Optimizing distributed DEVS simulations with partitioning and Hidden Markov Model learning methods

Christopher Herbez, Eric Ramat, Gauthier Quesnel

Optimizing distributed DEVS simulations with partitioning and Hidden Markov Model learning methods

Christopher Herbez & Eric Ramat
LISIC, ULCO
50 rue Ferdinand Buisson, Calais, France
email: [herbez,ramat]@lisic.univ-littoral.fr

Gauthier Quesnel
MIAT, INRA
24 chemin de Borde Rouge, Castanet-Tolosan, France
email: gauthier.quesnel@toulouse.inra.fr

KEYWORDS
Simulation, DEVS, Graph weighting, Partition, Hidden Markov Models

ABSTRACT

With the emergence of parallel computational infrastructures at low cost, reducing simulation time becomes again an issue of the research community in modeling and simulation. In this context, our previous papers presented a method to reduce the simulation time in a parallel DEVS context. This approach reduces simulation time without reaching the maximum gain. The partitioning method used does not take into account the dynamic of models. To address this problem, we propose in this paper an approach to weight the model graph to take into account this dynamic when partitioning. This paper presents the weighting graph process by learning of the dynamic of models states using Hidden Markov Models. The purpose of this article is to determine the quality of this weighting method compared to a simulation approach.

INTRODUCTION

Modeling and analysis of complex systems dynamics are becoming increasingly costly in time and memory capacity. The multi-modeling is a response to the increase demand for coupling heterogeneous models. Obviously, this process leads to the increase in computation demand and therefore, the increase of computation time. It is therefore important to think about the good use of new physical processor infrastructure (multicore, multiprocessor and grid). Work in this area is it not new : includes all work around distributed simulation [Chandy and Misra, 1979, Chandy and Misra, 1981] but also work on parallel computing [Fujimoto, 1990]. However, what interests us is the construction of an optimized organization of simulators as part of DEVS (Discrete Event Specification). The DEVS formalism [Zeigler et al., 2000] and Parallel-DEVS variant [Chow and Zeigler, 1994] is a candidate to develop a response to both formal and technical. That is a discrete events modeling and simulation theory with a hierarchical approach. The global model, called structure of the model in DEVS terminology, is a graph of coupled models. Our approach is to flattening the hierarchy in order to obtain the graph of models. This graph is partitioned in order to parallelize the models execution as efficiently as possible.
In [Herbez et al., 2015a], we presented this approach as well as the relative gains obtained for two types of partitioning. One is based on the connectivity of the graph, and the other is oriented modeler. In these examples, the gain obtained by the introduction of a good partitioning is about 20% compared to an initial model hierarchy. Thereafter, in [Herbez et al., 2015b], we showed the limits of this approach. Currently, the dynamic of models is not taken into account when graph partitioning, which explains the limited gain observed. In this paper we propose an approach to weight this graph in order to better reflect the dynamic of models.
The first part describes the Parallel-DEVS formalism and our approach. The second part present the methodology implemented to weight the model graph using the Hidden Markov Models. Finally, the last part evaluates the quality of the graph weighting obtained by our approach and allows to validate its.

EFFICIENT DISTRIBUTION OF MODELS IN A PARALLEL DEVS CONTEXT

Our researches is mainly centered around optimizing discrete simulations in a parallel DEVS context. This paper presents the changes that we propose to make in order to optimize these simulations in time and memory. For this, we propose a modification of the DEVS hierarchical structure. This section presents the DEVS formalism and the approach set up to change the hierarchy of models.

Parallel-DEVS context

DEVS [Zeigler et al., 2000] is a high level formalism based on the discrete events for the modeling of complex discrete and continuous systems. The model is a network of interconnections between atomic and coupled models. These models are in interaction via time-stamped events exchanges.

More specifically, we present the Parallel-DEVS (PDEVS) formalism [Chow and Zeigler, 1994, Chow, 1996]. This extension of the classic DEVS introduces the concept of simultaneity of events essentially by allowing bags of inputs to the external transition function. Bags can collect inputs that are built at the same date, and process their effects in future bags.
PDEVS defines an atomic model as a set of input and output ports and a set of state transition functions:

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

With: X, Y, S are respectively the set of input values, output values and sequential states

$ta : S \to \mathbb{R}_0^+$ is the time advance function

$\delta_{int} : S \to S$ is the internal transition function

$\delta_{ext} : Q \times X^b \to S$ is the external transition function

where:

