HAL open science

XBorne 2016: A Brief Introduction

Jean Michel Fourneau, Youssef Ait El Mahjoub, Franck Quessette, Dimitris Vekris

To cite this version:

Jean Michel Fourneau, Youssef Ait El Mahjoub, Franck Quessette, Dimitris Vekris. XBorne 2016: A Brief Introduction. Computer and Information Sciences, 659, Springer International Publishing, pp.134-141, 2016, Communications in Computer and Information Science, 10.1007/978-3-319-47217-1_15 . hal-03934357

HAL Id: hal-03934357

https://hal.science/hal-03934357v1

Submitted on 6 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

XBorne 2016: A Brief Introduction

Jean Michel Fourneau(✉), Youssef Ait El Mahjoub, Franck Quessette, and Dimitris Vekris

DAVID, UVSQ, Versailles, France
jean-michel.fourneau@uvsq.fr

Abstract. We present the new version of XBorne a software tool for the probabilistic modeling with Markov chains. The tool which has been developed initially as a testbed for the algorithmic stochastic comparisons of stochastic matrices and Markov chains, is now a general purpose framework which can be used for the Markovian modelling in education and research.

Keywords: Performance · Numerical analysis · Simulation · Markov chains

1 Introduction

The numerical analysis of Markov chains always deals with a tradeoff between complexity and accuracy. Therefore we need tools to compare the approaches, the codes and some well-defined examples to use as a testbed. After many years of development of exact or bounding algorithms for stochastic matrices, we have gathered the most efficient into XBorne, our numerical analysis tool [8]. Typically using XBorne, one can easily build models with tens of millions of states. Note that solving any questions with this size of models is a challenging issue. XBorne was developed with the following key ideas:

1. Build one software tool dedicated to only one function and let the tools communicate with file sharing
2. If another tool already exists for free and is sufficiently efficient, use it and write the export tool (only create tools you cannot find easily).
3. Allow to recompile the code to include new models.
4. Separate the data and the description of the data.

As a consequence, we have chosen to avoid the creation of a new modelling language. The models are written in C and included as a set of 4 functions to be compiled by the model generator. This aspect of the tool will be emphasized in Sect. 2 with the presentation of an example (a queue with hysteresis). The tool decomposition approach will also be illustrated in the paper.

XBorne is now a part of the French project MARMOTE which aims to build a set of tools for the analysis of Markovian models. It is based on PSI3 to perform perfect simulation (i.e. Coupling from the past) of monotone systems and

© The Author(s) 2016
T. Czachórski et al. (Eds.): ISCIS 2016, CCIS 659, pp. 134–141, 2016.
DOI: 10.1007/978-3-319-47217-1_15

their generalizations [5], MarmoteCore to provide an object interface to Markov objects and associated methods, and XBorne that we will present in this paper. The aim of XBorne (and the other tools developed in the MARMOTE project) is not to replace older modeling tools but to be included into a larger framework where we can share tools and models developed in well-specified frameworks which can be translated into one another. XBorne will be freely available upon request.

The technical part of the paper is as follows: in Sect. 2, we present how we can build a new model. We show in Sect. 3 how it can be solved and we present some numerical results. Sections 4 and 5 are devoted to two new solving techniques. In Sect. 4, we consider the quasi-lumpability technique. We modify the Tarjan and Paige approach used for the detection of macro-states for aggregation or bisimulation [12] to relax the assumption on the creation of macro states and accommodate a quasi-lumpable partition of the state space. Section 5 is devoted to the simulation of Markov chains and it is presented here to show how we have chosen to connect XBorne with other tools.

2 Building a Model with XBorne

XBorne can be used to generate a sparse matrix representation of a Discrete Time Markov Chain (DTMC) from a high level description provided in C. Continuous-time models can be considered after uniformization (see the example in the following). Like many other tools, the formalism used by XBorne is based on the description of the states and the transitions. All the information concerning the states and the transitions are provided by the modeler using 2 files (1 for the constants and one for the code, respectively denoted as "const.h" and "fun.c"). States belong to a hyper-rectangle the dimension of which is given by the constant NEt. The bounds of the hyper-rectangle must be given by function "InitEtendue()". The states belong to the hyper-rectangle and they are found by a BFS visit from an initial state given by the modeler through function "EtatInitial()".

