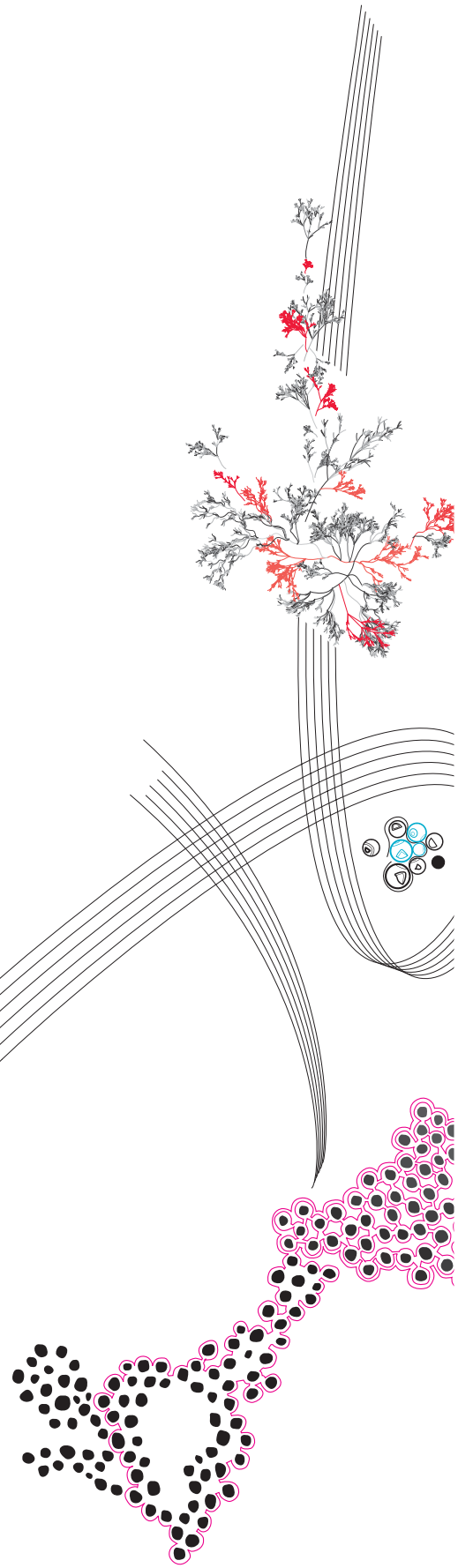


MSc Thesis Computer Science

On the Performance of Algorithms for Quantitative Verification

Ivan Jorick Hop

Supervisors: A. Hartmanns and B. Kohlen

November 2022

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Preface

This thesis marks the end of my study at the University of Twente in Computer Science. I would like to thank the employees of the University of Twente for the knowledge for making this thesis and my friends and family for their support. With that being said, I want to thank some people in particular.

First, I thank my supervisors, Arnd Hartmanns and Bram Kohlen. With Arnd, I have learned new and useful academic skills. I want to thank him for his great accessibility, allowing me to break into his office whenever needed. I thank Bram for meticulously reviewing this thesis multiple times and offering me his invaluable feedback. Secondly, I thank the members of the Formal Methods and Tools group for the supportive work environment.

Besides the academic guidance, I would like to thank friends and family. I want to specifically mention Shannon Plaul for proofreading this thesis, the members of my graduation group Wouter Bos, Daan Kooij, Jan Boerman, Bas Bleijerveld and Erik Kemp for their moral support, and my parents Edwin Hop and Dionne van der Giezen for their support throughout my whole studies. To all of you, thank you.

Abstract

Quantitative verification is used to verify quantitative properties of a system. The expected response time and the probability of failure are examples of properties that we can verify. Various algorithms can calculate these properties. This thesis contains an in-depth analysis into the performance of these algorithms. We identified 3 groups of algorithms that have similar performance. These groups roughly match the technique used to store the state space. The groups used the techniques explicit and symbolic (BDD-based) as well as statistical model checking, which explores the state space on-the-fly. The paper contains an overview of the performance of algorithms against each other. We observed that algorithms that were faster on average, had fewer best times. We investigated correlations between the performance of algorithms and the number of states, transitions and branches in the model of the system. We observed that most explicit algorithms have a linear correlation to these characteristics. The other algorithms did not have a statistically significant correlation.

Table of Contents

1 Introduction

We are building increasingly more complex systems, for example, airplane communication. It is essential that safety-critical systems are working correctly. Various methods have been developed to verify whether these systems are working correctly; however, not everything can be categorised as correct or incorrect. For example, driving a car. There is an inherent risk in driving a car. If we want to know this risk, we can find out through quantitative verification.

The risk is calculated using a mathematical model of the system. An expert makes a model of the system and writes the requirements more formally, which is a *property*. An example property for the car is "Do injuries happen in fewer than 5% of the accidents?". Formal models used for quantitative verification are discrete-time Markov chain (DTMC) [31], Markov decision process (MDP) [31], continuous-time Markov chain (CTMC) [31], Markov automata (MA) [22, 23] and probabilistic timed automata (PTA) [40]. The properties are written in formalisms such as Continuous Stochastic Logic (CSL) [6], Probabilistic real-time Computation Tree Logic (PCTL) [28] and Probabilistic Reward Computation Tree Logic (PRCTL) [4].

The property is verified using an algorithm for quantitative verification. The algorithm calculates the risk, which we refer to this as the *value*. This is, in our car example, "The chance that an injury happens in an accident". The *result value* is what the algorithm calculated. Based on the result value, the algorithm outputs a result that is either valid, invalid or not computed. Assuming the algorithm is finished, the result is valid if the model adheres to the property, otherwise, the result is invalid. The result is not computed when the algorithm runs out of memory or it takes too long. There are various ways to deal with a not computed result. These can be building a smaller model or using a different algorithm.

Various algorithms have been implemented [5, 12, 18, 19, 33, 39]. They differ in the precision of the result and the approach to verify the property. Calculating more precise results often requires more computations and memory, which results in a slower algorithm.

The three steps to calculate the result value are: parsing the input file, building the state space and calculating the result value. The algorithms differ in how they build the state space and calculate the result value. The state space can be built in four ways. These are: using sparse matrices, BDDs', partial state space [36] and on-the-fly methods. There are also various ways to calculate the value. Some examples are value iteration [47], topological value iteration [19] and policy iteration [9].

Deciding which algorithm to use is a challenging task. The algorithm must be fast and should not run out of memory. There is some research available. The quantitative verification competition (QComp) [16] is a competition among tools for quantitative verification. A tool contains a collection of algorithms. They measured the performance of tools on a benchmark. A *benchmark* consists of multiple benchmark instances. A *benchmark instance* consists of a model and a property to verify. The tools were tasked to verify a benchmark instance in 15 minutes. Overall the tool Storm [35] had the best performance. However, this does not mean that one should always use this tool. The fastest tool changes depending on the benchmark instance, which we also observed in other research [33, 37, 43, 52]. To combat this, Storm has an algorithm that automatically chooses an algorithm based on the characteristics of the benchmark instance. *Characteristics* can be anything measured from a benchmark instance. Two examples are the size of the domain of variables and the number of states.

Storm automatic engine and the results of QComp inspired the following research question:

How do the characteristics of a benchmark instance influence the performance of algorithms for quantitative verification?

We perform a benchmark to answer this question. This is performed in three phases: design, measurements and analysis. In the design phase, we decide which benchmark instances, algorithms and characteristics to use and how we want to answer the research questions from the analysis phase. The measurement phase focuses on getting the measurements. We create a benchmark tool named QVAnalyser written in Python. QVAnalyser is publicly available on Github[1] with a Creative Commons attribution 4.0 licence [2]. It is based on the tool used in QComp and an internal tool

[1] https://github.com/Vescatur/QVAnalyser
[2] https://creativecommons.org/licenses/by/4.0/

from the Storm developers. During the analysis phase, we analyse the measurements. The research questions are shown below. RQ3.3 is a direct result of the main research question. RQ3.1 and RQ3.2 are results that are side findings.

QComp is similar to our research, although different in a few ways. We focus on algorithms while, QComp focuses on tools. The goals are different. QComp is a competition, while our goal is to find statistical associations. Despite these differences, the foundation of QComp is useful. We make use of the benchmark instances used in QComp. These were from the quantitative verification benchmark set (QVBS) [34]. QComp splits the results based on precision which we do not do. However, we do keep it in mind when interpreting the results.

Phase 1. Design
- **RQ1.1.** Which algorithms do we want to analyse?
- **RQ1.2.** Which benchmark instances do we want to use?
- **RQ1.3.** Which characteristics do we want to investigate?
- **RQ1.4.** Which performance metrics do we want to investigate?
- **RQ1.5.** Which techniques do we use to compare performances?
- **RQ1.6.** Which techniques do we use to detect statistical associations between the characteristics of a benchmark instance and the performance of an algorithm?

Phase 2. Measurements
- **RQ2.1.** What is the performance of the algorithms on the benchmark instances?
- **RQ2.2.** What characteristics do the benchmark instances have?

Phase 3. Analysis
- **RQ3.1.** How does the performance between algorithms compare?
- **RQ3.2.** Which benchmark instances have similar performance according to the similarity metric from SAT?
- **RQ3.3.** What statistical associations exist between the characteristics of a benchmark instance and the performance of an algorithm?

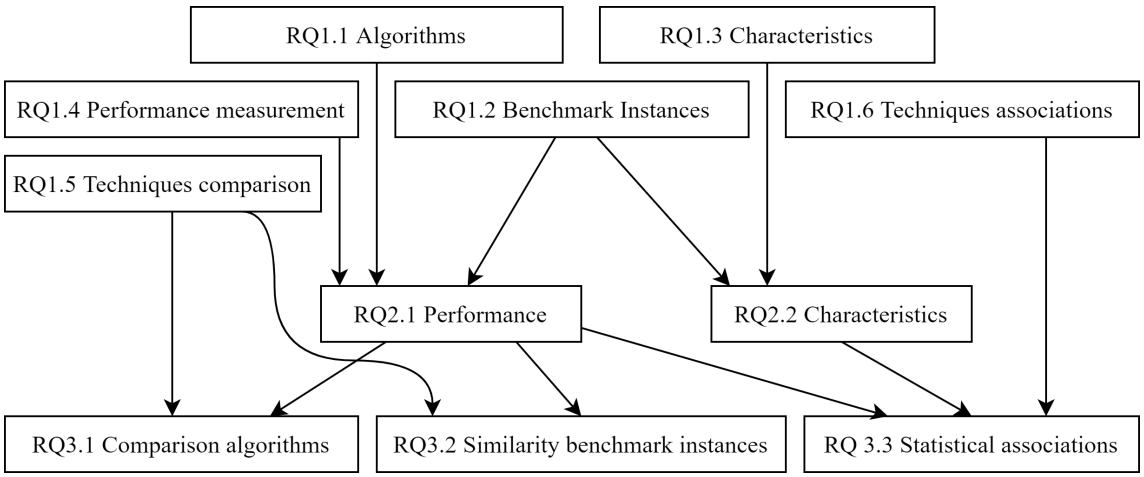


Fig. 1. Dependence of research questions

This thesis is structured as follows: starting with background knowledge for quantitative verification in Chapter 2, followed by related work about the speed of algorithms, competitions and characteristics of a benchmark instance in Chapter 3. After that, the design, measurements and analysis are discussed in Chapters 4, 5 and 6. Finally, the conclusion, limitations and recommendations for future research will be discussed in Chapter 7.

2 Background

This section provides a background for quantitative verification. First of all, we discuss model checking and the process of model verification. After that, we talk about the quantitative formal models and the JANI language. Finally, we look at the tools for quantitative verification.

2.1 Model checking

Quantitative verification is part of the broader field of model checking. Model checking is the technique of finding errors based on a model of the system. Quantitative verification is focused on probabilistic systems.

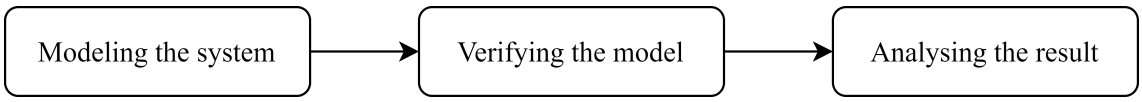


Fig. 2. Process of model checking

The process of model checking consists of three steps: modeling, verification and analysis [7, Chapter 1]. As Figure 2 shows, the first step is modeling. An expert models the systems. A part of this is writing the *properties*; these are the requirements in a formal language. Ambiguities in the design of the system are often discovered during the modeling phase. Based on these ambiguities, the expert can improve the design of the system. The second step is verification, in which a tool verifies the model. A tool is a program that contains one or more algorithms to verify models. The algorithm outputs a result that is either valid, invalid or not computed. The third step is analysis; an expert analyses the result. The result is valid if the model satisfies all properties. The result is invalid if the model violates one or more properties. The model can be invalid because either the system or the model is incorrect. The system is incorrect if it does not adhere to the requirements. The model is incorrect if its behaviour does not correspond to the behaviour of the system. An expert has to correct the model. There exist tools to produce counterexamples []. These tools help us to discover why the model violates the property. The algorithm is not computed if it runs out of memory or takes too long. An expert has to prevent this. An expert could make the model more memory efficient or use a different algorithm.

Model checking is not a silver bullet. In this section, we present the advantages and drawbacks of model checking as described by Baier et al. [7, Chapter 1]. The advantage of model checking is its sound mathematical foundation. It can find errors independent of how likely it is to occur. The same does not hold for other testing techniques. This makes model checking good for control-intensive applications.

However, model checking is not suitable for data-driven applications because the data often ranges over a huge domain, which blows up the state space. These state spaces are difficult to store, which makes it difficult to compute. Modeling moderately sized systems may require more memory or time than is available. Even though there are efficient algorithms for model checking, this limits the application to small systems or those with a lot of symmetry.

A modeling error could cause the model to verify even though the system is incorrect. This is because the model is verified instead of the system itself. Modeling errors that could occur are: the behaviour of the model does not correspond with the behaviour of the system and the requirements are incomplete or incorrect.

Given the advantages and drawbacks, model checking is good at finding design errors, given that the modeling is done carefully [7, Chapter 1].

2.2 The process of quantitative model verification

Model verification is the second step of model checking. In this step, a tool verifies the model. In this paper, we focus on quantitative model verification. Figure 3 shows the three steps of quantitative model verification. The first step is parsing the input file of the model. The second step is building

the state space. The third step is calculating the result. In this thesis, we focus on the second and third steps because parsing the input file takes a small amount of time compared to the other steps.

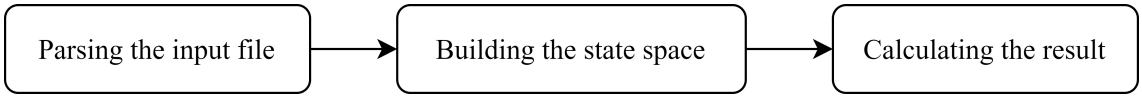


Fig. 3. Process of quantitative model verification

The second step is building the state space, which can be done in different ways. The state space can be stored *explicitly*, which means that the state space is stored in sparse matrices, which only store non-zero values. The explicit technique is often referred to as sparse matrices. Another way to store the state space is *symbolic*, which uses binary decision diagram (BDD) [35]. BDD is a data structure that stores data in a compressed format. The operations are performed on the compressed state space with decompressing. This allows BDDs' to save a lot of memory when there is a lot of structure in the state space. Finally, there are algorithms that avoid building the whole state space. *Partial state space* algorithms like general labeled real-time dynamic programming (GLRTDP) [36] work by exploring states that contribute the most to the result. The algorithm calculates the result using only the explored states. *Statistical model checking* algorithms calculate the result by simulating the model. The simulation only needs to build the states that it passes through. Partial state space and statistical model checking algorithms build the state space while calculating the result. This means that steps two and three are combined. In summary, the state space can be stored in two ways, explicitly or symbolically. Partial state space algorithms and statistical model checking algorithms save memory by avoid building the whole state space.

The third step is calculating the result. The next section discusses two types of properties for quantitative verification.

1. *Reachability probability* properties ask what the probability is of reaching a goal. An example property is "What is the probability that the coin falls on head?"
2. *Expected reward* properties ask what is the expected reward before reaching a goal. An example property is "What is the expected amount of times to flip the coin before it falls on head?"

The property can include comparisons, for example: "Is the probability that the coin falls on head less than 75%". They can be time bounded, which means the probability or reward must be reached in a certain time. However, these are outside the scope of this research.

The precision of the result differs between algorithms. Precise algorithms often require a lot of time and memory. Relaxing the precision allows for faster and more memory-efficient algorithms. QComp describes five precision guarantees [16].

1. *Often ϵ-correct* results must be correct up to ϵ; however, the algorithm may produce a result that is not ϵ-correct. This happens for some cyclic models [13,25]. Value iteration (VI) produces often ϵ-correct results [7].
2. *Probably ϵ-correct* results must be correct up to ϵ-correct 95% of the time. This can be achieved by statistical model checking algorithms [51]. The result is approximated using sufficient simulation runs. Due to the random nature of the algorithm, it can be wrong 5% of the time.
3. *ϵ-correct* result must be correct up to an ϵ. This can be achieved using interval iteration [25]. It has an approximate upper and lower bound. These approximations are improved until it is sufficient for the result.
4. *Floating point correct* result must be the correct value, though it may be incorrect due to floating point arithmetic.
5. *Correct* results must match the rational probability or reward value. This can be achieved by using arbitrary precision rational numbers [41].

2.3 Quantitative formal models

Research shows there are several formal models for quantitative verification [31]. In this paper, we focus on the formal models of QComp. These are discrete-time Markov chain (DTMC), Markov

decision process (MDP), continuous-time Markov chain (CTMC), Markov automata (MA) and probabilistic timed automata (PTA). The mathematical descriptions are based on work from Hartmanns et al. [33]. Table 1 and 2 show an overview of the formal models.

Table 1. Type of transitions of the formal models

Formal model	Probability distribution	Rates	Non-deterministic
DTMC	Yes	No	No
MDP	Yes	No	Yes
CTMC	No	Yes	No
MA	Yes	Yes	Yes
PTA	Yes	No	Yes

Table 2. Type of time of the formal models

Formal model	Range	Memoryless	Non-deterministic
DTMC	Discrete	yes	no
MDP	Discrete	yes	no
CTMC	Continuous	yes	no
MA	Continuous	yes	no
PTA	Continuous	no	yes

The first formal model is DTMC, which is used to model stochastic systems. An example for which this formal model can be used is flipping a coin. The corresponding property is: "What is the probability of getting three heads in a row with ten flips?"

A (discrete) *probability distribution* over S is a function $\mu : S \to [0, 1]$ with countable *support* $spt(\mu) \stackrel{\text{def}}{=} \{ s \in S \mid \mu(s) > 0 \}$ and $\sum_{s \in spt(\mu)} \mu(s) = 1$. $Dist(S)$ is the set of all probability distributions over S. $2^{Dist(S)}$ is the power set of all probability distributions over S [33]. A *rate* over S is a function $\phi : S \to [0, \infty)$ with countable *support*. $Rate(S)$ is the set of all rates over S.

Definition 1. *A discrete-time Markov chain (DTMC) is a triple*

$$M = \langle S, s_0, T \rangle$$

where S is a finite set of states with $s_0 \in S$, s_0 is the initial state and $T \colon S \to Dist(S)$ is the transition function.

If the DTMC is extended with non-determinism, the result is a Markov decision process (MDP). Non-determinism is an unknown in the system. It is used to model choices or for unknown probabilities. An MDP can be used to model a game of blackjack. In the game, a player decides whether or not to draw a card. There is no fixed probability for drawing a card, as the choice depends on external factors. Which card the player receives is probabilistic. There are two types of probability properties for MDP, maximum and minimum probability. A maximum probability property for MDP is "What is the maximum probability of getting twenty-one?"

Definition 2. *A Markov decision process (MDP) is a triple*

$$M = \langle S, s_0, T \rangle$$

where S is a finite set of states with $s_0 \in S$, s_0 is the initial state and $T \colon S \to 2^{Dist(S)}$ is the transition function. $T(s)$ must be finite and non-empty for all $s \in S$.

The previous two models have a discrete notion of time, meaning that time is measured in the number of discrete steps. The following section presents three models with continuous time, meaning that each transition progresses time with a real value. There are two ways of representing time. One is a memoryless way using the exponential distribution and the other uses memory and non-determinism.

We start with a model that uses memoryless time, namely a continuous-time Markov chain (CTMC). A CTMC has transitions with a rate. The rate indicates how fast we take the transition. The time between the transitions is exponentially distributed, which has the property of being memoryless. This means that the probability of leaving the state in a specific amount of time is independent of how long we are in the state. A CTMC does not have non-determinism. This formal model can be used to model the queue of a casino. The customers arrive at a certain rate. Before entering, the age of the customer is checked, which takes some time. A property that can be checked is "What is the probability that the queue is longer than 4 people?"

Definition 3. *A continuous-time Markov chain (CTMC) is a triple*

$$M = \langle S, s_0, T \rangle$$

where S is a finite set of states with $s_0 \in S$, s_0 is the initial state and $T \colon S \to Rate(S)$ is the transition function.

A Markov automaton (MA) [22, 23] is a model with probabilities, non-determinism and time. This model has two types of transitions. Timed transitions like CTMC and probabilistic non-deterministic transitions like MDP. The transitions, like an MDP, have an action associated with them. The actions are used for composition. An MA can be used to model the repair of a slot machine. For each use, there is some probability that it breaks down. Repairing the slot machine takes time. A property for MA is "What is the probability that it has broken down within an hour?"

Probabilistic timed automata (PTA) [40] have a non-deterministic notion of time, which is different from the previous two models with time. Probabilistic timed automata can express probabilities, non-determinism and time. The time is represented by a set of clocks. A clock interpretation is a function that assigns a time to each clock. A transition starts from a location and ends in a location. A transition can only be taken if the clock constraints are satisfied. A clock constraint can be part of a transition or location. There are two types of transitions. Either time passes or there is a probabilistic non-deterministic transition like MDP. A state is a location and clock interpretation.

A PTA is useful when it is known that something happens within hard deadlines. For example, the armored car for collecting the casino's money arrives between 16:00 and 17:00. It is unknown what the probability is that the truck arrives at a specific time. We do know that it is between 16:00 and 17:00. An example property is "What is the maximum probability that the safe is full before the truck arrives?"

2.4 JANI

JANI is a language for storing different kinds of formal models, including those mentioned in the previous section. JANI has been developed by Budde et al. [15]. This format is used in the Quantitative verification benchmark set (QVBS) [34], which is a benchmark for tools in quantitative verification. The benchmark instances were originally written in different languages. They have been transformed to the JANI language.

JANI Before JANI was developed, the tools had weak interoperability since each tool had its own input language. As a result, it was hard to compare algorithms that were implemented in different tools. JANI was developed to solve that problem. Most tools support JANI and there are transformations from JANI to and from other languages. The JANI language has been built on top of JSON. Since most programming languages have a parser for JSON, tool developers do not need to build a custom parser for JANI. The JANI specification stores the state space symbolically. This means that states are encoded using variables instead of being stored explicitly. Storing the state space explicitly creates huge files and the symbolic information is lost. The symbolic information is important for algorithms that store the state space in a BDD [35].

Languages which convert to JANI The languages Prism and Modest have been purposely made for model checking. Modest has a syntax based on classical process algebra. It was originally for stochastic timed systems [11] after which it was updated to stochastic hybrid systems [26]. The Prism language uses reactive modules, which run in parallel [2]. Unlike Modest it does not have control flow constructs like loops. The languages pGCL [42], PPDDL [54], Galileo (with extensions) [49] and GreatSPN [3] are not supported in most tools for quantitative verification; however, with a conversion to JANI they can be verified. This might come at a performance loss. A dedicated tool could be faster because it can make use of the structure. pGCL is a language for probabilistic programming, which converts to a DTMC. PPDDL is a domain language for probabilistic planning. It converts to an MDP. Galileo is used to store dynamic fault trees, which can be converted to a MA. GreatSPN is a language for storing generalised stochastic petri nets. It converts to a MA or CTMC. The languages IOSA [21] and xSADF [32] are not part of the benchmark set. They can be converted to JANI. IOSA is used to model a stochastic automaton. xSADF is used to model flexible scenario-aware dataflow.

