On Correctness, Precision, and Performance in Quantitative Verification*

QComp 2020 Competition Report

Carlos E. Budde[1], Arnd Hartmanns[1], Michaela Klauck[2], Jan Křetínský[3], David Parker[4], Tim Quatmann[5], Andrea Turrini[6,7], and Zhen Zhang[8]

[1] University of Twente, Enschede, The Netherlands
[2] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[3] Technical University of Munich, Munich, Germany
[4] University of Birmingham, Birmingham, UK
[5] RWTH Aachen University, Aachen, Germany
[6] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[7] Institute of Intelligent Software, Guangzhou, Guangzhou, China
[8] Utah State University, Logan, UT, USA

Abstract. Quantitative verification tools compute probabilities, expected rewards, or steady-state values for formal models of stochastic and timed systems. Exact results often cannot be obtained efficiently, so most tools use floating-point arithmetic in iterative algorithms that approximate the quantity of interest. Correctness is thus defined by the desired precision and determines performance. In this paper, we report on the experimental evaluation of these trade-offs performed in QComp 2020: the second friendly competition of tools for the analysis of quantitative formal models. We survey the precision guarantees—ranging from exact rational results to statistical confidence statements—offered by the nine participating tools. They gave rise to a performance evaluation using five tracks with varying correctness criteria, of which we present the results.

1 Introduction

Quantitative formal models feature probabilistic choices, real-time aspects, or continuous dynamics. They are used to study safety, dependability, or performance aspects of e.g. randomised algorithms, network protocols, biological processes, or cyber-physical systems [1, 58]. Probabilistic models need dedicated

* The authors are listed alphabetically. This work was supported by DFG grant 389792660 as part of TRR 248 (CPEC), DFG grant 383882557 (SUV), ERC Advanced Grant 787914 (FRAPPANT), ERC Advanced Grant 834115 (FUN2MODEL), ERC Advanced Grant 695614 (POWVER), the Guangdong Science and Technology Department (grant no. 2018B010107004), the National Natural Science Foundation of China (grant nos. 61761136011, 61532019, 61836005), National Science Foundation grant CCF-1856733, NWO project 15474 (SEQUOIA), and NWO VENI grant no. 639.021.754.

numeric algorithms to compute or approximate rational or real-valued probabilities, expected values, or long-run averages. In this paper, we focus on tools for the analysis of probabilistic formal models w.r.t. such quantitative properties.

Over the past two decades, a variety of algorithms have been devised for this purpose. Most of them can roughly be categorised as variants of *probabilistic model checking* (PMC) [9] and *statistical model checking* (SMC) [2], with *probabilistic planning* closely related to the former. In PMC, the model's state space is explored—partially or exhaustively—to obtain an in-memory representation of the model's underlying semantics, which is typically a Markov chain or some extension thereof. The value of interest can then be computed using numeric algorithms such as value iteration. PMC is thus subject to the state space explosion problem, limiting its ability to be applied to very large case studies. SMC, on the other hand, relies on Monte Carlo simulation—generating random runs through the model's semantics—to statistically estimate the value of interest. It does not need to store states other than the current and next one during run generation, and thus avoids state space explosion entirely. However, when faced with a rare event—e.g. when trying to estimate a reachability probability on the order of 10^{-9} with a suitable error of, say, 10^{-10}—the number of runs needed explodes. Furthermore, nondeterminism—controllable or adversarial unquantified choices, such as in Markov decision processes (MDP) [78]—turn the estimation problem into an optimisation problem, which SMC cannot directly handle. Probabilistic planning is similar to PMC, but crucially employs heuristics to try to avoid exploring the entire state space. Its focus is on *finding strategies* in MDP, i.e. the choices that lead to the maximum reward, whereas PMC traditionally *computes values* (e.g. expected rewards) and checks complex logical formulas.

With new algorithms come new tools: first academic prototypes, which may over time develop into extensive collections of algorithms or tools targeting various problems and use cases. In 2019, the first competition of tools for the analysis of quantitative formal methods, QComp 2019 [46], took place. Using selected benchmarks from the quantitative verification benchmark set (QVBS) [58], all of which are available in the tool-independent JANI model interchange format [19], it compared nine tools—ranging from general-purpose probabilistic model checkers to specialised SMC tools for rare events in dynamic fault trees—in terms of performance, versatility, and usability. A major concern that surfaced during the setup of QComp 2019 was that quantitative verification tools return *numbers*—and most of them use inexact methods to obtain these numbers, relying on floating-point arithmetic and iterative algorithms that only approximate the true values. Additionally, the long-time standard algorithm used by PMC tools, value iteration, is now known to be unsound [43]; and SMC tools can only deliver statistical guarantees that allow them to produce incorrect results with a certain probability (typically $\leq 5\,\%$ of the time). Thus, while we on the one hand should demand verification tools to always deliver correct verdicts, correctness in quantitative verification cannot effectively be achieved without admitting *some* error. The best we can do, then, is to accompany results with precise statements about *how* correct they are guaranteed to be.

In this paper, we report on QComp 2020, the second edition of this competition. We focus on the issue of correctness of results, in particular on the trade-off between strength of correctness guarantees and analysis performance. After an overview of the types of formalisms and properties considered by QComp 2020 in Sect. 2, we thus expand on this in Sect. 3. Subsequently, in Sect. 4, we describe the tools that participated in the competition, noting in particular which kinds of correctness guarantees each tool can provide. Finally, we describe in Sect. 5 the setup of the QComp 2020 performance evaluation, and present its outcomes.

2 Languages, Formalisms, and Properties

Formal models are specified in *modelling languages*: graphical or textual notations designed for human users to compactly describe complex systems. They are equipped with a semantics in terms of a mathematical *formalism* that provides the basis for various analysis algorithms. Models are accompanied by *properties* that specify a quantity of interest related to a set of behaviours of the model.

Modelling languages. QComp 2020 draws its benchmarks from the QVBS, which currently consists of 78 different models, many of them parametrised to scale from small to large state spaces, with a set of properties associated to each model. Every model is available in Jani, a JSON-based format designed as an intermediate representation that bridges tools and that other modelling languages can be transformed into, as well as in its "original" modelling language. The models used for QComp 2020 were originally specified in the Galileo format [86] for fault trees, the GreatSPN format [4] for generalised stochastic Petri nets, the process algebra-based high-level modelling language Modest [47], the PGCL specification for probabilistic programs [40], PPDDL for probabilistic planning domains [89], and the guarded-command Prism language [68].

Formalisms. Most modelling languages or higher-level formalisms map to some extension of automata, i.e. graphs of states (that may contain relevant structure) connected by transitions (possibly with several annotations). The benchmarks of QComp 2020 have a semantics in terms of discrete- and continuous-time Markov chains (**DTMC** and **CTMC**, respectively), which provide finite-support probabilistic choices and, in CTMC, stochastic delays that follow exponential distributions; Markov decision processes (**MDP**), which extend DTMC with nondeterministic choices; Markov automata (**MA**) [35], which combine CTMC and MDP in a compositional way; and probabilistic timed automata (**PTA**) [71], which marry MDP and timed automata [3], thus providing probabilistic choices together with nondeterministic continuous real-time behaviour.

Properties. For QComp 2020's performance evaluation, we consider basic types of quantitative properties only. This is to ensure that, for every property, we have more than one tool able to compute its value. In particular, we include unbounded probabilistic reachability ("what is the—maximum or minimum, in case of models with nondeterminism—probability to eventually reach a given set of goal states"), or P-type properties for short; bounded probabilistic reachability (P-type properties with the additional requirement of reaching the states before some quantity exceeds a specified bound, in particular time for Pt-type and an accumulated reward for Pr-type properties, both summarised as type Pb); expected accumulated rewards until a given set of states is reached, or E-type properties, including bounded variants (type Eb); and long-run average rewards for CTMC and MA (type S, with the special case of steady-state probabilities).

Beyond QComp. Many other quantitative modelling languages not yet represented in the QVBS exist such as UPPAAL's XML format [13] or those supported by MÖBIUS [26]. The formalisms of QComp are part of a larger family tree of quantitative automata-based formalisms as shown in the previous competition report [46, Fig. 1]. They are all 1- or 1.5-player games; a future QComp may expand to games with more players that capture competitive behaviour towards conflicting goals as tool support for stochastic games expands. From our basic properties, logics can be constructed that allow the expression of *nested* quantitative requirements, e.g. that with probability 1, we must reach a state within n transitions from which the probability of eventually reaching an unsafe state is $< 10^{-9}$. Examples are CSL [10] for CTMC, PTCTL [71] for PTA, and rPATL [25] for stochastic games. Of further interest are *multi-objective* trade-offs [36], which query for Pareto-optimal strategies balancing multiple goals.

3 Correctness and Precision

We now describe the challenges and trade-offs in evaluating and ensuring the correctness of quantitative analysis results, and how QComp 2020 addresses them.

