Probabilistic Model Checking of Randomized Java Code

Syyeda Zainab Fatmi

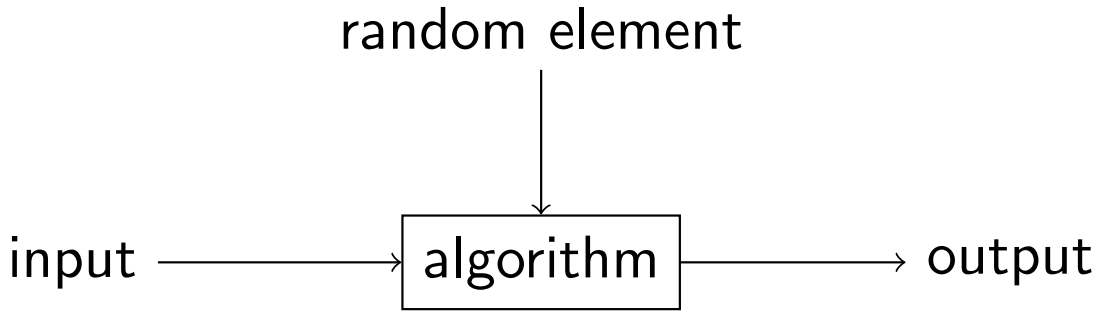
Supervisor: Franck van Breugel

York University

September 2020

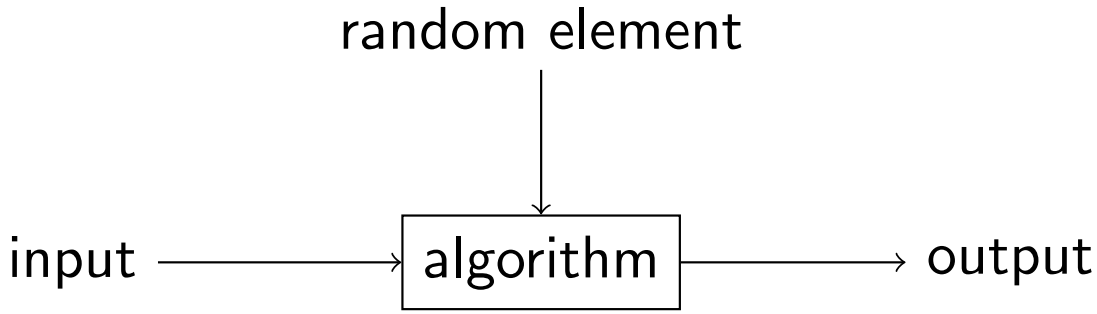
Introduction

A randomized algorithm is an algorithm that makes random choices during execution, so its behaviour can vary even on a fixed input.



Introduction

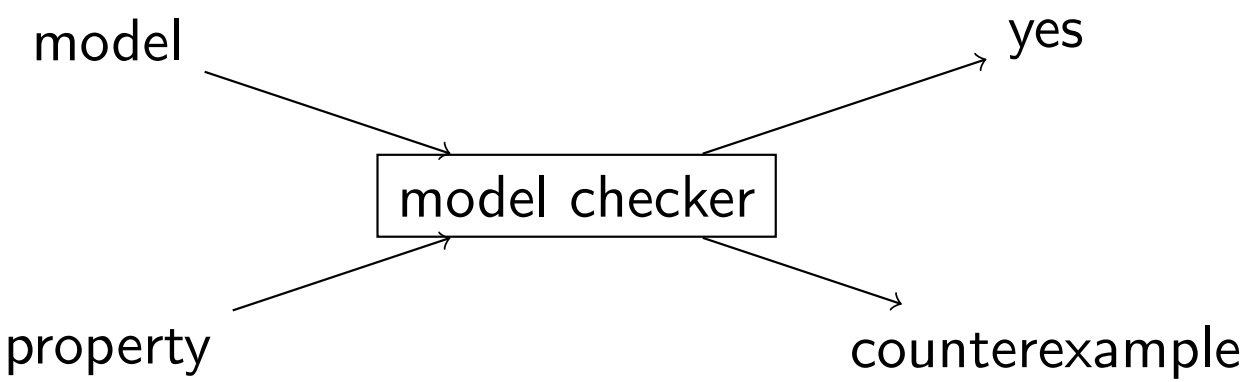
A randomized algorithm is an algorithm that makes random choices during execution, so its behaviour can vary even on a fixed input.



Software testing is most commonly used to show the presence of bugs in software systems. However, running a test on software with randomness multiple times does not guarantee that all possible executions are checked.

Model Checking

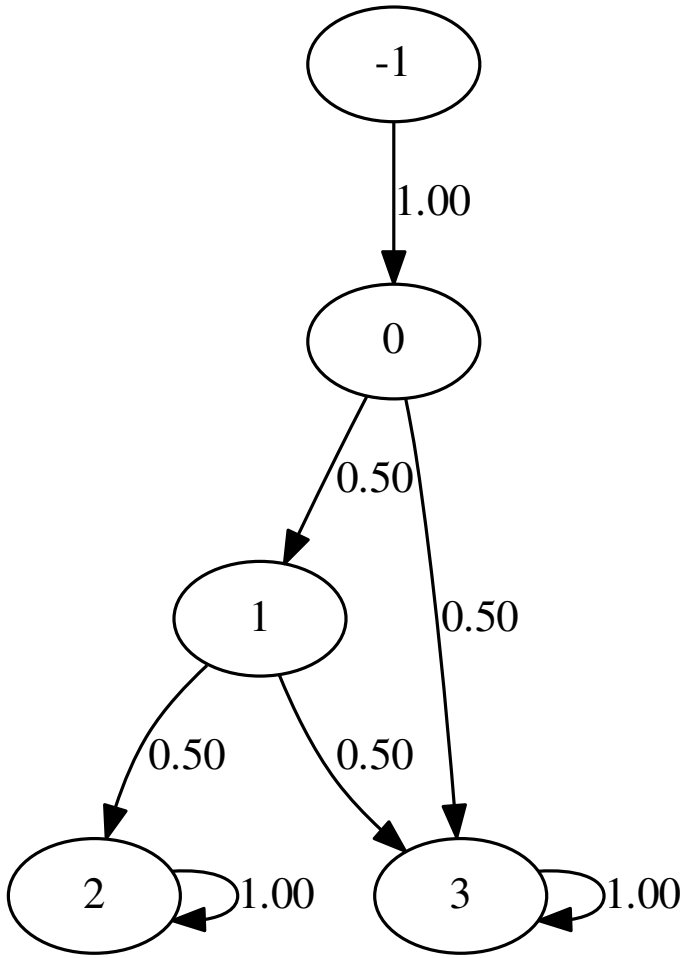
Model checking is a formal verification technique that can be used to show the absence of bugs in the presence of randomness and concurrency, introduced by the Turing award winners Clarke, Emerson, and Sifakis.