The transitions are given in a similar manner. The constant "NbEvtsPossibles" is the number of events which provoke a transition. The idea is that an event is a mapping applied to a state (not necessarily a one to one mapping). Each event has a probability given by function "Probabilite()" and its value may depend on the state description. The mapping realized by an event is described by function "Equation()". To conclude, it is sufficient to describe 4 functions in C and some definitions and recompile the model generator to obtain a new code which builds the transition probability matrix.

```
#define NEt   2              #define NbEvtsPossibles   4
#define AlwaysOn   10        #define BufferSize  20
#define OnAndOff    5        #define UPandDOWN  0
#define WARMING     1        #define ALL_UP  2
#define UP         10        #define DOWN  5
```

We now present an example for the various definitions and functions which are written in the files "const.h" and "fun.c" to describe the model developed by Mitrani in [11] to study the tradeoff between energy consumption and quality of service in a data-center. It is a model of a M/M/(a+b) queue with hysteresis and impatience. We have slightly changed the assumptions as follows: the queue is finite with size "BufferSize". The arrivals still follow a Poisson process with rate "Lambda". The services are exponential with rate "Mu". Initially only "AlwaysOn" servers are available. Once the number of customers in the queue is larger than "UP", another set (with size OnAndOff) of servers is switched on. The switching time has an exponential duration with rate "Nu". If the number of customers becomes smaller than "DOWN", this set of servers is switched off. This action is immediate. As NEt=2, a state is a two dimension vector. The first dimension is the number of customers and the second dimension encodes the state of the servers. The initial state is an empty queue with the extra block of servers which is not activated.

```
void InitEtendue()
{
    Min[0] = 0;  Max[0] = BufferSize; Min[1] = UPandDOWN;  Max[1] = ALL_UP;
}
void EtatInitial(E)
int *E;
{
    E[0] = 0; E[1] = UPandDOWN;
}
double Probabilite(int indexevt, int *E) {
   double p1, Delta;
    int nbServer, inserv;
    nbServer = AlwaysOn;
    if (E[1]==ALL_UP) {nbServer += OnAndOff;}
    inserv = min(E[0], nbServer);
    Delta = Lambda + Nu + Mu*(AlwaysOn + OnAndOff);
    switch (indexevt) {
         case ARRIVAL: p1 = Lambda/Delta; break;
         case SERVICE: p1 = (inserv)*Mu/Delta; break;
         case SWITCHINGON: p1 = Nu/Delta; break;
         case LOOP: p1 = Mu*(AlwaysOn + OnAndOff - inserv)/Delta; break;
    }
    return(p1);
}
```

The model is in continuous time. Thus we build an uniformized version of the model adding a new event to generate the loops in the transition graph which are created during the uniformization. After this process we have 4 events: ARRIVAL, SERVICE, SWITCHINGON, LOOP. In all the functions, E and F are states. The generation tool creates 3 files: one contains the transition matrix in sparse row format, the second gives information on the number of states and transitions and the third one stores the encoding of the states. Indeed the states are found during the BFS visit of the graph and they are ordered by this visit algorithm. Thus, we have to store in a file the mapping between the

state number given by the algorithm and the state description needed by the modeler and some algorithms.

```
void Equation(int *E, int indexevt, int *F, int *R)
{
    F[0] = E[0]; F[1] = E[1];
    switch (indexevt) {
      case ARRIVAL: if (E[0]<BufferSize) {F[0]++;}
              if ((E[0]>=UP) && (E[1]==UPandDOWN)) {F[1]=WARMING;}
              break;
      case SERVICE: if (E[0]>0) {F[0]--;}
              if ((F[0]==DOWN) && (E[1]>UpandDOWN)) {F[1]=UPandDOWN;}
              break;
      case SWITCHINGON: if (E[1]==WARMING) {F[1]=ALL_UP;}
              break;
      case LOOP: break;
    }
}
```

Once the steady-state distribution is obtained with some numerical algorithms, the marginal distributions and some rewards are computed using the description of the states obtained by the generation method and codes provided (and compiled) by the modeler to specify the rewards (see in the left part of Fig. 1 the marginal distribution for the queue size).

