CS531 Programming Assignment 3: SuDoKu

Michael Lam, Xu Hu
EECS, Oregon State University

November 5, 2012

Abstract

In this assignment we design, implement and discuss constraint propagation and backtracking search algorithms in order to solve a specific constraint satisfication problem, SuDoKu.

1 Introduction

SuDoKu is a puzzle and constraint satisfication problem in which every unit (i.e. row, column or box) is an all-diff constraint. Each of the 81 squares can be represented as a variable on a domain of $\{1, 2, 3, ..., 9\}$. SuDoKu may be solved by backtracking search with constraint propagation.

A SuDoKu problem can be classified as easy, medium, hard or evil depending on what rules are required (and also if backtracking search is required) to solve the puzzle.

2 Algorithm

The algorithm consists of two major components: constraint propagation and backtracking search. In fact, the constraint propagation step is incorporated into the backtracking search. Some puzzles can be solved with just an application of the constraint propagation. Some puzzles require "sophisticated" rules (e.g. naked triples) to solve with just constraint propagation. Other puzzles require search and backtracking.

2.1 Constraint Propagation

The constraint propagation step applies the following rules to the SuDoKu board in this order to reduce the domain values for variables:

Rule 1 Assign to any cell a value x if it is the only value left in its domain.

Rule 2 Assign to any cell a value x if x is not in the domain of any other cell in that row, column or box.

Rule 3 In any row, column or box, find k squares that each have a domain that contains the same k numbers or a subset of those numbers. Then remove those k numbers from the domains of all other cells of that unit. (Called the "naked double" and "naked triple" rule for $k = 2$ and $k = 3$ respectively.)

Afterwards, these rules are "propagated" across the board, updating the board to be consistent with the new assignments. The process repeats until the board no longer updates (converges) or a variable is found to have an empty domain (contradiction).

In our experiments section, we will show the performance of the constraint propagation step when toggling a subset of these rules on or off.

The pseudocode for the constraint propagation step can be found below: (ALGORITHM)

2.2 Backtracking Search

Backtracking search is a depth first search algorithm. For each step of the algorithm, it picks a variable to assign a value and then performs constraint propagation as described above. If there is a contradiction after the constraint propagation step, the search discards that assignment and backtracks, making an alternative assignment. If the constraint propagation step was successful, then the search continues making another assignment. The search continues until it finds a complete assignment that satisfies the puzzle, otherwise returns a failure.

There are several heuristics that can be implemented for the backtracking search. For our project, there are two different heuristics for choosing the variable to assign:

1. Pick the most constrained square (i.e. the variable with the least number of domain values other than those already assigned)

2. Pick any random square

In fact, the first heuristic performs better than the second heuristic in general constraint satisfication problems because choosing an assignment from the most constrained variable will help prune the search tree faster. We will demonstrate the validity of this theory in our experiments.

The pseudocode for the backtracking algorithm can be found below: (ALGORITHM)

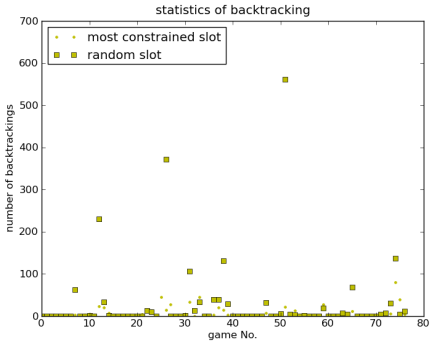
3 Experiments

We solved all the 77 problems. The first figure 1 shows the number of backtrackings for each problem. The "naked triple (double)" strategy exhibits impressive performance in constraint propagations. Without such kinds of rules, there are 1/4 problems need to use backtracking, no matter pick the most constrained slot or pick slot randomly. Meanwhile, only 2 or 3 problems require backtracking for the "naked triple" case.

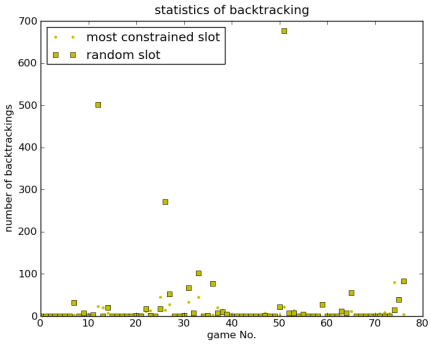
Next we show the problem completion in each difficulty level in table 1. From this table, we can see that the annotation for each problem are corrected approximately, except 3 evil problems which were solved by only simple rules. For comparing the effectiveness of most constrained slot heuristic and random slot heuristic, we make the same experiment with random slot heuristic, which placed in last 4 rows in table 1. It seems these two heuristics are getting same results. We will further do experiments for this comparison on number of rules. For justifying the difficulty of problems, the average number of filled-in numbers are concerned, as shown in table 2.

The number of rules being used is an important factor, we also use it to estimate the difficulty of a problem and show the average number of different kinds of rules used for each set of problems. The results is shown in table 3. The trendency is clear that more hard problems requires more number of rules. Note that for this experiment we use the combination of all rules and backtracking for making sure every problem can be solved.

