AI - Lab 3

Shivam Mittal - 2015csb1032

10th March 2017

1 Task1 - Sudoku solver using CSP

1.1 Few terminologies

Algo 0 refers to BS, which is simple backtracking search.
Algo 1 refers to BS-I, which is backtracking search with minimum remaining value (MRV) heuristic.
Algo 2 refers to BS-II, which is backtracking search with minimum remaining value (MRV) and Least Constraining Value (LCV) heuristic.
Algo 3 refers to BS-MAC, which is backtracking search with Minimum Remaining Value (MRV) and Least Constraining Value (LCV) heuristic and constraint propagation using MAC (Maintaining Arc Consistency)

1.2 Algorithms and statistics

Running the different algorithms on sample sudoku puzzles. These statistics are measured during run time and are given as standard ouput by the program.

1.2.1 Algo 0 - BS

The term backtracking search is used for a depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign. It repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution. If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.
Statistics:

1. The time taken is : 21.5697 seconds.
2. The number of backtracks are : 227502770
3. Virtual Memory: 13296; Resident Set Space : 1556

1.2.2 Algo 1 - BS + MRV

The intuitive idea of choosing the variable with the fewest "legal" values is called the minimum-remaining-values (MRV) heuristic. It also has been called the "most constrained variable" or "fail-first" heuristic. The MRV heuristic usually performs better than a random or static ordering.
Statistics:

1. The time taken is : 0.937036 seconds.
2. The number of backtracks are : 734417
3. Virtual Memory: 13296; Resident Set Space : 1556

1.2.3 Algo 2 - BS + MRV + LCV

Once a variable has been selected, the algorithm must decide on the order in which to examine its values. For this, the least-constraining-value heuristic can be effective in some cases. It prefers the value that rules out the fewest choices for the neighboring variables in the constraint graph.
Statistics:

1. The time taken is : 1.32149 seconds.
2. The number of backtracks are : 684262
3. Virtual Memory: 13296; Resident Set Space : 1532

1.2.4 Algo 3 - BS + MRV + LCV + MAC

Inference can be even more powerful in the course of a search: every time we make a choice of a value for a variable, we have a brand-new opportunity to infer new domain reductions on the neighboring variables.
After a variable Xi is assigned a value, the MAC procedure procedure calls AC-3, but instead of a queue of all arcs in the CSP, we start with only the arcs (Xj , Xi) for all Xj that are unassigned variables that are neighbors of Xi . From there, AC-3 does constraint propagation in the usual way, and if any variable has its domain reduced to the empty set, the call to AC-3 fails and we know to backtrack immediately.
Statistics:

1. The time taken is : 5.49669 seconds.
2. The number of backtracks are : 185453
3. Virtual Memory: 13436; Resident Set Space : 3220

1.3 Comparisons and Inferences

1.3.1 Search Time

The time taken to solve all the sample sudoku puzzles.

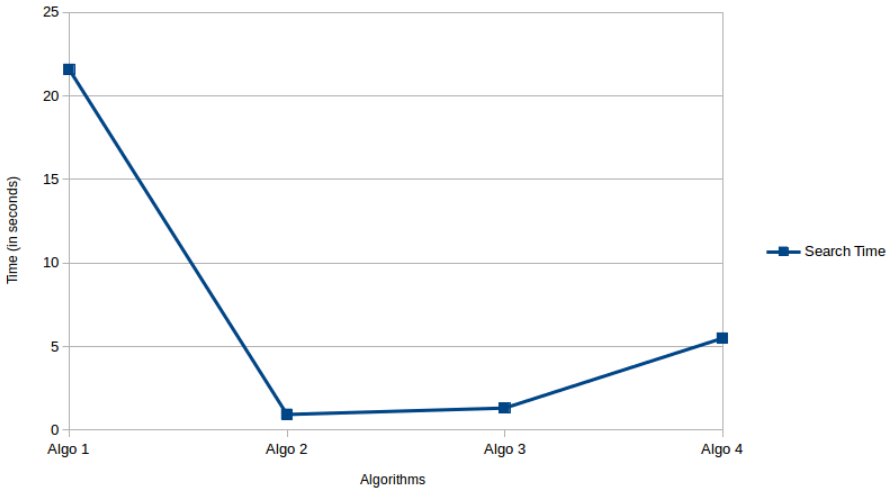

Figure 1: Search time of different algorithms

Explanation :

1. The general backtracking search is a blind search, it checks for all variables and all values, therefore checking many waste branches. Hence the search tree is very large and a lot of time is needed.