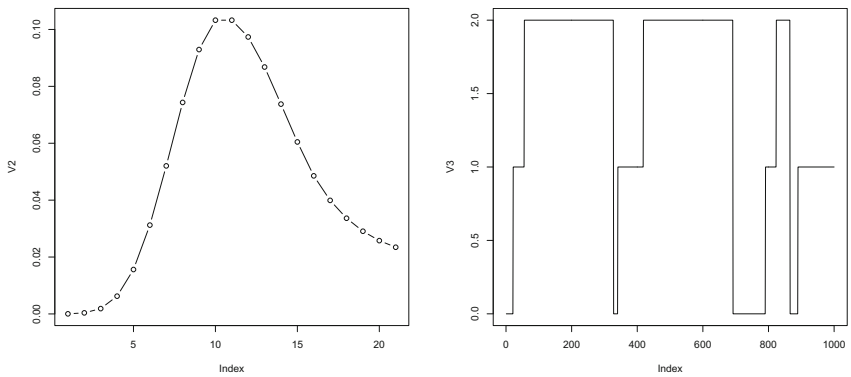

Fig. 1. Mitrani's model. Steady-state for the queue size (left). Sample path of the state of the servers (right).

3 Numerical Resolution

In XBorne, we have developed some well-known numerical algorithms to compute the steady-state distribution (GTH for small matrices), SOR and Gauss Seidel for large sparse matrices but we have chosen to export the matrices into

MatrixMarket format to use state of the art solvers which are now available on the web. But we also provide new algorithms for the stochastic bounds or the element-wise bound of the matrices, the stochastic bound or the entry-wise bounds of the steady-state distribution. These bounds are based on the algorithmic stochastic comparison of Discrete Time Markov Chain (see [10] for a survey) where stochastic comparison relations are mitigated with structural constraints on the bounding chains. More precisely, the following methods are available:

- Lumpability: to enforce the bounding matrix to be ordinary lumpable. Thus, we can aggregate the chain [9].
- Pattern based: to enforce the bounding matrix to follow a pattern which provides an ad-hoc numerical algorithm (think at a upper Hessenberg matrix for instance) [2].
- Censored Markov chain: only the useful part of the chain is censored and we provide bounds based on this partial representation of the chain [1,7].

Other techniques for entry-wise bounds of the steady state distribution have also been derived and implemented [3]. They allow in some particular cases to deal with infinite state space (otherwise not considered in XBorne).

More recently, we have developed a new low rank decomposition for a stochastic matrix [4]. This decomposition is adapted to stochastic matrices because it provides an approximation which is still a stochastic matrix while singular value decomposition gives a low rank matrix which is not stochastic anymore. Our low rank decomposition allows to compute the steady-state distribution and the transient distribution with a lower complexity which takes into account the matrix rank. For instance, for a matrix of rank k and size N, the computation of the steady-state distribution requires $O(Nk^2)$ operations. We also have derived algorithms to provide stochastic bounds with a given rank for any stochastic matrix (see [4]).

Note that the integration with other tools we mention previously is not limited to numerical algorithms provided by statistical package like R. We also

