Randomized Reachability Analysis in Uppaal: Fast Error Detection in Timed Systems*

Andrej Kiviriga, Kim Guldstrand Larsen, and Ulrik Nyman

Aalborg University, Selma Lagerløfs Vej 300, 9220 Aalborg, Denmark
`{kiviriga, kgl, ulrik}@cs.aau.dk`

Abstract. We introduce Randomized Reachability Analysis – an efficient and highly scalable method for detection of "rare event" states, such as errors. Due to the under-approximate nature of the method, it excels at quick falsification of models and can greatly improve the model-based development process: using lightweight randomized methods early in the development for the discovery of bugs, followed by expensive symbolic verification only at the very end. We show the scalability of our method on a number of Timed Automata and Stopwatch Automata models of varying sizes and origin. Among them, we revisit the schedulability problem from the Herschel-Planck industrial case study, where our new method finds the deadline violation three orders of magnitude faster: some cases could previously be analyzed by statistical model checking (SMC) in 23 hours and can now be checked in 23 seconds. Moreover, a deadline violation is discovered in a number of cases that where previously intractable. We have implemented the Randomized Reachability Analysis – and made it available – in the tool UPPAAL.

Keywords: model-checking · randomized · state-space explosion · schedulability analysis · timed automata · stopwatch automata.

1 Introduction

Formal verification of system designs in the form of model checking requires that reliable formal models of a system are created. Apart form the ability to verify formal queries, many model checking tools also give the modeller access to a simulator in order to understand the model behavior. Throughout the process of developing the models, a number of sanity queries can be used in the same way as unit tests in software development. Verifying these queries repeatedly between each addition to the model can be prohibitively time consuming, especially for complex systems that often grow large and become difficult to analyze. In this paper we present a solution to this problem.

*Supported by the ERC Advanced Grant Project: LASSO: Learning, Analysis, Synthesis and Optimization of Cyber-Physical Systems, and by the Villum Investigator project S4OS: Synthesis of Safe, Small, Secure and Optimal Strategies for Cyber-Phyiscal Systems.

The main contribution of this paper is the implementation of randomized reachability analysis in the tool UPPAAL. Randomized reachability analysis is a non-exhaustive efficient technique for the detection of "rare event" states, such as errors. The work is a continuation of [13] where similar randomized analysis was applied to refinement checking. The method can analyse Timed Automata and Stopwatch Automata models with the features already supported by UPPAAL. The randomized approach is based on repeated exploration of the model by means of *random walks* and was inspired by [10]. It explores the state-space in a light and under-approximate manner; hence, it can only perform conclusive verification when a single trace can demonstrate a property. However, our randomized method excels at reachability checking and in many cases outperforms existing model-checking techniques by up to several orders of magnitude. The benefits are especially notable in large systems where traditional model-checking is often intractable due to the state-space explosion problem. Randomized reachability analysis is particularly useful for an *efficient development process*: running cheap, randomized methods early in the development to discover violations and performing an expensive and exhaustive verification at the very end. Randomized reachability analysis supports the search for *shorter* traces which improves the usability of discovered traces in debugging the model. We have implemented randomized reachability analysis – and made it available – in the tool UPPAAL [1].

Timed Automata models can also be used in the domain of schedulability, which deals with resource management of multiple applications ranging from warehouse automation to advanced flight control systems. Viewing these systems as a collection of tasks, schedulability analysis allows to optimize usage of resources, such as processor load, and to ensure that tasks finish before their deadline. A traditional approach in preemptive priority-based scheduling is that of the worst-case response time (WCRT) analysis [12,5]. It involves estimating worst case scenarios for both the execution time of a task and the blocking time a task may have to spend waiting for a shared resource. Apart from certain applicability limitations, classical response time analysis is known to be over-approximate which may lead to pessimistic conclusions in that a task may miss its deadline, even if in practice such a scenario could be unrealizable. Model-based approach is a prominent alternative for verification of schedulability [2,3,4,16] as it considers such parameters as offsets, release times, exact scheduling policies, etc. Due to this, the model-based approach is able to provide a more exact schedulability analysis.

We continue the effort in using a model-based approach and the model checker UPPAAL to perform a Stopwatch Automata based schedulability analysis of systems [6]. Specifically, we re-revisit the industrial case study of the ESA Herschel-Planck satellite system [16,8]. The Danish company Terma A/S [19] developed the control software and performed the WCRT analysis for the system. The case we analyse consists of 32 individual tasks being executed on a single processor with the policy of fixed priority preemptive scheduling. In addition, a combination of priority ceiling and priority inheritance protocols is used, which in essence makes the priorities dynamic. Preemptive scheduling is encoded in the model with the help of stopwatches which allow to track the progress of each task and stop it when the task is preempted. In UPPAAL, existing symbolic reachability analysis for models with stopwatches is over-approximate, which may provide spurious traces. In such models, our randomized reachability analysis allows to obtain exact, non-spurious traces to target states.

In the previous work of [16] the schedulability of Herschel-Planck was "successfully" concluded, but with an unrealistic assumption of each task having a fixed execution time (ET). To improve on this, the analysis of [8] was carried out with each of the tasks given a non-deterministic execution time in the interval of [WCET, BCET]. Unfortunately, interval based execution times, preemption and shared resources that impose dependencies between tasks, makes schedulability of systems like Herschel-Planck undecidable [9].

