Noname manuscript No.
(will be inserted by the editor)

DAL-SMC: Distributed Statistical Model Checking with Abstraction and Learning

Kaiqiang Jiang · Yi Ao · Ping Huang · Hui Zan · Dehui Du

Received: date / Accepted: date

Abstract The core idea of Statistical Model Checking (SMC) is to decide whether the stochastic model satisfies a given property or to evaluate its probability of satisfaction by combining statistical techniques with Monte-Carlo simulation on model traces. However, SMC still encounters performance bottleneck due to the consumption of an extremely large number of traces, each of which itself could be extremely time-consuming. To solve these problems, we have proposed an optimized SMC approach called AL-SMC to reduce the number of traces in our early work. In order to reduce the time consumption for generating a single trace further, we propose a general framework for distributed SMC based on master/slaves architecture without introducing bias. A series of algorithms are introduced to show that bayesian interval estimation (BIE) algorithm and AL-SMC algorithm are easily parallelizable on master/slaves architecture with unbiased distributed sampling. Besides, we propose parameter optimization with a genetic algorithm to reduce the statistical error of distributed AL-SMC (DAL-SMC). We also implement DAL-SMC algorithms in our ModanaOnline platform to support the automatic process. To illustrate the feasibility of our approach, three benchmarks are presented to compare the number of simulation traces, time consumption and statistical error between DAL-SMC and classic SMC algorithms. The experiment results show that the time consumption of our toolset is effectively reduced and the accuracy is ensured within an acceptable error bound.

Keywords Statistical model checking · Statistical abstraction · Learning · Distributed technology · Bayesian interval estimation algorithm

1 Introduction

Statistical Model Checking techniques (SMC)[26, 23, 15] can be seen as a trade-off between testing and formal verification. Recently, SMC has been an alternative to standard model-checking in order to avoid the state-space explosion problem, especially for verifying Cyber-physical Systems (CPSs) [25]. The core idea of SMC is to decide whether the stochastic model satisfies a given property or to evaluate its probability of satisfaction by combining statistical techniques with Monte-Carlo simulation on model traces. Nowadays, SMC is getting increasing industrial attention and there are many model checkers which support SMC techniques to analyze the stochastic model more effectively (e.g., Uppaal-SMC [5], Prism [19]).

CPS focus on the coupling of cyber part viewed as distributed computation units and physical part covering the environment affecting the running of the system. The modeling of stochastic behaviors for CPS might be highly cumbersome [2] and the analysis of these models demands extremely high confidence [11]. SMC still encounters the performance bottleneck for verifying CPS. There are two factors having a direct influence on the performance of SMC, one is the huge number of simulation traces, the other is the length of a single trace. Both of them affect the performance of SMC. In this paper, we focus on these two factors to improve the efficiency

This work was supported by NSFC (Grant No.61472140, 61202104) and NSF of Shanghai (Grant No. 14ZR1412500).

East China Normal University, Shanghai Key Laboratory of Trustworthy Computing
School of Computer Science and Software Engineering, Shanghai, China
E-mail: dhdu@sei.ecnu.edu.cn

of SMC. Figure 1 is the technology roadmap of our approach. A statistical model checker contains three components: simulator, SMC algorithm and model checker. The simulator generates simulation traces. Depending on the property language, the checker verifies the traces and returns observations (boolean or numerical values). The SMC algorithm collects observations obtained from a checker component and computes the probability. To reduce the number of simulation traces, we have proposed abstraction and learning teachique with SMC algorithm called AL-SMC [12]. Through many experiments, we find that AL-SMC effectively reduces the number of traces, thereby improving the performance of SMC. As observed in [27], SMC can be distributed based on master/slave architecture where several slave computers are used to generate simulation traces. Inspired by it, we use distributed technology with simulator to reduce the time consumption for generating a single trace. Besides, we propose the distributed Bayesian Interval Estimation (BIE) algorithm [29]. In this paper, we combine distributed technology with AL-SMC technique, which is called DAL-SMC. The DAL-SMC technique reduces both the number of simulation traces and the time consumption for generating a single trace.

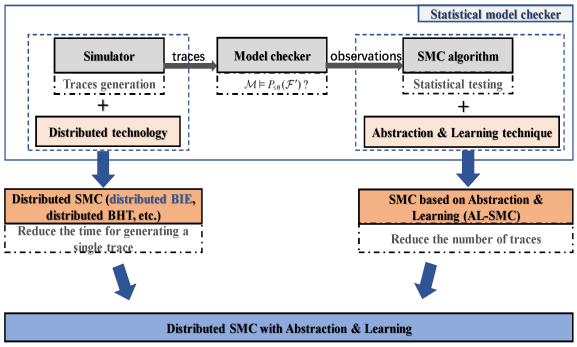


Fig. 1 Technology roadmap.

The main contributions of this paper include: (i)We propose a novel verification framework, which applies distributed technology to improve the efficiency of SMC. (ii)We propose the distributed BIE and DAL-SMC algorithms on the framework. The parameter optimization method is introduced to reduce the statistical error of DAL-SMC. This work is an extension of our previous work [12]. (iii)The distributed BIE and DAL-SMC algorithms are implemented in our ModanaOnline platform [7] (https://github.com /ECNU-MODANA/Modana-Online) to support the automatic process. (iv)Several experiments are conducted to demonstrate that our approach effectively reduces the time consumption within an acceptable error bound.

The remainder of this paper is organized as follows. In Section 2, we present the framework of our approach, several core algorithms of distributed BIE and DAL-SMC in details. Parameter optimization method of DAL-SMC is also presented. Section 3 provides the algorithm analysis of DAL-SMC. Section 4 presents our implementation in ModanaOnline platform. Several experiments are conducted with three benchmarks. The experimental results show that our DAL-SMC are efficient and feasible. The related work is discussed in Section 5. Finally, conclusions and directions of future research are presented in Section 6.

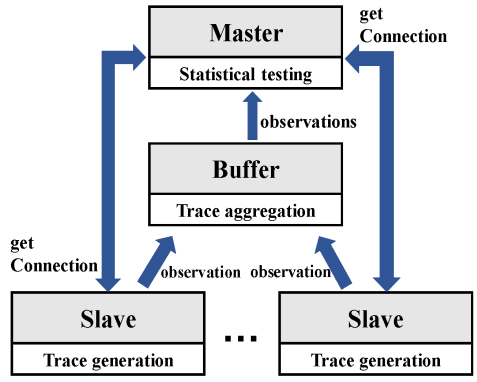
2 Distributed bayesian interval estimation and DAL-SMC

In this section, we first introduce the general framework of distributed SMC based on master/slaves architecture. Next, we apply the framework to BIE algorithm and present the core algorithms of distributed BIE. With the help of distributed BIE, the time for generating a single trace is reduced. Besides, in order to reduce the number of traces generated by distributed BIE, we propose the DAL-SMC technique and present the core algorithms of DAL-SMC, which is an improvement of distributed BIE. However, the statistical error of DAL-SMC is enlarged. To solve this problem, we propose the parameter optimization with genetic algorithm to reduce the statistical error of DAL-SMC.

