The 2019 Comparison of Tools for the Analysis of Quantitative Formal Models[⋆] (QComp 2019 Competition Report)

Ernst Moritz Hahn[1,2], Arnd Hartmanns[3], Christian Hensel[4], Michaela Klauck[5], Joachim Klein[6], Jan Křetínský[7], David Parker[8], Tim Quatmann[4], Enno Ruijters[3], and Marcel Steinmetz[5]

[1] School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast, United Kingdom
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
[3] University of Twente, Enschede, The Netherlands
[4] RWTH Aachen University, Aachen, Germany
[5] Saarland Informatics Campus, Saarland University, Saarbrücken, Germany
[6] Technische Universität Dresden, Dresden, Germany
[7] Technische Universität München, Munich, Germany
[8] University of Birmingham, Birmingham, United Kingdom

Abstract. Quantitative formal models capture probabilistic behaviour, real-time aspects, or general continuous dynamics. A number of tools support their automatic analysis with respect to dependability or performance properties. QComp 2019 is the first, friendly competition among such tools. It focuses on stochastic formalisms from Markov chains to probabilistic timed automata specified in the Jani model exchange format, and on probabilistic reachability, expected-reward, and steady-state properties. QComp draws its benchmarks from the new Quantitative Verification Benchmark Set. Participating tools, which include probabilistic model checkers and planners as well as simulation-based tools, are evaluated in terms of performance, versatility, and usability. In this paper, we report on the challenges in setting up a quantitative verification competition, present the results of QComp 2019, summarise the lessons learned, and provide an outlook on the features of the next edition of QComp.

1 Introduction

Classic verification is concerned with functional, *qualitative* properties of models of systems or software: Can this assertion ever be violated? Will the server always

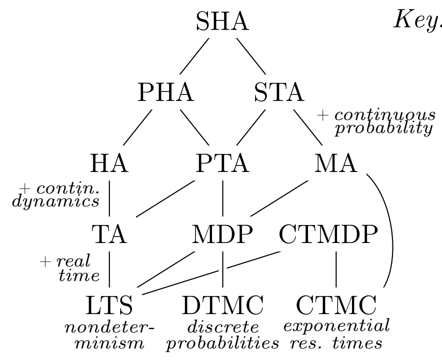
[⋆] The authors are listed in alphabetical order. This work was supported by BMBF grant 16KIS0656 (cispa), DFG grants 383882557 (suv), 389792660 (part of CRC 248), and HO 2169/5-1, DFG SFB 912 (haec), ERC Advanced Grants 695614 (powver) and 781914 (frappant), Natural Science Foundation of China (NSFC) grants 61761136011 and 61532019, NWO and BetterBe B.V. grant 628.010.006, NWO VENI grant 639.021.754, and the TUM IGSSE project 10.06 (parsec).

eventually answer a request? To evaluate aspects of dependability (e.g. safety, reliability, availability or survivability) and performance (e.g. response times, throughput, or power consumption), however, *quantitative* properties must be checked on *quantitative* models that incorporate probabilities, real-time aspects, or general continuous dynamics. Over the past three decades, many modelling languages for mathematical formalisms such as Markov chains or timed automata have been specified for use by quantitative verification tools that automatically check or compute values such as expected accumulated rewards or PCTL formulae. Applications include probabilistic programs, safety-critical and fault-tolerant systems, biological processes, queueing systems, privacy, and security.

As a research field matures, developers of algorithms and tools face increasing challenges in comparing their work with the state of the art: the number of incompatible modelling languages grows, benchmarks and case studies become scattered and hard to obtain, and the tool prototypes used by others disappear. At the same time, it is hard to motivate spending effort on engineering generic, user-friendly, well-documented tools. In several areas, *tool competitions* have successfully addressed these challenges: they improve the visibility of existing tools, motivate engineering effort, and push for standardised interfaces, languages, and benchmarks. Examples include ARCH-COMP [29] for hybrid systems, the International Planning Competition [18] for planners, the SAT Competition [51] for satisfiability solvers, and SV-COMP [8] for software verification.

In this paper, we present QComp 2019: the first, friendly competition among quantitative verification tools. As the first event of its kind, its scope is intentionally limited to five stochastic formalisms based on Markov chains and to basic property types. It compares the performance, versatility, and usability of four general-purpose probabilistic model checkers, one general-purpose statistical model checker, and four specialised tools (including two probabilistic planners). All competition data is available at qcomp.org. As a friendly competition in a spirit similar to ARCH-COMP and the RERS challenge [52], QComp's focus is less on establishing a ranking among tools, but rather on gathering a community to agree on common formats, challenges, and evaluation criteria. To this end, QComp is complemented by a new collection of benchmarks, the Quantitative Verification Benchmark Set (QVBS, [46]). All models in the QVBS are available in their original modelling language as well as the JANI model exchange format [15]. While JANI is intended as the standard format for QComp, not all tools implement support for it yet and were thus executed only on those benchmarks for which they support the original modelling language.

Quantitative verification is rich in formalisms, modelling languages, types of properties, and verification approaches, of which we give an overview in Sect. 2. We summarise the selections made by QComp among all of these options as well as the overall competition design in Sect. 3. The authors of the participating tools describe the features and capabilities of their tools in Sect. 4; we then compare their usability and versatility in Sect. 5. Finally, Sect. 6 contains the technical setup and results of the performance comparison, followed by an outlook on the next edition of QComp, based on the lessons learned in this round, in Sect. 7.



Key:	SHA	stochastic hybrid automata [28]
	PHA	probabilistic hybrid automata [70]
	STA	stochastic timed automata [9]
	HA	hybrid automata
	PTA	probabilistic timed automata [59]
	MA	Markov automata [25]
	TA	timed automata
	MDP	Markov decision processes
	CTMDP	continuous-time MDP
	LTS	labelled transition systems
	DTMC	discrete-time Markov chains
	CTMC	continuous-time Markov chains

Fig. 1. The family tree of automata-based quantitative formalisms

2 The Quantitative Verification Landscape

Quantitative verification is a wide field that overlaps with safety and fault tolerance, performance evaluation, real-time systems, simulation, optimisation, and control theory. In this section, we give an overview of the formalisms, modelling languages, property types, and verification methods considered for QComp.

2.1 Semantic Formalisms

The foundation of every formal verification approach is a *formalism*: a mathematically well-defined class of objects that form the semantics of any concrete model. Most modelling languages or higher-level formalisms eventually map to some extension of automata: states (that may contain relevant structure) and transitions (that connect states, possibly with several annotations). In Fig. 1, we list the automata-based formalisms supported by Jani, and graphically show their relationships (with a higher-up formalism being an extension of the lower-level formalisms it is connected to). LTS are included as the most basic non-quantitative automata formalism; TA then add the quantity of (continuous) time, while DTMC and CTMC provide probabilistic behaviour. The list is clearly not exhaustive: for example, every formalism is a 1- or 1.5-player game, and the list could be extended by games with two or more players that capture competitive behaviour among actors with possibly conflicting goals. It also does not include higher-level formalisms such as Petri nets or dataflow that often provide extra information for verification compared to their automata semantics.

