Foundations of Artificial Intelligence

M. Helmert
J. Seipp
Spring Term 2019

University of Basel
Computer Science

Exercise Sheet 6

Due: April 10, 2019

Exercise 6.1 (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 Hamilton path in a fully connected graph, i.e., there is an edge between all pairs of vertices in the graph. Is the COP a pure search problem, a pure optimization problem, or a combined search and optimization problem?

Exercise 6.2 (1.5+2+0.5+2 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.

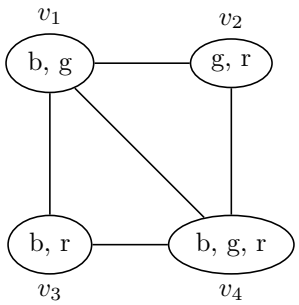
(a) Implement the heuristic for the 8 queens problem that is presented on Slide 22 of Chapter 20 (print version), where the heuristic value is equal to the number of pairs of queens threatening each other. To do so, implement the function `public int h(Configuration _conf)` in the file `EightQueensProblem.java`.

(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 statements on Slide 23 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.

(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. What is the percentage of successful runs and the average number of steps for your solution?

Exercise 6.3 (1.5+0.5+0.5+0.5 marks)
Consider the following example instance of the graph coloring problem:



(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.

Important: Solutions should be submitted in groups of two students. However, only one student should upload the solution. Please provide both student names on each file and each page you submit. We can only accept a single PDF or a ZIP file containing *.java or *.pddl files and a single PDF.