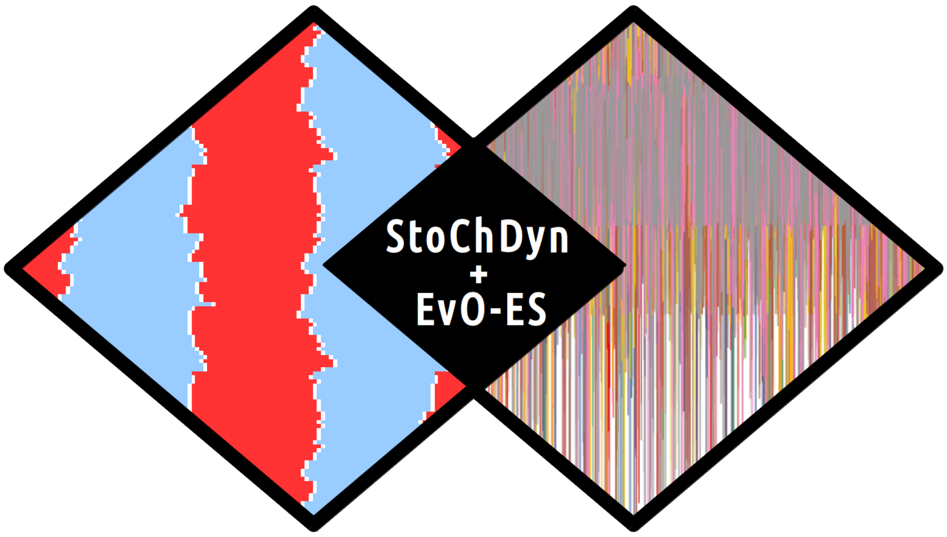
StoChDyn and EvO-ES - Stochastic modeling of chromatin state dynamics across cell divisions and evolutionary optimization of epigenetic stability



Documentation

Christian Arnold

June 2013

This document provides documentation for the `StoChDyn` and the `EvO-ES` software. The newest version of the software is available at the following URL:
`http://bioinf.uni-leipzig.de/Software/StoChDyn`

If you have questions or comments, feel free to contact me, Christian Arnold (achristian@bioinf.uni-leipzig.de). I will be happy to answer any questions related to this project or the implementation.

If you use this software, please cite the following reference:

Arnold, C., Stadler, PF, and Prohaska, SJ. 2013. Chromatin Computation: Epigenetic Inheritance as a Pattern Reconstruction Problem.*Journal of Theoretical Biology.* In review.

Contents

1 Description

The following abstract summarizes the publication that accompanies this software:

Eukaryotic histones carry a diverse set of specific chemical modifications that accumulate over the life-time of a cell and have a crucial impact on the cell state in general and the transcriptional program in particular. Replication constitutes a dramatic disruption of the chromatin states that effectively amounts to a partial erasure of the stored information. To preserve its epigenetic state the cell reconstructs (at least part of) the histone modifications by means of processes that are still very poorly understood. A plausible hypothesis proposes that the different combinations of reader and writer domains in histone-modifying enzymes implement local rewriting rules that are capable of "recomputing" the desired parental modification patterns on the basis of the partial information contained in that half of the nucleosomes that predate replication.

To test whether such a mechanism is indeed theoretically feasible we have developed a flexible stochastic simulation system for studying the dynamics of histone modification states. The implementation is based on Gillespie's approach, i.e., it models the master equation of a detailed chemical model. It is efficient enough to search by means of an evolutionary algorithm to find mixtures of combined reader/writer enzymes that are capable of propagating, with high accuracy, complex modification patterns across multiple cell divisions.

We found that it is relatively easy to evolve a system of enzymes that can stably maintain a particular chromatin state, even without explicit boundary elements separating differentially modified chromatin domains. However, the difficulty of this task depends on multiple previously unanticipated factors, such as the length of the initial state, the specific pattern that should be maintained, the time between replications, and chemical parameters such as binding and dissociation rates of the enzymes.

2 Installation

So far, the program has only been tested on a Linux-based system. We therefore cannot guarantee that the program can be compiled on computers using a Windows or MAC operating system, although the code should be principally portable. Perl and *R* must be installed, and a C compiler is also required. . In what follows, we describe the installation procedure for a Linux-based system.

