Last updated on Monday, March 28, 2022

Sokoban Assignment

Intelligent Search – Motion Planning in a Warehouse

Key information

- Submission due at the end of **Week 07** (Sunday 1st May , 23.59pm)
- Submit your work via Blackboard
- Recommended group size: three people per submission.
  Smaller groups are allowed (1 or 2 people OK, but completion of the same tasks is required).

Overview

Sokoban is a computer puzzle game in which the player pushes boxes around a maze in order to place them in designated locations. It was originally published in 1982 for the Commodore 64 and IBM-PC and has since been implemented in numerous computer platforms and video game consoles.

The screen-shot below shows the GUI provided for the assignment. While Sokoban is just a game, it models a robot moving boxes in a warehouse and as such, it can be treated as an automated planning problem. Sokoban is an interesting challenge for the field of artificial intelligence largely due to its difficulty. Sokoban has been proven NP-hard. Sokoban is difficult not because of its branching factor, but because of the huge depth of the solutions. Many actions (box pushes) are needed to reach the goal state! However, given that the only available actions are moving the worker up, down, left or right, the branching factor is small (only 4).

The worker can only push a single box at a time and is unable to pull any box. The boxes have individual weights. The weight of a box is taken into account when computing the cost of a push. The cost of an action is 1 + weight of the box being pushed (if any). That is, if the worker moves to an empty cell the action cost is 1. If the worker pushes a box that has a weight of 7, the action cost is 8 = 1+7.

The aim of this assignment is to design and implement a planning agent for Sokoban

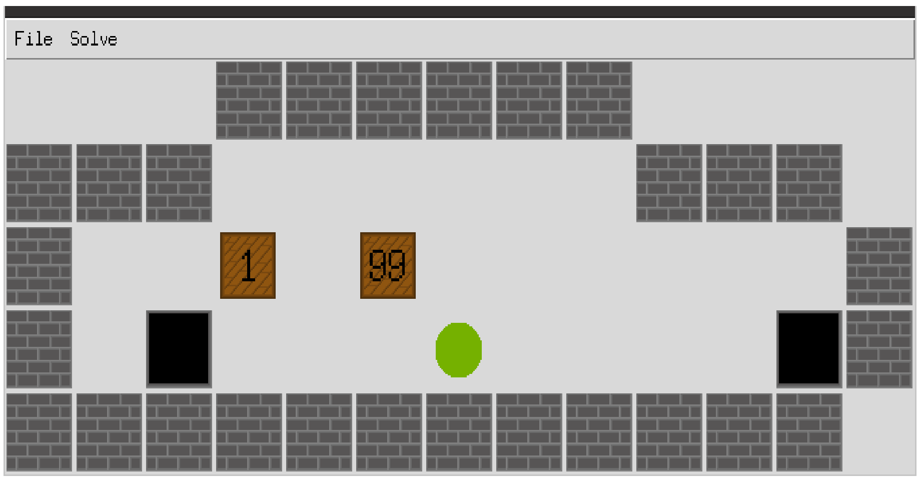


Illustration 1: Initial state of a warehouse. The green disk represents the agent/robot/player, the brown squares represent the boxes/crates. The black cells denote the target positions for the boxes. The left box has a weight of 1. The right box has a weight of 99.

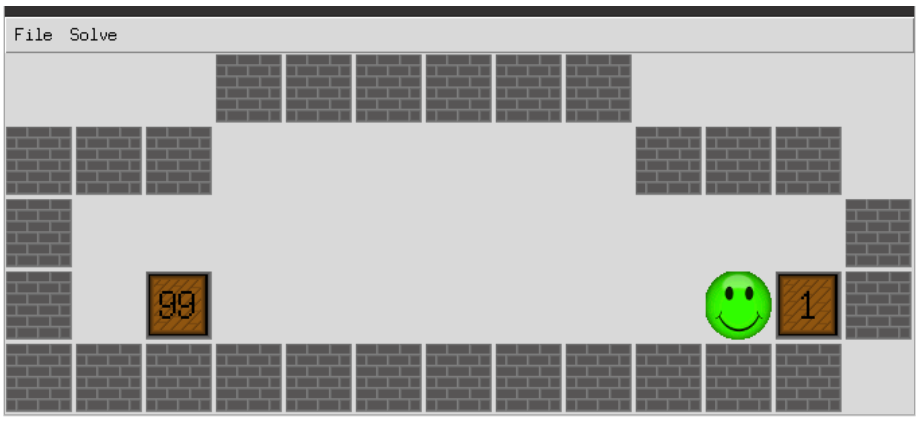


Illustration 2: Goal state reached: all the boxes have been pushed to a target position. Notice that the light box (weight of 1) has been moved to the target that was initially the farthest.