Even in the presence of unschedulability, two model-checking (MC) techniques were used in [8] to either verify or disprove schedulability for certain intervals of possible task execution times. First, the symbolic, zone-based, MC was used. Even though for stopwatch automata it is implemented as an over-approximation in UPPAAL which still suffices for checking of safety properties, e.g. if the deadline violation can never be reached. However, this technique cannot be used to disprove schedulability of the system as resulting traces may possibly be spurious. Second, the statistical model-checking (SMC) technique was used to provide concrete counterexamples witnessing unschedulability of the model in cases where symbolic MC finds a potential deadline violation and cannot conclude on schedulability. The idea of SMC [20,15] is to run multiple *sample traces* from a model and then use the traces for statistical analysis which, among all, estimates the probability of a property to be satisfied on a random run of a model. The probability estimate comes with some degree of confidence that can be set by the user among a number of other statistical parameters. Several SMC algorithms that require stochastic semantics of the model have been implemented in UPPAAL SMC [7].

Table 1. Summary of schedulability of Herschel-Planck system.

$f = \frac{\text{BCET}}{\text{WCET}}$	0-71%	72-80%	81-86%	87-90%	90-100%
Symbolic MC:	maybe	maybe	maybe	n/a	**Safe**
Statistical MC:	**Unsafe**	maybe	maybe	maybe	maybe
Randomized MC:	**Unsafe**	**Unsafe**	maybe	maybe	maybe

Our contribution to the Herschel-Planck case study is to use our proposed under-approximate randomized reachability analysis techniques in hope to witness unschedulability in places where previously not possible. The summary of (un)schedulability of Herschel-Planck that includes the new results is shown in Table 1. Symbolic MC finds no deadline violation with over-approximate analysis and is able to conclude schedulability for $\frac{\text{BCET}}{\text{WCET}} \geq 90\%$. SMC find a witness of unschedulability for $\frac{\text{BCET}}{\text{WCET}} \leq 71\%$. Finally, our randomized reachability methods are able to further "breach the wall" of undecidable problem by discovering concrete traces proving unschedulability for $\frac{\text{BCET}}{\text{WCET}} \leq 80\%$. Moreover, for the same $\frac{\text{BCET}}{\text{WCET}}$, randomized reachability finds the deadline violation by three orders of magnitude faster than SMC: the case that took 23 hours for SMC now only takes 23 seconds with randomized methods.

To further verify the proposed efficient development process, we look at several different models of the *Gossiping Girls* problem made by the Master's thesis students – future model developers – and explore the potential of our randomized method. We also perform experiments on a range of other (timed and stopwatch automata) models and compare the performance of our randomized reachability analysis in "rare event" detection to that of existing verification techniques of UPPAAL: Breadth First Search (BFS), Depth First Search (DFS), Random Depth First Search (RDFS) and SMC. The results are extremely encouraging - randomized reachability methods perform up to several orders of magnitude faster and scale significantly better with increasing model sizes. Furthermore, randomized reachability uses constant memory w.r.t. the size of the model and typically requires only up to 25MB of memory. This is a notable improvement in comparison to the symbolic verification of upscaled and industrial sized models. Each of the experiments in this study was given 16GB of memory.

The main contributions of the paper are:

– A new randomized reachability analysis technique implemented in UPPAAL
– Detection of "rare event" states up to several orders of magnitude faster than with other existing model-checking techniques
– Possibility to analyze previously intractable models, including particular settings for the Herschel-Planck case study.
– Searching for *shorter* or *faster* traces with randomized reachability analysis.

The rest of the paper is structured as follows: In Section 2 we describe the different randomized methods we tried in this study. Section 3 presents the new results on the Herschel-Planck industrial case study and Section 4 provides more experimental results on other schedulability models. Section 5 demonstrates the efficiency of our randomized method applied on student models of the *Gossiping Girls* problem and Section 6 gives the results on other upscaled models. Finally, Sections 7 and 8 give conclusions and future work.

2 Randomized Reachability Analysis

The purpose of the randomized methods is to explore the state-space quickly and be less affected by the state-space explosion. The method is based on a repeated execution of concrete-state based *random walks* through the system. Each random walk is quick and lightweight as it avoids expensive computations of symbolic zone-based abstractions and does not store any information about already visited states in memory. The flaw of such analysis is its under-approximate nature of exploration which does not allow to conclude on reachability if the target state has never been found. However, the results of [13] hint that randomized reachability analysis has a potential to provide substantial performance improvements in comparison to existing model-checking techniques.

An already existing method of SMC tries to give valid statistical predictions based on stochastic semantics. SMC is very similar to the randomized method as it performs cheap, non-exhaustive simulations of the model. In cases where symbolic model-checking techniques of Uppaal are expensive or even inconclusive (for stopwatch automata), SMC is often used as a remedy to provide concrete traces to target states. The stochastic semantics SMC operates on allows for a model to mimic the behavior of a real system; however, this may not be efficient for detection of "rare event" states. Consider the timed automaton model in Figure 1 with the `Goal` location representing the target state we want to discover. The guard `x<=1` on the edge leading to `Goal` requires clock `x` to be at most of 1 time unit. According to stochastic semantics, at the starting location `Init` SMC would select a delay uniformly in range $[0, 1000]$, which is bounded by the invariant `x<=1000`. This leaves a probability of $\frac{1}{1000}$ to discover `Goal` in 1 step; Alternatively, the "loop" edge is taken which resets clock `x` with the update `x=0` thus resetting all the progress back to the initial state.

