Problem Solving and Search - Exercise 1

Filip Darmanovic (01527089), Julian Vecera (01627770),
Alexander Selzer (01633655)

May 2, 2020

`https://github.com/arselzer/SudokuCSPSolver`

Backtracking - Implementation

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

- Implemented using recursive method
- Uses helper method isPossible(..) which checks, if value can be placed in given field (according to Sudoku-rules).
- Checks every digit from 1 to 9
- Continues until Sudoku is solved
- Returns everytime, when it reaches a dead end (meaning it is not possible to place any digit from 1-9 in a remaining field)

Forward Checking - Implementation

- Based on backtracking method
- Uses helper method getCandidates(..) which puts all possibilities in a list
- Tries every possible candidate without immediately checking
- Can reach dead ends. Thus, backtracking must be used too!

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Forward Checking with Dynamic Ordering - Implementation

- Based on Forward Checking method
- Uses helper method to get most constrained variables
- Uses helper method to get list of least constraining values
- Tries every possible candidate and uses backtracking.

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Forward Checking with Dynamic Ordering - Implementation 2

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

- Most constrained variable
  - Counts by how many set values in row, column, square a field is constrained
  - Ordered by number of set values
  - Fills fields with higher which are more constrained first
- Least constraining value
  - Counts how often a digit occurs in grid
  - Ordered by number of occurrences
  - Tries digits with low occurrences first (more likely)
- Both are updated every time a new field is set

Backtracking - Performance

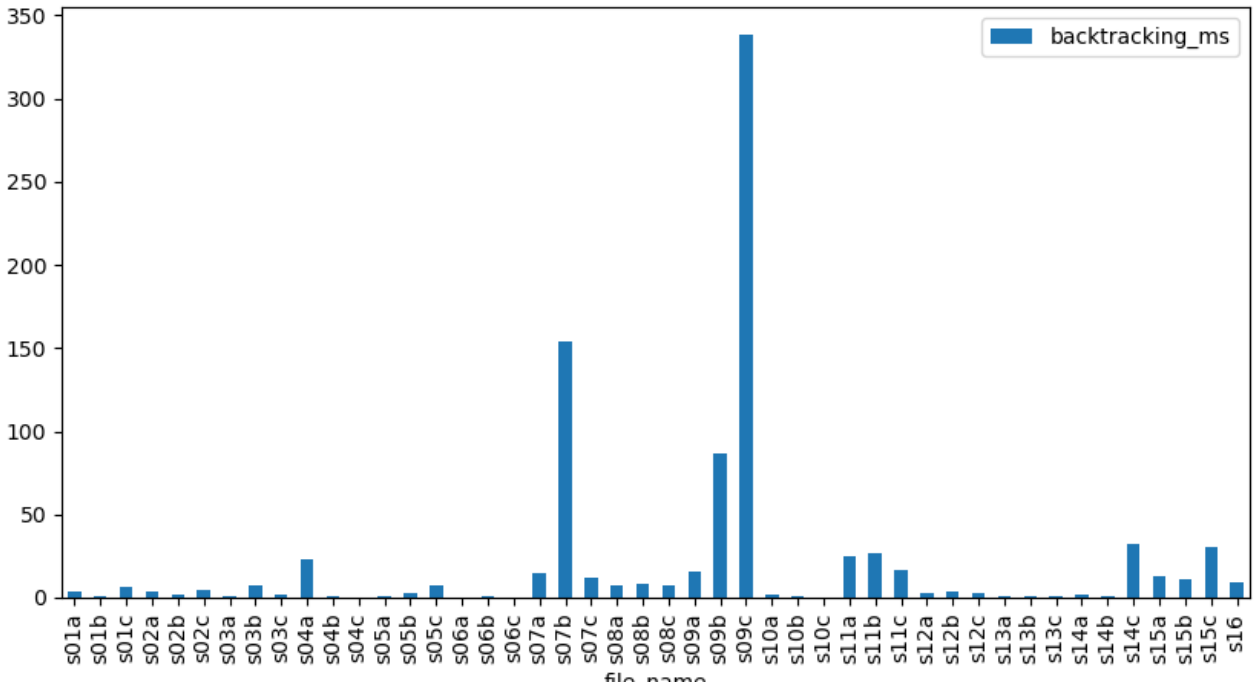

Figure: Backtracking performance on all instances

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Forward Checking - Performance