2.2 Modelling Languages

Modelling complex systems using the formalisms listed above directly would be cumbersome. Instead, domain experts use (textual or graphical) *modelling languages* to compactly describe large automata. Aside from providing a concrete human-writable and machine-readable syntax for a formalism, modelling languages typically add at least discrete variables and some form of compositionality. The current benchmarks in the QVBS were originally specified in the Galileo format [72] for fault trees, the GreatSPN format [1] for generalised stochastic Petri nets, the process algebra-based high-level modelling language Modest [36], the PGCL specification for probabilistic programs [32], PPDDL for probabilistic planning domains [77], and the lower-level guarded-command PRISM language [57]. For all benchmarks, the QVBS provides a translation to the tool-independent JSON-based Jani model exchange format [15]. The purpose of Jani is to establish a standard human-readable (though not easily human-writable) format for quantitiative verification that simplifies the implementation of new tools and fosters model exchange and tool interoperability. Many other quantitative modelling languages not yet represented in the QVBS exist such as Uppaal's XML format [7] for timed automata or those supported by Möbius [19].

2.3 Properties

Models are verified w.r.t. *properties* that specify a requirement or a query for a value of interest. The basic property types for stochastic models are probabilistic reachability (the probability to eventually reach a goal state), expected accumulated rewards (or costs; the expected reward sum until reaching a goal state), and steady-state values (the steady-state probability to be in certain states or the long-run average reward). In case of formalisms with nondeterminism, properties ask for the minimum or maximum value over all resolutions of nondeterminism. Probabilistic reachability and expected rewards can be bounded by a maximum number of transitions taken, by time, or by accumulated reward; we can then query for e.g. the maximum probability to reach a goal within a cost budget. We refer to properties that query for probabilities as *probabilistic*, to those that deal with expected rewards as *reward-based*, and to *steady-state* properties.

From these 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 steps from which the probability of eventually reaching an unsafe state is less than 10^{-9}. Examples are CSL [5] for CTMC, PTCTL [59] for PTA, rPATL [17] for stochastic games, and STL [61] for hybrid systems. Another interesting class of properties are *multi-objective* tradeoffs [26], which query for Pareto-optimal strategies balancing multiple goals.

2.4 Verification Methods and Results

The two main quantitative verification approaches are probabilistic model checking and statistical model checking a.k.a. Monte Carlo simulation. Probabilistic planners use ideas similar to probabilistic model checking, but focus on heuristics and bounding methods to avoid the state space explosion problem.

Probabilistic model checking [4] is to explore a model's state space followed by or interleaved with a numeric analysis, e.g. using value iteration, to compute probabilities or reward values. It aims for results with *hard* guarantees, i.e. precise statements about the relationship between the computed result and the actual value. For example, a probabilistic model checker may guarantee that the actual probability is definitely within $\epsilon = \pm 10^{-3}$ of the reported value. Due to the need for state space exploration, these tools face the state space explosion problem and their applicability to large models is typically limited by available memory.

Statistical model checking (SMC, [49,78]) is Monte Carlo simulation on formal models: generate n executions of the model, determine how many of them satisfy the property or calculate the reward of each, and return the average as an estimate for the property's value. SMC is thus not directly applicable to models with nondeterminism and provides only statistical guarantees, for example that $\mathbb{P}(|\hat{p} - p| > \epsilon) < \delta$ where p is the (unknown) actual probability, $\hat{p}$ is the estimate, and $1 - \delta$ is the confidence that the result is ϵ-correct. As ϵ and δ decrease, n grows. SMC is attractive as it only requires constant memory independent of the size of the state space. Compared to model checking, it replaces the state space explosion problem by a runtime explosion problem when faced with rare events: it is desirable that $\epsilon \ll p$, but since n depends quadratically on ϵ for a fixed δ (e.g. in the Okamoto bound [63]), n becomes prohibitively large as p reaches around 10^{-4}. Rare event simulation [68] provides methods to tackle this problem at the cost of higher memory usage, lack of automation, or lower generality.

Probabilistic planning uses MDP heuristic search algorithms, e.g. [10,11], that try to avoid the state space explosion problem by computing values only for a small fraction of the states, just enough to answer the considered property. Heuristics—admissible approximations of the optimal values—are used to initialise the value function, which is subsequently updated until the value for the initial state has provably converged. The order of updates depends on the current values; this sometimes allows to prove states to not be part of any optimal solution *before* actually visiting all of their descendants. Such states can safely be ignored. Many heuristic search algorithms assume a specific class of MDP. To apply them to general MDP, they need to be wrapped in FRET iterations [54]: between calls to the search algorithm, FRET eliminates end components from the subgraph of the state space induced by optimal actions w.r.t. the current values. FRET-π [71] is a variant that only picks a single optimal path to the goal.

Results. The answer to a property may be a concrete number that is in some relation to the actual value (e.g. within $\pm 10^{-3}$ of the actual value). However, properties—such as PCTL formulae—may also ask qualitative questions, i.e. whether the value of interest is above or below a certain constant bound. In that case, there is an opportunity for algorithms to terminate early: they may not have computed a value close to the actual one yet, but the current approximation may already be sufficient to prove or disprove the bound. In the case of models with nondeterminism, those choices can be seen as scheduling freedom, and a user may be more interested in an optimal or sufficient *strategy* than in the actual value, i.e. in a way to resolve the nondeterministic choices to achieve the optimal or a sufficient probability or reward. Further types of quantitative results include *quantiles* [73], Pareto curves in multi-objective scenarios, and a function in terms of some model parameter in case of parametric model checking.

3 Decisions and Competition Setup

Seeing the wide range of options in quantitative verification described in the previous section, and taking into account that QComp 2019 was the first event of its kind, several decisions had to be made to limit its scope. The first was to build on Jani and the QVBS: only benchmarks available in Jani and submitted to the QVBS with a description and extensive metadata would become part of the QComp performance evaluation. We further limited the formalisms to DTMC, CTMC, MDP, MA and PTA (cf. Fig. 1). We thus included only stochastic formalisms, excluding in particular TA and HA. This is because stochastic formalisms provide more ways to exploit approximations and trade precision for runtime and memory than non-stochastic ones where verification is rather "qualitative with more complicated states". Second, we only included formalisms supported by at least two participating tools, which ruled out STA, PHA and SHA. For the same reason, we restricted to the basic properties listed at the beginning of Sect. 2.3. While many competitions focus on performance, producing an overall ranking of tools w.r.t. their total runtime over all benchmarks, QComp equally considers versatility and usability (see Sect. 5). For the performance comparison, many technical decisions (such as comparing quantitative results with an a priori fixed precision and not considering comparisons or asking for strategies) were made as explained in Sect. 6. In particular, the set of benchmarks was determined based on the wishes of the participants and announced a priori; not expecting tool authors to dubiously tweak their tools for the selected benchmarks is in line with the friendly nature of QComp 2019. The entire competition was then performed *offline*: participants submitted benchmarks and tools, the performance comparison was done by the organisers on a central server according to tool setup instructions and scripts provided by the participants, and the evaluation of versatility and usability is based on submitted tool descriptions.