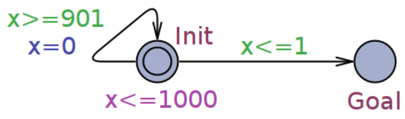

Fig. 1. Timed Automaton model with a `Goal` target state.

We aim to improve the efficiency of detecting "rare event" states with our new randomized method by experimenting with several different randomized heuristics and examining their efficiency through extensive experimental evaluation. A heuristic in this case dictates how a random walk is performed, i.e. how delays and transitions are chosen. The summary of the heuristics and their status is given in Table 2. We now explain each heuristic in detail.

Table 2. Randomized reachability analysis heuristics.

Acronym	Name	Origin	Status
SEM	Semantic exploration	New	Only experiments
RET	Random Enabled Transition	[13]	Implemented in Uppaal
RLC	Random Least Coverage	New	Only experiments
RLC-A	Random Least Coverage Accumulative	New	Only experiments

SEM An intuitive heuristic we tried, denoted as SEM, is based on the natural semantic exploration of the system. A meaningful delay, i.e. a delay that leads to an enabled transition, is selected uniformly at random and then a transition is picked uniformly from those available after the chosen delay has been made. In the model from Figure 1, SEM would choose a delay uniformly from two ranges – $[0, 1]$ and $[901, 1000]$, thus having a probability of $\frac{1}{100}$ to reach `Goal` in 1 step. Overall, we believe this heuristic will struggle the most in systems where certain specific delays are required to reach a target state, e.g. delaying exactly the lower or upper bound of the transition's availability range.

Differently from SEM, the heuristics we describe further (RET, RLC and RLC-A) require selecting a *target transition* first. The exact delay is then chosen only from that target transition's range of available delays. Selecting transition first makes exploration of the state-space more uniform and removes a bias towards transitions with larger availability range. The mechanism for choosing delays is common between the heuristics presented below and will be described later in this section.

RET As a continuation of our work on randomized techniques from [13] we implement them in UPPAAL for both Timed and Stopwatch Automata. The study proposed two different heuristics for selecting a target transition. A heuristic denoted as RET (Random Enabled Transition) selects one of the eventually enabled transitions, i.e. transitions that are either currently enabled or will become such after a delay, uniformly at random. This means that at each step each transition is equally likely to be selected. When used in the model from Figure 1, RET would first choose one of the two transitions at random, having a probability of $\frac{1}{2}$ to reach the `Goal` location in 1 step.

RLC, RLC-A Here we introduce a heuristic denoted as RLC that chooses a transition with the *least coverage* for the sending edge. If there is more than one such transition, RLC picks one uniformly at random. In systems that are cyclic or contain multiple loops, RLC provides a more uniform exploration of the state-space which may be useful for some models. Consider the model from Figure 2 that uses two integer variables `i` and `j`. The only initially available edge is the bottom loop edge at the `Init` location which increments the variable `i` by 1 upon each traversal. Once `i==2`, the leftmost loop edge can be taken, resulting in a reset of `i` and increment of `j` (`i=0,j++`). Crucially, if the variable `i` is incremented above the value 2, the leftmost loop edge becomes permanently unavailable. Hence, to reach `Goal` the leftmost edge has to be taken as soon as it becomes available and at least 7 times (`j>=7`) in one run. Since the coverage of the leftmost edge is always lower, the probability for RLC heuristic to discover `Goal` in 1 random walk is 100% while for RET it is less than 1%. The coverage counters, however, are reset at the start of a random walk, making each subsequent run independent of the previous one. We also experiment with a similar heuristic that does not reset the coverage counters and instead keeps them shared among all of the random walks. We denote such *accumulative* heuristic as RLC-A.

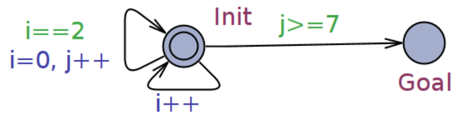

Fig. 2. Timed Automaton model of a difficult case for RET heuristic.

Other randomized methods investigated A number of tokenized heuristics, inspired by [14], have been attempted with the intent of storing a small, fixed number of tokens in a clever way to increase the likelihood of reaching the target state faster. Unfortunately, as no considerable improvements have been observed we decided to exclude these heuristics and leave them as future work.

We have also tried using traces of symbolic MC of Uppaal from verification of the Herschel-Planck model to guide the random walks towards the target state. However, even with the RDFS search strategy, all of the symbolic traces have appeared to be spurious due to the over-approximate analysis of stopwatch automata. Hence, we could not gain any useful results with this approach.

To reduce resource demands for the most expensive operation in a random walk – computation of eventually enabled transitions – an alternative heuristic to RET was used in [13] denoted as RCF (Random Channel First). Instead of computing all eventually enabled transitions, RCF first randomly picks a channel and only computes transitions labeled with that channel. However, during implementation of these techniques in Uppaal it became clear that the RCF does not give performance advantages over RET due to the differences in the underlying data structures of Uppaal and Java prototype from [13]. Therefore, we got rid of the RCF heuristic.

Choosing delay A naive way of choosing delays – uniformly at random from a given range – is likely to not be very efficient. While in some systems that are either small or not sensitive to specific delay values reaching target state can be doable, in more complex models such a strategy may not be optimal. In [13] we experimented with a few different strategies for choosing delay values, such as 1) uniformly at random, 2) based on predefined probability distribution and 3) based in changing (adapting) delay probability distributions. The experiments have shown the first strategy to be the least efficient, whereas the third one has shown the most potential. Hence, we reuse the third strategy here with slight modifications for RET, RLC and RLC-A heuristics.