Java PathFinder

Java PathFinder (JPF) is a model checker for Java code. Running JPF on Java code can be viewed as building on the fly a model of the code.

The extension of JPF, jpf-probabilistic, assigns probabilities to the transitions of the model, which reflect the random choices in the Java code.



jpf-label

JPF does not provide an easy way to label the states.
We developed an extension of JPF, named jpf-label, that allows users to easily label states with atomic propositions, by defining custom labelling functions. Our extension supports both

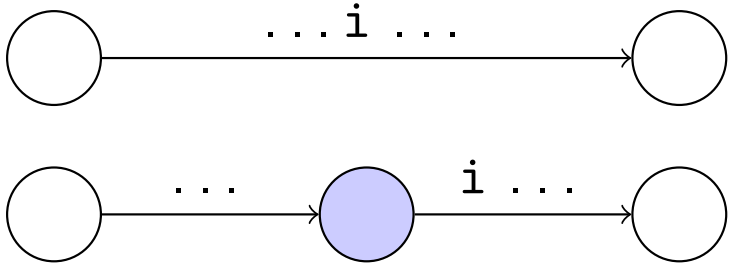
- state labelling and
- transition labelling.

jpf-label

Transitions between states represent the execution of a sequence of bytecode instructions. In the event that we wish to observe a specific instruction, for example `i`, we may break the transition

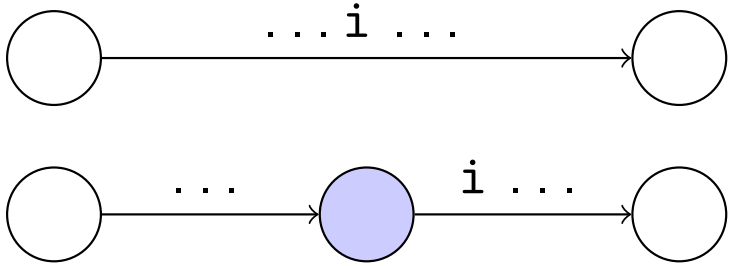
Transitions between states represent the execution of a sequence of bytecode instructions. In the event that we wish to observe a specific instruction, for example `i`, we may break the transition

- either before

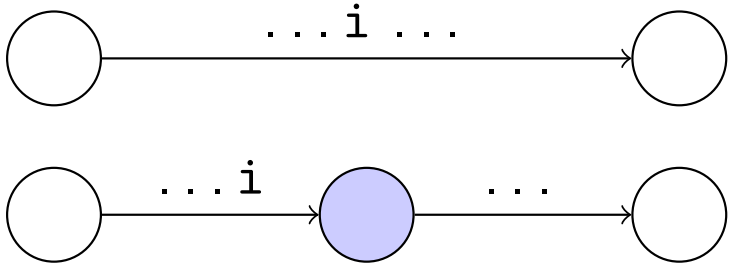


Transitions between states represent the execution of a sequence of bytecode instructions. In the event that we wish to observe a specific instruction, for example `i`, we may break the transition

- either before



- or after



the point of interest and label the newly generated state.

jpf-label

We provide twelve different ways to label states, including:

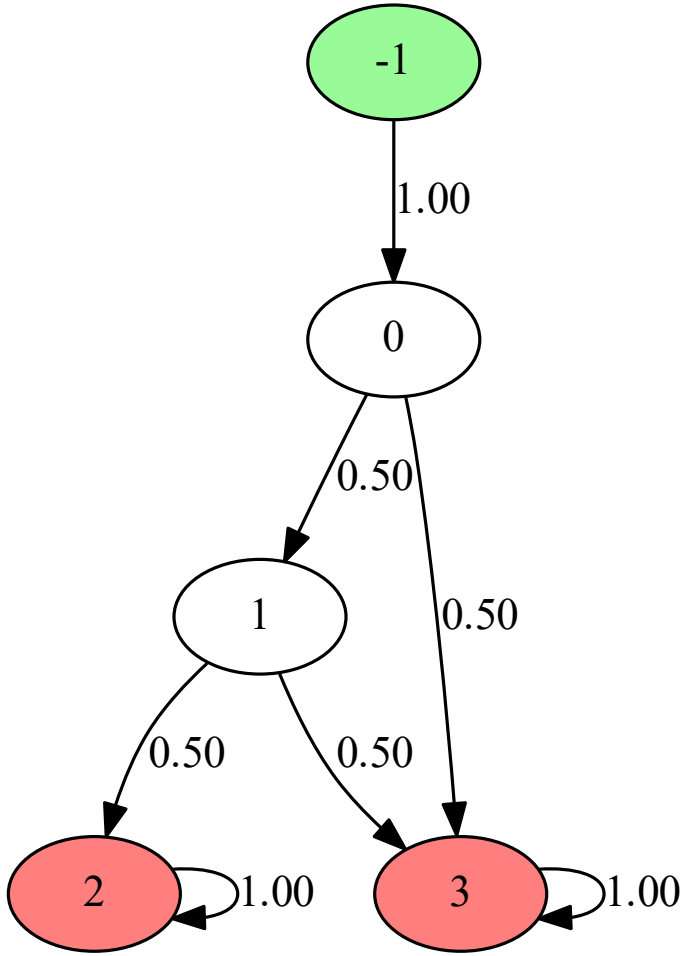
- initial and final states,
- static boolean fields and local boolean variables,
- boolean expressions built from local integer variables,
- static method invocations,
- method returns and the values returned, and
- thrown exceptions and the exception types.

jpf-label

Thus, using JPF, extended by both jpf-probabilistic and jpf-label, we can generate a labelled Markov chain.

The labelled Markov chain can be represented

- graphically or
- textually, by a transition file and a label file.