4 Participating Tools

QComp is open to every tool that can check a significant subset of the models and properties of the QVBS. In particular, a participating tool need not support all model types, the Jani format, or all included kinds of properties. For example, a tool specialising in the analysis of stochastic Petri nets is not expected to solve Jani DTMC models. Nine tools were submitted to QComp 2019: DFTRES [69] (by Enno Ruijters), ePMC [40] (by Ernst Moritz Hahn), mcsta [42] and modes [14] (by Arnd Hartmanns), Modest FRET-π LRTDP (by Michaela Klauck, MFPL for short), PRISM [57] (by Joachim Klein and David Parker), PRISM-TUMheuristics (by Jan Křetínský, P-TUM for short), Probabilistic Fast Downward [71] (by Marcel Steinmetz, PFD for short), and Storm [23] (by Christian Hensel). We summarise the tools' capabilities w.r.t. the supported modelling

Table 1. Tool capabilities

Tool	Galileo	GreatSPN	Jani	Modest	PGCL	PPDDL	PRISM	Properties																
								DTMC			CTMC				MDP			MA				PTA		
								P	P_r	E	P	P_t	E	S	P	P_r	E	P	P_t	E	S	P	P_t	E
ePMC			✓				✓	✓		✓	✓	✓	✓		✓		✓							
mcsta			✓	✓				✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		✓	✓	✓
PRISM							✓	✓		✓	✓	✓	✓	✓	✓		✓					✓	✓	✓
P-TUM							✓	✓			✓				✓									
Storm	✓	✓	✓		✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	(✓)		(✓)
DFTRES	✓		(✓)									✓		✓					(✓)		(✓)			
modes			✓	✓				✓	✓	✓	✓	✓	✓		(✓)	(✓)	(✓)	(✓)	(✓)	(✓)		(✓)	(✓)	(✓)
MFPL			✓	✓											(✓)									
PFD			(✓)			✓									(✓)		(✓)							

languages, formalisms, and properties in Table 1. We only include the property types most used in the QComp benchmarks; P, P_r, and P_t refer to unbounded, reward-bounded, and time-bounded reachability probabilities, respectively; E indicates expected accumulated rewards, and S steady-state probabilities. A (✓) entry signifies limited support as described in the tool-specific sections below.

4.1 Model Checkers

QComp 2019 included four general-purpose probabilistic model checkers that handle a variety of formalisms and property types as well as the more specialised PRISM-TUMheuristics tool focused on unbounded probabilistic properties.

ePMC (formerly iscasMC [40]) is mainly written in Java, with some performance-critical parts in C. It runs on 64-bit Linux, Mac OS, and Windows. ePMC particularly 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 [24]. In this way, ePMC can easily be extended for special purposes or experiments without affecting the stability of other parts. It supports the PRISM language and Jani as input, DTMC, CTMC, MDP, and stochastic games as formalisms, and PCTL* and reward-based properties. ePMC particularly targets the analysis of complex linear time properties [39] and the efficient analysis of stochastic parity games [41]. It has been extended to support multi-objective model checking [37] and bisimulation minimisation [38] for interval MDP. It also has experimental support for parametric Markov models [31,60]. Specialised branches of ePMC can model check quantum Markov chains [27] and epistemic properties of multi-agent systems [30]. The tool is available in source code form at github.com/liyi-david/ePMC.

mcsta is the explicit-state model checker of the Modest Toolset [42]. It is implemented in C# and works on Windows as well as on Linux and Mac OS via the Mono runtime. Built on common infrastructure in the Modest Toolset, it supports Modest, xSADF [44] and Jani as input languages, and has access to a fast state space exploration engine that compiles models to bytecode. mcsta computes unbounded and reward-bounded reachability probabilities and expected accumulated rewards on MDP and MA, and additionally time-bounded probabilities on MA. By default, it uses value iteration and Unif+ [16]; for probabilistic reachability, it can use interval iteration [33] instead. mcsta supports PTA via digital clocks [58] and STA via a safe overapproximation [35]. It can analyse DTMC and CTMC, but treats them as (special cases of) MDP and MA, respectively, and thus cannot achieve the performance of dedicated algorithms. To deal with very large models, mcsta provides two methods to efficiently use secondary storage: by default, it makes extensive use of memory-mapped files; alternatively, given a model-specific partitioning formula, it can do a partitioned analysis [43]. For reward-bounded properties with large bounds (including time bounds in PTA), mcsta implements two unfolding-free techniques based on modified value iteration and state elimination [34]. The Modest Toolset, including mcsta, is available as a cross-platform binary package at modestchecker.net. mcsta is a command-line tool; when invoked with `-?`, it prints a list of all parameters with brief explanations. The download includes example Modest models with mcsta command lines. Modest is documented in [36] and on the toolset's website.

PRISM [57] is a probabilistic model checker for DTMC, CTMC, MDP, PTA, and variants annotated with rewards. Models are by default specified in the PRISM language, but other formats, notably PEPA [50], SBML (see sbml.org), and sparse matrix files, can be imported. Properties are specified in a language based on temporal logic which subsumes PCTL, CSL, LTL, and PCTL*; it also includes extensions for rewards, multi-objective specifications, and strategy synthesis. PRISM incorporates a wide selection of analysis techniques. Many are iterative numerical methods such as Gauss-Seidel, value iteration, interval iteration [33], and uniformisation, with multiple variants. Others include linear programming, graph-based algorithms, quantitative abstraction refinement, and symmetry reduction. Their implementations are partly symbolic (typically using binary decision diagrams) and partly explicit (often using sparse matrices). PRISM also supports statistical and parametric model checking. It can be run from a graphical user interface (featuring a model editor, simulator, and graph plotting), the command line, or Java-based APIs. It is primarily written in Java, with some C++, and works on Linux, Mac OS, and Windows. PRISM is open source under the GPL v2.0. It has been connected to many other tools using language translators, model generators, and the HOA format [3]. The tool's website at prismmodelchecker.org provides binary downloads for all major platforms, extensive documentation, tutorials, case studies, and developer resources.

PRISM-TUMheuristics is an explicit-state model checker for DTMC, CTMC, and MDP. It is implemented in Java and works cross-platform. It uses PRISM as a library for model parsing and exploration, and hence handles models in the PRISM language, with Jani support planned. It supports probabilistic reachability, safety, propositional until, and step-bounded reachability properties on MDP and DTMC as well as unbounded reachability for CTMC. At its heart, PRISM-TUMheuristics uses the ideas of [12] to only partially explore state spaces: states which are hardly reached can be omitted from computation if one is only interested in an approximate solution. Sound upper and lower bounds guide the exploration and value propagation, focusing the computation on relevant parts of the state space. Depending on the model's structure, this can yield significant speed-ups. The tool and its source code are available at prism.model.in.tum.de.

Storm [23] features the analysis of DTMC, CTMC, MDP, and MA. It supports PRISM and Jani models, dynamic fault trees [74], probabilistic programs [32], and stochastic Petri nets [1]. Storm analyses PCTL and CSL properties plus extensions of these logics with rewards, including time- and reward-bounded reachability, expected rewards, conditional probabilities, and steady-state rewards. It includes multi-objective model checking [45,65], parameter synthesis [22,64], and counterexample generation [21]. Storm allows for explicit-state and fully symbolic (binary decision diagram-based) model checking as well as mixtures of these approaches. It implements many analysis techniques, e.g. bisimulation minimisation, sound value iteration [66], Unif+ [16], learning-based exploration [12], and game-based abstraction [56]. Dedicated libraries like Eigen, Gurobi, and Z3 [62] are used to carry out sophisticated solving tasks. A command-line interface, a C++ API, and a Python API provide flexible access to the tool's features. Storm and its documentation (including detailed installation instructions) are available at stormchecker.org. It can be compiled from source (Linux and Mac OS), installed via Homebrew (Mac OS), or used from a Docker container (all platforms).

4.2 Statistical Model Checkers

Two simulation-based tools participated in QComp 2019: the DFTRES rare event simulator for fault trees, and the general-purpose statistical model checker modes.

