Heuristics and Optimization

Lab assignment #2: *Logical Satisfiability and Heuristic Search*
Computer Science Engineering, 2018–2019

1. Goal

The goal of this assignment is to learn to model satisfiability (SAT) and heuristic search problems and to solve them using standard tools of each kind.

2. Problem statement

Paganizu [1] is an arcade videogame issued in the 90s, which is based on the classical game of *Sokoban* [2]. The main character, Alabama or just "Al" Smith is located in a *grid* maze as shown in Figure 1. We will address a simplified version of the game, where the goal is to help Al to collect all keys and then to escape the maze. Al can push stones (shown as balls) which otherwise block his path. There are also snakes which can fire Al if he is located in the same row than them unless there is an obstacle (either a stone or a key) in between. Hence, stones can be placed to avoid been fired by the snakes. To better understand the game it is highly recommended to play a few games [3], even if we will only address the simplified version described earlier.

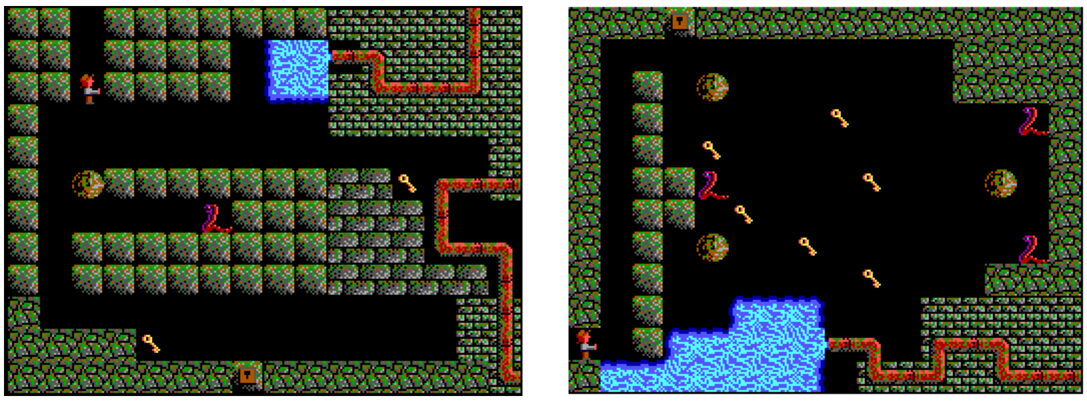
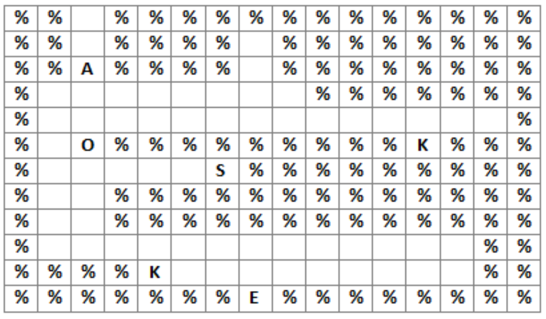


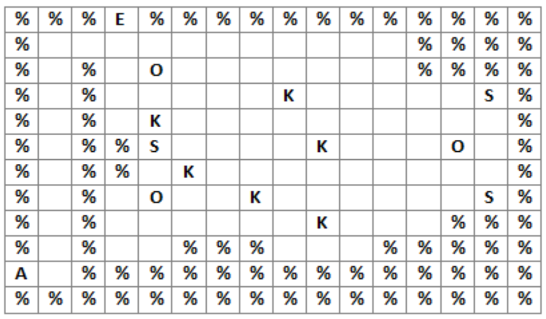
Figura 1: Example of a grid maze in the game of Paganitzu.

The maze is made of different cells, and each one can be either empty or can contain a single item: Al, a snake, a stone, a wall or obstacle that can not be traversed, keys or the exit. Each one of these items must be represented with a single character: 'A' for Al; 'S' for the snakes; 'O' for the stones; 'K' for the keys; 'E' for the exit; ' % for those obstacles that can not be traversed, and a blank space for empty cells. Figure 2 shows a schematic view of the same mazes shown in Figure 1 using this nomenclature.

