Research Proposal - Analysis of Stochastic Behaviour in Sokoban

BRAM HAGENS, University of Twente, The Netherlands

Fig. 1. Sokoban level from the original game, released by Thinking Rabbit in 1982.

Sokoban is a challenging puzzle game for both humans and computers. By modifying the game to include stochastic elements, it can be used to benchmark different probabilistic modelling tools. This research introduces a novel tool that can generate probabilistic models of Sokoban levels. These models can then be used to compare various aspects of probabilistic model checkers and find the best performing one.

Additional Key Words and Phrases: Sokoban, probabilistic model checking, stochastic behaviour, analysis.

1 INTRODUCTION

Systems that exhibit probabilistic behaviour exist everywhere: computer networks, security protocols and traffic, among others. Using probabilistic model checking, a technique used to verify the correctness of a model of a stochastic system, properties of models of these systems can be verified which can help avoid states that result in unintended behaviour, such as a system crash or a livelock. As these systems get more complex, it is important to use tools that can verify these models efficiently.

Sokoban is a puzzle game where a player moves boxes around. Solving Sokoban levels has been proven to be an NP-hard problem[3]. Introducing stochastic behaviour into this game results in good benchmarks for these different probabilistic model checking tools. This will be done by disregarding the player's input, and by moving boxes in random directions, both according to pre-defined probabilities.

This research aims to generate probabilistic models from Sokoban levels and a set of probabilities in various formats, so that these models can be used to benchmark existing probabilistic model checking tools in competitions such as QComp[1].

[1]https://qcomp.org/

TScIT 37, July 8, 2022, Enschede, The Netherlands

2 BACKGROUND

2.1 Sokoban

Sokoban is a single-player transport puzzle game where the player pushes boxes around into predetermined locations. A level is completed when all boxes are pushed into the correct spots. The player can move and push boxes in 4 directions: up, down, left and right, and is contained by walls surrounding the level. Figure 2 shows an example of a simple Sokoban level. Not only is solving Sokoban puzzles NP-hard, but it is also proven to be PSPACE-complete[2], making it significantly more difficult to solve than many other NP-hard problems. As a result, there exist many levels that state-of-the-art solvers cannot solve.

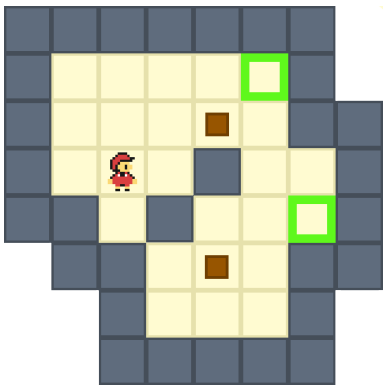


Fig. 2. An example Sokoban level. The gray tiles represent walls, the brown tiles represent boxes, and the green tiles are the targets.

2.2 Probabilistic model checking

Model checking is a way of verifying the correctness of requirements of models. These models are often represented as finite-state machines. Using model checking tools, one can run queries on these state machines to verify the correctness of certain aspects of the model. A model of a Sokoban level would include the player and box positions, and the solution can be found by querying a state where all boxes are in the solved position. Probabilistic model checking

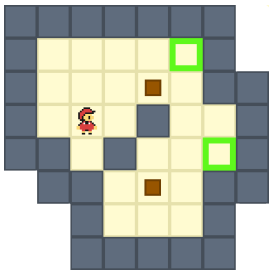
is a similar technique that allows for verifying properties of models that exhibit stochastic behaviour. An example of such a model would be the model of a Sokoban level, but all inputs are randomly determined. One can now query the probability that the model ends up in an unsolvable state (e.g a box is pushed in a corner and can no longer move) or what the average length is to complete the level.

2.3 Sok format

.sok files[2] are files that contain Sokoban levels. There are currently tens of thousands of user-created levels in this format freely available online. Apart from a few inconsistencies, the format is easy to parse and the large selection of levels makes it a good choice for this research. The contents of a .sok-file is depicted in Figure 3.

```
# # # # # #
#           . #
#       $   # #
#   @   #     #
# #   #     . #
  # #   $   # #
    #       #
    # # # # #
```

(a) ASCII representation of a .sok-file. '@' represents the player, '$' represents a box, '.' represents a target location, and '#' represents a wall.



(b) Graphical representation of a .sok-file. The brown tiles represent boxes, the green tiles represent target locations, and the tiles blocks represent walls.

Fig. 3. A .sok-file in both its ASCII and graphical representation.

3 RESEARCH QUESTIONS

(1) To what extent do the stochastic additions influence the solutions generated by the model checker?
(2) What is the effect of different encoding for the player and box positions on the run time and state-space of model checkers?

4 RELATED WORK

There exist many Sokoban solvers, such as Festival[3] and Sokolution[4]. The goal of this research however is not to create a solver for (a stochastic version of) Sokoban, but to create a model that will be solved by a model checker.

A project similar to this research has been implemented that generates Sokoban models, which are then used to benchmark three different model checking tools: LTSmin, DiVinE and nuXmv[6]. It solely researched non-stochastic models, whereas this research only considers stochastic models and probabilistic model checking tools. While there was a comparison between a simple and more optimized model in that research, there was no attempt to find the optimal encoding of the player and box positions like there is in this research.