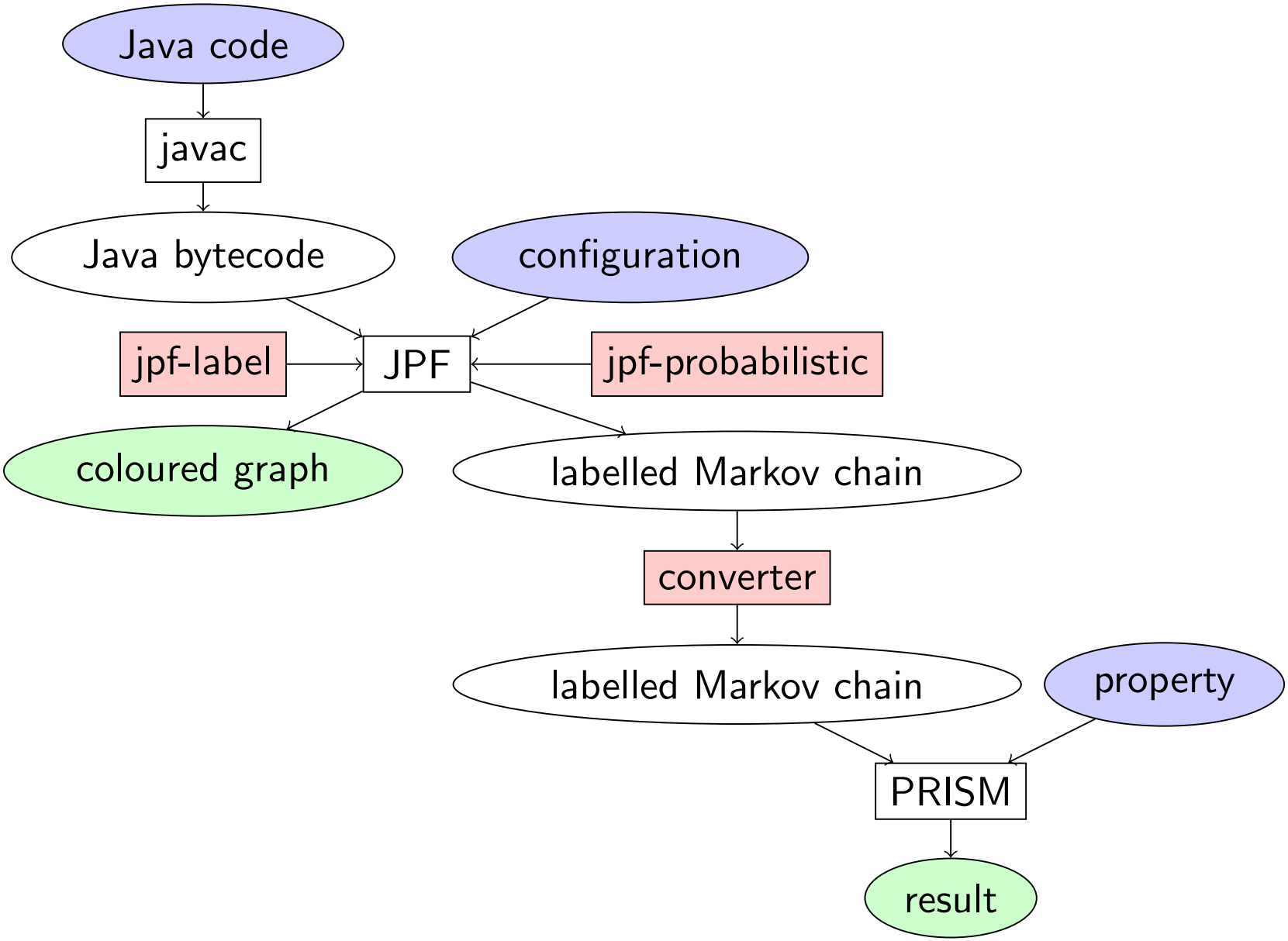
PRISM

PRISM is the most popular probabilistic model checker.

We have created a converter that transforms the transition and labelling files produced by JPF, with the help of jpf-label and jpf-probabilistic, into a format that can be fed into PRISM together with a probabilistic property.

PRISM then checks, among other things, whether the model satisfies the property.

Our Tool


Java code
javac
Java bytecode
configuration
jpf-label
JPF
jpf-probabilistic
coloured graph
labelled Markov chain
converter
labelled Markov chain
property
PRISM
result

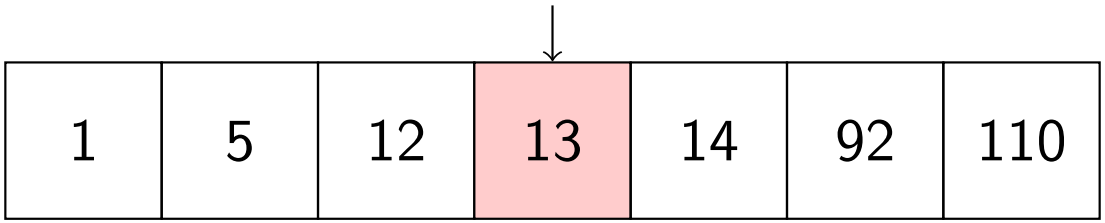
Example: Binary Search

Assume that we want to find the index of the value `12` in the sorted array below.

1	5	12	13	14	92	110

Example: Binary Search

Assume that we want to find the index of the value `12` in the sorted array below.



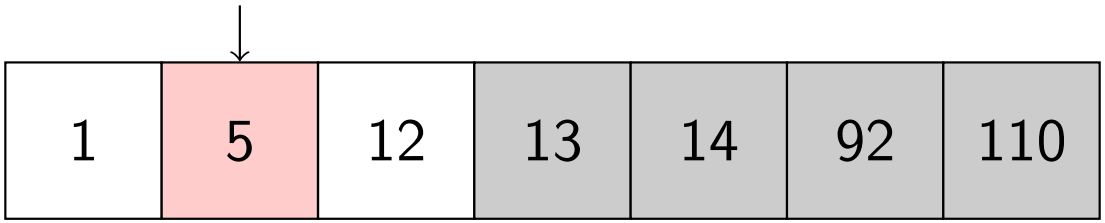
Example: Binary Search

Assume that we want to find the index of the value `12` in the sorted array below.

1	5	12	13	14	92	110

Example: Binary Search

Assume that we want to find the index of the value `12` in the sorted array below.