2. Backtracking search with MRV heuristic takes a lot less time because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree. If some variable X has no legal values left, the MRV heuristic will select X and failure will be detected immediately—avoiding pointless searches through other variables. Here, the benefit of the pruned tree by the heuristic is much more than the computation time of the heuristic.

3. The LCV heuristic chooses the value that is most likely to give a solution fast, it sure is more directional in the right direction. But the computation time is greater than the directional benefit it provides. Hence it takes a little more time than MRV.

4. In MAC, we infer new domain reductions after each assignment. One domain reduction takes time of order $Domain^2$. Performing this for all

neighbours of the assigned variable and then further propagating the constraint takes a lot of time. Here, the computational time is greater than the benefit of reduced domains.

1.3.2 Memory utilization

The virtual memory utilized in solving all the sample sudoku puzzles.

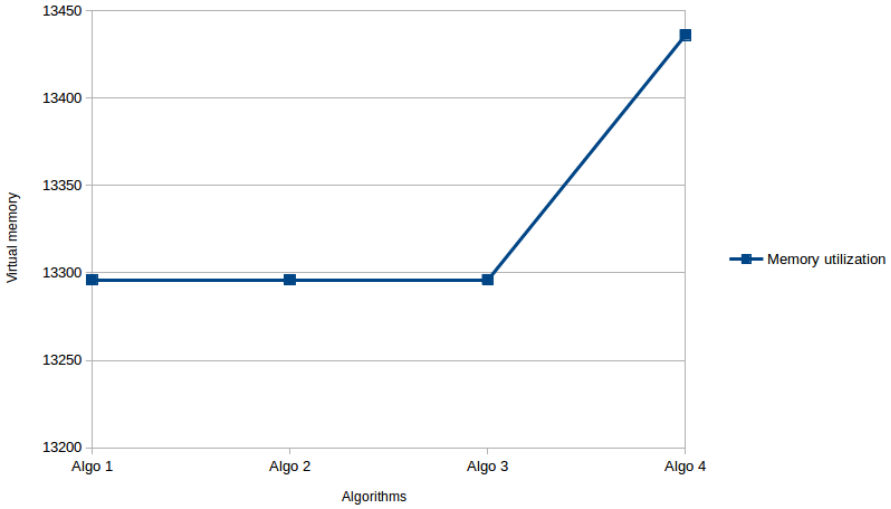

Figure 2: Memory utilization of different algorithms

Explanation :
The memory used in the algorithms depends largely on the implementation details. The memory trend here will be explained on the basis of the implementation done in my program.

1. The memory used by algo 1, 2 and 3 are the same because static domains are used in the program. There is no inference of domain reduction in the search. So, the memory used is the same in all the three cases.

2. The memory used by algo 4 is more than the other algorithms because we make inferences of domain reductions during the search and hence, make a copy of the domains (for backtracking) at various steps. Hence the memory used is more in MAC.

1.3.3 Number of backtracks

These are the total number of backtracks for all the sample sudoku puzzles summed together.

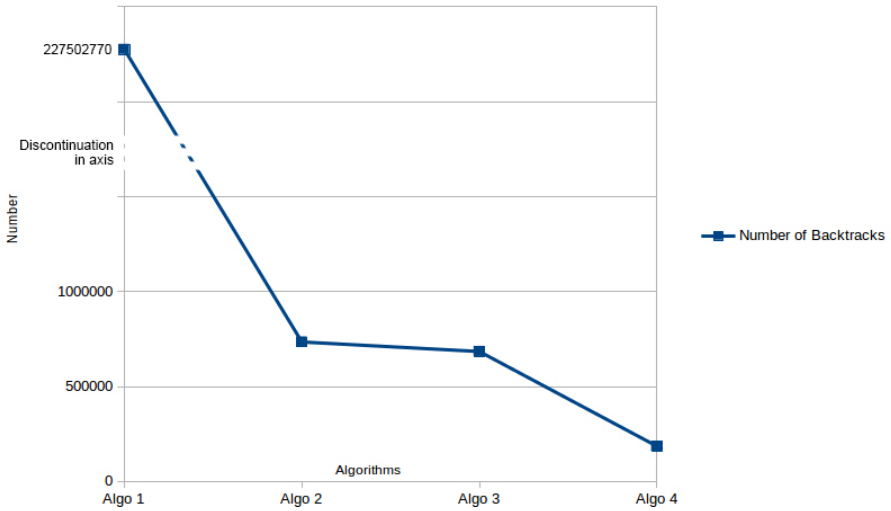

Figure 3: Number of backtracks of different algorithms

Explanation :