The idea behind the adaptive delay choice algorithm is the following: the delays are drawn in accordance to some predefined delay probability distribution which changes on each unsuccessful random walk. Such distribution in this case defines probability for lower bound (LB), upper bound (UB) or the values in between the bounds to be chosen. For example, a distribution of 40% LB/40% UB means that it is equally probable that either LB or UB will be selected as a delay, while leaving 20% chance for intermediate delay to be chosen uniformly at random from the range that excludes the bounds. Table 3 shows the sequence delay probability distributions used in this study. Upon reaching the last distribution in the sequence, the next random walk starts from the first one.

Table 3. Delay probability distributions used for RET, RLC and RLC-A.

Sequence	1	2	3	4	5	6	7	8	9	10	11
Lower bound	60%	70%	80%	90%	100%	0%	10%	20%	30%	40%	40%
Uniform	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	20%
Upper bound	40%	30%	20%	10%	0%	100%	90%	80%	70%	60%	40%

Previously, the cycle of delay probability distributions did not leave any room for intermediate time delays, considering only LB or UB values. The downside is that for some systems it means that parts of the state-space become unreachable by the algorithm; however, experiments have shown this strategy to be surprisingly efficient. To eliminate the flaw of intermediate delay values never being chosen, here we add a 40% LB/40% UB probability distribution, leaving 20% chance to select an intermediate time value. As the result, a target state, if one exists, will be eventually found in any system.

Random walk depth To explore the state-space gradually and reduce the risk of a random walk being stuck in an isolated part of the state-space with no target state, we increase the random walk depth dynamically as the exploration continues. Specifically, the first batch of random walks at most can perform 2^4 steps. After the full cycle of delay probability distributions is completed, the random walks in the next cycle have their maximum allowed depth doubled, but no further than 2^{18} steps. Should one have some apriori knowledge of the system, it is also possible to manually set the maximum allowed depth in Uppaal that is a constant value used for all of the conducted random walks.

Shorter or Faster trace Since our techniques cannot disprove reachability of a target state due to under-approximate analysis, searching for errors in large systems, where symbolic techniques struggle, is one of the main expected applications. To aid developer in analyzing error traces and fixing systems, we implement an option to search for an optimal trace being the either *shortest*, in the size of steps, or the *fastest*, in the amount of total delay. With either one of these options selected, the algorithm searches for the initial trace and afterwards restricts all subsequent random walks to the current smallest amount of steps or delay discovered. Every randomized heuristic can be used with the *shortest* or *fastest* option and we refer to those by appending "-S" or "-F", e.g. RET-S.

In symbolic model-checking, searching for an optimal trace requires an exhaustive exploration of the state-space. Thus, for larger systems, it often drastically increases time and memory demands up to an extent where it becomes impractical. As opposed to that, our randomized techniques do not require more memory as the old trace is being discarded as soon as the new, more optimal one is discovered. On the down side, being a non-exhaustive technique the randomized search cannot guarantee that any discovered trace is indeed the most optimal, endlessly continuing the search. In UPPAAL we let the user provide *timeout* value (in seconds) which is defaulted to 300 seconds.

3 New Results on Herschel-Planck

According to previous results on Herschel-Planck model [8], symbolic MC confirmed schedulability for $f = \frac{\text{BCET}}{\text{WCET}} \geq 90\%$. However, symbolic MC cannot be used for disproving schedulability due to over-approximate analysis of automata with stopwatches, used to encode preemption. Thus, SMC was used to generate concrete counterexamples, disproving schedulability for $f \leq 71\%$. For the rest of $f \in (71\%, 90\%)$ both symbolic and statistical MC were inconclusive due to either over-approximation or burden in computation time, respectively. All of the models used in the experiments are made available at: http://people.cs.aau.dk/~ulrik/submissions/874325/FMICS2021.zip

Table 4. Average time to detect non-schedulability in Herschel-Planck (in seconds). SMC search is limited to 160, 640 or 1280 cycles of 250ms.

f(%)	SMC(160)	SMC(640)	SMC(1280)	SEM	RET	RLC	RLC-A
68	3378.82	3656.0	2626.11	nf	**14.1**	14.35	14.48
69	6087.64	3258.13	3565.49	nf	15.91	14.32	**13.7**
70	19408.04	16875.89	24322.69	nf	17.59	14.47	**14.77**
71	85837.23	nf	nf	nf	22.54	**16.56**	16.75
72	nf	nf	nf	nf	27.81	**18.42**	18.96
73	nf	nf	nf	nf	31.56	**20.66**	20.68
74	nf	nf	nf	nf	52.53	**38.08**	40.31
75	nf	nf	nf	nf	72.16	**61.98**	68.35
76	nf	nf	nf	nf	**83.12**	328.03	327.32
77	nf	nf	nf	nf	**375.08**	nf	nf
78	nf	nf	nf	nf	**1155.50**	nf	nf
79	nf	nf	nf	nf	**2009.01**	nf	nf
80	nf	nf	nf	nf	**11194.43**	nf	nf
81	nf	nf	nf	nf	nf	nf	nf

In our experiments we compare SMC to our randomized reachability analysis techniques in attempt to detect non-schedulability in Herschel-Planck model for varying execution times in the interval of $[f \cdot \text{WCET}, \text{WCET}]$. The results are shown in Table 4 with each test case given 48 hours. As the f value gets higher we see the expected growth in computational demands with $f = 71\%$ requiring just under 24 hours for SMC to disprove schedulability, confirming results of [8]. On the other hand, 3 out of 4 of our randomized heuristics were able to detect an error for the same setting of $f = 71\%$ in less than 23 seconds, improving on performance of SMC by three orders of magnitude. Furthermore, RET heuristic appeared to give the best results, witnessing unschedulability for values of f up to and including 80%. We have also tried running longer experiments of up to 7 days for $f = 81\%$, but no errors were discovered which hints at possibility of the Herschel-Planck system being schedulable for $f > 80\%$. The SEM heuristic turned out to be the least efficient one, failing to discover any errors, which is likely due to the exponentially small probability of hitting the "right" time windows with chosen delays. Overall, these experiments showcase the strength of randomized reachability analysis being fit as a part of an efficient development process that speeds up falsification of models.