Approach

As already mentioned, Sokoban has a large search space with few goals, located deep in the search tree. However, admissible heuristics can be easily obtained. These observations suggest to adopt an informed search approach. Suitable generic algorithms include A* and its variations.

After playing a few games, you will realize that a bad move may leave the player in a doomed state from which it is impossible to recover. For example, a box pushed into a corner cannot be moved out. If that corner is not a goal, then the problem becomes unsolvable. We will call these cells that should be avoided ***taboo cells***. During a search, we should ignore the actions that move a box on a taboo cell.

You will implement a weighted variant of the classical Sokoban. In this variant, we assign an individual pushing cost to each box, whereas for the classical Sokoban, we simply count the number of actions executed.

In order to help you create an effective solver, you are also asked to implement a few auxiliary functions (see the python file provided, ***mySokobanSover.py***, for further details).

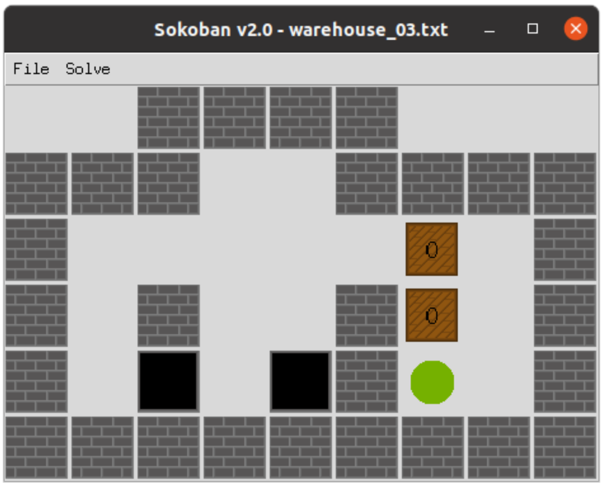
Warehouse representation in text files

To help you design and test your solver, you are provided with a number of puzzles. The first line of the file is the weights of the boxes indexed in the order the boxes are encountered when you scan the warehouse row by row. If the text file contains no weights, they are assumed to be 0.

The puzzles and their initial state are coded as follows in the text files,

- **space**, a free square
- ’**#**’, a wall square
- ’**$**’, a box
- ’**.**’, a target square
- ’**@**’, the player
- '**!**', the player on a target square
- '*****', a box on a target square

For example, the puzzle below is coded in a text file as



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

Files provided

- **search.py** contains a number of search algorithms and related classes.
- **sokoban.py** contains a class ***Warehouse*** that allows you to load puzzle instances from text files.
- **sokoban_gui.py** a GUI implementation of Sokoban that allows you to play and explore puzzles. This GUI program can call your planner function *solve_weighted_sokoban*
- **mySokobanSolver.py** code skeleton for your solution. You should complete all the functions located in this file. This is the only python file that you should submit.
- **sanity_check.py** script to perform very basic tests on your solution. The marker will use a different script with different warehouses. You should develop your own tests to validate your code
- A number of puzzles in the folder 'warehouses'

Your tasks

Your solution **has to comply to** the same search framework as the one used in the practicals. That is, you have to use the classes and functions provided in the file *search.py.*

All your code should be located in a single file called ***mySokobanSolver.py***. **This is the only Python file that you should submit**. In this file, you will find partially completed functions and their specifications. You can add auxiliary classes and functions to this file. When your submission is tested, it will be run in a directory containing the files *search.py* and *sokoban.py* and your file *mySokobanSolver.py*. **If you break this interface, your code will fail the tests!**

Deliverables

You should submit via Blackboard only two files

1. A **report** in **pdf** format **strictly limited to 4 pages in total** (be concise!) containing
   - One section that explains clearly your state representation, your heuristics, and any other important features needed to understand your solver.
   - Once section on your testing methodology. How did you validate your code? Did you create toy problems to verify that your code behave as expected?
   - Once section that describes the performance and limitations of your solver.
2. Your **Python file** **mySokobanSolver.py**

Marking Guide Overview

- **Report**: 5 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of explanations.
  - Figures and tables (use for explanations and to report performance).
- **Code quality**: 15 marks
  - Readability, meaningful variable names.
  - Proper use of Python idioms like dictionaries and list comprehension.
  - Header comments in classes and functions. In-line comments.
  - Function parameter documentation.