1. The number of backtracks in the general search is much much more than the other algorithms, it is greater than a multiplying factor of 300 compared other algorithms. This can be easily explained because the general backtracking search is a blind search, it checks for all variables and all values, therefore checking many waste branches. Hence the search tree is very large and there are large number of backtracks.

2. Backtracking search with MRV heuristic takes a lot less backtracks because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree.

3. The LCV heuristic chooses the value that is most likely to give a solution fast, it sure is more directional in the right direction. Hence, the backtracks are further reduced because we explore less branches.

4. In MAC, we infer new domain reductions after each assignment. If a domain of some variable is empty, then we backtrack. Since the domains of variables are reduced it results in early failure detection giving significantly lesser number of backtracks.

2 Sudoku solver using Minisat

2.1 Formulating the puzzle as boolean SAT problem

We have to formulate the sudoku problem as a boolean satisfiability problem. MiniSAT requires the boolean sentences to be in conjunctive normal form (CNF). We will first define the symbols and then, there are 4 types of constraints that we will formulate as clauses.

2.1.1 Defining the symbols

We define the symbols as x_{ijk} where:
i represents the row.
j represents the column.
k represents the value.
x_{ijk} is true, if sudoku[i][j] = k
x_{ijk} is false, if sudoku[i][j] != k

2.1.2 Cell constraints

Since, a cell should have atleast one value between 1-9. Exactly one of the x_{ij1}, x_{ij2},..., x_{ij9} should be true.
To formulate this, atleast one of the x_{ij1}, x_{ij2},..., x_{ij9} should be true.
Hence, the clause for this would be "x_{ij1} OR x_{ij2} OR ... OR x_{ij9}" . This clause would be for all $1 \leq i, j \leq 9$.
Example representation in DIMACS : (111 112 113 114 115 116 117 118 119 0)

Now, only one of the value should be assigned to one cell, this constraint will be automatically taken care by any of the row, column or box constraint, because if x_{ij1} and x_{ij2} are both true, then in no case the row, column and box can be filled.

2.1.3 Row constraints

Each value occurs at most once in the row. This can be represented by the conjunction of the clauses : "$-x_{rowcol1val}$ OR $-x_{rowcol2val}$" for all: $1 \leq row, col1, col2, val \leq 9$ and $col1 \neq col2$
Example representation in DIMACS: (-111 -121 0) and (-112 -122 0)

This takes care of the row constraints. Lets assume the contrary that i^{th} and j^{th} column of a^{th} row has same value x. Then, the clause "$-x_{aix}$ OR $-x_{ajx}$" (formulated above) would be false.

2.1.4 Column constraints

Each value occurs at most once in the column. This can be represented by the conjunction of the clauses : "$-x_{row1colval}$ OR $-x_{row2colval}$" for all: $1 \leq row1, row2, col, val \leq 9$ and $row1 \neq row2$
Example representation in DIMACS: (-111 -211 0) and (-111 -311 0)

This takes care of the column constraints. Lets assume the contrary that i^{th} and j^{th} row of a^{th} column has same value x. Then, the clause "$-x_{iax}$ OR $-x_{jax}$" (formulated above) would be false.

2.1.5 Box constraints

Each value occurs at most once in the box. This can be represented by the conjunction of the clauses : "$-x_{row1col1val}$ OR $-x_{row2col2val}$" for all: $1 \leq row1, row2, col1, col2, val \leq 9$ and (row1,col1) is in the box of (row2,col2)
Example representation in DIMACS: (-111 -221 0) and (-111 -231 0)

This takes care of the box constraints in the same way as explained with row and column constraints.

2.2 Statistics

1. The clauses in the CNF generated for one sudoku problem is 7394.
2. The number of symbols for one sudoku problem is 729.
3. The clause/symbol ratio is 10.142 which maybe considered over-constrained and hence the run time is less but giving solution for each.
4. The time taken for all the sample sudoku problems is 2.915 seconds.
5. Memory used for one sudoku problem is 16.25 MB