Artificial Intelligence and Knowledge Engineering Laboratory

Task 2. Constraint Satisfaction Problem

Authors: M. Piasecki, A. Zawisza, H. Kwaśnicka, M. Paradowski, M. Przewoźniczek

Task Objectives

Getting familiar with algorithms for Constraint Satisfaction Problems in practice through individual implementation.

Subtasks

- Get familiar with backtracking search algorithm and forward checking technique (of constraint propagation), heuristics applied in CSP – *see the lecture*
  - advanced students: learn about AC-3 and MAC algorithm, as a better alternative for backtracking + forward checking.
- Analyse problems shortly described below.
- Formulate both of them as a CSP problems, i.e. define: variables, their domains and constraints in a formal or at least semi-formal way.
- Implement searching + constraint propagation algorithm for solving the chosen CSPs (*see the lecture*), e.g. backtracking algorithm enhanced with selected heuristics and combined with the forward checking.
- Test the implementations with problems of different sizes or different difficulty, i.e. for different sizes of the boards (*n* x *m*), number of letters to be used, and the number words to be used on the board, whenever it is relevant to the given problem.
- Present most interesting results: solutions and numerical results (e.g. execution times, also for problems that cannot be solved).
- Discuss the results.
- The report should contain all the above points (except the implementation, which should be delivered in a form of the source code files).

Warming up

An easier problem to warm up.

Sudoku

Wikipedia: https://en.wikipedia.org/wiki/Sudoku

> “is a logic-based, combinatorial number-placement puzzle. The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution. Completed games are always an example of a Latin square which include an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column, or any of the nine 3×3 subregions of the 9×9 playing board.”

See also: https://sudoku.com

Datasets for this problem are described in the Datasets Section, below.

Entertaining Problem

A little more planning for good specification is required.

Fill-In (puzzle)
Wikipedia: https://en.wikipedia.org/wiki/Fill-In_(puzzle)

> "also known as Fill-It-Ins or Word Fills, are a variation of the common crossword puzzle in which words, rather than clues, are given."
> "The solver is given a grid and a list of words. To solve the puzzle correctly, the solver must find a solution that fits all of the available words into the grid. Generally, these words are listed by number of letters, and further alphabetically. Many times, one word is filled in for the solver to help him or her begin the game."

Datasets for this problem are described in the Datasets Section, below.

Datasets

A ZIP archive with data files described problem instances of both types is available on ePortal in addition to the assignment.
Sudoku.csv file contains 44 Sudoku instance of varied difficulty levels. Instances of the 0 level are intended only for testing programming code and there is no obligation to perform experiments on. them for the report. Solutions are given for them.
Students should perform experiments on the following number of files:

- 1 file from the group of easy problems (the levels 1 and 2),
- 1 file from the group of medium problems (the levels 3 - 5),
- 1 file from the group of difficult problems (6 and 7),
- 1 file from the group of mingled problems (the level: 8),
- 1 file from the group of mingled problems (the level: 9).

Instances of Fill-in puzzle are to be found in the *Jolka* folder (5 instances). Each instance is described by two files: *puzzle* and *wordsX*. The first file includes the description of the board, the second one provides words. The instance *puzzle0* is for testing code and its solution is given in the file *filled0*.

For the report, students should perform experiments on all Fill-in instances, except the test one.

Task rating

2 points – formulate the problems as CSP
4 points – implement backtracking combined with forward checking CSP for them
2 points – compare efficiency of processing for various problem parameters
2 points – give a correct interpretation of the results

Exciting Problem (for 5 Extra Points)

Formulate the problem below as a CSP problem and write a program solving it on the basis of the CSP program for the first problem.
Playing Scrabble is in fact searching for an optimal combination. In a slightly simplified version, a player selects randomly a sequence of letters with repetitions. Every letter is assigned some score points (see https://en.wikipedia.org/wiki/Scrabble_letter_distributions). There is a board of the size *n* x *m* with cells to store letters (at most one letter in a cell).
The player needs to form words from letters selected and put the on the board in way identical to the first problem. However, now, the task is to use as many letters as possible and to achieve as large sum of the scores from the individual letters used as possible. Only words from a subset of the size *k* of the set *S* (see the basic problem) are accepted as proper words.

Bibliography

1. Lecture notes
2. Russell and Norvig: http://aima.cs.berkeley.edu/2nd-ed/newchap05.pdf