Table 5. Trace length comparison.

f(%)	RET	RET-S	Timeout
68	6882	560	1h
69	7619	568	1h
70	8285	572	1h
71	10411	570	1h
72	12394	571	1h
73	15937	578	1h
74	26605	1549	1h
75	41003	1546	1h
76	40154	1529	1h
77	97258	1536	1h
78	119939	1540	5h
79	129387	1536	5h
80	145493	6455	20h

Once a trace leading to an error is discovered, it might be in the interest of a developer to analyze it to find the cause for the error. The trace, however, can be arbitrarily long, especially for larger systems, making its analysis difficult in practice. In our next experiment we look at the average length of traces found for Herschel-Planck system and compare RET heuristic from experiments in Table 4 against the version of RET with the shortest trace option enabled - RET-S. In order for non-exhaustive exploration of RET-S to terminate, we specify the Timeout value and increase it w.r.t. to the average time required by RET to find an error. The results are shown in Table 5. With the given timeout, RET-S shortens the length of the trace by a factor of 12 at minimum. Note that for $f \in [75\%, 79\%]$ the length of the shortest discovered trace is approximately

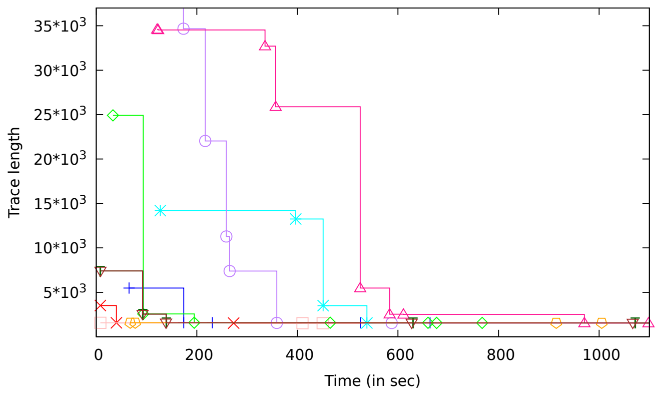

Fig. 3. 10 runs of RET-S for Herschel-Planck with $f = 75\%$.

the same – just under 1600 – while the effort to discover such trace is roughly proportional to the average time to detect the first trace (as shown in Table 4).

The exact value of the timeout has to be decided on by the user which may not be an easy parameter to estimate in the setting of randomized and unpredictable exploration. To better understand how RET-S behaves, we plot 10 runs of RET-S for Herschel-Planck system with $f = 75\%$ in Figure 3. In average it took 263.14 seconds to find a trace of sub 1600 steps, while the longest run took 970 seconds.

4 More Schedulability

As already stated, application of symbolic techniques to stopwatch models may provide spurious traces due to over-approximate analysis of Uppaal. If the target state in these models is potentially reachable, we can use SMC to generate concrete and exact traces witnessing the reachability of the goal state. However, SMC can only be applied to systems with broadcast channels as required by stochastic semantics SMC operates on. In stopwatch models that use handshake channels, our randomized methods become the only solution that can perform a more exact reachability analysis.

We consider more schedulability systems modelled as stopwatch automata. Table 6 shows experiments for two different sets of schedulability problems: ARINC-653 partition scheduling of integrated modular avionics systems [11] (denoted as IMAOptim) and schedulability of Java bytecode systems, originating from TetaSARTS project [21], that are encoded as networks of automata and represent the original layered structure of Java bytecode systems. Our randomized methods discover the target state within 20 seconds even for a huge system with almost 12 thousands of locations, where other techniques are either not applicable or run out of memory.

Table 6. Average time to find target state in stopwatch automata models. Symbolic MC techniques provide potentially spurious traces.

Model	#loc	BFS	DFS	RDFS	SMC	SEM	RET	RLC	RLC-A
IMAOptim-0	88	0.09	0.1	0.07	**0.04**	0.07	0.1	0.1	0.08
IMAOptim-1	88	0.21	0.2	0.08	**0.05**	**0.05**	0.08	0.08	0.06
IMAOptim-2	88	0.21	0.26	0.09	**0.06**	0.08	0.11	0.11	0.1
md5-jop	594	0.25	10.8	6.53	n/a	0.15	0.18	0.18	**0.12**
md5-hvmimp	476	0.41	0.85	0.49	n/a	0.1	0.14	0.14	**0.09**
md5-hvmexp	11901	oom	oom	oom	n/a	14.17	19.85	20.18	**8.71**
MP-jop	371	0.39	0.14	0.12	n/a	**0.08**	0.12	0.12	0.09
MP-hvmimp	371	0.35	0.14	0.12	n/a	**0.08**	0.12	0.12	0.09
MP-hvmexp	4388	oom	oom	oom	n/a	13.49	22.95	21.99	**8.59**
simplerts-opt	409	oom	oom	oom	n/a	2.43	**1.48**	nf	nf

5 Gossiping Girls

As claimed earlier, the randomized reachability analysis can serve as a useful tool particularly for an efficient development process. It can be used early in the development, as well as in late stages, for a quick falsification of models, i.e. discovery of errors or checking if another "rare event" state is actually reachable in practice.

