Applying Forward Checking With Fix Baseline System And Backtracking With Constraint Propagation To Solve A SuDoKu Puzzle

Zhiyuan He[1] and Zhengxian Lin[2]

***Abstract*— This paper mainly focus on applying forward checking with a fixed baseline system and backtracking with constraint propagation to solve a SuDoKu puzzle.**

I. INTRODUCTION

SuDoKu is a popular puzzle. It is a table which contains N x N cells. In general, the N is 9. Each cell of same row, column, and area of this table have to be filled by 9 distinct number, which are 1 to 9. The goal is to fill out all of them. This paper mainly focus on forward checking with fix baseline system, and backtracking with constraint propagation.In this paper, we will introduce how to design and implemented the both of algorithm and the two different ways of picking a cell, which is fixed baseline and Most Constrained Variable. This paper will also describe the experiment of running easy, medium, hard and evil SuDuKus and show the number of problems solved and the number of backtracks with each problem, the average number of filled-in numbers (in the beginning) for each of these types of problems, the effectiveness of rule subsets in reducing the search, the effectiveness of the most-constrained variable heuristic compared to fixed selection.

II. DESCRIPTION

A. Backtracking search

Backtracking search is 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. The algorithm is shown in Figure 6.5. 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.

In our project, the backtracking search is applied with the most constrained variable which is pick a slot that has the least number of values in its domain.And the backtracking search is also implemented with constraint propagation which will be introduced later. It maintains all possible candidate values, i.e., the current domain, for each cell after each assignment.

The pseudocode of backtracking search is showed in Fig.1.

```
function BACKTRACK(assignment,csp) returns
a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var,
assignment, csp) do
      if value is consistent with assignment then
        add var = value to assignment
        inferences ← INFERENCE(csp,var,value)
        if inferences != failure then
          add inferences to assignment
          result ← BACKTRACK(assignment,csp)
          if result != failure then
            return result
    remove {var = value} and inferences from assignment
  return failure
```

Fig. 1. Pseudocode of non-admissible heuristic function

B. Forward Checking

The Forward Checking algorithm consists in verifying, after each assignment of a value to a variable, all the constraints in which the variable appears. It helps reducing the domain of the free variables that appear in these constraints.

In Sudoku, each time a value is assigned to a variable, the value is removed from the domain of the free variables that are either in the same line, in the same column or in the same square as the assigned variable.

```
function FORWARDCHECKING(csp, cells) returns updated
domains
If a cell is filled with a number
  For every cells in the same column
      If domain contained the filled number
      Remove it from domain and return
  For every cells in the same row
      If domain contained the filled number
      Remove it from domain and return
  For every cells in the same box
      If domain contained the filled number
      Remove it from domain and return
```

Fig. 2. Pseudocode of forward checking function

[1]Zhiyuan He, Computer Science, Oregon State University. Email: hezhi@oregonstate.edu

[2]Zhengxian Lin,Computer Science, Oregon State University. Email: linzhe@oregonstate.edu

C. constraint propagation

The search begins by storing all possible values for each of the empty spots. Then it does constraint propagation through domain-specific inference rules. The following inference rules was applied in the given priority order, i.e., keep applying rule 1 as long as it applies. When it does not, apply rule 2, and go back to rule 1, and so on. The constraint propagation terminates when no rule is applicable, at which point a search action (assignment) is taken. At any search state, the program maintains the set of assignments made in that state, and the candidate numbers available in all other cells. Here are the inference rules to be tried in that sequence.

