Foundations of Artificial Intelligence

M. Helmert
S. Eriksson
Spring Term 2022

University of Basel
Computer Science

Exercise Sheet 6

Due: April 10, 2022

Important: for submission, consult the rules at the end of the exercise. Non-adherence to the rules will lead to your submission not being corrected.

Exercise 6.1 (1+0.5+0.5 marks)

An *independent set* of a given input graph $G = \langle V, E \rangle$ is a subset of the vertices of G such that no two vertices in the independent set are adjacent. The problem of finding an independent set for G of maximal size is called IndSet.

(a) Formalize IndSet as a combinatorial optimization problem.

(b) Is your formulation a pure search problem, a pure optimization problem, or a combined search and optimization problem?

(c) Define a suitable neighboring function for hill climbing, i.e., a function that ensures that every candidate can be reached and that bounds the amount of neighbors to a reasonable size. You do not need to show that the function is suitable.

Exercise 6.2 (1.5+0.5+0.5+0.5 marks)

Consider the following set of inequalities over numbers $a, b, c, d \in \{1, 2, 3\}$:

$$\begin{aligned} a + c &\geq 4 \\ c &\neq b + 1 \\ d &> b \\ a + d &\leq 3 \end{aligned}$$

(a) Formalize the problem as a binary constraint network. Also specify trivial constraints.

(b) Is the constraint network solvable? If yes, provide a solution of the constraint network. If not, justify your answer.

(c) Provide a consistent partial assignment that assigns a and d and that *cannot* be extended to a solution.

(d) Provide an inconsistent partial assignment that assigns two variables.

Exercise 6.3 (2.5+0.5+1.5+0.5 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) except for what is provided by us. If you encounter technical problems or have difficulties understanding the task, please let us – the tutor or assistant – know *sufficiently ahead of the due date*.
The archive `hill-climbing.zip` contains an incomplete implementation of hill climbing search for the 8 queens problem that was presented in the lecture.

(a) Implement hill climbing in the function `protected SearchResult search()` in the file `HillClimbing.java`. The implemented heuristic counts how many pairs of queens are threatening each other, which means we are considering a minimization variant here and you need to adapt the function presented on Slide 21 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.

(b) Test your implementation by verifying the statements on Slide 24 of Chapter 20 (print version), which state that hill climbing with a random initialization finds a solution in around 14% of the cases using 3-4 steps on average. You can compile and run your code with `javac HillClimbing.java` followed by the command `java HillClimbing 8queens`. Report the percentage of successful runs and the average number of steps.

(c) 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).

Hint: Since the 8 queens problem is a pure search problem, you can terminate as soon as a solution is found.

(d) Verify that approximately 96% of the runs with a bound of 100 steps yield a solution, and that a run took around 22 steps on average. What is the percentage of successful runs and the average number of steps for your solution?

Note: for exercises (a) and (c), only hand in the two files `HillClimbing.java` and `HillClimbingWithStagnation.java` respectively.

Submission rules:

- Exercise sheets must be submitted in groups of two 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. Either use page numbers on all pages or put your names on each page. Make sure your PDF has size A4 (fits the page size if printed on A4).
- For programming exercises, only create those code textfiles required by the exercise. 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.
- For the submission: if the exercise sheet does not include programming exercises, simply upload the single PDF. If the exercise sheet includes programming exercises, upload a ZIP file (ending .zip, .tar.gz or .tgz; *not* .rar or anything else) containing the single PDF and the code textfile(s) and nothing else. Do not use directories within the ZIP, i.e., zip the files directly.
- Do not upload several versions to ADAM, i.e., if you need to resubmit, use the same file name again so that the previous submission is overwritten.