Ant Colony Optimization for Solving Sudoku Puzzles

Neslisah Torosdagli
Computer Science Department
University Of Central Florida
Orlando, Florida
neslisah@knights.ucf.edu

ABSTRACT

In this research, NP-complete Sudoku, a combinatorial optimization problem, is solved using ant colony algorithm. The solution is provided in an hybrid manner using both analytical and ant colony algorithm steps in an iterative manner. In the analytical step, candidate values for unassigned cells are simplified, whereas, in the ant colony optimization step, an ant picks up a cell randomly from unassigned cells, decides on an assigment for this cell using probability of selection of candidate numbers according to pheromone accumulation of each candidate number, and starts the trip. At the end of the trip, pheromone value for the selection is updated according to validity of the trip. The algorithm runs in an iterative manner until a valid assignment for each cell is computed. Application is tested with puzzles provided in [3, 6].

Keywords

Sudoku; Ant Colony Optimization;

1. INTRODUCTION

Sudoku is a popular logical puzzle invented by Howard Garns in 1979 and published in Dell Magazines named as "Number Place". In 1986, the "Number Place" game is named as "Sudoku", in Japanese "the digits are limited to one occurrence." It became popular in late 2004s by becoming a regular daily feature in most of the newspapers and magazines all around the world, followed by many international Sudoku competitions.

Although there are a lot of variations of Sudoku puzzles, they are usually composed of 81 cells in a 9x9 grid format. Each cell should be assigned a number from the set [1..9] in such a way that this assigned number could not be repeated in any other cells in the column, row or 3x3 sub-grid of the assigned cell.

Eventhough, there is no commonly accepted method for rating difficulty level of a Sudoku puzzle, and a variety of difficulty levels are used by different organizations, they are usually rated a difficulty level in 1 to 5 (easy to evil). In addition to these regularly rated Sudoku puzzles, there are some specially designed Sudoku puzzles which have difficulty level more than 10 [6]. Dancing links, recursive, nondeterministic, depth-first, backtracking algorithm published by Donald Knutt in 2000[1], in addition to providing solution to the Sudoku puzzle, may be used to rate difficulty level of it.

Sudoku is generally a difficult puzzle, and to solve Sudoku a wide range of algorithms are applied in literature such as Brute-force, stochastic, genetic algorithms etc. Later, Mullaney [2] in 2006 applied ant colony optimization to provide a solution to Sudoku puzzle, however, his solution can provide solution for around %20 of the puzzles. Following Mullaney, both Sabuncu [3] and Schiff [5] published journal papers concurently in 2015 with promising performances in pretty small durations.

In this research, after development of application of ant colony optimization on Sudoku puzzles based on [3, 5], the performance is tested using puzzles provided in [3, 6] (provided in Appendix A and Appendix B respectively in this paper) , and the results are promising.

2. SOLVING SUDOKU PUZZLES

In this research, an hybrid solution is provided in order to solve Sudoku Puzzles. In an analytical step, which is refered as Simlification Step in the rest of the paper, candidate numbers for each cell is eliminated/simplified. In the beginning, all numbers in the range [1,9] is a candidate number, for each unassigned cell, according to assigned numbers in the row, column and 3x3 sub-grid of the unassigned cell, the set of candidate numbers for this unassigned cell is eliminated.

Rather than using a short integer for each candidate number for each cell, 9 bits of a short are utilized for this purpose, and candidate numbers are handled using bitwise operations. In my implementation, bit zero refers to number 1 being candidate or not, while bit eigth refers to number 9 being candidate or not.

Simplification for each cell in row-wise, column-wise and sub-grid-wise order continues until no more simplifications are possible. At this stage, Ant Colony Optimization is used to make further decisions on the puzzle where analytical method cannot make any further.

In the Ant Colony method, randomly an unassigned cell, which has at least 2 candidate numbers, is selected. Since there are more than 1 candidate for an unassigned cell, selection of an assignment is performed using pheromone acculation of each candidate number for that cell. A probability of selection is computed for each candidate according to its pheromone accumulation, and the candidate with maximum probability is assigned to the cell. In the beginning, all candidates for each cell has pheromone accumulation of zero, but after a few iterations, if an assignment results with a valid placement, pheromone accumulation of candidates used increases. Hence, spread search made in row-wise, column-wise and sub-grid-wise orders naturally converges to a solution by increasing pheromone accumulation of valid candidates and decreasing pheromone accumulation of in-

valid candidates.