2.5 Tools for quantitative verification

This section presents an overview of tools for quantitative verification, which includes the Modest Toolset [30], Prism [39], Storm [35], PET [13], Stamina [44], DFTRES [50] and ePMC [27]. In addition, the representation of the state space, quantitative formal models and languages of the tools are discussed in this section. This overview is based on the descriptions in QComp [16]. Most tools contain multiple algorithms. An *algorithm* consists of an engine and a solution method. The *engine* is code, which dictates how to build and represent the state space. The *solution method* is code, which calculates the property. The choice of solution method and engine are orthogonal to each other. However, not every solution method is compatible with every engine.

Table 3. Engines in the prominent tools

Tool	Storm	Prism	the Modest Toolset
explicit	sparse matrices jit abstraction refinement	explicit	mcsta
symbolic	decision diagram abstraction refinement	mtbdd	
explicit and symbolic	dd to sparse matrices hybrid	sparse hybrid	symblicit
partial state space	exploration		Modysh
statistical model checking	simulator		modes

The prominent tools are the Modest Toolset, Prism and Storm, based on the performance in QComp and the number of algorithms. Table 3 shows the engines in these tools. The Modest Toolset contains three engines, a probabilistic model checker mcsta, statistical model checker modes [14] and modysh a partial state space model checker. These support the languages Modest [26] and JANI [15]. Mcsta uses an explicit state space. It supports the aforementioned quantitative formal models. Modes is a statistical model checker. It does not support non-determinism by default. However, it can generate random schedulers that approximates the maximum and minimum probability. Modysh implements the algorithm GLRTDP [36]. It has support for DTMC and MDP.

Prism contains the engines explicit, MTBDD, sparse and hybrid. The engine explicit uses sparse matrices. The engine mtbdd uses BDD. Sparse and hybrid use a combination of both sparse matrices and BDD. It supports the Prism language [2] and the JANI language indirectly, because JANI can be transformed to the Prism language. The formal models DTMC, CTMC, MDP and PTA are supported.

Storm is a probabilistic model checker with support for DTMC, CTMC, MDP and MA. It supports PTA using a conversion from PTA to MDP from the Modest Toolset. Storm supports the

languages JANI and Prism. Storm contains the engines exploration, dd, jit, sparse, dd-to-sparse, hybrid and abstraction-refinement. Exploration uses a partial state space. dd is a symbolic engine. Jit and sparse use sparse matrices, which means they are explicit engines. Dd-to-sparse and hybrid use a combination of BDD and sparse matrices. Abstraction refinement uses either BDD or sparse matrices. The default setting uses sparse matrices. The sparse engine of Storm is not the same as the sparse engine of Prism, as can be seen in Table 3.

The tools PET and Stamina use a partial state space. PET and Stamina build on top of PRISM, which is why they support the Prism language and indirectly the JANI language. PET supports DTMC, CTMC and MDP. Stamina can truncate infinite CTMC to finite CTMC.

DFTRES is a statistical model checker. It supports Galileo (with extensions) [49] and JANI for the formal models DTMC and CTMC. ePMC (formerly iscasMC) is a probabilistic model checker with support for JANI and Prism. It supports DTMC, CTMC and MDP.

3 Related Work

This section provides an overview of the related work in the field of performance testing and quantitative verification. Quantitative verification competition (QComp), competitions for quantitative verification and Storm's automatic engine will be discussed in the next section.

3.1 QComp

Quantitative verification competition (QComp) [16] is a competition among tools for quantitative verification. Nine tools are evaluated on a subset of the quantitative verification benchmark set (QVBS) [34]. The tools are compared based on solving speed and precision guarantee. The section below focuses on the results of QComp. Section 2.2 describes the precision guarantee of QComp in more detail. Section 2.5 describes the formal models, supported languages and engines of the tools compared in QComp.

Only Storm participates in the correct result track. Therefore, the competition could not make a performance comparison for the correct result track. Mcsta and Storm are participating in the floating-point correct track. Storm has an automatic engine selector, which selects an algorithm based on the benchmark instance. It participated with and without the automatic engine selector. The automatic engine selector is referred to as Storm-automatic and using one algorithm is referred to as Storm-static. Mcsta and Storm-static have similar performance in the floating-point correct track. Storm-automatic has a better performance compared to Mcsta and Storm-static. The ϵ-correct track has Prism, mcsta and Storm-static and Storm-automatic participating. Storm-automatic has the best performance. The other tools have similar performance to each other. From highest to lowest number of fastest benchmark instances for the probably ϵ-correct track is Storm, mcsta, modes, Prism, Pet and DFTRES. All of the nine tools participate in the ϵ-correct track. Storm-automatic is the fastest on 46 benchmark instances, mcsta on 26, Prism on 17, MFPL 8, modes on 8, DFTRES on 2, ePMC on 2 and Stamina on 1. Storm-automatic has on average the best performance and each tool had at least one benchmark instance in which it excelled.

The QComp compares the performance of the tools. A default algorithm is used for each tool; however, in this paper, we focus on algorithms. This means that we can not use the measurements of QComp, because we want to know the performance of other algorithms besides the default one. However, we can use the foundation of QComp. We reuse the benchmark, tools and precision guarantee for our analysis.

3.2 Competitions for automated provers

We investigated the analysis of competitions for automated provers. Toolympics presents an overview of competitions for formal methods [8]. It was published as part of the 25th anniversary of TACAS. We investigated competitions that focus on automated provers, because this paper focuses on algorithms for quantitative verification, which are also automated provers. The competitions that we investigate are SAT Competition [24](SAT), Model checking competition [38](MCC), CADE automated theorem proving [53](CASC), and Software verification comparative evaluation [10](SV-COMP).

SAT Competition SAT is a competition for Boolean satisfiability solving. Boolean satisfiability solvers are often used as a back-end for tools for real-world problems. The real-world focused tools benefit from the maturity and performance of SAT solvers. The competition has been split into multiple tracks. These are main, multi-core, cloud, incremental, planning, glucose hack and no-limit. The main track is concerned with sequential solvers. Multi-core and cloud tracks are for asynchronous solvers. Incremental and planning tracks are focused on applying SAT to a real-world problem. Glucose hack concerns solvers, which change a small piece of code in an existing solver. No-limit is for solvers, which do not qualify for the main track.

The benchmark consists of new benchmark instances submitted by participants and old ones from previous years. The benchmark instances are randomly selected from all the submissions while taking into account the balance between SAT and UNSAT.

Solvers are ranked using Par-2 scoring. The score for an algorithm on a benchmark instance is 10000 if it took more than 5000 seconds; otherwise, it is the solving time in seconds. The Par-2 score is the average score over all the benchmark instances. Solvers are disqualified if they give an incorrect result. The results are analysed more in-depth beside the ranking. One of the analysis is to look at the score similarity between two solvers.

Model checking competition MCC is a competition for Petri net solvers. Petri nets are used to assess the safety and security of asynchronous concurrent systems. A petri net solver answers queries related to state space generation, deadlock detection, upper bounds computation, reachability and formulas in CTL and LTL. The models come from academia and industry. The three best performing tools are highlighted in the form of a medal and podium. Kordon et al. made an analysis of the MCC from 2015 to 2019, which includes an overview of the techniques used by the tools [38]. They observed that the winning tools implement multiple techniques that sometimes run concurrently.

CADE automated theorem proving competition CASC is a competition for fully automated theorem provers. It focuses on classical logic. The solvers are ranked based on the number of instances solved. The CPU time and wall clock time are also reported. The old version of last year's best tool is rerun. This is to gain insight into the progression of that year.

Software verification comparative evaluation competition SV-COMP is concerned with the verification of software. They have a sizable benchmark with C and Java verification tasks. The ranking is based on a score, which increases for a correct proof and decreases more drastically for an incorrect proof. The score stays the same when a verification task is not solved. The scores are split into different types of verification problems. The most correct and energy-efficient verifiers are also highlighted.

Summary We have discussed four competitions. Each of them focuses on ranking the tools. The rankings are based on the total performance of the benchmark. The performance is often defined as time. However, energy efficiency and correctness are also used. We investigate whether the in-depth analysis of SAT is helpful in answering our research question.

3.3 Storm's automatic engine

The tool Storm has an automatic engine selector, which selects an algorithm based on the benchmark instance [35]. The selector is a decision tree generated using the tool scikit-learn [46]. The selector uses characteristics of the JANI document. The next section looks at the algorithms and characteristics of the selector.

The selector chooses among four algorithms and eight different characteristics [3] [4] [5]. The engines are sparse engine, sparse engine with rational arithmetic, hybrid engine and dd-to-sparse engine. Five of the characteristics describe the size of the benchmark instance. These are: the number of non-transient variables, the domain of states (range of the non-transient variables multiplied by the number of locations), the average range of the variables, number of edges and number of automata. The domain of states is an overestimation of the number of states. Three characteristics describe the type of benchmark instance. These are whether the property is bounded by time or reward, whether the model contains continuous-time and whether it contains non-determinism. We take inspiration from these characteristics for our research. Storm is only concerned with characteristics of the syntax, whereas we focus on characteristics that use the state space.

[3] https://github.com/moves-rwth/storm/blob/a7a10136fa074a7463c785deb06927c061361ba8/src/storm/utility/AutomaticSettings.cpp

[4] https://github.com/moves-rwth/storm/blob/a7a10136fa074a7463c785deb06927c061361ba8/src/storm/storage/jani/traverser/InformationCollector.h

[5] https://github.com/moves-rwth/storm/blob/a7a10136fa074a7463c785deb06927c061361ba8/src/storm/storage/jani/Model.h

4 Design

The following section discusses the design of the benchmark. The subsections are ordered by the research questions presented in Section **??**, which are algorithms, benchmark instances, characteristics, performance metrics, comparing performance and detecting statistical associations.

4.1 Which algorithms do we want to analyse?

This section presents an overview of algorithms for quantitative verification, after which the question is answered.

What algorithms are available for quantitative verification? Due to time constraints, we looked at a subset of the tools discussed in Section 2.5. We decided to scope our search to the three prominent tools Storm[6], Prism[7] and the Modest Toolset[8], because these have a wide spread of algorithms implemented and the performance is fast, as can be seen in QComp [16].

We present a detailed overview of algorithms for quantitative verification. Table 4 shows algorithms for DTMC and CTMC, Table 5 shows algorithms for MDP and MA and Table 6 shows algorithms for PTA. The overview considers algorithms for calculating the (maximum and minimum) reachability probability or the (maximum and minimum) expected reward.

Our overview goes beyond what was previously available in the literature. QComp [16] has an overview; however, it focuses on tools, while this study focuses on algorithms. Our overview has been primarily based on help messages, guides and papers of the tools [35, 45]. Personal communication with the maintainer and the tool were used to clear up any ambiguities.

The Tables show which solution method is implemented in which engine. As mentioned in Section 2.5, an algorithm consists of an engine and a solution method. The type of marker shows the support for formal models and properties.

1. ✓means it supports both the (maximum and minimum) reachability probability and the (maximum and minimum) expected reward for the formal model specified under the table.
2. $\checkmark^{P}$means it only supports the reachability probability.
3. $\checkmark^{D}$means it only supports discrete-time models (DTMC and MDP).
4. $\checkmark^{DP}$is a combination of $\checkmark^{P}$and $\checkmark^{D}$.

Some solution methods have settings e.g., Prism simulator has a maximum path length and Storm has bisimulation minimisation simulation. We did not investigate these settings. 3 We have used the full name of algorithms and engines in the tables where possible instead of the acronyms. “dd to sparse matrices” stands for “decision diagram to sparse matrices” and “mtbdd” stands for “multi-terminal binary decision diagram”. In this thesis, we refer to power iteration in Storm and Prism as value iteration to ensure a consistent name of the algorithm across tools.

[6] version 1.6.4
[7] version 4.7
[8] version 3.1.190

Table 4. Algorithms for DTMC and CTMC

Tool	Storm							Prism					the Modest Toolset		
Engine															
Solution method	decision diagram	sparse matrices	dd to sparse matrices	hybrid	jit	exploration	abstraction refinement	sparse	explicit	hybrid	mtbdd	simulator	mcsta	symblicit	modes
value iteration [47]	✓^{D}	✓	✓	✓	✓			✓	✓	✓	✓		✓		
topological value iteration [19]		✓	✓	✓	✓			✓	✓	✓	✓				
sound value iteration [48]		✓	✓	✓	✓								✓		
topological sound value iteration		✓	✓	✓	✓										
optimistic value iteration [33]		✓	✓	✓	✓								✓		
topological optimistic value iteration		✓	✓	✓	✓										
interval iteration [25]		✓	✓	✓	✓								✓		
topological interval iteration		✓	✓	✓	✓										
sequential interval iteration [29]													✓^{P}		
Walker-Chae [17]		✓	✓	✓	✓										
topological Walker-Chae		✓	✓	✓	✓										
Gauss-Seidel		✓	✓	✓	✓			✓	✓	✓					
topological Gauss-Seidel		✓	✓	✓	✓			✓	✓	✓					
backwards Gauss-Seidel								✓	✓	✓					
topological backwards Gauss-Seidel								✓	✓	✓					
pseudo Gauss-Seidel										✓					
topological pseudo Gauss-Seidel										✓					
backwards pseudo Gauss-Seidel										✓					
topological backwards pseudo Gauss-Seidel										✓					
successive over relaxation		✓	✓	✓	✓			✓	✓	✓					
topological successive over relaxation		✓	✓	✓	✓			✓	✓	✓					
backwards successive over-relaxation								✓	✓	✓					
topological backwards successive over-relaxation								✓	✓	✓					
pseudo successive over-relaxation										✓					
topological pseudo successive over-relaxation										✓					
backwards pseudo successive over-relaxation										✓					
topological backwards pseudo successive over-relaxation										✓					
Jacobi	✓^{D}	✓	✓	✓	✓			✓	✓	✓	✓				
topological Jacobi		✓	✓	✓	✓			✓	✓	✓	✓				
Jacobi with over-relaxation								✓	✓	✓	✓				
topological Jacobi with over-relaxation								✓	✓	✓	✓				
elimination		✓	✓	✓	✓										
topological elimination		✓	✓	✓	✓										
symblicit state elimination														✓^{P}	
linear programming													✓^{P}		
gmm++		✓	✓	✓	✓										
topological gmm++		✓	✓	✓	✓										
eigen		✓	✓	✓	✓										
topological eigen		✓	✓	✓	✓										
rational search	✓^{D}	✓	✓	✓	✓										
topological rational search		✓	✓	✓	✓										
acyclic		✓	✓	✓	✓										
exploration						✓^{DP}									
abstraction refinement							✓^{DP}								
confidence interval												✓			✓
asymptotic confidence interval												✓			
approximate probabilistic model checking (Okamoto)												✓			✓
adaptive															✓

Table 5. Algorithms for MDP and MA

Tool	Storm							Prism				the Modest Toolset		
Engine	decision diagram	sparse matrices	dd to sparse matrices	hybrid	jit	exploration	abstraction refinement	sparse	explicit	hybrid	mtbdd	mcsta	symblicit	Modysh
Solution method														
value iteration [47]	✓^{D}	✓	✓	✓	✓			✓^{D}	✓^{D}	✓^{D}	✓^{D}	✓		
topological value iteration [19]		✓	✓	✓	✓			✓^{D}	✓^{D}	✓^{D}	✓^{D}			
sound value iteration [48]		✓	✓	✓	✓							✓		
topological sound value iteration		✓	✓	✓	✓									
optimistic value iteration [33]		✓	✓	✓	✓							✓		
topological optimistic value iteration		✓	✓	✓	✓									
interval iteration [25]		✓	✓	✓	✓							✓		
topological interval iteration		✓	✓	✓	✓									
sequential interval iteration [29]												✓^{P}		
policy iteration [9]	✓^{D}	✓	✓	✓	✓				✓^{D}					
topological policy iteration		✓	✓	✓	✓									
modified policy iteration									✓^{D}					
value iteration to policy iteration		✓	✓	✓	✓									
topological value iteration to policy iteration		✓	✓	✓	✓									
linear programming		✓	✓	✓	✓							✓^{P}		
topological linear programming		✓	✓	✓	✓									
rational search	✓^{D}	✓	✓	✓	✓									
topological rational search		✓	✓	✓	✓									
acyclic		✓	✓	✓	✓									
exploration						✓^{DP}								
abstraction refinement							✓^{DP}							
Gauss-Seidel									✓^{D}					
topological Gauss-Seidel									✓^{D}					
symblicit state elimination													✓^{P}	
general labeled real-time dynamic programming [36]														✓^{P}

Table 6. Algorithms for PTA

Tool	Prism			the Modest Toolset
Engine				
Solution method	stochastic games	digital clocks	backwards reachability	mcsta
stochastic games	✓^{P}			
digital clocks		✓		
backwards reachability			✓^{P}	
value iteration [47]				✓
interval iteration [25]				✓
sequential Interval iteration [29]				✓^{P}
sound value iteration [48]				✓
optimistic value iteration [33]				✓
linear programming				✓^{P}

Which algorithms do we want to analyse? We would like to analyse every combination. However, there are over two hundred combinations. Some algorithms have settings that influence their behaviour. Trying all different settings will further increase the number of combinations. It is only possible to analyse some of these combinations due to time constraints. We would like to include every engine and solution method at least once.

We decided to analyse the algorithms shown in Table 7 and 8. The default settings are used for all algorithms. From now on, we will use the short names shown in Table 7 and 8 for the algorithms.

Table 7 shows most solution methods in combination with a sparse matrices engine, if available. A different engine is chosen if the solution method is not implemented in a sparse matrices engine. This ensures that every solution method is included.

Table 8 shows variations of value iteration in combination with most engines. The variations of value iteration that we include are topological, bisimulation minimization and both topological and bisimulation. This ensures that every engine is included.

We did not include the solution method acyclic and engines exploration and jit from Storm. Acyclic is not included because it does not support cyclic models. The exploration engine was not included since it did not support JANI files. The jit engine was not included because it is not part of the paper of Storm and is similar to the sparse engine [35].

4.2 Which benchmark instances do we want to use?

We used benchmark instances from the Quantitative Verification Benchmark Set (QVBS) [34]. QVBS consists of benchmark instances from PRISM Benchmark Suite [39], International Planning Competitions [54] and various smaller collections. We believe that a diverse set of benchmark instances supports the analysis. To ensure this, we decided to include every property once.

We did not have time to handpick a parameter for each benchmark instance, which is why the parameters are randomly selected. The parameter is uniformly selected from QVBS per property. The random selection is based on the random benchmark instance selection from the SAT competition [24]. The SAT competition randomly generates a benchmark from new and old benchmark instances, as discussed in Section 3.2.

Tables 26, 27 and 28 in the appendix show the selected benchmark instances and parameters. The maximum allowed time to solve a benchmark is 15 minutes.

Table 7. Algorithms with shortname

Tool	Engine	Algorithm	Short name
Storm	sparse matrices	sound value iteration	S sparse SVI
		optimistic value iteration	S sparse OVI
		interval iteration	S sparse II
		walkerchae	S sparse Walker-Chae
		Gauss-Seidel	S sparse GS
		successive over relaxation	S sparse SOR
		Jacobi	S sparse J
		elimination	S sparse elimination
		gmm++	S sparse gmm++
		eigen	S sparse eigen
		policy iteration	S sparse PI
		value iteration to policy iteration	S sparse VI to PI
		linear programming	S sparse LP
		rational search	S sparse rational
	abstraction refinement	abstraction refinement	S abs
Prism	explicit	Gauss-Seidel	P explicit GS
		backwards Gauss-Seidel	P explicit back GS
		successive over relaxation	P explicit SOR
		backwards successive over-relaxation	P explicit back SOR
		Jacobi	P explicit J
		Jacobi with over-relaxation	P explicit JOR
		policy iteration	P explicit PI
		modified policy iteration	P explicit MPI
	simulator	confidence interval	P simulator CI
		asymptotic confidence interval	P simulator ACI
		approximate probabilistic model checking (Okamoto)	P simulator APMC
	stochastic games	stochastic games	P stochastic games
	digital clocks	digital clocks	P digital clocks
	backwards reachability	backwards reachability	P back reachability
The Modest Toolset	mcsta	sound value iteration	M mcsta SVI
		optimistic value iteration	M mcsta OVI
		interval iteration	M mcsta II
		sequential interval iteration	M mcsta SII
		linear programming	M mcsta LP
	Modysh	general labeled real-time dynamic programming	M GLRTDP
	modes	confidence interval	M modes CI
		approximate probabilistic model checking (Okamoto)	M modes APMC
		adaptive	M modes adaptive

Table 8. Value iteration variants with shortname

Tool	Engine	Algorithm	Short name
Storm	sparse matrices	value iteration	S sparse VI
		topological value iteration	S sparse TVI
		value iteration with bisimulation	S sparse bis VI
		topological value iteration with bisimulation	S sparse bis TVI
	dd to sparse matrices	value iteration	S dd to sparse VI
		topological value iteration	S dd to sparse TVI
		value iteration with symbolic bisimulation	S dd to sparse bis VI
		topological value iteration with symbolic bisimulation	S dd to sparse bis TVI
	hybrid	value iteration	S hybrid VI
		topological value iteration	S hybrid TVI
		value iteration with symbolic bisimulation	S hybrid bis VI
		topological value iteration with symbolic bisimulation	S hybrid bis TVI
	decision diagram	value iteration	S dd VI
		value iteration with symbolic bisimulation	S dd bis VI
Prism	sparse	value iteration	P sparse VI
		topological value iteration	P sparse TVI
	explicit	value iteration	P explicit VI
		topological value iteration	P explicit TVI
	hybrid	value iteration	P hybrid VI
		topological value iteration	P hybrid TVI
	mtbdd	value iteration	P mtbdd VI
		topological value iteration	P mtbdd TVI
The Modest Toolset	mcsta	value iteration	M mcsta VI

4.3 Which characteristics do we want to investigate?

To get ideas for characteristics for benchmark instances to investigate, we interviewed Tom van Dijk, Hajo Broersma and Arnd Hartmanns from the University of Twente. Tom van Dijk does research into high-performance algorithms and binary decision diagrams. Hajo Broersma does research into graphs. Arnd Hartmanns researchers model checking and is one of the supervisors of this research.

The next section discusses the characteristics gathered using preliminary knowledge. After that, the characteristics gathered using interviews will be discussed. Lastly, the answer to the question of this section "which characteristics do we want to investigate?" will be answered.

Preliminary knowledge The characteristics gathered using preliminary knowledge are categorized into three categories. These categories are characteristics related to the size, the structure of one state and the structure of the state space. Characteristics related to size are the number of states, the number of transitions and the number of branches. Characteristics related to the structure of one state are the average number of transitions, the average number of reachable states from a state using one transition and the number of self-loops. A self-loop is a transition going to the state it originates from. Finally, characteristics related to the structure of the state space are the sparseness of a transition matrix, the number of reachable states (from the initial state), the number of reachable states using an efficient policy and the number of reachable states which can not reach the goal state.

Interviews The characteristics that Tom van Dijk mentioned are the number of vertices in the binary decision diagram, the number of states with similar behaviour and the number of maximal strongly connected components.

Hajo Broersma mentioned sinks (states with only incoming transitions) and sources (states with only outgoing transitions). After this we discussed characteristics related to the cyclicity of a graph, namely: the number of connections between components, the number of paths between the initial and goal state, the number of self-loops, the number of cycles and graph entropy.

