**Introduction to Artificial Intelligence**

Prof. Dr. Sven Seuken, Clemens Büchner, Ermis Soumalias
Nina Emmermann, Stephan Richard Saxer

University of Zürich
Spring Semester 2024

# Exercise Sheet 2

**Due: March 5, 2024**

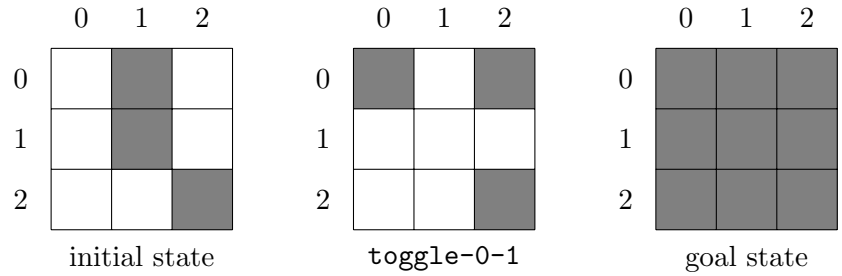
Points total: 20 marks

## Exercise 2.1 – Formalizing Problems

**4 marks**

In the *lights-out* problem we have a grid of $n \times m$ cells, which can be *on* and *off*. Action `toggle-i-j` changes not only the status of the cell in row $i$ and column $j$, but also of the cells above, below, left and right of it (if they exist). The action cost is equal to the number of cells that switch status, e.g., for a cell in a corner the action cost is 3, while for a cell in the center it is 5. The goal is to turn all the lights off.

The following figure shows an example instance where a white background represents a cell that is on, and a gray background a cell that is off:




Formalize the state space of the above lights-out problem instance. Specify all parts of the state space, i.e., the set of states, the set of actions, the cost function, the set of transitions, the initial state and the set of goal states.

## Exercise 2.2 – Search Algorithms: Theory

**5 marks**

Are the following statements true or false? Justify your answer in one sentence.

(a) A finite state space can result in an infinite search tree. (1 mark)

(b) Depth-first (tree) search is never useful because it is not complete. (1 mark)

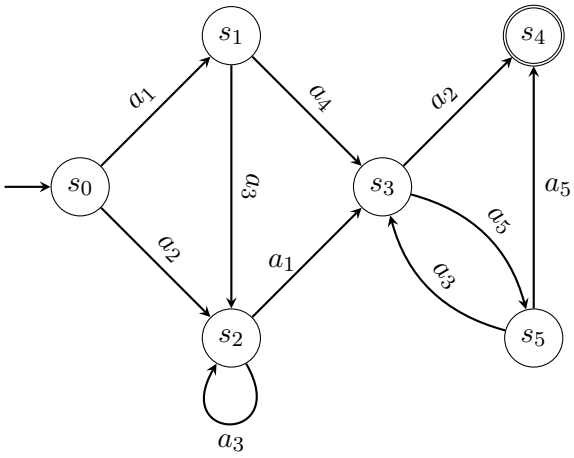
(c) Breadth-first (graph) search and uniform-cost (graph) search perform identically on problems where all actions have the same cost. (1 mark)

(d) Uniform cost search can be used to compute the shortest path between two cities, when we know the distences of the roads that connect pairs of citites. (1 mark)

(e) Let $\mathcal{S} = \langle S, A, \mathit{cost}, T, s_I, S_\star \rangle$ be a state space with actions $a_1, a_2 \in A$ such that $\mathit{cost}(a_1) = \mathit{cost}(a_2)$. We can define a state space $\mathcal{S}' = \langle S, A', \mathit{cost}', T', s_I, S_\star \rangle$ where $A' = A \setminus \{a_1, a_2\} \cup \{a_{\text{new}}\}$, $\mathit{cost}'(a) = \mathit{cost}(a)$ for all $a \in A \setminus \{a_1, a_2\}$, $\mathit{cost}'(a_{\text{new}}) = \mathit{cost}(a_1)$, and $T'$ is similar to $T$ except all $\langle s, a_1, s' \rangle$ and $\langle s, a_2, s' \rangle$ are replaced with $\langle s, a_{\text{new}}, s' \rangle$. (1 mark)

## Exercise 2.3 – Search Algorithms: Practice **6 marks**

Consider a state space with states $S = \{s_0, s_1, s_2, s_3, s_4, s_5\}$, initial state $s_I = s_0$, goal state $S_\star = \{s_5\}$, actions $A = \{a_1, a_2, a_3, a_4, a_5\}$ with cost $cost(a_i) = i$ for $i = 1, \ldots, n$ and transitions as given by the following graphical representation:




In the following tasks, you are asked to perform two kinds of uninformed search algorithms. If a state has multiple successors, expand the states with larger indices first (e.g., if there were a state with successors $s_1$, $s_3$ and $s_5$, you would expand the search node with state $s_5$ before the one with state $s_3$ before the one with $s_1$).

(a) Perform breadth-first (graph) search and draw the resulting search tree. Mark duplicate nodes that are getting pruned (due to the graph variant). (2 marks)

(b) Just by looking at the graph, what can you say about the uniqueness and optimality of the solution found in breadth-first (graph) search? You can assume that all edges hold equal weight. (1 mark)

(c) Perform depth-first (tree) search and draw the resulting search tree. (2 marks)

(d) Just by looking at the graph, what would happen in depth-first (tree) search if we were to expand states with the lower index before states with lower ones in the case of multiple successors? (1 mark)

## Exercise 2.4 – Search Algorithms: Programming **5 marks**

Implement uniform-cost search in the framework provided in `sheet02-programming.zip`. It contains an implementation of search problems as well as a generic search interface and implementations of breadth-first and depth-first search. Use the template `uniform_cost_search.py` for your implementation. You can use `breadth_first_search.py` and `depth_first_search.py` as examples. You will only need to submit the file `uniform_cost_search.py`. *Hint: In the class `UCSearchNode` implement the "less than" operator for the g values in order for the priority queue to handle nodes as items.*

Test your implementation using the file `experiment.py`. Make sure the plan cost (i.e., the length of the state sequences) found by your implementation of uniform-cost search are identical to those of breadth-first search in search problems where all actions have cost 1.

**Submission rules:**

- Exercise sheets must be submitted in groups of three students. Please submit a single copy of the exercises per group (only one member of the group does the submission).
- Create a single PDF file (ending `.pdf`) for all non-programming exercises. Use a file name that does not contain any spaces or special characters other than the underscore "_". If you want to submit handwritten solutions, include their scans in the single PDF. Make sure it is in a reasonable resolution so that it is readable, but ensure at the same time that the PDF size is not astronomically large. Put the names of all group members on top of the first page. Make sure your PDF has size A4 (fits the page size if printed on A4). Submit your single PDF file to the corresponding exercise assignment in moodle.
- For programming exercises, only create those code text files required by the exercise and only the files that contain changes, i.e. if we provide additional files that don't need to be changed, then omit those files from your submission. Put your names in a comment on top of each file. Make sure your code compiles and test it. Code that does not compile or which we cannot successfully execute will not be graded. Create a ZIP file (ending `.zip`, `.tar.gz`, or `.tgz`; *not* `.rar` or anything else) containing the code text file(s) (ending `.py`) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly.
- Do not upload several versions to moodle, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.