3.1 Correctness Challenges

Unsound algorithms. For a long time, the standard algorithm for PMC was value iteration (VI). It associates a value to each state that approximates the local value of the quantity of interest (e.g. the probability to reach the goal from that state), then iteratively improves those values. VI converges towards the true correct values, but may never reach them. However, it also lacks an effective criterion to determine whether the current value is within some ε-interval around the true value. Tools thus used the standard relative-error criterion: if $v_i(s)$ is the value for state s in iteration i, then they stopped as soon as $\max_s |v_i(s) - v_{i-1}(s)| \leq \alpha \cdot v_i(s)$. However, this does not guarantee $|v_i(s) - v_{true}(s)| \leq \alpha \cdot v_{true}(s)$, where $v_{true}(s)$ is the (unknown) correct value [43]. QComp 2019 allowed the use of VI in this way. Since the benchmark problems and associated results were known, every tool could have chosen to use, for every benchmark instance, the highest α that produces a result satisfying the QComp 2019 correctness criterion of a relative error with $\varepsilon = 10^{-3}$, achieving correctness at optimal VI performance. This would unrealistically over-tweak tools for the competition in a way that no user would be able to do themselves, not knowing the true value on their own model a priori. As a workaround, all participants agreed to use $\alpha = 10^{-6}$, which is the default setting of the Prism model checker, for VI. Although this levelled the playing field for tools using VI, it puts other tools that only implement slower algorithms guaranteeing the required error bound at a disadvantage: they were essentially penalised for producing correct results.

Statistical errors. Those participants that use SMC are unaffected by the VI problem. However, they cannot satisfy the correctness criterion of always ensuring at most an error of relative $\varepsilon = 10^{-3}$ at all: SMC tools estimate the value of interest using random sampling. As such, there is always a chance that the samples happen to be so bad that the result is more than ε off. A typical guarantee is that $\mathbb{P}(|v - v_{true}| > \varepsilon) < \delta$ for $\delta = 0.05$, i.e. one in twenty results may be incorrect. Similar guarantees can be established for the relative error, though fewer statistically correct methods exist for that case. To check whether a tool statistically satisfies the QComp 2019 correctness criterion in such a way would require a statistical test involving many repeated tool executions for each benchmark instance, which is not feasible in a small-scale competition like QComp.

3.2 Correct Algorithms

Since the unsoundness of VI came to the attention of the PMC community, several extensions appeared that compute *intervals* of values v_l and v_u guaranteed to be lower and upper bounds on the true values, respectively. Then a sound relative-error criterion is to stop when $v_u(s_0) - v_l(s_0) \leq \varepsilon \cdot v_l(s_0)$. The algorithms mainly differ in how the upper values are computed. The first was *interval iteration* [11, 44], originally proposed in 2014 [43] concurrently with a learning-based approach [15] that uses the same idea. *Sound value iteration* [80] and most recently *optimistic value iteration* [56] are newer variants with improved performance. Implementations use (double-precision) floating-point arithmetic since the smaller and smaller increments from iteration to iteration do not play well with using unlimited-precision rational numbers. Thus we may still get incorrect approximations due to floating-point imprecisions and error accumulation.

It is possible to obtain exact rational results for some formalism and property type combinations. The algorithms that do so, for example rational search [12] or the topological approaches implemented in Storm (see Sect. 4), are usually much slower and less scalable to large models than the approximative approaches, though. Most of these may also be implemented using floating-point arithmetic, sacrificing unconditional correctness to gain some performance; the only errors caused by such implementations are then due to floating-point imprecisions.

3.3 Correctness in QComp 2020

As a *verification* competition, QComp should in principle not allow tools to deliver incorrect results. However, as we saw above, correctness comes in various forms, and comparing all tools under the least commonly achievable form is

Table 1. Tool capabilities overview (with changes compared to QComp 2019 in red)

Tool	Galileo	GreatSPN	Jani	Modest	PGCL	PPDDL	Prism	Properties															
								DTMC			CTMC				MDP			MA				PTA		
								P	Pr	E	P	Pt	E	S	P	Pr	E	P	Pt	E	S	P	Pt	E
DFTRES	✓		✓					✓			✓	✓		✓				✓	✓		✓			
ePMC			✓				✓	✓		✓	✓	✓	✓		✓		✓							
mcsta			✓	✓				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
modes			✓	✓				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MFPL			✓	✓											✓		✓							
Prism							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓					✓	✓	✓
PET							✓	✓			✓				✓									
Stamina							✓					✓												
Storm	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓

unfair. For QComp 2020, we thus adopted five tracks whose requirements match the different kinds of guarantees provided by the various available approaches:

correct results must match the rational true value, if known, i.e. $\varepsilon = 0$.

floating-point correct results must come from an algorithm that would produce an exact result, except that it may use floating-point arithmetic; correctness is checked w.r.t. $\varepsilon = 10^{-14}$ as an approximation of `double`'s precision.

ε-correct results must *always* be correct up to $\varepsilon = 10^{-6}$; this track matches with the guarantees provided by sound variants of VI.

probably ε-correct results must be correct up to $\varepsilon = 5 \cdot 10^{-2}$ with probability 0.95; this requirement can be satisfied by SMC tools, thus also the higher ε.

often ε-correct results must be correct up to $\varepsilon = 10^{-3}$, but we allow algorithms that do not *always* deliver such precision; thus VI can be used here.

often ε-correct results (10'): instead of being asked to deliver a fixed-precision result, every tool has 10 minutes to obtain as precise a value as possible.

All checks for ε-correctness are for the relative error. The often ε-correct track mirrors the requirements of QComp 2019.

4 Participating Tools

Nine tools participated in QComp 2020. Compared to the previous edition, Probabilistic Fast Downward dropped out, and STAMINA is a new entrant. Table 1 shows the modelling languages, formalisms, and property types supported by all tools. Smaller checkmarks indicate limited support as explained below; red checkmarks highlight new capabilities compared to the version used in QComp 2019.

In the following, we give a brief description of each tool, with more detailed information on the algorithms it uses to achieve the requirements of the different tracks. Table 2 shows the tracks that each tool participates in. For every benchmark instance, tools could provide a default and a specific command line; see Sect. 5 for a detailed explanation of this distinction.

Table 2. Participation of tools in QComp 2020 tracks

track	DFTRES	ePMC	mcsta	modes	MFPL	Prism	PET	Stamina	Storm
correct	—	—	—	—	—	—	—	—	✓
floating-p.	—	—	✓	—	—	—	—	—	✓
ε-correct	—	—	✓	—	—	✓	✓	—	✓
probably ε	✓	—	✓	✓	—	✓	✓	✓	✓
often ε	✓	✓	✓	✓	✓	✓	✓	✓	✓
often ε (10')	✓	—	✓	✓	✓	—	✓	✓	✓

DFTRES [83], the *dynamic fault tree rare event simulator*, is a statistical model checker for dynamic fault trees (DFT) that uses the Path-ZVA algorithm [81] for rare event simulation. Implemented in Java, it works on Linux, macOS, and Windows. It is free and open source, available at github.com/utwente-fmt/DFTRES.

By default, DFTRES uses DFTCALC [5] to parse the Galileo format, with extensions such as repairs and inspections [82]. DFTRES supports Galileo DFT and a subset of Jani with DTMC, CTMC, and MA semantics. In MA, nondeterminism must be spurious, i.e. different choices must result in the same measures. DFTRES implements statistical estimation of system reliability, availability, and mean time to failure (covering Pt-, S-, and P-type properties). Simulations run in parallel on all available processor cores, resulting in near-linear speedup on multi-core systems. Each thread can run importance sampling, e.g. forcing [73] and Path-ZVA, allowing for efficient analysis of rare event behaviour in a modest amount of memory. Path-ZVA is optimised for S properties, but also supports probabilistic reachability. Since it performs a statistical analysis, the guarantees that DFTRES provides—confidence-interval estimates with nominal real-value coverage—match with the probably ε-correct track. Accordingly, it also participates in the often ε-correct track, including the 10-minute variant, without any specific parameters or optimisations for its more relaxed requirements.

The current version of DFTRES is 1.0.1. Since its participation in QComp 2019, it gained support for DTMC and some optimisations: First, the automata in parallel composition are *reduced*: if the composition of two automata will have fewer than 256 states (overapproximated as the product of the individual state space sizes), the automata are replaced by their composition, which is minimised modulo weak bisimulation. Second, the *don't-care optimisation* removes transitions once they can no longer affect observable behaviour. For instance, if one child of a DFT OR gate fails, transitions from the other children are pruned. Finally, for Pt properties—where high-performance cycles cannot be collapsed—a new basic importance sampling scheme boosts runs leaving the cycle.

ePMC (formerly iscasMC [51]) is mainly written in Java, with some performance-critical parts in C. It runs on 64-bit Linux, Mac OS, and Windows. It is available open-source at github.com/ISCAS-PMC/ePMC. It supports the Prism language and Jani as input; DTMC, CTMC, MDP, and stochastic games as formalisms; and PCTL* and reward-based properties. ePMC targets extensibility:

it consists of a small core while plugins provide the ability to parse models, model-check properties of certain types, perform graph-based analyses, or integrate BDD packages [34]. In this way, ePMC can easily be extended for special purposes or experiments without affecting the stability of other parts. ePMC focuses on complex linear-time properties [50] and stochastic parity games [52]. It has been extended to support multi-objective model checking [48] and bisimulation minimisation [49] for interval MDP. It also has experimental support for parametric Markov models [39, 74]. Specialised branches model check quantum Markov chains [37] and epistemic properties of multi-agent systems [38]. However, ePMC so far only implements VI for QComp's formalisms and property types, and thus only participates in the often ε-correct track (but not its 10-minute variant, since it cannot return partial results on early termination).

mcsta is the Modest Toolset's [53] explicit-state probabilistic model checker. The toolset is centred around the Modest modelling language, but also supports Jani. It is implemented in C# and works on 64-bit Linux, macOS, and Windows. Currently at version 3.1, it is freely available at modestchecker.net.

mcsta provides state-of-the-art PMC algorithms for MDP and MA [21]. It also supports PTA (as MDP via digital clocks [70]) as well as DTMC and CTMC (as special cases of MDP and MA, respectively), but does not provide specialised higher-performance algorithms for these submodels. The distinguishing features of mcsta are its disk-based exploration and analysis [54], which allows checking large unstructured models by making use of secondary storage like hard disks and solid-state drives, and its comprehensive support for MA. mcsta participates in the floating-point correct track by attempting to run VI until a (floating-point) fixpoint is reached (not approximated) for P- and E-type properties, and by using state elimination [45] for Pb properties on DTMC, MDP, and PTA. In the ε-correct and probably ε-correct tracks, it uses optimistic value iteration, switching to VI for the often ε-correct track.

