Foundations of Artificial Intelligence

M. Helmert
T. Keller
Spring Term 2018

University of Basel
Computer Science

Exercise Sheet 5

Due: April 18, 2017

Exercise 5.1 (1+1 marks)

Show that the following two statements do not hold in general:

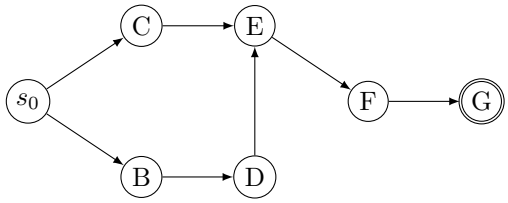
(a) If a heuristic is admissible, it is also consistent.

(b) If a heuristic is consistent, it is also admissible.

Hint: For each statement it suffices to present a counterexample consisting of a state space and a heuristic operating on this state space for which the left side of the statement holds, but the right side does not. In both cases, counterexamples containing no more than three states can be found.

Exercise 5.2 (1.5+1 marks)

(a) Consider the problem where the goal is to find the shortest path from A to G in the following directed graph under unit cost, i.e., where each transition incurs a cost of 1.



Show that A* *without* reopening used with an admissible but *inconsistent* heuristic can find suboptimal solutions: first provide a heuristic with the required properties for the depicted problem, then use A* without reopening to solve the problem and show that the found solution is not optimal. Specify the expansion order of the nodes with their g-, h-, and f-values.

(b) Which part of the proof of optimality of A* without reopening (chapter 19) becomes invalid if using an inconsistent heuristic? Justify your answer.

Exercise 5.3 (1.5+1.5 marks)

(a) A *vertex cover* of a given input graph G is a subset of the vertices of G such that every edge of G has at least one of its end points in the subset. Formalize the combinatorial optimization problem (COP) of finding a vertex cover of minimal size. Is the COP a pure search problem, a pure optimization problem, or a combined search and optimization problem?

(b) A *Hamilton path* of a directed graph G is a path through G that visits each vertex exactly once. The *longest Hamilton path* of a directed graph G with (positive) weighted edges is the Hamilton path with the maximal sum of edge weights.

Formalize the combinatorial optimization problem (COP) of finding a longest simple path in a graph that is **fully connected**, i.e., there is an edge between all pairs of vertices. Is the COP a pure search problem, a pure optimization problem, or a combined search and optimization problem?

Exercise 5.4 (1.5+1.5+0.5+1 marks)

The task in this exercise is to write a software program. We expect you to implement your code on your own, without using existing code (such as examples you find online). If you encounter technical problems or have difficulties understanding the task, please let us know.
Download the file `hill-climbing.tar.gz` from the website of the course. It contains an incomplete implementation of hill climbing search for the 8 queens problem that was presented in the lecture.

(a) Implement the heuristic for the 8 queens problem that is presented on Slide 22 of Chapter 20 (print version), where the heuristic value is equal to the number of pairs of queens threatening each other. To do so, implement the function `public int h(Configuration _conf)` in the file `EightQueensProblem.java`.

(b) Implement hill climbing in the function `protected SearchResult search()` in the file `HillClimbing.java`. Since our heuristic is such that smaller values are better, we are considering a minimization variant here, so adapt the function presented on Slide 20 of Chapter 20 (print version) accordingly. Break ties among neighbors with minimal heuristic value uniformly at random. Note that `protected SearchResult search()` returns a `SearchResult` object, which contains information if hill climbing found a solution and on the number of steps.

(c) Test your implementation by verifying the statement on Slide 23 of Chapter 20 (print version), which states that hill climbing with a random initialization finds a solution in around 14% of the cases. You can compile and run your code with `javac HillClimbing.java` followed by the command `java HillClimbing 8queens`.

(d) Copy your hill climbing implementation into a new file `HillClimbingWithStagnation.java`. Adapt the implementation such that steps without improvement (stagnation) are allowed as described on Slide 8 of Chapter 21 (print version). Verify that approximately 94% of the runs with a bound of 100 steps yield a solution, and that a solution is found after 21 steps on average.