Arnd Hartmanns talked about characteristics related to size, structure and cyclicity. The characteristics for size are the range of variables, the number of locations, the number of states, the number of goal states and the number of avoid states. The structures he mentions are strongly connected components, maximal end components and zero reward end components. A characteristic can be formed based on the number and size of the components. Characteristics related to cyclicity are: is the graph acyclic, measurement of the cyclicity, number of transitions and how big is the support of the probability distribution of a transition. Characteristics that did not fall into a category are the type of property and the complexity of the policy.

Characteristics Due to time constraints, we do not have time to implement code to calculate these characteristics. The number of states, the number of transitions and the number of branches are calculated by the tools, which is why we choose those characteristics.

4.4 Which performance metrics do we want to investigate?

Memory usage, speed and precision can be seen as the performance of an algorithm. We decided to investigate the speed of algorithms. An algorithm spends time on parsing, state space building and calculating the result, as discussed in Section 2.1. We decide to investigate wall-clock time, state space time and property time. *Wall-clock time* is the total time to verify the model. This includes starting the tool and printing the result. *State space time* is the time to build the state space. This includes time to apply bisimulation minimization. *Property time* is the time to calculate the result. Algorithms that use statistical model checking or a partial state space do not show the state space time. The state space time is therefore included in the property time. We do not measure the time to parse the file since it is small. Therefore, it is not a limitation in most cases.

4.5 Which techniques do we use to compare performances?

We came across several techniques to compare the performance of algorithms, as discussed in Section 3. This section highlights the similarity metric from the SAT competition [24] and scatter plots used in QComp [16]. We adapt both these techniques so that we can use them in our analysis.

The similarity metric uses a score called PAR-2. The PAR-2 score of an algorithm on a benchmark instance is the wall-clock time if it is solved; otherwise, it is twice the maximum allowed time. As mentioned in Section 4.2, the maximum allowed time is 15 minutes. The similarity metric between two algorithms in the SAT competition is the average difference in PAR-2 scores for all benchmark instances normalized to the interval $[0, 1]$. The metric is 1 if the PAR-2 scores are the same. The metric is 0 if the difference is always twice the maximum allowed time. We changed the metric by scaling it to the interval of $[0, 10]$ and removing benchmark instances, which are unsupported by one of the algorithms. We remove unsupported benchmark instances because the goal is to compare the performance of algorithms and not the support of benchmark instances. The modified similarity metric between two algorithms is the average difference in PAR-2 scores for benchmark instances, which both algorithms support normalized to the interval $[0, 10]$.

Definition 4. *The similarity metric between two algorithms A and A' is*

$$Similarity(A, A') = \left(1 - \frac{\sum(|A_i - A'_i| * IsSupported(A, i) * IsSupported(A', i))}{Supported(A, A') * TimeOut * 2}\right) * 10$$

where A_i is the PAR-2 score of algorithm A on the i'th benchmark instance, $Supported(A, A')$ is the number of benchmark instances that both A and A' support, $IsSupported(A, i)$ is 1 if A supports the i'th benchmark instance; otherwise, it is 0 and TimeOut is the maximum allowed time to solve a benchmark instance.

Notice that the metric uses the absolute difference. Two benchmark instances with a PAR-2 score of 1 and 10 have the same similarity as those with PAR-2 score of 101 and 110, even though the first benchmark instances have a relative difference of 10. We did not change this metric because for the end user of a tool, waiting 1 second or 10 seconds does not make that much difference. The metric de-emphasis fast measurements, which will be important later when we analyse the results.

The SAT competition only uses this metric for the analysis of algorithms. However, we want to compare the similarity of benchmark instances, which is why we flipped the metric around.

Definition 5. *The similarity metric between two benchmark instances B and B' is*

$$Similarity(B, B') = \left(1 - \frac{\sum(|B_i - B'_i| * IsSupported(B, i) * IsSupported(B', i))}{Supported(B, B') * TimeOut * 2}\right) * 10$$

where B_i is the PAR-2 score of the i'th algorithm on B; $Supported(B, B')$ is the number of algorithms that support both B and B'; $IsSupported(B, i)$ is 1 if the i'th algorithm supports B; otherwise it is 0 and $TimeOut$ is the maximum allowed time to solve a benchmark instance.

The scatter plots used in QComp [16], give a deep insight into the performance of the tools. However, this approach does not scale to the number of algorithms we have. That is why we propose the win metric.

Definition 6. *The win metric between two algorithms A and A' is*

$$Win(A, A') = \left(\frac{Wins(A, A')}{Wins(A, A') + Wins(A', A)}\right) * 10$$

where $Wins(A, A')$ the number of benchmark instances where A has a lower PAR-2 score compared to A'. Benchmark instances where one of the algorithms does not support it are excluded.

The formula reads as the win metric is the wins divided by the wins and losses normalized to the interval $[0, 10]$. Given an algorithm and an opponent algorithm. The metric is 10 if the algorithm is faster than the opponent on all benchmark instances. The algorithms may have the same PAR-2 score. This means it is a draw, where both algorithms do not win. In case there are only draws and no wins or losses, the metric is 0.

4.6 Which techniques do we use to detect statistical associations between the characteristics of a benchmark instance and the performance of an algorithm?

We use the Pearson correlation coefficient to detect linear relations [1]. In this research, we do not look at exponential or multivariate relations for the simplicity of this research. However, that will remain for future work.

The Pearson correlation classifies how linearly correlated the data is. A correlation of 1 means that the data points form a line going up, a correlation of 0 means the data is scattered and a correlation of -1 means the data points form a line going down. Dancey et al. [20] classify a coefficient above 0.7 as strong, between 0.4 and 0.7 as moderate and between 0 and 0.4 as weak. The same holds for negative coefficients. The p-value of the Pearson correlation says how likely the correlation was due to random chance. To decrease the chance of finding correlations due to random chance, we will only look at correlations with a p-value of less the 0.05.

5 Measurements

The following section will discuss the research questions "What is the performance of the algorithms on the benchmark instances?" and "What characteristics do the benchmark instances have?" We create a tool to automatically collect the performance of algorithms and characteristics of benchmark instances[9]. The design of the tool and the results will be explained in the following section. Starting with the results.

5.1 Results

Tables 9, 10 and 11 show the number of results per algorithm. We observed that the number of finished benchmark instances differs widely between algorithms and some algorithms throw errors. The next section discusses those observations more in-depth.

The number of finished benchmark instances differs between the algorithms. This is mainly attributed to the support in formal models and properties, as discussed in Section 4.1. However, there is also a technical reason. The algorithms in Prism only support Prism files, while most of the benchmark instances are in JANI files. There is a conversion between JANI and Prism files. Unfortunately, not all features of JANI are in the Prism language. Due to the unsupported features, we decided to test the Prism algorithms only on benchmark instances with Prism files.

The measured states, transitions and branches were different between the tools. A possible explanation for this is the optimizations of the tools and differences between Prism and JANI files. The average amount is used in the analysis of benchmark instances in Section 6.2. The amount reported by the tool is used for the statistical analysis in Section 6.3. However, not all algorithms report the characteristics of benchmark instances. These are those that do not build the whole state space. For these algorithms, we use the characteristics reported by another algorithm of the same tool. Algorithms `P simulator CI`, `P simulator ACI` and `P simulator APMC` use the characteristics of `P explicit VI`. Algorithms `P stochastic games` and `P back reachability` use the characteristic of `P digital clocks`. The algorithms `M GLRTDP`, `M modes CI`, `M modes APMC` and `M modes adaptive` use the characteristics of `M mcsta VI`.

[9] https://github.com/Vescatur/QVAnalyser

Table 9. Results for algorithms in Storm

Algorithm	Supported	Threw error	Timed out	Finished
S sparse VI	111	1	22	88
S sparse TVI	111	0	22	89
S sparse bis VI	86	0	34	52
S sparse bis TVI	86	0	34	52
S sparse SVI	111	0	23	88
S sparse OVI	111	0	23	88
S sparse II	111	0	23	88
S sparse walkerchae	29	0	20	9
S sparse GS	29	0	7	22
S sparse SOR	29	0	7	22
S sparse J	29	0	7	22
S sparse elimination	29	0	9	20
S sparse gmm++	29	0	7	22
S sparse eigen	29	0	8	21
S sparse PI	82	0	19	63
S sparse VI to PI	82	0	18	64
S sparse LP	82	1	35	46
S sparse rational	111	3	59	49
S abs	43	5	12	26
S dd to sparse VI	99	13	9	77
S dd to sparse TVI	99	11	11	77
S dd to sparse bis VI	99	9	16	74
S dd to sparse bis TVI	99	11	14	74
S hybrid VI	99	9	9	81
S hybrid TVI	99	8	11	80
S hybrid bis VI	99	11	15	73
S hybrid bis TVI	99	10	16	73
S dd VI	69	5	8	56
S dd bis VI	69	6	12	51

Table 10. Results for algorithms in Prism

Algorithm	Supported	Threw error	Timed out	Finished
P explicit VI	57	15	0	42
P explicit TVI	57	17	0	40
P explicit GS	22	6	0	16
P explicit back GS	57	14	0	43
P explicit SOR	22	5	0	17
P explicit back SOR	6	6	0	0
P explicit J	6	6	0	0
P explicit JOR	6	6	0	0
P explicit PI	35	9	0	26
P explicit MPI	21	6	0	15
P sparse VI	52	4	0	48
P sparse TVI	52	4	0	48
P hybrid VI	52	4	0	48
P hybrid TVI	52	4	0	48
P mtbdd VI	58	8	0	50
P mtbdd TVI	58	9	0	49
P stochastic games	7	0	0	7
P digital clocks	3	0	0	3
P back reachability	5	2	0	3
P simulator CI	21	9	0	12
P simulator ACI	21	9	0	12
P simulator APMC	13	5	0	8

Table 11. Results for algorithms in the Modest Toolset

Algorithm	Supported	Threw error	Timed out	Finished
M mcsta VI	136	4	22	110
M mcsta SVI	136	3	24	109
M mcsta OVI	136	2	25	109
M mcsta II	136	6	24	106
M mcsta SII	86	2	12	72
M mcsta LP	86	9	22	55
M GLRTDP	57	1	23	33
M modes CI	27	2	12	13
M modes APMC	16	1	6	9
M modes adaptive	16	0	6	10

5.2 Tool

We created the tool QVAnalyser (Quantitative Verification Analyser) to collect the results. The tool can automatically collect the performance metrics and characteristics by running the algorithms on the benchmark instances. Tables 9 to 28 have been generated by the tool.

QVAnalyser is based on the tool used in QComp [16] and an internal tool from the Storm developers. It is programmed in Python. QVAnalyser is available on Github[10] and it has a Creative Commons attribution 4.0 licence[11].

The tool was created for three reasons. The first reason is that it is indispensable for this thesis since it is infeasible to run all the algorithms and create the tables manually. The second reason is to ensure the reproducibility of this research. There is a manual reproducing the result in Appendix A.1. The third reason is the possibility for future researchers to extend the research performed in this study by using or adapting the tool. The next section discusses how adaptability is incorporated into the architecture of the tool. The explanation serves as a guide on how to change the tool.

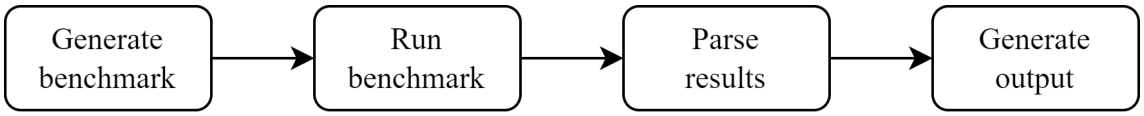


Fig. 4. Process of QVAnalyser

Architecture Figure 4 shows the four steps to go from a research question to results.

1. The first step is to create the benchmark. This is a class with a list of benchmark instances and algorithms. In our benchmark, the parameters of the benchmark instances are randomly selected. We use QVAnalyser to generate a class with random parameters.
2. The second step is to run the benchmark. This can last from an hour to weeks, depending on the number of algorithms and benchmark instances. The results are saved into separate files in case something goes wrong. The results are merged into one file when the benchmark is finished.
3. The third step is to parse the results. In this step, QVAnalyser parses the logs of the algorithms to gather the characteristics and performance metrics.
4. The fourth step is to generate the output. Examples of the output are Tables 9 to 28 and Figures 6 to 11.

Each step in Figure 4 can be started independently. This allows us to run the benchmark on a different machine and change the output without having to execute previous steps.

The tool is split into two folders to ensure adaptability. There is the library folder and the specific folder. The library folder contains generic code to run a benchmark. It can be kept the same if future researchers want to test other algorithms or benchmark instances. The specific folder contains code specific to this thesis. It contains connections to tools and algorithms, a benchmark with benchmark instances and code to generate the tables seen in this thesis.

Figure 5 shows the classes in the library folder. The arrow shows which classes have a reference to which classes. Some references are not shown for clarity. The library has three folders. These are benchmark, results and tools. The benchmark folder contains classes for benchmark instances. The `Benchmark` class is abstract and is extended in the specific folder. `Benchmark` can contain multiple objects of `BenchmarkSequence`. `BenchmarkSequence` can contain multiple objects of `BenchmarkInstance`. The idea was to run the same model with multiple parameters. However, this idea was never executed. In this thesis, a `BenchmarkSequence` always contains one `BenchmarkInstance`. `BenchmarkModel` contains a reference to the JANI files. The results folder contains

[10] https://github.com/Vescatur/QVAnalyser
[11] https://creativecommons.org/licenses/by/4.0/

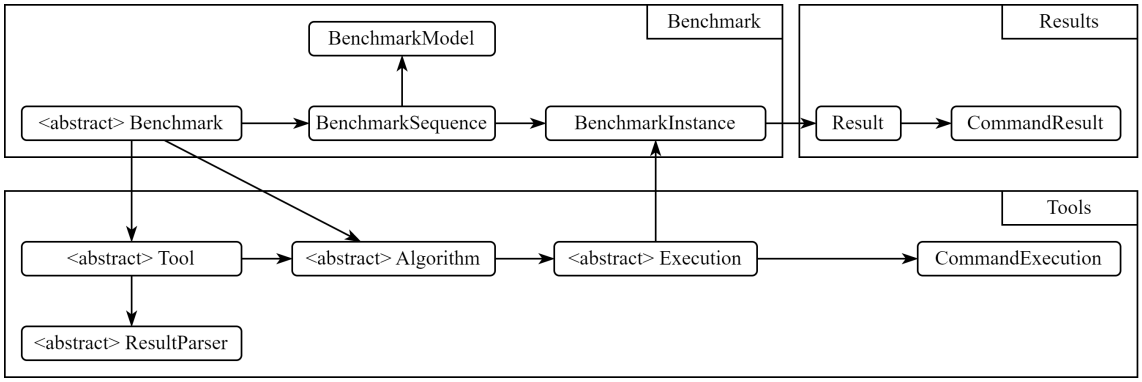


Fig. 5. Architecture of QVAnalyser

`Result` and `CommandResult`. `Result` contains the characteristics and performance metrics on a benchmark instance for one algorithm. `CommandResult` contains the log for one command. The tools folder contains classes for the execution of algorithms. `Tool` contains multiple objects of `Algorithm`. `Algorithm` specifies which benchmark instances are supported by the algorithm and starts `Execution`. `Execution` runs the algorithm on one benchmark instance. It uses `CommandExecution` to execute a command on the command line. `ResultParse` parses the log to find characteristics and performance metrics.

6 Analysis

6.1 How does the performance between algorithms compare?

We compare the performance using the similarity metric and win metric as defined in Section 4.5. Tables 12 and 13 show the similarity metric and Tables 14 and 15 show the win metric.

The next section discusses the similarity metric in which we identify clusters of algorithms. After this, we discuss the win metric in which we look into the algorithms with a high number of wins. Unfortunately, we do not have enough measurements for the algorithms of PTA, which is why they are discussed separately.

Similarity Table 12 shows the similarity of algorithms on the formal models DTMC and CTMC. Table 13 shows the similarity of algorithms on the formal models MDP and MA. The similarity metric gives an indication of how similar the performance is between two algorithms. Where 10 is the same (shown with a plus in the diagram) and 0 is completely different.

The cells in the tables show the similarity metric between the algorithm on the row and the algorithm on the column. The rows and columns have the same ordering, which means that the cells on the diagonal compare the algorithm to itself. That is why they all contain a plus. The cells have a colour based on the similarity metric. The value in the cell has been rounded, which can cause the same value to be coloured differently. 0 is coloured red, 8 is coloured white and 10 is coloured green. Any value in between is a mixture of these colours.

We are going to look at clusters of algorithms. A *cluster* is a set of algorithms that have a high similarity to each other. The clusters are created manually, after which we compare them to clusters created by a hierarchical clustering algorithm.

The algorithms are ordered based on similarity, which has the effect that the clusters become visible as green squares. To improve readability, the ordering is also based on the engine and state space storage technique. The ordering of state space storage techniques is statistical model checking, partial state space, explicit, and symbolic. These are explained in Section 2.2 and Section 2.5 explains which engines use which technique. Appendix A.3 contains similarity tables based on the ordering of the clustering algorithm.

We identify three clusters of algorithms in Table 12 with the formal models DTMC and CTMC. The first cluster contains algorithms 1 to 6. These are the statistical model checking algorithms. The second cluster is algorithm 7 and algorithms 11 to 41. It contains algorithms that use explicit techniques and algorithms that use a combination of explicit and symbolic. Algorithms 8, 9 and 10 also use explicit techniques. Nevertheless, they are not part of the cluster because these are not similar. We do not know why these algorithms are not similar to the cluster with explicit algorithms. The third cluster contains algorithms 34 to 39 and algorithms 42 to 49. This cluster contains algorithms that use a combination of explicit and symbolic techniques and algorithms that only use symbolic techniques. The algorithm identified slightly different clusters. A notable difference is that the algorithm does not allow overlap in clusters. Therefore, algorithms 36 to 39 are not part of the second cluster and 34 to 35 are not part of the third cluster.

Table 13 shows the similarity score of algorithms for the formal models MDP and MA. We identify three clusters for these algorithms. Cluster one contains algorithms 5 to 30. It contains explicit algorithms and algorithms that use both explicit and symbolic. The algorithms overlap with the second cluster of algorithms for DTMC and CTMC. However, algorithm `S abs` is added and algorithms `S sparse bis VI` and `S sparse bis TVI` are removed. Cluster two contains algorithms 33 to 38. These are algorithms that only use symbolic and algorithms that use both explicit and symbolic from Storm. Cluster three contains the algorithms 25 to 28, 39 and 40. These are from Prism and use the same techniques. The algorithm identified different clusters. The most significant difference in the clustering from the algorithm is that the explicit algorithms from Prism are together with the symbolic algorithm. The other differences can be seen in Appendix A.3.

Using the similarity metric, we identified clusters of algorithms. The clusters were roughly split along the techniques used in the engines. The solution method has an influence on the similarity in the exceptional case. The tool also has some influence.

Table 12. Similarity of algorithms on DTMC and CTMC benchmark instances

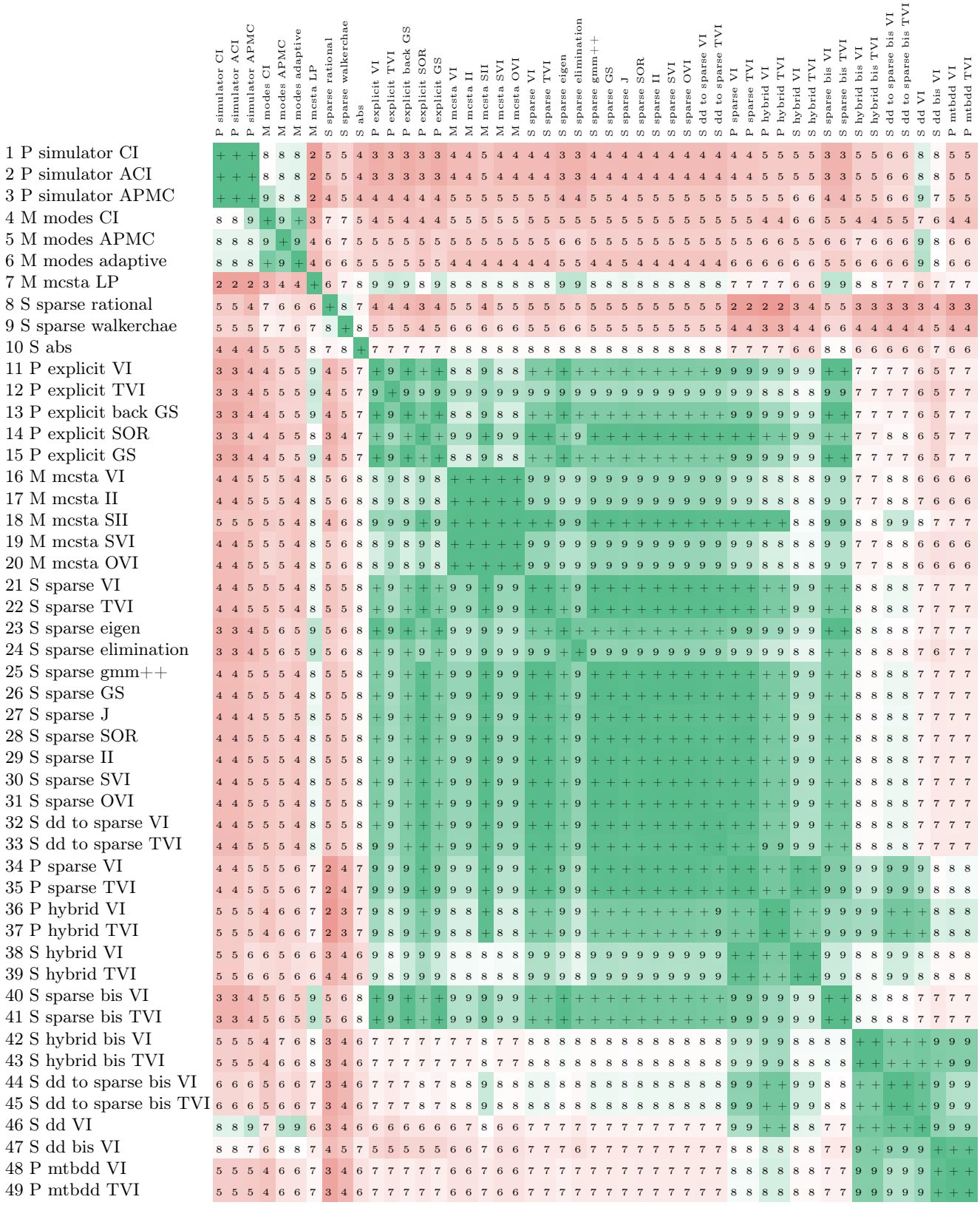
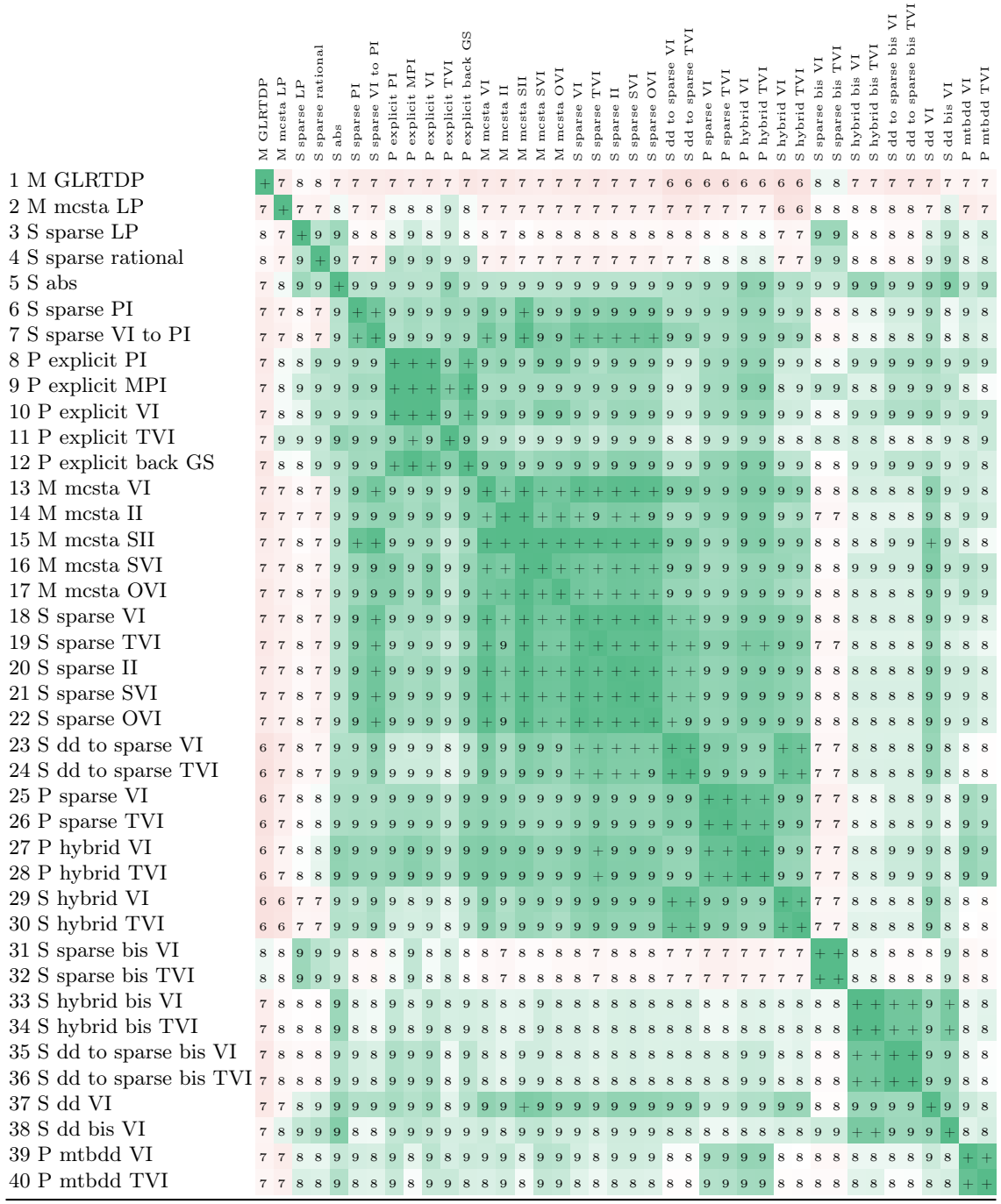


Table 13. Similarity of algorithms on MDP and MA benchmark instances



Wins The win metric is shown in Tables 14 and 15. Table 14 shows the win metric for algorithms for DTMC and CTMC and Table 15 shows it for algorithms for MDP and MA. The win metric is between an algorithm and an opponent algorithm. A win is when an algorithm is faster on one benchmark instance compared to the opponent algorithm. The win metric is based on the wins and losses. A 10 means the algorithm always wins and a 0 means it always loses. The draws are not taken into account.