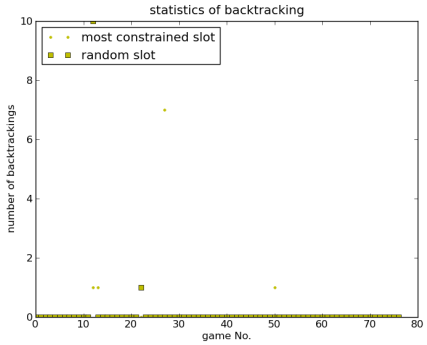
We further compare with two heuristics in the number of rules being taken. Figure 2 shows all pair of comparisons. By applying these two heuris-



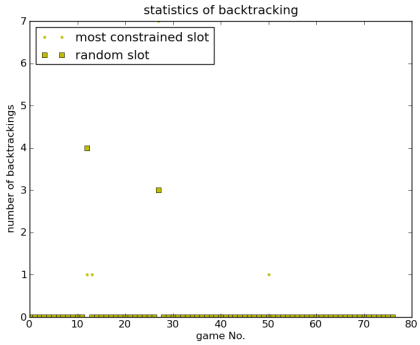
(a) Rule 1



(b) Rule 1,2



(c) Rule 1 + Naked double and triple



(d) Rule 1,2 + Naked double and triple

Figure 1: Number of backtracing with respect to each problem.

tics, a problem uses more or less the same number of rules. This is consistent with our experiment in number of complete problems.

4 Discussion

Discussion here.

Complexity	r1	r1,2	r1,2+n2	r1,2+n3	r1,2+n2,3	r1,2+bt	r1,2+n2,3+bt
Easy (23):	21	21	23	23	23	23	23
Medimum (21)	3	3	10	14	19	21	21
Hard (18)	0	0	7	4	15	18	18
Evil (15)	3	3	4	3	8	15	15
Easy rand:	21	21	23	23	23	23	23
Medimum rand	3	3	10	14	19	21	21
Hard rand	0	0	7	4	15	18	18
Evil rand	3	3	4	3	8	15	15

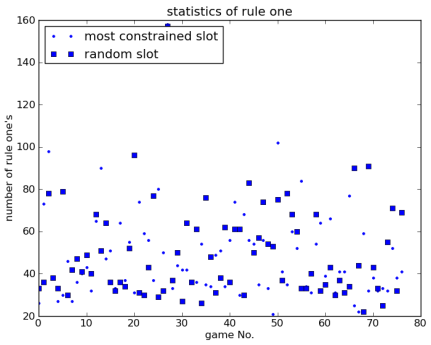
Table 1: Number of complete problems in different combinations of rules. The default heuristic is the most constrained slot. However, it seems picking most constrained slot or picking random slot is the same in this case. Here, r1: rule 1, r1,2: rule1 + rule2, n2: naked double, n3: naked triple, n2,3: n2 + n3.

	easy	medimum	hard	evil
average number	34.70	29.05	26.28	26.13

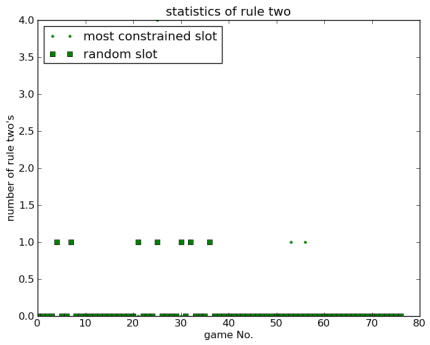
Table 2: Average filled-in numbers for each problem level.

	r1	r2	n2	n3	bt
Easy (23):	42.26	0.04	22.00	12.70	0
Medimum (21)	46.95	0.14	31.14	16.81	0
Hard (18)	49.06	0.33	32.11	15.94	0.06
Evil (15)	59.40	0	38.20	22.33	0.60

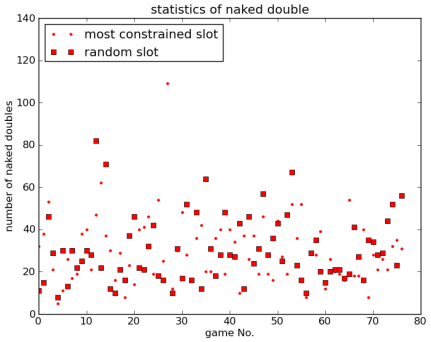
Table 3: Average number of rules used by problems in different levels. Here, r1: rule 1, r2: rule2, n2: naked double, n3: naked triple, bt: number of backtrackings.



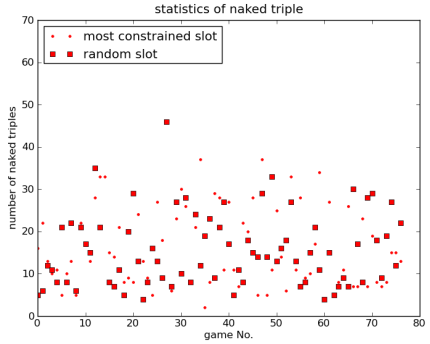
(a) Rule 1



(b) Rule 2



(c) Naked double



(d) Naked triple

Figure 2: Comparisons between most constrained slot heuristic and random slot heuristic in number of rules taken.