CSP-solver for Sudoku

Because sudokus are still fun.

Ilse van der Linden & Sander van Dorsten

Quick run-through

- Our algorithm:
  - constraint propagation
  - variable ordening
- Our implementation
- Our experiment
- Relation to SAT-experiment

Our Algorithm

Algorithm

- Complete search
- Constraint propagation
  - Forward checking
- Variable ordening
  - minimal remaining values
  - degree heuristic

Constraint propagation

forward checking:

prevent mistakes by making the constraints involving the most recently instantiated variable arc consistent

Variable ordering

Minimal remaining values:

choose the variable with the smallest domain first.

Degree heuristic:

Select the variable that is involved in the largest number of constraints on other unassigned variables

for sudoku -> Select the variable that has the most unassigned variables in its constraints

The code

Initialize

```
for sudoku x:
     initialize CSP variables for a sudoku
     initialize CSP constraints for a sudoku
     for givens in x:
          initialize domain to [given]
```

```
{
 (1,1) : [1,2,3,4,5,6,7,8,9],
 (1,2) : [1,2,3,4,5,6,7,8,9],
 (1,3) : [3],
   :        :
   :        :
 (9,9) : [1,2,3,4,5,6,7,8,9]
}
```

figure 1: mapping of sudoku variable to it’s domain

Constraints

```
for sudoku x:
    initialize CSP variables for a sudoku
    initialize CSP constraints for a sudoku
    for givens in x:
        initialize domain to [given]
```

```
{          [Row, Col, Box]
 (1,1) : [C1, C2, C3],
 (1,2) : [C1, C4, C3],
 (1,2) : [C1, C4, C3],
   :        :
   :        :
 (9,9) : [C1, Cx, Cy]
}
```

figure 2: mapping of sudoku variable to it's constraints

The Constraints are All-Different constraints.

We save all the CSP data in a class **`Problem()`**

Solve

```
solve(problem):
    update_all_domains()                    // because of forward_checking
    return recursive_backtrack(problem)    // returns assignment, or False
```

```
recursive_backtrack(problem):
    unassigned_list = all unassigned variables in problem
    sort_list_by_heuristic(unassigned_list)// we sort our list depending on heuristics

    if unassigned_list is empty:
        return assignment                               // the assignment for the problem
    else:
        unassigned = unassigned_list.first()                    // grab first from list
        for value in unassigned.domain:
            new_problem = deepcopy(problem)
            new_problem.assignvalue(value)
            new_problem.update_domains(unassigned)              // ONLY update locally
            if new_problem.checkassigment() is true:
                result = recursive_backtrack(new_problem)
                if result is not False:
                    return result
    return False
```

Cases:

A: fc

B: fc + mrv

C: fc + dh

D: fc + mrv + dh

fc = forward checking, mrv = minimal remaining values, dh = degree heuristic

MRV vs DH

- splits on first ten sudokus
- best heuristic = MRV
- DH == MRV + DH?

#splits

	MRV	DH	MRV + DH	None
1	2928	30248	30248	48052
2	565	13198	13198	75537
3	9305	29735	29735	56094
4	54	175	175	17
5	400	3056	3056	242480
6	123	22	22	100
7	886	1094	1094	526
8	369	4370	4370	4360
9	200	40519	40519	373532
10	57	269	269	2079

FC = True in all of the above

choose heuristic MRV

The experiment

Conclusions from SAT project

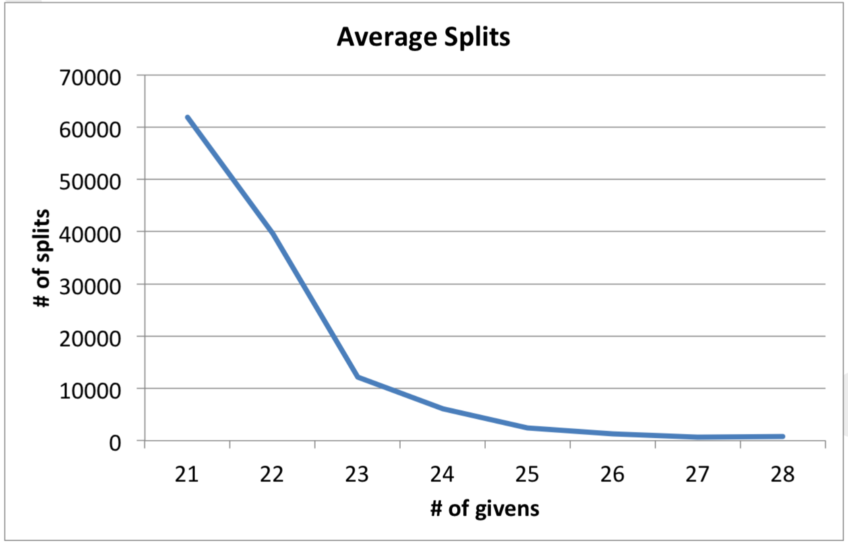
- fewer numbers given -> harder for humans
- fewer numbers given -> harder for SAT solver

- fewer numbers given -> harder for CSP solver?