[1] https://en.wikipedia.org/wiki/Paganitzu
[2] https://es.wikipedia.org/wiki/Sokoban
[3] https://www.playdosgames.com/online/paganitzu/



%	%		%	%	%	%	%	%	%	%	%	%	%	%	%
%	%		%	%	%	%		%	%	%	%	%	%	%	%
%	%	A	%	%	%	%		%	%	%	%	%	%	%	%
%									%	%	%	%	%	%	%
%															%
%		O	%	%	%	%	%	%	%	%	%	K	%	%	%
%						S	%	%	%	%	%	%	%	%	%
%			%	%	%	%	%	%	%	%	%	%	%	%	%
%			%	%	%	%	%	%	%	%	%	%	%	%	%
%														%	%
%	%	%	%	K										%	%
%	%	%	%	%	%	%	E	%	%	%	%	%	%	%	%



%	%	%	E	%	%	%	%	%	%	%	%	%	%	%	%
%												%	%	%	%
%		%		O								%	%	%	%
%		%						K						S	%
%		%		K											%
%		%	%	S					K				O		%
%		%	%		K										%
%		%		O			K							S	%
%		%							K				%	%	%
%		%			%	%	%				%	%	%	%	%
A		%	%	%	%	%	%	%	%	%	%	%	%	%	%
%	%	%	%	%	%	%	%	%	%	%	%	%	%	%	%

Figura 2: Representation of the grid mazes shown in Figure 1.

2.1. Part 1: SAT

A key question in the design of scenarios is to decide where to locate each item to make the game more appealing. To this end, it has been decided to automate the placement of Al and the snakes in a given grid maze —i. e., walls, stones, keys and the exit are already located. The constraints to take into account are listed below:

- Al and the snakes shall be placed on empty cells.
- Two snakes can not be located in the same row
- It is not allowed to locate a snake either in the same row or column than Al.

It is requested:

- To model the problem as a SAT instance, in Conjunctive Normal Form.
- Use JaCoP to develop a program that codes the model proposed to determine the location of Al and the snakes. The program shall be executable from a console or terminal with the following command:

  ```
  java SATPaganitzu < maze > < n >
  ```

  where:

  1. *maze*: it is the name of a file that contains a grid following the nomenclature described above. The grid should contain only walls, keys, the exit and stones. For instance, a valid filename is `lab1_part1.lab`.
  2. *n*: number of snakes to place in the grid.

  In case the instance is satisfiable, the problem must generate a file with the same grid maze given as input and the location of Al and the n snakes explicitly shown on it. The name of this file shall be the same used for the input file adding the extension `.output`. For instance, `lab1_part1.lab.output`. If the problem is not satisfiable, a message shall be shown on the standard console reporting it.

- It is required that each team generates its own test cases, i. e. its own grid mazes. They should be of different sizes and should be tested with a different number of snakes to locate on them.

2.2. Part 2: Heuristic Search

In this part it is requested to compute the shortest-path to collect all keys and to escape the maze, while not being fired by snakes. For this, Al can move between either horizontally or vertically adjacent cells provided that his path is not block by an item. In addition, Al can push the stones by been located next to them. Of course, the goal location of the stone shall be empty as well. When a stone is pushed, both Al and the stone move simultaneously in the same direction. Take into account that traversing cells have different costs as shown next:

- Empty cells can be traversed with a cost equal to 2 units.
- Pushing a stone has a cost equal to 4 units.

It is required to:

- Model the problem of computing the shortest path as a *state space*.
- Develop an implementation of the search algorithm A* that solves the problem. For this, implement two different informed and admissible heuristic functions. The program shall be executable from a terminal or console with the command:

  `java AstarPaganitzu` $<$ *maze* $>$ $<$ *heuristic* $>$

  where:

  - *maze*: name of the file that contains a textual representation of the grid maze in the format specified in Figure 2, for instance: `lab1_part2.map`.
  - *heurisic*: Name of the heuristic to use.

