Lab class: uninformed search

Steven Schockaert

Week 2

At the end of the exercise class, please post a message on the discussion board on learning central, indicating which parts of the assignment you have been able to solve and what problems you faced. Feedback in the subsequent lecture will address the issues that were highlighted in this way.

Getting Started

Download `lab1-source.zip` from learningcentral. In the archive, you will find an implementation of breadth-first search and of A*. It uses the following classes and interfaces:

BreadthFirstSearch is the main class in which the breadth-first search algorithm is implemented.

Astar is the main class in which A* is implemented.

Node represents a node of the search tree. It contains a reference to a `State` (which encodes the state of the problem to be solved), as well as a pointer to its predecessor in the search tree. This is important to retrieve the complete path from the initial state to a goal state once a solution has been found, and to avoid revisiting states that have been considered before when expanding nodes. The class `Node` also provides methods to access the cost of the partial solution from the initial state.

AstarNode is an extension of Node which implements `Comparable`. In particular, the implementation of `compareTo` in the class determines in which order the nodes in the frontier are ordered, when using A*.

State is an interface specifying what methods need to be implemented to encode how the states of a problem are represented.

Action is an interface specifying what methods need to be implemented to encode which actions are available to move from one state to another.

The former four classes are generic. When solving a particular problem, it suffices to provide implementations of the interfaces `State` and `Action`. To illustrate how this works in practice, an implementation of the 8-puzzle has been provided.

To compile the code command-line, you can use:

```
javac */*.java
```

To solve the 8-puzzle, simply execute the main method of the `BreadthFirstSearch` class:

```
java search/BreadthFirstSearch
```

Alternatively, you can also import the code into your IDE of choice (e.g. NetBeans or Eclipse).

Implementing uninformed search methods

1. Implement the following two uninformed search methods:

   - Depth-first search
   - Iterative deepening

   Depth-first search can be implemented by changing the data structure used to represent the frontier. Iterative deepening requires some further modifications to the code of `BreadthFirstSearch`, but does not require you to alter any of the other classes.

2. The implementation of breadth-first search which was provided uses cycle checking as its pruning method. Modify the implementation such that it uses multiple path pruning instead. This requires you to maintain a `HashSet` with all states that have already been encountered. Before processing a new state, you should then verify that it has not been processed yet, i.e. that it is not contained in the closed list.