An actual installation for the `StoChDyn` and the `EvO-ES` software is not necessary. Simply do the following:

1. Copy the root directory of the source file that you can download into a folder of your choice
2. Open a console, navigate to the *src* folder, and type `make` to generate an executable file for the stochastic simulation. This should not produce any errors. If it does, contact the author of the program (see Section 7. If compiled successfully, the executable file should automatically be copied to the *bin* subdirectory and is called `runSimulation`. Thus, the presence of that particular file indicates that the compilation was successful.
3. The evolutionary algorithm (`EvO-ES`) requires a particular Perl module (`Clone`) that may not be already available on your system. To install it, follow the instructions in the README in the `lib/Clone-0.31` folder.

After installing, two different programs are of relevance:

- An individual stochastic simulation (`StoChDyn`). For more details, see section 3.
- The evolutionary algorithm (`EvO-ES`). For more details, see section 4.

3 Running an individual stochastic simulation

3.1 Configuration file

There are two main options to provide the required parameters when running an individual stochastic simulation: The configuration file and, optionally, the command-line parameters. If additional command-line parameters are used except the -c and the -o option, the command-line parameters override the values of the corresponding parameters in the configuration file.

3.1.1 Usage of the configuration file

The configuration file is a central component of the stochastic simulation. All required parameters are specified in the configuration file. The syntax is similar to an XML-like structure, with tags and arguments within these tags for each parameter.

If a particular line in the configuration file starts with a "#" character, the line is ignored and not parsed.

In what follows, we give a summary of all currently supported parameters that may be specified in the configuration file. The parameters are sorted by sections for clarity, in analogy to the configuration file.

Nucleosome structure

- `histoneModifications`
  1. Parameter description: This tag specifies the histone modification(s) and the valid states for each modification. This is a nested tag, and each distinct histone modification must be described in its own separate tag. Note that currently, only one modification is supported.
  2. Supported types: An arbitrary name for each distinct state. Each value must be associated with one type description, separated by |. The unmodified state must have the symbol 0; otherwise, there are no restrictions.
  3. Can be disabled: Individual rules within a `histoneModifications` tag yes, the whole `histoneModifications` tag no.
  4. Supported values: All possible values that the particular modification that is modelled can have. Values must be separated by |. Each individual state must have only one and only one character.
  5. Example:
     ```
     <par name="histoneModifications">
     <par name="H3K12" type="unmodified|methylated|acetylated" enabled="1" value="0|1|2"/>
     </par>
     ```

Spatial organization and neighborhood

- `nNucleosomes`
  1. Parameter description: The number of nucleosomes that should be modelled.
  2. Supported types: none
  3. Can be disabled: no
  4. Supported values: an integer > 0.
  5. Example: `<par name="nNucleosomes" value="16"/>`
- `cyclicChromosomes`
  1. Parameter description: The nucleosome string can either be treated as linear or cyclic, which may have an effect for the boundary nucleosomes, depending on the rule set. In the former case, for example, rules that require the state of both neighboring nucleosomes may not be able to match the boundary nucleosomes. In the latter case, the two boundary nucleosomes are directly connected.
  2. Supported types: none
  3. Can be disabled: no
  4. Supported values: 0|1
  5. Example: `<par name="cyclicChromosomes" value="1"/>`

Simulation-specific parameters

- `initialState`
  1. Parameter description: This parameter specifies the starting state of the system.
  2. Supported types: equal |twoParts |threeParts |fourParts |fiveParts |sixParts |custom
  3. Can be disabled: no
  4. Supported values: For `equal` a single character, for `twoParts` to `sixParts` two to six characters that define the state of the block, separated by "|".