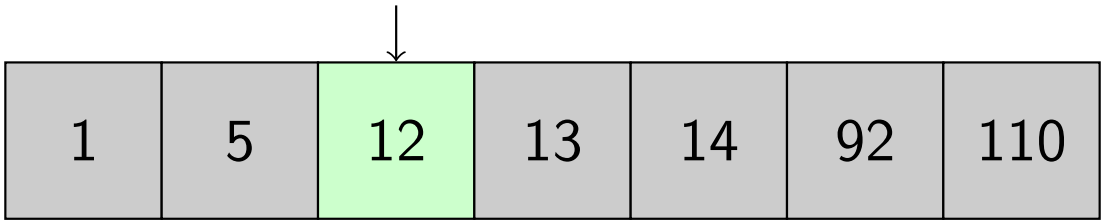
Example: Binary Search

Assume that we want to find the index of the value `12` in the sorted array below.

1	5	12	13	14	92	110

Example: Binary Search

Assume that we want to find the index of the value `12` in the sorted array below.



Example: Random Binary Search

This deterministic binary search algorithm can be randomized by choosing a random element from the current range instead of the middle element.

Example: Random Binary Search

This deterministic binary search algorithm can be randomized by choosing a random element from the current range instead of the middle element.

We add the ghost boolean variable `isWorse` to the algorithm, which captures whether the randomized algorithm performs worse than the deterministic algorithm. We then label the states with the value of that variable.

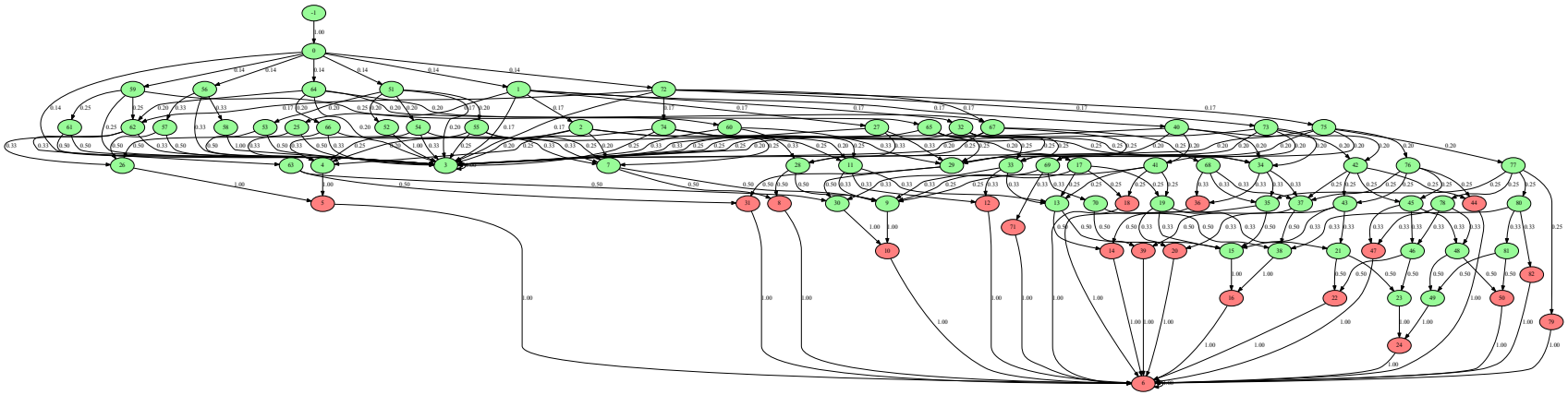
We extract the underlying labelled Markov chain with JPF and compute the property `P=? [ F "true__RandomBinary_isWorse" ]` with PRISM.

