Foundations of Artificial Intelligence

M. Helmert
S. Sievers
Spring Term 2023

University of Basel
Computer Science

Exercise Sheet 6

Due: April 16, 2023

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.

Note: due to the Easter break, this exercise sheet covers two weeks with three lecture days in total. It therefore contains exercises worth 15 marks.

Exercise 6.1 (1+0.5+0.5 marks)

A *clique* of a given undirected graph $G = \langle V, E \rangle$ is a subset of the vertices of G such that there is an edge between every two vertices in the subset. The problem of finding a clique of at least size n for a given $n \in \mathbb{N}$ for G is called CLIQUE.

(a) Formalize CLIQUE 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 neighborhood for hill climbing by providing a formal definition of a function mapping a candidate to a set of neighbors. A neighborhood function is suitable for hill climbing if it ensures that that every candidate can be reached and if it 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+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 tutors 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 on whether hill climbing found a solution and on the number of steps.

Only submit the file `HillClimbing.java`

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

(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).

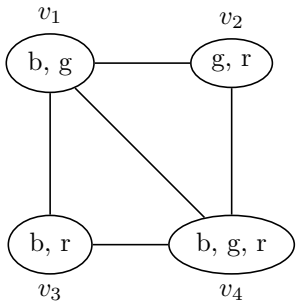
Only submit the file `HillClimbingWithStagnation.java`

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

(d) Verify the statements of Chapter 21 which state that hill climbing with allowed stagnation using a bound of 100 steps finds a solution in around 96% of the cases using 22 steps on average. To do so, report the percentage of successful runs and the average number of steps of your implementation. You can compile and run your code with `javac HillClimbingWithStagnation.java` followed by the command `java HillClimbingWithStagnation 8queens`.

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

Consider a variant of the graph coloring problem where a graph is given and each vertex has a set of allowed colors it can be colored with. The problem then is to find a color for each vertex from its set of allowed colors such that no two adjacent vertices have the same color. Consider the following instance with four vertices $\{v_1, \ldots, v_4\}$ that be can colored with one of the colors given in the node.



(a) Formalize the example as a binary constraint network.

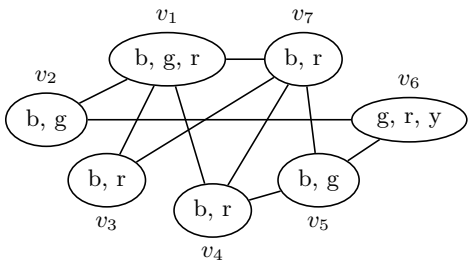
(b) Is the constraint network solvable? If yes, provide a solution of the constraint network. If not, justify your answer.

(c) Provide a consistent partial assignment that *cannot* be extended to a solution and that is such that the partial assignment assigns values to as few variables as possible.

(d) Provide an inconsistent partial assignment.

Exercise 6.4 (1+2 marks)

Consider a variant of the graph coloring problem where a graph is given and each vertex has a set of allowed colors it can be colored with. The problem then is to find a color for each vertex from its set of allowed colors such that no two adjacent vertices have the same color. Consider the following instance with seven vertices $\{v_1, \ldots, v_7\}$ that be can colored with one of the colors given in the node.



Use the following static strategies on variable and value orders:

- Variable order:
  - select a variable according to the *minimum remaining values* variable order criterion;
  - if there is more than one such variable, break ties according to the *most constraining variable* variable order criterion;
  - if the choice is still not unique, break ties by selecting the variable with the smallest index.
- Value order: reverse alphabetical

(a) Provide the variable and value orders induced by the static order strategies.

(b) Provide the search tree that is created by applying naive backtracking on the depicted problem using the static variable and value orders from the first part of the exercise. Depict the search tree in a similar style as in the lecture slides.

Exercise 6.5 (1+2 marks)

Consider a constraint network $\mathcal{C}$ denoting a Latin square of size 3 with the partial assignment $\alpha = \{\mathrm{A1} \mapsto 1, \mathrm{B1} \mapsto 2, \mathrm{A3} \mapsto 3\}$:

	1	2	3
A	1		3
B	2		
C			

In the following, you may assume that the already filled out squares are fixed, i.e., that the domain of the corresponding variables contains only the single entry that encodes the depicted value. The domain of the remaining variables contains all 3 possible values, which leads to the following domains for all variables:

$$\begin{array}{lll} \mathrm{dom}(\mathrm{A1}) = \{1\} & \mathrm{dom}(\mathrm{A2}) = \{1,2,3\} & \mathrm{dom}(\mathrm{A3}) = \{3\} \\ \mathrm{dom}(\mathrm{B1}) = \{2\} & \mathrm{dom}(\mathrm{B2}) = \{1,2,3\} & \mathrm{dom}(\mathrm{B3}) = \{1,2,3\} \\ \mathrm{dom}(\mathrm{C1}) = \{1,2,3\} & \mathrm{dom}(\mathrm{C2}) = \{1,2,3\} & \mathrm{dom}(\mathrm{C3}) = \{1,2,3\} \end{array}$$

(a) Determine the domains of all variables after applying forward checking in α.

(b) Apply the AC-3 algorithm that has been presented in Chapter 25 of the lecture slides on the constraint network $\mathcal{C}$ with the domains that are the result of (a) until arc consistency is enforced. Select the variables u and v in each iteration of the while loop such that the domain

of u changes in the call to `revise`$(\mathcal{C}, u, v)$. Provide u, v, and $\mathrm{dom}(u)$ in each iteration. Note that you do *not* have to provide the elements that are inserted into the queue, and you may stop the algorithm as soon as there are no variables u and v such that $\mathrm{dom}(u)$ changes.

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.