DFTRES is the *dynamic fault tree rare event simulator* [69]: a statistical model checker for dynamic fault trees that uses importance sampling with the Path-ZVA algorithm [67]. It is implemented in Java and works cross-platform. It supports the Galileo format [72] by using DFTCalc [2] as a converter, and a subset of Jani for CTMC and MA provided any nondeterminism is spurious. Path-ZVA allows for efficient analysis of rare event models while requiring only a modest amount of memory. This algorithm is optimised for steady-state properties, but also supports probabilistic reachability (currently implemented for time-bounded properties). Simulations run in parallel on all available processor cores, resulting in a near-linear speedup on multi-core systems. DFTRES is a command-line tool; its source code is available at github.com/utwente-fmt/DFTRES, with instructions provided in a `README` file. Galileo format support requires the installation of DFTCalc, available at fmt.ewi.utwente.nl/tools/dftcalc, and its dependencies.

modes [14] is the Modest Toolset's statistical model checker. It shares the input languages, supported property types, fast state space exploration, cross-platform support, and documentation with mcsta. modes supports *all* formalisms that can be specified in JANI. It implements methods that address SMC's limitation to purely stochastic models and the rare event problem. On nondeterministic models, modes provides lower (upper) bounds for maximum (minimum) reachability probabilities via lightweight scheduler sampling [20]. For rare events, it implements automated importance splitting methods [13]. Simulation is easy to parallelise, and modes achieves near-linear speedup on multi-core systems and networked computer clusters. It offers multiple statistical methods including confidence intervals, the Okamoto bound [63], and the SPRT [75]. Unless overridden by the user, it automatically selects the best method per property.

4.3 Probabilistic Planners

The probabilistic planners that participated in QComp 2019 consider the analysis of maximum reachability in MDP specifically. They both incorporate FRET-π, but differ in the MDP heuristic search algorithm and the heuristic used.

Modest FRET-π LRTDP implements FRET-π with LRTDP to solve maximum probabilistic reachability on MDP. It is implemented within the Modest Toolset and motivated by an earlier performance comparison between planning algorithms usable for model checking purposes [53]. LRTDP [11] is an asynchronous heuristic search dynamic programming optimisation of value iteration that does not have to consider the entire state space and that converges faster than value iteration because not all values need to be converged (or even updated) before terminating. The tool supports the same input languages as mcsta and modes, and runs on the same platforms. Modest FRET-π LRTDP is available as a binary download at dgit.cs.uni-saarland.de that includes a detailed `README` file. When invoked on the command line with parameter `-help`, it prints a list of all command-line parameters with brief explanations.

Probabilistic Fast Downward [71] is an extension of the classical heuristic planner Fast Downward [48]. It supports expected accumulated rewards and maximum probabilistic reachability on MDP specified in PPDDL [77]. Limited JANI support is provided by a translation to PPDDL [53]. Probabilistic Fast Downward features a wide range of algorithms, including two variants of FRET [54,71] complemented by various heuristic search algorithms such as LRTDP [11], HDP [10], and other depth-first heuristic search algorithms [71]. Due to being based on Fast Downward, plenty of state-of-the-art classical planning heuristics are readily available. To make them usable for MDP, Probabilistic Fast Downward supports different methods to determinise probabilistic actions, notably the all-outcomes determinisation [76]. The code is a mixture of C++ and Python, and should compile and run on all common systems. The tool version that participated in QComp 2019 has some functionality removed but also adds performance enhancements. Both versions can be downloaded at fai.cs.uni-saarland.de, and include `README` files detailing how to build and run the tool. The configuration used for QComp 2019 was FRET-π with HDP [10] search and the h^1-heuristic [47] via the all-outcomes determinisation to obtain an underapproximation of the states that cannot reach the goal with positive probability.

5 Versatility and Usability Evaluation

Once a tool achieves a base level of performance, its versatility and usability may arguably become more important to its acceptance among domain experts than its performance. As versatility, we consider the support for modelling languages and formalisms, for different and complementary analysis engines, and configurability (e.g. to make runtime–precision tradeoffs). Usability is determined by the tool's documentation, the availability of a graphical interface, its installation process, supported platforms, and similar aspects. A user-friendly tool achieves consistently good performance with few non-default configuration settings.

Versatility. The five general-purpose tools—ePMC, mcsta, modes, PRISM, and Storm—support a range of modelling languages, formalisms, and properties (cf. Table 1 and Sect. 4). In terms of languages, Storm is clearly the most versatile tool. Those based on the Modest Toolset and ePMC connect to many languages via JANI. mcsta and modes implement analysis methods for *all* of the formalisms supported by JANI (cf. Fig. 1) while Storm still covers all of those considered in QComp. PRISM only lacks support for MA. However, on the formalisms that they support, PRISM and Storm implement the widest range of properties, followed by ePMC. These three tools in particular support many properties not considered in QComp 2019 such as LTL, PCTL*, multi-objective queries, and parametric model checking. PRISM and Storm also implement many algorithms for the user to choose from that provide different tradeoffs and performance characteristics; Probabilistic Fast Downward is similar in this regard when it comes to planning algorithms and heuristics. While modes is limited to deterministic MDP, MA and PTA when exact results are required as in QComp, it *can* tackle the nondeterminism via lightweight scheduler sampling to provide bounds.

Usability. The most usable among all tools is clearly PRISM: it provides extensive online documentation, a graphical user interface, and binary downloads for all platforms that only depend on Java. The Modest Toolset is less documented and contains command-line tools only, but again ships cross-platform binaries that only require the Mono runtime on non-Windows systems. All in all, the tools based on the Modest Toolset and those mainly implemented in Java (ePMC, DFTRES, PRISM, and PRISM-TUMheuristics) provide the widest platform support. Storm is notably not available for Windows, and Fast Downward partly works cross-platform but is only supported for Linux. The default way to install Storm, and the only way to install DFTRES, ePMC, PRISM-TUMheuristics, and Probabilistic Fast Downward, is to compile from source code. Storm in particular requires a large number of dependencies in a long build process, which however is well-documented on its website. All tools come with a default analysis configuration adequate for QComp except for Probabilistic Fast Downward, which requires the explicit selection of a specific engine and heuristics. The performance evaluation results in Sect. 6.2 highlight that PRISM and Storm can benefit significantly from using non-default configuration settings tuned by experts to the individual benchmarks, with mcsta showing moderate improvements with simpler tuning.

6 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. Prior to the performance evaluation, all participants submitted a *wishlist* of (challenging) instances, from which the organisers chose a final set of 100 for the competition: 18 DTMC, 18 CTMC, 36 MDP, 20 MA and 8 PTA instances covering 40 unbounded and 22 bounded probababilistic reachability, 32 expected-reward, and 6 steady-state properties. The selection favoured models selected by multiple participants while aiming for a good balance in terms of formalisms, modelling languages, and property types. As a baseline, every tool should have a good number of supported instances included; still, some tools that were particularly restricted in terms of languages and property types (such as DFTRES and Probabilistic Fast Downward) could only check up to 10 of them. By taking *every* participant's wishlist into account, QComp naturally included instances that a certain tool would do well on (suggested by the participant who submitted the tool) as well as instances that it was not expected to perform best with (suggested by the authors of other tools).

After finalisation of the benchmark instances, participants submitted *tool packages*: installation instructions for the tool (or the tool itself) and a script to generate a JSON file (or the file itself) containing, for every instance, up to two command lines to invoke the tool. One of them was required to run the tool in its default configuration, while the other could use instance-specific parameters to tweak the tool for maximum performance. The performance evaluation was then done by the organisers on one central computer: a standard desktop machine with an Intel Core i7-920 CPU and 12 GB of RAM running 64-bit Ubuntu Linux 18.04. Tools were given 30 minutes per instance. The choice for a rather modest machine was intentional: the slower CPU increased the performance differentiation for moderately-challenging instances, and the moderate amount of memory allowed for some evaluation of memory efficiency by observing the number of out-of-memory results. In particular, a tool's actual memory usage is not a good measure of quality since the ideal tool will make use of all available memory to speed up the verification as much as possible on challenging instances.