$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$

Q is the set of total states,

e is the time elapsed since last transition

X^b is a set of bags over elements in X

$\delta_{con} : S \times X^b \to S$ is the confluent transition function, subject to $\delta_{con}(s, \emptyset) = \delta_{int}(s)$

$\lambda : S \to Y$ is the output function

If no external event occurs, the system will stay in state s for $ta(s)$ time. When $e = ta(s)$, the system changes to the state δ_{int}. If an external event, of value x, occurs when the system is in the state (s, e), the system changes its state by calling $\delta_{ext}(s, e, x)$. If it occurs when $e = ta(s)$, the system changes its state by calling $\delta_{con}(s, x)$. Every atomic model can be coupled with one or several other atomic models to build a coupled model. This operation can be repeated to form a hierarchy of coupled models. A coupled model is defined by:

$$N = \langle X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\} \rangle$$

Where X and Y are input and output ports, D the set of models and:

$\forall d \in D, M_d$ is a PDEVS model

$\forall d \in D \cup \{N\}, I_d$ is the influencer set of d :

$I_d \subseteq D \cup \{N\}, d \notin I_d, \forall d \in D \cup \{N\}$,

$\forall i \in I_d, Z_{i,d}$ is a function,

the i-to-d output translation:

$Z_{i,d} : X \to X_d$, if $i = N$

$Z_{i,d} : Y_i \to Y$, if $d = N$

$Z_{i,d} : Y_i \to X_d$, if $i \neq N$ and $d \neq N$

The influencer set of d is the set of models that interact with d and $Z_{i,d}$ specifies the types of relations between models i and d.
The atomic and coupled models are respectively associated with simulators and coordinators. The aim of simulators is to compute the various functions while the coordinators manage the synchronization of exchanges between simulators (or coordinators in a hierarchical view).

Optimization of the DEVS hierarchical structure

Many DEVS implementations reduce the hierarchy to only one level : a coordinator and several simulators. [Muzy and Nutaro, 2005] introduce this type of structure called flattening. The first step of our approach is to flattening the hierarchy in order to obtain a graph of model. This work is possible thanks to the closure under coupling property of DEVS [Zeigler et al., 2000]. This property describes formally the coupled model is equivalent to an atomic model. Thus an atomic model can be move into a new coupled model and all the hierarchy of coupled model can be merge into a unique coupled model. The connections between atomic models of this coupled model give an oriented graph called graph of model. The Figure 1 give an example of flattening of the hierarchy.

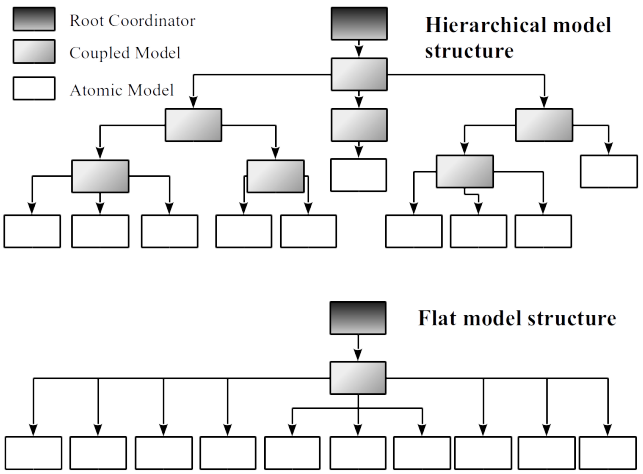

Figure 1: At the top, an example of hierarchical structure of a DEVS model. At the bottom, new structure after flattening of the hierarchy

Our approach is centered around the restructuring of the hierarchy so that it is optimal for distributed simulation. The optimality depends on two main factors:

- charge balance between different parties to execute a maximum of models in parallel
- minimizing of exchanges between different parties to avoid increase of transfers cost related to information exchange between hosts

The restructuring is done by a partitioning of the models graph. It is made in order to comply with the conditions presented above.
The partitioning phase of our approach is presented in details in our previous paper [Herbez et al., 2015a]. It consist to apply the multilevel method presented in [Karypis and Kumar, 1998]. This method is divided into 3 phases:

- Coarsening: Graph reduction by successive vertices matching, while keeping the nature of the original graph. The Heavy Edge Matching introduced in [Karypis and Kumar, 1998] is implemented for this phase.
- Partitioning: Creating of a partition P_k of the coarsening graph G_n using a partitioning heuristic. We choose an expanding region method: the Greedy Graph Growing Partitioning presented in [Bichot and Siarry, 2013].
- Uncoarsening: Projection of the partition P_k on each contraction graph levels with a refinement for keep a good quality. We use a local optimization algorithm based on Kernighan-Lin algorithm [Kernighan and Lin, 1970].

