# CSP Assignment

Knowledge Representation 2015

## 1. Introduction

Nowadays, a large amount of real life problems and games such as timetables scheduling or sudokus can be re-written as Constraint satisfaction problems (CSPs) and solved by a CSP solver.
Whereas the first assignment of the KR course was about analysing the properties of sudokus by running experiments on SAT solvers, this second assignment is about implementing a CSP solver and experiment different efficiency improvement techniques such as constraint propagation, intelligent backtracking and variable heuristics.

After having implemented a basic but working CSP solver, we chose to focus on constraint propagation and variable heuristics and decided to implement :

- Forward checking : simple constraint propagation method which consists of pruning in advance the domains of unassigned variables which don't match with the constraints
- Arc consistency : a variable is arc-consistent with another variable if each value of the domain of the first variable is consistent with some values of the domain of the other variable
- Minimum remaining values (MRV) : start the search algorithm with the variable which has the smallest domain. It reduces the branching factor and it is more likely to pick the right variable.

Our hypothesis was that forward checking, arc consistency, and MRV would make the CSP more efficient in terms of number of splits and eventually in terms of running time.

Further in this paper, we detail the experimental setup along with the implementation architecture and the optimization techniques, we provide and analyse the results and finally state our conclusions.

# 2. Experimental setup

## a. Implementation

The code was implemented in python 2.7, using the PyCharm IDE on Windows 8.1. Our code is divided in two modules : *my-csp-solver.py* and *csp.py*.

`my-csp-solver.py` Is the main module, and takes two arguments : an *input* file and an *output* file. If no output file is specified, the solutions are printed on the standard output.
The input file may contain one or more CSP problems, one per line.
The module converts each line of the input file into a `CSPProblem`, solves it and prints the solution.

`csp.py` contains 3 classes : `CSPProblem`, `AllDifferentConstraint`, and `BacktrackingSolver`.

### i. `CSPProblem`

The `CSPProblem` class is the implementation of the CSP problem, and has as attributes :

- `variables : dictionary` (key = variable name, value = domain)
- `constraints : list`
- `solver : BacktrackingSolver`
- `assignments : dictionary` (key = variable, value = assigned value)

Along with some initialization purpose functions (add_variable, assign_variable, add_constraint,...), the class also implements 4 solving functions that call the BacktrackingSolver for “basic” backtracking solving, solving with forward check, solving with MRV and solving with forward check and MRV.
Finally, the class also has a pre-processing function called just before the solver and which prunes the domains and maps the variables with their constraints.

### ii. `AllDifferentConstraint`

The `AllDifferentConstraint` class is the implementation of the Constraints. It only handles all different constraints, i.e. all variables of the constraint must have a different assigned value.

The class contains 2 functions : `domain_pruning` and `satisfied`.

- `domain_pruning` is used to prune the domains before solving the CSP problem

- `satisfied` returns `True` if the assignments satisfy the constraint, `False` otherwise. The function also takes a boolean parameter which must be set to `True` if forward checking is wanted.

#### iii. `BacktrackingSolver`

The `BacktrackingSolver` class is the implementation of the Solver class. It is a recursive backtracking solver. and has an attribute `domain_stack`, which is a stack containing all the previous states of the domains and allows to backtrack the forward checking.

## b. Optimization techniques

### i. Forward check

Simple constraint propagation algorithm which consists of pruning in advance the domains of unassigned variables which don't match with the constraints :

```
FOR EACH c_variable of the constraint:
      IF variable hasn't been assigned yet:
            FOR EACH assigned a_variable of the constraint:
                  remove a_variable value from c_variable domain
                  IF c_variable's domain length = 1:
                        add c_variable to the constraint's assigned
vars
                        break
                  IF c_variable's domain length = 0:
                        CONSTRAINT NOT SATISFIED : rollback
CONSTRAINT SATISFIED
```

### ii. Minimum remaining value

Start the search algorithm with the variable which has the smallest domain. It reduces the branching factor and it is more likely to pick the right variable.

```
FOR EACH variable :
      LIST = variables sorted from the smallest to the biggest domain
      FOR EACH var in LIST:
            IF var hasn't been assigned yet:
                  FOR EACH value in var's domain:
                        assign var = value
                        IF assignment satisfies the constraint:
```

```
break
```

### iii. Arc consistency

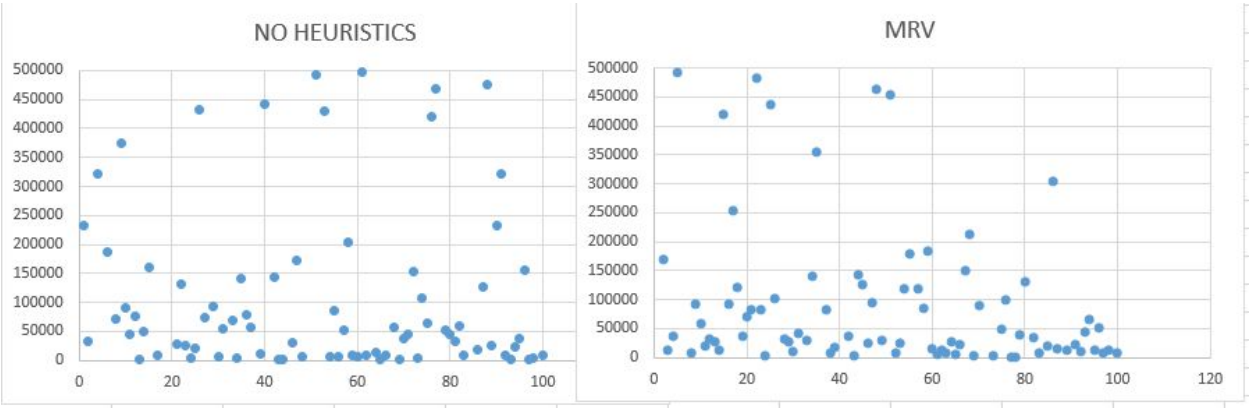
A variable is arc-consistent with another variable if each value of the domain of the first variable is consistent with some values of the domain of the other variable

We didn't have the time to have a working arc-consistency algorithm. But the idea was to use the AC-3 algorithm (https://en.wikipedia.org/wiki/AC-3_algorithm)

# 3. Results

As said earlier, our hypothesis was that forward checking, arc consistency, and MRV would make the CSP more efficient in terms of number of splits and eventually in terms of running time.

While comparing the number of splits between a MRV heuristic and a "first come" heuristic, we could not see any obvious improvement.
As the MRV algorithm is quite simple, we believe that our backtracking algorithm is more likely to be buggy. However, not without trying, we couldn't find the problem.




Unfortunately, we couldn't test the Forward check, as it finds solutions for only high number of givens. We believe that it doesn't rollback the domains correctly when an assignment isn't satisfying.

# 4. Conclusion

We managed to write a working basic CSP solver for sudokus, and although the “improvements” we wanted to implement don’t work as they should, we have highly improved our knowledge and understanding of the CSP problems and algorithms.
Moreover, having a very limited knowledge of python, writing this program is such a short time was quite challenging.

Improvements that should be added to this project aren’t hard to find : finding the problems for the MRV and forward checking, implementing Arc consistency, back jumping, adding other constraints, …

# 5. References

While implementing the solver and the algorithms, in addition to the lecture notes, the following references where our main support :

http://www.codeproject.com/Articles/34403/Sudoku-as-a-CSP

http://www.constraintsolving.com/solvers/python-based-constraint-solver

https://labix.org/python-constraint