Although, in [5], evaporation of pheromone accumulation is used, in this study, it is not applied.

2.1 Simplification

Algorithm 1 Row-wise Simplification

```
1: numberOfUpdates ← 0
2: for each row i do
3:    numbersInRow ← {}
4:    for each column j do
5:       c ← board.cells[i][j]
6:       if c.mValue > −1 then
7:          numbersInRow ← numbersInRow + {c.mValue}
8:    for each column j do
9:       c ← board.cells[i][j]
10:      if c.mValue == −1 then
11:         for each number in numbersInRow do
12:            if (bit[number] & c.mCandidates) != 0 then
13:               c.mCandidates ← c.mCandidates&resetBit[number]
14:               numberOfUpdates ← numberOfUpdates + 1
15:
16: return numberOfUpdates
```

Algorithm 2 Column-wise Simplification

```
1: numberOfUpdates ← 0
2: for each column j do
3:    numbersInColumn ← {}
4:    for each row i do
5:       c ← board.cells[i][j]
6:       if c.mValue > −1 then
7:          numbersInColumn ← numbersInColumn + {c.mValue}
8:    for each row i do
9:       c ← board.cells[i][j]
10:      if c.mValue == −1 then
11:         for each number in numbersInColumn do
12:            if bit[number] & c.mCandidates != 0 then
13:               c.mCandidates ← c.mCandidates&resetBit[number]
14:               numberOfUpdates ← numberOfUpdates + 1
15:
16: return numberOfUpdates
```

Simplification is performed using 3 functions provided in Algorithms [1-3]. The Row-wise Simplification algorithm,for each row, makes a set of assigned numbers, and then resets corresponding bit of the candidate number of the unassigned cell according to set of assigned numbers for that row. Similarly, column-wise and subgrid-wise algorithms performs eliminations in accordance with their names.

In the simplification step, these 3 functions are called iteratively in an infinite loop, until no further elimination is possible.

Algorithm 3 Subgrid-wise Simplification

```
1: numberOfUpdates ← 0
2: for each sub-grid k do
3:    numbersInSubgrid ← {}
4:    for each row i in subgrid k do
5:       for each column j in subgrid k do
6:          c ← board.cells[i][j]
7:          if c.mValue > −1 then
8:             numbersInSubgrid ← numbersInSubgrid + {c.mValue}
9:    for each row i in subgrid k do
10:      for each column j in subgrid k do
11:         c ← board.cells[i][j]
12:         if c.mValue == −1 then
13:            for each number in numbersInSubgrid do
14:               if bit[number] & c.mCandidates != 0 then
15:                  c.mCandidates ← c.mCandidates&resetBit[number]
16:                  numberOfUpdates ← numberOfUpdates + 1
17:
18: return numberOfUpdates
```

Algorithm 4 Simplification

```
1: while (true) do
2:    numberOfUpdates ← 0
3:    numberOfUpdates ← numberOfUpdates + rowWiseSimplify()
4:    numberOfUpdates ← numberOfUpdates + columnWiseSimplify()
5:    numberOfUpdates ← numberOfUpdates + subGridWiseSimplify()
6:    if numberOfUpdates == 0 then
7:       break
```

2.2 Ant Colony Optimization

After no further elimination can be computed in the simplification step, ant colony step is performed. In the ant colony step, randomly selected unassigned cell is assigned a number according to pheromone accumulation of its candidate numbers. According to validity of the assignment, pheromone values are updated, either incremented as award, or decremented as punishment.

It is also probable that, after application of a set of assignments, an infeasible/invalid board may be generated, in that case, board is set to initial state without resetting pheromone accumulations of the candidates, as pheromone accumulations guide the algorithm to converge to the valid board.

In my implementation, award is used as 1, while punishment is used as 0.005. This choice works optimum for a varity of puzzles tested. Since probability of invalid board is much higher than probability of valid board, when punishment and award are used the same value, the algorithm converges in much longer duration.