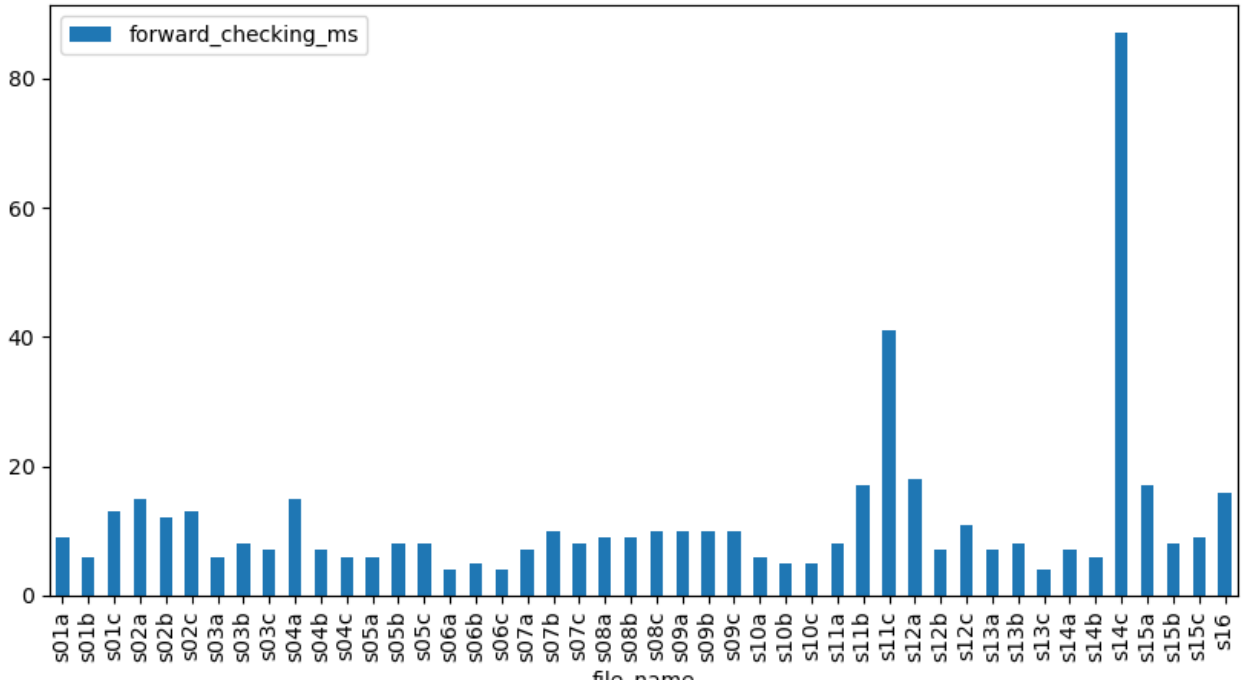

Figure: Forward Checking performance on all instances

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Forward Checking with Dynamic Ordering - Performance

