Programming Assignment 3

Aditya Gupta
2015CSB1003

March 10, 2017

1 Sudoku Solver using CSP

1.1 Description

A Sudoku Solver was implemented using CSP. The base class `SudokuSolver` creates a `Collection` of unassigned variables while parsing the input lines. In case of `BS` the collection is made into `LinkedList` and every time a new `Variable` is taken from the front of the list and then if the solver backtracks, the variable is added back to front of the list. In case of `BSI` it is made into a `HashSet` for constant time removal after selection of a suitable `Variable` from the set compared according to `MRV` heuristic, i.e. the one with the least consistent values (tie broken with comparing degreees.). In case of `BSII` the legal/consistent values are first collected into a `List` and then sorted according to the one which allows most values for all unassigned neighbours (related variables). In case of `BSMAC` domains are also explicitly introduced in form of `Set<Integer>` and all domains are maintained in a `HashMap<Variable, HashSet<Integer>>`, after every `Variable` is assigned some value, arc-consistency is maintained and if some domain gets reduced to ϕ, the solver backtracks otherwise we now work with reduced domains.

1.2 Observations

After implementing the three heuristics; the time, number of backtracks and memory utilization were observed:

Metrics	`BS`	`BSI`	`BSII`	`BSMAC`
Time (seconds)	40.244	13.097	14.329	21.809
Backtracks	227,502,770	370,626	364,783	159,498

We can see that each heuristic makes the total backtracks quite less but in case of `BSII` and `BSMAC`, the added cost of calculating heuristic takes more time.

2 Sudoku Solver using MiniSAT

A Sudoku Solver was implemented which converts the constraints in the Sudoku to the following binary constraints.

2.1 Constraints

- Every cell cannot have two different values at the same time that is

$$\forall (i,j) \in \{0,1,\dots 9\}^2$$

$$[\neg(\text{grid}(i,j) = k \wedge \text{grid}(i,j) = k')\forall k, k' \in \{1,2,\dots 9\} \text{ and } k \neq k']$$

- Every cell must have atleast one value out of 1 to 9 that is

$$\forall (i,j) \in \{0,1,\dots 9\}^2 \left[\bigvee_{k=1}^{9} \text{grid}(i,j) = k\right]$$

- Two cells in same row or column or box can't have same values

$$\forall (i,j) \in \{0,1,\dots 9\}^2 \; \forall (i',j') \in \{0,1,\dots 9\}^2$$

$$\left[(i,j) \neq (i',j') \wedge \left(i = i' \vee j = j' \vee \left(\left\lfloor \frac{i}{3} \right\rfloor = \left\lfloor \frac{i'}{3} \right\rfloor \wedge \left\lfloor \frac{j}{3} \right\rfloor = \left\lfloor \frac{j'}{3} \right\rfloor\right)\right)\right]$$

$$\implies [\forall k \in \{1,2,\dots 9\} \neg (\text{grid}(i,j) = k \wedge \text{grid}(i',j') = k)]$$

2.2 Working

Now we can encode each constraint in this way:

$$\text{grid}(i,j) = k \longrightarrow X_{81i+9j+k}$$

The code encodes these from the input files and then decodes back the X_i's that are true and fills the grid based on these and then gets a solution.