The tables have the same structure as the similarity metric tables. The cells are coloured based on the metric. 0 is red, 5 is white and 10 is green. Any colour in between is a mixture. The 10 is shown with +, because of spacing. The algorithms have been ordered by the total number of wins, which means that on average faster algorithms are at the top. The cell contains a + if the algorithm to the left always wins from the algorithm at the top. The cells on the diagonal are all zeros because an algorithm will never win from itself. The comparison to itself does not count towards the numbers to the right of the table. These are the total number of wins, draws, losses and best. The column "best" contains the number of benchmark instances for which the algorithm had the best time.

We investigate the top 50% of algorithms. Algorithms 1 to 24 in Table 14 are almost all algorithms in Storm. Only two algorithms are from Prism. Algorithms 1 to 20 in Table 15 are 5 from the Modest Toolset, 1 from Prism and 14 from Storm. From these algorithms, Storm occupies the top 7 spots.

Can we conclude that the algorithms in the top 50% are the fastest? No, the conclusion is more nuanced. The algorithm with the most wins for DTMC and CTMC loses 19% of the time and the algorithm with the most wins for MDP and MA loses 10% of the time. Furthermore, the bottom 50% has more best times. The top 50% of DTMC and CTMC algorithms have 5 best times. However, the bottom 50% of algorithms have 23 best times. The same can be seen for MDP and MA algorithms. The top 50% of algorithms collectively have 33 best times compared to the bottom 50% of algorithms having 48 best times.

There are two other reasons why an algorithm might be lower in the table. The first reason is that the precision guarantees on the result as described in Section 2.2 varies between the algorithms, which means these algorithms are calculating different things. This can either positively or negatively affect the speed. We do not include the precision guarantee in the comparison because we want to focus on the speed of the algorithms. The second reason is the support of formal models and properties. `M mcsta SII` has a total of 1488. Even if it would have won everything it would still not be above `S sparse VI` with 1650 wins.

We looked at the top 50% of algorithms. We observed that 36 of the 44 top 50% of algorithms were from Storm. The other algorithms were from the Modest Toolset and a couple from Prism. The top 50% of algorithms are not always the fastest. The top 50% of algorithms still lose on some of the benchmark instances and the bottom 50% has more best times.

Table 14. Speed comparison of algorithms on DTMC and CTMC benchmark instances

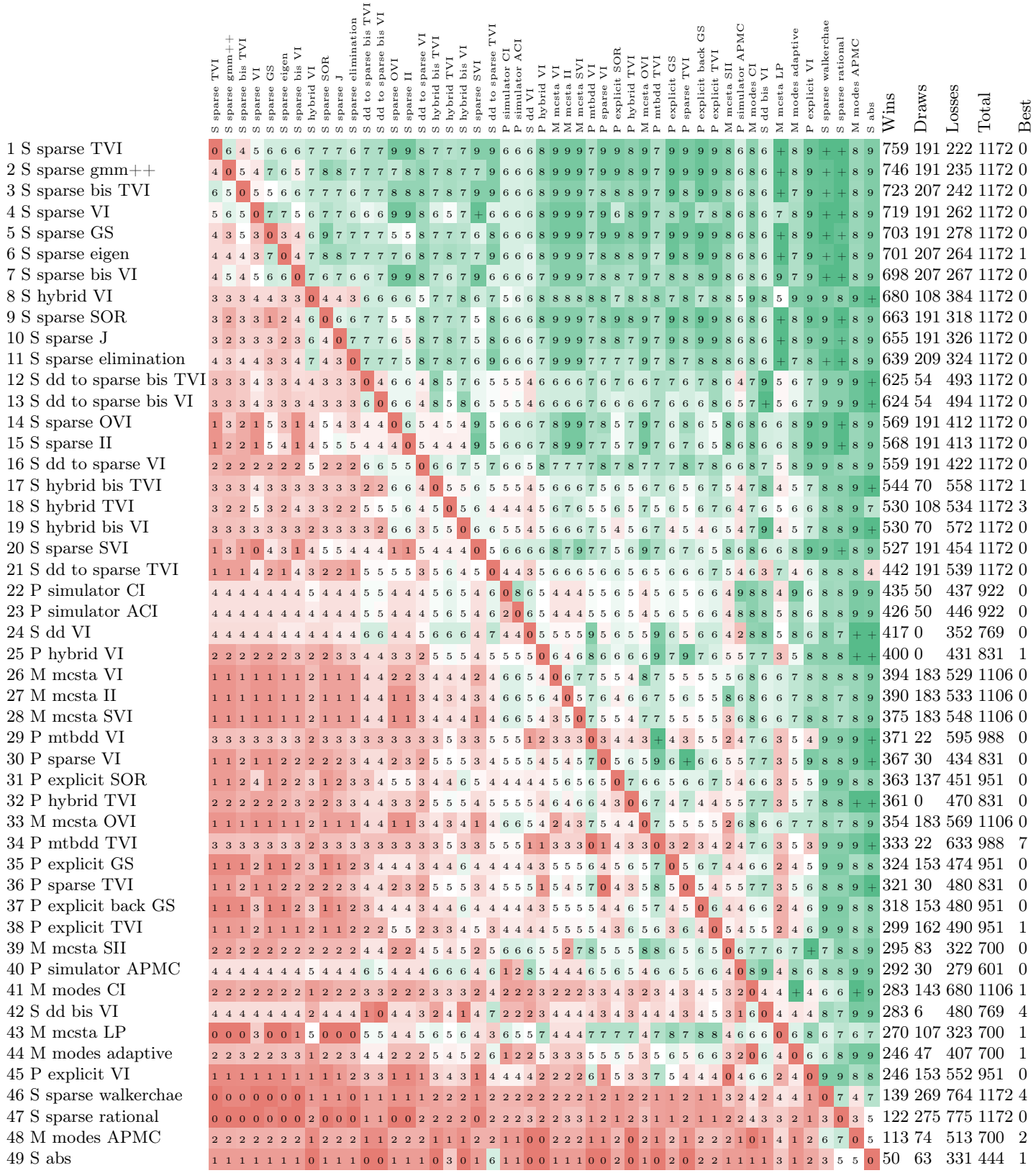


Table 15. Speed comparison of algorithms on MDP and MA benchmark instances

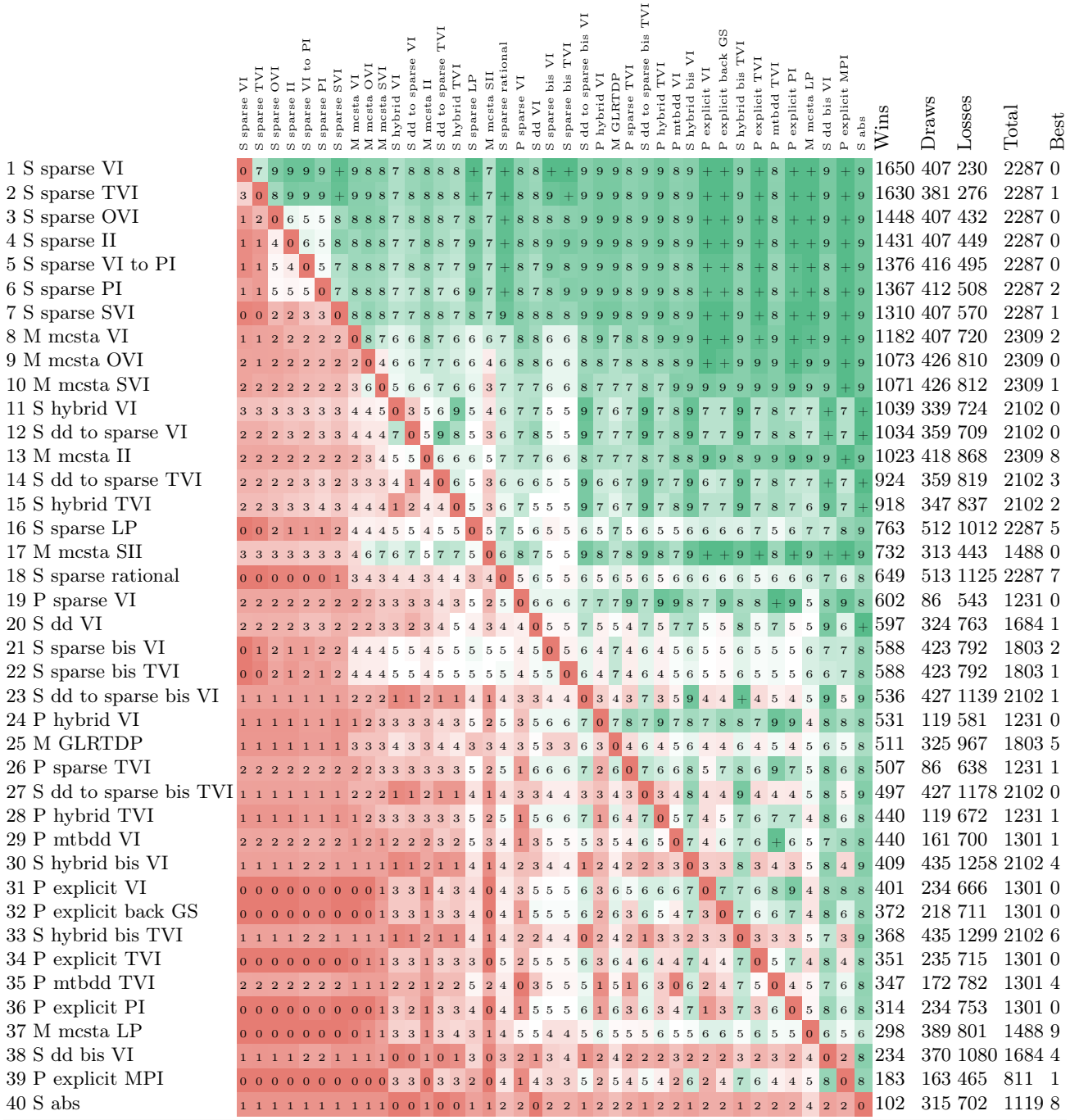


Table 16. Similarity of algorithms on PTA benchmark instances

	P back reachability	P stochastic games	P digital clocks	M mcsta VI	M mcsta II	M mcsta SII	M mcsta SVI	M mcsta OVI	M mcsta LP
1 P back reachability	+	6	+	+	+	+	+	+	+
2 P stochastic games	6	+	+	+	+	+	+	+	9
3 P digital clocks	+	+	+	+	+	+	+	+	9
4 M mcsta VI	+	+	+	+	9	9	9	9	9
5 M mcsta II	+	+	+	9	+	+	9	9	+
6 M mcsta SII	+	+	+	9	+	+	+	+	9
7 M mcsta SVI	+	+	+	9	9	+	+	+	+
8 M mcsta OVI	+	+	+	9	9	+	+	+	+
9 M mcsta LP	+	9	9	9	+	9	+	+	+

Table 17. Speed comparison of algorithms on PTA benchmark instances

	M mcsta VI	M mcsta OVI	M mcsta SVI	M mcsta II	M mcsta LP	M mcsta SII	P stochastic games	P digital clocks	P back reachability	Wins	Draws	Losses	Total	Best
1 M mcsta VI	0	9	8	9	9	8	7	7	+	84	3	15	102	0
2 M mcsta OVI	1	0	5	8	8	8	7	7	+	59	3	40	102	1
3 M mcsta SVI	2	5	0	8	6	7	7	7	+	55	3	44	102	3
4 M mcsta II	1	2	2	0	5	7	7	7	+	35	3	64	102	4
5 M mcsta LP	1	2	4	5	0	7	7	7	+	35	0	52	87	4
6 M mcsta SII	2	2	3	3	3	0	7	7	+	26	0	61	87	6
7 P stochastic games	3	3	3	3	3	3	0	3	6	10	0	16	26	5
8 P digital clocks	3	3	3	3	3	3	7	0	+	9	0	13	22	0
9 P back reachability	0	0	0	0	0	0	4	0	0	2	0	10	12	1

PTA There were a total of 25 PTA benchmark instances. However, Prism and the Modest Toolset have only 3 benchmark instances in common. Table 16 shows that the algorithms are similar. Table 17 shows that the Modest Toolset is faster. However, both of these conclusions are based on only 3 benchmark instances.

6.2 Which benchmark instances have similar performance according to the similarity metric from SAT?

Tables 18, 19, 20, 21 and 22 show the similarity in performance for the benchmark instances as defined in Section 4.5. The benchmark instances have been split by formal model. The DTMC and CTMC benchmark instances are in Table 18. MDP benchmark instances are split further by properties because of the number of MDP benchmark instances. MDP benchmark instances with reachability probability property are in Table 19. Table 20 shows MDP benchmark instances with expected rewards property. Table 21 shows MA benchmark instances. The PTA benchmark instances are in Table 22.

The clusters are identified manually, after which they are verified using a clustering algorithm. We observed that benchmark instances with the same average wall-clock time are similar. Therefore, to identify the clusters, we ordered the benchmark instances by average wall-clock time.

As for the tables in the previous section, the cells are coloured based on the similarity metric. A cell is coloured red for 0, gradually turning white as we approach 8, after which it turns green for 10 and we omit decimals. The number of average states is used in the analysis of the clusters.

All the tables have a cluster that starts at the first benchmark instance. The clusters are benchmark instances 1 to 19 for Table 18, benchmark instances 1 to 21 for Table 19, benchmark instances 1 to 11 for Table 20, benchmark instances 1 to 21 for Table 21 and benchmark instances 1 to 20 for Table 16. The average wall-clock time in the clusters ranges from 1 to 325 seconds, which seems strange. How can a benchmark instance with an average wall-clock time of 1 second be similar to one with an average time of 325 seconds? The reason is that we are comparing absolute performance instead of relative performance. If the wall-clock times always differ by 360 seconds, the similarity metric is 8. This is how a benchmark instance with an average wall-clock time of 325 seconds can be in the same cluster as a benchmark instance with an average wall-clock time of 1 second.

Tables 18, 19 and 21 have a cluster with an average wall-clock time of above 1168 seconds. These are benchmark instances 23 to 29 for Table 18, benchmark instances 31 to 39 for 19 and 27 to 31 for 21. For this cluster, most algorithms timed out or gave an error and the number of average states is much higher than in the previous cluster.

There are some things we would like to point out. Tables 18 and 19 have a third cluster. These are benchmark instances 11 to 18 in Table 18 and benchmark instances 22 to 27 in Table 19. In Tables 19 and 21 there are high similarity scores between benchmark instances with a big difference in average wall-clock time. These are benchmark instances 10, 11 and 13 for Table 19 and benchmark instances 4 to 8 for Table 21. Those benchmark instances are not supported by all algorithms. The benchmark instances have a high similarity score because the similarity metric only compares the supported algorithms. Table 22 has blank cells. This means that no algorithm was able to solve both benchmark instances.

The clusters from the clustering algorithms differ slightly because not all algorithms are in a cluster. The clusters from the algorithms contain the algorithms which were not in a cluster. The clusters can be seen in Appendix A.4.

The clusters from the clustering algorithms differ slightly because we did not put all algorithms in a cluster. The algorithms not in a cluster are in a cluster for the clusters from the algorithm. The clusters from the algorithm can be seen in Appendix A.4.

Using the similarity metric, we found clusters of benchmark instances. The clusters with a high average time had more states compared to clusters with a lower average time. The clusters with a low average time contained benchmark instances with an average wall-clock time of 1 second and 325 seconds. Therefore, we think the similarity metric incorrectly assesses the similarity of benchmark instances. We recommend changing the metric or using a completely different metric in future work.

This immediately raises the question of whether the similarity analysis for algorithms is correct. This is not the case because the spread in measurements is different. Remember that the metric de-emphasises fast measurements. A fast benchmark instance is quickly solved by all algorithms, which are all fast measurements. These will all be classified as similar by the metric.

A fast algorithm does not quickly solve all benchmark instances. A slow benchmark instance will still be slow for a fast algorithm, which results in a mix of fast and slow measurements.

The similarity metric represents the similarity for algorithms because there is a mix of fast and slow measurements. The slower measurements have a more significant effect compared to the fast measurements.