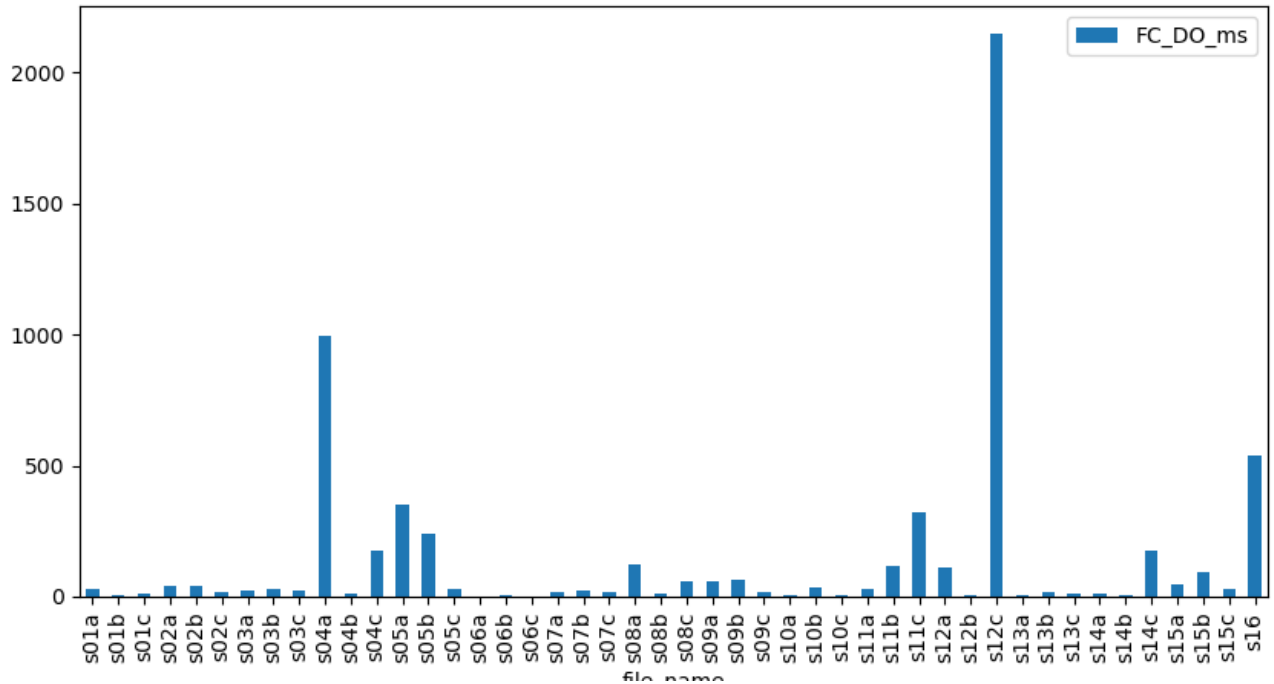

Figure: Forward Checking with DO performance on all instances

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Performance Comparison

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

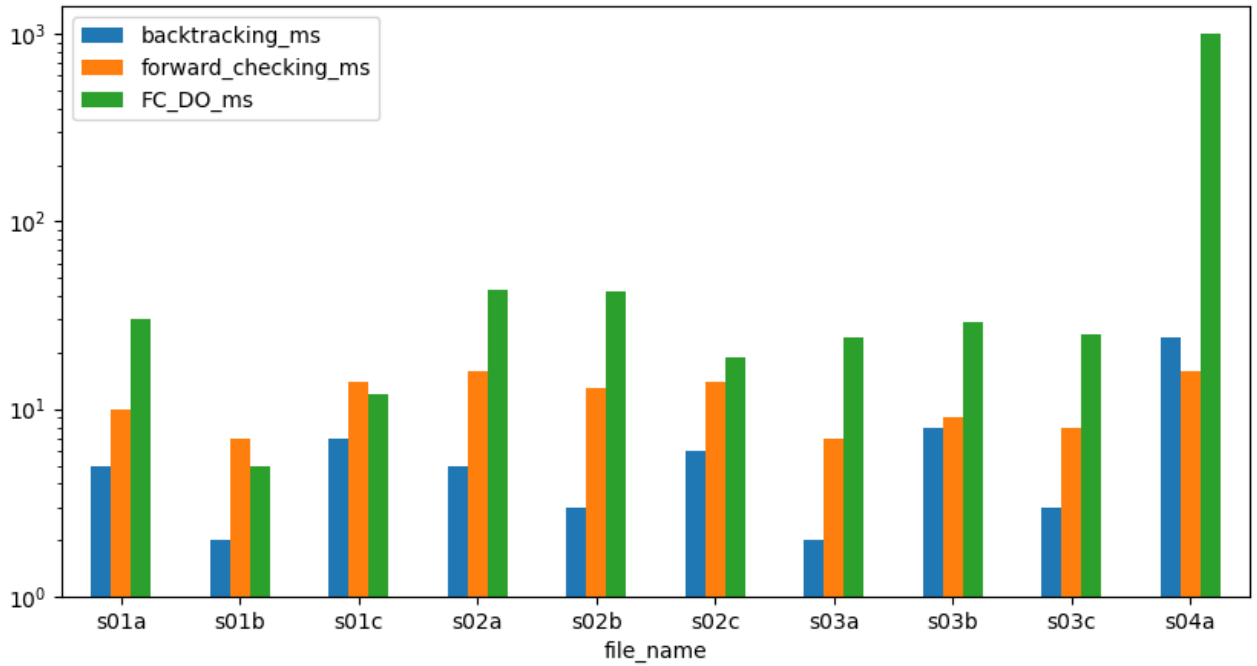

Figure: Performance comparison of different algorithms (log scale)