To test the efficiency of our randomized methods and challenge them with different model development styles, we look at models of the same problem created by different developers. Specifically, we consider the *Gossiping Girls* problem, where a number of girls n each know a distinct secret and wish to share it with the rest of the girls. They can do so by calling each other and exchanging either only their initial or all of currently known secrets. The girls are organized as a total graph, allowing them to talk with each other concurrently, but with a maximum of 2 girls per call. Some variations of the problem have specific time constraints on the duration of the call or exhibit a different secret exchange pattern, but all with the same final goal of all the girls discovering all of the secrets. This is a combinatorial problem with each girl having a string of n bits which can at most take 2^n values. For a total of n girls this amounts to a string of n^2 with at most 2^{n^2} values. This makes it an incredibly hard combinatorial problem which, when scaled up, quickly exposes the limits of symbolic model-checking due to the state-space explosion problem.

We have gathered 10 models of the Gossiping Girls problem made by Master's thesis students as the final assignment for the course on model-checking at Aalborg University in Denmark. These students represent potential future model developers and we use their model to further experiment on applicability of the randomized methods. The implementation details vary from model to model, including timing constraints and secret exchange patterns. We leave the models unchanged and only scale them up to a certain amount of nodes to challenge both symbolic and randomized methods.

We first experiment on the models scaled up to 8 girls and look for a state with of all the girls having exchanged their secrets, while bounded by a certain global time constraint. The results are shown in Table 7 where each cell represent the average time for each found trace within 2 hours. For 9 out of 10 of the models our randomized heuristic RET shows a massive improvement in performance compared to symbolic methods, whereas in 1 model the performance is on the same level. Since the problem is time constrained, the worst performance is that of SEM heuristic which fails to find our target state due to an inefficient way of selecting delays. Importantly, for some models some of the RDFS runs were "lucky" to discover the target state almost immediately, while other "unlucky" tries instead ran out of memory (oom). The oom attempts of RDFS contribute to the performance by noticeably dragging up the average time to find the goal state. Another important factor is memory: unlike symbolic methods, that are given 16GB of memory, our randomized techniques do not run out of memory as its usage is constant w.r.t to the size of the model and amounts to at most 14MB for any of the heuristics for this set of experiments.

Table 7. Gossiping Girls with 8 nodes. Each cell represent avg. time for each found trace within 2 hours. Searching for a state with all secrets shared within a certain time.

Model	BFS	DFS	RDFS	SEM	RET	RLC	RLC-A
Gosgirls-1	oom	oom	697.13	nf	**0.39**	6949.95	nf
Gosgirls-2	oom	oom	**0.02**	nf	0.04	0.04	0.04
Gosgirls-3	oom	oom	44.49	nf	**0.02**	**0.02**	0.09
Gosgirls-4	oom	oom	28.35	nf	**0.03**	0.03	nf
Gosgirls-5	oom	oom	229.98	nf	**0.02**	**0.02**	**0.02**
Gosgirls-6	oom	oom	64.00	nf	**3.71**	167.44	1530.99
Gosgirls-7	oom	oom	55.61	nf	**0.17**	15.16	15.6
Gosgirls-8	oom	oom	13.96	nf	0.04	**0.03**	**0.03**
Gosgirls-9	oom	oom	2.08	nf	0.08	**0.07**	0.08
Gosgirls-10	oom	oom	598.64	nf	**0.24**	1.72	nf

Discovery of the state where all the secrets are known is arguably an easy target as such state will eventually always appear as we traverse the state-space. This also explains why RDFS was sometimes “lucky” to detect the searched state before it ran out of memory. We now experiment with searching for a particular configuration of secrets in models with 6 girls and show results in Table 8. Concretely, we divide the 6 girls into two clusters of 2 and 4 girls, and search for a state where each girl knows all the secrets of the other girls in the same cluster, but none from the other cluster. Such a state occurs less often in the state-space and is easy to miss, making it a more challenging problem; Hence, only 6 girls are considered. Unlike in previous experiments, the most efficient symbolic search strategy is different for each individual model due to the variance in model implementations. The randomized methods appear largely superior in almost all cases, with the RET heuristic being the most consistent and efficient across all the models. Note that even for 6 girls in a lot of cases symbolic techniques still run out of memory, whereas our random methods use less than 15MB.

Table 8. Gossiping Girls with 6 nodes. Each cell represent avg. time for each found trace within 2 hours. Searching for a particular configuration of secrets known.

