Mathematical Formalization of the Number Place Solving Algorithm

Abstract

This paper presents a formal mathematical formulation of a backtracking-based algorithm for solving the Number Place (Sudoku) puzzle. The algorithm is modeled as a constraint satisfaction problem (CSP), and the process of candidate generation, constraint propagation, and backtracking search is rigorously described.

1 Problem Formulation

1.1 Grid and Variables

Let $I = \{1, \ldots, 9\}$ be the set of row and column indices. Each cell of the 9×9 grid is represented by a variable $X_{r,c}$, where $r, c \in I$. The collection of all variables is

$$\mathcal{V} = \{X_{r,c} \mid r, c \in I\}.$$

Given digits (the clues) are represented by the set

$$\text{Givens} \subseteq I \times I \times I,$$

and the initial domain of each variable is defined as

$$D_{r,c}^{(0)} = \begin{cases} \{v\} & \text{if } (r, c, v) \in \text{Givens}, \\ \{1, \ldots, 9\} & \text{otherwise.} \end{cases}$$

1.2 Block Structure

Define the block index functions $B(r) = \lceil r/3 \rceil$ and $B(c) = \lceil c/3 \rceil$. Each 3×3 block is represented as

$$\text{Blk}(b_r, b_c) = \{(r, c) \in I^2 \mid B(r) = b_r,\ B(c) = b_c\}.$$

1.3 Constraints

The CSP is subject to the following constraints:

(Row constraint)	$\forall r, v: \ \lvert\{c \mid X_{r,c} = v\}\rvert \leq 1,$
(Column constraint)	$\forall c, v: \ \lvert\{r \mid X_{r,c} = v\}\rvert \leq 1,$
(Block constraint)	$\forall b_r, b_c, v: \ \lvert\{(r, c) \in \text{Blk}(b_r, b_c) \mid X_{r,c} = v\}\rvert \leq 1,$
(Domain constraint)	$\forall r, c: \ X_{r,c} \in \{1, \ldots, 9\}.$

2 Constraint Propagation and Candidate Refinement

2.1 State Representation

At any stage t, the algorithm maintains a family of candidate domains

$$\mathcal{D}^{(t)} = \{D_{r,c}^{(t)} \subseteq \{1,\dots,9\}\}_{r,c\in I}.$$

A *satisfied state* occurs when all $D_{r,c}^{(t)}$ are singletons and all constraints are satisfied. If any $D_{r,c}^{(t)} = \varnothing$, the state is inconsistent.

2.2 Forward Elimination

When a cell (r,c) is assigned v, the candidate sets are updated as:

$$\begin{aligned} D_{r,c'} &\leftarrow D_{r,c'} \setminus \{v\}, && \forall c' \neq c, \\ D_{r',c} &\leftarrow D_{r',c} \setminus \{v\}, && \forall r' \neq r, \\ D_{r',c'} &\leftarrow D_{r',c'} \setminus \{v\}, && \forall (r',c') \in \mathrm{Blk}(B(r),B(c)),\ (r',c') \neq (r,c). \end{aligned}$$

If any domain becomes a singleton, its unique value is immediately propagated using the same rule (this corresponds to forward checking).

2.3 Block-Based Candidate Initialization

For each block $\mathrm{Blk}(b_r,b_c)$, define the set of unused digits:

$$U_{b_r,b_c} = \{1,\dots,9\} \setminus \{X_{r,c} \mid (r,c) \in \mathrm{Blk}(b_r,b_c), |D_{r,c}| = 1\}.$$

Each undecided cell in the block initially receives candidates from U_{b_r,b_c}, which are then refined by removing digits already present in the same row or column.

2.4 Unique Occurrence Rule

For each block (and analogously for rows and columns), define

$$S_v = \{(r,c) \in \mathrm{Blk}(b_r,b_c) \mid v \in D_{r,c}\}.$$

If $|S_v| = 1$, the value v is fixed at that cell. This rule corresponds to identifying *hidden singles*.

3 Search and Backtracking

3.1 Variable Selection

Let $U = \{(r,c) \mid |D_{r,c}| > 1\}$ be the set of unsolved cells. The algorithm selects the variable with the fewest candidates (Minimum Remaining Values heuristic):

$$(r^*,c^*) \in \arg\min_{(r,c)\in U} |D_{r,c}|.$$

If multiple cells share the same minimal size, one is chosen at random.

3.2 Backtracking and Forward Checking

For each $v \in D_{r^*,c^*}$, assign $X_{r^*,c^*} = v$ and apply constraint propagation to obtain an updated domain family $\mathcal{D}'$. If any domain becomes empty, the branch fails and the algorithm tries the next candidate. Otherwise, recursion continues on $\mathcal{D}'$.

3.3 Termination

If all domains become singletons, a valid solution is found. If all candidates have been tested and failed, the algorithm backtracks to the previous step. To detect multiple solutions, the search would continue beyond the first solution, which is omitted in the present version.

4 Formal Algorithm

Algorithm 1 Recursive Sudoku Solver

function SOLVE($\mathcal{D}$)
  PROPAGATE($\mathcal{D}$) ▷ Apply rules in Sections 2.2–2.4
  if $\exists (r, c) : D_{r,c} = \varnothing$ **then**
    return FAIL
  if $\forall (r, c) : |D_{r,c}| = 1$ **then**
    return $\mathcal{D}$ ▷ Solution found
  Select $(r^*, c^*) = \arg\min_{(r,c):|D_{r,c}|>1} |D_{r,c}|$
  for all $v \in D_{r^*,c^*}$ **do**
    $\mathcal{D}' \leftarrow \text{Clone}(\mathcal{D})$
    $D'_{r^*,c^*} \leftarrow \{v\}$
    $R \leftarrow \text{SOLVE}(\mathcal{D}')$
    if $R \neq$ FAIL **then**
      return R
  return FAIL

5 Correctness and Complexity

5.1 Soundness

Each propagation step (row, column, and block elimination) removes only values inconsistent with the Sudoku constraints. Thus, any solution found by the algorithm satisfies all constraints.

5.2 Completeness

Because every unassigned cell is eventually tested with all possible candidates through backtracking, any existing solution will eventually be discovered.

5.3 Computational Complexity

In the worst case, the search tree has branching factor at most 9 and depth up to 81, yielding $O(9^{81})$ complexity. However, constraint propagation and MRV selection drastically reduce the effective search space in practice.

6 Conclusion

We have presented a formal description of the Number Place solving process as a constraint satisfaction problem with heuristic-guided backtracking. This formulation enables both theoretical analysis and efficient implementation.