This method generates a partition of atomic models minimizing the optimality criteria for distributed simulation. From this partition, a new two-level hierarchy is built. This hierarchy consists of the same number of coupled models that there are parts in the partition. The atomic models of the same part are assigned to the same coupled model. And these new coupled models are distributed on the network of machines available for the simulation. The Figure 2 illustrates the construction of the new hierarchy from a partitioning on the previous example.

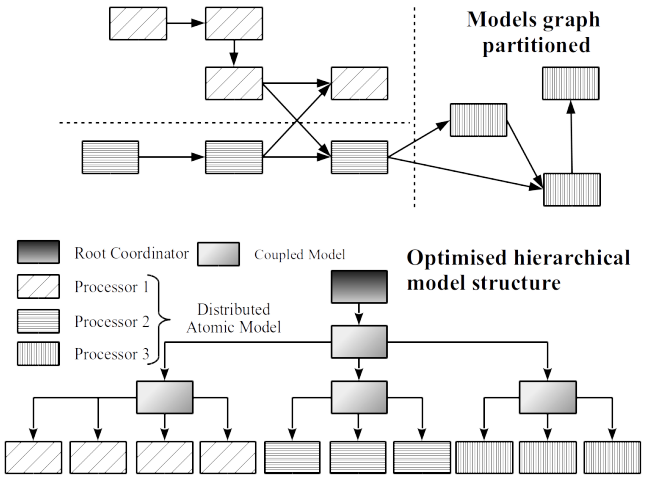

Figure 2: At the top, example of partitioning of the models graph. At the bottom, new hierarchical structure at 2 levels obtained from this partition

The current problem is that the model graph as it is constructed does not contain information regarding the information transmission frequency from one model to another or on their execution time. However, these information are crucial for the partitioning quality because they correspond to the weighting of the graph and reflect the dynamic of models. For now, this weighting is set to 1 by lack of information, which damages the partitioning quality. To address this problem, the following section presents an approach to weight the graph by learning of model dynamics.

Weighting of model graph

The model graph is a oriented graph where each vertex corresponds to an atomic model and each edge at information flow linking two atomic models. Graph theory, presented in [Bichot and Siarry, 2013], authorizes the weighting of these elements. For a vertex, weight is a quantification of the running time of the model that it symbolizes. More a model has a long execution time, more the weight of the vertex associated with it will be great. For an edge, the weight quantifies for one hand the amount of data that flows between two models but also the frequency at which they pass. The aim of our approach is to develop a graph weighting system without knowledge of the internal dynamic of models.

Each atomic model is considered individually and as a black box, the internal dynamics of the model is assumed unknown. The unique information provided by model are the duration d_i corresponding to the elapsed time between each emission of external events via the output function λ. Moreover, models tells us if it is in a "Infinite state" or not. We call "Infinite state", a state whose the duration is infinite. This assumption is strong but essential for the effective convergence of learning algorithms.

From these information, a learning algorithm is performed to create a probabilistic states automaton. These automata must be able to generate sequences of observations statistically close to that of the model. The advantage of this approach is to generate a very large number of observations without being penalized by the calculation times that require multiple executions of a model. This is possible provided that the learning model is less expensive than the generation of observation sequences by simulation.

The observation sequences obtained by simulation of probabilistic states automata allow to estimate the emission frequency of outputs by calculating a weighted mean. Consider an observation sequence of emission duration between two outputs consisting of k distinct values $d_1, \cdots, d_k$ and denote n_i $\forall i = 1, \cdots, k$ the number of times where the duration d_i appears in the sequence. The average duration of emission between two outputs

is given by:

$$d\_mean = \frac{\sum_{i=1}^{k} d_i \times n_i}{\sum_{i=1}^{k} n_i}$$

Its provides the average frequency of issue:

$$f\_mean = \frac{1}{d\_mean}$$

if $f\_mean \longrightarrow 0$ slow model

if $f\_mean > 1$ fast model

The weighting of the edges is given by emission frequency average f_mean. When generating the observation sequence by simulation, the elapsed time to obtain for a set of N durations is recorded. With this information, we can weight the edges of the graph by this time.
We start from the assumption that our simulation consists of K types of different models. The goal is to achieve a learning for each type of model in order to establish a weighting rule. This approach enables minimize the number of learning. The following section outlines the learning method used to model the dynamics of the models.