Since its participation in QComp 2019, interval iteration for E-type properties, sound value iteration, and optimistic value iteration were implemented in mcsta, considerably improving support for ε-correct results. State-of-the-art algorithms for the analysis of MA were added [21], providing the switch-step algorithm [20] for Pt properties as an alternative to Unif+, and adding support for long-run average rewards (S-type properties). Finally, the essential states reduction [29] brings significant speedups for some models at minimal overhead.

modes [18] is the Modest Toolset's statistical model checker. As a sibling of mcsta, it supports the same platforms and modelling languages. By default, modes rejects models with nondeterminism—since that cannot be simulated—and thus supports DTMC and CTMC. To efficiently estimate rare event probabilities, modes provides rare event simulation methods based on importance splitting [16], with a high degree of automation [17]. It implements lightweight scheduler sampling (LSS) [72] to bring SMC to nondeterministic models like MDP, MA [28], and PTA [27, 59]. LSS chooses m random schedulers resolving the nondeterminism and performs an SMC analysis on the DTMC or CTMC induced by each. Its key insight is how to represent a scheduler in just 32 bits. It needs an adaptated statistical evaluation that takes the repeated tests into account. However, since LSS can only provide upper/lower bounds on minimum/maximum probabilities or rewards with no guaranteed error, and the best choice of m is highly model-dependent, MODES only uses LSS to check MDP, MA, and PTA in the 10-minute variant of the often ε-correct track, sampling as many schedulers as possible within the time limit. In the regular probably ε-correct and often ε-correct tracks, MODES only considers DTMC and CTMC. It does not use rare event simulation in the competition. The main addition to MODES since QComp 2019 is support for S-type properties.

Modest FRET-π LRTDP (MFPL) implements *probabilistic planning* for quantitative formal models, motivated by earlier performance comparisons of using planning algorithms for model checking [64, 65]. Built upon the MODEST TOOLSET in C#, it supports the same input languages as MCSTA and MODES and runs on the same platforms. It is freely available at dgit.cs.uni-saarland.de.

Probabilistic planning uses MDP heuristic search to try to avoid state space explosion by computing values only for a small fraction of the states, just enough for the given property and precision. The algorithms are usually designed for maximum reachability and maximum expected rewards, and assume a specific class of MDP. To apply them to QComp's general MDP problems, they need to be wrapped in FRET iterations [66, 85]. MFPL uses the FRET-π [85] variant of FRET together with the LRTDP [14] heuristic search optimisation of value iteration. Compared to the version used in QComp 2019, which calculated maximum reachability probabilities only, it has been extended with support for minimum and maximum P- and E-type properties. Because MFPL's core is based on VI, it takes part in the often ε-correct track and its 10-minute variant only.

PET is the *partial exploration tool*: an explicit-state model checker for unbounded reachability in discrete-time models. Implemented in Java, it works cross-platform. It uses PRISM as a library for model parsing and exploration, and hence handles PRISM language models, with migration to JANI planned.

PET only partially explores a model's state space, focusing computation on "important" areas [15]: states that are rarely reached can be omitted from the computation if one is only interested in an approximate solution. For each state in the system, the algorithm stores sound upper and lower bounds. It repeatedly samples paths (like in simulation) and back-propagates the bounds on the paths' states as in interval iteration, until convergence, with proper treatment of end components. PET can thus participate in the ε-correct track and all tracks with weaker requirements. Its performance depends on the structure of the model: on some, the PET approach is orders of magnitude faster than standard interval iteration; on the other hand, it is inherently ill-suited for e.g. strongly connected models like restarting mutual exclusion protocols. PET supports (unbounded) P-type properties on MDP, DTMC, and CTMC, plus step-bounded reachability on MDP and DTMC. Truly continuous-time dynamics (such as Pt properties for MA) are not handled yet due to the technical subtleties of such an extension [6].

Since QComp 2019, PET was extended by an SMC module [8] that uses the same basic idea to solve problems where the transition dynamics are not known, and thus have to be learnt. It however is not a competitor to the other tools in QComp since it intentionally ignores information present in the models. Other branches of PET support stochastic games [63] and mean-payoff/S-type properties on DTMC and MDP [7], which, however, are not part of QComp.

Prism [68] is a general-purpose probabilistic model checker with support for a wide range of formalisms and property types. It has been actively developed for 20 years; the first formal release was in 2001. It is implemented in C++ and Java, runs cross-platform, and is open-source, available at prismmodelchecker.org.

Prism supports DTMC, CTMC, MDP, and PTA models specified in the guarded command-based Prism language. It focuses on the ε-correct and often ε-correct tracks. For the former, Markov chains and MDP are solved using interval iteration; for the latter, iterative numerical methods are used for Markov chains and VI for MDP. Bounded properties are always (except on PTA) solved using iterative numerical methods (for DTMC and MDP) or uniformisation (for CTMC), which provide guaranteed error bounds. PTA are solved using stochastic-game abstraction refinement [67]. Prism participates in the probably ε-correct track using the same algorithms as for ε-correct results (thus guaranteeing the requested error with probability 1). While Prism includes an SMC engine, which would more closely match the requirements of the probably ε-correct track, that engine only provides absolute error bounds, not relative ones as required in QComp. Prism does not provide a mechanism for delivering partial results when terminated early, thus it does not participate in the 10-minute often ε-correct variant. Prism incorporates simple heuristics to choose appropriate solution methods based on the type and size of the model and the property being checked; these are mostly used for the specific invocations. In particular, Prism automatically switches to its MTBDD engine for very large models, with a lower threshold for QComp since the larger models here (as in the Prism benchmark suite [69], from which many of them derive) are more likely to perform well with symbolic approaches than might be expected in typical verification scenarios.

Prism participates in QComp 2020 with its current public release, version 4.6. Since the previous edition of the competition, most development on Prism focused on support for models (e.g. stochastic games) or properties (e.g. automata-based specifications) which are not yet part of QComp.

Stamina [76], the *stochastic approximate model checker for infinite-state analysis*, was created in early 2019 with a focus on complex synthetic biological network models. It supports CTMC written in the Prism language and upper-bounded transient CSL properties. Implemented in Java, it runs on Linux and macOS. Stamina iteratively performs state space expansion and calls Prism to perform CTMC analysis. Based on the truncation method [77], Stamina uses property-guided pruning [76] to reduce large and possibly infinite-state CTMC models to finite state representations. Truncation assumes that the probability

Table 3. Overview of algorithms used by Storm

formalism	prop.	(floating-point) correct	(probably) ε-correct	often ε-correct
DTMC, CTMC	P, E	LU-factorisation	optimistic value iter.	gmres
MDP, MA	P, E	policy iter.	optimistic value iter.	value iteration
DTMC, CTMC	S	LU-factorisation	value iteration	gmres
MDP, MA	S	linear programming	value iteration	value iteration
DTMC, MDP	Pb, Eb	matrix-vector mult. (steps), sequential approach (rewards)		
CTMC	Pb, Eb	–	uniformisation	uniformisation
MA	Pb	–	Unif+	Unif+

mass concentrates on a small number of states, and does not distribute uniformly as time progresses. Therefore, Stamina only participates for CTMC with Pt-type properties. Its approach delivers upper and lower bounds on the probabilities being approximated, the difference representing the states that are cut off. Stamina thus participated in the probably ε-correct and often ε-correct tracks.

Motivated by addressing large and infinite-state probabilistic models, Stamina does not require a user to manually bound variables in a Prism model. Its runtime advantage starts to manifest as the state space size grows, as evidenced in [76]. However, QComp only includes three Prism-language CTMC benchmark instances with Pt properties, and in particular no infinite-state models, meaning that Stamina cannot show its strengths in the competition.

Storm [32] is a probabilistic model checker that supports many modelling languages including Jani, the Prism language, DFT, and generalised stochastic Petri nets. Markov models can be built and checked using explicit and decision diagram-based representations. Storm's modular design, efficient C++ core, and extensive Python API yield a powerful toolbox for PMC, parameter synthesis, counterexample generation, fault tree analysis, and many other purposes. Storm has been in active development since 2012. It runs on Linux and macOS, and is open source, available at stormchecker.org.