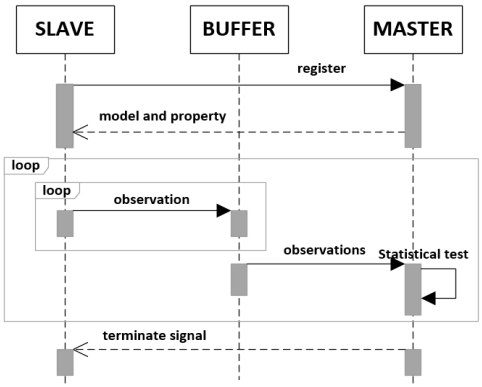
2.1 Framework of distributed SMC

SMC encounters the performance bottleneck in that the high confidence required by an answer may demand large number of traces, each of which itself may be time-consuming. Fortunately, we find that statistical methods which use independent traces are trivially parallelizable. Therefore, we can solve this problem by parallel computation based on master/slaves architecture (Figure 2(a)): multiple slave processes register their abilities to generate traces. The master process is used to collect traces and perform the statistical test. When using distributed sampling with sequential test, the number of simulation traces is unknown in advance. Therefore, it is important to avoid introducing bias when collecting the traces generated by the slave processes. To solve this problem, we adopt the method proposed in [4] which aggregates traces by batches and a buffer.

For Unified Temporal Stochastic Logic (UTSL) model checking [26], each observation involves the generation of a trajectory prefix through the discrete event simulation and the verification of a path formula over



(a) Distributed architecture



(b) Communication protocol

Fig. 2 Framework of distributed SMC.

the generated trajectory prefix. Figure 2(b) illustrates the communication protocol of distributed acceptance sampling: (i)Multiple slave processes register their abilities with the master process. The master process sends the model and property to the slaves. (ii)The slave processes generate observation and send it to the buffer which is used to aggregate observations, and then the buffer send observations to the master. (iii)The master process executes statistical test. Until the statistical test terminates, it sends terminate signal to the slaves. The framework of distributed SMC is general and flexible. Based on the framework, we present our distributed BIE algorithm in the next subsection. Actually, you can adopt the framework to any other SMC algorithm, not just limits to BIE.

2.2 Distributed bayesian interval estimation

We can compute an interval estimate of $p = Prob(M \models \phi)$ with BIE which is a faster statistical model checking algorithm based on estimation. ϕ is a Probabilistic Bounded Linear Temporal Logic (PBLTL) formula [1]. M is a Stochastic Hybrid Automata (SHA) model

Algorithm 1: Bayesian estimation algorithm

Input: BLTL property ϕ, half-interval size $\delta \in (0, 1/2)$, interval coverage coefficient $c \in (1/2, 1)$, Prior Beta distribution with parameters α,β for the (unknown) probability p that the system satisfies ϕ
Output: An interval (t_0, t_1) of width 2 δ with posterior probability at least c, estimate p' for the true probability p

$x = 0$; $n = 0$;
repeat
  σ = draw a simulation of the model;
  if $\sigma \models \phi$;
  then
    $x = x + 1$;
  end
  $n = n + 1$;
  t_0,t_1,p',γ = **CallAlgorithm2(**δ, α,β x, n**)**;
until $\gamma >= c$;

Algorithm 2: Statistical test algorithm

Input: half-interval size δ, α, β, positive trace number x, total trace number n
Output: An interval (t_0, t_1) of width 2δ, estimate probability p', posterior probability γ

$p' = (x+\alpha)/(n+\alpha+\beta)$;
$(t_0, t_1) = (p'-\delta, p'+\delta)$;
if $t_1 > 1$;
then
  $(t_0, t_1) = (1-2\delta, 1)$;
else
  if $t_0 > 0$;
  then
    $(t_0, t_1) = (0, 2\delta)$;
  end
end
$\gamma = \int_{t_0}^{t_1} f(u|x_1, ..., x_n)du$;

[10]. BIE algorithm (Algorithm 1) iteratively generates trace, verifies whether it satisfies ϕ, and then calls statistical test algorithm (Algorithm 2) to compute the estimate probability (p'), interval (t_0, t_1) of width 2δ and posterior probability (γ). Until $\gamma >= c$, the algorithm terminates and returns t_0, t_1 and p', otherwise it generates another trace and repeats.

Based on the BIE algorithm, we implement distributed BIE algorithm with the help of our distributed SMC framework. Algorithm 3 is the slave algorithm of distributed BIE. B is the size of batch which aggregates the outcomes (x) to reduce communication. The algorithm iteratively generates trace and verifies whether it satisfies ϕ. Until $runs == B$, the algorithm terminates and returns the number of positive traces ($sats$), otherwise it generates another trace and repeats.

Algorithm 4 is the master algorithm of distributed BIE. K is the size of buffer which is used to improve concurrency since the slaves can do K runs ahead of

Algorithm 3: Slave algorithm of distributed BIE

Input: BLTL property ϕ, model M, batch size B
Output: The number of positive traces $sats$
$sats = 0$, $runs = 0$;
repeat
  σ := getSimulationTrace(M);
  if $\sigma \models \phi$;
  then
    $sats$ ++;
  end
  $runs$ ++ ;
until $runs == B$;

the slowest slave, and N is the number of slaves. The main steps of master algorithm are as follows: (i)The slave processes register their abilities with master. The master sends model and property to the slaves. (ii)The master algorithm chooses a slave process randomly, and then it updates batch and buffer only if the buffer size of the slave is smaller than K. (iii)If the buffers of all slaves are not empty, the algorithm updates the value of n and x. (iv)Call algorithm 2 to compute the estimate probability (p'), interval (t_0, t_1) of width 2δ and posterior probability (γ). Until $\gamma >= c$, the algorithm terminates and returns t_0, t_1 and p', otherwise it repeats step ii, iii and iv.