LEARNING OF ATOMIC MODELS DYNAMIC USING HIDDEN MARKOV MODELS

Knowing that, for each model, the set S of states is unknown and that only a series of observations $Y = \{y_1, \cdots, y_k\}$ is available through simulation, we choose to model the dynamic of models from a Hidden Markov Model (HMM) introduced by Baum and his colleagues in [Baum et al., 1970]. The following section presents the hidden Markov models.

Generalities on Hidden Markov Models

A Markov chain is a sequence of random variables ($S_n,\ n \in N$) which allows to model the dynamics of a system. The feature of this process is that the current state S_n is independent of the past state S_{n-1}, which is expressed mathematically by:

$$P(S_n = j | S_1 = i_1, \cdots, S_{n-1} = i_{n-1}) =$$
$$P(S_n = j | S_{n-1} = i_{n-1})$$

Having no knowledge of the dynamics of the model, it is necessary to assume that every state is reachable from any system status. Moreover, each transition is associated at an observation which is the duration of emission d_i between two outputs. Considering this information, it appears that an ergodic Hidden Markov Model is a good candidate to model our state graph. One of the features of a HMM is a finite symbol alphabet is associated with each state. In our case, symbols correspond to period d_i observed by simulation.

The HMMs are defined in [Rabiner, 1989] by five parameters:

- N the number of states in the model. Where the states are defined by $S = \{S_1, \cdots, S_N\}$.
- M the number of symbols. Where symbols are defined by $V = \{v_1, \cdots, v_M\}$.
- $A = \{a_{i,j} |\ i, j = 1, \cdots, N\}$ the state transition probability distribution.
- $B = \{b_{i,j} |\ i = 1, \cdots, N\ ;\ j = 1, \cdots, M\}$ the observation symbol probability distribution. Where $b_{i,j}$ is the probability that the state S_i emit the symbol v_j.
- $\pi = \{\pi_1, \cdots, \pi_N\}$ the initial state distribution.

For clarity, we call an HMM by $HMM = \{\pi, A, B\}$, where π, A and B are built such that:

$$\sum_{i=1}^{N} \pi_i = 1\ ,\ \sum_{j=1}^{N} a_{i,j} = 1 \text{ and } \sum_{j=1}^{M} b_{i,j} = 1$$

The Figure 3 illustrates an ergodic Hidden Markov Model by an example.

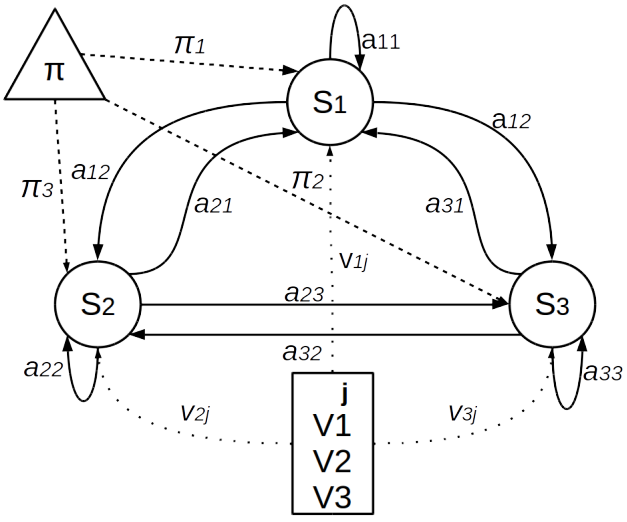

Figure 3: Example of ergodic HMM with 3 states and 3 symbols

For each learning, an HMM is created with a view to answering the following problem: *Starting from a set of observations $O = \{O_1, \cdots, O_k\}$, how to adjust settings of an HMM to model at best the process?*
The following section presents the algorithms used for the HMM learning.

Learning Methods of HMMs

Learning determines the best HMM parameters to represent observations O ie, which attribute to O the best

probability of occurrence. We denote $P_{Hmm}(O)$ the probability of occurrence of the sequence O assigned by the HMM, the goal is to estimate the parameters $\widehat{HMM}$ that maximizes $P_{Hmm}(O)$:

$$\widehat{Hmm} = arg \max_{Hmm} P_{Hmm}(O)$$

The learning process begins with system initialization $Hmm = \{\pi, A, B\}$. As we have no knowledge except the observation sequence $O = \{O_1, \ cdots, O_k\}$, it is necessary to begin by arbitrarily fix the number of system states. For the number of symbol, it suffices to determine the different number of observations in the sequence (assuming that the sequence contains all the symbols). The sets π, A and B can be randomly constructed (provided they respect the properties of an HMM) or with equal probability distributions.
It is very difficult if not impossible to get the perfect $\widehat{HMM}$. However, there is a method to get a good approximation: the Baum-Welch algorithm presented in section 4 [Bilmes, 1998]. The Baum-Welch algorithm is a learning algorithm derived from the EM algorithm (Expectation Maximization). Given a set of observation sequences O and an initial model Hmm, the Baum-Welch algorithm undertakes a re-estimation of the parameters (π, A, B) of the model so as to increase generating probabilities of these observation sequences. To optimize computation steps, the Baum-Welch algorithm uses the Forward and Backward algorithms also detailed in [Bilmes, 1998]. The procedure is to re-estimate the parameters of the model and to recalculate the likelihood of the new model. The process is re-iterated until the current maximum likelihood.
The following section presents the relationship between DEVS models and HMMs.

Building of learning sets from atomic DEVS model simulations

The atomic models of which we seek to model the dynamics are classified into two categories:

- models without input : they are simply represented by a conventional Hmm (see Figure 3).
- models with inputs: a "modified" HMM is used to be able to represent the states called "Infinite state"

Indeed, it is essential to represent the states where the model is waiting for an input event. For this we add the symbol ∞ corresponds to a state of infinite duration. This additional symbol will also lead us to modify the generation algorithm by simulation.
On the other hand, the learning of models with inputs requires the creation of a generator. This generator is intended to simulate potential model inputs. To ensure the best exploration of the dynamics of the model, the generator must explore the widest possible spectrum of the input values (d_i). For simplicity, we consider that the events do not carry data.
The generation of learning sets from individual atomic models, produces sequences of type $\{d_i\}$ where d_i is either a positive integer value or infinity for "Infinite states". There is one sequence by model type. This sequence is then divided into sub-sequences. Each subsequence has a random length and ends with an end symbol. This splitting is random. It is assumed that the splitting will not lead to bias in learning. From these sub-sequences, HMMs are constructed by learning. Figure 4 illustrates the learning process.

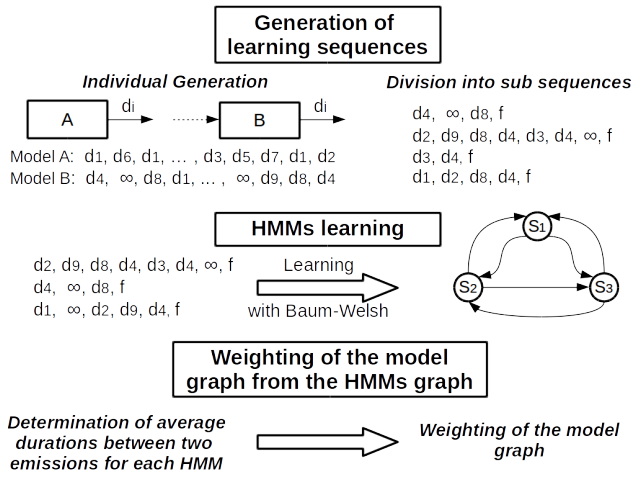

Figure 4: Learning scheme of the weighting of the model graph

The following section present how the HMM obtained by learning allows to create the weighting of edges of the model graph.

Building weighted graph from HMM models

The learning of models is performed using one or more sequences of observations obtained by simulation. The sequences should be neither too large not to lose time with the calculations nor too short not to lose information on symbols. From these observations, we first construct the set of symbols V and then initializes the Hidden Markov Model $Hmm = (\pi, A, B)$. For models with inputs, we add the symbol ∞ to the set V. The number of states is fixed at $N = 6$ and the distribution probabilities are almost equal for A and for B.
Once the HMM constructed from the Baum-Welch algorithm, the complete graph is reconstructed by replacing the atomic models by their HMM. We simulate the automaton graph at probabilistic states. This simulation is based on the observation sequences generation algorithm for models without entry (see Algorithm 1) and on a variant of the previous algorithm for models with inputs see Algorithm 2).

