
Εργασία 1

Constraint Propagation Methods

Sudoku

- ***Sudoku*** is a logic-based placement puzzle. It consists in placing numbers from 1 to 9 in a 9-by-9 grid made up of nine 3-by-3 subgrids, called *regions* or *boxes* or *blocks*, starting with various numerals given in some cells, the *givens* or *clues*. It can be described with a single rule:
- *Each row, column and region must contain all numbers from 1 to 9*
- We can immediately deduce that for each row, column, and region the values in the cells have to be different. Moreover, this condition is sufficient; thus, the unique rule could be reformulated as:
- *Each row, column and region must contain numbers from 1 to 9 that are all different*

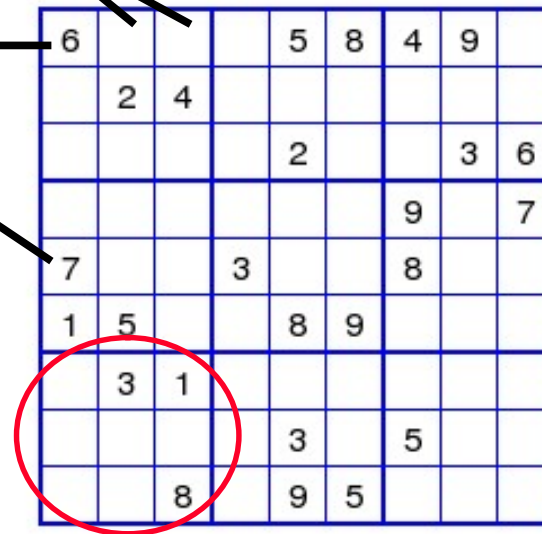
6				5	8	4	9	
	2	4						
				2			3	6
						9		7
7			3			8		
1	5			8	9			
	3	1						
				3		5		
		8		9	5			

Sudoku as a CSP

81 variables, one for each cell

Variables with domain $\{1, \dots, 9\}$

Variables with
singleton domains



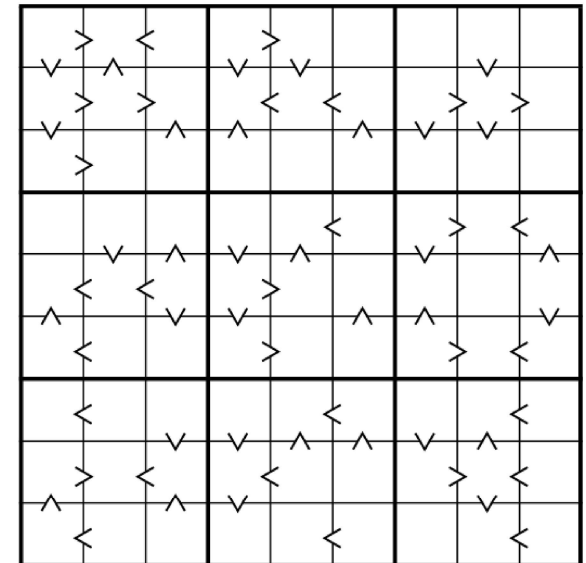
6				5	8	4	9	
	2	4						
				2			3	6
						9		7
7			3			8		
1	5			8	9			
	3	1						
				3		5		
		8		9	5			

27 cliques of 36 \neq constraints in each one
810 constraints in total

Binary \neq constraints

Greater than Sudoku

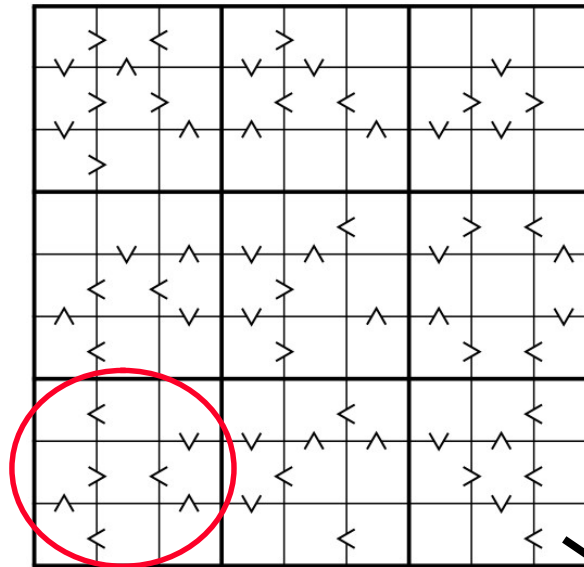
- ***Greater than Sudoku*** is a logic-based placement puzzle that is a variant of the well-known ***Sudoku***. It consists in placing numbers from 1 to 9 in a 9-by-9 grid made up of nine 3-by-3 subgrids, called *regions* or *boxes* or *blocks*, starting with empty cells. It can be described with two rules:
- *Each row, column and region must contain numbers from 1 to 9 that are all different*
- *For some adjacent cells, one must have a greater value than the other*



Greater than Sudoku as a CSP

81 variables, one for each cell

All variables with domain $\{1, \dots, 9\}$



27 cliques of 36 \neq constraints in each one
810 \neq constraints in total

Additional $>$ constraints between some variables.
Their number depends on the specific problem