2.3 Distributed AL-SMC

BIE algorithm has its shortcoming, as it needs more traces when it verifies the property whose probability is close to 0.5 [29]. We have proposed AL-SMC to solve this problem in our recent work. The framework of AL-SMC is shown in Figure 3. AL-SMC contains three main steps: (i)Simulation traces are drawn from the model and input into the abstraction process to obtain abstract traces. The abstraction process contains three steps: **property-based projection, PCA-based dimension reduction [13] and key states extraction**. (ii)**Building and optimization of Prefix Frequency Tree (PFT)** adopts the learning technique [6] to construct optimized PFT with abstract traces. By means of PFT, the original probability space is partitioned into several **sub-spaces**. (iii)**Probability evaluation via multi-BIE** invokes multiple BIE algorithms to evaluate the probabilities of sub-spaces in parallel.

AL-SMC reduces the number of traces when using BIE to verify the property whose probability is close to 0.5. Therefore, the number of traces and the time consumption for generating a single trace will be effectively reduced if we can apply the distributed SMC framework to AL-SMC. Fortunately, we find that AL-SMC

Algorithm 4: Master algorithm of distributed BIE

Input: BLTL property ϕ, model M, half-interval size $\delta \in (0, 1/2)$, interval coverage coefficient $c \in (1/2, 1)$, Prior Beta distribution with parameters α,β for the (unknown) probability p that the system satisfies ϕ, buffer size K, batch size B, the number of nodes N
Output: An interval (t_0, t_1) of width 2 δ with posterior probability at least c, estimate p' for the true probability p
$x = 0$; $n = 0$;
batch[0...N-1][0...K-1], level[0...K-1], slave[0...N-1];
for *slave[i]* $\in$ *slave[0...N]* **do**
  getConnection(slave[i]);
  sendModelandProperty(M,ϕ,slave[i]);
end
repeat
  node = random(0,N-1);
  if $level[node] < K$;
  then
    batch[node][level[node]] = **CallAlgorithm3(node)**;
    level[node]++;
  end
  if $forall(i < N) level[i] > 0$;
  then
    for $i < N$ **do**
      x += batch[i][0];
      n += B;
      level[i]–;
      batch[i][level[i]] = 0;
    end
  end
  t_0,t_1,p',γ = **CallAlgorithm2(**δ, α,β x, n**)**;;
until $\gamma >= c$;

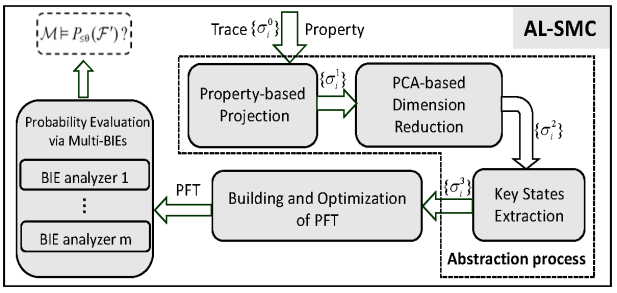


Fig. 3 The framework of AL-SMC.

can be parallelized with our distributed SMC framework. Therefore, we propose DAL-SMC, whose core algorithms are shown in Algorithm 5 and Algorithm 6.

Algorithm 5 is the master algorithm of DAL-SMC, where SN is the number of sample traces which are used to construct PFT. The master algorithm contains the following steps: (i)The master process generates a set of sample traces $\sigma[0...SN-1]$ as inputs into abstraction process to obtain a set of abstract traces $\sigma'[0...SN-1]$, and then the optimized PFT is constructed with

$\sigma'[0...SN-1]$. (ii)The slave processes get connection with the master. The master process sends model, property and **PFT** to the slaves. (iii)The algorithm chooses a slave randomly, and then updates $batch[i][j]$ only if the buffer size of this slave is smaller than K. (iv)If the buffers of all slaves are not empty, the algorithm evaluates the probability with multi-BIE and obtains $p'[j]$ and $\gamma[j]$ of each BIE, if $\gamma[j] > c$, the jth BIE terminates. (v)Until all BIEs terminate, the algorithm computes the estimation probability with $p' := \sum_{i=0}^{S-1} p[i]$, otherwise it repeats step iii, iv and v.

Algorithm 6 is the slave algorithm of DAL-SMC. T is the optimized PFT which is constructed in master algorithm. d is the number of leaf nodes in optimized PFT, i.e., the number of sub-spaces. $sats[i]$ is the number of satisfying traces in ith sub-space. The slave process iteratively generates trace (σ) which is the input of abstraction process to obtain the abstract trace (σ'). If σ' satisfies ϕ, the algorithm finds a sub-space with σ' and T. Until $runs == B$, the algorithm terminates and returns $sats[0...S-1]$, or otherwise it generates another trace and repeats.

DAL-SMC is more efficient than distributed BIE, and the advantages of DAL-SMC are obvious. Firstly, the slave process of DAL-SMC generates less traces, particularly in verifying the property whose probability is close to 0.5. Secondly, the master of DAL-SMC evaluates the probability with multiple BIEs, thus the statistic process is faster than that of distributed BIE algorithm. In the remainder of the paper, we focus on the performance analysis of DAL-SMC. However, DAL-SMC has bigger statistical error due to the usage of multi-BIE in statistic process. In the next subsection, we will introduce our parameter optimization method to reduce the statistical error of DAL-SMC.

Algorithm 5: Master algorithm of DAL-SMC

Input: BLTL property ϕ, model M, half-interval size $\delta \in (0, 1/2)$, interval coverage coefficient $c \in (1/2, 1)$, Prior Beta distribution with parameters α, β for the (unknown) probability p that the system satisfies ϕ, buffer size K, batch size B, the number of nodes N, the number of sample traces SN

Output: Prefix frequency tree T, estimate p' for the true probability p

```
σ[0...SN − 1] = getSampleTraces(M);
σ'[0...SN − 1] = abstractTraces(σ[0...SN − 1]);
T = ConstructPFT(σ'[0...SN − 1]);
d = getSubSpacesNum(T);
for slave[i] ∈ slave[0...N] do
    getConnection(slave[i]);
    sendModelandPropertyandPFT(M,φ,T,slave[i]);
end
x[0...d-1], n;
batch[0...N-1][0...K-1][0...d-1], level[0...K-1],
slave[0...N-1];
repeat
    node = random(0,N-1);
    if level[node] < K ;
    then
        batch[node][level[node]][0...d-1] =
        CallAlgorithm6(node);
        level[node]++;
    end
    if forall(i < N)level[i] > 0 ;
    then
        for i < N do
            x[0...d-1] += batch[i][0][0...d-1];
            n += B;
            level[i]–;
            batch[i][level[i]] = Φ;
        end
    end
    ter[0...d-1] = false;
    j = findUnTerminateBIE(ter[0...d-1]);
    p'[j],γ[j] = CallAlgorithm2(δ, α,β x[j], n);;
    if γ[j] > c;
    then
        ter[j] = true ;
    end
until forall(ter[i] ∈ ter[0...d-1])==true;
p' = Σ_{i=0}^{d−1} p'[i];
```