Model	BFS	DFS	RDFS	SEM	RET	RLC	RLC-A
Gosgirls-1	16.98	oom	oom	2.17	**1.35**	1.60	0.23
Gosgirls-2	**0.04**	oom	360.43	**0.04**	**0.04**	**0.04**	**0.04**
Gosgirls-3	77.96	oom	oom	nf	1.44	0.19	**0.10**
Gosgirls-4	oom	oom	oom	nf	0.03	**0.02**	nf
Gosgirls-5	oom	oom	oom	nf	**0.02**	**0.02**	**0.02**
Gosgirls-6	oom	244.66	2596.62	**5.92**	7.10	nf	nf
Gosgirls-7	oom	oom	oom	nf	**0.14**	75.44	141.20
Gosgirls-8	32.63	oom	oom	nf	**0.11**	3.24	505.99
Gosgirls-9	oom	oom	199.77	**0.10**	13.04	3.65	2.07
Gosgirls-10	oom	oom	209.36	nf	**0.02**	0.03	0.04

6 Scalability Experiments

We further investigate the efficiency of our randomized methods on a set of standard Uppaal timed automata models. The models are scaled up in order to challenge both symbolic and randomized techniques and the results are provided in Table 9. The results are truly impressive – randomized methods perform up to 4 orders of magnitude faster and scale significantly better.

Table 9. Average time to find target state in Timed Automata.

Model	BFS	DFS	RDFS	SEM	RET	RLC	RLC-A
csma-cd-20N	20.2	oom	**0.02**	0.03	0.07	0.06	0.21
csma-cd-22N	37.48	oom	oom	**0.03**	0.08	0.08	0.31
csma-cd-25N	91.0	oom	oom	**0.05**	0.09	0.1	0.55
csma-cd-30N	313.54	oom	oom	**0.05**	0.12	0.19	1.43
csma-cd-50N	oom	oom	oom	**0.46**	0.84	1.19	15.29
Fischer-10N	0.9	22.84	4.3	**0.04**	0.05	1.21	nf
Fischer-15N	8.35	6037.63	9038.96	**0.09**	**0.09**	5.06	nf
Fischer-20N	72.61	oom	oom	0.3	**0.28**	17.28	nf
Fischer-25N	452.45	oom	oom	**0.64**	0.73	36.93	nf
Fischer-50N	oom	oom	90.01	**21.78**	23.79	233.67	nf
FischerME-10N	7.15	0.14	0.02	0.01	0.02	**0.01**	0.02
FischerME-15N	oom	11.45	0.05	0.04	0.04	**0.03**	0.16
FischerME-20N	oom	970.33	0.4	0.11	0.09	0.05	**0.04**
FischerME-25N	oom	oom	83.29	0.25	0.21	0.08	**0.07**
FischerME-50N	oom	oom	174.32	14.87	15.26	**0.49**	4.04
LE-Chan-3N	0.03	0.35	0.04	**0.01**	**0.01**	**0.01**	**0.01**
LE-Chan-4N	oom	oom	107.7	0.95	0.54	4.36	**0.07**
LE-Chan-5N	oom	oom	1167.41	53.21	**31.38**	102.08	nf
LE-Hops-3N	0.02	0.02	0.02	**0.01**	**0.01**	**0.01**	**0.01**
LE-Hops-4N	oom	oom	oom	49.40	**14.57**	428.96	1588.33
LE-Hops-5N	oom	oom	1108.15	63.44	**35.15**	36.49	49.00
Milner-N100	0.45	0.16	2.72	nf	**0.11**	**0.11**	0.12
Milner-N500	44.44	10.56	1619.75	nf	**1.19**	1.2	1.43
Milner-N1000	488.41	110.35	36455.73	nf	**4.44**	4.45	4.59
Train-200N	oom	5.64	6.06	5.91	**5.4**	16699.98	nf
Train-300N	oom	28.19	30.28	**25.62**	26.53	nf	nf
Train-400N	oom	85.22	90.66	**67.91**	70.87	nf	nf
Train-500N	oom	210.89	223.13	**181.99**	188.9	nf	nf
Train-1000N	nf	3461.17	3542.08	**2192.12**	2541.57	nf	nf
Train-2000N	nf	71286.92	oom	**19229.02**	23233.21	nf	nf

Even though the SEM heuristic shows the best performance of many models, its inefficient way of selecting delays causes it to completely miss target states on some models as demonstrated by all of the experiments in this study. Due to under-approximation, it is possible to construct "evil" examples for any heuristic, rendering it inefficient. We then make all of the heuristics available in Uppaal.

7 Conclusion

We have presented a new method of randomized reachability analysis in the domain of model-based verification. The method excels at detection of "rare event" states, such as errors, by means of quick and lightweight random walks through the system. Randomized reachability analysis explores the state-space in an under-approximate manner and can only conclude on reachability if the target state is discovered. However, in many cases this method significantly outperforms other existing techniques at reachability checking. Randomized reachability analysis is therefore a very useful addition to the process of model development: it provides an efficient way of checking models for potential bugs or violations during the development and can be followed by exhaustive and expensive symbolic verification at the very end. The randomized method also supports the search for either *shorter* or *faster* trace to the target state, which improves the process of debugging the model. The randomized reachability analysis is implemented and made available for use in the model checker Uppaal.

To validate the efficiency of our method, we have performed extensive experiments on models of varying size and origin. The results are extremely encouraging: randomized reachability analysis discovers "rare event" states up to several orders of magnitude faster. In particular, a case that could previously be analyzed by SMC in 23 hours now only takes 23 seconds. Moreover, our randomized methods discover traces to target states in cases that were previously intractable by any of the existing techniques either due to state-space explosion or inconclusiveness in verification of stopwatch models.

8 Future Work

Further investigations into tokenized, coverage-based and guided methods can be done to improve the efficiency of the method. Some combinations of static analysis of the models with either fixed or dynamic look-ahead for the random walk could result in better performance of the method.

One future goal is to perform a more thorough and independent user evaluation of the benefits of the randomized reachability analysis. This could indicate the need for more parameters to be manually set by the user, such as custom delay probability distribution, or could highlight other areas for improvement of randomized methods.