5. Example:
`<par name="initialState">`
`<par name="H3K12" type="threeParts" value="1|2|1"/>`
`</par>` or
`<par name="initialState">`
`<par name="H3K12" type="custom" value="123212212212"/>` (if nNucleosomes=12)
`</par>`

- `simulationAlgorithm`
  1. Parameter description: The specific method of the stochastic Gillespie algorithm. Currently, only "orig" is supported, which executes the direct method, as proposed by Gillespie 1977.
  2. Supported types: none
  3. Can be disabled: no
  4. Supported values: "orig"
  5. Example: `<par name="simulationAlgorithm" value="orig"/>`

- `stopCriterion`
  1. Parameter description: When should the simulation be stopped? Two types of stopping criteria are currently supported, which boils down to the question whether DNA replication should be included in the model: "afterAllReplications" stops the simulations after the specified number of replications (see parameter `nReplications` or after a fixed amount of time (as specified with this parameter).
  2. Supported types: afterAllReplications |fixed
  3. Can be disabled: no
  4. Supported values: Value is ignored when type equals "afterAllReplications". For "fixed", any floating point value > 0 is valid.
  5. Example:
     `<par name="stopCriterion" type="afterAllReplications" value=""/>`
     `<par name="stopCriterion" type="fixed" value="30"/>`

- `nSimulations`
  1. Parameter description: The number of independent Gillespie realizations that should be performed.
  2. Supported types: none
  3. Can be disabled: no
  4. Supported values: any positive integer value
  5. Example: `<par name="nSimulations" value="1"/>`

- `visualizationOutput`
  1. Parameter description: This parameter specifies if the state of the system should be visualized. Three values are supported: If set to 0, no state visualization will be created. If set to 1, only a summary visualization will be created (the state of the system before and after each DNA replication). If set to 2, in addition to the summary visualization, a detailed state visualization is created that also shows the dynamics between two consecutive DNA replications. Note: The size of the detailed visualization files can be quite large and therefore memory-consuming. Make sure you have enough disk space available. As a rule of thumb, reserve at least 10 Mb of disk space for each independent run if `visualizationOutput` is set to 2.
  2. Supported types: none
  3. Can be disabled: no
  4. Supported values: 0 |1 |2
  5. Example: `<par name="visualizationOutput" value="1"/>`

- `deleteContentOfOutputDir`
  1. Parameter description: This parameter controls the behaviour if the output directory does already exist. If set to 0, the program aborts to prevent potential data loss. Please delete the directory manually in that case. If set to 1, the program will continue, but potentially overwriting existing files (i.e., files that are created during the simulation) without further notice. If set to 2, the output directory will be deleted first (including all subdirectories, so be careful) and then re-created.

2. Supported types: none
3. Can be disabled: no
4. Supported values:0 |1 |2
5. Example: `<par name="deleteContentOfOutputDir" value="1"/>`

- `verbose`
  1. Parameter description: Enable detailed output? Should only be enabled for debugging purposes. Note: A lot of output will be produced, which may substantially increase the running time.
  2. Supported types: none
  3. Can be disabled: no
  4. Supported values: 0 or 1
  5. Example: `<par name="verbose" value="0"/>`

- `seed`
  1. Parameter description: The seed is needed for the random number generator. If set to 0 or disabled, a seed is taken either from /dev/urandom (if existing) or from the current time (as a fallback). If a fixed seed is used, the random numbers are identical in each run (mainly for debugging purposes)
  2. Supported types: none
  3. Can be disabled: yes
  4. Supported values: a floating point value
  5. Example: `<par name="seed" enabled="1" value="0"/>`

Enzymes and reactions

In this section, the enzymes and their rules are defined. Each enzyme must be specified with an individual `<par name="enzymeDef"> ... </par>` tag. Within this tag, the following additional tags must be specified:

- `name`
  1. Parameter description: The name of the enzyme (arbitrary).
  2. Supported types: none
  3. Can be disabled: yes
  4. Supported values: an arbitrary name
  5. Example: `<par name="name" enabled="0" value="0[0]0:020"/>`

- `size`
  1. Parameter description: The binding size of the enzyme. The value of this parameter must match with the
  2. Supported types: "symmetric"
  3. Can be disabled: no
  4. Supported values: an integer > 0.
  5. Example: `<par name="size" type="symmetric" value="1"/>`

- `concentration`
  1. Parameter description: The concentration (number of enzymes) for that particular enzyme.
  2. Supported types: "absolute"
  3. Can be disabled: no
  4. Supported values: an integer ≥ 0
  5. Example: `<par name="concentration" type="absolute" value="0"/>`