Storm supports DTMC, CTMC, MDP, and MA. Some PTA models can be checked after converting them to MDP using the Modest Toolset to apply digital clocks [70]. Storm participates in all tracks of QComp 2020. An overview of the algorithms used for each combination of track, formalism, and property type is given in Table 3. For P- and E-type properties, Storm divides the model into strongly connected sub-models that can then be solved individually with the method indicated in the first two rows of Table 3. For the correct track, numbers are represented as infinite-precision rationals. LU-factorisation solves linear equation systems exactly; it is performed within Eigen. Gmres is a fast numerical solution method for systems of linear equations implemented in Gmm++. VI for S-type properties on CTMC and MA [23] provides sound precision guarantees. Storm can also check such properties in MDP and MA exactly by solving a linear program [42] using z3 [75]. Reward-bounded properties are solved using a sequential approach [45, 55] that avoids an expensive unfolding of the model.

Time-bounded properties on CTMC are solved via uniformisation following Fox and Glynn [61]. Unif+ [22, 41] extends uniformisation to MA. Time-bounded properties for CTMC and MA cannot be solved exactly. DFT without repairs are solved with methods that exploit the fault tree structure [87].

Compared to the version used in QComp 2019, STORM now applies optimistic value iteration for ε-correct P- and E-type properties. The implementation of Unif+ [22, 41] has been revamped and now supports relative precision requirements. Model construction has been improved, including support for symbolic MA. Upon timeouts, Storm now reports the best result known so far. The Python interface has been extended and the command line interface streamlined. While experts can still select specific analysis engines, first-time users now benefit from an automatic engine choice: using features of the input JANI model, such as the number of parallel automata or the average variable range, a decision tree predicts the most appropriate model checking approach. To avoid over-fitting, the automatic choice currently only selects among four alternatives: *sparse* (explicit-state), *hybrid* (BDD-based exploration, but explicit data structures for numeric computations), *exact* (like sparse, but using rational arithmetic), and *symbbisim* (like hybrid, but additionally applying symbolic bisimulation minimisation).

STORM implements many alternatives to the aforementioned algorithms. For example, optimistic value iteration can be replaced by interval iteration or sound value iteration. STORM can synthesise high-level counterexamples [30] useful for synthesis loops [24]. In multi-objective model checking, STORM computes Pareto fronts for multi-objective MDP [55] and MA [79] under general and more restricted strategies [33]. Parametric model checking is supported by techniques to (i) compute closed-form solution functions, (ii) divide the parameter space into satisfying and rejecting regions, and (iii) analyse and exploit monotonicities [84]. STORM serves as the backend for the parameter synthesis tool PROPhESY [31, 62]. STORMPY provides a simple Python interface to STORM's underlying data structures, algorithms, and engines which enables rapid prototyping. More details on these and other features of STORM are given in [60].

5 Performance Evaluation

To evaluate the performance of the participating tools, they were executed on benchmark *instances*—a model, fixed values for the model's parameters, and a property—taken from the QVBS. QComp 2020 used the same set of 100 instances as QComp 2019. We also ran the performance evaluation on the same system: a standard desktop with an Intel Core i7-920 CPU and 12 GB of RAM running 64-bit Ubuntu Linux 18.04. Tools were given 30 minutes wall-clock time per instance. We can thus compare the current and previous results in the often ε-correct track. We again allowed every tool to submit two command lines per instance—one running the tool in a default configuration, the other being allowed to use instance-specific parameters to tweak for maximum performance. However, we relaxed the requirements for the default invocations: they need not run the tool in its default configuration (modulo any parameters necessary to achieve the track's correctness requirements), but could instead use the parameters that the tool's authors would *today recommend as defaults* for the given combination of formalism and property type. This is because a tool's default settings may be considered part of its interface, which authors may not want to change for compatibility reasons, even though they would implement different defaults today. This slightly reduces the ability to compare with QComp 2019. The ability to submit specific invocations was not used by all tools, and overall only made a significant difference for Storm, and a noticeable but smaller difference for Prism. In the remainder of this section, we thus mostly show the performance of the default runs. As Storm was the only tool that participated in the correct track, we do not show performance comparison results for this track. Similarly, we found that the model checkers were able to obtain exact results on almost all instances within the time limit of the 10-minute often ε-correct track, rendering our intended comparison of the achieved relative error useless.

On Storm's automatic engine choice. Storm can now automatically select a specific configuration for each benchmark instance, and its authors recommend doing so by default. This, however, would render QComp's distinction between default and specific invocations somewhat pointless. While QComp participants agree that such automatic self-configuration is necessary to improve the usability of quantitative verification tools as they gain more and more analysis engines, algorithms, and parameters, it was not expected to appear in tools for QComp 2020. We will thus drop the default/specific distinction for future competitions. For QComp 2020, we adopted the following pragmatic approach: Storm uses its automatic engine choice by default, and does not use specific invocations. However, this configuration runs *hors concours* for the individual tool comparisons in Sect. 5.2. In addition to Storm, we also evaluate "Storm-static" (abbreviated St.-static): the same version of Storm, but without automatic engine choice. It thus uses today's recommended defaults for the default invocations, and hand-tweaked command lines for the specific comparison. Storm-static *is* included in *all* comparisons. Sect. 5.1 and the bottom-middle plot in Fig. 8 show the drastic performance gains achieved by the automatic engine selection.

Incorrect results. For most—but not all—instances, we have *reference results* obtained via exact algorithms, or reference intervals obtained via sound algorithms using a low ε. Where available, we use these to establish whether a tool delivers an incorrect result. Note that some incorrect results may go undetected because no reference value is available. In all but the often ε-correct and probably ε-correct tracks, tools shall not deliver incorrect results. In the probably ε-correct track, we should expect no more than 5 % of a tool's results to be incorrect.

In the correct track, Storm did not deliver any incorrect results. In the floating-point correct track, mcsta delivered 9, Storm-static 7, and Storm 3 incorrect results. In particular, mcsta terminated on several cyclic models where VI was not expected to reach a fixpoint, indicating that the termination was entirely due to rounding in floating-point computations. In the ε-correct track, mcsta, PET, Prism, and Storm only returned correct result, with Storm-

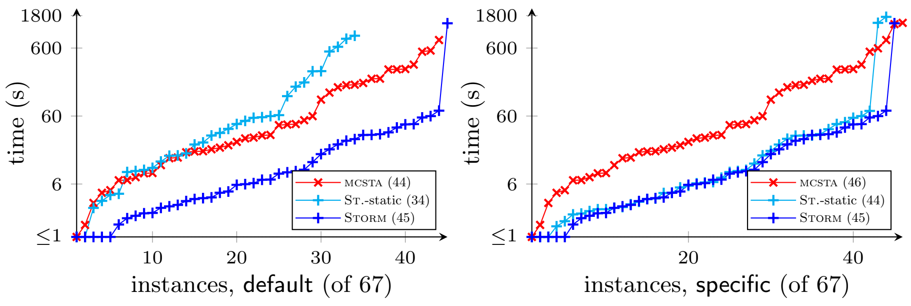

Fig. 1. Quantile plots for the floating-point correct track

static having just one incorrect result in its default invocations. In the probably ε-correct track, where some incorrect results are allowed, one was delivered by each of STORM-static, PRISM, and STAMINA. As tools switched to unsound algorithms for the often ε-correct track, more incorrect results were delivered; see the respective plots in Sect. 5.2 for an indication of their numbers per tool.

5.1 Quantile Plots

We first look at selected subsets of tools via *quantile plots*. We usually only consider the instances supported by *all* of the tools shown in the plot; this is to avoid unsupported instances having the same visual effect as timeouts and errors. For example, for Fig. 1, the intersection of what MCSTA and STORM support contained $n = 67$ instances (shown as "of n" in the x-axis label). The plots' legends indicate the number of correctly solved benchmarks for each tool in parenthesis (i.e. where no timeouts or error occurred and the result was correct). A point $\langle x, y\rangle$ on the line of a tool in this type of plot signifies that the *individual* runtime for the x-th fastest instance as solved by the tool was y seconds.

By ordering instances independently for each tool, quantile plots only allow a comparison of the *total* performance of tools over the included instances. In particular, cases where e.g. a tool is slower overall, but manages to solve some hard instances much faster than any other, will not be visible in a quantile plot. We thus exclude the specialised tools, whose the entire purpose is to solve *some hard* instances better than anyone else, from most of the quantile plots we show.

floating-point correct. The quantile plots in Fig. 1 show that MCSTA's ad-hoc "just try VI" approach to get floating-point correct results turned out to be rather competitive. STORM-static more often timed out and delivered four more incorrect results. The automatic engine selection moves STORM into a class of its own. As the right-hand side of Fig. 1 shows, its performance is only nearly matched by the hand-optimised configurations of the same tool.