Example: Random Binary Search

1	5	12	13	14	92	110

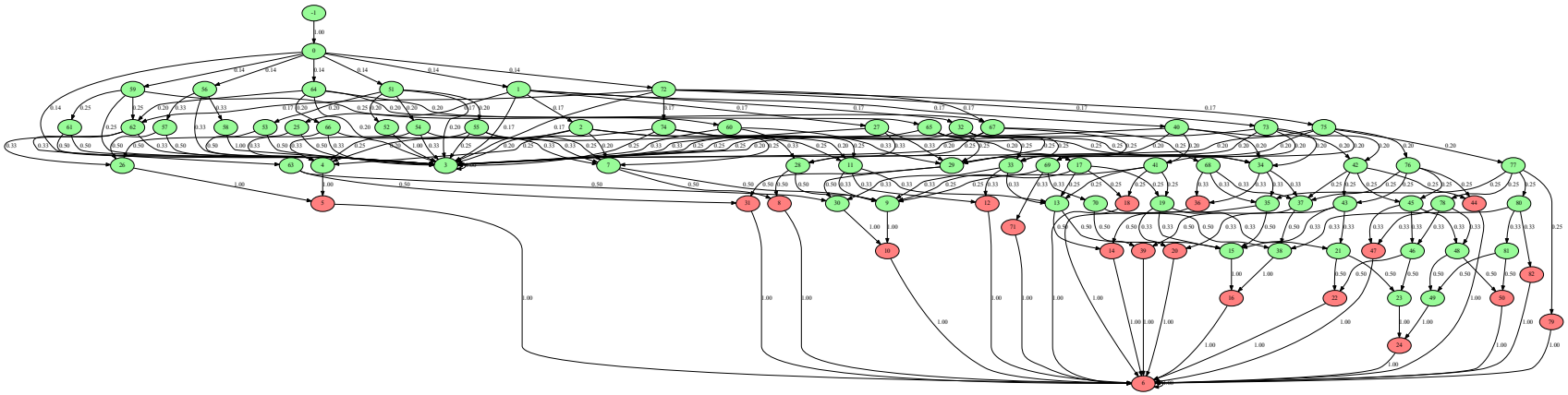
Example: Random Binary Search

1	5	12	13	14	92	110



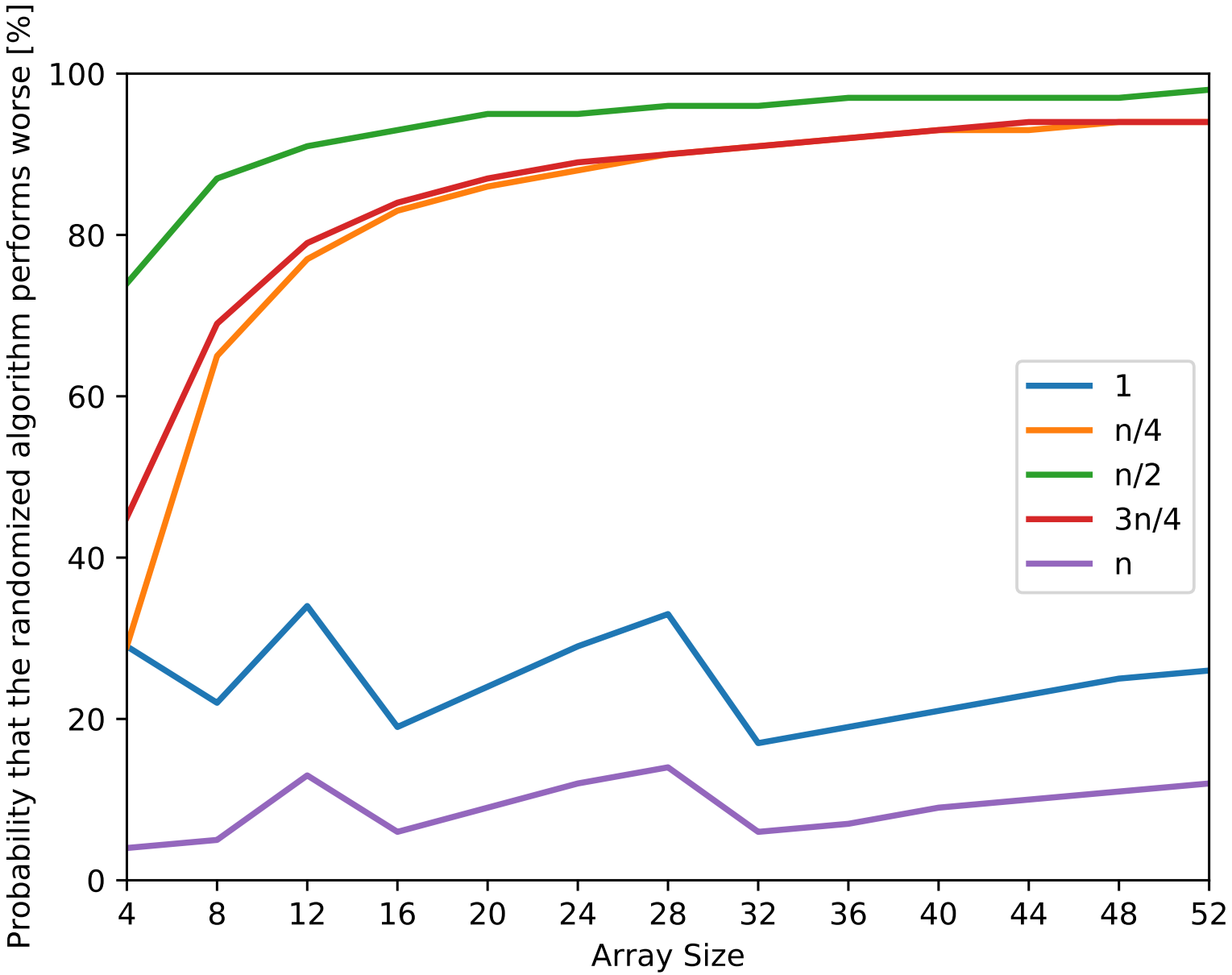
Example: Random Binary Search

1	5	12	13	14	92	110



PRISM returns the probability `0.39841`, which denotes the probability that the randomized algorithm performs worse than the deterministic algorithm.

Example: Random Binary Search



State Space Explosion

One of the major challenges of model checking is that the number of states in a model is often too large to check non-trivial properties of the system. In such a case, the model checker may run out of time or memory before successfully verifying whether the property holds in the model. This is called the *state space explosion* problem.

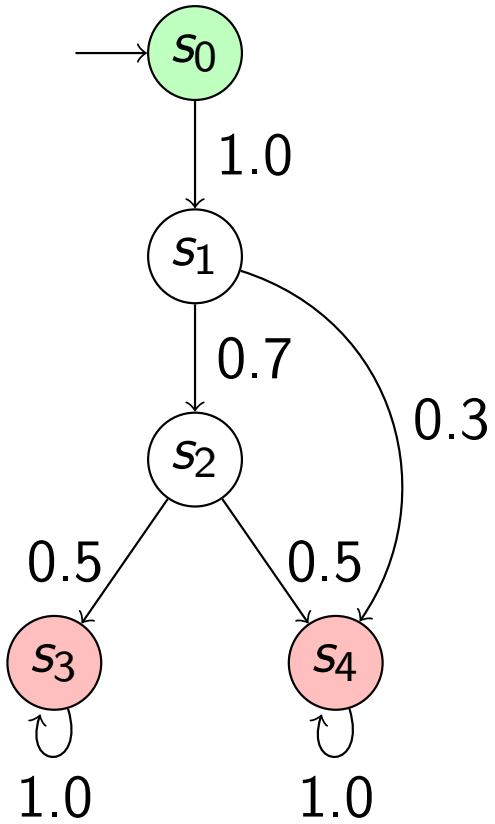
Probabilistic Bisimilarity

A probabilistic bisimulation on a system is an equivalence relation such that for every pair of equivalent states,

- both states have the same set of labels, and
- both states have equal probability of transitioning to each equivalence class.

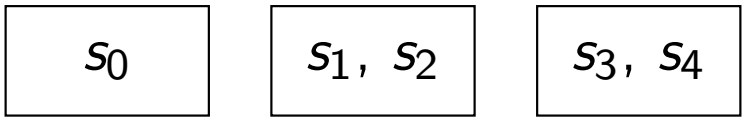
Probabilistic bisimilarity is the largest probabilistic bisimulation.

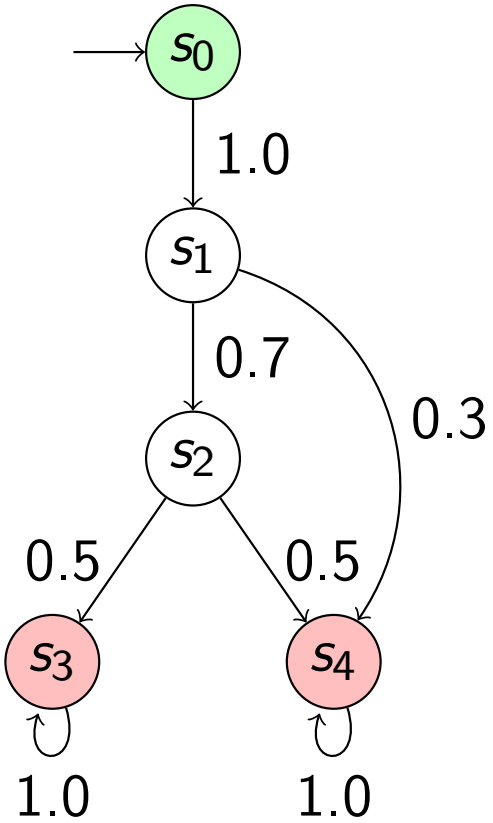
Example



Example

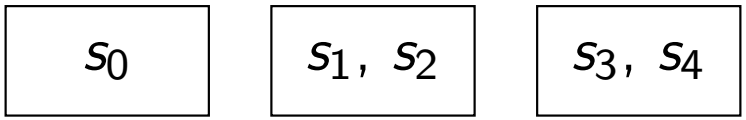
The initial partition:



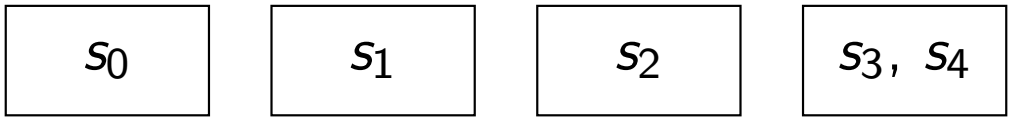


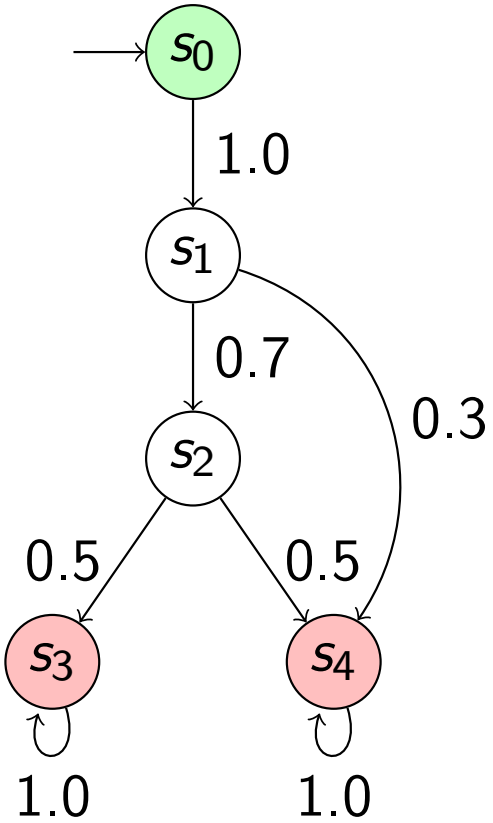
Example

The initial partition:



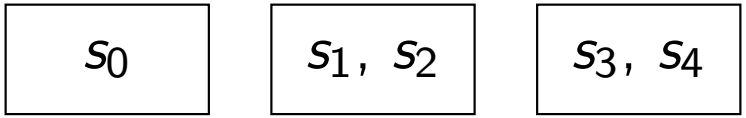
The first refinement:



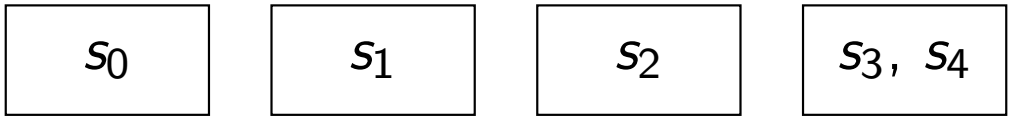


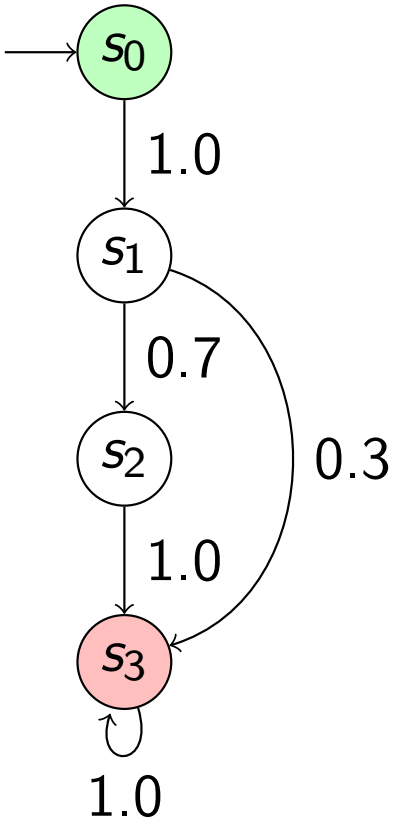
Example

The initial partition:



The first refinement:





Algorithms

We analyzed and implemented the following partition refinement algorithms to compute probabilistic bisimilarity for labelled Markov chains:

- Buchholz[1]
- Derisavi, Hermanns and Sanders[2]
- Valmari and Franceschinis[3]
- PRISM

[1] Peter Buchholz. "Efficient computation of equivalent and reduced representations for stochastic automata". In: *International Journal of Computer Systems Science and Engineering* 15.2 (Apr. 2000), pp. 93–103.

[2] Salem Derisavi, Holger Hermanns, and William Sanders. "Optimal state-space lumping of Markov chains". In: *Information Processing Letters* 87.6 (Sept. 2003), pp. 309–315.