The random selection for a state or a observation, according to the probabilities π, A ans B, requires the creation of sets of cumulative probabilities πc, Ac and Bc. Where

$$\pi c_i = \sum_{k<i} \pi_k + \pi_i$$
$$Ac_{ij} = \sum_{k<j} A_{ik} + A_{ij}$$
$$Bc_{ij} = \sum_{k<j} B_{ik} + B_{ij}$$

To simulate this selection, we select three values $k_\pi, k_a, k_b \in [0,1]$ with a uniform distribution. To determine the initial state, we search j such as $k_\pi \in [\pi c_{j-1}, \pi c_j]$. Similarly, to determine the future state and the emitted symbol, we search j such as $k_a \in [Ac_{ij-1}, Ac_{ij}]$ and $k_b \in [Bc_{ij-1}, Bc_{ij}]$.

Algorithm 1 Observation sequence generation for model without input

```
1:  procedure
2:      select an initial state S_i with probabilities π
3:      set initial state S ← S_i
4:      select a symbol v_k with probabilities b_ik
5:      seq ← {v_k}
6:      while S ≠ S_N do
7:          select a future state S_j with probabilities a_ij
8:          S ← S_j
9:          select a symbol v_k with probabilities b_jk
10:         seq ← seq ∪ {v_k}
        return seq
```

Changes that we must achieve in the simulation algorithm are twofold: to take into account the symbols ∞ and interaction between the HMMs. In algorithm 1, the sequence may be entirely generated because it is not dependent on external information. However, in the case of models with inputs, we must take account of events produced by the upstream models of the simulated model. The sequences $\{d_i\}$ are converted into events. Events are placed in input of the models and the arrival dates are defined by the d_i. Events are considered in the simulation of HMMs when the automaton sends the symbol ∞. Indeed, when the symbol ∞ is generated, the automaton waits for an external event (as defined in the DEVS formalism). However, if the transmitted symbol is not ∞ and an external event happens then the event is ignored. So unlike the conventional case, the HMM simulator is piloted by the presence of the symbol ∞ and the arrival of events. Finally, emission time average is determined between two outputs on all edges of the graph. This weighted average is the computation of the expectation of the variable of symbols derived of the symbol ∞: $V - \{\infty\} = \{v_1, \cdots, v_{M-1}\}$. At each v_j is associated a probability $P(v_j) = \sum_{i=1}^{N} b_{i,j}$ which is

Algorithm 2 Observation sequence generation for model with inputs

```
1:  procedure
2:      D, set of arrival dates of external events
3:      I = 0, index of current date in D
4:      select an initial state S_i with probabilities π
5:      set initial state S ← S_i
6:      select a symbol v_k with probabilities b_ik
7:      seq ← {v_k}
8:      t = 0
9:      while t < t_max do
10:         select a future state S_j with probabilities a_ij
11:         S ← S_j
12:         if S = S_N then
13:             select a new initial state S_j with probabilities π
14:             S ← S_j
15:         select a symbol v_k with probabilities b_jk
16:         if v_k ≠ ∞ then
17:             t ← t + v_k
18:             seq ← seq ∪ {t − t_last + v_k}
19:             t_last ← t
20:             while D_I <= t do I ← I + 1
21:         else
22:             t ← D_I
23:             I ← I + 1
        return seq
```

the sum of the emission probabilities of the symbol j for each state S_i. The expectation of the variable V is:

$$E(V) = \sum_{j=1}^{M-1} v_j P(v_j) = \sum_{j=1}^{M-1} v_j \sum_{i=1}^{N} b_{ij} = \sum_{j=1}^{M-1} \sum_{i=1}^{N} v_j b_{ij}$$

The entire process of weight computation will be illustrated in an example of model graph presented in section *Model and graph.*

DATA PRESENTATION AND RESULTS

To illustrate the process, we will use an example based on a graph of atomic models. This graph consists of a single type of deterministic model. For each model, the parameters are different which implies different dynamics from the point of view of d_i. Using this example, we will show the impact of the approximation due to HMM. It seems obviously that the building by learning of HMMs leads to a loss of accuracy compared to the original models. We will quantify this loss and show that the impact on the estimated weight is low.

Models and graph

Our tests are made from only one type of models. An atomic model represents the fill of N tanks of same size, where each has its own filling speed s_i. The dynamic is

this: when K tanks have reached the maximum capacity Q_{max}, they are emptied. If all tanks are full so an output event is generated. At this time, a d_i is calculated. Figure 5 illustrates the operation of this model. Depending on the setting, one sees a dynamic that can be represented by a finite state automaton. The d_i are shown on the automaton edges and vertices represent states where the tanks are all full.

