How to extend TreeTime

Lin Himmelmann
www.linhi.com

Dirk Metzler
www.zi.biologie.uni-muenchen.de/evol/StatGen.html

March 18, 2009

TreeTime has been developed by Lin Himmelmann as part of his PhD thesis in the group of Dirk Metzler. It is freely available from
www.zi.biologie.uni-muenchen.de/evol/statgen/software/treetime
under the terms of the GPL.

The modular design of TreeTime enhances the extensibility of the software. Its core is the implementation of the Metropolis-coupled Markov Chain Monte Carlo (MCMCMC) algorithm. Following the template method[1] it delegates the central tasks to abstract classes. The concrete classes are then defined in the namespace `TreeTime_Core`. An example for this is the class `StateSampler`, which is accessed through the interface of the abstract class. When an alternative method for the sampling is needed, it must be implemented as an new class that uses the interface of the abstract class.

A `State` in the module `TreeTime_Core` is a composite consisting of (pointers to) `Tree`, `DnaEvolutionModel`, `RateChangeModel` and `TopologicalConstraint`.

Implementing Sequence Evolution Models

Sequence evolution models are accessed exclusively via the interface `DnaEvolutionModel`. New sequence evolution models must be derived from this base class, i.e. its virtual member functions have to be implemented. All sequence evolution models are instantiated by the static *factory method* `newDnaEvolutionModel(string p_stringRepresentation)` in

[1] cf. E. Gamma, R. Helm, R. Johnson, J. Vlissides (1995) *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley.

the class `DnaEvolutionModel`. The string `p_stringRepresentation` must have a prefix that identifies the model and is followed by a pair of round brackets containing the model parameters. When new sequence evolution models are added, the selection of models in the factory method in class `DnaEvolutionModel` must be extended. The prefix of `p_stringRepresentation` can then be used in the Nexus input file to select the new sequence evolution model. New sequence evolution models can readily be combined with the assumption of invariant sites or gamma distributed rate categories.

For example, in the implementation of the general time reversible (GTR) model a class `GtrModel` is defined in the namespace `TreeTime_MolecularEvolution`. The new class is derived from the base class `DnaEvolutionModel` and implements its interface. The implementation can be found in the files "GtrModel.hpp" and "GtrModel.cpp" in the directory "TreeTime_MolecularEvolution/DnaModels", which contains the source code of all sequence evolution models. The following lines from the file "DnaEvolutionModel.cpp" show how the GTR model is added to the factory method `newDnaEvolutionModel` in class `DnaEvolutionModel`.

```
01 DnaEvolutionModel* newDnaEvolutionModel(
02       string p_stringRepresentation)
03 {
04    string model = p_stringRepresentation.substr(0,
05          p_stringRepresentation.find_first_of("+("));
06    bool hasInvariantSites = false;
07    bool hasGammaRates     = false;
08    int nGammaRates        = 1;
09    vector<double> par;
10    /* Parsing of p_stringRepresentation
11    ... */
12    if(model=="JC")
13    {
14       return new JukesCantorModel(hasInvariantSites,
15           propOfInvariantSites, nGammaRates,
16           gammaShapeParameter);
17    }
18    else if(model=="GTR")
19    {
20       return new GtrModel(
21             // stationary Distribution Parameters:
22             par[0], par[1], par[2], par[3],
```

```
23              // Rate Parameters
24              par[4], par[5], par[6], par[7], par[8], par[9],
25              // additional Parameters
26              hasInvariantSites, propOfInvariantSites,
27              nGammaRates, gammaShapeParameter );
28    }
29    else
30    {
31       return new JukesCantorModel(false, 0.0, 1, 1.0);
32    }
33 }
```

The method returns a pointer to the sequence evolution model that is determined by the prefix of the input string. In the original file lines 10 and 11 are replaced by code that extracts the parameters values from the input string and stores them in the vector par. Only lines 18 to 28 had to be added for the implementation of the GTR model. The class `GtrModel` was made accessible from the source code file "DnaEvolutionModel.cpp" by adding the line `#include ''DnaModels/GtrModel.hpp''`.