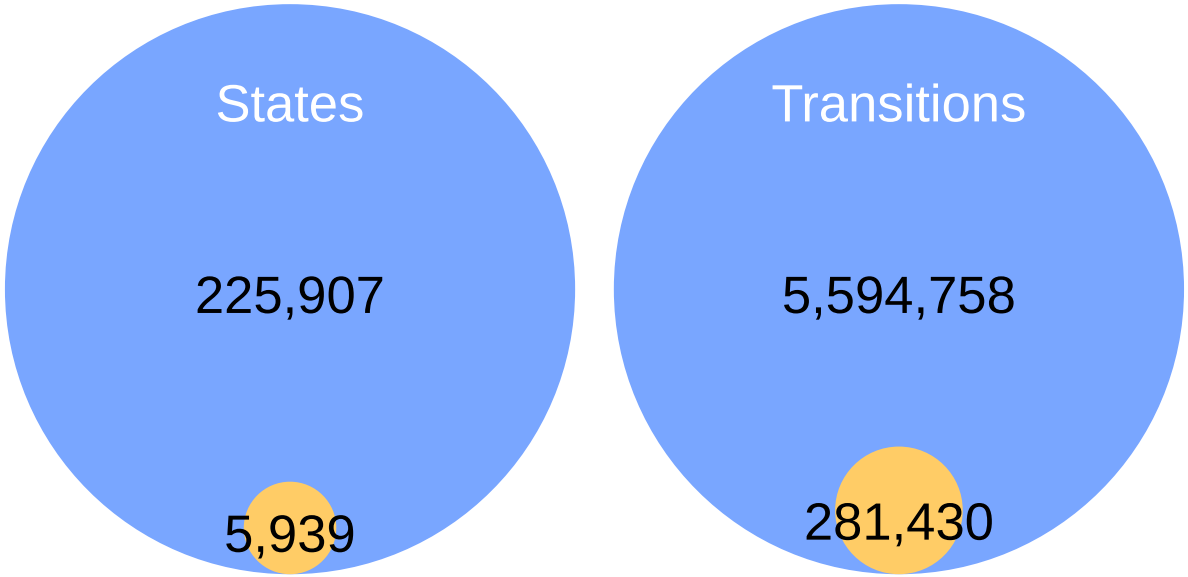
[3] Antti Valmari and Giuliana Franceschinis. "Simple $O(m \log n)$ Time Markov Chain Lumping". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Javier Esparza and Rupak Majumdar. Vol. 6015. Lecture Notes in Computer Science. Paphos, Cyprus: Springer-Verlag, Mar. 2010, pp. 38–52.

Experiments

Consider, for example, using the random binary search algorithm to find the element at the 50^{th} position in an array of size 86.

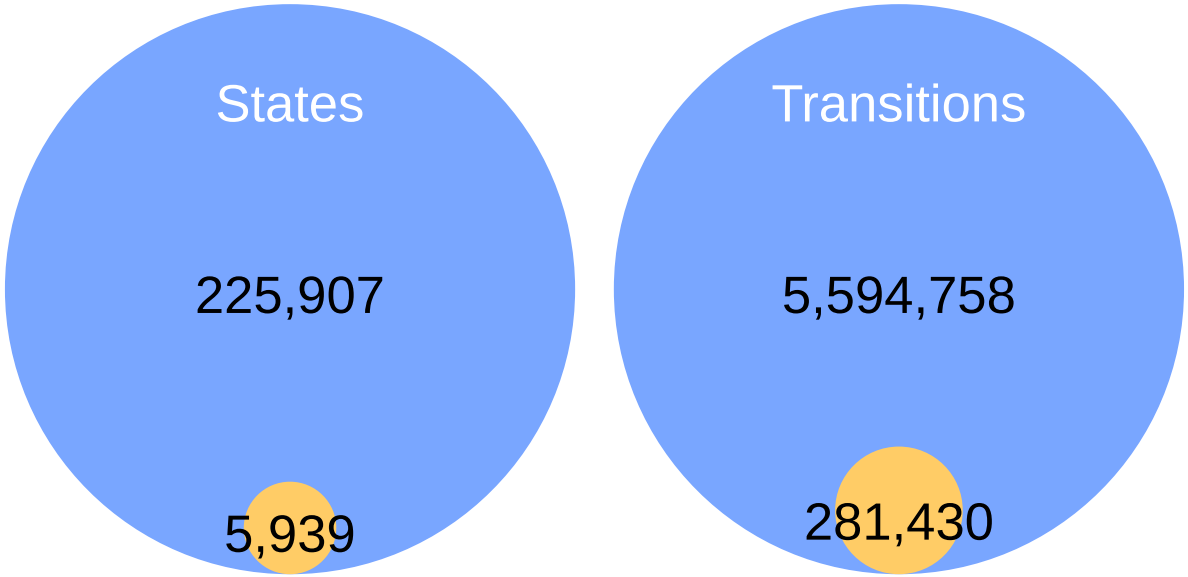
Experiments

Consider, for example, using the random binary search algorithm to find the element at the 50^{th} position in an array of size 86.