2.4 Parameter optimization of DAL-SMC

In order to reduce the statistical error of DAL-SMC, we introduce two parameter optimization methods: r **optimization** is used to partition the original probability space more evenly, and δ **prediction** is used to predict half-interval size of each BIE.

r **optimization**: In DAL-SMC, we partition the original probability space into several sub-spaces by building and optimizing the PFT. Note that the number of sub-spaces is determined by the leaf nodes of optimized PFT. Therefore, we need to input the reduction degree of PFT (r) into the optimization algorithm to generate optimized PFT. However, the value of r is difficult to determine. On one hand, if the value of r is large, we will obtain a large number of sub-spaces and the error of the algorithm will be enlarged. On the other hand, if the value of r is small, we will obtain few sub-spaces and the efficiency of the algorithm will be reduced. To solve the problem, the optimized value of r is computed with the following formula, which is defined with the genetic algorithm [24].

$$r_k = M - \sum_{i=1}^{k} \sum_{j=1}^{k} |T_i - T_j| \quad (1)$$

Suppose k is the population quantity, i.e., the number of sub-spaces of each generation. T_i is the number

Algorithm 6: Salve algorithm of DAL-SMC

Input: BLTL property ϕ, model M, batch size B, prefix frequency tree T, the number of sub-spaces d
Output: The number of positive traces in sub-spaces $sats[0...d-1]$

```
sats[0...d − 1], runs = 0;
repeat
    σ = getSimulationTrace(M);
    σ′ = abstractTrace(σ);
    if σ ⊨ φ;
    then
        i = findSubSpace(σ′, T);
        sats[i] ++;
    end
    runs ++ ;
until runs == B;
```

of satisfying traces in *i-th* sub-space. In order to get the optimized value of r, we need to partition the original probability space more evenly and ensure the number of sub-space is a moderate value. Therefore, we adopt formula 1 as evaluation function and $k \geq 6 \&\& k \leq 10$ as constraint function. M is a large positive number, if we partition the original probability space more evenly, the value of $\sum_{i=1}^{k}\sum_{j=1}^{k}|T_i - T_j|$ is smaller. Thus, we obtain the optimized value of r which makes maximum M.

δ **prediction**: Through building and optimizing of PFT, we obtain several sub-spaces. Then we execute BIE algorithm on each sub-space, and each BIE has its half-interval size and interval coverage coefficient. In algorithm 6, we suppose all BIEs have the same δ and c. However, the probabilities of some sub-spaces may have a big difference. For instance, we suppose the half-interval size of all sub-spaces is 0.1, but there may be a sub-space whose probability is less than 0.1, thus the half-interval size of this sub-space is unreasonable. It will lead to large statistical error. To solve this problem, we use the leaf nodes of PFT to predict the half-interval size of *m-th* BIE (δ_m) as shown in formula 2. Note that k is the number of sub-spaces, T_m is the number of satisfying traces in *m-th* sub-space, and N_i is the number of total traces in *i-th* sub-space. η is half-interval rate, the bigger η is, the more precise the probability is. By means of formula 2, the half-interval size of each BIE is more accurate.

$$\delta_m = T_m / \sum_{i=1}^{k} N_i / \eta \tag{2}$$

Table 1 The time and space complexity of DAL-SMC.

Algorithm phase	Time	Space
Trace abstraction	$O(mn)+O(min(k^3, n^3))$	$O(k^2)$
PFT construction	$O(mn)+O(d \log d))$	$O(\log d)$
Probability evaluation	$O(B * (\log d + i))$	$O(1)$

3 Algorithm Analysis of DAL-SMC

3.1 Time and space complexity

We analyse the time and space complexity of DAL-SMC as shown in Table 1.

(i)The time and space complexity of abstraction process are $O(mn)+O(min(k^3, n^3))$ and $O(k^2)$ respectively, where m denotes the length of the trace, n denotes the number of sampling traces and k denotes the dimension of each trace.

(ii)The time and space complexity of PFT construction are respectively $O(mn) + O(d \log d))$ and $O(\log d)$, where d denotes the number of leaf nodes in PFT.

(iii)Suppose the iterations of BIE algorithm is B, the time complexity of each statistical test iteration is $O(i)$, and the time of searching leaf node is $\log d$. Therefore, the time complexity of probability evaluation is $O(B * (\log d + i))$.

3.2 Error bound

Models in step i and ii of Algorithm 5 are probabilistically equivalent in terms of a certain property. Therefore, the statistical error is only generated in probability evaluation. We use multiple BIEs to evaluate the probablity in DAL-SMC, therefore the error is enlarged compared with distributed BIE. Suppose the half-interval size of each BIE is δ and the number of BIEs is M, thus the error of multi-BIE is less than $\delta * M$. Besides, we use δ prediction method in section 2.4 to reduce the statistical error. Consequently, the error (ξ) of DAL-SMC with parameter optimization satisfies the formula 3.

$$|\xi| \leq \sum_{m=1}^{M} (T_m / \sum_{i=1}^{k} N_i / \eta) \tag{3}$$

4 Implementation and Benchmark Experiments

The distributed SMC framework has been implemented in our ModanaOnline platform which is an online platform for modeling and analysis CPS. We have implemented the core algorithms of distributed BIE and DAL-SMC in this platform. To illustrate the feasibility of our approach, we explore some experiments with

three benchmarks: train-gate [9], energy-aware building [8] and robots path planning [22].

4.1 DAL-SMC implementation

In our previous work, we have implemented Modana platform which is an integrated modeling and verification environment for CPS [7]. Further, we implement the online version of modana, called ModanaOnline, in which the distributed BIE and DAL-SMC are implemented. ModanaOnline is a web project, whose back end is implemented in Java based on SpringMVC, Spring and Mybatis. The front end is implemented in JavaSc ript based on AngularJS which supports information transmission of distributed framework with web service. Figure 4 is the user interface of DAL-SMC in ModanaOnline. Users can import the model files(such as uppaal [3] and prism [19]) and the property to verify the model. Besides, there are several parameters (v, t, s, n, η, c) should be customized, in which (v, t, s) are used in abstraction and leaning phase, (n, η, c) are used in probability evaluation where,

(i)v denotes the number of sample traces in abstract process, t denotes the threshold in PCA-based dimension reduction and s denotes the number of extracted states during the key states extraction.