- **Functions of mySokobanSolver.py :** 20 marks

  The markers will run python scripts to test your function.
  - **my_team():** 1 mark
  - **taboo_cells()**: 3 marks
  - **check_elem_action_seq()**: 4
  - **solve_weighted_sokoban()**: 12 marks

Marking criteria

- **Report**: 5 marks
  - Structure (sections, page numbers), grammar, no typos.
  - Clarity of the text. No padding (paragraphs with no information)
  - Figures and tables (use for explanations and to report performance).

Levels of Achievement

5 Marks	**4 Marks**	**3 Marks**	**2 Marks**	**1 Mark**
+Report written at the highest professional standard with respect to spelling, grammar, formatting, structure, and language terminology. + Discussion about the admissibility or consistency of the heuristic used	+Report is very-well written and understandable throughout, with only a few insignificant presentation errors. +Testing methodology and experiments are clearly presented.	+The report is generally well-written and understandable but with a few small presentation errors that make one of two points unclear. +Clear figures and tables. +Clear explanation of the heuristics used	The report is readable but parts of the report are poorly-written, making some parts difficult to understand. +Use of sections with proper section titles.	The entire report is poorly-written and/or incomplete. +**The report is in pdf format.**

To get "i Marks", the report needs to satisfy all the positive items of the columns "j Marks" for all j≤i. For example, if your report is not in pdf format, you will not be awarded more than 1 mark.

- **Code quality**: 15 marks

Levels of Achievement

[13-15] Marks	[10-12] Marks	[7-9] Marks	[4-6] Marks	[1-3] Mark
+Code is generic, well structured and easy to follow. Use of auxiliary functions that help increase the clarity of the code.	+Proper use of data-structures. +No unnecessary loops. +Useful in-line comments. +Header comments are clear. The new functions can be unambiguously implemented by simply looking at their header comments.	+No magic numbers (that is, all numerical constants have been assigned to variables with meaningful names). +Each function parameter documented (including type and shape of parameters) +return values clearly documented	+Header comments for all new classes and functions. +Evidence of testing	Code has some structure but gives headaches to the markers.

To get "i Marks", the report needs to satisfy all the positive items of the columns "j Marks" for all j≤i.

Miscellaneous Remarks

- Do not underestimate the workload. Start early. You are strongly encouraged to ask questions during the practical sessions or use the assignment channel on Teams.
- Don't forget to **list all the members of your group in the report and the code**!
- Only one person in your group should submit the assignment. Feedback via Blackbloard will be given to every member of the group.
- Enjoy the assignment!

FAQ

Running time; Here are examples of running time of my model solution (no optimization!)