Table 18. Similarity of DTMC and CTMC benchmark instances

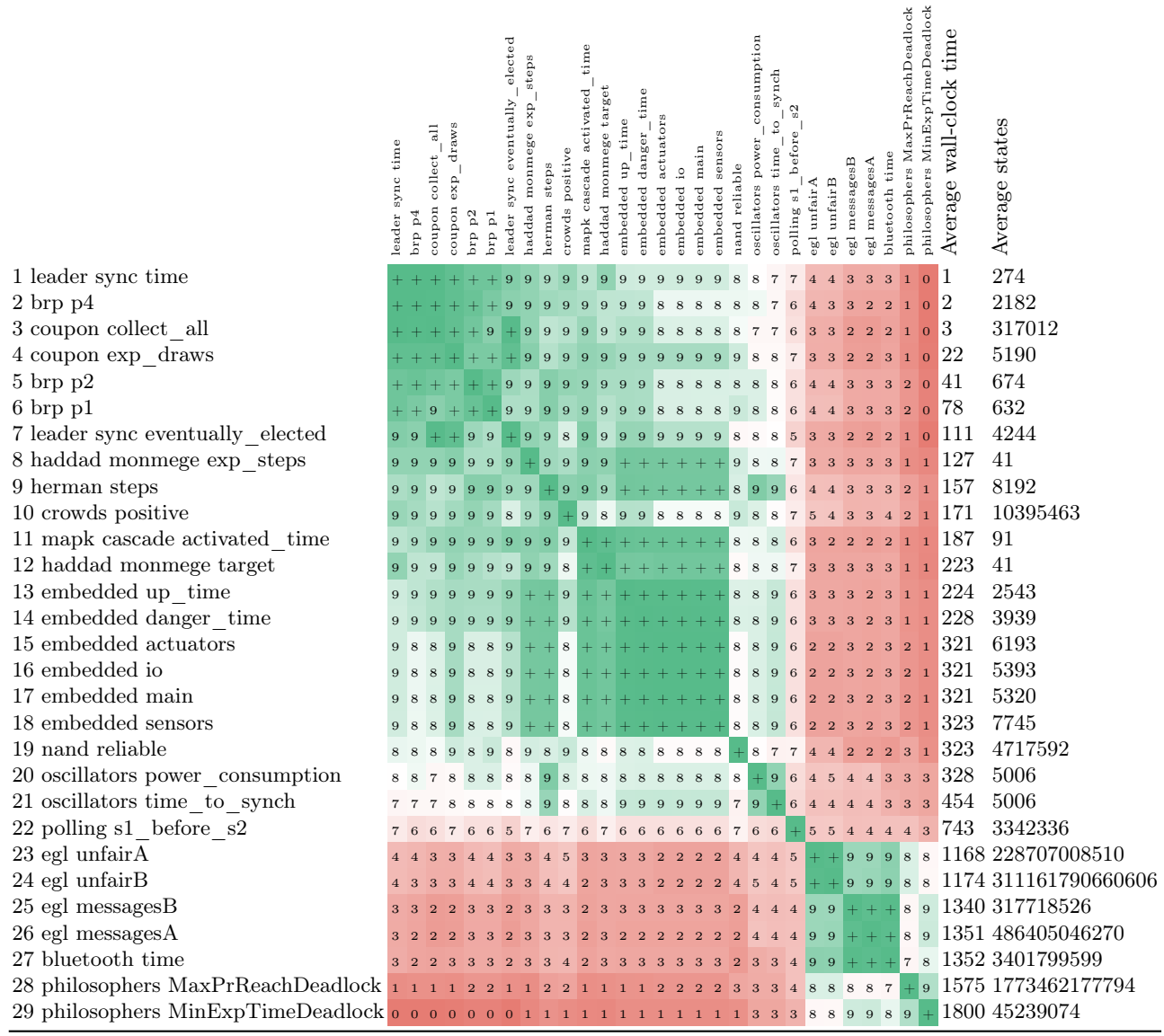


Table 19. Similarity of MDP reachability probability benchmark instances

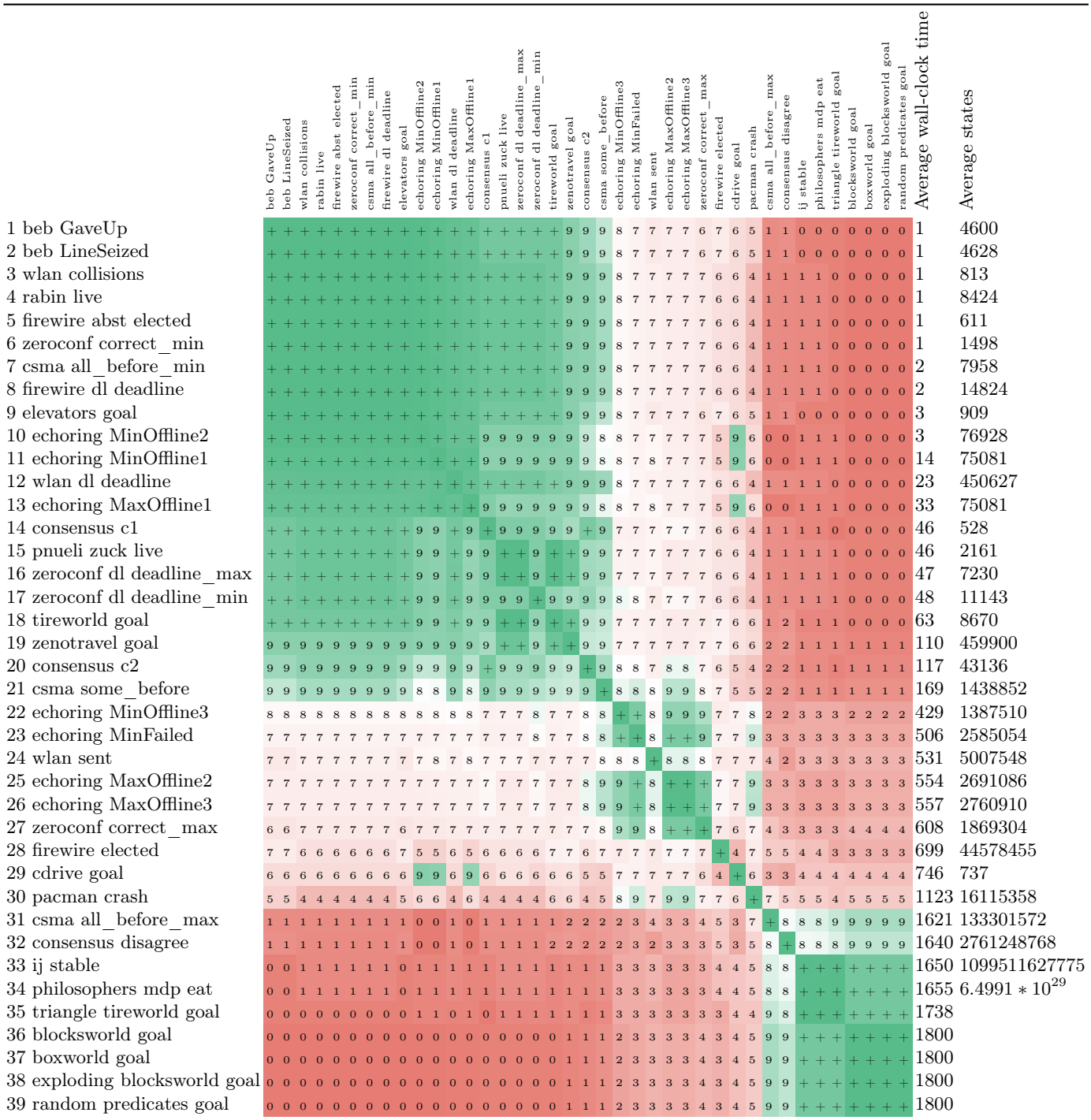


Table 20. Similarity of MDP expected rewards benchmark instances

	firewire abst rounds	firewire abst time_min	firewire abst time_max	wlan time_min	wlan cost_min	firewire time_sending	wlan num_collisions	csma time_min	resource gathering expsteps	firewire time_min	firewire time_max	eajs ExpUtil	wlan cost_max	consensus steps_max	wlan time_max	csma time_max	consensus steps_min	Average wall-clock time	Average states
1 firewire abst rounds	+	+	+	+	+	+	+	+	9	9	9	8	8	7	4	2	2	1	776
2 firewire abst time_min	+	+	+	+	+	+	+	+	9	9	9	8	8	7	4	2	2	1	776
3 firewire abst time_max	+	+	+	+	+	+	+	+	9	9	9	8	8	7	4	2	2	1	611
4 wlan time_min	+	+	+	+	+	+	+	+	9	9	9	8	8	7	4	2	2	5	28480
5 wlan cost_min	+	+	+	+	+	+	+	+	9	9	9	8	8	7	4	2	2	6	28480
6 firewire time_sending	+	+	+	+	+	+	+	9	9	9	9	8	8	7	4	1	2	6	4093
7 wlan num_collisions	+	+	+	+	+	+	+	+	9	9	9	8	8	8	5	2	2	52	8625
8 csma time_min	+	+	+	+	+	9	+	+	9	9	9	8	8	8	5	2	2	69	36850
9 resource gathering expsteps	9	9	9	9	9	9	9	9	+	8	8	8	7	7	5	2	3	139	1
10 firewire time_min	9	9	9	9	9	9	9	9	8	+	9	7	9	8	5	3	3	163	1078419
11 firewire time_max	9	9	9	9	9	9	9	9	8	9	+	7	9	8	5	3	3	178	83030
12 eajs ExpUtil	8	8	8	8	8	8	8	8	8	7	7	+	6	7	5	3	2	298	12828
13 wlan cost_max	8	8	8	8	8	8	8	8	7	9	9	6	+	8	6	4	3	355	345000
14 consensus steps_max	7	7	7	7	7	7	8	8	7	8	8	7	8	+	6	4	4	492	1258240
15 wlan time_max	4	4	4	4	4	4	5	5	5	5	5	5	6	6	+	7	4	1028	5007548
16 csma time_max	2	2	2	2	2	1	2	2	2	3	3	3	4	4	7	+	7	1498	84856004
17 consensus steps_min	2	2	2	2	2	2	2	2	3	3	3	2	3	4	4	7	+	1512	61018112

Table 21. Similarity of MA benchmark instances

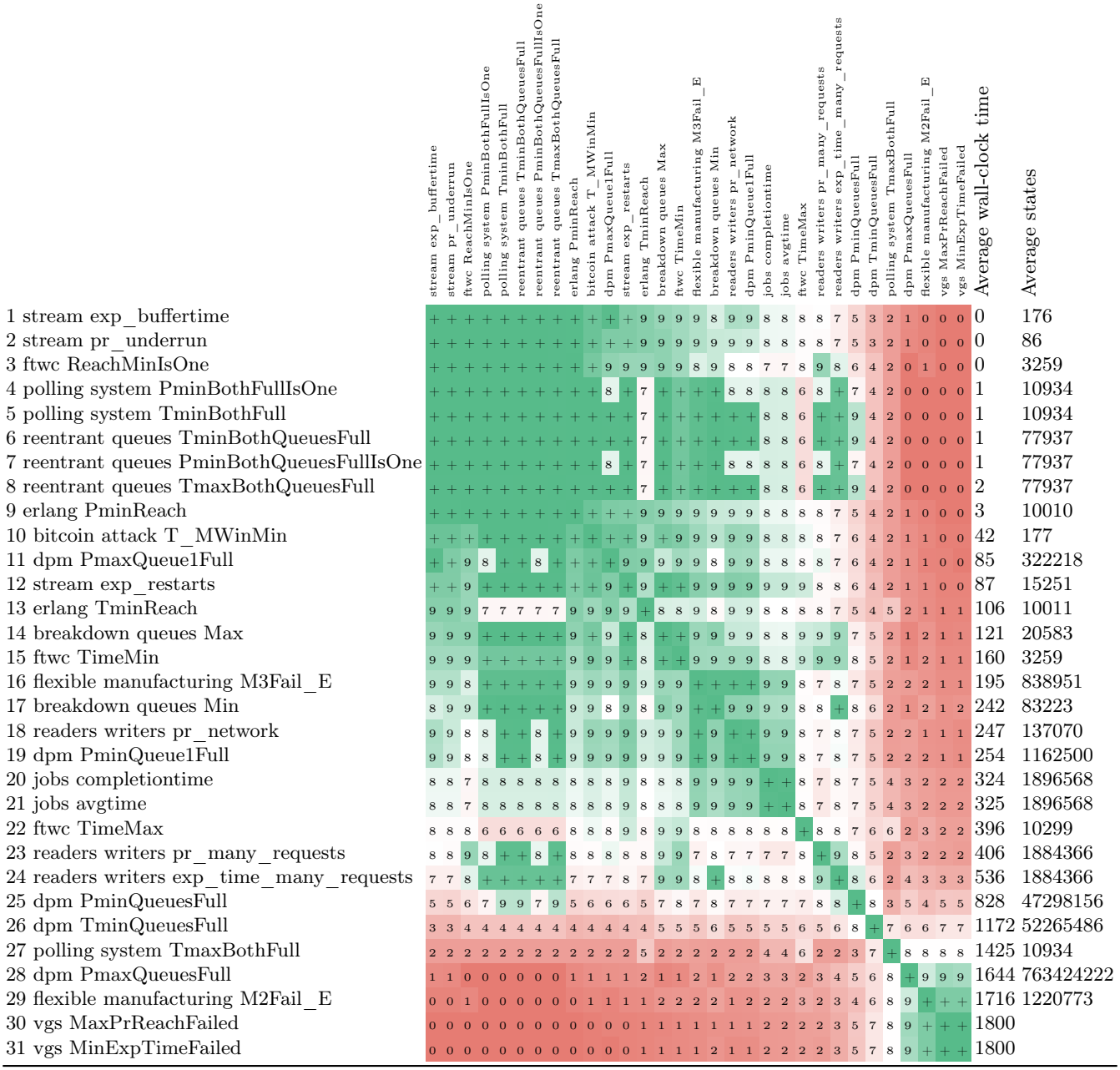
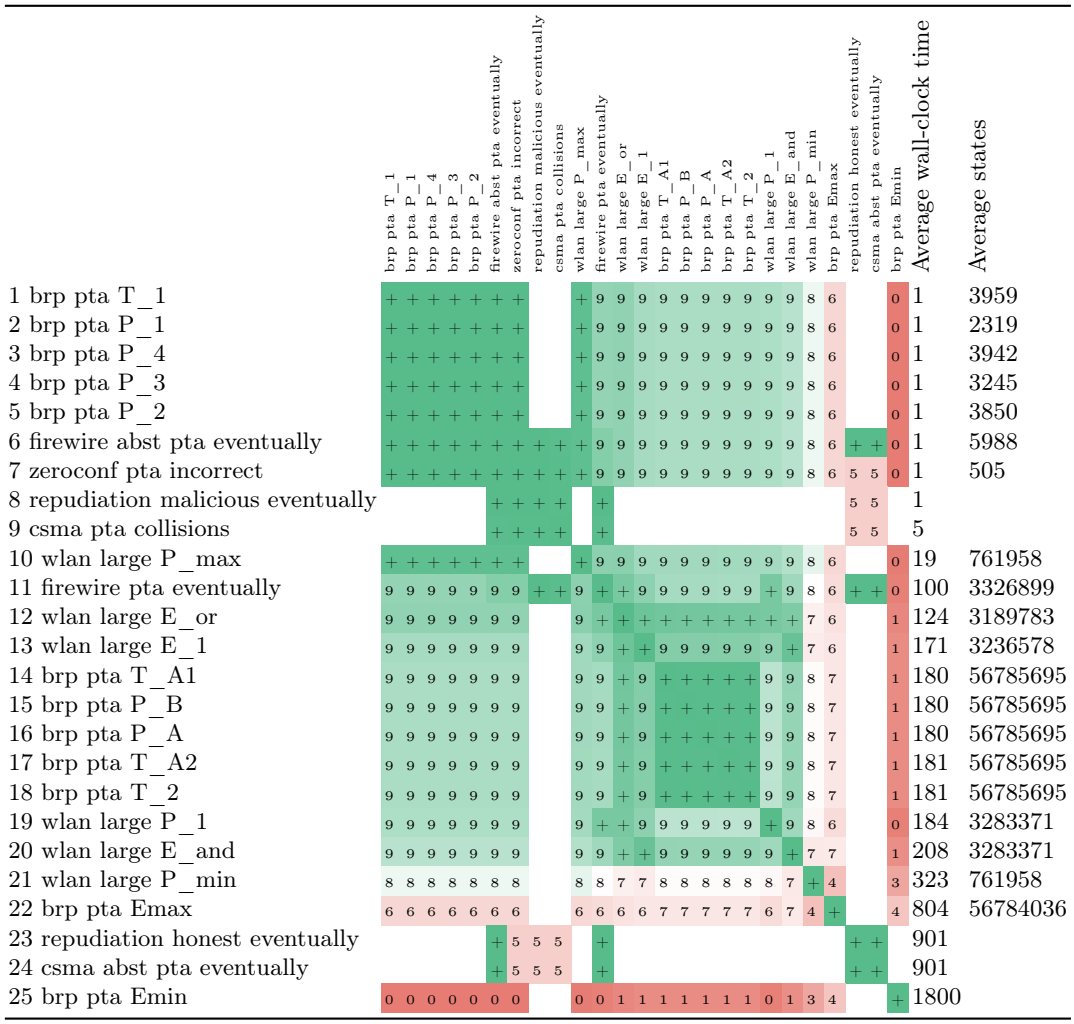


Table 22. Similarity of PTA benchmark instances



6.3 What statistical associations exist between the characteristics of a benchmark instance and the performance of an algorithm?

Tables 23, 24 and 25 show the Pearson correlation between performance metric and characteristic. The Pearson correlation (r-value) conveys how linearly correlated the data is. A correlation of 1 means that the data points form a line and a correlation of 0 means the data points are scattered. The p-value is how likely the correlation was due to random chance. To decrease the chance of finding correlations due to random chance, we only discuss correlations with a p-value of less than 0.05. The formula is the time in seconds where x is the number of the characteristics in millions. We only used the measurements, for which the algorithm did not time out or threw an error.

Table 23 shows the correlations for the state space time. There is no data for algorithms 1 to 7, 56 and 57 because these do not show the state space time. Algorithms 1 to 37 strongly correlates with states, transitions and branches. These are explicit algorithms. Algorithms 46 and 47 are also explicit algorithms; however, they only have a weak correlation with transitions. Algorithms 38 to 45 and 48 to 55 have a too high p-value, which means we can not say anything about it. These algorithms use symbolic or a combination of symbolic and explicit techniques. Algorithm `P digital clocks` has a strong correlation with the characteristics.

Table 24 shows the correlations for property time. Algorithm 56 had only one benchmark instance for which characteristics were available. This means we can not use the Pearson correlation. Algorithm 3 has a moderate correlation with the characteristics, which is one of the six statistical algorithms. Statistical algorithms 1, 2 and 4 to 6 have a too high p-value. Noteworthy is that the formula of algorithm 3 has a slope of zero. This would mean that it is unaffected by the characteristics which we would not expect. Further research is needed to investigate this. Algorithm 7 has a too high p-value. This is a partial state space algorithm. Algorithms 13 to 22, 24 to 28 and 30 to 37, 46 and 47 have a weak, moderate or strong correlation to one or more characteristics. These are explicit algorithms. From the explicit algorithms 8 to 12,23 and 29 have a too high p-value. Algorithms 38 and 39, which use both symbolic and explicit techniques, have a moderate or strong correlation with the characteristics. Algorithms 40 to 45 and 48 to 55 use a combination of symbolic and explicit or only symbolic. These have a too high p-value. Algorithm 58 has a strong correlation. Algorithm 57 has a too high p-value.

Table 25 shows the wall-clock time, which is a combination of parse time, state space time, property time and startup time. One of the statistical algorithms has a moderate correlation. The others have a too high p-value. Algorithm 7, which uses a partial state space, has a weak correlation to states and branches. Algorithms 9,13 to 37, 46 and 47 have a strong and some a moderate correlation to the characteristics. These are the explicit algorithms. Algorithms 8, 10, 11 and 12, which are also explicit, did correlate. As for the previous tables, the algorithms using symbolic or a combination of symbolic and explicit have a too high p-value. Algorithm 57 has a strong negative correlation, which is different from what we expected. We think the randomness of the startup time caused the negative correlation. The algorithm solved the benchmark instances in under two seconds. This means that the startup time represents a significant portion of the wall-clock time.

Most of the explicit algorithms correlate with state, transitions and branches. This is most pronounced for the state space time. Only one of the statistical algorithms had a correlation to the characteristics. Algorithms using symbolic or a combination of symbolic and explicit had a too high p-value except for Algorithms 38 and 39 for the property time.

Table 23. Statistical analysis for state space time

	State space time								
	States			Transitions			Branches		
	r-value	p-value	formula	r-value	p-value	formula	r-value	p-value	formula
1 P simulator CI	No data			No data			No data		
2 P simulator ACI	No data			No data			No data		
3 P simulator APMC	No data			No data			No data		
4 M modes CI	No data			No data			No data		
5 M modes APMC	No data			No data			No data		
6 M modes adaptive	No data			No data			No data		
7 M GLRTDP	No data			No data			No data		
8 M mcsta LP	0.99	0.00	$2.98x + 1.23$	1.0	0.00	$2.66x + 0.91$	1.0	0.00	$2.65x + 0.81$
9 S sparse LP	1.0	0.00	$11.34x - 0.7$	1.0	0.00	$5.04x + 0.18$	1.0	0.00	$4.98x - 0.79$
10 S sparse rational	0.77	0.00	$18.95x + 1.5$	0.68	0.00	$10.87x + 1.73$	0.73	0.00	$7.25x + 1.68$
11 S sparse walkerchae	1.0	0.00	$2.37x + 0.02$	1.0	0.00	$2.37x + 0.02$	1.0	0.00	$1.36x + 0.02$
12 S abs	0.93	0.00	$0.44x + 1.22$	0.93	0.00	$0.2x + 1.42$	0.89	0.00	$0.18x + 1.28$
13 S sparse PI	0.92	0.00	$11.84x + 2.79$	0.9	0.00	$5.35x + 6.62$	0.92	0.00	$5.36x + 3.69$
14 S sparse VI to PI	0.8	0.00	$6.42x + 7.88$	0.9	0.00	$4.59x + 6.58$	0.82	0.00	$3.19x + 7.51$
15 P explicit PI	1.0	0.00	$7.48x + 0.28$	0.93	0.00	$4.51x + 0.33$	0.97	0.00	$2.98x + 0.44$
16 P explicit MPI	1.0	0.00	$7.72x + 0.11$	0.99	0.00	$5.79x + 0.04$	1.0	0.00	$3.41x + 0.31$
17 P explicit VI	0.93	0.00	$11.37x - 1.12$	0.87	0.00	$9.49x - 1.59$	0.99	0.00	$3.74x - 0.49$
18 P explicit TVI	0.92	0.00	$11.37x - 0.98$	0.85	0.00	$9.38x - 1.25$	0.95	0.00	$3.13x - 0.2$
19 P explicit back GS	0.93	0.00	$11.24x - 1.19$	0.86	0.00	$9.24x - 1.59$	0.99	0.00	$3.82x - 0.52$
20 P explicit SOR	0.93	0.00	$12.91x - 0.2$	0.93	0.00	$12.91x - 0.2$	0.95	0.00	$3.19x - 0.74$
21 P explicit GS	0.94	0.00	$12.35x - 1.08$	0.94	0.00	$12.35x - 1.08$	0.99	0.00	$3.86x - 0.64$
22 M mcsta VI	0.76	0.00	$3.18x + 9.15$	0.77	0.00	$2.57x + 8.78$	0.78	0.00	$2.24x + 7.43$
23 M mcsta II	0.75	0.00	$3.43x + 9.61$	0.76	0.00	$2.71x + 9.28$	0.77	0.00	$2.49x + 7.53$
24 M mcsta SII	0.93	0.00	$3.55x + 4.19$	0.93	0.00	$2.78x + 4.42$	0.93	0.00	$2.51x + 2.61$
25 M mcsta SVI	0.78	0.00	$3.36x + 8.98$	0.79	0.00	$2.7x + 8.74$	0.8	0.00	$2.35x + 7.38$
26 M mcsta OVI	0.78	0.00	$3.39x + 9.16$	0.79	0.00	$2.72x + 8.9$	0.8	0.00	$2.37x + 7.54$
27 S sparse VI	0.8	0.00	$6.23x + 6.05$	0.89	0.00	$4.48x + 5.29$	0.81	0.00	$3.0x + 4.79$
28 S sparse TVI	0.91	0.00	$7.72x + 3.69$	0.9	0.00	$5.89x + 3.24$	0.89	0.00	$3.94x + 1.74$
29 S sparse eigen	0.98	0.00	$5.6x + 0.31$	0.98	0.00	$5.6x + 0.31$	1.0	0.00	$1.69x + 0.64$
30 S sparse elimination	0.98	0.00	$5.74x - 0.09$	0.98	0.00	$5.74x - 0.09$	1.0	0.00	$1.72x + 0.43$
31 S sparse gmm++	0.89	0.00	$6.42x + 1.26$	0.89	0.00	$6.42x + 1.26$	1.0	0.00	$1.75x + 0.6$
32 S sparse GS	0.88	0.00	$6.47x + 1.31$	0.88	0.00	$6.47x + 1.31$	1.0	0.00	$1.78x + 0.59$
33 S sparse J	0.89	0.00	$6.41x + 1.27$	0.89	0.00	$6.41x + 1.27$	1.0	0.00	$1.75x + 0.6$
34 S sparse SOR	0.89	0.00	$6.35x + 1.26$	0.89	0.00	$6.35x + 1.26$	1.0	0.00	$1.74x + 0.59$
35 S sparse II	0.81	0.00	$6.46x + 6.0$	0.9	0.00	$4.62x + 5.28$	0.82	0.00	$3.1x + 4.73$
36 S sparse SVI	0.82	0.00	$6.65x + 5.89$	0.91	0.00	$4.75x + 5.18$	0.83	0.00	$3.2x + 4.57$
37 S sparse OVI	0.81	0.00	$6.42x + 6.03$	0.9	0.00	$4.59x + 5.31$	0.82	0.00	$3.09x + 4.73$
38 S dd to sparse VI	-0.03	0.77		-0.03	0.77		-0.04	0.75	
39 S dd to sparse TVI	-0.03	0.81		-0.03	0.81		-0.03	0.79	
40 P sparse VI	0.26	0.06		-0.02	0.88		-0.02	0.87	
41 P sparse TVI	0.25	0.06		-0.02	0.87		-0.02	0.86	
42 P hybrid VI	0.26	0.06		-0.02	0.88		-0.02	0.87	
43 P hybrid TVI	0.25	0.06		-0.02	0.88		-0.02	0.87	
44 S hybrid VI	-0.01	0.92		-0.01	0.92		-0.01	0.92	
45 S hybrid TVI	0.11	0.33		0.11	0.33		0.11	0.34	
46 S sparse bis VI	0.27	0.06		0.31	0.02	$6.89x + 11.08$	0.26	0.06	
47 S sparse bis TVI	0.27	0.05		0.32	0.02	$6.88x + 11.01$	0.27	0.05	
48 S hybrid bis VI	0.04	0.72		0.04	0.72		0.04	0.72	
49 S hybrid bis TVI	0.04	0.74		0.04	0.74		0.04	0.74	
50 S dd to sparse bis VI	0.05	0.67		0.05	0.67		0.05	0.67	
51 S dd to sparse bis TVI	0.05	0.67		0.05	0.67		0.05	0.67	
52 S dd VI	0.01	0.95		0.01	0.95		0.01	0.95	
53 S dd bis VI	0.26	0.07		0.26	0.07		0.26	0.07	
54 P mtbdd VI	0.25	0.07		-0.02	0.89		-0.02	0.88	
55 P mtbdd TVI	0.25	0.08		-0.02	0.90		-0.02	0.89	
56 P back reachability	No data			No data			No data		
57 P stochastic games	No data			No data			No data		
58 P digital clocks	1.0	0.01	$107.68x - 0.05$	1.0	0.03	$78.0x - 0.05$	1.0	0.03	$78.09x - 0.06$