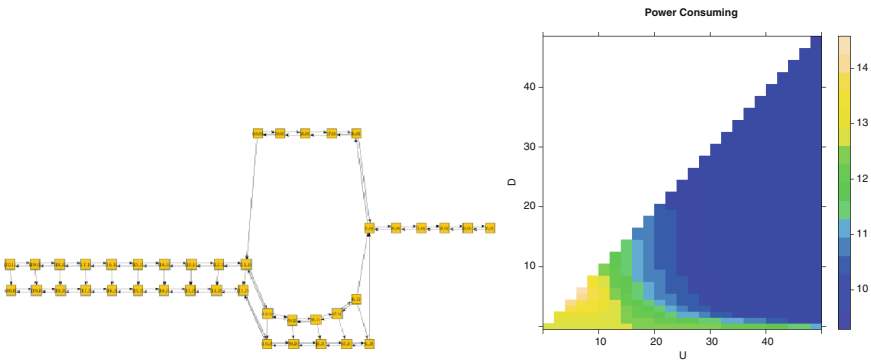

Fig. 2. Mitrani's model. Directed graph of the chain (left). Energy consumption (right).

use their graphic capabilities and the layout algorithms. We illustrate these two aspects in Fig. 2. In the left part we have drawn the layout of the Markov chain associated with Mitrani's model for a small buffer size (i.e. 20). We have developed a tool which reads the Markov chains description and write it as a labelled directed graph in "tgf" format. With this graph description, we use the graph editors available on the web to obtain a layout of the chain and to visualize the states and their transitions. On the right part of the figure, we have depicted a heat diagram for the energy consumption associated to Mitrani's model for all the values of the thresholds U and D.

4 Quasi-Lumpability

Quasi-Lumpability testing has been recently added into XBorne to analyze very large matrices. The numerical algorithms which have been developed are also used to analyze stochastic matrices which are not completely specified. It is well-known now that Tarjan's algorithm can be used to obtain the coarsest partition of the state space of a Markov chain which is ordinary lumpable and which is consistent with an initial partition provided by the modeler. Lumpable matrix can be aggregated to obtain a smaller matrix, easier to analyze. Logarithmic reduction in size are often reported in the literature. We define quasi-lumpability of partition $A_1, A_2, \ldots, A_k$ with threshold ϵ of stochastic matrix M as follows: for all macro-states A_i and A_j we have

$$\max_{l1,l2 \in A_i} \left| \sum_{k \in A_j} M(l1,k) - \sum_{k \in A_j} M(l2,k) \right| = E(i,j) \leq \epsilon. \tag{1}$$

When $\epsilon = 0$ we obtain the definition of ordinary lumpability. We have modified Tarjan's algorithm to obtain a partition which is quasi-lumpable given an initial partition and a maximum threshold ϵ. The output of the algorithm is the coarsest partition consistent with the initial partition and the real threshold needed in the algorithm (which can be smaller than ϵ). Note that the algorithm always returns a partition. However the partition may be useless as it may have a large number of nodes. The next step is to lump matrix M according to the partition found by the modified Tarjan's algorithm. If the real threshold needed is equal to 0, the matrix is lumpable and the aggregated matrix is stochastic. It is solved with classical methods.

If the threshold needed is positive, we obtain two aggregated matrices Up and Lo: one where the transition probability between macro states A_i and A_j is equal to $\max_{l \in A_i} \sum_{k \in A_j} M(l,k)$ and one where it is equal to $\min_{l \in A_i} \sum_{k \in A_j} M(l,k)$. Up is super-stochastic while Lo is sub-stochastic. These two bounding matrices also appear when the Markov chains are not completely specified and transitions are associated with intervals of probability. We have implemented Courtois and Semal algorithm [6] to obtain entry-wise bounds on the steady-state distribution of all matrices between Up and Lo. We are still conducting new research to improve this algorithm.

5 Simulation

We have added several simulation engines in XBorne, mainly for educational purpose and for verification. All of them define a model with the same functions we have previously presented to design a Markov chain. The modeler just needs to add the simulation time and the seed for the generator when a random number generator is used by the simulation code. Thus, the same model description (i.e. the four C functions) is used for the simulation and the Markov chain generation.