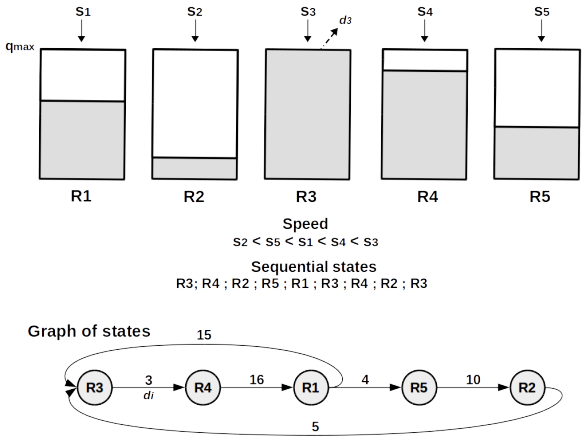

Figure 5: Tank model example and state graph

Then, we introduce an external dynamic at the deterministic model. The tanks are under the influence of upstream tanks. When tanks were full k times, downstream tanks fill faster. We model this mechanism by: when k external events arrived, the filling speed is increased for one of the tanks. The speed will be reset to the initial state when all speeds have been increased.

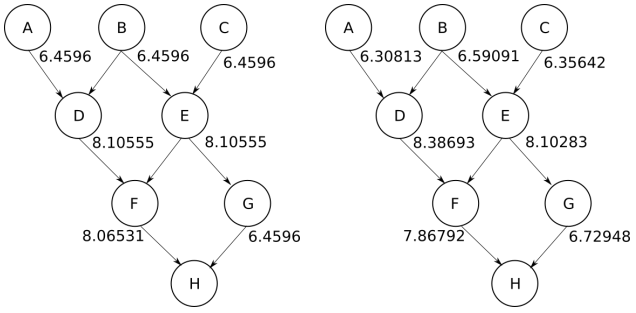

Figure 6: On the left, atomic model graph with d_{mean}. On the right, HMM graph with d_{mean}

Results

The first step of the method allows us to get the d_i for all atomic models. The models A, B and C have same values for parameters. In this way, a single simulation is required (see figure 7). In the general case, it is important to identify identical models to minimize the number of simulations. For the other five models, three simulations are needed: D and E are identical and H have no exit. These three simulations require the setting of input generators: each generator produces d_i randomly according to a uniform law whose boundaries are between the minimum and maximum of d_i observed at the output of the upstream models. We have four sets of d_i.

$$15; 3; 15; 1; 2; 12; 1; 17; 2; 4;$$
$$\{9; 3; 12; 3; 3; 4; 8; 10; 15; 1; 2; 2; 3; 8; 7; 8; 4; 11;$$
$$3; 12; 3; 12; 1; 2; 1; 11; 6; 9; 15; 3;$$
$$1; 14; 3; 9; 3; 15; 3; 1; 11; 3; 15; 3; 2; 13; 1; 2; 12; 6;$$
$$13; 2; 3; 12; 15; 3; 1; 5\};$$
$$\{9; 3; ...\}; ...$$

Figure 7: d_i of A model with stable sub-sequence

According to the method, the d_i are subdivided to obtain a set of observation sub-sequences for the learning step. We took the opportunity to compute the duration average d_{mean} (see figure 6).
Two parameters are to be set for the learning step (Baum-Wesh algorithm): the number of maximum of iterations and the number of hidden states. In our case, we fix them respectively 100 and 10. The number of hidden states leads to highly variable quality results. It is important to fix them an adequate way. By example, one would be to vary them to retain the optimal number. Some methods exist (for example, [Geiger et al., 2010]).

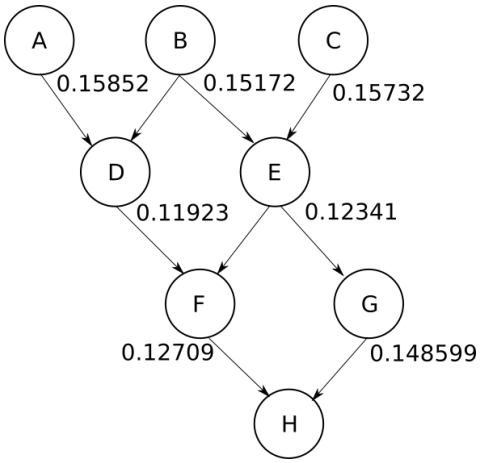

Figure 8: HMM graph with weights

Then, from HMMs constructed by learning, an HMMs graph is built and simulated. We then obtain the simulated d_{mean}. Figure 6 shows the differences between the simulated d_i from DEVS models and simulated d_i from HMMs. There are different but the results are close. To improve these results, it is necessary to introduce a method for optimizing the number of hidden states, to study the impact of the initial conditions of learning and also to study the setting of generators for models with inputs. These three elements determine the quality of HMMs. In addition, the DEVS models treated in the

example are deterministic and show sequences that are repeated in the d_i. What is the impact? What happens if there is no repetition or if the sequences are too short? The last step is to convert the d_i to weight (see figure 8) and make partitions from the flattening graph [Herbez et al., 2015b]. Partitioning is not addressed in this paper.

