Informed Search Methods in AI

Fall 2023 Assignment 1

In this assignment, you will implement a generic *CSP solver* and then use it to solve *SuDoKu* puzzles. A skeleton of a *Simple-CSP solver* is provided. It currently uses only *Generate-and-test backtracking* for solving, but your project is to augment and improve it using some of the techniques we have been studying. More specifically, you should implement the following backtracking-based search algorithms:

- *Chronological Backtracking (BT)*
- *Backjumping search (BJ)*
- *Conflict-directed Backjumping search (CBJ*)

as well as *arc consistency* using the *AC-3* algorithm.

Run the different algorithms on the provided test suite (*benchmark.txt*) to compare their run-time efficiency, both with and without forcing *arc consistency* on the *constraint network*. For your experiments, run the *BT*, *BJ*, and *CBJ* algorithms on the 20 puzzles in the *benchmark.txt* file, both without and with *arc consistency* (note: *GTBT* should be disabled).

Write a brief report (ca. 2 pages + an appendix) summarizing your findings. Include a table in your report showing the total nodes searched and running time over the benchmark test set for each algorithm configuration above. In an appendix, list the number of nodes searched for each configuration, the running time, and the (abbreviated) solution for each puzzle (you can copy the output written to the *output.txt* file). Also, report whether your algorithms were able to solve all the puzzle instances and, if not, how often it was unsuccessful). *Finally, briefly discuss how you think the solver could be further improved.*

PART 1

Your first task is to convert Sudoku puzzles into constraint networks in a format recognized by the *Simple-CSP* solver. You are provided a skeleton program for this (*sudoku.py*), but you must implement the missing routines (see code for details).

PART II

In the second part, you implement the missing backtracking-based search algorithms in the solver (*solvers.py*). Also, add *arc consistency* pre-processing to the constraint network by implementing the missing routine (also in *solvers.py*). You can start this part even though the first part is not finished because example constraint/domain files are provided. See code for details. *Hint:* When initially testing your backtracking algorithms, it might be a good idea to do so on trivial CSP problems (for example, the first puzzle in *sudoku.txt*). Additionally, the *Simple-CSP* solver can take in the specification of any

problem (not necessarily *SuDoKu* puzzles) that uses only *binary not-equal constraints*. You can use the *-i* flag to run one or more individual puzzles in a given test file (e.g., *-i 2 -i 5* to run only puzzles 2 and 5).

You can work on the assignment in a **group of two students. Hand in** the two Python programs you modified (*sudoku.py* and *solvers.py;* you should not change the other Python files) and your report (as a PDF file called *report.pdf*).