LINGI2261: Artificial Intelligence
Assignment 1: Solving Problems with Uninformed Search

François Aubry, Michael Saint-Guillain, Thanh Khong Minh, Yves Deville
September 25, 2017

Guidelines

- This assignment is due on **Wednesday 11 October, 18:00**.
- *No delay* will be tolerated.
- *Document* your source code (at least the difficult or more technical parts of your programs). Python docstrings for important classes, methods and functions are also welcome.
- Indicate clearly in your report if you have *bugs* or problems in your program. The online submission system will discover them anyway.
- Copying code or answers from other groups (or from the internet) is strictly forbidden. Each source of inspiration must be clearly indicated.
- Source code shall be submitted on the online *INGInious* system. Only programs submitted via this procedure will be graded. No report or program sent by email will be accepted.
- Respect carefully the *specifications* given for your program (arguments, input/output format, etc.) as the program testing system is *fully automated*.

Deliverables

- The following files are to be submitted on *INGInious* inside the *Assignment 1* task(s):
  - `kubmic.py`: The file containing your implementation of the Kubmic puzzle solver. Your program should take one argument, namely the instance file's path. It should print a minimal solution to the problem to the standard output, respecting the format described further. The file must be encoded in **utf-8**.
  - `report_A1_group_XX.pdf`: Answers to all the questions in a single report, named. Remember, the more concise the answers, the better.

Anti plagiat charter

As announced in the class, you'll have to electronically sign an anti plagiat charter. This should be done ***individually*** in the **INGInious** task entitled *Assignment 1: Anti plagiat charter*. Both students of a team must sign the charter.

Important

- For the implementation parts of the assignments, you are required to use *Python 3*
- Python 3.4.3 can be downloaded at `http://python.org/download/`
- On your computer, after installing Python 3, you will be able to launch your programs using *python3 <your_program>*
- In the labs, you can find Python 3 under */opt/python3/bin/python3*. You can of course add */opt/python3/bin* to your PATH to be able to launch your programs using *python3 <your_program>*.
- Python 3 documentation can be found at `http://docs.python.org/py3k/`
- The assignments must be submitted via the *INGInious* tool. You first have to *create groups of two*. Only then will you be granted access to the submission tool. The procedure to register and submit your hard work is described below. *Don't wait until the last minute* to create your group and to familiarize with the tool.
- If you want to ask us questions, we are on Tuesdays from 1:30pm to 4:30pm, in office A 377.20 at Réaumur, third floor. *For general questions, do not hesitate to use the forum set up espacially for this assignment, on Moodle*.

Submitting your programs

Python programs must be submitted on the INGInious website: `https://inginious.info.ucl.ac.be`. In order to do so, you must first create groups of two. To do so, assign yourself in an available group on the INGInious page of the course. Inside INGInious, you can find different courses. Inside the course 'LINGI2261: Artificial Intelligence', you will find the tasks corresponding to the different assignments due for this course. The task at hand for this assignment is *Assignment 1: kubmic*. In the task, you can submit you program (one python file containing the kubmic solver, encoded in utf-8). Once submitted, your program will immediately be evaluated on the set of given instances and also on a hidden set of instances. The results of the evaluation will be available directly on INGInious. You can, off course, make as many submissions as you want. For the grade, only the last fully correct submission (or the last submission if no fully correct submission has been made) will be used. You thus know the grade you will receive for the program part of the assignment! If you have troubles with INGInious, use the dedicated forum on Moodle.

Important

Although your programs are graded automatically, they will still be checked for plagiarism !

1 Python AIMA (3 pts)

Many algorithms of the textbook "AI: A Modern Approach" are implemented in Python. Since you are required to use Python 3, we will provide a Python 3 compliant version of the AIMA library. The Python modules can be downloaded on Moodle in *Documents (S2)*. All you have to do is to decompress the archive of aima-python3 and then put this directory in your python path: **export PYTHONPATH=path-to-aima-python3**. As we will use our own version of the library to test your programs, no modification inside the package is allowed.

The objective of these questions is to read, understand and be able to use the Python implementation of the *uninformed methods* (inside *search.py* in aima-python3 directory).