6.1 The Precision Challenge

Almost all properties queried for a value, with only few asking whether a probability is equal to 1. Participants were required to submit a script that extracts the value of an instance's property from the tool output. Since quantitative verification tools can often trade precision for performance, QComp required a tool's result r_i for instance i to be within $[0.999 \cdot v_i, 1.001 \cdot v_i]$ with v_i being the instance's property's correct result—i.e. we required a relative error of at most 10^{-3}. We chose this value as a tradeoff between the advantages of model checkers (which easily achieve high precision but quickly run out of memory on large state spaces) and simulation-based tools (which easily handle large state spaces but quickly run out of time when a high precision is required).

Reference results. Unfortunately, the actual result for a property is difficult to obtain: tools that scale to large models use inexact floating-point arithmetic, and any tool result may be affected by tool bugs. At the same time, it does not make sense to report performance data when a tool provides an incorrect result as this may be due to an error that drastically reduces or increases the analysis time. QComp 2019 adopted the following pragmatic approach: the organisers used the "most trustworthy" analysis approach available (usually an exact-arithmetic solver for small and a model checker using a sound iterative numerical method for large models) to produce reference results for all selected instances. Participants were then invited to use any other tool to try and refute the correctness of those results, and would discuss the result or benchmark in case of refutation. In the end, only one of the reference results was shown to be incorrect, and this was due to a model translation error that could be corrected before the competition.

Sound and unsound model checking. Practical quantitative model checkers typically use iterative numerical algorithms relying on floating-point arithmetic. Here, certain algorithms can ensure error bounds (such as interval iteration [6,12,33] and sound value iteration [66] for probabilistic reachability, and uniformisation for time-bounded reachability in CTMC). The most common approaches, e.g. value iteration for probabilistic reachability with the standard termination criterion, however provide "good enough" results for many models encountered in practice but may also be widely off for others. It is clearly unfair to compare the runtimes of tools that provide proper precision guarantees against tools without such guarantees where the result happens to be just close enough to the reference value, perhaps even after heavy parameter tweaking to find the sweet spot between runtime and precision. For QComp 2019, since it is the first of its kind and a friendly event, participants agreed to avoid such parameter tweaking. In particular, for iterative methods with an "unsound" convergence check, all participants agreed on using a relative error threshold of $\epsilon = 10^{-6}$ for checking convergence.

6.2 Performance Results

The QComp 2019 performance evaluation produced a large amount of data, which is available at qcomp.org; we here summarise the outcomes in comparative plots. In all of them, we use a logarithmic scale for runtime.

Configurations. mcsta, modes, PRISM and Storm provided instance-specific tool parameters that significantly changed their performance characteristics. All three model checkers switched to an exact-arithmetic or sound iterative method for models with known numerical issues (i.e. the *haddad-monmege* model). Other than that, mcsta was run with some runtime checks disabled (as was modes), and its disk-based methods were disabled for models with relatively small state spaces. On PTA, it was configured to compress linear chains of states, and to use state elimination for time-bounded properties. PRISM was configured to use the best-performing of its four main analysis engines for every instance. This typically meant switching from the default "hybrid" engine to "sparse" for added

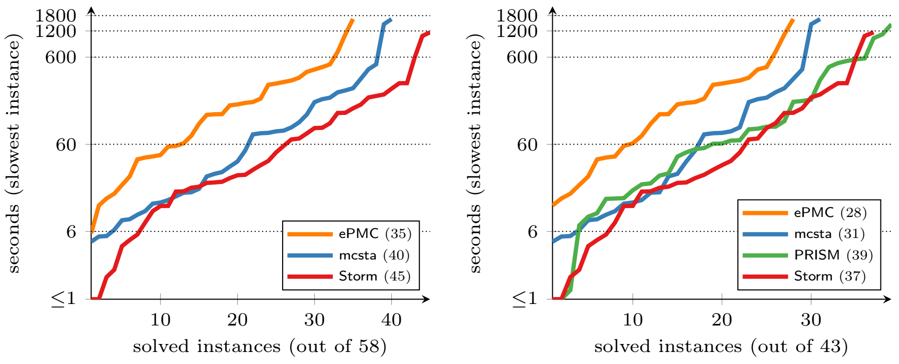


Fig. 2. Quantile plots for the general-purpose model checkers (default configuration)

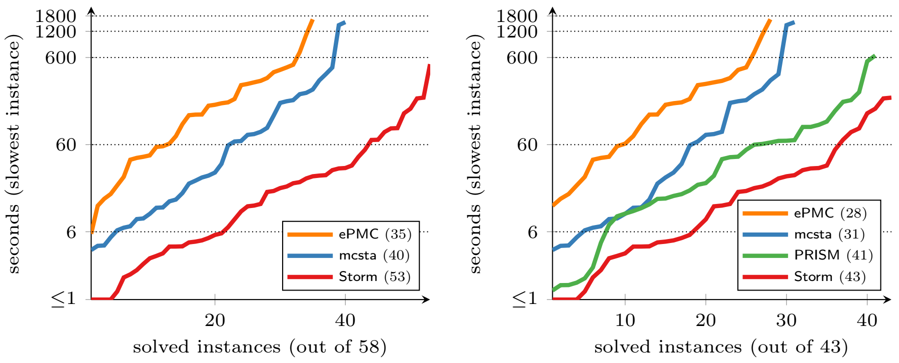


Fig. 3. Quantile plots for the general-purpose model checkers (specific configurations)

speed when the state space does not result in memory issues, and to "mtbdd" for larger models with regularity. A Gauss-Seidel variant of each analysis method was used for acyclic models. Storm's specific configurations were set in a similar way to use the fastest out of its four main engines ("sparse", "hybrid", "dd", and "dd" with symbolic bisimulation minimisation) for every instance. Observe that the specific configurations of PRISM and Storm could only be obtained by testing all available engines a priori, which cannot be expected from normal users.

modes by default rejects models with nondeterminism, and runs until the required error is met with 95 % confidence, often hitting the 30-minute timeout. In the specific configurations, modes was instructed to resolve nondeterminism ad hoc, and to return the current estimate irrespective of statistical error after 28 minutes. It can thus solve more instances (where the nondeterminism is spurious, and where the statistical method is too strict), but risks returning incorrect results (when nondeterminism is relevant, or the error is too large).

Quantile plots. We first compare the performance of the general-purpose model checkers by means of *quantile plots* in Figs. 2 and 3. Each plot only considers the instances that are supported by *all* of the tools shown in the plot; this is to avoid unsupported instances having a similar visual effect to timeouts and errors. 58 instances are supported by all three of ePMC, mcsta and Storm, while still 43 instances (those in the PRISM language) are also supported by PRISM. The plots' legends indicate the number of correctly solved benchmarks for each tool (i.e. where no timeouts or error occurred and the result was relatively correct up to 10^{-3}). 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 solved by the tool was y seconds.

We see that PRISM and Storm are the fastest tools for *most* of the common instances in the default configuration, closely followed by mcsta. The performance of PRISM and Storm improves significantly by selecting instance-specific analysis engines, with Storm taking a clear lead. PRISM solves the largest number of instances in default configuration while Storm leads in specific configurations.