ε-correct. In Fig. 2, we compare the general-purpose tools that participated in the ε-correct track. Since PRISM only supports models in the PRISM language,

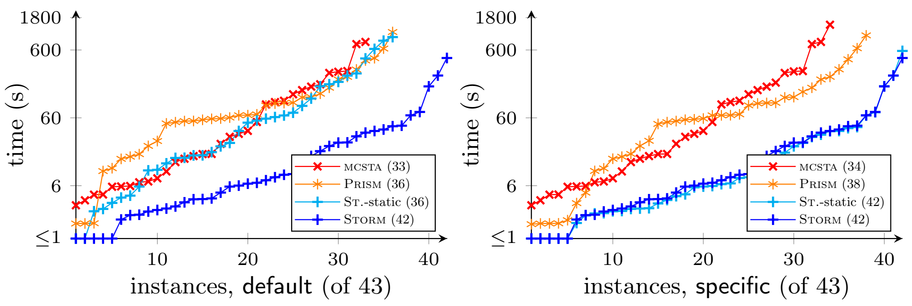

Fig. 2. Quantile plots for the ε-correct track

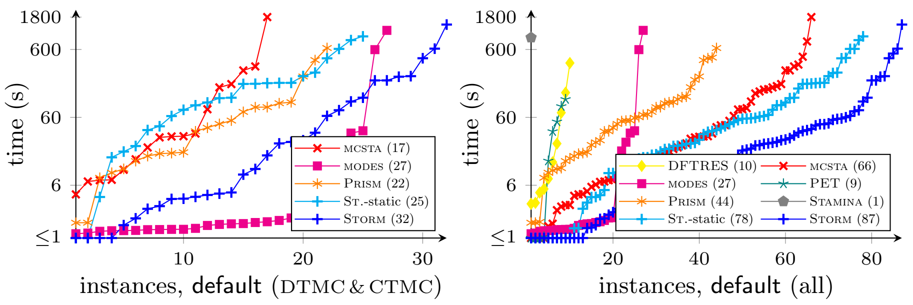

Fig. 3. Quantile plots for the probably ε-correct track

the plots only range over 43 instances; the intersection of what MCSTA and STORM support covers 86 instances. We see that MCSTA, PRISM, and STORM-static perform similarly with default settings. Once it can make use of its wide range of different engines and algorithms, however, STORM cannot be matched.

probably ε-correct. Once statistical model checkers can join in, the competition becomes more diverse. If we plot the results of the probably ε-correct track for the general-purpose tools, the overall relationships remain the same as in Fig. 2, thus we do not show these plots. Instead, we restrict to DTMC and CTMC. Then, we can make a useful comparison that includes MODES, as shown on the left-hand side of Fig. 3. We see that MODES is drastically faster than the model checkers in most cases, needing just a few seconds for more than 20 of the instances. Its runtime only rises significantly when confronted with somewhat rare events (due to the relative-error requirement), and for some complex models where computing the available transitions in itself takes significant computation time. On the right-hand side of Fig. 2, we show a quantile plot over *all* 100 instances and *all* tools in the track. This mainly shows how many instances each tool can solve, but does not do justice to the specialised tools.

often ε-correct. All QComp 2020 participants compete in the often ε-correct track, including in particular the fourth general-purpose model checker, ePMC.

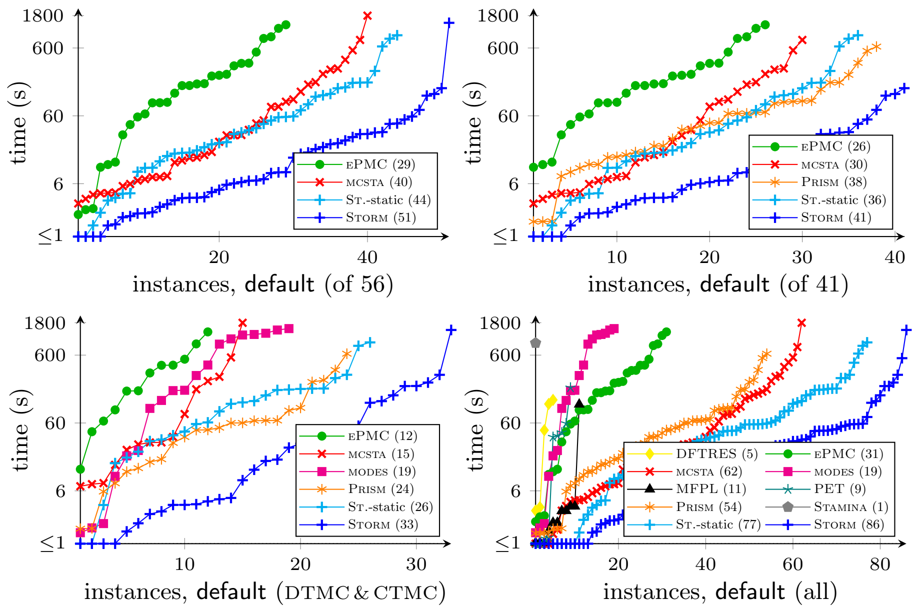

Fig. 4. Quantile plots for the often ε-correct track

We show the results in Fig. 4, limited to default results since few tools supplied and gained from specific invocations. The top two plots in Fig. 4 can be compared with QComp 2019 [46, Fig. 2] modulo the relaxed definition of default settings, while the bottom two plots correspond to Fig. 3. In particular, we see that SMC in the form of MODES is no longer competitive given the much increased precision requirement of $\varepsilon = 10^{-3}$. This confirms the results of earlier comparisons between PMC- and SMC-based methods in different settings [88].

5.2 Scatter Plots

We next show scatter plots that compare the performance of each tool over all individual instances to the best-performing other tool for each instance, using default invocations only. A point $\langle x, y \rangle$ states that the runtime of the plot's tool on one instance was x seconds while the best runtime on the same instance among all other tools *except* STORM *with automatic engine selection*[9] was y seconds. Thus points above the solid diagonal line indicate instances where the plot's tool was the fastest; it was more than ten times faster than any other tool on points above the dotted line. Points on the "TO", "ERR" and "INC" lines indicate instances where the plot's tool encountered a timeout, reported an error (such as running out of memory), or returned an incorrect result, respectively. Points on the "n/a" line indicate instances that none of the other tools was able

[9] STORM, on the other hand, is not compared with STORM-static, thus its "wins n" numbers, marked *, are not part of the same sum as those of the other tools.

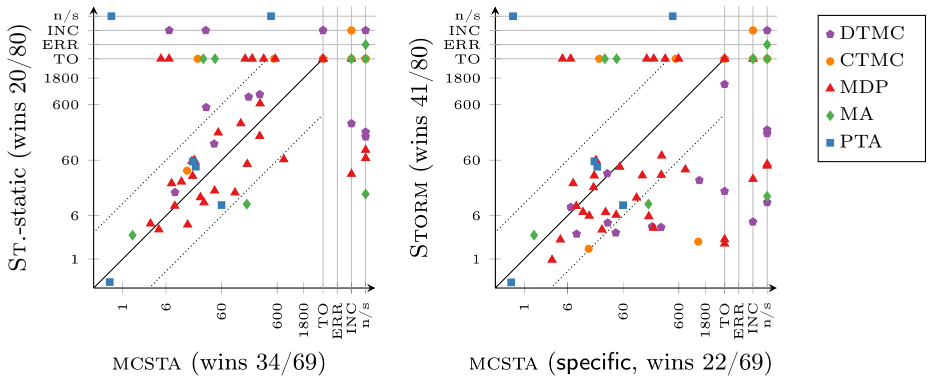

Fig. 5. Scatter plots for the floating-point correct track

to solve. These plots provide more detailed information than the quantile plots since they compare the performance on individual instances, and also include instances outside of the intersections of what is supported by multiple tools. For example, the right-hand plot of Fig. 5 shows that mcsta manages to be faster than Storm on a few instances whereas Fig. 1 looked like mcsta is always slower.

floating-point correct. Fig. 5 compares mcsta to Storm-static and Storm using floating-point correct algorithms. The "n/s" lines indicate instances not supported by the other tool. Storm behaves nearly like Storm-static in specific mode, which is why we show Storm on the right-hand side. Both tools solve several instances where the other fails with a timeout; in the default case, performance is similar when we exclude timeouts. As mentioned, mcsta's approach surprisingly worked, usually correctly, on models where it was not expected to terminate. In summary, the two tools' very different approaches appear complementary, together being able to solve many more instances than each on its own.