(ii)n denotes the number of slaves, η denotes the half-interval rate in Section 2.4, and c denotes the interval coverage coefficient of BIE algorithm.

Then users can verify the property with DAL-SMC and generate a bar graph as shown in Figure 4. Users can observe the distribution of traces more obviously with the bar graph.

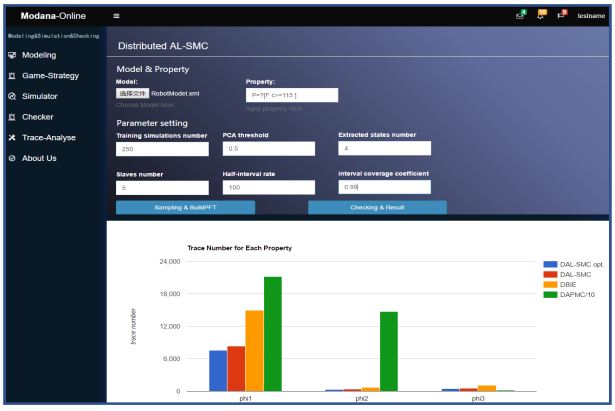


Fig. 4 The user interface of DAL-SMC.

4.2 Benchmark experiments

Uppaal-SMC [5] is a new version of UPPAAL which supports SMC and adopts many SMC algorithms (B-HT [17],SPRT [28], BIE [29], APMC [16] etc.). In this paper, we model the benchmark with Uppaal-SMC, and then import the model (.xml) into our platform to verify the properties. Here, we use a simple benchmark to exhibit the distribution of traces in each slave and compare the probability partition between DAL-SMC with parameter optimization (DAL-SMC opt.) and DAL-SMC. Finally, two complex benchmarks are introduced to compare the efficiency and accuracy among distributed BIE, DAL-SMC, DAL-SMC opt. and distributed APMC (DAPMC) which is a classical quantitative SMC algorithm.

4.2.1 Train-gate

A number of trains are approaching a gate on which there is only one track, gate controls the trains to avoid collisions, and restarts them when it is possible to make sure that trains will eventually cross the gate. Figure 5(a) is the template of the train. Trains delay according to an exponential distribution and synchronize with the gate. The gate keeps track of the trains with an internal queue data structure as show in Figure 5(b). We use Formula 4 to evaluate the probability of Train(0) crossing the gate with PBLTL property:

$$P_{=?}(F^{\leq 100}\ Train(0).Cross) \tag{4}$$

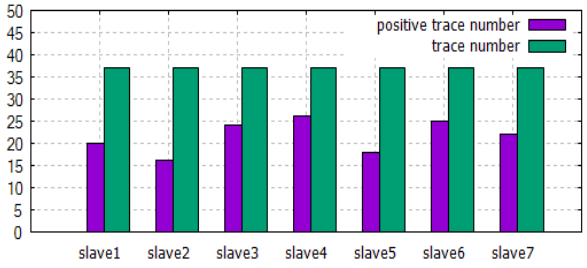
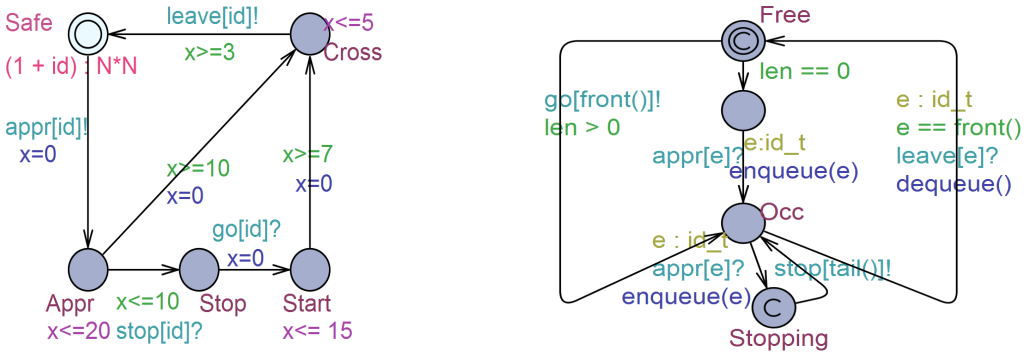


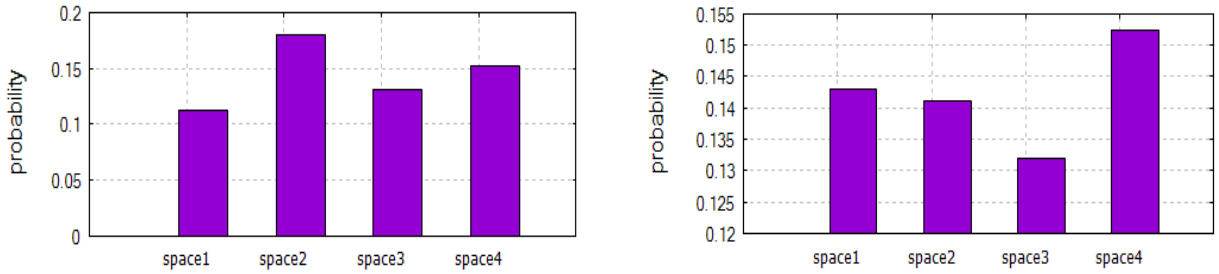
Fig. 6 Trace number of each slave

The evaluation result is in [0.52707,0.626969] with half-interval size of ±0.05 and interval coverage coefficient of 95%. Further, we plot the number of traces and satisfying traces of each slave in Figure 6. It shows that the slaves generate same number of traces, however, the number of satisfying traces are not equal. Besides, in order to compare the probability partition between DAL-SMC and DAL-SMC opt, we plot the probability of each sub-space as shown in Figure 7. Figure 7(a) and Figure 7(b) are the probability of each sub-space in DAL-SMC and DAL-SMC opt. respectively. We find



(a) Train (b) Gate

Fig. 5 The SHA templates for train-gate.



(a) DAL-SMC (b) DAL-SMC opt

Fig. 7 Probability of each sub-space.

that the probability distribution of DAL-SMC opt. is more evenly than that of DAL-SMC, therefore, the parameter optimization is effective. In the next subsections, we will use two complex CPS benchmarks to compare the performance of algorithms.

4.2.2 Energy-aware building

Energy-aware buildings play an important role in achieving an energy efficient society. The goal is to evaluate and compare the energy consumption of various control strategies with varying environmental settings. There are five SHA models: *room temperature, heater, controller, weather* and *user profile*. Figure 8 shows the main SHA templates for energy-aware building. In Figure 8(a), the room needs to be heated when the temperature is lower than a threshold. The heater moves between locations "Off" and "On" based on the temperature thresholds $on[r]$ and $off[r]$, where r denotes the number of heated room. The central controller decides how to move the heater from one room to another room as shown in Figure 8(c).

