**Foundations of Artificial Intelligence**

M. Helmert
S. Eriksson
Spring Term 2021

University of Basel
Computer Science

# Exercise Sheet 6

**Due: April 14, 2021**

**Exercise 6.1** (1+0.5+0.5 marks)

In the PARTITION problem we are given a set of natural numbers $N$ and try to partition them into two sets $A_1$ and $A_2$ such that the sum of the numbers in $A_1$ is as close as possible to the sum of the numbers in $A_2$.

(a) Formalize PARTITION 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.

**Exercise 6.2** (2.5+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). If you encounter technical problems or have difficulties understanding the task, please let us know.
The archive `hill-climbing.tar.gz` 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 around 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.

**Exercise 6.3** (1.5+0.5 *bonus* marks)

*This is a bonus exercise which builds upon your solution from Exercise 6.2. Bonus marks count normally but are not considered for the sum of total achievable marks.*

(a) 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.*

(b) Verify that approximately 96% of the runs with a bound of 100 steps yield a solution, and that if a solution is found, it took around 22 steps on average. What is the percentage of successful runs and the average number of steps in case of success for your solution?

**Submission rules:**

- Create a single PDF file (ending .pdf) for all non-programming exercises. If you want to submit handwritten parts, include their scans in the single PDF. Put the names of all group members on top of the first page. Use page numbers 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, create only those Java textfiles (ending .java) required by the exercise. Put your names in a comment on top of each file. Make sure your code compiles and test it!
- For the submission, you can either upload the single PDF or prepare a ZIP file (ending .zip, .tar.gz or .tgz; not .rar or anything else) containing the single PDF and the Java textfile(s) and nothing else. Please do not use directories within the ZIP, i.e., zip the files directly.