Performance Comparison

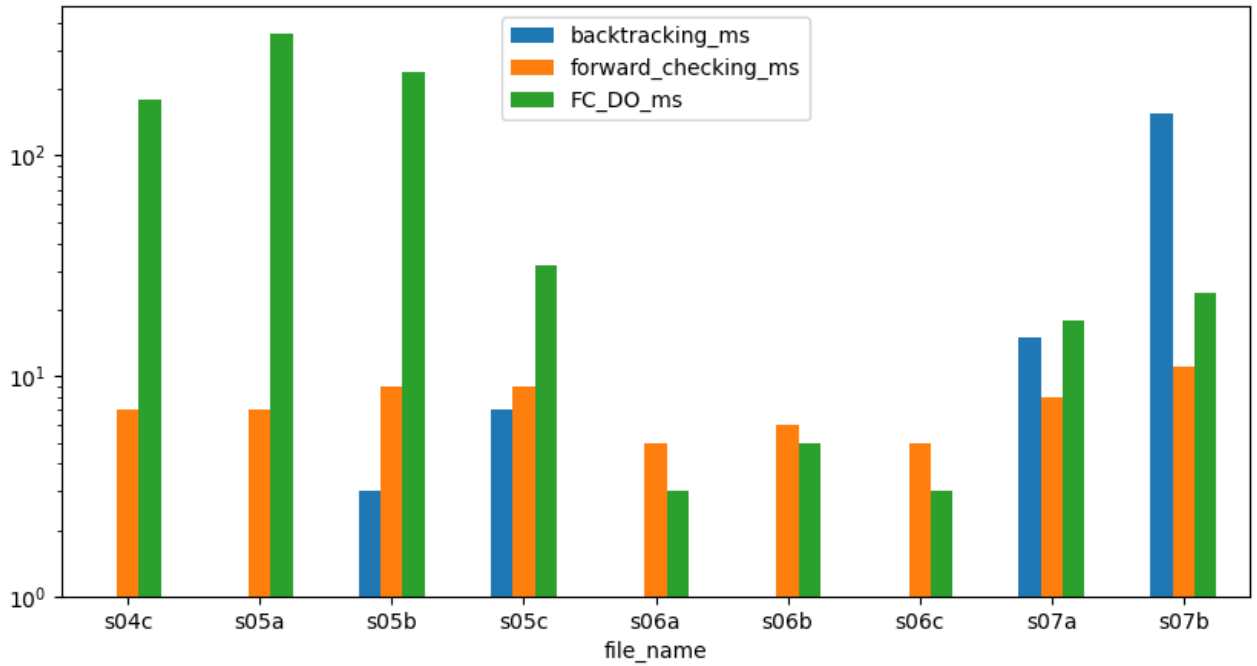

Figure: Performance comparison of different algorithms (log scale)