Table 24. Statistical analysis for property time

	Property time								
	States			Transitions			Branches		
	r-value	p-value	formula	r-value	p-value	formula	r-value	p-value	formula
1 P simulator CI	-0.06	0.82		-0.06	0.82		-0.06	0.82	
2 P simulator ACI	-0.07	0.80		-0.07	0.80		-0.07	0.80	
3 P simulator APMC	0.68	0.02	$0.0x + 0.97$	0.68	0.02	$0.0x + 0.97$	0.68	0.02	$0.0x + 0.97$
4 M modes CI	0.15	0.69		0.15	0.69		0.55	0.10	
5 M modes APMC	0.23	0.62		0.23	0.62		0.0	1.00	
6 M modes adaptive	0.38	0.40		0.38	0.40		0.14	0.77	
7 M GLRTDP	0.34	0.06		0.16	0.39		0.15	0.44	
8 M mcsta LP	-0.09	0.54		-0.08	0.59		-0.07	0.61	
9 S sparse LP	0.02	0.89		0.02	0.88		0.02	0.89	
10 S sparse rational	-0.05	0.75		-0.05	0.76		-0.05	0.75	
11 S sparse walkerchae	-0.24	0.53		-0.24	0.53		-0.24	0.53	
12 S abs	0.14	0.49		0.17	0.41		0.16	0.45	
13 S sparse PI	0.38	0.00	$1.42x + 3.91$	0.27	0.03	$0.46x + 4.9$	0.31	0.01	$0.53x + 4.4$
14 S sparse VI to PI	0.84	0.00	$3.66x + 0.16$	0.65	0.00	$1.81x + 3.24$	0.82	0.00	$1.72x + 0.55$
15 P explicit PI	0.93	0.00	$115.0x - 10.72$	0.9	0.00	$71.81x - 11.73$	0.96	0.00	$48.94x - 11.55$
16 P explicit MPI	0.95	0.00	$117.13x - 17.73$	0.92	0.00	$86.12x - 17.56$	0.97	0.00	$52.66x - 16.06$
17 P explicit VI	0.39	0.01	$21.57x + 13.91$	0.54	0.00	$26.91x + 5.68$	0.3	0.05	
18 P explicit TVI	0.67	0.00	$38.82x + 4.16$	0.75	0.00	$40.08x - 0.24$	0.58	0.00	$10.66x + 11.96$
19 P explicit back GS	0.5	0.00	$33.73x + 12.09$	0.6	0.00	$35.91x + 3.11$	0.37	0.02	$7.9x + 21.34$
20 P explicit SOR	0.34	0.19		0.34	0.19		0.7	0.00	$1.39x + 0.22$
21 P explicit GS	0.77	0.00	$1.89x + 0.83$	0.77	0.00	$1.89x + 0.83$	0.6	0.01	$0.44x + 1.35$
22 M mcsta VI	0.35	0.00	$1.76x + 13.35$	0.4	0.00	$1.61x + 11.93$	0.48	0.00	$1.66x + 9.0$
23 M mcsta II	0.04	0.70		0.1	0.29		0.1	0.29	
24 M mcsta SII	0.26	0.03	$0.58x + 4.23$	0.49	0.00	$0.83x + 1.28$	0.47	0.00	$0.73x + 0.97$
25 M mcsta SVI	0.41	0.00	$2.01x + 8.82$	0.48	0.00	$1.84x + 7.2$	0.56	0.00	$1.85x + 4.24$
26 M mcsta OVI	0.41	0.00	$2.23x + 11.41$	0.45	0.00	$1.96x + 10.12$	0.52	0.00	$1.93x + 7.3$
27 S sparse VI	0.68	0.00	$3.25x - 0.53$	0.54	0.00	$1.64x + 1.61$	0.66	0.00	$1.49x - 0.74$
28 S sparse TVI	0.61	0.00	$2.34x + 0.47$	0.54	0.00	$1.6x + 1.18$	0.63	0.00	$1.26x - 0.58$
29 S sparse eigen	0.11	0.64		0.11	0.64		0.14	0.55	
30 S sparse elimination	0.95	0.00	$2.19x + 0.04$	0.95	0.00	$2.19x + 0.04$	0.99	0.00	$0.68x + 0.2$
31 S sparse gmm++	0.66	0.00	$0.67x + 0.29$	0.66	0.00	$0.67x + 0.29$	0.89	0.00	$0.22x + 0.09$
32 S sparse GS	0.46	0.03	$1.65x + 1.1$	0.46	0.03	$1.65x + 1.1$	0.78	0.00	$0.68x + 0.11$
33 S sparse J	0.34	0.12		0.34	0.12		0.66	0.00	$3.38x + 1.33$
34 S sparse SOR	0.4	0.06		0.4	0.06		0.72	0.00	$1.4x + 0.53$
35 S sparse II	0.57	0.00	$4.6x + 4.44$	0.45	0.00	$2.31x + 7.55$	0.56	0.00	$2.11x + 4.09$
36 S sparse SVI	0.6	0.00	$4.8x + 3.46$	0.47	0.00	$2.41x + 6.73$	0.58	0.00	$2.2x + 3.12$
37 S sparse OVI	0.58	0.00	$1.96x + 3.0$	0.47	0.00	$1.02x + 4.18$	0.57	0.00	$0.91x + 2.8$
38 S dd to sparse VI	0.72	0.00	$2.11x + 3.62$	0.68	0.00	$1.74x + 3.56$	0.72	0.00	$1.09x + 2.84$
39 S dd to sparse TVI	0.73	0.00	$2.45x + 2.19$	0.69	0.00	$2.02x + 2.08$	0.74	0.00	$1.29x + 1.08$
40 P sparse VI	-0.04	0.75		-0.04	0.77		-0.04	0.76	
41 P sparse TVI	-0.05	0.75		-0.04	0.77		-0.04	0.76	
42 P hybrid VI	-0.05	0.74		-0.04	0.76		-0.04	0.75	
43 P hybrid TVI	-0.05	0.74		-0.04	0.77		-0.04	0.76	
44 S hybrid VI	-0.02	0.84		-0.02	0.84		-0.03	0.81	
45 S hybrid TVI	-0.02	0.87		-0.02	0.87		-0.02	0.85	
46 S sparse bis VI	0.39	0.00	$0.5x + 0.2$	0.37	0.01	$0.46x + 0.18$	0.17	0.24	
47 S sparse bis TVI	0.39	0.00	$0.06x + 0.03$	0.37	0.01	$0.06x + 0.03$	0.17	0.24	
48 S hybrid bis VI	-0.02	0.84		-0.02	0.84		-0.02	0.84	
49 S hybrid bis TVI	-0.02	0.84		-0.02	0.84		-0.02	0.84	
50 S dd to sparse bis VI	-0.04	0.75		-0.04	0.75		-0.04	0.75	
51 S dd to sparse bis TVI	-0.03	0.77		-0.03	0.77		-0.03	0.77	
52 S dd VI	-0.06	0.65		-0.06	0.65		-0.06	0.65	
53 S dd bis VI	-0.05	0.70		-0.05	0.70		-0.05	0.70	
54 P mtbdd VI	-0.06	0.66		-0.06	0.67		-0.06	0.66	
55 P mtbdd TVI	-0.06	0.66		-0.06	0.68		-0.06	0.66	
56 P back reachability	No data			No data			No data		
57 P stochastic games	0.96	0.17		0.96	0.18		0.95	0.19	
58 P digital clocks	1.0	0.03	$195.21x - 0.11$	1.0	0.04	$141.36x - 0.12$	1.0	0.05	$141.47x - 0.14$

Table 25. Statistical analysis for wall-clock time

	Wall-clock time								
	States			Transitions			Branches		
	r-value	p-value	formula	r-value	p-value	formula	r-value	p-value	formula
1 P simulator CI	-0.09	0.72		-0.09	0.72		-0.09	0.72	
2 P simulator ACI	-0.09	0.71		-0.09	0.71		-0.09	0.71	
3 P simulator APMC	0.68	0.02	$0.0x + 0.97$	0.68	0.02	$0.0x + 0.97$	0.68	0.02	$0.0x + 0.97$
4 M modes CI	-0.29	0.23		-0.29	0.23		-0.14	0.57	
5 M modes APMC	-0.26	0.38		-0.26	0.38		-0.06	0.85	
6 M modes adaptive	-0.26	0.39		-0.26	0.39		-0.05	0.87	
7 M GLRTDP	0.33	0.03	$19.69x + 308.13$	0.29	0.06		0.31	0.04	$8.51x + 311.0$
8 M mcsta LP	0.14	0.30		0.15	0.27		0.16	0.26	
9 S sparse LP	0.77	0.00	$11.55x + 13.53$	0.77	0.00	$5.14x + 14.41$	0.77	0.00	$5.07x + 13.45$
10 S sparse rational	0.05	0.73		0.04	0.78		0.05	0.76	
11 S sparse walkerchae	-0.24	0.53		-0.24	0.53		-0.24	0.53	
12 S abs	0.3	0.13		0.24	0.23		0.23	0.25	
13 S sparse PI	0.9	0.00	$13.26x + 6.73$	0.85	0.00	$5.82x + 11.55$	0.88	0.00	$5.89x + 8.13$
14 S sparse VI to PI	0.95	0.00	$10.09x + 8.07$	0.94	0.00	$6.41x + 9.86$	0.95	0.00	$4.91x + 8.1$
15 P explicit PI	0.94	0.00	$122.44x - 9.6$	0.9	0.00	$76.29x - 10.56$	0.96	0.00	$51.91x - 10.28$
16 P explicit MPI	0.95	0.00	$124.81x - 16.71$	0.93	0.00	$91.88x - 16.61$	0.97	0.00	$56.06x - 14.84$
17 P explicit VI	0.56	0.00	$32.92x + 13.62$	0.69	0.00	$36.37x + 4.93$	0.48	0.00	$8.85x + 19.0$
18 P explicit TVI	0.76	0.00	$48.33x + 1.74$	0.78	0.00	$44.09x - 3.77$	0.59	0.00	$9.89x + 14.49$
19 P explicit back GS	0.62	0.00	$44.94x + 11.84$	0.71	0.00	$45.12x + 2.48$	0.51	0.00	$11.72x + 21.76$
20 P explicit SOR	0.82	0.00	$15.61x + 4.5$	0.82	0.00	$15.61x + 4.5$	0.99	0.00	$4.57x + 0.58$
21 P explicit GS	0.98	0.00	$14.2x + 0.84$	0.98	0.00	$14.2x + 0.84$	1.0	0.00	$4.29x + 1.8$
22 M mcsta VI	0.66	0.00	$4.98x + 23.06$	0.7	0.00	$4.21x + 21.29$	0.76	0.00	$3.93x + 17.03$
23 M mcsta II	0.36	0.00	$3.86x + 42.05$	0.42	0.00	$3.5x + 39.16$	0.43	0.00	$3.22x + 36.9$
24 M mcsta SII	0.83	0.00	$4.25x + 9.0$	0.92	0.00	$3.69x + 6.39$	0.91	0.00	$3.31x + 4.17$
25 M mcsta SVI	0.7	0.00	$5.44x + 18.4$	0.74	0.00	$4.58x + 16.57$	0.8	0.00	$4.24x + 12.27$
26 M mcsta OVI	0.69	0.00	$5.7x + 21.14$	0.72	0.00	$4.74x + 19.62$	0.77	0.00	$4.35x + 15.44$
27 S sparse VI	0.9	0.00	$9.49x + 5.58$	0.9	0.00	$6.12x + 6.98$	0.9	0.00	$4.49x + 4.11$
28 S sparse TVI	0.95	0.00	$10.06x + 4.23$	0.93	0.00	$7.49x + 4.48$	0.95	0.00	$5.2x + 1.22$
29 S sparse eigen	0.72	0.00	$6.17x + 3.72$	0.72	0.00	$6.17x + 3.72$	0.75	0.00	$1.91x + 3.98$
30 S sparse elimination	0.97	0.00	$7.92x + 0.14$	0.97	0.00	$7.92x + 0.14$	1.0	0.00	$2.4x + 0.81$
31 S sparse gmm++	0.87	0.00	$7.07x + 1.75$	0.87	0.00	$7.07x + 1.75$	0.99	0.00	$1.97x + 0.89$
32 S sparse GS	0.78	0.00	$8.11x + 2.6$	0.78	0.00	$8.11x + 2.6$	0.97	0.00	$2.46x + 0.89$
33 S sparse J	0.51	0.02	$13.45x + 8.61$	0.51	0.02	$13.45x + 8.61$	0.8	0.00	$5.13x + 2.12$
34 S sparse SOR	0.68	0.00	$9.54x + 4.14$	0.68	0.00	$9.54x + 4.14$	0.91	0.00	$3.13x + 1.31$
35 S sparse II	0.85	0.00	$11.06x + 10.5$	0.83	0.00	$6.93x + 12.9$	0.85	0.00	$5.22x + 8.88$
36 S sparse SVI	0.87	0.00	$11.46x + 9.43$	0.84	0.00	$7.16x + 11.98$	0.86	0.00	$5.41x + 7.76$
37 S sparse OVI	0.87	0.00	$8.38x + 9.1$	0.9	0.00	$5.62x + 9.56$	0.88	0.00	$4.0x + 7.61$
38 S dd to sparse VI	-0.03	0.76		-0.03	0.76		-0.04	0.74	
39 S dd to sparse TVI	-0.03	0.79		-0.03	0.79		-0.03	0.76	
40 P sparse VI	0.02	0.88		-0.04	0.75		-0.05	0.74	
41 P sparse TVI	0.03	0.85		-0.04	0.75		-0.05	0.74	
42 P hybrid VI	0.02	0.88		-0.05	0.74		-0.05	0.73	
43 P hybrid TVI	0.03	0.84		-0.04	0.75		-0.05	0.74	
44 S hybrid VI	-0.02	0.83		-0.02	0.83		-0.03	0.81	
45 S hybrid TVI	0.0	0.99		0.0	0.99		0.0	0.97	
46 S sparse bis VI	0.29	0.04	$6.48x + 12.05$	0.33	0.02	$7.35x + 11.35$	0.27	0.05	
47 S sparse bis TVI	0.27	0.05	$6.08x + 11.8$	0.32	0.02	$6.94x + 11.12$	0.27	0.05	
48 S hybrid bis VI	-0.01	0.90		-0.01	0.90		-0.01	0.90	
49 S hybrid bis TVI	-0.02	0.90		-0.02	0.90		-0.02	0.90	
50 S dd to sparse bis VI	-0.01	0.93		-0.01	0.93		-0.01	0.93	
51 S dd to sparse bis TVI	-0.01	0.93		-0.01	0.93		-0.01	0.93	
52 S dd VI	-0.06	0.66		-0.06	0.66		-0.06	0.66	
53 S dd bis VI	-0.01	0.95		-0.01	0.95		-0.01	0.95	
54 P mtbdd VI	0.0	0.98		-0.06	0.66		-0.07	0.64	
55 P mtbdd TVI	0.02	0.91		-0.06	0.66		-0.07	0.65	
56 P back reachability	No data			No data			No data		
57 P stochastic games	-0.98	0.14		-0.98	0.13		-0.98	0.12	
58 P digital clocks	1.0	0.03	$232.12x + 1.11$	1.0	0.04	$168.09x + 1.1$	1.0	0.05	$168.23x + 1.08$

7 Conclusion

Our main research question is "How do the characteristics of a benchmark instance influence the performance of algorithms for quantitative verification?". We answered these questions by performing a benchmark of 58 algorithms and 141 benchmark instances from QVBS [34]. We made an overview of algorithms for quantitative verification.

We built the tool QVAnalyser to run the benchmark. It can measure the performance metrics and characteristics needed for this thesis. Furthermore, the tool can format the results in a table, as seen in Chapter 6. The tool is publicly available[12], which allows future researchers to use and adapt the tool for their research.

For the main research question, we investigated the characteristics: number of states, transitions and branches and the performance metrics: state space time, property time and wall-clock time. Using the Pearson correlation, we observed that the explicit algorithms have a linear relation with the number of states, transitions and branches, especially for the state space time. We did not find linear relations for symbolic, partial state space and statistical model checking algorithms.

Besides the main research question, we investigated the similarity of algorithms and benchmark instances using the similarity metric of the SAT competition [24]. We observed four groups of similar algorithms. These groups roughly match the technique to store the state space, which are explicit, symbolic, partial state space and statistical model checking.

The similarity investigation into benchmark instances was not as fruitful because the metric grouped all quickly solved benchmark instances together. This was caused by the metric using absolute differences. Nevertheless, the metric is useful for the similarity analysis of algorithms because the wall-clock time for an algorithm is distributed along a broader range.

We compared the speed of algorithms against each other based on the win metric. The algorithms from Storm had the highest amount of wins. However, this does not mean that they are the fastest. The algorithm with the most wins still loses 10% of the time. Furthermore, we observed that algorithms with fewer wins had more best times.

This thesis improved the knowledge on the performance of algorithms for quantitative verification. We compared the performance, investigated the similarity and performed a statistical analysis involving characteristics. Furthermore, we built the tool QVAnalyser, which allows for the continuation of this research.

7.1 Limitations and future work

There are three limitations in this study that could be addressed in future work. The first limitation is that we used the default parameters of the algorithms because the parameters were not part of the investigation. However, the default parameter might not result in the fastest wall-clock time. The second limitation is that we have one measurement per algorithm and benchmark instance. This obstructs us from analysing the variation in measurements. We observed that the measurements were stable when we manually measured. However, this was only on a small number of the algorithms and benchmark instances. The third limitation is that the comparison of PTA algorithms was unsuccessful because there were not enough PTA benchmark instances, which both the Modest Toolset and Prism support.

We want to highlight three ways this research can be extended. First, we observed numerous errors because the algorithm was out of memory. The state space explosion is a big problem in the computability of quantitative verification. We recommend investigating memory usage. Second, investigate other characteristics besides the number of states, transitions and branches. We found states, transitions and branches to be a good predictor of the state space time for explicit algorithms. We think it is useful to investigate characteristics for symbolic, partial state space and statistical model checking algorithms. To estimate the property time, we suggest the convergence rate of interval iteration [25]. Third, investigate non-linear correlations like exponential or multivariate relations.

[12] https://github.com/Vescatur/QVAnalyser

References

1. Akoglu, H.: User's guide to correlation coefficients (Aug 2018), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6107969/
2. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods Syst. Des. **15**(1), 7–48 (1999). https://doi.org/10.1023/A:1008739929481
3. Amparore, E.G.: A new greatspn GUI for GSPN editing and CSLTA model checking. In: Norman, G., Sanders, W.H. (eds.) Quantitative Evaluation of Systems - 11th International Conference, QEST 2014, Florence, Italy, September 8-10, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8657, pp. 170–173. Springer (2014). https://doi.org/10.1007/978-3-319-10696-0_13
4. Andova, S., Hermanns, H., Katoen, J.: Discrete-time rewards model-checked. In: Larsen, K.G., Niebert, P. (eds.) Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers. Lecture Notes in Computer Science, vol. 2791, pp. 88–104. Springer (2003). https://doi.org/10.1007/978-3-540-40903-8_8
5. Ashok, P., Daca, P., Kretínský, J., Weininger, M.: Statistical model checking: Black or white? In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12476, pp. 331–349. Springer (2020). https://doi.org/10.1007/978-3-030-61362-4_19
6. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.: Model-checking algorithms for continuous-time markov chains. IEEE Trans. Software Eng. **29**(6), 524–541 (2003). https://doi.org/10.1109/TSE.2003.1205180
7. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
8. Bartocci, E., Beyer, D., Black, P.E., Fedyukovich, G., Garavel, H., Hartmanns, A., Huisman, M., Kordon, F., Nagele, J., Sighireanu, M., Steffen, B., Suda, M., Sutcliffe, G., Weber, T., Yamada, A.: Toolympics 2019: An overview of competitions in formal methods. In: Beyer, D., Huisman, M., Kordon, F., Steffen, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25 Years of TACAS: TOOLympics, Held as Part of ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part III. Lecture Notes in Computer Science, vol. 11429, pp. 3–24. Springer (2019). https://doi.org/10.1007/978-3-030-17502-3_1
9. Bertsekas, D.P.: Dynamic Programming and Optimal Control, vol. I. Athena Scientific, Belmont, MA, USA, 3rd edn. (2005)
10. Beyer, D.: Software verification: 10th comparative evaluation (SV-COMP 2021). In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12652, pp. 401–422. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_24
11. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.: MODEST: A compositional modeling formalism for hard and softly timed systems. IEEE Trans. Software Eng. **32**(10), 812–830 (2006). https://doi.org/10.1109/TSE.2006.104
12. Bonet, B., Geffner, H.: Labeled RTDP: improving the convergence of real-time dynamic programming. In: Giunchiglia, E., Muscettola, N., Nau, D.S. (eds.) Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy. pp. 12–21. AAAI (2003), http://www.aaai.org/Library/ICAPS/2003/icaps03-002.php
13. 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: Cassez, F., Raskin, J. (eds.) Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings. Lecture Notes in Computer Science, vol. 8837, pp. 98–114. Springer (2014). https://doi.org/10.1007/978-3-319-11936-6_8
14. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: An efficient statistical model checker for nondeterminism and rare events. Int. J. Softw. Tools Technol. Transf. **22**(6), 759–780 (2020). https://doi.org/10.1007/s10009-020-00563-2
15. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10206, pp. 151–168 (2017). https://doi.org/10.1007/978-3-662-54580-5_9
16. Budde, C.E., Hartmanns, A., Klauck, M., Kretínský, J., Parker, D., Quatmann, T., Turrini, A., Zhang, Z.: On correctness, precision, and performance in quantitative verification - qcomp 2020

competition report. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 12479, pp. 216–241. Springer (2020). https://doi.org/10.1007/978-3-030-83723-5_15
17. Chae, M., Walker, S.G.: An EM-based iterative method for solving large sparse linear systems. Linear and Multilinear Algebra **68**(1), 45–62 (jul 2018). https://doi.org/10.1080/03081087.2018.1498061
18. Clarke, E.M., Klieber, W., Novácek, M., Zuliani, P.: Model checking and the state explosion problem. In: Meyer, B., Nordio, M. (eds.) Tools for Practical Software Verification, LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures. Lecture Notes in Computer Science, vol. 7682, pp. 1–30. Springer (2011). https://doi.org/10.1007/978-3-642-35746-6_1
19. Dai, P., Goldsmith, J.: Topological value iteration algorithm for markov decision processes. In: Veloso, M.M. (ed.) IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007. pp. 1860–1865 (2007), http://ijcai.org/Proceedings/07/Papers/300.pdf
20. Dancey, C., Reidy, J.: Statistics Without Maths for Psychology (01 2004)
21. D'Argenio, P.R., Lee, M.D., Monti, R.E.: Input/output stochastic automata - compositionality and determinism. In: Fränzle, M., Markey, N. (eds.) Formal Modeling and Analysis of Timed Systems - 14th International Conference, FORMATS 2016, Quebec, QC, Canada, August 24-26, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9884, pp. 53–68. Springer (2016). https://doi.org/10.1007/978-3-319-44878-7_4
22. Eisentraut, C., Hermanns, H., Zhang, L.: Concurrency and composition in a stochastic world. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6269, pp. 21–39. Springer (2010). https://doi.org/10.1007/978-3-642-15375-4_3
23. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom. pp. 342–351. IEEE Computer Society (2010). https://doi.org/10.1109/LICS.2010.41
24. Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M.: SAT competition 2020. Artif. Intell. **301**, 103572 (2021). https://doi.org/10.1016/j.artint.2021.103572
25. Haddad, S., Monmege, B.: Reachability in mdps: Refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8762, pp. 125–137. Springer (2014). https://doi.org/10.1007/978-3-319-11439-2_10
26. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst. Des. **43**(2), 191–232 (2013). https://doi.org/10.1007/s10703-012-0167-z
27. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasmc: A web-based probabilistic model checker. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8442, pp. 312–317. Springer (2014). https://doi.org/10.1007/978-3-319-06410-9_22
28. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5), 512–535 (1994). https://doi.org/10.1007/BF01211866
29. Hartmanns, A.: Correct probabilistic model checking with floating-point arithmetic. CoRR **abs/2110.08785** (2021), https://arxiv.org/abs/2110.08785
30. Hartmanns, A., Hermanns, H.: The modest toolset: An integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51
31. Hartmanns, A., Hermanns, H.: In the quantitative automata zoo. Sci. Comput. Program. **112**, 3–23 (2015). https://doi.org/10.1016/j.scico.2015.08.009
32. Hartmanns, A., Hermanns, H., Bungert, M.: Flexible support for time and costs in scenario-aware dataflow. In: Eles, P., Mangharam, R. (eds.) 2016 International Conference on Embedded Software, EMSOFT 2016, Pittsburgh, Pennsylvania, USA, October 1-7, 2016. pp. 3:1–3:10. ACM (2016). https://doi.org/10.1145/2968478.2968496
33. Hartmanns, A., Kaminski, B.L.: Optimistic value iteration. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12225, pp. 488–511. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_26

34. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427, pp. 344–350. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_20
35. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker storm. CoRR **abs/2002.07080** (2020), https://arxiv.org/abs/2002.07080
36. Klauck, M., Hermanns, H.: A modest approach to dynamic heuristic search in probabilistic model checking. In: Abate, A., Marin, A. (eds.) Quantitative Evaluation of Systems - 18th International Conference, QEST 2021, Paris, France, August 23-27, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12846, pp. 15–38. Springer (2021). https://doi.org/10.1007/978-3-030-85172-9_2
37. 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). https://doi.org/10.1613/jair.1.11595
38. Kordon, F., Hillah, L., Hulin-Hubard, F., Jezequel, L., Paviot-Adet, E.: Study of the efficiency of model checking techniques using results of the MCC from 2015 to 2019. Int. J. Softw. Tools Technol. Transf. **23**(6), 931–952 (2021). https://doi.org/10.1007/s10009-021-00615-1
39. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012. pp. 203–204. IEEE Computer Society (2012). https://doi.org/10.1109/QEST.2012.14
40. 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). https://doi.org/10.1016/S0304-3975(01)00046-9, https://doi.org/10.1016/S0304-3975(01)00046-9
41. Mathur, U., Bauer, M.S., Chadha, R., Sistla, A.P., Viswanathan, M.: Exact quantitative probabilistic model checking through rational search. Formal Methods Syst. Des. **56**(1), 90–126 (2020). https://doi.org/10.1007/s10703-020-00348-y
42. McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Monographs in Computer Science, Springer (2005). https://doi.org/10.1007/b138392
43. McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: Raedt, L.D., Wrobel, S. (eds.) Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005. ACM International Conference Proceeding Series, vol. 119, pp. 569–576. ACM (2005). https://doi.org/10.1145/1102351.1102423
44. Neupane, T., Myers, C.J., Madsen, C., Zheng, H., Zhang, Z.: STAMINA: stochastic approximate model-checker for infinite-state analysis. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11561, pp. 540–549. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_31
45. Parker, D.: Prism manual version 4.7 (Mar 2021), https://www.prismmodelchecker.org/manual/Main/AllOnOnePage, accessed: 11 July 2022
46. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. CoRR **abs/1201.0490** (2012), http://arxiv.org/abs/1201.0490
47. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887
48. Quatmann, T., Katoen, J.: Sound value iteration. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10981, pp. 643–661. Springer (2018). https://doi.org/10.1007/978-3-319-96145-3_37
49. Ruijters, E., Budde, C., Chenariyan Nakhaee, M., Stoelinga, M., Bucur, D., Hiemstra, D., Schivo, S.: Ffort: A benchmark suite for fault tree analysis. pp. 878–885 (01 2019). https://doi.org/10.3850/978-981-11-2724-3_0641-cd
50. Ruijters, E., Reijsbergen, D., de Boer, P., Stoelinga, M.: Rare event simulation for dynamic fault trees. Reliab. Eng. Syst. Saf. **186**, 220–231 (2019). https://doi.org/10.1016/j.ress.2019.02.004
51. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3114, pp. 202–215. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_16

52. 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). https://doi.org/10.1613/jair.5153
53. Sutcliffe, G.: The CADE-26 automated theorem proving system competition - CASC-26. AI Commun. **30**(6), 419–432 (2017). https://doi.org/10.3233/AIC-170744
54. 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). https://doi.org/10.1613/jair.1880

A Appendix

A.1 Manual for reproducing results

QVAnalyser has been tested on Ubunutu 20. It should also work on other Linux distributions, however, we did not test this. For Windows users we recommend to use the Windows Subsystem for Linux (WSL).

1. Install the following dependencies.
   1.1 Python 3.10 https://computingforgeeks.com/how-to-install-python-on-ubuntu-linux-system/
   1.2 matplotlib https://matplotlib.org/
   1.3 numpy https://numpy.org/
2. Install the tools for quantitative verification in the folder Resources/Tools. Create this folder if it does not exist.
   2.1 Install version v3.1.190-g35b359a78 Modest via https://www.modestchecker.net/Downloads/
   2.2 Install Storm from the following repository https://github.com/tquatmann/storm/tree/dcd82f3604e2be5881c77d827d6afd901eca9023 Installation instructions of Storm are on https://www.stormchecker.org/documentation/obtain-storm/build.html
   2.3 Install Prism version 4.7 via https://www.prismmodelchecker.org/download.php
3. The benchmark has been generate using the following script. You do not have run this script, because the generated file is already in the repository.
   3.1 0Generate_BenchmarkFromQVBSOneInstancePerPropertyRandom.py
4. The benchmark can be started using the following scripts.
   4.1 1Benchmark_Thesis.py to start the benchmark
   4.2 2Continue_WithLatestSave.py to continue with the latest benchmark in case QVAnalyser was stopped
5. The benchmark can be started using the following scripts.
   5.1 3Parse_ResultsForThesis.py
6. Use the following scripts to generate the tables. The tables will be in Resources/Output.
   6.1 4Output_Thesis_Matrices.py This generates Tables 12 to 22
   6.2 4Output_Thesis_Statistics.py This generates Tables 23 to 25
   6.3 4Output_Thesis_PropertiesInBenchmark.py This generates Tables 26 to 28
   6.4 4Output_Thesis_SimilarityClustering.py This generates Tables 29 to 34 and Figures 6 to 11.
7. The output is in LaTeX. Add the following lines to the preamble.
   7.1
```
\usepackage{booktabs}
\usepackage{graphicx}
\newcommand{\fonttopsimilar}[0]{\fontsize{0pt}{0pt}\selectfont}
\newcommand{\fontcellsimilar}[0]{\fontsize{0pt}{0pt}\selectfont}
\usepackage[table,xcdraw]{xcolor}
```

A.2 Benchmark instances

Table 26. Relevant benchmark instances 1

Model name	Property name	Parameters
embedded	actuators	MAX_COUNT=6, T=12
	danger_time	MAX_COUNT=4, T=12
	io	MAX_COUNT=5, T=12
	main	MAX_COUNT=5, T=12
	sensors	MAX_COUNT=8, T=12
	up_time	MAX_COUNT=2, T=12
mapk cascade	activated_time	N=1, T=30
philosophers	MaxPrReachDeadlock	TIME_BOUND=1
	MinExpTimeDeadlock	TIME_BOUND=1
polling	s1_before_s2	T=16
bluetooth	time	mrec=1
brp	p1	N=16, MAX=2
	p2	N=16, MAX=2
	p4	N=32, MAX=4
coupon	collect_all	B=5
	exp_draws	B=5
crowds	positive	TotalRuns=6, CrowdSize=20
egl	messagesA	N=15, L=8
	messagesB	N=10, L=8
	unfairA	N=15, L=4
	unfairB	N=20, L=4
haddad monmege	target	N=20, p=0.7
	exp_steps	N=20, p=0.7
herman	steps	
leader sync	eventually_elected	
	time	
nand	reliable	N=60, K=1
oscillators	time_to_synch	mu=0.1, lambda=1.0
	power_consumption	mu=0.1, lambda=1.0
bitcoin attack	T_MWinMin	MALICIOUS=20, CD=6
breakdown queues	Min	K=32
	Max	K=8
dpm	PminQueuesFull	N=6, C=8, TIME_BOUND=5
	PmaxQueuesFull	N=8, C=6, TIME_BOUND=5
	PminQueue1Full	N=6, C=4, TIME_BOUND=5
	PmaxQueue1Full	N=4, C=8, TIME_BOUND=5
	TminQueuesFull	N=8, C=4, TIME_BOUND=5
erlang	PminReach	K=5000, R=100, TIME_BOUND=50
	TminReach	K=5000, R=100, TIME_BOUND=5
flexible manufacturing	M2Fail_E	T=1
	M3Fail_E	T=1
ftwc	ReachMinIsOne	N=4, TIME_BOUND=5
	TimeMin	N=4, TIME_BOUND=5
	TimeMax	N=8, TIME_BOUND=5
jobs	completiontime	
	avgtime	
polling system	PminBothFullIsOne	JOB_TYPES=3, C=3, TIME_BOUND=5
	TminBothFull	JOB_TYPES=3, C=3, TIME_BOUND=5
	TmaxBothFull	JOB_TYPES=3, C=3, TIME_BOUND=5

Table 27. Relevant benchmark instances 2

Model name	Property name	Parameters
readers writers	pr_many_requests	
	exp_time_many_requests	
	pr_network	
reentrant queues	PminBothQueuesFullIsOne	JOB_TYPES=3,C_LEFT=3, C_RIGHT=3, TIME_BOUND=5
	TminBothQueuesFull	JOB_TYPES=3,C_LEFT=3, C_RIGHT=3, TIME_BOUND=5
	TmaxBothQueuesFull	JOB_TYPES=3,C_LEFT=3, C_RIGHT=3, TIME_BOUND=5
stream	exp_buffertime	N=10
	exp_restarts	N=100
	pr_underrun	N=10
vgs	MaxPrReachFailed	TIME_BOUND=10000
	MinExpTimeFailed	TIME_BOUND=10000
beb	LineSeized	N=3
	GaveUp	N=3
blocksworld	goal	
boxworld	goal	
cdrive	goal	
consensus	c1	K=4
	c2	K=4
	disagree	K=2
	steps_max	K=2
	steps_min	K=2
csma	all_before_max	
	all_before_min	
	some_before	
	time_max	
	time_min	
eajs	ExpUtil	energy_capacity=100, B=5
echoring	MinFailed	ITERATIONS=100
	MinOffline1	ITERATIONS=2
	MaxOffline1	ITERATIONS=2
	MinOffline2	ITERATIONS=2
	MaxOffline2	ITERATIONS=100
	MinOffline3	ITERATIONS=50
	MaxOffline3	ITERATIONS=100
elevators	goal	
exploding blocksworld	goal	
firewire	elected	delay=36, deadline=400
	time_max	delay=3, deadline=200
	time_min	delay=3, deadline=600
	time_sending	delay=3, deadline=200
firewire abst	elected	delay=3
	rounds	delay=36
	time_max	delay=3
	time_min	delay=36
firewire dl	deadline	delay=3, deadline=200
ij	stable	

Table 28. Relevant benchmark instances 3

Model name	Property name	Parameters
pacman	crash	MAXSTEPS=100
philosophers mdp	eat	
pnueli zuck	live	
rabin	live	
random predicates	goal	
rectangle tireworld	goal	
resource gathering	expsteps	B=1000000,GOLD_TO_COLLECT=0, GEM_TO_COLLECT=0
tireworld	goal	
triangle tireworld	goal	
wlan	collisions	COL=0
	cost_max	COL=0
	cost_min	COL=0
	num_collisions	COL=0
	sent	COL=0
	time_max	COL=0
	time_min	COL=0
wlan dl	deadline	deadline=80
zenotravel	goal	
zeroconf	correct_max	N=1000, K=8, reset=False
	correct_min	N=20, K=6, reset=True
zeroconf dl	deadline_max	N=1000, K=1, reset=True, deadline=20
	deadline_min	N=1000, K=1, reset=True, deadline=30
brp pta	T_1	N=16, MAX=2, TD=1, TIME_BOUND=64
	T_2	N=64, MAX=12, TD=32, TIME_BOUND=256
	T_A1	N=64, MAX=12, TD=32, TIME_BOUND=256
	T_A2	N=64, MAX=12, TD=32, TIME_BOUND=256
	P_A	N=64, MAX=12, TD=32, TIME_BOUND=256
	P_B	N=64, MAX=12, TD=32, TIME_BOUND=256
	P_1	N=16, MAX=2, TD=1, TIME_BOUND=64
	P_2	N=16, MAX=2, TD=1, TIME_BOUND=64
	P_3	N=16, MAX=2, TD=1, TIME_BOUND=64
	P_4	N=16, MAX=2, TD=1, TIME_BOUND=64
	Emax	N=64, MAX=12, TD=32, TIME_BOUND=256
	Emin	N=64, MAX=12, TD=32, TIME_BOUND=256
csma pta	collisions	K=2, COL=8
csma abst pta	eventually	K=1, T=1800
firewire pta	eventually	delay=30, T=7500
firewire abst pta	eventually	delay=360, T=5000
repudiation honest	eventually	T=40
repudiation malicious	eventually	T=10
wlan large	P_1	K=2
	P_min	K=2
	P_max	K=2
	E_and	K=2
	E_or	K=2
	E_1	K=2
zeroconf pta	incorrect	T=200

A.3 Similarity of algorithms using clustering algorithm

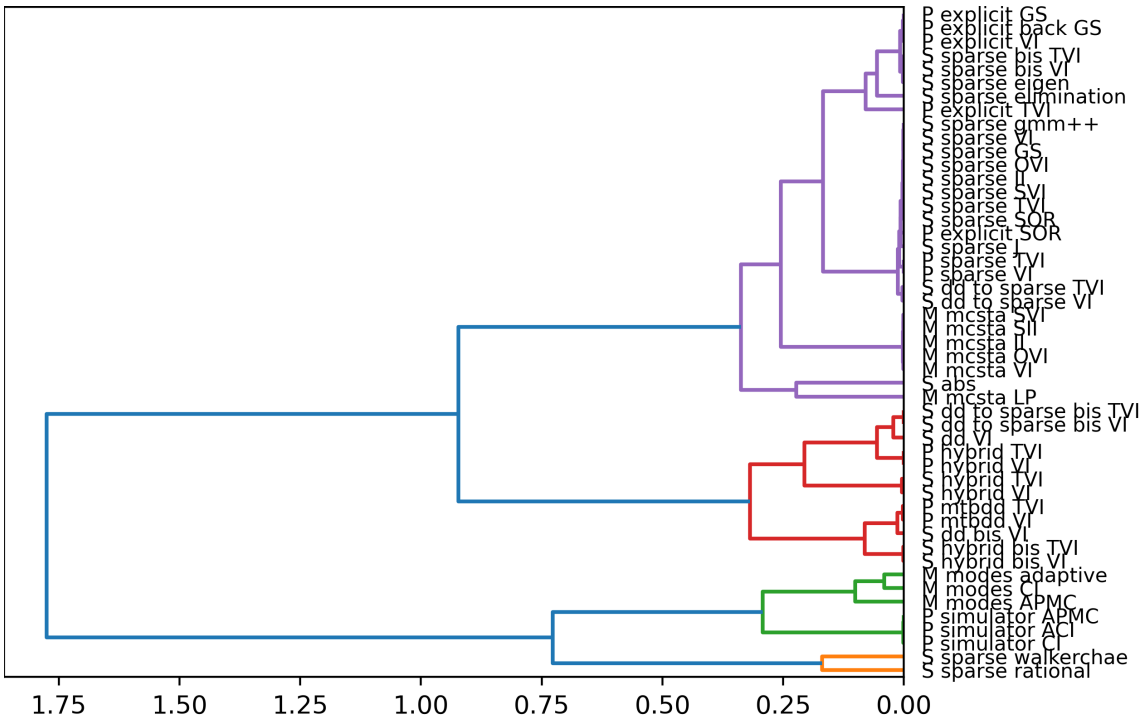


Fig. 6. Dendrogram for similarity of algorithms for DTMC and CTMC benchmark instances

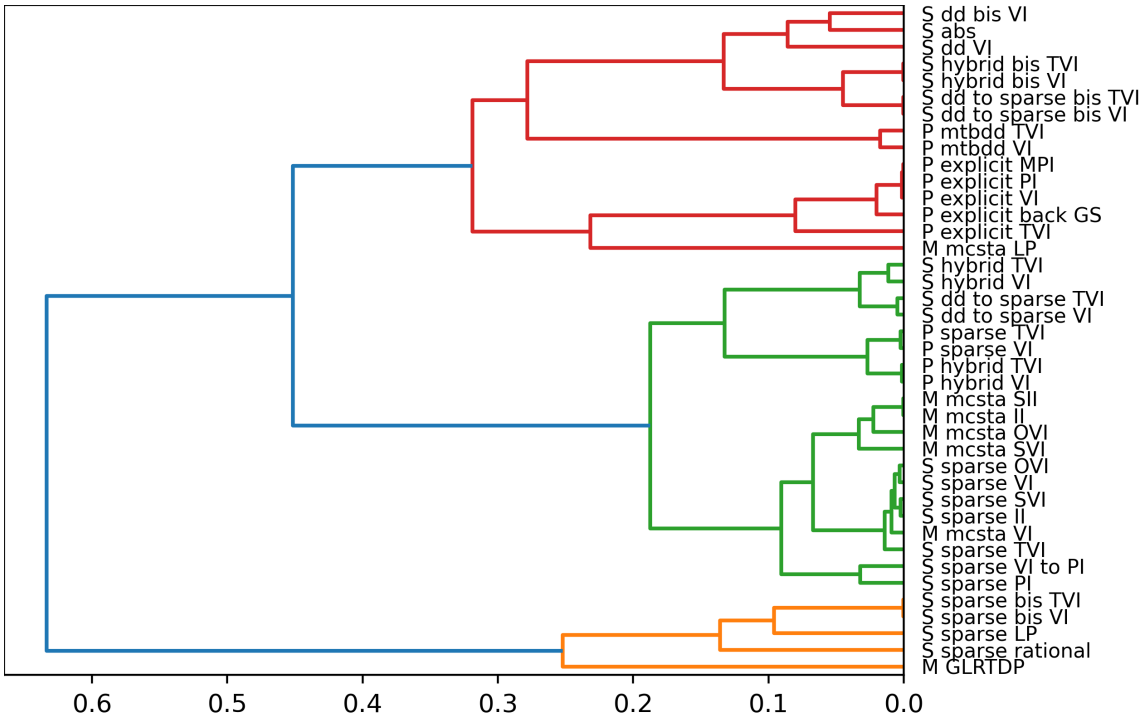


Fig. 7. Dendrogram for similarity of algorithms for MDP and MA benchmark instances

