TH Ingolstadt
Algorithms for AI (S. Hahndel)

Studiengang CAI BA
SS 2023

Demonstation Task 4

(Search space reduction with constraints using the example of Sudoku

1 Problem description and task description

In this task, depth-first search is to be combined with an algorithm for search-space restriction by forward checking or local consistency. The problem is Sudoku, because most people probably already know it. probably already know it. In Sudoku, the goal is to find a 9x9 matrix that is already partially filled with digits from 1 to 9, so that the following conditions are met conditions are fulfilled:

- In each row and each column of the matrix, each digit from 1 to 9 occurs occurs exactly once
- In each of the 3x3 submatrices shown, each digit from 1 to 9 also occurs exactly once. digit from 1 to 9 occurs exactly once

Example for Sudoku:

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

Given unsolved Sudoku

4	2	*6*	5	*7*	1	*3*	9	*8*
8	*5*	*7*	2	*9*	3	*1*	*4*	6
1	3	*9*	*4*	6	*8*	*2*	7	*5*
9	*7*	1	*3*	*8*	*5*	6	*2*	*4*
5	4	*3*	*7*	*2*	*6*	*8*	1	9
6	*8*	2	*1*	*4*	*9*	7	*5*	*3*
7	9	*4*	*6*	3	*2*	*5*	8	*1*
2	*6*	*5*	8	*1*	4	*9*	*3*	7
3	1	*8*	9	*5*	7	*4*	6	*2*

Solved Sudoku

The problem with this type of problem is that a "'normal'" search algorithm would quickly run into problems due to the combinatorial explosion. would run into problems.
In the example above, 50 fields are free. Each field can be assigned a digit from 1-9, i.e. there would be a search space of

$$9^{50} = 515377520732011331036461129765621272702107522001$$

possibilities. This rules out a brute force search even on the fastest computers. A depth or breadth search would also not be appropriate, as we have very long calculation times. computing times. Therefore, we need a possibility to restrict the search space as can be achieved, for example, by forward checking or - even better - local constraint propagation. local constraint propagation.

2 Procedure and programming notes

It will combine a depth-first search with a forward-checking or local consistency algorithm. local consistency. This is up to you.o

- The data structure for the problem is clear. It is a two-dimensional 9x9 field with digits from 0-9, the 0 describes an empty field.
- So you have a constraint satisfaction problem with 81 variables.
- The domain of each variable consists of the digits 1-9.
- You consider an appropriate data structure that describes which digits are are left in a domain
- You have a total of 27 constraints: z1-z9 for the rows 1-9, s1-s9 for the columns 1-9, q1-q9 for the 9 sub-squares.
- Each of these constraints is responsible for 9 variables and describes, that no digit other than 0 occurs twice in the 9 squares. Otherwise the respective constraint is violated.

2.1 Notes on search space constraints with forward checking

If you use simple forward checking for the solution, take the depth-first search as before, but control the variable selection with the following heuristic:

1. You always select the variable with the fewest remaining values as the next variable. This ensures that you always have the lowest degree of branching when searching the search tree. have the lowest degree of branching.
2. As soon as you have selected a value for the variable, you eliminate all the values from the values from the sets of values of the other variables that violate a constraint.
3. Caution: You must undo such assignments and changes to value sets during backtracking. backtracking to undo them.
4. easiest way to do this is to let the computer do the work for you. by programming the whole thing recursively.

2.2 Notes on search space restriction using local consistency

Informally, the algorithm with local consistency works something like this:

1. In the first step, the problem is read in. For each already occupied variable (field element), the corresponding domain becomes one-element. The free variables are (still) occupied with the values 1-9.

2. In this step, the domains are made locally consistent: First, all the 27 constraints are marked for filtering.
   - You check whether there is a one-element domain under the variable covered by this constraint is covered by this constraint. If so, select one of them and delete the corresponding element from the other domains. Repeat this step for the constraint until you no longer find a single-element domain. one-element domain. If a domain has only 0 elements The constraint is violated.
   - Remove this constraint from the filtering pre-list and re-mark all constraints for filtering. re-mark all constraints for filtering that cover a variable, whose domain was changed by the last step, provided that the constraint is not is not already included.
   - The last two steps are repeated until no constraint is marked for filtering. is marked for filtering.
   - If all domains are now already **einelementig**, you have a globally consistent globally consistent solution that you can output.

3. Otherwise, choose a variable that is not yet occupied, preferably one with a domain as small as possible with at least 2 values.

4. You select the next value from the domain of the variable selected in the last step and test the domain with this value and go to step 2. to step 2

5. If there is still a value, repeat the last step.

6. If there is still a variable you have more than 1 element in the domain, go back to step 3

7. If you cannot find a variable anymore: no solution.

You may write the program again in the programming language of your choice (Python, Java, C, Prolog, Scala, ...). write.

3 Example programme flow

Your programme should read the given Sudoku from a text file with the following format:

```
530070000
600195000
098000060
800060003
400803001
700020006
060000280
000419005
000080079
```

This makes it easier to test. Here is a typical programme sequence:

```
./sudoku sudoku1.txt
-------------
|53 | 7 |   |
|6  |195|   |
| 98|   | 6 |
-------------
|8  | 6 |  3|
|4  |8 3|  1|
|7  | 2 |  6|
-------------
| 6 |   |28 |
|   |419|  5|
|   | 8 | 79|
-------------
=============
For the solution 4 positions were examined!
-------------
|534|678|912|
|672|195|348|
|198|342|567|
-------------
|859|761|423|
|426|853|791|
|713|924|856|
-------------
|961|537|284|
|287|419|635|
|345|286|179|
-------------
```