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Performance Comparison

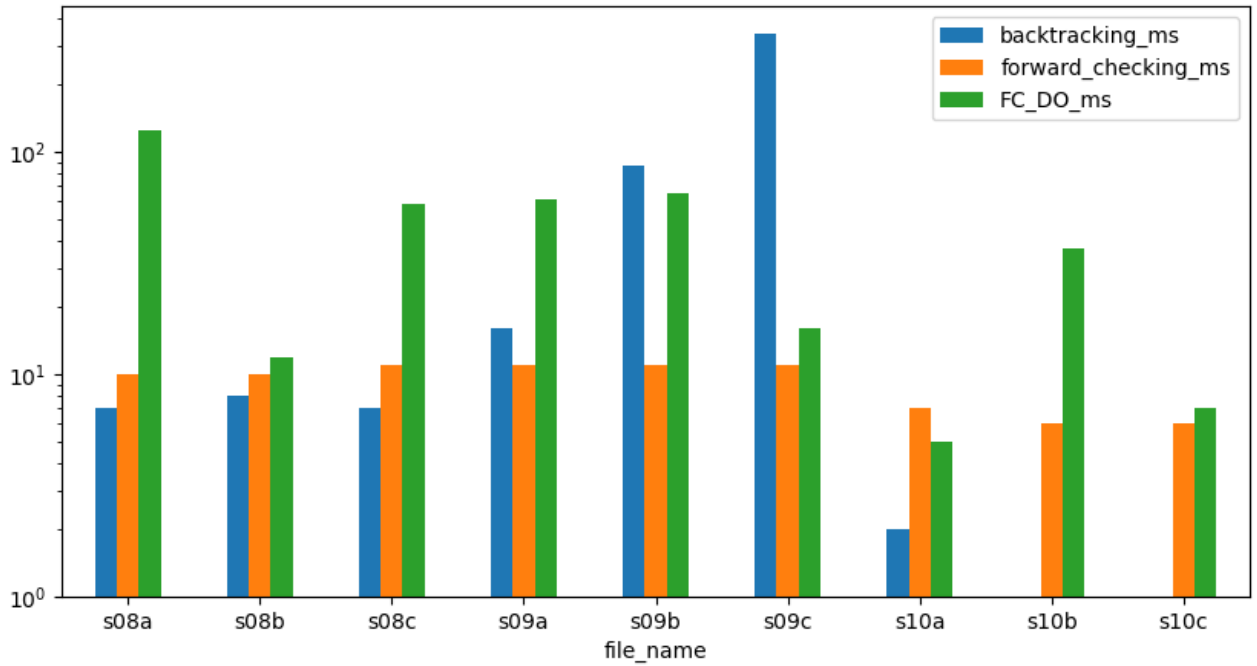

Figure: Performance comparison of different algorithms (log scale)

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Performance Comparison

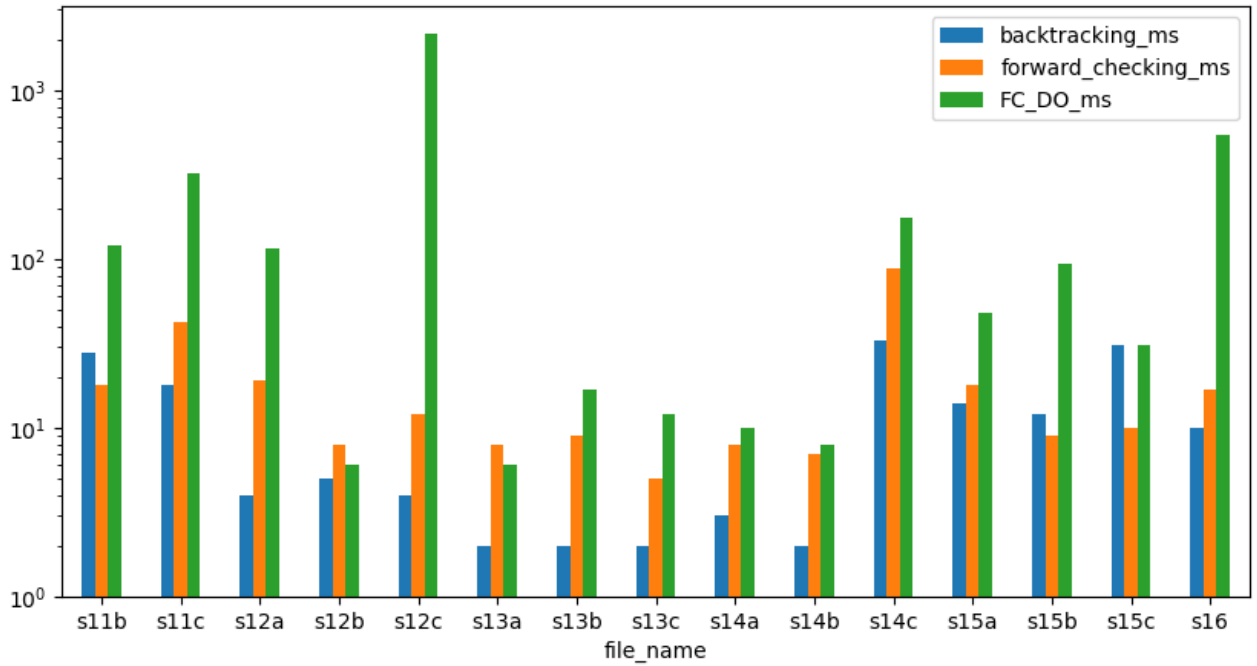

Figure: Performance comparison of different algorithms (log scale)

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Further Improvements

- Design decisions not optimal
- Explicit modelling of constraints would allow more effective optimization
- Efficiency of dynamic ordering - re-sorting the the bottleneck
- Extension to the general CSP problem
- Extension to 12x12 Sudokus

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)

Conclusion

- Forward checking performs best out of all implementations
- There is still room for further optimization
- The different algorithms have strengths and weaknesses and none is the best in all cases

Filip Darmanovic (01527089), Julian Vecera (01627770), Alexander Selzer (01633655)