CONCLUSION AND PERSPECTIVES

The optimization of the hierarchical structure of the DEVS simulation is not limited to a simple partitioning of the model graph. As we have shown in our previous work, the partitioning process does not take into account the dynamics of the models, which is reflected by a lower gain in performance. In this article, we proposed a method of weighting of the model graph with Hidden Markov models.

As we have seen in the results section, several factors may impact the quality of learning. It is therefore important to study the different impacts. In addition, the DEVS models that were used to validate the approach, belong to a subclass of DEVS models (deterministic and finite states). We must therefore expand the types of models (eg, stochastic). Similarly, d_i are integer and their number is finite. It is important to extend our results to models whose d_i are real and whose number is infinite. The definition of d_i classes could be a solution. There are also two other restrictions: the models have only one input port and one output port, and events do not carry data. The first restriction is not complicated to take into account, in particular, in the simulation algorithms of HMM. However, it is more difficult to take into account data. We should build specific generators for models with inputs.

One last important perspective is the study of the transformation function of d_i in weight. This function is now quite simple and non-linear. Another way, by example linear, might be a better choice.

REFERENCES

[Baum et al., 1970] Baum, L. E., Petrie, T., Soules, G., and Weiss, N. (1970). A maximization technique occurring in statistical analysis of probabilistic functions in markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171.

[Bichot and Siarry, 2013] Bichot, C. E. and Siarry, P. (2013). *A Partitioning Requiring Rapidity and Quality: The Multilevel Method and Partitions Refinement Algorithms*, pages 27–63. John Wiley & Sons, Inc.

[Bilmes, 1998] Bilmes, J. A. (1998). A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *University of California Berkeley*.

[Chandy and Misra, 1981] Chandy, K. M. and Misra, I. (1981). Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(4):198–206.

[Chandy and Misra, 1979] Chandy, K. M. and Misra, J. (1979). Deadlock absence proofs for networks of communicating processes. *Information Processing Letters*, 9(4):185–189.

[Chow, 1996] Chow, A. C.-H. (1996). Parallel devs: A parallel, hierarchical, modular modeling formalism and its distributed simulator. *Trans. Soc. Comput. Simul. Int.*, 13(2):55–67.

[Chow and Zeigler, 1994] Chow, A. C. H. and Zeigler, B. P. (1994). Parallel devs: A parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th Conference on Winter Simulation*, WSC '94, pages 716–722, San Diego, CA, USA. Society for Computer Simulation International.

[Fujimoto, 1990] Fujimoto, R. M. (1990). Parallel discrete event simulation. *Communications of the ACM - Special issue on simulation*, 33:30–53.

[Geiger et al., 2010] Geiger, J. T., Schenk, J., Wallhoff, F., and Rigoll, G. (2010). Optimizing the number of states for hmm-based on-line handwritten whiteboard recognition. In *International Conference on Frontiers in Handwriting Recognition, ICFHR 2010, Kolkata, India, 16-18 November 2010*, pages 107–112.

[Herbez et al., 2015a] Herbez, C., Quesnel, G., and Ramat, E. (2015a). Building partitioning graphs in parallel-devs context for parallel simulations. In *Proceedings of the 2015 Spring Simulation Conference*.

[Herbez et al., 2015b] Herbez, C., Quesnel, G., and Ramat, E. (2015b). Optimization of parallel-devs simulations with partitioning techniques. In *Proceedings of the 2015 Simultech Conference*.

[Karypis and Kumar, 1998] Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392.

[Kernighan and Lin, 1970] Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307.

[Muzy and Nutaro, 2005] Muzy, A. and Nutaro, J. (2005). Devs & dsdevs abstract simulators. *1st Open International Conference on Modeling and Simulation (OICMS, Clermont-Ferrand)*, pages 273–279.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected application in speech recognition. *Proceeding of the IEEE*, 77(2):257–286.

[Zeigler et al., 2000] Zeigler, B. P., Kim, D., and Praehofer, H. (2000). *Theory of modeling and simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2nd edition.