Our results

- Difficulty: # of givens
- Steep curve
- Similar to SAT

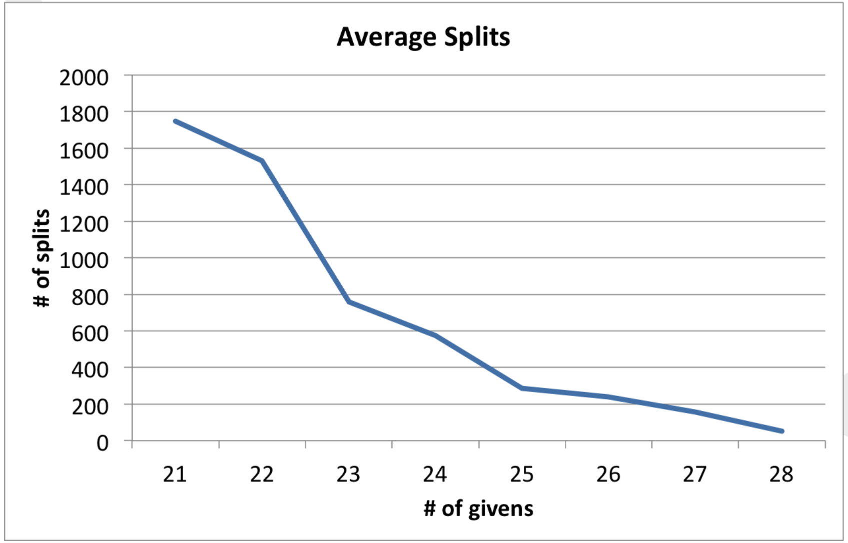
A: FC



Our results

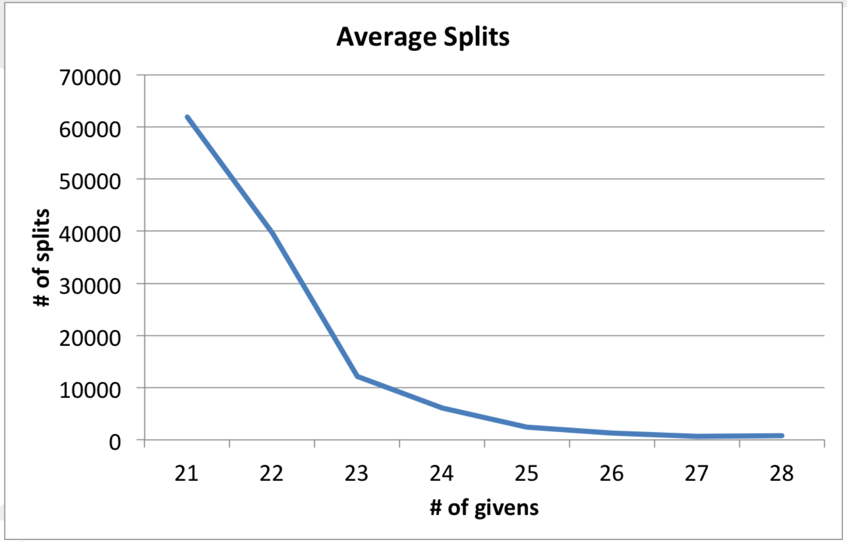
- Curve less steep
- # of givens makes less of a difference
- the harder the problem, the more MRV makes a difference

B: FC + MRV

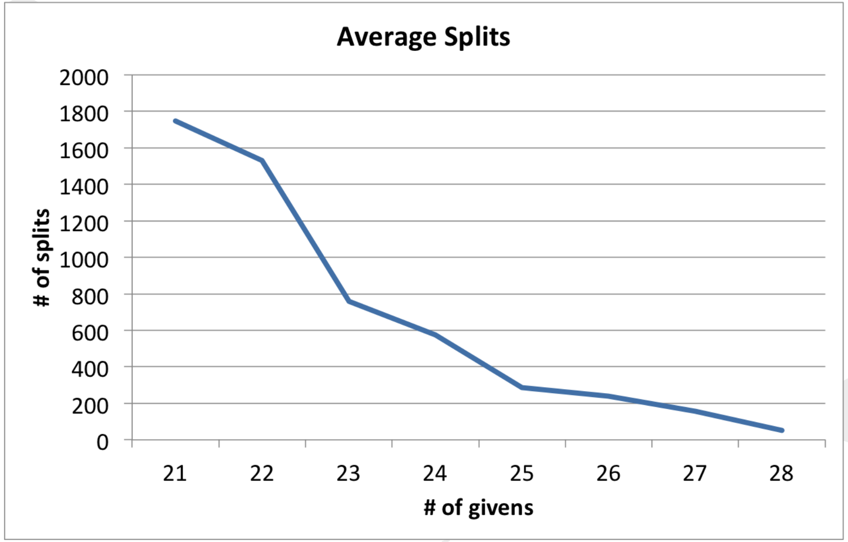


Our results

A: FC



B: FC + MRV



Relation to SAT experiment

Recall: Hypothesis

We predict that a sudoku with a **balanced frequency distribution** would be **easier** to solve by a SAT solver than a sudoku with an **unbalanced frequency distribution**, i.e. a **larger variance**.

Does this hold for a CSP solvers as well?

Thank you

Ilse van der Linden & Sander van Dorsten