- `bindingRatesRuleSet`

1. Parameter description: This tag specifies the rewriting rules for the enzyme. Each rule has to be specified using an individual tag within the `bindingRatesRuleSet` tag. For each enzyme, an arbitrary number of rules can be specified. At least one rule per enzyme must be active/specified. The `name` tag for each rule can be an arbitrary description.
2. Supported types: So far, type must be ”explicitRate”.
3. Can be disabled: Individual rules within a `bindingRatesRuleSet` tag yes, the whole `bindingRatesRuleSet` tag no.
4. Supported values: A valid rule specification, consisting of three parts: First, the required states of the left neighbors (optionally), the required states of the binding site (within a [ ] pair. The binding size must match the size of the enzyme, as specified in the `size` parameter of the corresponding enzyme), and the required states of the right neighbors (optionally). The second part specifies the rate (any floating point value > 0), and the third part the right part of the rule specification. Note that in the right part, the [ ] pair is not neccessary. The three parts must be separated by ”:”. See examples below. Note that all state symbols must be specified as values in the `histoneModifications` parameter. Furthermore, each state must have only one character.
5. Example:
```
<par name="bindingRatesRuleSet">
<par name="0[0]0:020" type="explicitRate" enabled="1" value="0[0]0:1:020"/>
<par name="2[0]0:220" type="explicitRate" enabled="0" value="22[0]0:1:2220"/>
<par name="[0]:2" type="explicitRate" enabled="1" value="[0]:1:2"/>
</par>
```

- `dissociationRate`

1. Parameter description: The rate of the dissociation reaction after the enzyme bound to a particular position.
2. Supported types: absolute
3. Can be disabled: no
4. Supported values: a floating point value > 0
5. Example: `<par name="dissociationRate" type="absolute" value="5"/>`

DNA replication and nucleosome dynamics

- `nReplications`

1. Parameter description: The number of DNA repliations that should be done. If set to 0, no DNA replication is done.
2. Supported types: none
3. Can be disabled: Yes
4. Supported values: an integer ≥ 0.
5. Example:
```
<par name="nReplications" enabled="1" value="25"/>
```

- `replicationFrequency`

1. Parameter description: After which fixed time increments should the replication be done?
2. Supported types: Currently, only ”fixedTime” is supported.
3. Can be disabled: No
4. Supported values: a floating point value > 0.
5. Example:
```
<par name="replicationFrequency" type="fixedTime" value="5"/>
```

- `replicationModel`

1. Parameter description: The specific replication model. Currently, only the random model is available, where histones/nucleosomes are randomly distributed between the leading and lagging strand. Thus, each nucleosome position has a probability of 50% to maintain its parental state.
2. Supported types: Currently, only "random" is supported.
3. Can be disabled: No
4. Supported values: Value is ignored when type equals "random"
5. Example:
   `<par name="replicationModel" type="random" enabled="1" value=""/>`

Phases

- `phaseDefinition`

  1. Parameter description: This tag can be used to define different phases (e.g., think of the different phases after a DNA replication), each of which may have a different enzyme availability. For example, if l phases are defined, then the simulation starts with the first phase and uses only the enzymes that are defined to be available in that particular phase. Then, after some time, the next phase starts, and finally, after phase l, a DNA replication event takes place. To disable phases, use the first line (and only the first line) of the 2 phase example below.
  2. Supported types: none
  3. Can be disabled: Individual phases yes, but at least one phase must be defined.
  4. Supported values: The number of the phase, followed by ":", followed by the names of all enzymes that should be active in this particular phase, separated by comma. The names of the enzymes here must match the enzyme names as specified in the enzyme definition and the enzyme must also be enabled. If all enzymes should be active, the abbreviation "all" can be used. The number must be an integer >0).
  5. Examples:
     ```
     <par name="phaseDefinition" enabled="1" value="1:all"/>
     <par name="phaseDefinition" enabled="1" value="2:HAT,HMT"/>
     <par name="phaseDefinition" enabled="1" value="3:HAT"/>
     ```