Algorithm 5 Ant Colony Optimization

```
award ← 1
punishment ← 0.005
unassignedSet ← {}
initialBoard ← board
for row i in 0-9 do
    for column j in 0-9 do
        c ← board.cells[i][j]
        if c.mValue == -1 then
            unassignedSet ← unassigned + {c}
selectedUCell ← Random(unassignedSet)
candidates ← selectedUCell.mCandidates
maxPheromone ← −MAX_FLOAT
aNumber ← −1
for k in 0-9 do
    if bit[k]&candidates == 1 then
        pheromone[k] ← selectedUCell.mPheromone[k]
        if pheromone[k] > maxPheromone then
            maxPheromone ← pheromone[k]
            aNumber ← k
if assignedNumber > -1 then
    selectedUCell.mValue ← aNumber
    if Valid(board) then
        selectedUCell.mPheromone[aNumber]+ =
        award
    else
        selectedUCell.mPheromone[aNumber] =
        max(selectedUCell.mPheromone[aNumber] −
        punishment, 0)
        selectedUCell.mValue ← −1
        selectedUCell.mCandidates ←
        resetBits[aNumber]&selectedUCell.mCandidates
else
    for row i in 0-9 do
        for column j in 0-9 do
            board.mCells[i][j].mValue ←
            initialBoard.mCells[i][j].mValue
            board.mCells[i][j].mCandidates ← 0x1FF
```

Algorithm 6 Sudoku Solver

```
board ← read()
board.print()
while (!Valid(board)) || (!board.completed())) do
    Simplify()
    AntColonyOptimization()
board.print()
```

3. EXPERIMENTAL RESULTS

The algorithm is implemented in C++ on MacBook Pro 2.6 GHz Intel Core i5 with 8GB of memory. The project is developed using IDE XCode on MACOS.

The running durations for the puzzles provided in Appendix A and Appendix B are as follows:

Table 1: Puzzle Performances

Puzzle Name	Duration (sec.)
Puzzle 1	0
Puzzle 2	0.25
Puzzle 3	7.45
Puzzle 4	0.77
Puzzle 5	0
Puzzle 6	4.03
Puzzle 7	1.1
Puzzle 8	6.8
Puzzle 9	1.72
Puzzle 10	0
Puzzle 11	1092.11

4. CONCLUSIONS

In this research, an hybrid algorithm of Ant Colony Optimization is applied on NP-complete puzzle Sudoku in an iterative manner. In the simplification step, candidate numbers for each cell is eliminated according to assigned numbers in the row, column, and sub-grid of the cell in question. After no further elimination can be computed, ant colony step is performed. In the ant colony step, randomly selected unassigned cell is assigned to one of its candidates according to probability of selection computed using pheromone accumulation of the cell's candidates. According to validity of the assignment, pheromone values are updated. The algorithm runs until all cells are assigned a valid number. Puzzles provided in [3, 5] are executed. To sum up, although a very basic implementation of Ant Colony Optimization is introduced in this paper, strength of Ant Colony based method on solving NP-Complete Sudoku problem is proved. This research may be improved by observing type of Sudoku puzzles this basic implementation is more successful to solve, and type of Sudoku puzzles it is slower to solve. After such a classification, performance of the algorithm can definitely be improved.

5. REFERENCES

[1] D. Knuth. Dancing links. *Millennial Perspectives in Computer Science*, pages 187–214, 2000.

[2] D. Mullaney. Using ant systems to solve sudoku problems. http://ncra.ucd.ie/comp40580/crc2006/mullaney.pdf, 2006.

[3] I. Sabuncu. Solving sudoku puzzles using hybrid ant colony optimization algorithm. *Proceedings of the 1st International Conference on Industrial Networks and Intelligent Systems*, 2015.

[4] S. Salas and E. Hille. *Calculus: One and Several Variable*. John Wiley and Sons, New York, 1978.

[5] K. Schiff. An ant algorithm for the sudoku problem. *Journal of Automation, Mobile Robotics and Intelligent Systems JAMRIS*, pages 24–27, 2015.

[6] Telegraph. World's hardest sudoku: Can you crack it? http://www.telegraph.co.uk/news/science/science-news/9359579/Worlds-hardest-sudoku-can-you-crack-it.html, November 2015.

APPENDIX

A. SUDOKU PUZZLES - I

This appendix lists the test puzzles which are provided in [3].