Questions

1. In order to perform a search, what are the classes that you must define or extend? Explain precisely why and where they are used inside a *tree_search*. Be concise! (e.g. do not discuss unchanged classes).
2. Both *breadth_first_graph_search* and *depth_first_graph_search* are making a call to the same function. How is their fundamental difference implemented?
3. In the *expand* method of the class *Node* what is the advantage of using a *yield* instead of building a list and returning it afterwards?
4. What is the difference between the implementation of the *graph_search* and the *tree_search* methods and how does it impact the search methods?
5. What kind of structure is used to implement the *closed list*? What are the methods involved in the search of an element inside it? What properties must thus have the elements that you can put inside the closed list?
6. How technically can you use the implementation of the closed list to deal with symmetrical states? (hint: if two symmetrical states are considered by the algorithm to be the same, they will not be visited twice)

2 The Kubmic Problem (17 pts)

"Kubmic" is a two-dimensional Rubik's Cube like game, consisting in restituting some goal pattern from an initial configuration of the cube. In the two-dimensional version Kubmic, both the initial and target configurations are $n \times n$ grids, composed on n^2 tiles, as shown on Figure 1. By sliding either a row or a column of blocks, one obtains a new pattern. The objective is to reach the goal pattern by applying a minimum number of such actions. A free online version of the game is available at `http://www.zebest-3000.com/jeux/jeu-5373.html`.

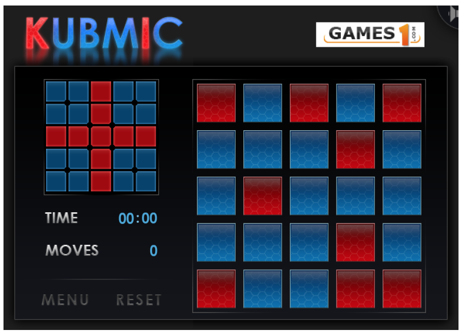
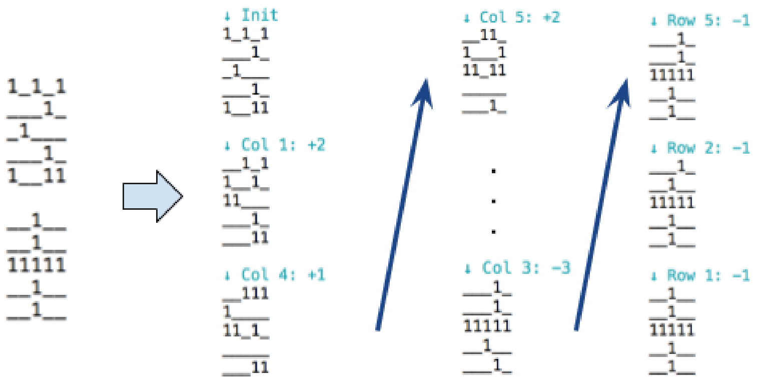


Figure 1: Screenshot of Kubmic game interface. On the left hand side is shown the targeted goal pattern to be reached. On right and side is the current (here, initial) state of the square.

The objective is then to reach the goal pattern by moving the blocks in different directions, whilst performing as few moves as possible. We provide a set of 10 problems, composed of 5 easy 4×4 instances and of the first 5 problems of 5×5 sized instances from the online game. Also, don't forget that we will also use hidden instances for the evaluation.

Each problem is described in an instance file, containing both the initial and goal configurations of the grid. We use ASCII symbols in order to represent the colored blocks we see in Figure 1. The output of the program should be a minimal sequence of every intermediate grids, represented in the same way, starting with the initial state and finishing with the goal state. Figure 2 shows the content of instance file `instances/b02` together with an overview of a possible corresponding solution output. Note that some instances involve more than two different block colors, as shown in Figure 3.