Table 29. Similarity of algorithms on DTMC and CTMC benchmark instances

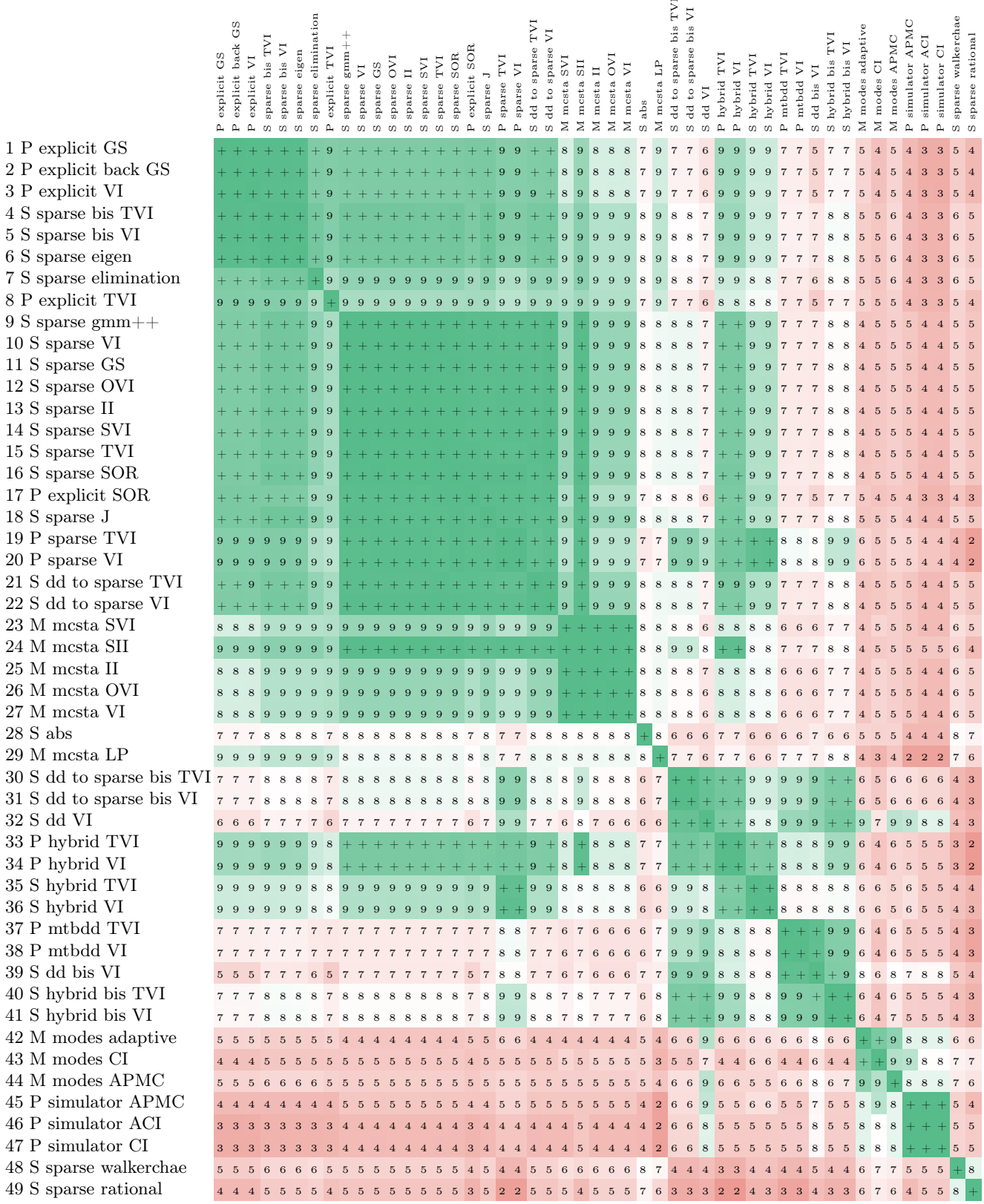


Table 30. Similarity of algorithms on MDP and MA benchmark instances

	S dd bis VI	S abs	S dd VI	S hybrid bis TVI	S hybrid bis VI	S dd to sparse bis TVI	S dd to sparse bis VI	P mtbdd TVI	P mtbdd VI	P explicit MPI	P explicit PI	P explicit VI	P explicit back GS	P explicit TVI	M mcsta LP	S hybrid TVI	S hybrid VI	S dd to sparse TVI	S dd to sparse VI	P sparse TVI	P sparse VI	P hybrid TVI	P hybrid VI	M mcsta SII	M mcsta II	M mcsta OVI	M mcsta SVI	S sparse OVI	S sparse VI	S sparse SVI	S sparse II	M mcsta VI	S sparse TVI	S sparse VI to PI	S sparse PI	S sparse bis TVI	S sparse bis VI	S sparse LP	S sparse rational	M GLRTDP
1 S dd bis VI	+	9	9	+	+	9	9	8	8	9	9	9	9	9	8	8	8	8	8	8	8	8	8	9	8	9	9	9	9	9	9	9	8	8	8	9	9	9	9	7
2 S abs	9	+	9	9	9	9	9	9	9	9	9	9	9	9	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	7
3 S dd VI	9	9	+	9	9	9	9	8	9	9	9	9	9	8	7	9	9	9	9	9	9	9	9	+	9	9	9	9	9	9	9	9	9	9	9	8	8	8	9	7
4 S hybrid bis TVI	+	9	9	+	+	+	+	8	8	8	9	9	9	8	8	8	8	8	8	8	8	8	8	8	8	8	9	8	8	8	8	8	8	8	8	8	8	8	8	7
5 S hybrid bis VI	+	9	9	+	+	+	+	8	8	8	9	9	9	8	8	8	8	8	8	8	8	8	8	8	8	8	9	8	8	8	8	8	8	8	8	8	8	8	8	7
6 S dd to sparse bis TVI	9	9	9	+	+	+	+	8	8	9	9	9	9	8	8	8	8	8	8	8	8	9	9	9	8	8	9	8	8	8	8	8	8	8	8	9	8	8	8	7
7 S dd to sparse bis VI	9	9	9	+	+	+	+	8	8	9	9	9	9	8	8	8	8	8	8	8	8	9	9	9	8	8	9	8	8	8	8	8	8	8	8	9	8	8	8	7
8 P mtbdd TVI	8	9	8	8	8	8	8	+	+	8	9	9	8	9	7	8	8	8	8	9	9	9	9	8	9	9	9	8	8	8	8	8	8	8	8	8	8	8	8	7
9 P mtbdd VI	8	9	9	8	8	8	8	+	+	8	9	9	9	8	7	8	8	8	8	9	9	9	9	8	9	9	9	9	9	9	9	9	9	8	8	9	8	8	8	7
10 P explicit MPI	9	9	9	8	8	9	9	8	8	+	+	+	+	+	8	9	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	7
11 P explicit PI	9	9	9	9	9	9	9	9	9	+	+	+	+	9	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	8	8	8	9	7
12 P explicit VI	9	9	9	9	9	9	9	9	9	+	+	+	+	9	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	8	8	8	9	7
13 P explicit back GS	9	9	9	9	9	9	9	8	9	+	+	+	+	9	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	8	8	8	9	7
14 P explicit TVI	9	9	8	8	8	8	8	9	8	+	9	9	9	+	9	8	8	8	8	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	8	8	9	7
15 M mcsta LP	8	8	7	8	8	8	8	7	7	8	8	8	8	9	+	6	6	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	8	8	7	7	7
16 S hybrid TVI	8	9	9	8	8	8	8	8	8	9	9	9	9	8	6	+	+	+	+	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	7	7	7	7	6
17 S hybrid VI	8	9	9	8	8	8	8	8	8	8	9	9	9	8	6	+	+	+	+	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	7	7	7	7	6
18 S dd to sparse TVI	8	9	9	8	8	8	8	8	8	9	9	9	9	8	7	+	+	+	+	9	9	9	9	9	9	9	9	+	+	+	9	+	9	9	7	7	8	7	6
19 S dd to sparse VI	8	9	9	8	8	8	8	8	8	9	9	9	9	8	7	+	+	+	+	9	9	9	9	9	9	9	9	+	+	+	+	9	+	9	9	7	7	8	7	6
20 P sparse TVI	8	9	9	8	8	8	8	9	9	9	9	9	9	9	7	9	9	9	9	+	+	+	+	9	9	9	9	9	9	9	9	9	9	9	9	9	7	7	8	8	6
21 P sparse VI	8	9	9	8	8	8	8	9	9	9	9	9	9	9	7	9	9	9	9	+	+	+	+	9	9	9	9	9	9	9	9	9	9	9	9	9	7	7	8	8	6
22 P hybrid TVI	8	9	9	8	8	9	9	9	9	9	9	9	9	9	7	9	9	9	9	+	+	+	+	9	9	9	9	9	9	9	9	9	+	9	9	9	7	7	8	8	6
23 P hybrid VI	8	9	9	8	8	9	9	9	9	9	9	9	9	9	7	9	9	9	9	+	+	+	+	9	9	9	9	9	9	9	9	9	+	9	9	9	7	7	8	8	6
24 M mcsta SII	9	9	+	8	8	9	9	8	8	9	9	9	9	9	7	9	9	9	9	9	9	9	9	+	+	+	+	+	+	+	+	+	+	+	+	8	8	8	7	7
25 M mcsta II	8	9	9	8	8	8	8	9	9	9	9	9	9	9	7	9	9	9	9	9	9	9	9	+	+	+	+	9	+	+	+	+	9	9	9	7	7	7	7	7
26 M mcsta OVI	9	9	9	8	8	8	8	9	9	9	9	9	9	9	7	9	9	9	9	9	9	9	9	+	+	+	+	+	+	+	+	+	+	9	9	8	8	8	7	7
27 M mcsta SVI	9	9	9	9	9	9	9	9	9	9	9	9	9	9	7	9	9	9	9	9	9	9	9	+	+	+	+	+	+	+	+	+	+	9	9	8	8	8	7	7
28 S sparse OVI	9	9	9	8	8	8	8	8	9	9	9	9	9	9	7	9	9	9	+	9	9	9	9	+	9	+	+	+	+	+	+	+	+	+	9	8	8	8	7	7
29 S sparse VI	9	9	9	8	8	8	8	8	9	9	9	9	9	9	7	9	9	+	+	9	9	9	9	+	+	+	+	+	+	+	+	+	+	+	9	8	8	8	7	7
30 S sparse SVI	9	9	9	8	8	8	8	8	9	9	9	9	9	9	7	9	9	+	+	9	9	9	9	+	+	+	+	+	+	+	+	+	+	+	9	8	8	8	7	7
31 S sparse II	9	9	9	8	8	8	8	8	9	9	9	9	9	9	7	9	9	+	+	9	9	9	9	+	+	+	+	+	+	+	+	+	+	+	9	8	8	8	7	7
32 M mcsta VI	9	9	9	8	8	8	8	8	9	9	9	9	9	9	7	9	9	9	9	9	9	9	9	+	+	+	+	+	+	+	+	+	+	+	9	8	8	8	7	7
33 S sparse TVI	8	9	9	8	8	8	8	8	8	9	9	9	9	9	7	9	9	+	+	9	9	+	+	+	9	+	+	+	+	+	+	+	+	+	9	7	7	8	7	7
34 S sparse VI to PI	8	9	9	8	8	8	8	8	8	9	9	9	9	9	7	9	9	9	9	9	9	9	9	+	9	9	9	+	+	+	+	+	+	+	+	8	8	8	7	7
35 S sparse PI	8	9	9	8	8	9	9	8	9	9	9	9	9	9	7	9	9	9	9	9	9	9	9	+	9	9	9	9	9	9	9	9	+	+	8	8	8	7	7
36 S sparse bis TVI	9	9	8	8	8	8	8	8	8	9	8	8	8	8	8	8	7	7	7	7	7	7	7	7	8	7	8	8	8	8	8	8	8	7	8	8	+	+	9	9	8
37 S sparse bis VI	9	9	8	8	8	8	8	8	8	9	8	8	8	8	8	8	7	7	7	7	7	7	7	7	8	7	8	8	8	8	8	8	8	7	8	8	+	+	9	9	8
38 S sparse LP	9	9	8	8	8	8	8	8	8	9	8	8	8	9	7	7	7	8	8	8	8	8	8	8	7	8	8	8	8	8	8	8	8	8	8	8	9	9	+	9	8
39 S sparse rational	9	9	9	8	8	8	8	8	8	9	9	9	9	9	7	7	7	7	7	8	8	8	8	7	7	7	7	7	7	7	7	7	7	7	7	7	9	9	9	+	8
40 M GLRTDP	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	6	6	6	6	6	6	6	6	7	7	7	7	7	7	7	7	7	7	7	7	7	8	8	8	8	+

A.4 Similarity of benchmark instances using clustering algorithm

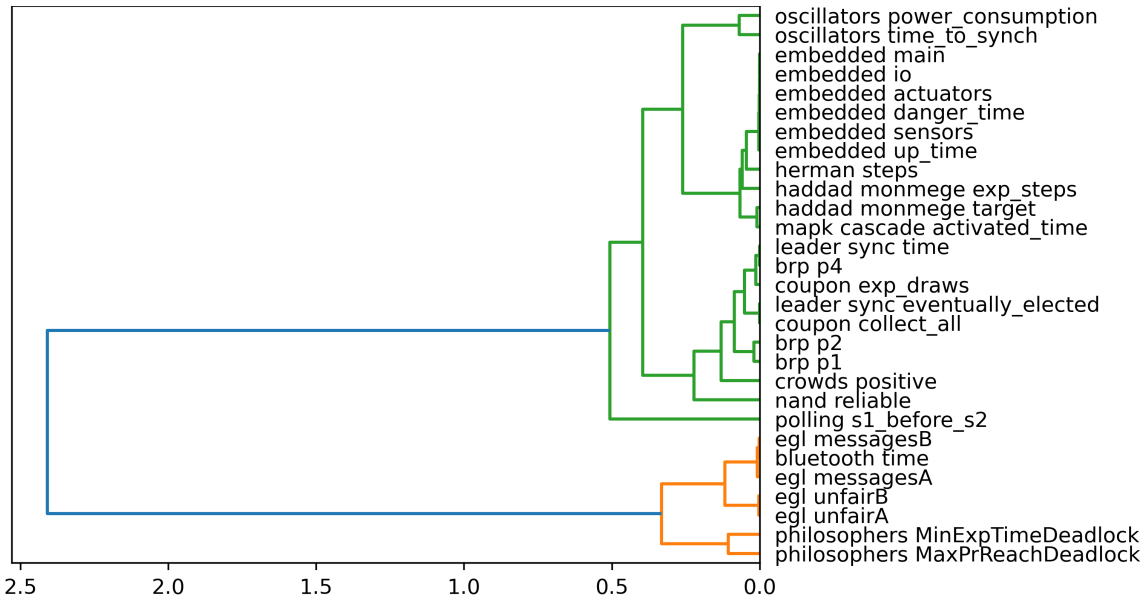


Fig. 8. Dendrogram for similarity of DTMC and CTMC benchmark instances

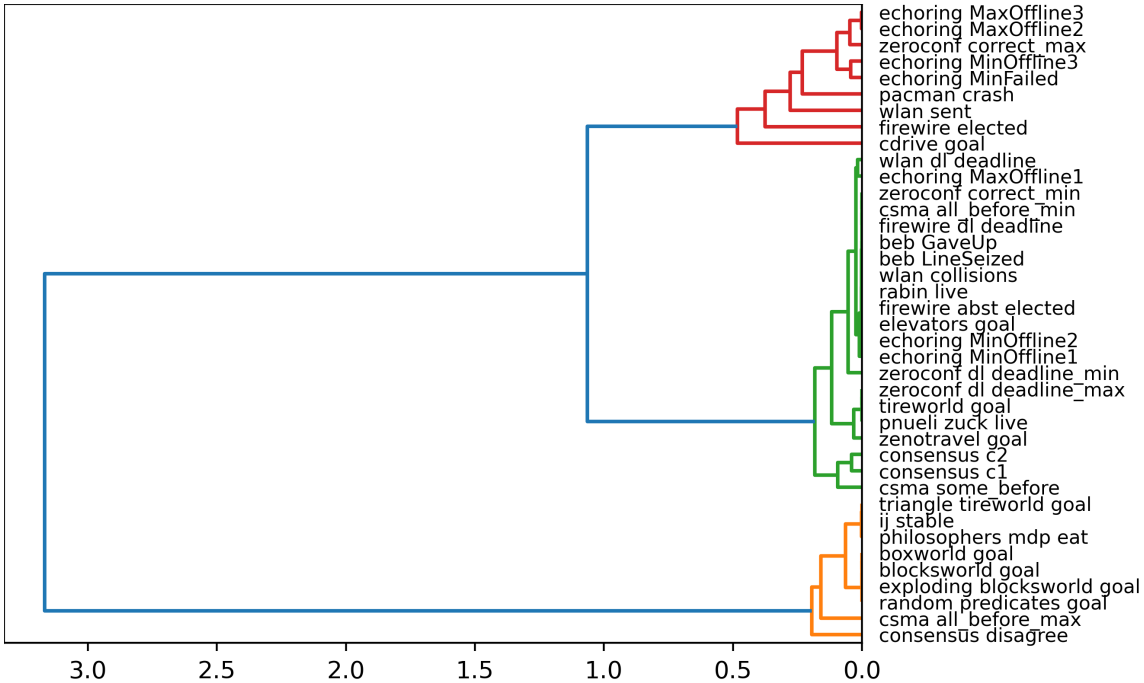


Fig. 9. Dendrogram for similarity of MDP reachability probability benchmark instances

Table 31. Similarity of DTMC and CTMC benchmark instances

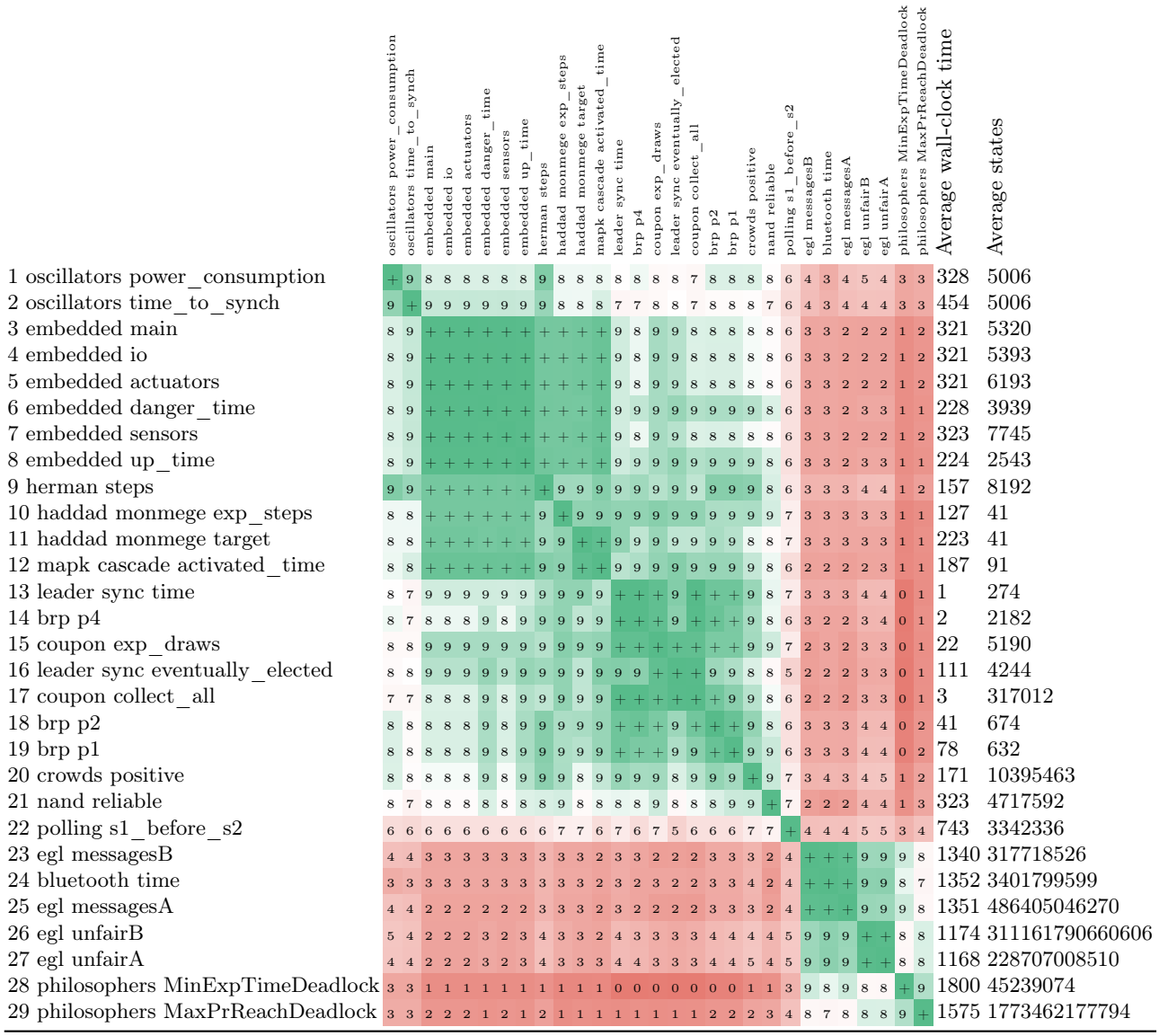


Table 32. Similarity of MDP reachability probability benchmark instances

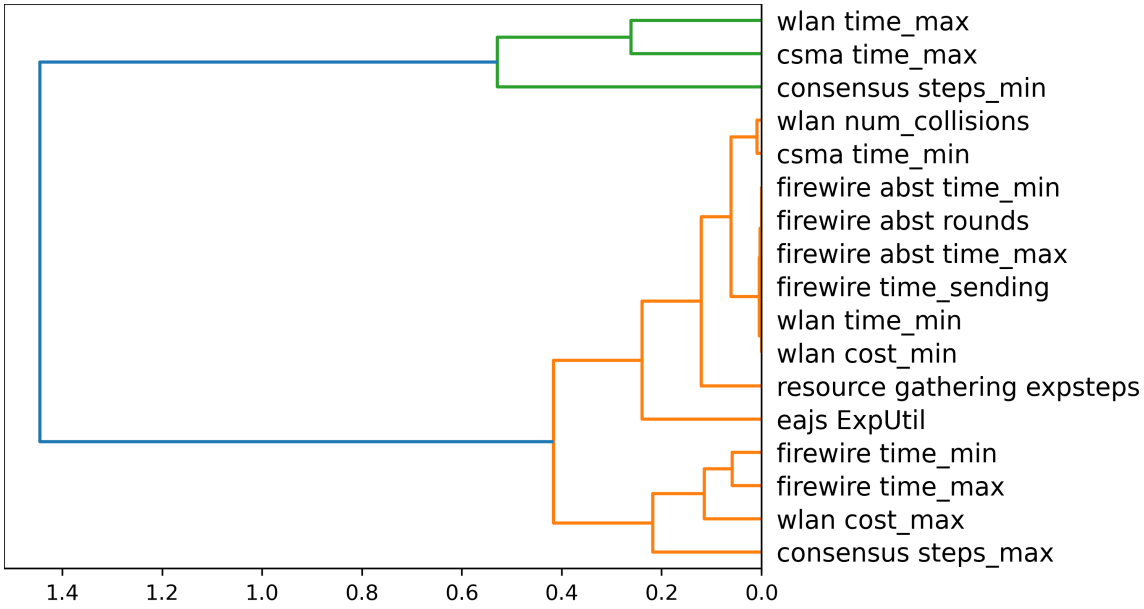


Fig. 10. Dendrogram for similarity of MDP expected rewards benchmark instances

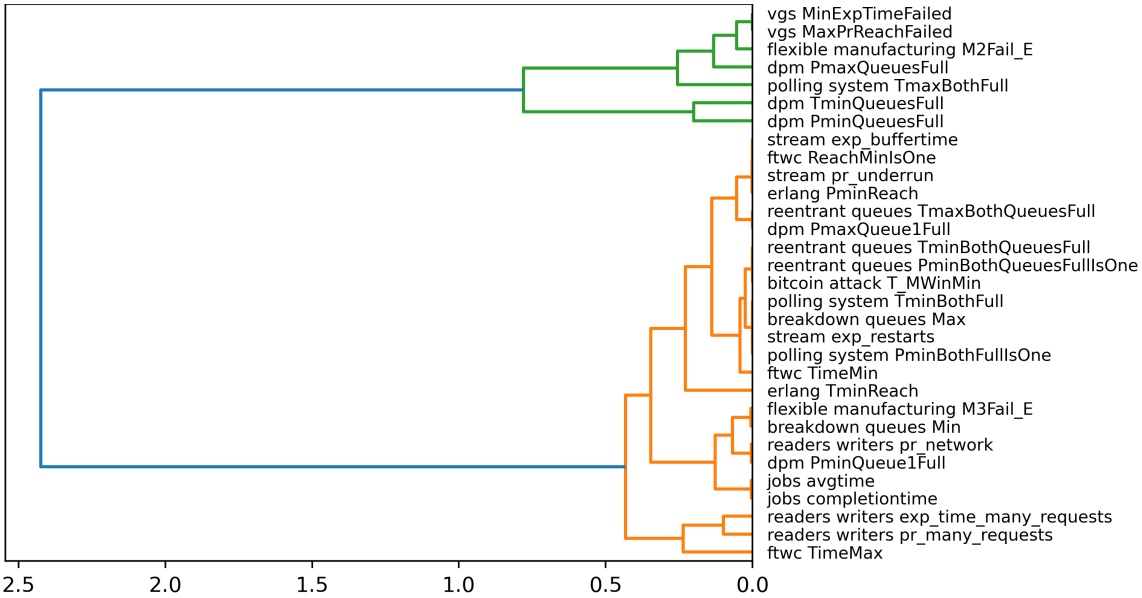


Fig. 11. Dendrogram for similarity of MA rewards benchmark instances

Table 33. Similarity of MDP expected rewards benchmark instances

	wlan time_max	csma time_max	consensus steps_min	wlan num_collisions	csma time_min	firewire abst time_min	firewire abst rounds	firewire abst time_max	firewire time_sending	wlan time_min	wlan cost_min	resource gathering expsteps	eajs ExpUtil	firewire time_min	firewire time_max	wlan cost_max	consensus steps_max	Average wall-clock time	Average states
1 wlan time_max	+	7	4	5	5	4	4	4	4	4	4	5	5	5	5	6	6	1028	5007548
2 csma time_max	7	+	7	2	2	2	2	2	1	2	2	2	3	3	3	4	4	1498	84856004
3 consensus steps_min	4	7	+	2	2	2	2	2	2	2	2	3	2	3	3	3	4	1512	61018112
4 wlan num_collisions	5	2	2	+	+	+	+	+	+	+	+	9	8	9	9	8	8	52	8625
5 csma time_min	5	2	2	+	+	+	+	+	9	+	+	9	8	9	9	8	8	69	36850
6 firewire abst time_min	4	2	2	+	+	+	+	+	+	+	+	9	8	9	9	8	7	1	776
7 firewire abst rounds	4	2	2	+	+	+	+	+	+	+	+	9	8	9	9	8	7	1	776
8 firewire abst time_max	4	2	2	+	+	+	+	+	+	+	+	9	8	9	9	8	7	1	611
9 firewire time_sending	4	1	2	+	9	+	+	+	+	+	+	9	8	9	9	8	7	6	4093
10 wlan time_min	4	2	2	+	+	+	+	+	+	+	+	9	8	9	9	8	7	5	28480
11 wlan cost_min	4	2	2	+	+	+	+	+	+	+	+	9	8	9	9	8	7	6	28480
12 resource gathering expsteps	5	2	3	9	9	9	9	9	9	9	9	+	8	8	8	7	7	139	1
13 eajs ExpUtil	5	3	2	8	8	8	8	8	8	8	8	8	+	7	7	6	7	298	12828
14 firewire time_min	5	3	3	9	9	9	9	9	9	9	9	8	7	+	9	9	8	163	1078419
15 firewire time_max	5	3	3	9	9	9	9	9	9	9	9	8	7	9	+	9	8	178	83030
16 wlan cost_max	6	4	3	8	8	8	8	8	8	8	8	7	6	9	9	+	8	355	345000
17 consensus steps_max	6	4	4	8	8	7	7	7	7	7	7	7	7	8	8	8	+	492	1258240

Table 34. Similarity of MA benchmark instances