1) Naked Singles: If there is one cell that contains a single candidate, then that candidate is the solution for that cell.
2) Hidden Singles: Hidden Single means that for a given digit and house only one cell is left to place that digit. The cell itself has more than one candidate left, the correct digit is thus hidden amongst the rest.
3) Naked Pairs: A naked pair is two identical candidates in a particular row, column, or region.It is safe to eliminate those two numbers from all other cells in the row, column, or region the pair reside in.
4) Hidden Pairs: If you can find two cells within a house such as that two candidates appear nowhere outside those cells in that house, those two candidates must be placed in the two cells. All other candidates can therefore be eliminated.
5) Naked Triples:If you can find three cells that have only the same three candidates left, it is safe to eliminate that candidates from all other cells in that house.
6) Hidden Triples: Hidden Triples work in the same way as Hidden Pairs only with three cells and three candidates.

```
function CONSTRAINT-PROPAGATION(csp) returns
updated domains
if rule 1 can be performed
   perform rule 1
else if rule 2 can be performed
        perform rule 2 and go back to rule 1 check
else if rule 3 can be performed
        perform rule 3 and go back to rule 2 check
else if rule 4 can be performed
        perform rule 4 and go back to rule 3 check
else if rule 5 can be performed
        perform rule 5 and go back to rule 4 check
else if rule 6 can be performed
        perform rule 6 and go back to rule 5 check
else
return false
```

Fig. 3. Pseudocode of constraint propagation

III. EXPERIMENTAL SETUP

As mentioned previously, this paper experiment 77 different 9 * 9 size SuDoKus with 4 different complexity, easy, medium, hard and evil. And experiments are conducted with the following subsets of rules: 1.No inference; 2.Naked and Hidden Singles; 3.Naked and Hidden Singles and Pairs; 4.Naked and Hidden Singles, Pairs, and Triples.

And each SuDoKu problem will be experimented under these 4 subsets of rules for 2 different heuristics function (one is the fixed baseline and another is the most constrained variable function).The goal of this experiment is to answer several questions. The first question is whether the grade (complexity) of SuDoKu problems depend on the complexity of rules needed to solve them or just filled-in numbers (in the beginning) for each of these types of problems. The second question is how is the effectiveness of the most-constrained variable heuristic and fixed selection. The third question is how is the effectiveness of rule subsets in reducing the search and whether the number of backtracks is reduced by increased inference rules. The fourth question is how are the results and total time vary with the difficulty of the problems for fixed baseline backtracking and most constrained variable backtracking with different rule subsets.

```
For each backtracking search function,
    For fixed baseline function,
        For each of the 4 subsets of rules,
            For each of the SuDoKu problem,
                1) Solve p with forward checking until
                max search steps are used.
                2) Record needed data.
    For Most Constrained Variable function,
        For each of the 4 subsets of rules,
            For each of the SuDoKu problem,
                1) Solve p with forward checking until
                max search steps are used.
                2) Record needed data.
```

Fig. 4. The process of the experiment

IV. RESULTS

After the experiment, we get the results for former questions. For the question one, TABLE I show data of the grade of problem and the average number of filled-in numbers (in the beginning).

TABLE I
THE AVERAGE NUMBER OF FILLED-IN NUMBERS (IN THE BEGINNING) FOR DIFFERENT GRADES OF PROBLEMS

Grade of problem	The average number of filled-in numbers
Easy	34.69565217391305
Medium	29.047619047619047
Hard	26.27777777777778
Evil	26.133333333333333

And the the complexity of rules needed to solve every SuDoKu problems without backtracking is showed in TABLE II.

TABLE II

THE COMPLEXITY OF RULES FOR DIFFERENT GRADES OF PROBLEMS

Grade of problem	complexity of rules
Easy	can be all solved by single
Medium	can be all solved by single, pair and triple
Hard	can't be all solved by rules
Evil	can't be all solved by rules

The experiment also documents the number of problems solved and the number of backtracks with each problem by using backtracking with fixed baseline and Most Constrained Variable (MCV) with different rules subsets. And the results are showed in TABLE III.