ε-correct. Data for the ε-correct track is plotted in Fig. 6. These now include useful data for PET: we see that it times out on most instances, but is the fastest of all tools on nearly half of the ones that it does solve in time. This matches the expectations for an approach highly dependent on the models' structure.

probably ε-correct. Fig. 7 now includes SMC tools for the probably ε-correct track. We do not show Stamina since it works for only three instances, out of which it solves one successfully; the current QComp benchmarks simply do not match Stamina's purpose as discussed in Sect. 4. We now see the typical behaviour of a specialised tool for DFTRES again, with PET showing markedly improved performance relative to the other tools due to the relaxed precision requirement. modes' ability to solve many models in almost no time is evident.

often ε-correct. Fig. 8 provides the details for QComp 2020's largest track, with often ε-correct results. We omit DFTRES (it only solves two instances now,

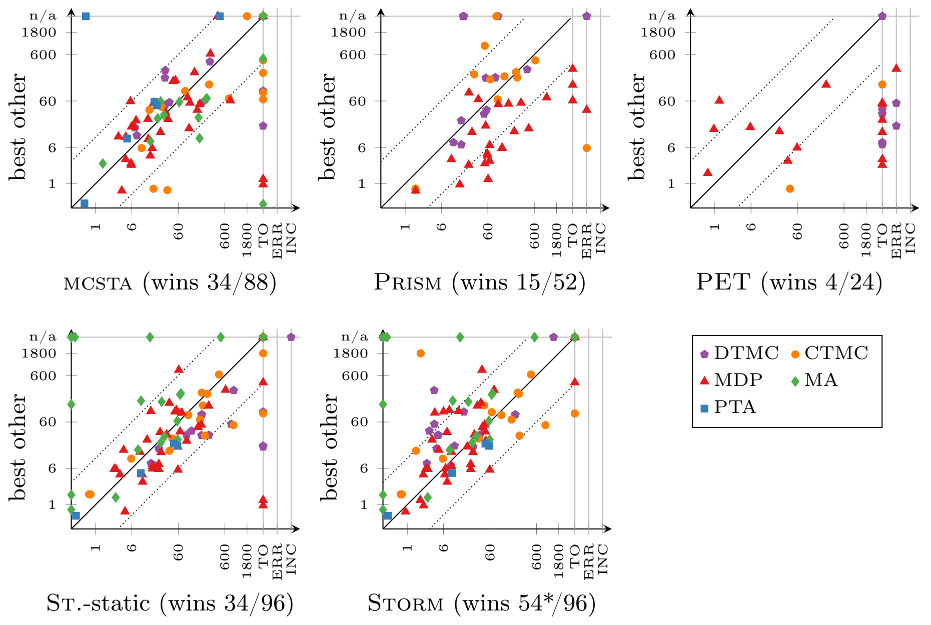

Fig. 6. Scatter plots for the ε-correct track

facing the same problems as MODES from the increased precision requirement), PET (with the same pattern as in the probably ε-correct track at somewhat worse performance), and STAMINA (as before). These plots can be compared with QComp 2019 [46, Figs. 4–6]. The bottom-middle plot compares STORM to STORM-static (default), again highlighting the gains of automatic engine choice.

6 Conclusion

QComp 2020 conservatively extended QComp 2019, focusing on the critical field of problems and performance trade-offs around the correctness and precision of results in quantitative verification. The different tools provide different ranges of guarantees, from exact rational results to no sure guarantees at all in the often ε-correct track. Overall, STORM with its new automatic engine selection dominates the competition. As the first significantly self-configuring model checker in QComp, it advances the usability of PMC tools but also poses challenges to competition design. Still, once we look more deeply into the results—e.g. via scatter plots—we see that *each* tool contributes to solving the QComp benchmark set, and several specialised tools successfully occupy clearly defined niches.

QComp 2020 did not evaluate usability: aside from STORM's improved automation, little has changed, with still only PRISM providing a graphical user interface. In particular, we learned from the previous competition that a usability evaluation needs clear and widely agreed-upon criteria to be useful, and plan

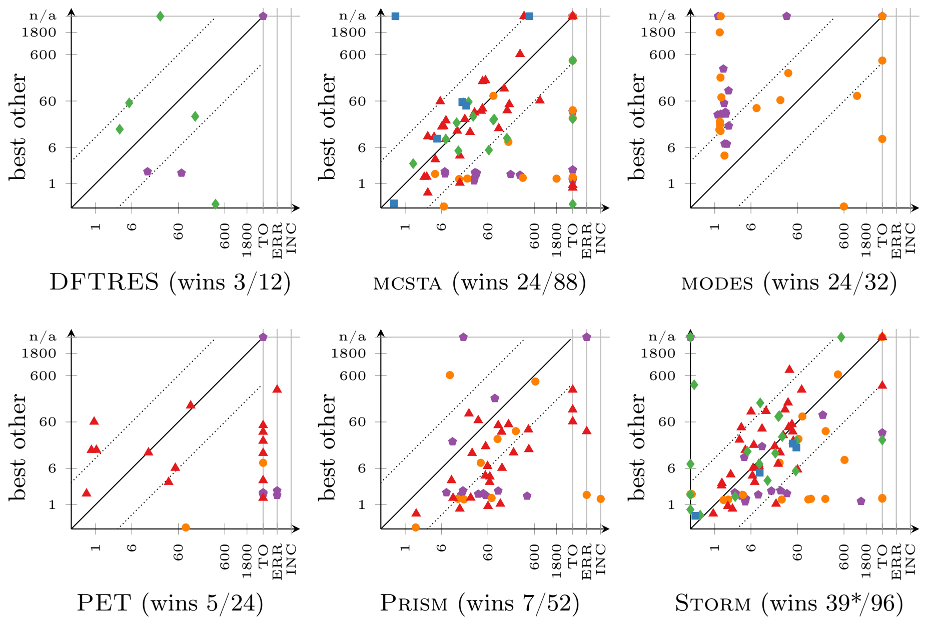

Fig. 7. Scatter plots for the probably ε-correct track

to create such a usability scorecard for a future edition of QComp. More tools now venture into stochastic games, opening a direction to expand QComp.

Roles of authors and acknowledgments. Arnd Hartmanns and Michaela Klauck organised QComp 2020. Carlos E. Budde submitted DFTRES; the tool's main developer is Enno Ruijters. Andrea Turrini submitted ePMC; its main developer is Ernst Moritz Hahn. Arnd Hartmanns develops and submitted MCSTA and MODES; Yuliya Butkova added many new MA model checking algorithms to MCSTA. Michaela Klauck develops and submitted Modest FRET-π LRTDP. Jan Křetínský submitted PET; it is developed by Pranav Ashok, Tobias Meggendorfer, and Maximilian Weininger. David Parker submitted Prism with support from Joachim Klein. Tim Quatmann submitted Storm; it is co-developed by Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Jip Spel, Matthias Volk, and many others. Zhen Zhang submitted Stamina; it is developed by Thakur Neupane, Brett Jepsen, Riley Roberts, and Zhen Zhang.

Data availability. The tools used and data generated in the performance evaluation are archived at qcomp.org and DOI 10.5281/zenodo.3965313 [57].

References

1. Abate, A., Blom, H., Cauchi, N., Degiorgio, K., Fränzle, M., Hahn, E.M., Haesaert, S., Ma, H., Oishi, M., Pilch, C., Remke, A., Salamati, M., Soudjani, S., van Hui-

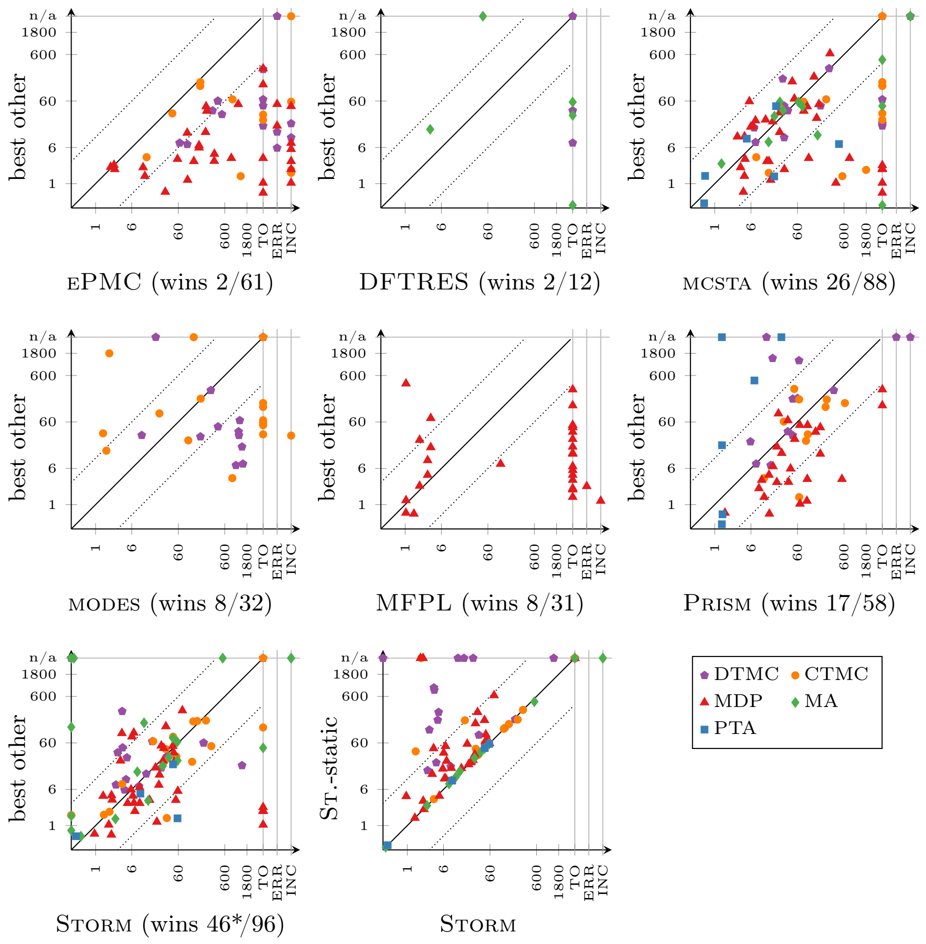

Fig. 8. Scatter plots for the often ε-correct track

jgevoort, B., Vinod, A.P.: ARCH-COMP19 category report: Stochastic modelling. In: ARCH. EPiC Series in Computing, vol. 61, pp. 62–102. EasyChair (2019). doi.org/10.29007/f2vb

2. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comput. Simul. **28**(1), 6:1–6:39 (2018). doi.org/10.1145/3158668
3. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994). doi.org/10.1016/0304-3975(94)90010-8
4. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 years of GreatSPN. In: Principles of Performance and Reliability Modeling and Evaluation. pp. 227–254. Springer (2016). doi.org/10.1007/978-3-319-30599-8_9
5. Arnold, F., Belinfante, A., van der Berg, F., Guck, D., Stoelinga, M.: DFTCalc: A tool for efficient fault tree analysis. In: SAFECOMP. LNCS, vol. 8153, pp. 293–301. Springer (2013). doi.org/10.1007/978-3-642-40793-2_27

6. Ashok, P., Butkova, Y., Hermanns, H., Kretínský, J.: Continuous-time Markov decisions based on partial exploration. In: ATVA. LNCS, vol. 11138, pp. 317–334. Springer (2018). doi.org/10.1007/978-3-030-01090-4_19
7. Ashok, P., Chatterjee, K., Daca, P., Kretínský, J., Meggendorfer, T.: Value iteration for long-run average reward in Markov decision processes. In: CAV. LNCS, vol. 10426, pp. 201–221. Springer (2017). doi.org/10.1007/978-3-319-63387-9_10
8. Ashok, P., Kretínský, J., Weininger, M.: PAC statistical model checking for Markov decision processes and stochastic games. In: CAV. LNCS, vol. 11561, pp. 497–519. Springer (2019). doi.org/10.1007/978-3-030-25540-4_29
9. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking, pp. 963–999. Springer (2018). doi.org/10.1007/978-3-319-10575-8_28
10. Baier, C., Katoen, J.P., Hermanns, H.: Approximate symbolic model checking of continuous-time Markov chains. In: CONCUR. LNCS, vol. 1664, pp. 146–161. Springer (1999). doi.org/10.1007/3-540-48320-9_12
11. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for MDPs. In: CAV. LNCS, vol. 10426, pp. 160–180. Springer (2017). doi.org/10.1007/978-3-319-63387-9_8
12. Bauer, M.S., Mathur, U., Chadha, R., Sistla, A.P., Viswanathan, M.: Exact quantitative probabilistic model checking through rational search. In: FMCAD. pp. 92–99. IEEE (2017). doi.org/10.23919/FMCAD.2017.8102246
13. Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: QEST. pp. 125–126. IEEE Computer Society (2006). doi.org/10.1109/QEST.2006.59
14. Bonet, B., Geffner, H.: Labeled RTDP: Improving the convergence of real-time dynamic programming. In: ICAPS. pp. 12–21. AAAI Press (2003)
15. Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: ATVA. LNCS, vol. 8837, pp. 98–114. Springer (2014). doi.org/10.1007/978-3-319-11936-6_8
16. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Better automated importance splitting for transient rare events. In: SETTA. LNCS, vol. 10606, pp. 42–58. Springer (2017). doi.org/10.1007/978-3-319-69483-2_3
17. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Automated compositional importance splitting. Sci. Comput. Program. **174**, 90–108 (2019). doi.org/10.1016/j.scico.2019.01.006
18. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. STTT (2020), to appear.
19. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative model and tool interaction. In: TACAS. LNCS, vol. 10206, pp. 151–168 (2017). doi.org/10.1007/978-3-662-54580-5_9
20. Butkova, Y., Fox, G.: Optimal time-bounded reachability analysis for concurrent systems. In: TACAS. LNCS, vol. 11428, pp. 191–208. Springer (2019). doi.org/10.1007/978-3-030-17465-1_11
21. Butkova, Y., Hartmanns, A., Hermanns, H.: A Modest approach to modelling and checking Markov automata. In: QEST. LNCS, vol. 11785, pp. 52–69. Springer (2019). doi.org/10.1007/978-3-030-30281-8_4
22. Butkova, Y., Hatefi, H., Hermanns, H., Krcál, J.: Optimal continuous time Markov decisions. In: ATVA. LNCS, vol. 9364, pp. 166–182. Springer (2015). doi.org/10.1007/978-3-319-24953-7_12

23. Butkova, Y., Wimmer, R., Hermanns, H.: Long-run rewards for Markov automata. In: TACAS. LNCS, vol. 10206, pp. 188–203 (2017). doi.org/10.1007/978-3-662-54580-5_11
24. Ceska, M., Hensel, C., Junges, S., Katoen, J.P.: Counterexample-driven synthesis for probabilistic program sketches. In: FM. LNCS, vol. 11800, pp. 101–120. Springer (2019). doi.org/10.1007/978-3-030-30942-8_8
25. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods Syst. Des. **43**(1), 61–92 (2013). doi.org/10.1007/s10703-013-0183-7
26. Courtney, T., Gaonkar, S., Keefe, K., Rozier, E., Sanders, W.H.: Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In: DSN. pp. 353–358. IEEE Computer Society (2009). doi.org/10.1109/DSN.2009.5270318
27. D'Argenio, P.R., Hartmanns, A., Legay, A., Sedwards, S.: Statistical approximation of optimal schedulers for probabilistic timed automata. In: iFM. LNCS, vol. 9681, pp. 99–114. Springer (2016). doi.org/10.1007/978-3-319-33693-0_7
28. D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: ISoLA. LNCS, vol. 11245, pp. 336–353. Springer (2018). doi.org/10.1007/978-3-030-03421-4_22
29. D'Argenio, P.R., Jeannet, B., Jensen, H.E., Larsen, K.G.: Reduction and refinement strategies for probabilistic analysis. In: PAPM-PROBMIV. LNCS, vol. 2399, pp. 57–76. Springer (2002). doi.org/10.1007/3-540-45605-8_5
30. Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.P.: Fast debugging of PRISM models. In: ATVA. LNCS, vol. 8837, pp. 146–162. Springer (2014). doi.org/10.1007/978-3-319-11936-6_11
31. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.P., Ábrahám, E.: PROPhESY: A probabilistic parameter synthesis tool. In: CAV. LNCS, vol. 9206, pp. 214–231. Springer (2015). doi.org/10.1007/978-3-319-21690-4_13
32. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: CAV. LNCS, vol. 10427, pp. 592–600. Springer (2017). doi.org/10.1007/978-3-319-63390-9_31
33. Delgrange, F., Katoen, J.P., Quatmann, T., Randour, M.: Simple strategies in multi-objective MDPs. In: TACAS. LNCS, vol. 12078, pp. 346–364. Springer (2020). doi.org/10.1007/978-3-030-45190-5_19
34. van Dijk, T., Hahn, E.M., Jansen, D.N., Li, Y., Neele, T., Stoelinga, M., Turrini, A., Zhang, L.: A comparative study of BDD packages for probabilistic symbolic model checking. In: SETTA. LNCS, vol. 9409, pp. 35–51. Springer (2015). doi.org/10.1007/978-3-319-25942-0_3
35. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS. pp. 342–351. IEEE Computer Society (2010). doi.org/10.1109/LICS.2010.41
36. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. Logical Methods in Computer Science **4**(4) (2008). doi.org/10.2168/LMCS-4(4:8)2008
37. Feng, Y., Hahn, E.M., Turrini, A., Ying, S.: Model checking omega-regular properties for quantum Markov chains. In: CONCUR. LIPIcs, vol. 85, pp. 35:1–35:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2017). doi.org/10.4230/LIPIcs.CONCUR.2017.35