  The different names of the parameter *heuristic* must be clearly described in the final report and also in the online help provided by the application. The program shall generate two output files in the same directory where the grid maze is located and their name must start with the filename of the grid maze (including the extension) plus an additional extension as detailed below:

  - Output maze. It should describe the path followed by Al for collecting all keys and to get to the exit. First, the input grid maze must be shown and immediately below the path followed by Al, for example, $\{(0,0) \rightarrow (0,1) \rightarrow (1,1) \rightarrow\}\ldots$, where each location (x, y) refers to the cell in row x and column y. This file should be labeled with the extension `.output`, for instance: `lab1_part2.map.output`.
  - Statistics. This file shall contain various statistics of the search algorithm such as the overall running time, total cost, length of the path (or number of steps), overall number of expansions, etc. For instance:

    ```
    Overall running time: 145
    Total cost: 54
    Step length: 27
    Number of expansions: 132
    ```

    This file should be labeled with the extension `.statistics`. For example: `lab1_part2.map.statistics`.

- Suggest different test cases with different grid mazes among those generated in the first part and others specifically generated for this part, and solve them using the implementation provided.
- Perform an empirical evaluation comparing the two heuristic functions —number of expansions, overall running time, etc.
- **Optionally**, it is proposed to enhance the implementation provided allowing Al to move between two cells diagonally adjacent. Devise a heuristic that takes into account this move in addition to horizontal and vertical movements and perform a comparison with the previous results.

  This optional part will be scored with an additional point.

3. Requirements for the report

The report must be delivered in PDF format and it should consist of a maximum of 15 pages, including the cover, the table of contents and the back cover. It should contain, at least:

1. Brief introduction explaining the contents of the document.
2. Description of the models, discussing the decisions carried out.
3. Analysis of results.
4. Conclusions.

Important: Reports delivered in a format other than pdf will be penalized with 1 point.

The report **should not include source code** in any case.

4. Grading

This lab assignment will be graded over 10 points. Nevertheless, it is possible to get up one additional point if the extra part is submitted —hence, scoring for a maximum of 11 points.
To assure that the assignment is graded you must do at least the first part and the report.
The distribution of points will be as follows:

1. Part 1 (3 points)
   - SAT model (0.75 points)
   - Implementation of the model in JaCoP (1.25 point)
   - Resolution and analysis of different test cases (1 point)
2. Part 2 (7 points)
   - Problem Modeling (0.75 points)
   - Model Implementation (2.25 points)
   - Resolution and analysis of different test cases (1.75 point)
   - Comparative analysis (2.25 point)

When grading the model proposed, a correct model will result in half of the points. To obtain the whole score, the model must:

- Be correctly formalized in the report.
- Be simple and concise.
- Be properly explained (it should remain clear what is the reason for each component of the model).
- Justify in the report all the design decisions taken.

When grading the model implementation, a correct implementation will result in half of the points. The implementation provided must compile without trouble and should implement the specifications described herein. To obtain the whole score, the implementation must:

- Faithfully implement the model proposed above.
- Deliver source code correctly organized and commented. Names shall be descriptive.
- Contain a good deal of different test cases that prove the vaidity of your implementation.

When grading the results analysis, it will be positively valued to include in the report personal conclusions about the assignment difficulty and about what you learnt while carrying it out.

5. Submission

The deadline for submitting the assignment is December, 16 at 23:55. Only one member of each team must upload:

1. A unique `.zip` file to the submission site in 'Aula Global'. Note that the submission entry point is managed by Turniting.

   This file must be named `p2-NIA1-NIA2.zip`, where `NIA1` and `NIA2` are the last 6 digits of each student's NIA —padding with `0`s if necessary, e.g., `p2-054000-671342.zip`.

2. The report, in pdf format, that should be called `NIA1-NIA2.pdf` (after properly substituting the NIA of each student and padding with `0`s if necessary, e.g., `054000-671342.pdf`) through another submission entry point. Note that this submission point is also managed by Turnitin.

Uncompressing the `.zip` file must generate, at least, two directories called "`part-1`" and "`part-2`". If the optional part is submitted also, then a third directory should be generated with the name "`part-3`".

Uncompressing the `.zip` file must also generate the report (even if it was submitted additionally through a different entry point), in pdf format, and it shall be located at the root of these directories. The solutions should be included in their corresponding directory.

Important: If the requirements for the submission are not carefully observed, the practice will be penalized with 1 point.