Automatic sanity checks is another improvement that can noticeably enhance the user experience and aid during model development. An implementation [17] for Uppaal of such sanity checks has been undertaken as a master thesis project [18] in the Formal Methods & Tools group at University of Twente. This report demonstrates the usefulness of such sanity checks and highlights the need for quick feedback to the tool user. Our randomized method is highly suitable for this purpose.

References

1. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures. Lecture Notes in Computer Science, vol. 3185, pp. 200–236. Springer (2004). https://doi.org/10.1007/978-3-540-30080-9_7
2. Boudjadar, A., David, A., Kim, J., Larsen, K., Mikučionis, M., Nyman, U., Skou, A.: Statistical and exact schedulability analysis of hierarchical scheduling systems. Science of Computer Programming **127**, 103–130 (May 2016). https://doi.org/10.1016/j.scico.2016.05.008
3. Boudjadar, A., David, A., Kim, J., Larsen, K., Mikučionis, M., Nyman, U., Skou, A.: A reconfigurable framework for compositional schedulability and power analysis of hierarchical scheduling systems with frequency scaling. Science of Computer Programming **113**(3), 236–260 (Dec 2015). https://doi.org/10.1016/j.scico.2015.10.003
4. Brekling, A., Hansen, M.R., Madsen, J.: Moves — a framework for modelling and verifying embedded systems. In: 2009 International Conference on Microelectronics - ICM. pp. 149–152 (2009). https://doi.org/10.1109/ICM.2009.5418667
5. Burns, A.: Preemptive Priority-Based Scheduling: An Appropriate Engineering Approach, p. 225–248. Prentice-Hall, Inc., USA (1995)
6. David, A., Illum, J., Larsen, K.G., Skou, A.: Model-based framework for schedulability analysis using uppaal 4.1. Model-based design for embedded systems **1**(1), 93–119 (2009)
7. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Uppaal SMC tutorial. International Journal on Software Tools for Technology Transfer **17**(4), 397–415 (2015)
8. David, A., Larsen, K.G., Legay, A., Mikucionis, M.: Schedulability of herschel-planck revisited using statistical model checking. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part II. Lecture Notes in Computer Science, vol. 7610, pp. 293–307. Springer (2012). https://doi.org/10.1007/978-3-642-34032-1_28, https://doi.org/10.1007/978-3-642-34032-1_28
9. Fersman, E., Krcal, P., Pettersson, P., Yi, W.: Task automata: Schedulability, decidability and undecidability. Information and Computation **205**(8), 1149–1172 (2007). https://doi.org/https://doi.org/10.1016/j.ic.2007.01.009, https://www.sciencedirect.com/science/article/pii/S0890540107000089
10. Grosu, R., Smolka, S.A.: Monte Carlo Model Checking. In: Halbwachs, N., Zuck, L.D. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 271–286. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
11. Han, Pujie and Zhai, Zhengjun and Nielsen, Brian and Nyman, Ulrik: Model-based optimization of arinc-653 partition scheduling. International Journal on Software Tools for Technology Transfer (Feb 2021). https://doi.org/10.1007/s10009-020-00597-6, https://doi.org/10.1007/s10009-020-00597-6
12. Joseph, M., Pandya, P.: Finding Response Times in a Real-Time System. The Computer Journal **29**(5), 390–395 (01 1986). https://doi.org/10.1093/comjnl/29.5.390, https://doi.org/10.1093/comjnl/29.5.390

13. Kiviriga, A., Larsen, K.G., Nyman, U.: Randomized refinement checking of timed I/O automata. In: Pang, J., Zhang, L. (eds.) Dependable Software Engineering. Theories, Tools, and Applications - 6th International Symposium, SETTA 2020, Guangzhou, China, November 24-27, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12153, pp. 70–88. Springer (2020). https://doi.org/10.1007/978-3-030-62822-2_5, https://doi.org/10.1007/978-3-030-62822-2_5
14. Larsen, K., Peled, D., Sedwards, S.: Memory-Efficient Tactics for Randomized LTL Model Checking. In: Paskevich, A., Wies, T. (eds.) Verified Software. Theories, Tools, and Experiments. pp. 152–169. Springer International Publishing, Cham (2017)
15. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) Runtime Verification. pp. 122–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
16. Mikučionis, M., Larsen, K.G., Rasmussen, J.I., Nielsen, B., Skou, A., Palm, S.U., Pedersen, J.S., Hougaard, P.: Schedulability analysis using uppaal: Herschel-planck case study. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation. pp. 175–190. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
17. Onis, R.: UrPal. https://github.com/utwente-fmt/UrPal, Accessed: 2021-05-18
18. Onis, R.: Does your model make sense? : Automatic verification of timed systems (December 2018), http://essay.utwente.nl/77031/
19. Palm, S.: Herschel-planck acc asw: sizing, timing and schedulability analysis. Tech. rep., Tech. rep., Terma A/S (2006)
20. Sen, K., Viswanathan, M., Agha, G.: Statistical Model Checking of Black-Box Probabilistic Systems. In: Alur, R., Peled, D.A. (eds.) Computer Aided Verification. pp. 202–215. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
21. Søe Luckow, K., Bøgholm, T., Thomsen, B.: A Flexible Schedulability Analysis Tool for SCJ Programs. http://people.cs.aau.dk/~boegholm/tetasarts/, Accessed: 2021-05-07