- `phaseDuration`

  1. Parameter description: This tag specifies the individual duration of each phase, specified as a percentage based on the full cycle length as specified in the parameter `replicationFrequency`. It is thus possible to define phases of varying length.
  2. Supported types: "equal" and "custom" are supported.
  3. Can be disabled: No
  4. Supported values: If the type is "equal", the value is ignored. If the type is "custom", a comma-separated list of phase durations (interpreted as percentages of the whole cycle, as specified in the parameter `replicationFrequency`) must be specified that sum up to 100. The same number of durations must be specified as phases have been defined in the `phaseDefinition` tag. Values must be at least 1.
  5. Examples:
     ```
     <par name="phaseDuration" type="equal" value="equal"/>
     <par name="phaseDuration" type="custom" value="20,60,20"/>
     ```

3.1.2 Usage of the command-line parameters

Selected parameters can be specified using command-line arguments. If specified, they override the values of the configuration file. The list may be extended in the future and currently contains only some of the more frequently used parameters. The only two command-line arguments that are required when running the stochastic simulation are the -c option (unless the configuration file has the default name "configFile"). In what follows, we give a list of the currently supported command-line arguments and the names of the corresponding parameters (see section above).

- -c (path to the configuration file where all the required parameters are specified, MANDATORY unless the configuration file is located in the current directory and is called "configFile")

- -o (`outputDirectory`. The default output directory "simOut" (within the directory where the configuration file is located) will be used if the user does not provide this option. WARNING: If the output folder already exists, then the parameter `deleteContentOfOutputDir` controls how to proceed. See the parameter description for `deleteContentOfOutputDir` for more details.)
- -m (`nNucleosomes`)
- -i (`initialState`)
- -n (`nSimulations`)
- -v (`verbose`)
- -e (`cyclicChromosomes`)
- -r (`nReplications`)
- -f (`replicationFrequency`)

For example, to override certain parameters from the configuration file, one may type:

```
./runSimulation -c "configFileTest" -v 0 -o "OUT" -n 5 -r 10 -m 50 -f 2 -e 1
```

4 Running the evolutionary algorithm

4.1 Methodological details

Methodological details for the evolutionary algorithm can be found in the publication.
In each iteration of the evolutionary algorithm and for each of the independent runs, a new solution is proposed by modifying the current best solution of that particular run by randomly selecting one of seven different "moves", six of which are mutations from one particular solution and one of which is a recombination (cross-over) of two solutions.

- `moveEnzymesLocal`: Decreasing the enzyme concentration c_{e_1} of one randomly selected enzyme e_1 by up to n_s molecules, and adding these molecules to the enzyme concentration of a second randomly selected enzyme e_2. If $c_{e_1} = 0$ after this move, the enzyme is deleted, and therefore, this move may decrease the number of available enzymes by one.
- `moveEnzymesSemiglobal`: Replacing n_s of the available enzymes with randomly selected new enzymes (their original enzyme concentrations, rate constants for both binding and dissociation reactions and phase availabilities were not changed).
- `moveEnzymesGlobal`: Combining two of the current best solutions s_i and s_j from two randomly selected runs i and j using a random linearization factor r_l. For each enzyme from s_i and s_j, the new concentration was calculated using the formula $s_{i+j} = r_l * s_i + (1 - r_l) * s_j$. Thus, the new solution may increase the number of available enzymes. To ensure that the same number of enzymes was present after this recombination, the enzymes with the lowest concentrations were stepwise eliminated until the original number of enzymes was restored. [Details how the phase availabilities are calculated are missing here]
- `moveRatesBinding`: Modifying the rate constants for the binding reactions (either be doubling the original rate or dividing it by 2).
- `moveRatesDissociation`: Modifying the rate constants for the dissociation reactions (either be doubling the original rate or dividing it by 2).
- `movePhaseAvailability`: Modifying the availability of the enzymes for the different phases. If case the random selection procedure deactivated all available enzymes for a particular phase, one enzyme was randomly re-activated to prevent that no enzyme is available in that phase.
- `movePhaseDuration`: Modifying the individual phase durations (in 10 per cent steps with regard to the full length of one replication cycle).

Note that each move can be disabled by putting a "#" character in front of the line where the moves are defined. In the example that follows, two moves have been disabled, and thus, the evolutionary algorithm will select among five moves. It is also possible and sometimes even reasonable to include moves multiple times (e.g., see `movePhaseAvailability`) to increase the probability that they are selected. This is useful for example, if it is expected that a particular move has an important influence on the solution and therefore, selecting a move more often than others may lead to better solutions in less time.