Scatter plots. In Figs. 4 to 6, we show scatter plots for all tools that compare their performance over all individual instances to the best-performing other tool for each instance. These plots provide more detailed information compared to the previous quantile plots since they compare the performance on individual instances. 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 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 vertical "TO", "ERR" and "INC" lines respectively indicate instances where the plot's tool encountered a timeout, reported an error (such as nondeterminism not being supported or a crash due to running out of memory), or returned an incorrect result (w.r.t. the relative 10^{-3} precision). Points on the horizontal "n/a" line indicate instances that none of the other tools was able to solve. The "default" plots used the default configuration for all tools, while the "specific" plots used the specific per-instance configurations for *all* tools. We do not show plots for the specific configurations of the four specialised tools since they are not significantly different.

Overall, we see that every tool is the fastest for some instances. PRISM (default), Storm (specific) and modes in particular can solve several models that no other tool can. The specialised and simulation-based tools may not win in terms of overall performance (except for Probabilistic Fast Downward, on the few instances that it supports), but they all solve certain instances uniquely—which is precisely the purpose of a specialised tool, after all. The selected instances contain a few where unsound model checkers are expected to produce incorrect results, in particular the *haddad-monmege* model from [33]; we see this clearly in the plots for ePMC, mcsta and Storm. PRISM aborts with an error when a numeric method does not "converge" within 10000 iterations, which is why such instances appear on the "ERR" line for PRISM. ePMC and mcsta do not yet implement exact or sound iterative methods, which is why they keep incorrect results in the specific configurations. The difference between default and specific configurations for modes is different, as explained; it shows that several instances

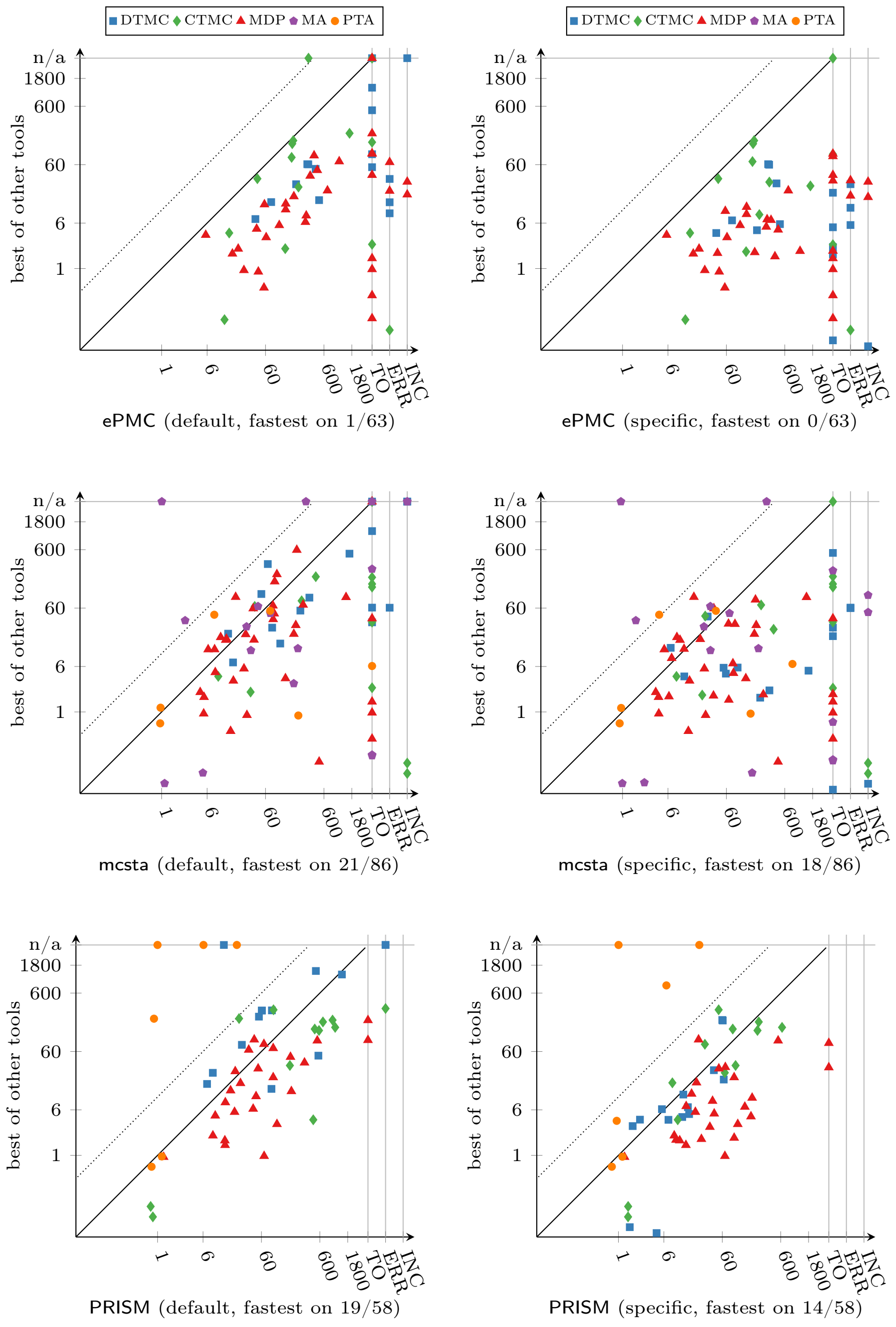


Fig. 4. Runtime of specific tools compared with the best results (1/3)