TABLE III

THE NUMBER OF PROBLEMS SOLVED AND THE NUMBER OF BACKTRACKS WITH EACH PROBLEM

Problem with solution	problems solved	backtracks
Fixed Easy No inference	21	385.6
Fixed Easy Singles	23	0
Fixed Easy Singles and Pairs	23	0
Fixed Easy Singles,Pairs, Triples	23	0
Fixed Medium No inference	8	755.3
Fixed Medium Singles	21	0.7
Fixed Medium Singles and Pairs	21	0
Fixed Medium Singles,Pairs, Triples	21	0
Fixed Hard No inference	2	936.2
Fixed Hard Singles	18	4.6
Fixed Hard Singles and Pairs	18	0.7
Fixed Hard Singles,Pairs, Triples	18	0.05
Fixed Evil No inference	0	1000
Fixed Evil Singles	15	6.3
Fixed Evil Singles and Pairs	15	0.3
Fixed Evil Singles,Pairs, Triples	15	0.07
MVC Easy No inference	23	46.2
MVC Easy Singles	23	0
MVC Easy Singles and Pairs	23	0
MVC Easy Singles,Pairs, Triples	23	0
MVC Medium No inference	21	117.4
MVC Medium Singles	21	0.4
MVC Medium Singles and Pairs	21	0.09
MVC Medium Singles,Pairs, Triples	21	0
MVC Hard No inference	18	228.1
MVC Hard Singles	18	5.1
MVC Hard Singles and Pairs	18	1.5
MVC Hard Singles,Pairs, Triples	18	0.3
MVC Evil No inference	15	175.1
MVC Evil Singles	15	4.1
MVC Evil Singles and Pairs	15	0.9
MVC Evil Singles,Pairs, Triples	15	0.4

And the average number of backtracks with each problem by using backtracking with fixed baseline and Most Constrained Variable (MCV) with different rules subsets is showed in Fig. 5.

And the average number of backtracks with each problem by using backtracking with fixed baseline and Most Constrained Variable (MCV) with different rules subsets can also be represented as showed in Fig. 6.

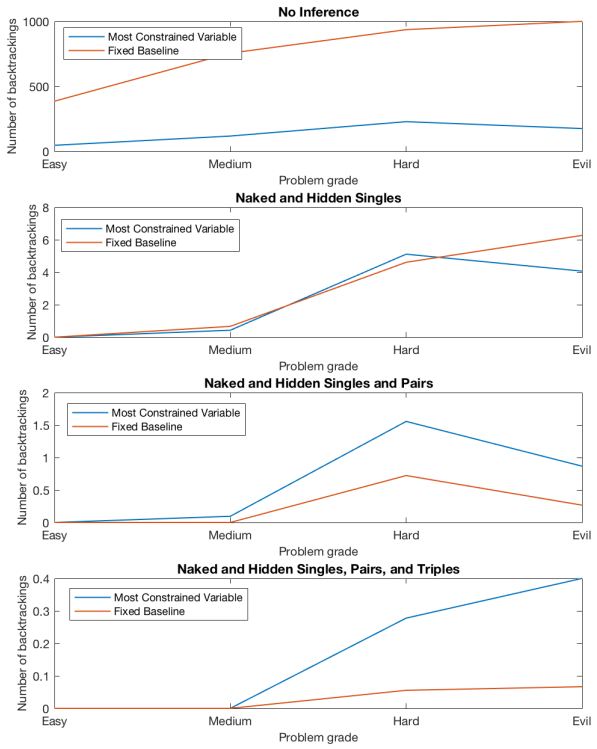

Fig. 5. The average number of backtracks with each problem by using backtracking with fixed baseline and Most Constrained Variable (MCV) with different rules subsets

And the average time for solving each problem by using backtracking with fixed baseline and Most Constrained Variable (MCV) with different rules subsets can also be represented as showed in Fig. 7.

V. DISCUSSION

In the discussion section, we will focus on answer the questions that mentioned in the experimental setup section.