The base class `DnaEvolutionModel` contains four interfaces for functions that change parameters of sequence evolution models during the MCMC procedure. The functions `changeInvariantSites()` and `changeGammaRates()` change the proportion of invariant sites and the distribution of rate categories along the sequence, respectively. These changes are implemented in the base class `DnaEvolutionModel`. The functions `changeStationaryDistribution()` and `changeTransitionParameters()` must be implemented separately for each sequence evolution model.

Implementing Rate Change Models

Relaxed molecular clock models, i.e. models for the change of mutation rates along the phylogenetic tree, are accessed exclusively via the interface `RateChangeModel`. New rate change models must be derived from the base class `RateChangeModel` and its virtual member functions have to be implemented. All rate change models are instantiated by the static *factory method* `newRateChangeModel(string string p_stringRepresentation)` in the class `RateChangeModel`. The string `p_stringRepresentation` must consist of a prefix that identifies the model and is followed by a pair of round brackets containing the model parameters. When new rate change models are added, the selection of models in the factory method in class `RateChangeModel` must be extended. The prefix of

p_stringRepresentation can then be used in the Nexus input file to select the new rate change model. If the new rate change model requires additional data for internal nodes, an additional associative container map<int key,DATA> can be added to the class NodeData. The key specifies the locus to which the model applies. (For key= -1 the model is applied to all loci.)

To extend TreeTime with the Dirichlet rate change model for example, only changes in the the module TreeTime_RateChangeModel are needed. Since this model does not require additional data for internal nodes, it is not necessary to add an additional data container to the class NodeData. The new class DirichletModel is defined in the namespace TreeTime_RateChangeModel. The source code files "DirichletModel.hpp" and "DirichletModel.cxx" are placed in the directory "TreeTime_RateChangeModel". The source code for changes in the rate change model during the MCMC procedure are stored in the subdirectory "Changes". For the Dirichlet model the change classes DmChangeAllRates, DmChangeTwoRates and DmChangeVariance are implemented according to the interface of the class Change in the namespace TreeTime_Mcmcmc. All these classes are accessed via the interface getChange. The following lines from the file "RateChangeModel.cxx" show how the Dirichlet model (DM) is added to the factory method newRateChangeModel in class RateChangeModel.

```
01 template<class STATE>
02 RateChangeModel<STATE>* RateChangeModel<STATE>::
03       newRateChangeModel(string p_stringRepresentation)
04 {
05   if(p_stringRepresentation == "")
06   {
07      return new MolecularClock<STATE>();
08   }
09   string model = p_stringRepresentation.substr(0,
10         p_stringRepresentation.find_first_of("("));
11   vector<double> par(7,1.0);
12   /* Parsing of p_stringRepresentation
13   ... */
14   if(model=="CPP")
15   {
16      return new CompoundPoissonProcess<STATE>(par[0], par[1],
17            par[2], par[3], par[4], par[5], par[6]);
18   }
19   else if(model=="DM")
```

```
20   {
21      return new DirichletModel<STATE>(par[0], par[1],
22            static_cast<int>(par[2]), par[3], par[4], par[5]);
23   }
24   else if(model=="ULN")
25   {
26      return new UncorrelatedLognormal<STATE>(par[0], par[1],
27            static_cast<int>(par[2]), par[3], par[4], par[5]);
28   }
29   else if(model=="UEX")
30   {
31      return new UncorrelatedExponential<STATE>(
32            static_cast<int>(par[0]), par[1], par[2]);
33   }
34   else
35   {
36      return new MolecularClock<STATE>();
37   }
38 }
```

In lines 09-10 the identifier of the model (DM in the case of the Dirichlet model) is extracted from the input string. In the original file, lines 11-13 are replaced by code that reads the model parameters from the input string. Lines 19-23 are the lines that were added for the DM model. The factory function returns a pointer to the newly added model. Finally, the class `DirichletModel` is made accessible in "RateChangeModel.cxx" by the include directive `#include ''DirichletModel.hpp''`.