- warehouse_07.txt is medium difficulty,
  Analysis took 300.645556 seconds
  Solution found with a cost of 26
  ['Up', 'Up', 'Right', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Down', 'Right', 'Up', 'Down', 'Right', 'Down', 'Down', 'Left', 'Up', 'Down', 'Left', 'Left', 'Up', 'Left', 'Up', 'Up', 'Right']
- warehouse_09.txt is easy
  Analysis took 0.009575 seconds
  Solution found with a cost of 396
  ['Up', 'Right', 'Right', 'Down', 'Up', 'Left', 'Left', 'Down', 'Right', 'Down', 'Right', 'Left', 'Up', 'Up', 'Right', 'Down', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right', 'Up', 'Left', 'Down', 'Left', 'Up', 'Right', 'Up', 'Left']
- warehouse_47.txt is easy
  Analysis took 0.122561 seconds
  Solution found with a cost of 179
  ['Right', 'Right', 'Right', 'Up', 'Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Right', 'Down', 'Down', 'Left', 'Left', 'Left', 'Left', 'Up', 'Up', 'Right', 'Right', 'Up', 'Right', 'Right', 'Right', 'Right', 'Down', 'Left', 'Up', 'Left', 'Down', 'Down', 'Up', 'Up', 'Left', 'Left', 'Down', 'Left', 'Left', 'Down', 'Down', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right', 'Down', 'Right', 'Right', 'Up', 'Left', 'Left', 'Left', 'Left', 'Left', 'Left', 'Down', 'Left', 'Left', 'Up', 'Up', 'Up', 'Right', 'Right', 'Right', 'Up', 'Right', 'Down', 'Down', 'Up', 'Left', 'Left', 'Left', 'Left', 'Down', 'Down', 'Down', 'Right', 'Right', 'Up', 'Right', 'Right', 'Left', 'Left', 'Down', 'Left', 'Left', 'Up', 'Right', 'Right']
- warehouse_81.txt is easy
  Analysis took 0.170240 seconds
  Solution found with a cost of 376
  ['Left', 'Up', 'Up', 'Up', 'Right', 'Right', 'Down', 'Left', 'Down', 'Left', 'Down', 'Down', 'Down', 'Right', 'Right', 'Up', 'Left', 'Down', 'Left', 'Up', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Right', 'Up', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Down', 'Right', 'Down', 'Down', 'Left', 'Down', 'Down', 'Right', 'Up', 'Up', 'Up', 'Down', 'Left', 'Left', 'Up', 'Right']
- warehouse_147.txt is medium difficulty
  Analysis took 197.910769 seconds
  Solution found with a cost of 521
  ['Left', 'Left', 'Left', 'Left', 'Left', 'Left', 'Down', 'Down', 'Down', 'Right', 'Right', 'Up', 'Right', 'Down', 'Right', 'Down', 'Down', 'Left', 'Down', 'Left', 'Left', 'Up', 'Up', 'Down', 'Down', 'Right', 'Right', 'Up', 'Right', 'Up', 'Up', 'Left', 'Left', 'Left', 'Down', 'Left', 'Up', 'Up', 'Up', 'Left', 'Up', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right', 'Down', 'Right', 'Right', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Left', 'Left', 'Left', 'Left', 'Left', 'Left', 'Down', 'Down', 'Down', 'Right', 'Right', 'Up', 'Left', 'Down', 'Left', 'Up', 'Up', 'Left', 'Up', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right', 'Left', 'Left', 'Left', 'Left', 'Left', 'Down', 'Down', 'Down', 'Down', 'Right', 'Down', 'Down', 'Right', 'Right', 'Up', 'Up', 'Right', 'Up', 'Left', 'Left', 'Left', 'Down', 'Left', 'Up', 'Up', 'Up', 'Left', 'Up', 'Right', 'Right', 'Right', 'Right', 'Right', 'Down', 'Right', 'Down', 'Right', 'Right', 'Up', 'Left', 'Right', 'Right', 'Up', 'Up', 'Left', 'Left', 'Down', 'Left', 'Left', 'Left', 'Left', 'Left', 'Left', 'Right', 'Right', 'Right', 'Right', 'Right', 'Right', 'Up', 'Right', 'Right', 'Down', 'Down', 'Left', 'Down', 'Left', 'Left', 'Up', 'Right', 'Right', 'Down', 'Right', 'Up', 'Left', 'Left', 'Up', 'Left', 'Left']
- warehouse_111.txt is hard; did not complete overnight
- warehouse_5n is impossible
  Analysis took 1.504564 seconds
  Solution found with a cost of None
  Impossible

State representation; is a Warehouse a good state representation? Here are some clues.

- Think about what is static and what is dynamic in the problem.
- Where do you think static things should go? Problem instance or state?
- Where do you think dynamic things should go? Problem instance or state?

Where should we start the assignment?

- First, make sure that you understand the Problem classes that we saw in the pracs (the sliding puzzle and the pancake puzzle). Then implement in the following order the functions
    1. taboo_cells
    2. check_elem_action_seq
    3. solve_weighted_sokoban

Computation time and marking (CRA)

- With respect to the computation time, we will test the submissions on warehouses that require at most a couple of seconds with our solution.
- We will let the submissions run for one minute before aborting them. If they return within one minute, then the submission is considered fine with respect to time.
- The markers will run a test set on each submitted function.
- The test marks are binary: either correct or incorrect. If we run 10 tests for a function and you get 8 correct, your mark for this function will be 8/10.
- For functions like *solve_weighted_sokoban*, your returned solution does not have to be the same as our solution, but it needs to be of the same minimal cost to be considered correct.

How many submissions can I make?

- You can make multiple submissions. Only the last one will be marked.

How do I find team-mates?

- The Blackboard groups are only used to facilitate group formation. If you have already found team-mates, you do not need to register in a Blackboard group.
- When marking the assignment, **we simply look at the names that appear in the report and in the code. We ignore the Blackboard groups**.
- Use the assignment channel on MS Teams
- Make sure you discuss early workload with your team-mates. It is not uncommon to see groups starting late or not communicating regularly and eventually submitting separately bits of work.
- If your team mates are unresponsive, put deadlines and submit without them if appropriate.