Cα distances of the ensemble
• caca_map.eps caca_map_stdv.eps	Graphical representation of the information contained in the caca_map.txt file.

Beside the main analyzing program analyzens, some tools have been designed to get simple information about an ensemble. The following list explains their role:

- rg list_file [-v]

This program displays the average radius of gyration (and standard deviation) of all conformers of the list file specified as argument. This latter can be a structural file or an ENSEMBLE file.
If the –v flag is specified, then the radius of gyration of each conformer will also be indicated.

- extract_csp –f CSP_file [-p pathway] [–x prefix] [–l 1 [2 3 8-10]]

This program allows extracting conformers from a CSP file. The optional flag –p indicates the directory (absolute or relative pathway) in which conformers will be extracted. The –x flag is optional and specifies the prefix used to generate the name of the extracted conformers. By default, "conf_" is used. Finally, the –l flag (optional) restrains the number of conformers to be extracted by specifying a list.
The first conformer of each CSP file starts at 1. If a number higher than the number of conformers in the CSP file is specified, then the program will issue a warning before stopping.

- make_csp –f list_file [-o output.csp] [–h]

This program acts in the opposite way of extract_csp by generating a CSP file from a list file. The –o flag is optional and allows specifying the name of the output CSP file. A relative or absolute pathway can be specified. By default, the name of the output file is the name of the list_file followed by the extension ".csp". The –h flag displays some help. All specified conformer of the list file must exist and atoms must have the same order. Otherwise, the program will issue a warning before stopping.

- get_info ENSEMBLE_file [ENSEMBLE_file…]

This program provides the characteristics of any file generated by ENSEMBLE. For instance, if a CSP file is given as argument, get_info yields the number of conformers and atoms that it contains.

- pdb2seq structural_file

This tool display the one-letter sequence of a structural file given as argument.

- read_best best.dat

The best.dat file, found in the Results of the RUN directory, contains all best ensembles found along the calculations. This tool reads best.dat and displays the evolution of the best ensembles.

- read_ens ENSEMBLE_file

This tool displays the conformer numbers found in an ENSEMBLE file given as arguments, as well as its energy.

- ss_distr list_file

By making use of STRIDE, this tool displays the secondary structure distribution of a set of conformers. The input can be a PDB list file or an ENSEMBLE file.

Some other analysis tools will be provided in the future.

REFERENCES

(1) Choy, W. Y.; Forman-Kay, J. D. *J Mol Biol* **2001**, *308*, 1011-32.

(2) Marsh, J. A.; Neale, C.; Jack, F. E.; Choy, W. Y.; Lee, A. Y.; Crowhurst, K. A.; Forman-Kay, J. D. *J Mol Biol* **2007**, *367*, 1494-510.

(3) Marsh, J. A.; Forman-Kay, J. D. *J Mol Biol* **2009**, *391*, 359-74.

(4) Marsh, J. A.; Forman-Kay, J. D. *Proteins* **2011**.

(5) Mittag, T.; Marsh, J.; Grishaev, A.; Orlicky, S.; Lin, H.; Sicheri, F.; Tyers, M.; Forman-Kay, J. D. *Structure* **2010**, *18*, 494-506.

(6) Marsh, J. A.; Dancheck, B.; Ragusa, M. J.; Allaire, M.; Forman-Kay, J. D.; Peti, W. *Structure* **2010**, *18*, 1094-103.

(7) Pinheiro, A. S.; Marsh, J. A.; Forman-Kay, J. D.; Peti, W. *J Am Chem Soc*, *133*, 73-80.

(8) Neal, S.; Nip, A. M.; Zhang, H.; Wishart, D. S. *J Biomol NMR* **2003**, *26*, 215-40.

(9) Marsh, J. A.; Baker, J. M.; Tollinger, M.; Forman-Kay, J. D. *J Am Chem Soc* **2008**, *130*, 7804-5.

(10) Lee, B.; Richards, F. M. *J Mol Biol* **1971**, *55*, 379-400.

(11) Garcia De La Torre, J.; Huertas, M. L.; Carrasco, B. *Biophys J* **2000**, *78*, 719-30.

(12) Svergun, D.; Barberato, C.; Koch, M. H. J. *Journal of Applied Crystallography* **1995**, *28*, 768-773.

(13) Feldman, H. J.; Hogue, C. W. *Proteins* **2000**, *39*, 112-31.