marmoteCore: a software platform for Markov modeling

Alain Jean-Marie, Issam Rabhi

marmoteCore : a software platform for Markov modeling

Alain Jean-Marie, Issam Rabhi

Inria Sophia-Antipolis Méditerranée {Alain.Jean-Marie,Issam.Rabhi}@inria.fr

Mots-clés : *Stochastic Operations Research, Markov Chain, modeling software*

1 Introduction

We present the marmoteCore software, an open environment for modeling with Markov chains, developed within the MARMOTE project. [1] This platform aims at providing the general scientific user with tools for creating Markov models and computing, or simulating, their behavior. It provides a software library implementing both basic models and methods "from the book" and advanced solution methods : either general, or for specific classes of models. marmoteCore is devised to be a collaborative platform, able to embed solution methods and software developed by independent teams. We describe its architecture and some of its features, including its capability to communicate with established scientific software such as R.

The need for marmoteCore. Modeling with Markov chains is an activity common to many fields of science and engineering. Interacting particle systems of Physics, genome evolution models of Biology, epidemic models of Medicine, population models of Ecology, queueing systems of Operations Research, all those popular models are based on Markov chains. Monte-Carlo simulations of Markov chains are commonly used for producing samples of distributions of combinatorial objects, physical systems etc. This practical importance has prompted the development of many modeling software for specific areas : GreatSPN for Petri Nets, CADP and PRISM for Model Checking to name some of the most known, as well as demonstration packages in general modeling software such as scilab or R. On the other hand, there exists up to now no software environment providing the general scientific user with, at the same time, a collection of ready-to-use well-known models and general modeling constructions and solution methods, all accessible using an uniform programming interface. Realizing the prototype of such a platform is one of the purposes of the marmoteCore project.

2 Architecture

An object-oriented perspective. The objectives of the project make the choice of object-oriented programming almost obvious. Object-oriented languages give the possibility of designing high-level abstractions for mathematical concepts, representing objects with common properties. Yet, through the mechanism of inheritance, the user has the flexibility of controling the implementation of specific instances of the model. We illustrate this idea below. The organization of marmoteCore relies a lot on hierarchies of models, and polymorphism. We have chosen the C++ language, in part due to the fact that the legacy code available to us is written in C/C++.

Core objects. marmoteCore is based on four principal abstractions : markovChain, statSpace, transitionStructure and distribution.

The distribution class implements the probabilistic concepts that are underlying stochastic models in general, including the ubiquitous Bernoulli, Geometric and Exponential distributions.

1. MARMOTE (MARkovian MOdeling Tools and Environments), is a project funded by the French research agency, grant ANR-12-MONU-00019. See : https://wiki.inria.fr/MARMOTE/Welcome

The `transitionStructure` class is an abstraction for the labeled state-to-state transitions that are common to discrete-time and continuous-time Markov chains. Typical methods of this object include `get/setEntry()` accessors, and `evaluateMeasure()`/ `evaluateValue()` representing respectively right- and left- vector/multiplications in the language of linear algebra. This class can be implemented with regular two-dimensional arrays (full or sparse), but also with methods not based on the comprehensive storage of values but on symbolic manipulations. It is possible to represent this way processes on potentially infinite state spaces.

The `stateSpace` class implements discrete sets of states, with elementary objects such as integer intervals, and elementary constructions such as Cartesian products and unions. Typical methods such as `nextState()`, `index()` and `decodeState()` are convenient for traversing complex state spaces and constructing transition structures in an algorithmic way.

The hierarchy of Markov models. The fourth principal class, `markovChain`, is developped in a large variety of specific Markov models. The idea is that to each particular family of model, there can be specific implementation of data structures and, above all, of solution methods optimized for the particular family.

As an example, consider the following sequence of inclusions of well-known models in Markov modeling : `Poisson` $\subset$ `MMPP` $\subset$ `MMPP/M/1` $\subset$ `HomogeneousQBD` $\subset$ `QBD` $\subset$ `MarkovChain`. At the level of Poisson processes, the representation of the transition structure is reduced to one parameter, the arrival rate. Also, most metrics (e.g. transient probabilities, hitting times) are available in closed form. As we move up in the hierarchy, algorithms for computing the same metrics can be implemented. For instance : specific algorithms exist to compute level passage times in QBDs (Quasi-Birth-Death processes). At the highest level, the algorithms are applicable to any Markov chain, *a fortiori* to subclasses.

Solution methods. This object architecture offers a large flexibility for implementing and using solution algorithms. The practitioner, developing a model for a specific situation, has the choice between several algorithms for computing performance metrics and can choose the one that turns out to be the more efficient. The developper of new solution methods uses the existing ones for checking the validity of the new computations, then as benchmark for assessing their precision and speed. The current version of `marmoteCore` comes with generic algorithms for finding stationary distribution and average hitting times, and performing Monte-Carlo simulation for trajectories, stationary distributions and hitting times.

3 Integration with existing software

The architecture of MARMOTE has been devised to ease up the interaction with existing scienfic software. This can be done in two ways. First, MARMOTE can be made available to existing platforms in the form of "plugins" or "libraries". In the other direction, MARMOTE can make use of functionalities of such platforms through calls to their specific libraries.

The first possibility can used notably in the interaction with *Workflow Management Systems* and related graphical programming tools, so as to make easier the use of the software by scientists not fluent in `C++`.

To illustratethe second possibilty, we have developped an interface with the `markovchain` package of the scientific software `R`. [2] Functions from this package such as `is.irreducible()`, `is.accessible()` or `rmarkovchain()` are wrapped and provided as `C++` calls in the API of `marmoteCore`. Technically : using the `RInside` library of `R`, an execution engine for the `R` language is run inside `marmoteCore`. Markov models are passed to this engine, as well as calls to the wrapped package. We are currently wrapping other external packages such as Markov chain procedures in `scilab`, or the `Xborne` package developped at the Versailles Saint-Quentin-en-Yvelines University, and the `PSI3` package developped in the MESCAL team. [3]

2. `https://cran.rstudio.com/web/packages/markovchain/index.html`
3. `http://psi.gforge.inria.fr/dokuwiki/doku.php?id=psi3:start`