38. Fu, C., Turrini, A., Huang, X., Song, L., Feng, Y., Zhang, L.: Model checking probabilistic epistemic logic for probabilistic multiagent systems. In: IJCAI. pp. 4757–4763. ijcai.org (2018). doi.org/10.24963/ijcai.2018/661
39. Gainer, P., Hahn, E.M., Schewe, S.: Accelerated model checking of parametric Markov chains. In: ATVA. LNCS, vol. 11138, pp. 300–316. Springer (2018). doi.org/10.1007/978-3-030-01090-4_18
40. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: FOSE. pp. 167–181. ACM (2014). doi.org/10.1145/2593882.2593900
41. Gros, T.P.: Markov automata taken by Storm. Master's thesis, Saarland University, Germany (2018)
42. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: QEST. LNCS, vol. 8054, pp. 55–71. Springer (2013). doi.org/10.1007/978-3-642-40196-1_5
43. Haddad, S., Monmege, B.: Reachability in MDPs: Refining convergence of value iteration. In: RP. LNCS, vol. 8762, pp. 125–137. Springer (2014). doi.org/10.1007/978-3-319-11439-2_10
44. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. **735**, 111–131 (2018). doi.org/10.1016/j.tcs.2016.12.003
45. Hahn, E.M., Hartmanns, A.: A comparison of time- and reward-bounded probabilistic model checking techniques. In: SETTA. LNCS, vol. 9984, pp. 85–100 (2016). doi.org/10.1007/978-3-319-47677-3_6
46. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models (QComp 2019 competition report). In: TACAS: TOOLympics. LNCS, vol. 11429, pp. 69–92. Springer (2019). doi.org/10.1007/978-3-030-17502-3_5
47. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013). doi.org/10.1007/s10703-012-0167-z
48. Hahn, E.M., Hashemi, V., Hermanns, H., Lahijanian, M., Turrini, A.: Multi-objective robust strategy synthesis for interval MDPs. In: QEST. LNCS, vol. 10503, pp. 207–223. Springer (2017). doi.org/10.1007/978-3-319-66335-7_13
49. Hahn, E.M., Hashemi, V., Hermanns, H., Turrini, A.: Exploiting robust optimization for interval probabilistic bisimulation. In: QEST. LNCS, vol. 9826, pp. 55–71. Springer (2016). doi.org/10.1007/978-3-319-43425-4_4
50. Hahn, E.M., Li, G., Schewe, S., Zhang, L.: Lazy determinisation for quantitative model checking. CoRR **abs/1311.2928** (2013), arxiv.org/abs/1311.2928
51. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: A web-based probabilistic model checker. In: FM. LNCS, vol. 8442, pp. 312–317. Springer (2014). doi.org/10.1007/978-3-319-06410-9_22
52. Hahn, E.M., Schewe, S., Turrini, A., Zhang, L.: A simple algorithm for solving qualitative probabilistic parity games. In: CAV. LNCS, vol. 9780, pp. 291–311. Springer (2016). doi.org/10.1007/978-3-319-41540-6_16
53. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: TACAS. LNCS, vol. 8413, pp. 593–598. Springer (2014). doi.org/10.1007/978-3-642-54862-8_51
54. Hartmanns, A., Hermanns, H.: Explicit model checking of very large MDP using partitioning and secondary storage. In: ATVA. LNCS, vol. 9364, pp. 131–147. Springer (2015). doi.org/10.1007/978-3-319-24953-7_10

55. Hartmanns, A., Junges, S., Katoen, J.P., Quatmann, T.: Multi-cost bounded reachability in MDP. In: TACAS. LNCS, vol. 10806, pp. 320–339. Springer (2018). doi.org/10.1007/978-3-319-89963-3_19
56. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: CAV. LNCS, vol. 12225, pp. 488–511. Springer (2020). doi.org/10.1007/978-3-030-53291-8_26
57. Hartmanns, A., Klauck, M.: The 2020 Comparison of Tools for the Analysis of Quantitative Formal Models: Results and Reproduction. Zenodo (2020). doi.org/10.5281/zenodo.3965313
58. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS. LNCS, vol. 11427, pp. 344–350. Springer (2019). doi.org/10.1007/978-3-030-17462-0_20
59. Hartmanns, A., Sedwards, S., D'Argenio, P.R.: Efficient simulation-based verification of probabilistic timed automata. In: Winter Sim. Conf. pp. 1419–1430. IEEE (2017). doi.org/10.1109/WSC.2017.8247885
60. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker Storm. CoRR **abs/2002.07080** (2020), arxiv.org/abs/2002.07080
61. Jansen, D.N.: Understanding Fox and Glynn's "Computing Poisson probabilities". CTIT technical report series (2011)
62. Junges, S., Ábrahám, E., Hensel, C., Jansen, N., Katoen, J.P., Quatmann, T., Volk, M.: Parameter synthesis for Markov models. CoRR **abs/1903.07993** (2019), arxiv.org/abs/1903.07993
63. Kelmendi, E., Krämer, J., Kretínský, J., Weininger, M.: Value iteration for simple stochastic games: Stopping criterion and learning algorithm. In: CAV. LNCS, vol. 10981, pp. 623–642. Springer (2018). doi.org/10.1007/978-3-319-96145-3_36
64. Klauck, M., Steinmetz, M., Hoffmann, J., Hermanns, H.: Compiling probabilistic model checking into prob. planning. In: ICAPS. pp. 150–154. AAAI Press (2018)
65. Klauck, M., Steinmetz, M., Hoffmann, J., Hermanns, H.: Bridging the gap between probabilistic model checking and probabilistic planning: Survey, compilations, and empirical comparison. J. Artif. Intell. Res. **68**, 247–310 (2020). doi.org/10.1613/jair.1.11595
66. Kolobov, A., Mausam, Weld, D.S., Geffner, H.: Heuristic search for generalized stochastic shortest path MDPs. In: ICAPS. AAAI Press (2011)
67. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: FORMATS. LNCS, vol. 5813, pp. 212–227. Springer (2009). doi.org/10.1007/978-3-642-04368-0_17
68. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011). doi.org/10.1007/978-3-642-22110-1_47
69. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: QEST. pp. 203–204. IEEE Computer Society (2012). doi.org/10.1109/QEST.2012.14
70. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods Syst. Des. **29**(1), 33–78 (2006). doi.org/10.1007/s10703-006-0005-2
71. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. **282**(1), 101–150 (2002). doi.org/10.1016/S0304-3975(01)00046-9
72. Legay, A., Sedwards, S., Traonouez, L.M.: Scalable verification of Markov decision processes. In: WS-FMDS at SEFM. LNCS, vol. 8938, pp. 350–362. Springer (2014). doi.org/10.1007/978-3-319-15201-1_23

73. Lewis, E., Böhm, F.: Monte Carlo simulation of Markov unreliability models. Nuclear Eng. and Design **77**(1), 49–62 (1984). doi.org/10.1016/0029-5493(84)90060-8
74. Li, Y., Liu, W., Turrini, A., Hahn, E.M., Zhang, L.: An efficient synthesis algorithm for parametric Markov chains against linear time properties. CoRR **abs/1605.04400** (2016)
75. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008). doi.org/10.1007/978-3-540-78800-3_24
76. Neupane, T., Myers, C.J., Madsen, C., Zheng, H., Zhang, Z.: STAMINA: Stochastic approximate model-checker for infinite-state analysis. In: CAV. LNCS, vol. 11561, pp. 540–549. Springer (2019). doi.org/10.1007/978-3-030-25540-4_31
77. Neupane, T., Zhang, Z., Madsen, C., Zheng, H., Myers, C.J.: Approximation techniques for stochastic analysis of biological systems. In: Automated Reasoning for Systems Biology and Medicine, Computational Biology, vol. 30, pp. 327–348. Springer (2019). doi.org/10.1007/978-3-030-17297-8_12
78. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). doi.org/10.1002/9780470316887
79. Quatmann, T., Junges, S., Katoen, J.P.: Markov automata with multiple objectives. In: CAV. LNCS, vol. 10426, pp. 140–159. Springer (2017). doi.org/10.1007/978-3-319-63387-9_7
80. Quatmann, T., Katoen, J.P.: Sound value iteration. In: CAV. LNCS, vol. 10981, pp. 643–661. Springer (2018). doi.org/10.1007/978-3-319-96145-3_37
81. Reijsbergen, D., de Boer, P.T., Scheinhardt, W.R.W., Juneja, S.: Path-ZVA: General, efficient, and automated importance sampling for highly reliable Markovian systems. ACM Trans. Model. Comput. Simul. **28**(3), 22:1–22:25 (2018). doi.org/10.1145/3161569
82. Ruijters, E., Budde, C.E., Nakhaee, M.C., Stoelinga, M., Bucur, D., Hiemstra, D., Schivo, S.: FFORT: A benchmark suite for fault tree analysis. In: ESREL (2019). doi.org/10.3850/978-981-11-2724-3_0641-cd
83. Ruijters, E., Reijsbergen, D., de Boer, P.T., Stoelinga, M.: Rare event simulation for dynamic fault trees. Reliab. Eng. Syst. Saf. **186**, 220–231 (2019). doi.org/10.1016/j.ress.2019.02.004
84. Spel, J., Junges, S., Katoen, J.P.: Are parametric Markov chains monotonic? In: ATVA. LNCS, vol. 11781, pp. 479–496. Springer (2019). doi.org/10.1007/978-3-030-31784-3_28
85. Steinmetz, M., Hoffmann, J., Buffet, O.: Goal probability analysis in probabilistic planning: Exploring and enhancing the state of the art. J. Artif. Intell. Res. **57**, 229–271 (2016). doi.org/10.1613/jair.5153
86. Sullivan, K.J., Dugan, J.B., Coppit, D.: The Galileo fault tree analysis tool. In: FTCS. pp. 232–235. IEEE Computer Society (1999). doi.org/10.1109/FTCS.1999.781056
87. Volk, M., Junges, S., Katoen, J.P.: Fast dynamic fault tree analysis by model checking techniques. IEEE Trans. Ind. Informatics **14**(1), 370–379 (2018). doi.org/10.1109/TII.2017.2710316
88. Younes, H.L.S., Kwiatkowska, M.Z., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. Int. J. Softw. Tools Technol. Transf. **8**(3), 216–228 (2006). doi.org/10.1007/s10009-005-0187-8
89. Younes, H.L.S., Littman, M.L., Weissman, D., Asmuth, J.: The first probabilistic track of the International Planning Competition. J. Artif. Intell. Res. **24**, 851–887 (2005). doi.org/10.1613/jair.1880