Puzzle 1

1	2	7	6	-	-	4	8	5
8	4	-	1	-	5	-	-	7
-	9	5	7	4	-	3	-	2
2	6	9	-	-	-	5	-	-
-	-	-	8	5	-	6	4	-
-	5	-	-	7	-	2	-	1
3	1	4	-	-	-	-	2	-
-	-	6	2	3	7	-	-	-
-	-	-	-	6	-	8	5	-

Puzzle 2

7	5	-	9	8	-	-	-	-
6	-	-	-	5	-	-	-	8
-	-	-	-	-	-	4	2	-
-	-	-	3	9	5	-	-	-
-	2	3	-	-	-	-	8	1
-	-	-	8	-	-	-	5	-
-	-	4	-	-	-	3	-	-
-	-	-	-	7	9	-	-	-
-	-	8	-	-	-	-	1	2

Puzzle 3

8	4	2	-	-	-	-	-	-
-	-	-	5	-	1	7	-	-
-	-	-	-	-	-	3	8	-
9	5	-	-	-	-	-	-	2
-	-	-	-	5	-	-	-	-
-	-	-	-	-	-	-	4	6
1	-	-	-	-	7	4	-	-
-	-	8	-	6	-	-	-	-
-	-	4	-	-	-	-	3	8

Puzzle 4

5	-	9	6	-	-	-	-	-
-	3	-	8	-	7	9	2	-
-	-	-	3	-	-	8	-	-
-	-	-	-	1	6	-	8	-
-	5	-	-	-	-	-	1	-
-	-	-	-	-	-	-	3	2
1	-	4	-	3	-	-	-	-
-	-	6	7	-	9	-	-	-
-	-	-	-	-	-	-	-	3

Puzzle 5

-	-	4	-	8	6	-	-	-
9	-	3	4	7	-	1	-	-
-	8	2	5	-	-	-	6	7
-	9	-	8	-	-	3	-	2
-	5	-	-	-	-	-	4	-
2	-	6	-	-	1	-	5	-
3	4	-	-	-	9	2	7	-
-	-	5	-	3	4	8	-	6
-	-	-	7	1	-	5	-	-

Puzzle 6

-	-	5	3	-	-	-	-	-
8	-	-	-	-	-	-	2	-
-	7	-	-	1	-	5	-	-
4	-	-	-	-	5	3	-	-
-	1	-	-	7	-	-	-	6
-	-	3	2	-	-	-	8	-
-	6	-	5	-	-	-	-	9
-	-	4	-	-	-	-	3	-
-	-	-	-	-	9	7	-	-

Puzzle 7

-	-	-	-	6	8	-	-	-
-	2	-	7	-	-	5	-	-
-	-	-	-	4	-	-	2	-
8	-	-	4	-	-	-	-	3
3	-	-	-	8	9	-	7	-
4	6	1	-	-	-	-	-	-
-	7	6	-	-	-	9	-	-
-	-	-	-	-	-	-	-	8
-	-	-	-	-	1	6	-	-

Puzzle 8

2	-	-	-	9	3	7	-	-
5	-	8	-	-	-	-	-	-
-	6	7	-	-	-	-	-	-
-	9	-	-	-	4	-	2	5
-	-	-	-	-	-	9	-	7
-	-	-	-	8	-	-	-	-
-	-	-	-	-	5	4	-	-
-	-	-	3	-	1	-	5	-
-	7	-	8	-	-	-	6	-

Puzzle 9

6	-	-	-	4	-	-	1	-
-	1	-	-	-	-	-	-	3
-	-	2	-	-	8	-	4	-
-	2	-	-	-	-	-	-	4
-	-	7	3	8	2	6	-	-
5	-	-	-	-	-	-	2	-
-	9	-	5	-	-	1	-	-
4	-	-	-	-	-	-	7	-
-	5	-	-	9	-	-	-	2

Puzzle 10

-	3	-	-	4	6	2	-	-
8	-	-	3	1	-	7	4	-
-	2	-	-	-	8	-	-	-
4	1	-	-	-	-	6	-	-
-	-	-	-	7	1	8	5	2
5	8	2	-	3	-	-	7	4
3	-	1	5	-	4	9	2	-
-	-	5	-	6	7	-	3	-
-	4	8	2	9	-	5	-	7

B. SUDOKU PUZZLES - II

This appendix lists the test puzzle which is provided in [6]. This puzzle is prepared by Finnish mathematician Arto Inkala. It is specifically designed to be unsolvable to many and very difficult even to the sharpest minds.

Puzzle 11

8	-	-	-	-	-	-	-	-
-	-	3	6	-	-	-	-	-
-	7	-	-	9	-	2	-	-
-	5	-	-	-	7	-	-	-
-	-	-	-	4	5	7	-	-
-	-	-	1	-	-	-	3	-
-	-	1	-	-	-	-	6	8
-	-	8	5	-	-	-	1	-
-	9	-	-	-	-	4	-	-