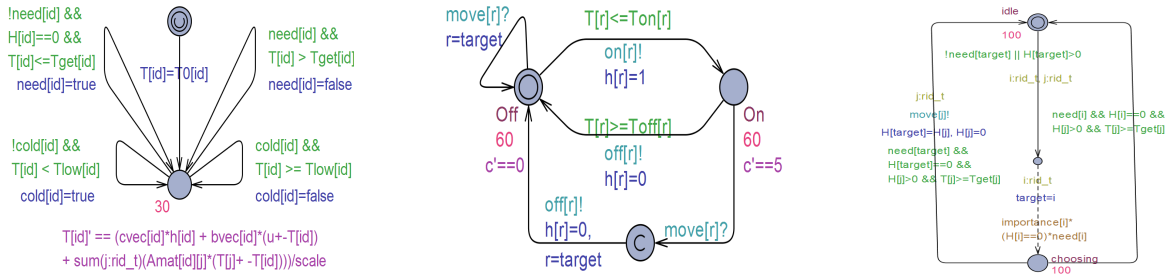
The experiment has been performed on five slaves on a cluster with Intel Core(TM) i7-4790 (octa-cores at 3.6GHz) interconnected with infiniband. To compare the effciency and accuracy of the algorithms, three properties are verified, which are listed in Table 2. δ and c denotes half-interval size and interval coverage coefficient respectively.

Table 2 Properties of energy-aware building

PID	(δ, c)	Property
ϕ_1	$(0.05, 0.99)$	$P_{=?}(F^{\leq 48}\ energy \geq 210)$
ϕ_2	$(0.01, 0.99)$	$P_{=?}(F^{\leq 48}\ discomfort \geq 15)$
ϕ_3	$(0.02, 0.9)$	$P_{=?}(F^{\leq 48}\ discomfort \leq 15 \wedge energy \geq 170)$

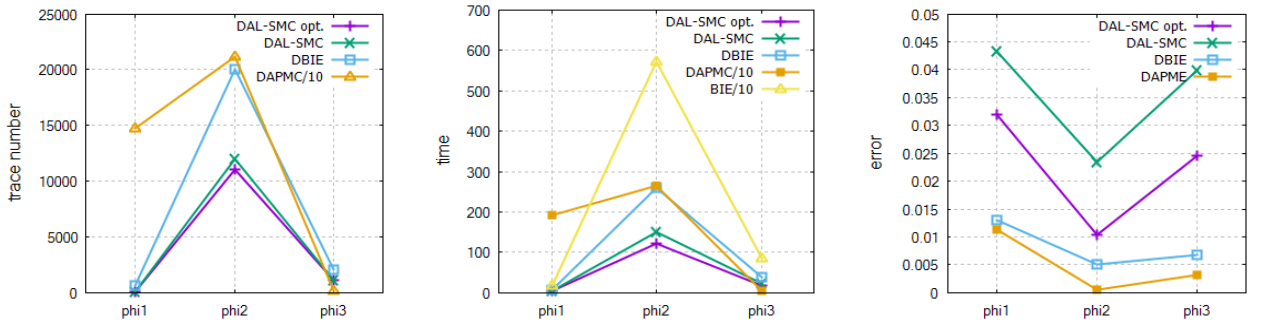
We execute each algorithm many times, and compare them in three aspects: the number of traces, time consumption and statistical error. In order to analyse the statistical error of algorithm, we verify the property with high interval coverage coefficient and small half-interval size to obtain the probability (P_r). We suppose P_r is the true probability, therefore, the statistical error of each algorithm is $P_r - P_a$, where P_a is the estimation of the true probability. Figure 9 shows the number of traces, time consumption and statistical error of each algorithm.

Figure 9(a) shows the trace number of energy-aware building benchmark generated by each algorithm. We can find that DAPMC algorithm needs 200000 traces to verify property ϕ_2, however, distributing BIE (DBIE)



(a) Room (b) Heater (c) Controller

Fig. 8 The main SHA templates for energy-aware building.



(a) Trace number of each algorithm (b) Verification time of each algorithm (c) Statistical error of each algorithm

Fig. 9 Algorithms comparison with energy-aware building.

algorithm only needs 20000 traces. Further, DAL-SMC and DAL-SMC opt. need less traces (about 10000). The time consumption of each algorithm is shown in Figure 9(b). The time consumption for verifying property ϕ_2 with BIE algorithm is 6000 seconds, while it is around 250 seconds consumed by distributed BIE. DAL-SMC and DAL-SMC opt. consume less time. Figure 9(c) is the statistical error of each algorithm. The statistical error of DAPMC and distributed BIE are about 0.013 for verifying property $\phi 1$. The statistical error of DAL-SMC and DAL-SMC opt are about 0.045 and 0.032 respectively. The detailed experiment results are listed in Table 4.

4.2.3 Robots path planning

Analysis of robots path planning has attracted public attention these years [20]. The basic goal of a mobile robot is to avoid collision with the moving obstacles, which conduct irregular movement around the environment. As soon as the robot observes the moving obstacle, it will take actions immediately. While different actions will result in different energy cost, we add some variables to monitor the total energy consumption. Figure 10 shows the main SHA templates for robot path planning.

Table 3 Properties in robots path planning

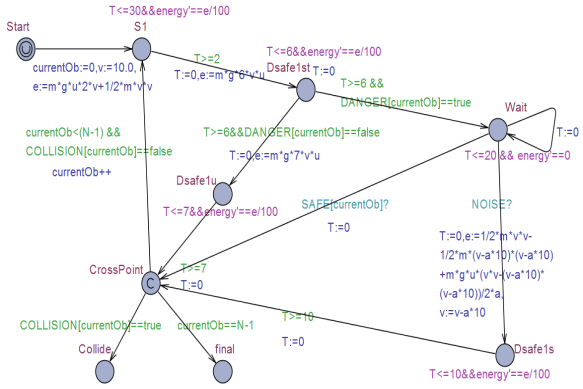
PID	(δ, c)	Property
ϕ_4	$(0.01, 0.99)$	$P_{=?}(F^{\leq 100}\ robot.collision)$
ϕ_5	$(0.05, 0.99)$	$P_{=?}(F^{\leq 100}\ energy \geq 500)$
ϕ_6	$(0.02, 0.9)$	$P_{=?}(F^{\leq 100}\ robot.collision \wedge energy \geq 500)$

We also verify three properties (ϕ_4, ϕ_5, ϕ_6) for this benchmark as shown in Table 3. The experiment results are shown in Figure 11.