Two types of engines have been developed: a simulator with random number generation in C and a trace base version where the random number generation (and generally the random variables generation) are outside the simulation code and previously stored in a file by some statistical packages (typically R). Similarly, the output of the simulations are sample paths which are stored in separate files to be analyzed by state of the art statistical packages where various test algorithms and confidence intervals computations are performed by efficient methods already available in these packages. Thus, the modeler is expected to concentrate on the development of the model simulation, leaving the statistical details to other packages. Similarly, the drawing of the paths can be obtained from the statistical package like in the right part of Fig. 1 where we depict the evolution of the second component of Mitrani's model (i.e. the state of the server). The trace based simulation is also used to simulate Semi-Markov processes.

The simulation engines also differ by the definition of paths: the general purpose simulation engine builds one path per seed for the simulation time, while the regenerative Markovian simulation stores one path per regenerative cycle. Furthermore, to deal with the complexity of the simulation of discrete distribution by the reverse transform method, we have implemented two types of engine: a general inverse distribution method when the distribution of probability for the next event changes with the state, and an alias method when this distribution is the same for all the states.

Acknowledgments. This work was partially supported by project MARMOTE (ANR-12-MONU-00019). Y. Ait El Mahjoub is supported by Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program Investissement d'Avenir Idex Paris-Saclay (ANR-11-IDEX-0003-02).

Open Access. This chapter is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, duplication, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, a link is provided to the Creative Commons license and any changes made are indicated.

The images or other third party material in this chapter are included in the work's Creative Commons license, unless indicated otherwise in the credit line; if such material is not included in the work's Creative Commons license and the respective action is not permitted by statutory regulation, users will need to obtain permission from the license holder to duplicate, adapt or reproduce the material.

References

1. Busic, A., Djafri, H., Fourneau, J.M.: Bounded state space truncation and censored Markov chains. In: 51st IEEE Conference on Decision and Control (CDC 2012) (2012)
2. Busic, A., Fourneau, J.M.: A matrix pattern compliant strong stochastic bound. In: 2005 IEEE/IPSJ International Symposium on Applications and the Internet Workshops (SAINT Workshops), Italy, pp. 260–263. IEEE Computer Society (2005)
3. Busic, A., Fourneau, J.M.: Iterative component-wise bounds for the steady-state distribution of a Markov chain. Numer. Linear Algebra Appl. **18**(6), 1031–1049 (2011)
4. Busic, A., Fourneau, J.M., Ben Mamoun, M.: Stochastic bounds with a low rank decomposition. Stochast. Models **30**(4), 494–520 (2014). Special Issue with selected papers from the Eighth Int. Conf. on Matrix-Analytic Methods in Stochastic Models
5. Busic, A., Gaujal, B., Gorgo, G., Vincent, J.M.: Psi2: Envelope perfect sampling of non monotone systems. In: QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Virginia, USA, pp. 83–84. IEEE Computer Society (2010)
6. Courtois, P.J., Semal, P.: On polyhedra of Perron-Frobenius eigenvectors. Linear Algebra Appl. **65**, 157–170 (1985)
7. Dayar, T., Pekergin, N., Younès, S.: Conditional steady-state bounds for a subset of states in Markov chains. In: Structured Markov Chain (SMCTools) workshop in VALUETOOLS. ACM (2006)
8. Fourneau, J.M., Le Coz, M., Pekergin, N., Quessette, F.: An open tool to compute stochastic bounds on steady-state distributions and rewards. In: 11th International Conference on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Orlando. IEEE Computer Society (2003)
9. Fourneau, J.M., Le Coz, M., Quessette, F.: Algorithms for an irreducible and lumpable strong stochastic bound. Linear Algebra Appl. **386**, 167–185 (2004)
10. Fourneau, J.M., Pekergin, N.: An algorithmic approach to stochastic bounds. In: Calzarossa, M.C., Tucci, S. (eds.) Performance 2002. LNCS, vol. 2459, pp. 64–88. Springer, Heidelberg (2002). doi:10.1007/3-540-45798-4_4
11. Mitrani, I.: Service center trade-offs between customer impatience and power consumption. Perform. Eval. **68**(11), 1222–1231 (2011)
12. Valmari, A., Franceschinis, G.: Simple $O(m \log n)$ time Markov chain lumping. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 38–52. Springer, Heidelberg (2010)