It is also possible to implement new moves. For more details, contact us.

```
my @moves = (

\&moveEnzymesLocal,
\&moveEnzymesSemiglobal,
\&moveEnzymesGlobal,
#\&moveRatesBinding,
#\&moveRatesDissociation,
\&movePhaseAvailability,
\&movePhaseAvailability,
\&movePhaseDuration
);
```

We found that if the moves for the binding and dissociation rates are selected, the solutions are mostly equally good, but they 1) may contain strange enzymes and 2) are much more sensitive to the specific pattern and the pattern length in particular. Thus, we recommend to not include these moves in real applications.

4.2 Excuting the script

Before running the evolutionary algorithm, a configuration file has to be produced, analogous to run an individual stochastic simulation. In fact, there is no difference to a configuration file that is used for an individual stochastic simulation, except that you have to define a list of enzymes that the evolutionary algorithm may select from during its search. Keep in mind to set the `enabled` tag to 1 in the `name` parameter for the enzyme if it should be included in the list of enzymes to select from.

One important parameter is `$continueFromExistingSim`. If set to 0, an existing evolutionary optimization in the target folder will be deleted and the evolutionary algorithm will start anew. If set to 1, the script will attempt to automatically determine where the last simulation stopped and continue from there on until the desired number of iterations have been reached ()as specified in `$maxIterations`).

Another important parameter in the script is `$randomlyInitialize`, which is only relevant if `$continueFromExistingSim` is set to 0. This parameter controls if a random initial solution will be constructed (`$randomlyInitialize=1`) or if the initial solution should be based exactly on the values from the template configuration file (`$randomlyInitialize=0`).

If `$randomlyInitialize` is set to 1, the initial solution will be constructed as follows:

- a random mix of k enzymes out of all defined enzymes (based on the parameter k = `$nEnzymesMaxActivatedMax`)
- random concentrations for these k enzymes, so that the sum of all concentrations equals `$sumAllEnzymesTotal`
- all binding and dissociation rates are initialized with 1 and therefore have equal rates at the beginning
- all k enzymes are available in each phase
- all phases have an equal duration (i.e., the time between two DNA replications divided by the number of phases)

Furthermore, a few of the parameters that can be specified in the configuration file are in fact also specified in the `startEvolAlgNew.pl` script. As the latter overrides the values from the configuration file, make sure to specify the parameter value of your choice in the Perl script and not the configuration file.

If the user sets the `$randomlyInitialize` parameter to 0, however, all these values will be directly taken from the configuration template file, and therefore, the initial solutions for all independent runs will be identical and all subsequent variations are based on the particular configuration as specified in the configuration file.

The actual evolutionary algorithm for a specific template configuration file is then started by simply typing the command: `perl startEvolAlgNew.pl`. No command-line parameters are necessary, they all have to be specified in the script itself. See the script itself for parameter details. Currently, the binary executable (`runSimulation`) must be located in the same folder where the script `startEvolAlgNew.pl` is located. You can also copy add the `bin` folder to your include path to execute the scripts from arbitrary locations.

The script writes all its output into a log file that is typically located within the `sim` directory within the specified output directory.

5 Examples

We included two example simulations that can be directly executed. In the examples folder, there are two subdirectories:

- `individualSimulation`: This directory contains an individual example configuration file that can be executed using the following command (execute this from the directory where the `runSimulation` executable is located): `./runSimulation -c 'PATH/example/individualSimulation/configFile'`
- `evolutionaryAlgorithm`: This directory contains an example configuration file that can be used for the evolutionary algorithm (execute this from the directory where the `runSimulation` executable is located): `perl startEvolAlgNew.pl`. The `$simRootFolder` variable in the script is initially set to the path of the example file for the evolutionary algorithm.

6 Copyright

This software is published under the GNU Public License. For more details, see http://gnu.org/licenses/gpl.html or the file GNU Public License.txt in the src folder.

7 Bugs and Feature Requests

Please report any bug that you encounter as well as any feature request that you may have to Christian Arnold (achristian@bioinf.uni-leipzig.de).