In order to analyse the performance of SMC algorithms, the detailed data is listed in Table 4. Property ϕ_2 is used to illustrate the experimental results, we can conclude that:

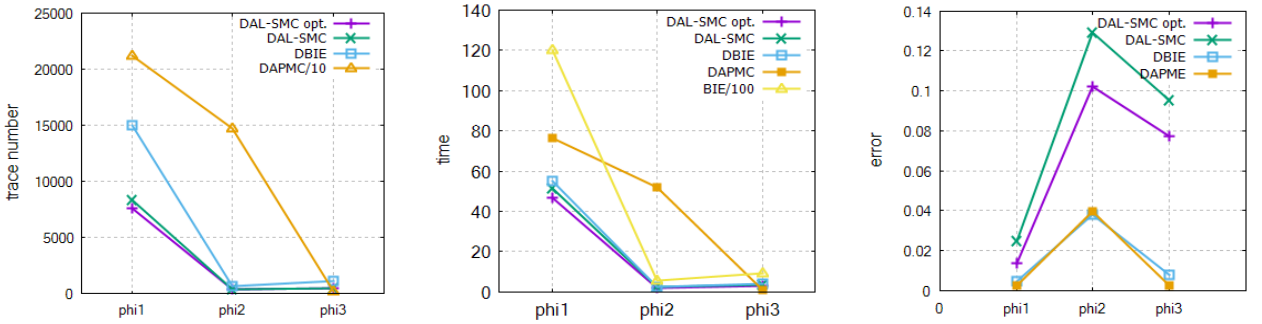
(i)DAPMC generates the most traces for verifying the property, while distributed BIE needs less traces compared with DAPMC. Furthermore, DAL-SMC reduces the number of traces (about 50%) relative to distributed BIE.

(ii)The main time consumption of verification is generating traces (more than 90%). DAPMC consumes most time, and DAL-SMC consumes less time compared with distributed BIE profitting from less traces. For the benchmark, we use 40 cores to implement distributed algorithms, and we can find that the time consumption of



(a) Robot (b) Obstacle

Fig. 10 The main SHA templates for robot path planning.



(a) Trace number of each algorithm (b) Verification time of each algorithm (c) Statistical error of each algorithm

Fig. 11 Algorithms comparison with robots path planning.

BIE algorithm is nearly 25 times as much as distributed BIE which shows it is effective to improve the performance of SMC with distributed techniques.

(iii)The statistical error of DAPMC and DBIE are similar. The error of DAL-SMC is bigger than that of DAPMC and DBIE due to use multi-BIE. Besides, we can find that the error of DAL-SMC opt. is less than that of DAL-SMC. It proves the parameter optimization method is effective.

In general, DAL-SMC opt. generates less traces and consumes less time for verification with the help of distributed technology and AL-SMC technique. The parameter optimization method reduces the statistical error of DAL-SMC to an acceptable range.

5 Related Works

Statistical Model Checking technique was first proposed by R.Grosu [14]. Some variations [21] [26] [28][17] [29] [15] based on the basic SMC have been proposed in the past few years. Some related work are summarized as follows:

Basic SMC. SMC refers to a series of simulation-based techniques that can be used to answer two questions: (1)Qualitative: Is the probability of model (s) satisfying property (ϕ) greater than or equal to a certain threshold? and (2)Quantitative: What is the probability of model (s) satisfying property (ϕ)? For qualitative SMC, Kim et al. [18] have given an empirical evaluation. BHT and SPRT are more effective than SSP. BHT generates more traces when checking the property whose estimation probability is close to its real probability, thus SPRT is faster than BHT in this situation, otherwise BHT is obviously more efficient than SPRT. For quantitative SMC, Zuliani et al. [29] have compared the number of traces analyzed by APMC and BIE, and they have concluded that BIE excels remarkably in performance. Our approach focuses on the performance of BIE algorithm.

SMC with abstraction and learning. BIE algorithm needs more traces when checking the property whose probability is close to 0.5, while the number of traces is drastically reduced when the probability approaches to 0 or 1 [29]. In our recent work [12], we have

Table 4 Experimental results

Algorithm	Property	Trace Number	Time consumption	Statistical Error
DAPMC	$\phi1(0.05, 0.99)$	147000	1934.31	0.0113
	$\phi2(0.01, 0.99)$	**211932**	**2649.15**	**0.0005**
	$\phi3(0.02, 0.9)$	1950	38.05	0.0032
	$\phi4(0.01, 0.99)$	211930	76.282	0.0024
	$\phi5(0.05, 0.99)$	147550	52.206	0.0399
	$\phi6(0.02, 0.9)$	1850	1.159	0.0026
DBIE	$\phi1(0.05, 0.99)$	600	7.895	0.0131
	$\phi2(0.01, 0.99)$	**20000**	**259.275**	**0.0051**
	$\phi3(0.02, 0.9)$	2100	38.057	0.0068
	$\phi4(0.01, 0.99)$	15000	55.281	0.0049
	$\phi5(0.05, 0.99)$	702	2.483	0.0382
	$\phi6(0.02, 0.9)$	1120	4.226	0.0078
DAL-SMC	$\phi1(0.05, 0.99)$	103	5.685	0.0433
	$\phi2(0.01, 0.99)$	**12000**	**151.785**	**0.0235**
	$\phi3(0.02, 0.9)$	1137	22.841	0.0399
	$\phi4(0.01, 0.99)$	8318	41.68	0.0246
	$\phi5(0.05, 0.99)$	384	2.32	0.1295
	$\phi6(0.02, 0.9)$	520	3.601	0.0751
DAL-SMC opt.	$\phi1(0.05, 0.99)$	95.79	4.65	0.0319
	$\phi2(0.01, 0.99)$	**11040**	**121.425**	**0.0103**
	$\phi3(0.02, 0.9)$	1091	18.73	0.0246
	$\phi4(0.01, 0.99)$	7569	36.512	0.0138
	$\phi5(0.05, 0.99)$	349	1.81	0.102
	$\phi6(0.02, 0.9)$	494	3.171	0.0575
BIE	$\phi1(0.05, 0.99)$	590	175.44	0.0121
	$\phi2(0.01, 0.99)$	**19586**	**5730.67**	**0.0047**
	$\phi3(0.02, 0.9)$	2040	845.73	0.0063
	$\phi4(0.01, 0.99)$	15400	1202.467	0.0044
	$\phi5(0.05, 0.99)$	762	55.221	0.0352
	$\phi6(0.02, 0.9)$	1150	91.98	0.0069

partitioned the original probability space (Ω) into many sub-spaces Ω_1,... ,Ω_m, and evaluated the probability of each sub-space in parallel. Therefore, the trace number for evaluating the original probability will be decreased and depends on the maximum number of traces for evaluating sub-spaces theoretically. We find that the number of traces is effectively reduced while ensuring the accuracy of the probability within an acceptable error bound.