Binary \neq and $>$ constraints

Ζητούμενο

- Υλοποιήστε πρόγραμμα που εφαρμόζει τεχνικές συνέπειας (consistency techniques) σε προβλήματα sudoku και greater than sudoku. Οι εξής τεχνικές πρέπει να υλοποιηθούν:
 - **Arc Consistency**
 - Με χρήση της παραλλαγής του AC-3 που βάζει μεταβλητές στην ουρά
 - **Restricted Path Consistency (RPC)**
 - Με χρήση του αλγορίθμου RPC-1
 - **Neighborhood Singleton Arc Consistency (NSAC)**
 - Με χρήση του αλγορίθμου NSACQ
- Ο κάθε αλγόριθμος πρέπει να εκτελείται στο αρχικό πρόβλημα
 - στο τέλος της εκτέλεσης του κάθε αλγορίθμου πρέπει να τυπώνεται το πλήθος των διαγραφών τιμών που έχει κάνει, το πλήθος των μεταβλητών που έχουν μείνει με μια τιμή στο domain τους και ο χρόνος εκτέλεσης

Εργασία 1

- Προθεσμία υποβολής: 03/05/2022
- Η εργασία είναι ομαδική (σε ομάδες των τεσσάρων το πολύ ατόμων)
- Η γλώσσα υλοποίησης μπορεί να είναι οποιαδήποτε επιθυμείτε ανάμεσα στις C, C++, Java, Python (δεκτή γίνεται και υλοποίηση σε Matlab)
 - Εκτός από τον κώδικα πρέπει να υποβάλετε και μια σύντομη αναφορά όπου θα περιγράφονται τα βασικά σημεία του προγράμματος και θα αναφέρονται πιθανά προβλήματα που προέκυψαν
- Το πρόγραμμα σας πρέπει να μπορεί να «φορτώσει» τα δεδομένα των προβλημάτων που βρίσκονται σε αρχεία σε κατάλληλες δομές δεδομένων στην μνήμη.

Θέματα υλοποίησης

Πως θα αποθηκευτούν οι μεταβλητές, τα πεδία τιμών και οι περιορισμοί στη μνήμη;

D[81][9]: an array with 81 rows and 9 columns holding info about the domain values of any variable x (if they are still in the domain, i.e. available, or not)

VALID: a constant denoting that a value is available

valid_values[x]: the number of available values in $D[x]$ at any time

D[0]	-1	-1	-1	-1	-1	-1	-1	-1	
D[1]	-1	-1	-1	-1	-1	-1	-1	-1	

...

$\text{valid_values}[0] = \text{valid_values}[1] = \dots = 9$

assuming **VALID** = -1
the domains of all
variables will be
initialized to -1

the deletion of value
can be marked with
REMOVED (-2)

Θέματα υλοποίησης

Πως θα αποθηκευτούν οι μεταβλητές, τα πεδία τιμών και οι περιορισμοί στη μνήμη;

C[81][81]: an array with 81 rows and 81 columns holding info about the constraints. A cell $C[x][y]$ is set to (for example) 0 if there is no constraint between x and y , to 1 if there is a \neq constraint between them, to 2 if there is a $>$ constraint and to 3 if there is a $<$ constraint

	0	1	2	3	4	...
0	0	2	1	1	1	
1	3	0	1	1	1	
...						

This array allows checking a constraint in constant time

How do we modify AC-3?

X : the set of variables, D : the set of domains, C : the set of constraints

$neigh(x)$: the variables connected to x in the constraint graph

Procedure AC-3 (X, C, D):

```
1   $Q \leftarrow \{ (x_i) \mid x_i \in X \}$ 
2  While  $Q$  is not empty do
3    Begin
4      select and delete a variable  $(x_i) \in Q$ 
5      for each variable  $x_j \in neigh(x_i)$ 
6        if  $C[x_j][x_i] \neq 0$ 
7           $updated \leftarrow REVISE(x_j, x_i)$ 
8          if  $D(x_j) = \{ \}$  then return false
9      if  $updated$  then  $Q \leftarrow Q \cup \{ x_j \}$ 
10 End
```

We need to specify the
CHECK function

No need to call REVISE if
no constraint exists
between the two variables!

REVISE(x_i, x_j): Pseudocode

REVISE(x_i, x_j)

1. $revised \leftarrow \text{false}$
2. **for** each $a \in D_{x_i}$
3. $found \leftarrow \text{SUPPORTED}(\langle x_i, a \rangle, x_j)$
4. **If** $found = \text{false}$ **then**
5. $revised \leftarrow \text{true}$
6. $D_{x_i} \leftarrow D_{x_i} \setminus \{a\}$
7. **RETURN** $revised$

SUPPORTED($\langle x_i, a \rangle, x_j$)

1. $support \leftarrow \text{false}$
2. **for** each $b \in D_{x_j}$
3. **if** CHECK($\langle x_i, a \rangle, \langle x_j, b \rangle$) **then**
4. $support \leftarrow \text{true}$
5. **RETURN** $support$
6. **RETURN** $support$

The implementation of CHECK depends on the type of constraint

CHECK($\langle x_i, a \rangle, \langle x_j, b \rangle$)

CHECK($\langle x_i, a \rangle, \langle x_j, b \rangle$)

1. **if** $C\langle x_i, x_i \rangle = 1$ **then**
2. **if** $(a \neq b)$
3. **RETURN** *true*
4. **else if** $C\langle x_i, x_i \rangle = 2$ **then**
5. **if** $(a > b)$
6. **RETURN** *true*
7. **else if** $C\langle x_i, x_i \rangle = 3$ **then**
8. **if** $(a < b)$
9. **RETURN** *true*

C[81][81]: an array with 81 rows and 81 columns holding info about the constraints. A cell $C[x][y]$ is set to (for example) 0 if there is no constraint between x and y , to 1 if there is a \neq constraint between them, to 2 if there is a $>$ constraint and to 3 if there is a $<$ constraint