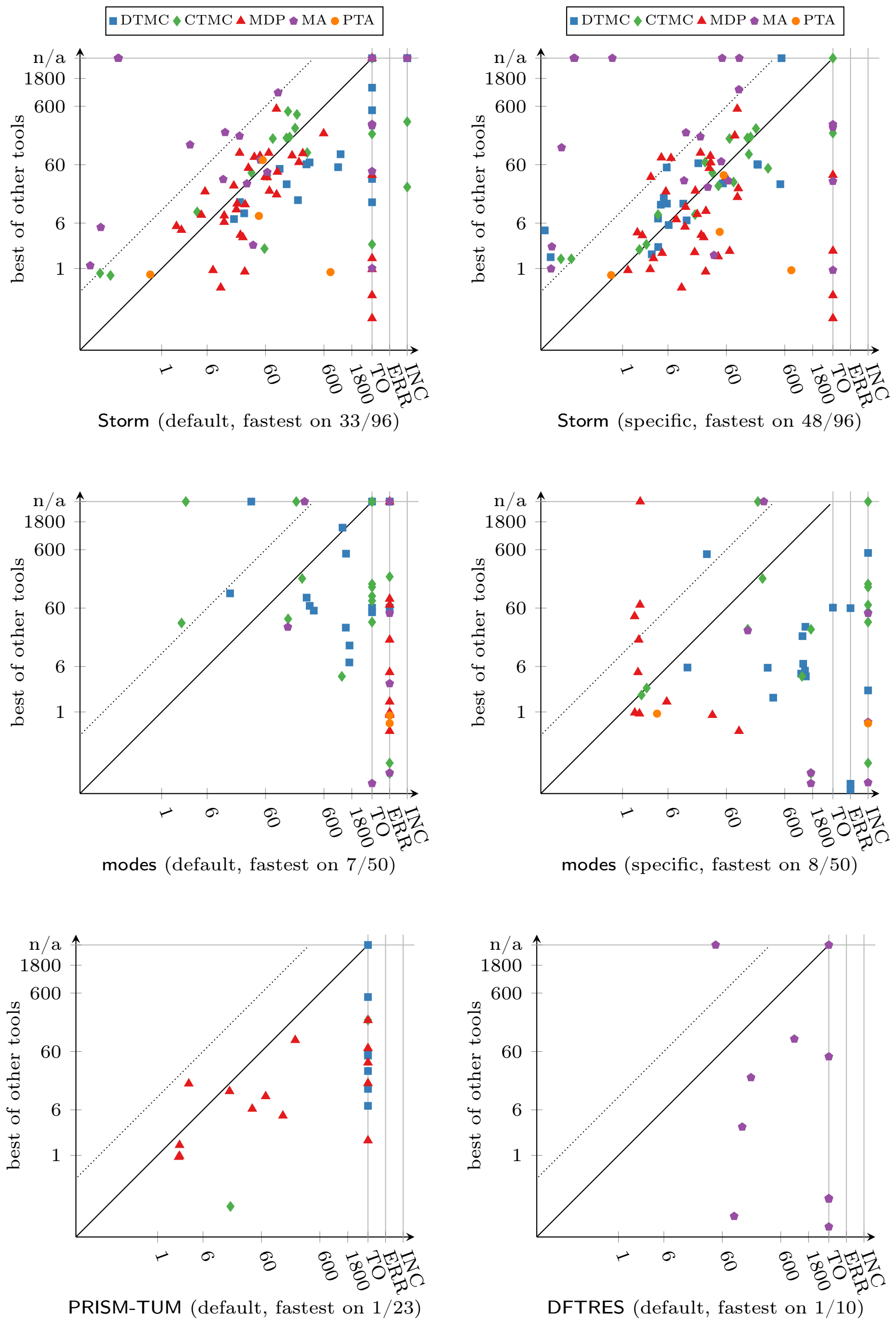
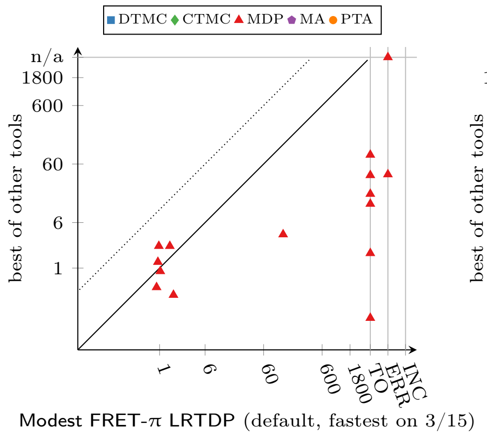


Fig. 5. Runtime of specific tools compared with the best results (2/3)



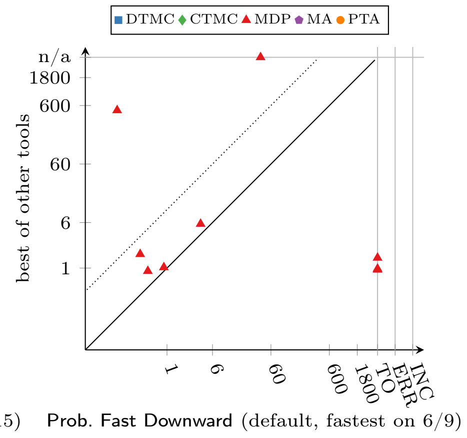


Fig. 6. Runtime of specific tools compared with the best results (3/3)

are spuriously nondeterministic, and several results are good enough at a higher statistical error, but many instances also turn from errors to incorrect results.

7 Conclusion and Outlook

QComp 2019 achieved its goal of assembling a community of tool authors, motivating the collection of a standardised benchmark set in the form of the QVBS, and sparking discussions about properly comparing quantitative verifiers. It also improved JANI tool support and resulted in a set of reusable scripts for batch benchmarking and plotting. Throughout this process, some lessons for changes and requests for additions to the next instance of QComp surfaced:

- The issue that caused most discussion was the problem of how to treat tools that use "unsound" methods as explained in Sect. 6.1. In the future, we plan to provide several tracks, e.g. one where exact results up to some precision are required without per-instance tweaking of parameters, and one that allows fast but "imprecise" results with a nuanced penalty depending on the error.
- The evaluation of default and specific configurations provided important insights, but might not be continued; we expect tools to use the QComp 2019 results as a push to implement heuristics to choose good defaults automatically.
- The current versatility and usability evaluation was very informal and needs to move to clear pre-announced criteria that tool authors can plan for.
- The only addition to formalisms requested by participants is stochastic games, e.g. as in PRISM-games [55]; however, these first need standardisation and JANI support. In terms of properties, LTL is supported by several tools and will be included in the next edition of QComp. Other desirable properties include multi-objective queries, and the generation of strategies instead of just values.

– Finally, all benchmarks of QComp 2019 were known a priori. As QComp slowly transitions from a "friendly" to a more "competitive" event, the inclusion of obfuscated or a priori unknown benchmarks needs to be considered.

Acknowledgements. QComp 2019 was organised by Arnd Hartmanns and Tim Quatmann. The authors thank their tool co-developers: Yi Li (Peking University), Yong Li (Chinese Academy of Sciences), Andrea Turrini, and Lijun Zhang (Chinese Academy of Sciences and Institute of Intelligent Software) for ePMC; Pranav Ashok, Tobias Meggendorfer, and Maximilian Weininger (Technische Universität München) for PRISM-TUMheuristics; and Sebastian Junges and Matthias Volk (RWTH Aachen) for Storm.

Data availability. The tools used and data generated in the performance evaluation are archived and available at qcomp.org/competition/2019.

References

1. 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)
2. Arnold, F., Belinfante, A., van der Berg, F., Guck, D., Stoelinga, M.I.A.: DFTCalc: a tool for efficient fault tree analysis. In: SAFECOMP. LNCS, vol. 8153, pp. 293–301. Springer (2013)
3. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Kretínský, J., Müller, D., Parker, D., Strejcek, J.: The Hanoi omega-automata format. In: CAV. LNCS, vol. 9206, pp. 479–486. Springer (2015)
4. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
5. 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)
6. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In: CAV. LNCS, vol. 10426, pp. 160–180. Springer (2017)
7. 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)
8. Beyer, D.: Competition on software verification (SV-COMP). In: TACAS. LNCS, vol. 7214, pp. 504–524. Springer (2012)
9. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MODEST: A compositional modeling formalism for hard and softly timed systems. IEEE Trans. Software Eng. 32(10), 812–830 (2006)
10. Bonet, B., Geffner, H.: Faster heuristic search algorithms for planning with uncertainty and full feedback. In: IJCAI. pp. 1233–1238. Morgan Kaufmann (2003)
11. Bonet, B., Geffner, H.: Labeled RTDP: Improving the convergence of real-time dynamic programming. In: ICAPS. pp. 12–21. AAAI (2003)
12. 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)

13. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Better automated importance splitting for transient rare events. In: SETTA. LNCS, vol. 10606. Springer (2017)
14. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: A statistical model checker for nondeterminism and rare events. In: TACAS. LNCS, vol. 10806, pp. 340–358. Springer (2018)
15. 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. Springer (2017)
16. Butkova, Y., Hatefi, H., Hermanns, H., Krcál, J.: Optimal continuous time Markov decisions. In: ATVA. LNCS, vol. 9364, pp. 166–182. Springer (2015)
17. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. FMSD 43(1), 61–92 (2013)
18. Coles, A.J., Coles, A., Olaya, A.G., Celorrio, S.J., Linares López, C., Sanner, S., Yoon, S.: A survey of the seventh international planning competition. AI Magazine 33(1) (2012)
19. 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)
20. 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)
21. 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)
22. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J., Ábrahám, E.: PROPhESY: A probabilistic parameter synthesis tool. In: CAV. LNCS, vol. 9206, pp. 214–231. Springer (2015)
23. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: CAV. LNCS, vol. 10427. Springer (2017)
24. 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)
25. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS. pp. 342–351. IEEE Computer Society (2010)
26. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. LMCS 4(4) (2008)
27. 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 fuer Informatik (2017)
28. Fränzle, M., Hahn, E.M., Hermanns, H., Wolovick, N., Zhang, L.: Measurability and safety verification for stochastic hybrid systems. In: HSCC. ACM (2011)
29. Frehse, G., Althoff, M., Bogomolov, S., Johnson, T.T. (eds.): ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems, EPiC Series in Computing, vol. 54. EasyChair (2018)
30. Fu, C., Turrini, A., Huang, X., Song, L., Feng, Y., Zhang, L.: Model checking probabilistic epistemic logic for probabilistic multiagent systems. In: IJCAI (2018)
31. Gainer, P., Hahn, E.M., Schewe, S.: Accelerated model checking of parametric Markov chains. In: ATVA. LNCS, vol. 11138, pp. 300–316. Springer (2018)
32. Gordon, A.D., Henzinger, T.A., Nori, A.V., Rajamani, S.K.: Probabilistic programming. In: FOSE. pp. 167–181. ACM (2014)
33. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. 735, 111–131 (2018)

34. Hahn, E.M., Hartmanns, A.: A comparison of time- and reward-bounded probabilistic model checking techniques. In: SETTA. LNCS, vol. 9984. Springer (2016)
35. Hahn, E.M., Hartmanns, A., Hermanns, H.: Reachability and reward checking for stochastic timed automata. Electronic Communications of the EASST 70 (2014)
36. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. FMSD 43(2), 191–232 (2013)
37. Hahn, E.M., Hashemi, V., Hermanns, H., Lahijanian, M., Turrini, A.: Multi-objective robust strategy synthesis for interval Markov decision processes. In: QEST. LNCS, vol. 10503, pp. 207–223. Springer (2017)
38. 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)
39. Hahn, E.M., Li, G., Schewe, S., Zhang, L.: Lazy determinisation for quantitative model checking. CoRR abs/1311.2928 (2013)
40. 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)
41. Hahn, E.M., Schewe, S., Turrini, A., Zhang, L.: A simple algorithm for solving qualitative probabilistic parity games. In: Chaudhuri, S., Farzan, A. (eds.) CAV. LNCS, vol. 9780, pp. 291–311. Springer (2016)
42. 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)
43. 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)
44. Hartmanns, A., Hermanns, H., Bungert, M.: Flexible support for time and costs in scenario-aware dataflow. In: EMSOFT. pp. 3:1–3:10. ACM (2016)
45. 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)
46. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS. LNCS, vol. 11427. Springer (2019)
47. Haslum, P., Bonet, B., Geffner, H.: New admissible heuristics for domain-independent planning. In: AAAI/IAAI. pp. 1163–1168. AAAI/MIT Press (2005)
48. Helmert, M.: The Fast Downward planning system. J. Artif. Intell. Res. 26, 191–246 (2006)
49. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI. LNCS, vol. 2937, pp. 73–84. Springer (2004)
50. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
51. Järvisalo, M., Berre, D.L., Roussel, O., Simon, L.: The international SAT solver competitions. AI Magazine 33(1) (2012)
52. Jasper, M., Mues, M., Schlüter, M., Steffen, B., Howar, F.: RERS 2018: CTL, LTL, and reachability. In: ISoLA. LNCS, vol. 11245, pp. 433–447. Springer (2018)
53. Klauck, M., Steinmetz, M., Hoffmann, J., Hermanns, H.: Compiling probabilistic model checking into probabilistic planning. In: ICAPS. pp. 150–154. AAAI (2018)
54. Kolobov, A., Mausam, Weld, D.S., Geffner, H.: Heuristic search for generalized stochastic shortest path MDPs. In: ICAPS. AAAI (2011)
55. Kwiatkowska, M., Parker, D., Wiltsche, C.: PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. STTT 20(2), 195–210 (2018)

56. Kwiatkowska, M.Z., Norman, G., Parker, D.: Game-based abstraction for Markov decision processes. In: QEST. pp. 157–166. IEEE Computer Society (2006)
57. 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)
58. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. FMSD 29(1), 33–78 (2006)
59. 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)
60. Li, Y., Liu, W., Turrini, A., Hahn, E.M., Zhang, L.: An efficient synthesis alg. for param. Markov chains against linear time properties. CoRR abs/1605.04400 (2016)
61. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: FORMATS/FTRTFT. LNCS, vol. 3253, pp. 152–166. Springer (2004)
62. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: TACAS. LNCS, vol. 4963, pp. 337–340. Springer (2008)
63. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Annals of the Institute of Statistical Mathematics 10(1), 29–35 (1959)
64. Quatmann, T., Dehnert, C., Jansen, N., Junges, S., Katoen, J.P.: Parameter synthesis for Markov models: Faster than ever. In: ATVA. LNCS, vol. 9938, pp. 50–67 (2016)
65. Quatmann, T., Junges, S., Katoen, J.P.: Markov automata with multiple objectives. In: CAV. LNCS, vol. 10426, pp. 140–159. Springer (2017)
66. Quatmann, T., Katoen, J.P.: Sound value iteration. In: CAV. LNCS, vol. 10981, pp. 643–661. Springer (2018)
67. Reijsbergen, D., de Boer, P.T., Scheinhardt, W., Juneja, S.: Path-ZVA: General, efficient, and automated importance sampling for highly reliable Markovian systems. TOMACS 28(3), 22:1–22:25 (2018)
68. Rubino, G., Tuffin, B.: Rare Event Simulation Using Monte Carlo Methods. Wiley (2009)
69. Ruijters, E., Reijsbergen, D., de Boer, P.T., Stoelinga, M.I.A.: Rare event simulation for dynamic fault trees. Reliability Engineering & System Safety To appear.
70. Sproston, J.: Decidable model checking of probabilistic hybrid automata. In: FTRTFT. LNCS, vol. 1926, pp. 31–45. Springer (2000)
71. 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)
72. Sullivan, K.J., Dugan, J.B., Coppit, D.: The Galileo fault tree analysis tool. In: FTCS-29. pp. 232–235. IEEE Computer Society (1999)
73. Ummels, M., Baier, C.: Computing quantiles in Markov reward models. In: FOSSACS. LNCS, vol. 7794, pp. 353–368. Springer (2013)
74. Volk, M., Junges, S., Katoen, J.P.: Fast dynamic fault tree analysis by model checking techniques. IEEE Trans. Industrial Informatics 14(1), 370–379 (2018)
75. Wald, A.: Sequential tests of statistical hypotheses. The Annals of Mathematical Statistics 16(2), 117–186 (1945)
76. Yoon, S.W., Fern, A., Givan, R.: FF-Replan: A baseline for probabilistic planning. In: ICAPS. p. 352. AAAI (2007)
77. Younes, H.L.S., Littman, M.L., Weissman, D., Asmuth, J.: The first probabilistic track of the Int. Planning Competition. J. Artif. Intell. Res. 24, 851–887 (2005)
78. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: CAV. LNCS, vol. 2404, pp. 223–235. Springer (2002)