Distributed SMC. As observed in [27], SMC algorithms can be distributed with master/slave architecture where multiple slave processes are used to generate traces. When working with an estimation algorithm, the number of traces for verifying the property is known in advance and can be equally distributed between the slaves. When working with the sequential algorithms, the situation gets more complicated, so we need to avoid introducing bias when collecting the traces generated by the slave processes. To solve this problem, H. L. S. Younes proposed a method in [26] where the bias is avoided by committing ,α *priori*, to the order in which observations will be taken into account. Peter Bulychev et al. generalized the above method with batches and buffer [4]. Batches aggregate the outcomes for reducing communication and the buffer is used to improve concurrency since the nodes are more loosely synchronized. They also implemented the distributed Hypothesis testing algorithm without introducing bias. The algorithm effectively reduce the time consumption for generating a single trace. Our work is different from the existing work, we use abstraction and learning technique (AL-SMC) to reduce the number of simulation traces, and adopt distributed technology with AL-SMC to reduce both the number of traces and time consumption for generating a single traces.

6 Conclusions

In this paper, we focus on improving the performance of SMC with distributed technology. When we analyse the complex CPSs with SMC, huge-sized models and high confidence of probability estimation require a great number of traces. It is extremely time-consuming. To solve this problem, we present a distributed framework for SMC without introducing bias, and propose distributed BIE and DAL-SMC algorithms based on the framework. The framework is implemented in ModanaO nline platform which is an online platform for modelling and analysis of CPSs. Besides, the parameter optimization methods are proposed to reduce error and improve

efficiency of DAL-SMC. Finally, to illustrate the feasibility of our approach, three benchmarks are presented. The experiment results show that the improved DAL-SMC is more efficient than other SMC algorithms. The parameter optimization method reduces statistical error effectively.

As part of future work, in addition to focusing on the effciency of our implementation, we plan to apply our framework to more complex systems. Furthermore, we will improve our tool to make it more extendable.

Acknowledgements This work was supported by NSFC (Grant No.61472140, 61202104) and NSF of Shanghai (Grant No. 14ZR1412500).

References

1. Baier, C., Katoen, J.P., et al.: Principles of model checking, vol. 26202649. MIT press Cambridge (2008)
2. Basu, A., Bensalem, S., Bozga, M., Caillaud, B., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. In: Formal Techniques for Distributed Systems, pp. 32–46. Springer (2010)
3. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: International Conference on the Quantitative Evaluation of Systems, pp. 125–126 (2006)
4. Bulychev, P., David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Checking and distributing statistical model checking. Lecture Notes in Computer Science **7226**, 449–463 (2012)
5. Bulychev, P., David, A., Larsen, K.G., Mikučionis, M.: Uppaal-smc: Statistical model checking for priced timed automata. Electronic Proceedings in Theoretical Computer Science **85** (2012)
6. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Grammatical Inference and Applications, pp. 139–152. Springer (1994)
7. Cheng, B., Wang, X., Liu, J., Du, D.: Modana: An integrated framework for modeling and analysis of energy-aware cpss. In: IEEE Computer Software and Applications Conference, pp. 127–136 (2015)
8. David, A., Du, D., Larsen, K.G., Mikučionis, M., Skou, A.: An evaluation framework for energy aware buildings using statistical model checking. Science China information sciences **55**(12), 2694–2707 (2012)
9. David, A., Larsen, K.G., Legay, A., Miku, Ionis, M., Poulsen, D.B., gsted: Uppaal smc tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397–415 (2015)
10. David, A., Larsen, K.G., Legay, A., Poulsen, D.B.: Statistical model checking of dynamic networks of stochastic hybrid automata. Francisco Javier Fuente Fernandez **66**, 91–104 (2014)
11. Dehui Du Bei Cheng, J.L.: Statistical model checking for rare-event in safety-critical system. Journal of Software (2), 305–320 (2015)
12. Dehui Du Ping Huang, K.J.: Al-smc: Optimizing statistical model checking by automatic abstraction and learning. International Journal of Software and Informatics **10**, 4:0 (2016)
13. Dunteman, G.H.: Principal components analysis, vol. 69. Sage (1989)
14. Grosu, R., Smolka, S.A.: Monte carlo model checking. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 271–286. Springer (2005)
15. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Verification, Model Checking, and Abstract Interpretation, pp. 73–84. Springer (2004)
16. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Verification, Model Checking, and Abstract Interpretation, pp. 73–84. Springer (2004)
17. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Computational Methods in Systems Biology, pp. 218–234. Springer (2009)
18. Kim, Y., Kim, M., Kim, T.H.: Statistical model checking for safety critical hybrid systems: An empirical evaluation. In: Hardware and Software: Verification and Testing, pp. 162–177. Springer (2012)
19. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic symbolic model checker. Lecture Notes in Computer Science **2324**, 200–204 (2002)
20. Lahijanian, M., Wasniewski, J., Andersson, S.B., Belta, C.: Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees. In: Proc. 2010 IEEE International Conference on Robotics and Automation, pp. 3227–3232 (2010)
21. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Runtime Verification, pp. 122–135. Springer (2010)
22. Miura, J., Shirai, Y.: Modeling motion uncertainty of moving obstacles for robot motion planning. In: IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA, pp. 2258–2263 vol.3 (2000)
23. Sen, K., Viswanathan, M., Agha, G.: Statistical Model Checking of Black-Box Probabilistic Systems. Springer Berlin Heidelberg (2004)
24. Sivanandam, S.N., Deepa, S.N.: Introduction to genetic algorithms. Springer (2008)
25. Yoo, H., Shon, T.: Challenges and research directions for heterogeneous cyber physical system based on iec 61850: Vulnerabilities, security requirements, and security architecture. Future Generation Computer Systems **61**, 128–136 (2016)
26. Younes, H., Kan, L.S.: Planning and verification for stochastic processes with asynchronous events. In: Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, Usa, pp. 1001–1002 (2004)
27. Younes, H.L.: Ymer: A statistical model checker. In: Computer Aided Verification, pp. 429–433. Springer (2005)
28. Younes, H.L., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Information and Computation **204**(9), 1368–1409 (2006)
29. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to stateflow/simulink verification. Formal Methods in System Design **43**(2), 338–367 (2013)