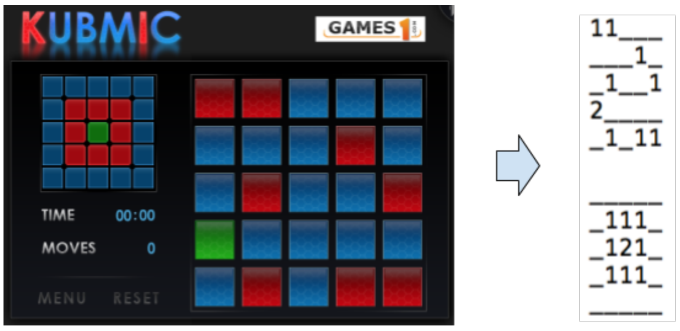
```
1_1_1
___1_
_1___
___1_
1__11

__1__
__1__
11111
__1__
__1__
```

```
↓ Init
1_1_1
___1_
_1___
___1_
1__11

↓ Col 1: +2
__1_1
1__1_
11___
___1_
___11

↓ Col 4: +1
__111
1____
11_1_
_____
___11
```

```
↓ Col 5: +2
__11_
1___1
11_11
_____
___1_

.
.
.

↓ Col 3: -3
___1_
___1_
11111
_1___
___1_
```

```
↓ Row 5: -1
___1_
___1_
11111
__1__
__1__

↓ Row 2: -1
___1_
__1__
11111
__1__
__1__

↓ Row 1: -1
__1__
__1__
11111
__1__
__1__
```

Figure 2: Left: content of instance file `instance/b02`. Two grids, corresponding to the initial and the goal states respectively and separated by an empty line. Right: example of program output. Each grid is preceeded by *exactly one* commenting line, and *one* empty line (the content of those two lines won't be taken into account by the testing system).



```
11___
___1_
_1__1
2____
_1_11

_____
_111_
_121_
_111_
_____
```

Figure 3: Left: online game interface, level 04. Right: content of corresponding problem instance file `instances/b04`.

You will implement at least one class *Kubmic(Problem)* that extends the class *Problem* such that you will be able to use search algorithms of AIMA. A small template (*kubmic.py*) is provided in the resources for this problem. Before diving into the code, we recommend you to first have a look at the questions below that need to be answered in your written report.

Questions

1. **Describe** the set of possible actions your agent will consider at each state. Considering a problem of size $n \times n$, what would be your branching factor?
2. **Problem analysis.**
   (a) Consider a *tree-search* strategy. Provided a 4×4 Kubmic problem and the sequence of actions mentioned above, how many nodes the *depth-first tree-search* algorithm must consider in order to enumerate all the states normally reachable in at most ≤ 3 moves? And under *breadth-first tree-search*?
   (b) Consider now a *breadth-first graph-search* algorithm, and propose an *upper bound* on the same number of nodes. The bound must be lower than the numbers you provided in (a). Explain.
   (c) What do you conclude? How do you expect these approaches to behave?
3. **Implement** a Kubmic solver in Python 3. You shall extend the *Problem* class and implement the necessary methods -and other class(es) if necessary- allowing you to test the following four different approaches:
   - *depth-first tree-search (DFSt)*;
   - *breadth-first tree-search (BFSt)*;
   - *depth-first graph-search (DFSg)*;
   - *breadth-first graph-search (BFSg)*.

   Your file must be named `kubmic.py`.
4. **Experiments** must be realized (*not yet on INGInious!* use your own computer or one from the computer rooms) with the provided 10 instances.
   (a) Report in a table the results on the 10 instances for depth-first and breadth-first strategies on both tree and graph search (4 settings above). Run each experiment for a maximum of 3 minutes. You must report the time, the number of explored nodes and the number of moves in the solution.
   (b) Try out the *iterative deepening search* algorithm provided by Python AIMA (still inside `search.py`) and report the results on the table as well. Intuitively, how would you improve this algorithm to get better results ?
   (c) Consider the *bidirectional search* algorithm described at the end of section 3.4 of AIMA; is it applicable to your problem? If so, implement it in `kubmic.py` as well and report its results in the table. Is your algorithm optimal? (explain why it should be and verify it experimentally)
5. **Submit** your program (the `kubmic.py` file, encoded in **utf-8**) on INGInious. According to your experimentations, it must use the *optimal* algorithm that leads to the best results. Your program must take as only input the path to the instance file of the problem to solve, and print to the standard output a solution to the problem satisfying the format described in Figure 2.
   Under INGInious (only 30s timeout per instance!), we expect you to solve at least 10 out of the 15 ones. Solving more than 12 would give you a bonus!
6. **Conclusion and further work.**
   (a) Are your the experimental results consistent with the conclusions you drew based on your problem analysis (Q2)?
   (b) Which algorithm seems to be the more promising? Do you see any improvement directions for this algorithm? (Note that since we're still in uninformed search, *we're not talking about informed heuristics*).