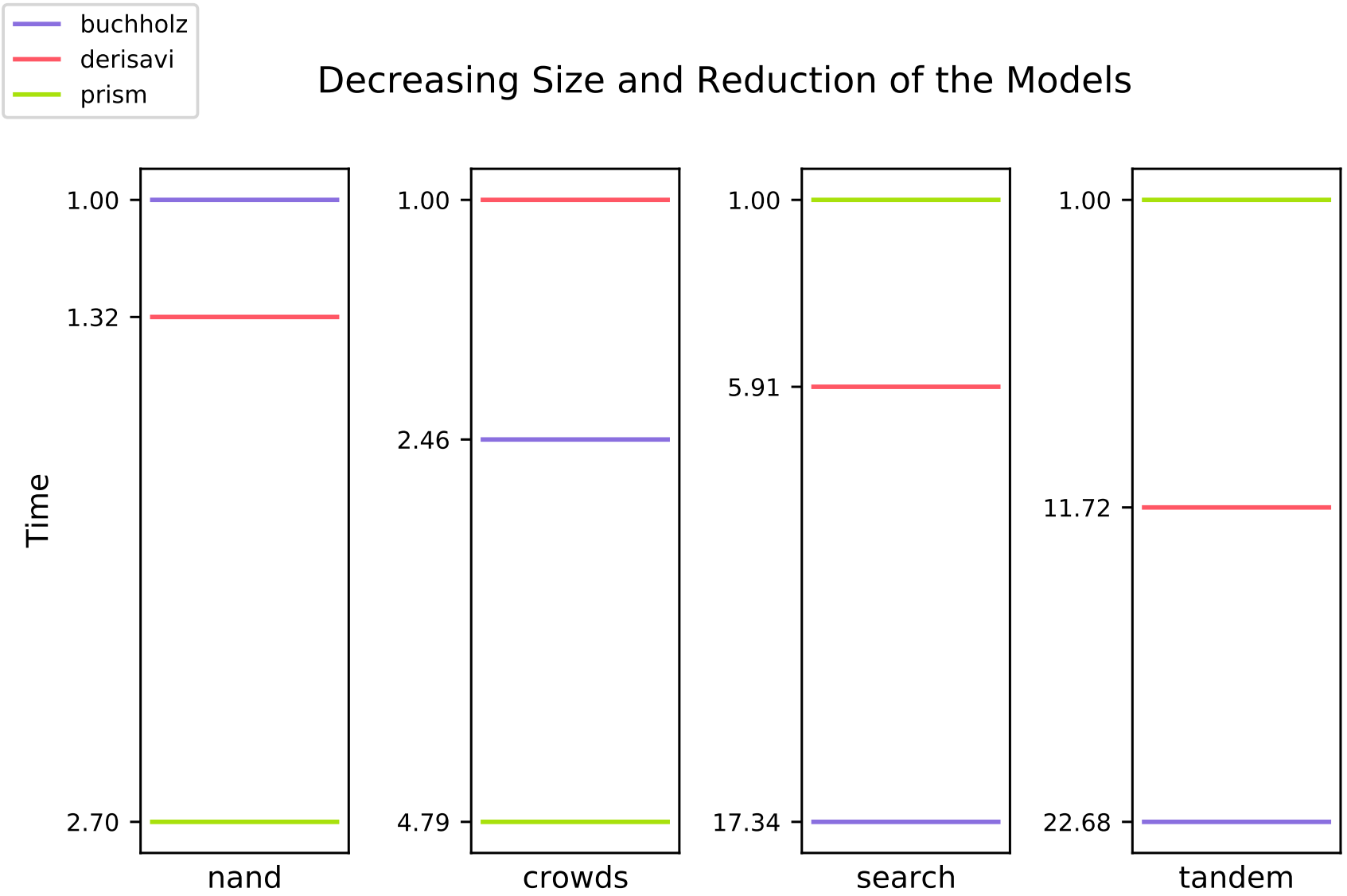
Experiments

Consider, for example, using the random binary search algorithm to find the element at the 50^{th} position in an array of size 86.



We varied the parameters and ran several experiments to compare the practical performance of these algorithms.

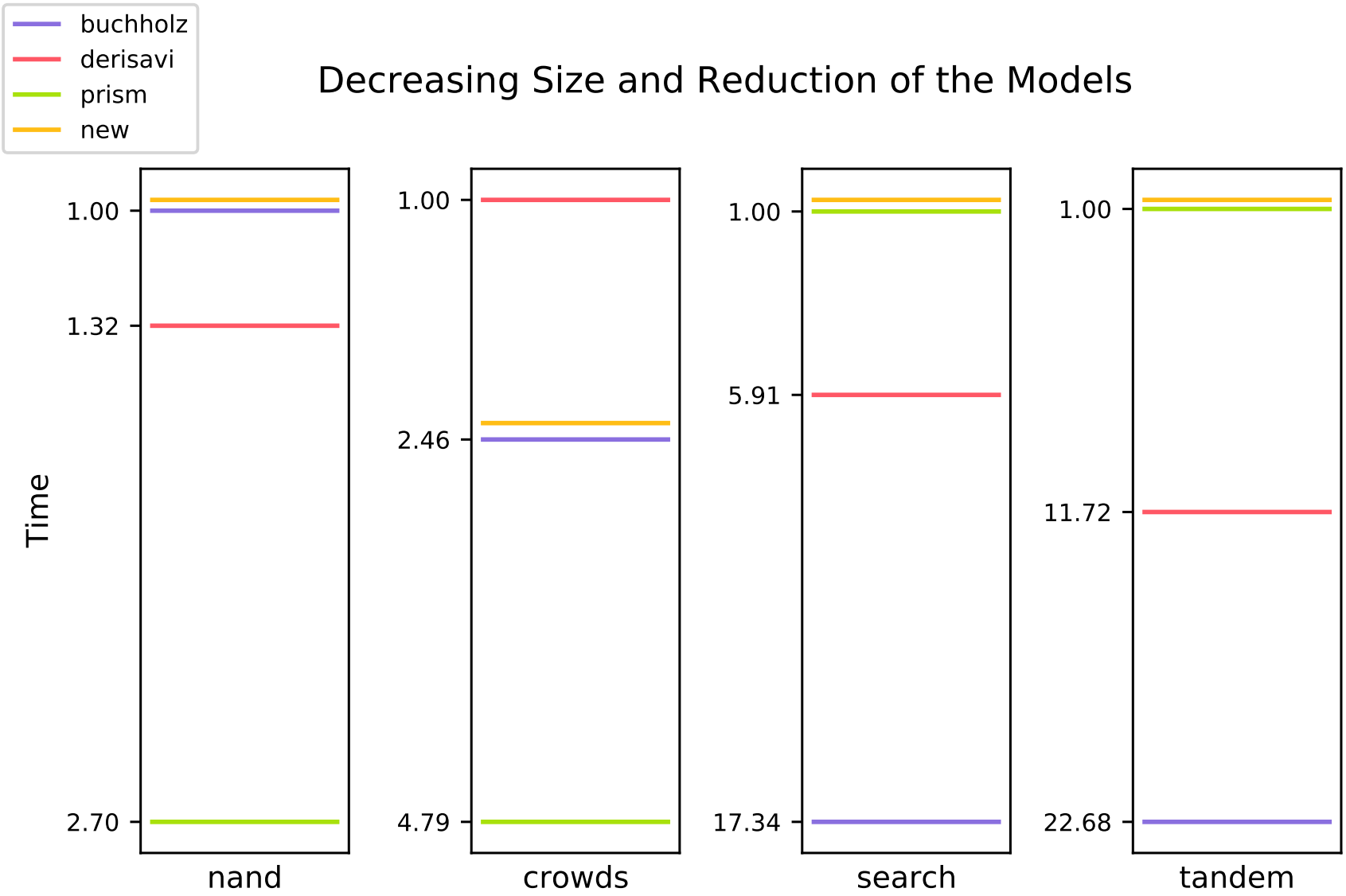
Results


buchholz
derisavi
prism
Decreasing Size and Reduction of the Models
Time
1.00
1.32
2.70
nand
1.00
2.46
4.79
crowds
1.00
5.91
17.34
search
1.00
11.72
22.68
tandem

Improvements

Over the last month we developed a new algorithm, not described in the thesis yet, that improves the PRISM algorithm by incorporating some ideas of the other algorithms.

Results


buchholz
derisavi
prism
new
Decreasing Size and Reduction of the Models
Time
1.00
1.32
2.70
nand
1.00
2.46
4.79
crowds
1.00
5.91
17.34
search
1.00
11.72
22.68
tandem

Future work

- Apply our tool to many more examples and analyze the results.
- Use rationals instead of reals for the transition probabilities, as we can compare rationals for exact equality.
- Run many more experiments and confirm for which types of labelled Markov chains each algorithm is best suited.
- Determine how the algorithms translate to other models of computation.

Thank You