1) whether the grade (complexity) of SuDoKu problems depend on the complexity of rules needed to solve them or just filled-in numbers (in the beginning) for each of these types of problems?
   As showed in TABLE II, we found that problems in Easy and Medium grade in this experiment all can be solved by just using inference rules without backtracking. Hard and Evil problems also can be solved by applying inference rules but can not be solved all. And the number of solved problems with different inferences is showed in following table.
   Form the TABLE IV, the grade of problem can not be determined by just checking whether it can be solved by different complexity rules but more easy the question is, more problems in the same grade can be solved by a same inference rule. As for the filled-in

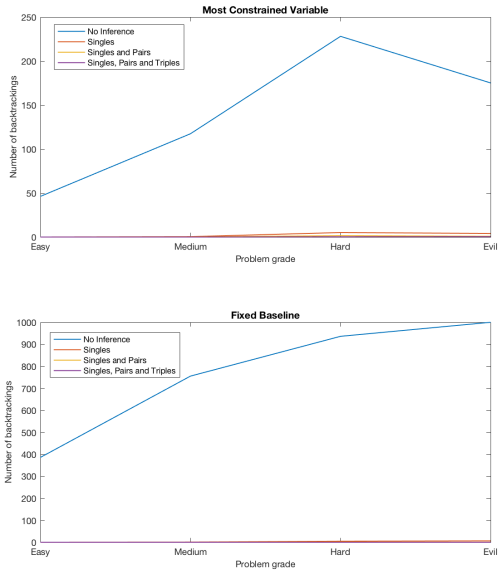

Fig. 6. The average number of backtracks with each problem by using backtracking with fixed baseline and Most Constrained Variable (MCV) with different rules subsets

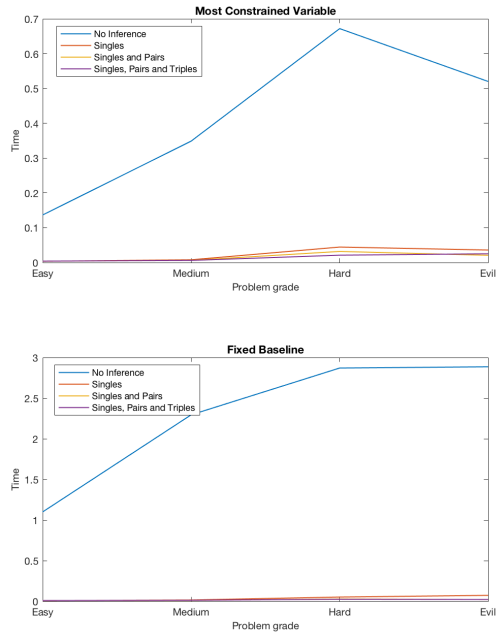

Fig. 7. The average time for solving each problem by using backtracking with fixed baseline and Most Constrained Variable (MCV) with different rules subsets

TABLE IV

THE NUMBER OF SOLVED PROBLEMS WITH DIFFERENT INFERENCES

Grade of problem and inferences	complexity of rules
Easy–single	23
Medium–single	12
Hard–single	2
Evil–single	3
Easy–single and pair	23
Medium–single and pair	20
Hard–single and pair	12
Evil–single and pair	6
Easy–single, pair and Triple	23
Medium–single, pair and Triple	21
Hard–single, pair and Triple	15
Evil–single, pair and Triple	10

numbers (in the beginning), as showed in TABLE I, the more average filled-in number in the beginning, the less difficulty the problem is.

2) How's the effectiveness of the most-constrained variable heuristic and fixed selection?
As showed in Fig.6, backtracking search with most-constrained variable heuristic have higher effectiveness than fixed selection when no inferences. But with more complex inference rules applying, the difference in effectiveness is getting smaller.

3) How's the effectiveness of rule subsets in reducing the search and whether the number of backtracks is reduced by increased inference rules?
As showed in Fig.5, it is obvious that more rule subsets, the less backtracking need to do. And the number of backtracks is reduced by increased inference rules.

4) How are the total time vary with the difficulty of the problems for fixed baseline backtracking and most constrained variable backtracking with different rule subsets?
As showed in Fig.7, the total time is reduced by increased inference rules.

5) Some interesting results:
There is a interesting finding in Fig.5, Fig.6 and Fig.7. The difference on effectiveness of Evil problems by using MCV sometimes higher than Hard problems. For example, as showed in Fig.5, when using single rules, as well as single and pair rules, Evil problem uses less backtracks.

REFERENCES

[1] S.J.Russell & P.Norvig, Artificial Intelligence: a modern approach, 3rd edition.