[2]http://www.sokobano.de/wiki/index.php?title=Sok_format
[3]https://festival-solver.site/
[4]http://codeanalysis.fr/sokoban/

Similarly, another project focuses on generating models for various logic puzzles, including Sokoban, for LTSmin[4]. This research is different from the mentioned project for the same reasons as mentioned above.

5 METHOD

The research consists of two parts: (a) generating probabilistic models from existing Sokoban levels and a set of probabilities, and (b) experimenting with the generated models.

For part a., stochastic behaviour has to be introduced to the non-stochastic game Sokoban. This is done by defining the following probabilities:

(1) The probability that decides if the player's move is respected, or if a different move is selected. Additionally, probabilities for each movement direction are defined.
(2) The probability that a box moves when not pushed by a player. Additionally, probabilities for each movement direction are defined.

The probabilistic models are derived from .sok files, and the models are described by the PRISM language[5] and the JANI specification[1], so that a wide range of probabilistic model checkers can be targetted. The goal is to create a tool that automates this process: using a .sok file and a set of probabilities as input, it should be able to output a probabilistic model in either the PRISM language, JANI, or both.

Additionally, the position of the player and the boxes are represented in multiple ways. Depending on how these values are encoded, the performance of the model checkers may differ.

The generated models abide by the following transition rules:

(1) The player can walk in any of the four directions (up, down, left, right) if:
  (a) there is not a box or wall at the destination. This action updates the player's position to the destination.
  (b) there is a box at the respective destination and the tile behind the destination is not a wall nor box. This action moves the box one tile backwards, and updates the player's position to the destination.

These transition rules do mean that it is possible to end up in unsolvable states: a player is able to position boxes in such a way that they can no longer push them, or they can lock themselves into an area of the level without being able to escape. However, with the stochastic additions it is possible that these situations resolve themselves eventually, whereas for the non-stochastic version of Sokoban one of those scenarios would most certainly require the player to reset from the start.

Part b. involves experimenting with the generated models using existing probabilistic modelling tools. The tests will be conducted in virtual machines with identical specifications running an identical operating system. This creates a consistent playing field for the model checkers so that the run time performance can be measured fairly. Depending on the duration of the tests, they will be run multiple times to reduce inconsistencies caused due to measurement errors. A test is deemed completed once the target state is reached.

6 PLANNING

The planning is described in Table 1 and Figure 4, the table describing the deadlines imposed by the module, and the figure describing the planning required to reach those deadlines.

Date	Assignment/Task
8th May	Final proposal submission
10th May	Proposal peer feedback submission
26th June	Draft paper submission
3rd July	Final paper submission
8th July	Conference presentation

Table 1. Research project deadlines for handing in assignments and doing tasks.

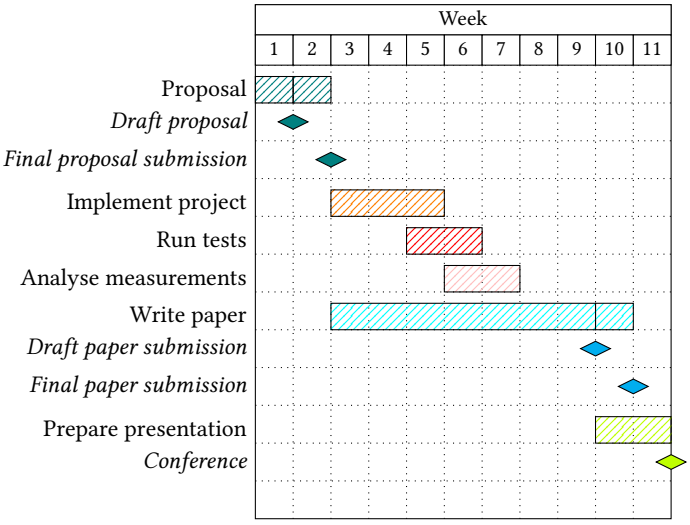


Fig. 4. Research project goals and milestones per week.

REFERENCES

[1] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. 2017. JANI: Quantitative Model and Tool Interaction. In *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)*, Axel Legay and Tiziana Margaria (Eds.). 151–168. https://doi.org/10.1007/978-3-662-54580-5_9

[2] Joseph Culberson. 1997. *Sokoban is PSPACE-complete.* https://doi.org/10.7939/R3JM23K33

[3] Dorit Dor and Uri Zwick. 1999. SOKOBAN and other motion planning problems. *Computational Geometry* 13, 4 (1999), 215–228. https://doi.org/10.1016/S0925-7721(99)00017-6

[4] Bram Kamies. 2016. Solving Logic Puzzles using Model Checking in LTSmin. In *24th Twente Student Conference on IT, Enschede, The Netherlands.*

[5] Marta Kwiatkowska, Gethin Norman, and David Parker. 2018. *Probabilistic Model Checking: Advances and Applications.* Springer International Publishing, Cham, 73–121. https://doi.org/10.1007/978-3-319-57685-5_3

[6] Kasper Wijburg. 2016. Comparing Model Checkers Through Sokoban. In *24th Twente Student Conference on IT, Enschede, The Netherlands.*