

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 assignment 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 2000s 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 concurrently 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 Simplification 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 eighth 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 accumulation 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: 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: return numberOfUpdates

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: return numberOfUpdates

Algorithm 4 Simplification

1: while (true) do
2: numberOfUpdates ← 0
3: rowWiseSimplify() ← numberOfUpdates + rowWiseSimplify()
4: columnWiseSimplify() ← numberOfUpdates + columnWiseSimplify()
5: subGridWiseSimplify() ← 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 variety 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

1: award ← 1
2: punishment ← 0.005
3: unassignedSet ← \{\}
4: initialBoard ← board
5: for row i in 0-9 do
6:   for column j in 0-9 do
7:     c ← board.cells[i][j]
8:     if c.mValue == -1 then
9:       unassignedSet ← unassigned + \{c\}
10:  selectedUCell ← Random(unassignedSet)
11:  candidates ← selectedUCell.mCandidates
12:  maxPheromone ← -MAX_FLOT
13:  aNumber ← -1
14:  for k in 0-9 do
15:     if bit[k] & candidates == 1 then
16:       pheromone[k] ← selectedUCell.mPheromone[k]
17:     if pheromone[k] > maxPheromone then
18:       maxPheromone ← pheromone[k]
19:      aNumber ← k
20:  if aNumber > -1 then
21:    selectedUCell.mValue ← aNumber
22:  if Valid(board) then
23:    selectedUCell.mPheromone[aNumber] += award
24: else
25:    selectedUCell.mPheromone[aNumber] =
26:    max(selectedUCell.mPheromone[aNumber] - punishment, 0)
27:    selectedUCell.mValue ← -1
28:    selectedUCell.mCandidates ←
29:    resetBits[aNumber] & selectedUCell.mCandidates
30:  else
31:    for row i in 0-9 do
32:      for column j in 0-9 do
33:       board.mCells[i][j].mValue ←
34:       initialBoard.mCells[i][j].mValue
35:      board.mCells[i][j].mCandidates ← 0x1FF

Algorithm 6 Sudoku Solver

1: board ← read()
2: board.print()
3: while (!Valid(board)) || (!board.completed()) do
4:    Simplify()
5:    AntColonyOptimization()
6: 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>
<thead>
<tr>
<th>Puzzle Name</th>
<th>Duration (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Puzzle 1</td>
<td>0</td>
</tr>
<tr>
<td>Puzzle 2</td>
<td>0.25</td>
</tr>
<tr>
<td>Puzzle 3</td>
<td>7.45</td>
</tr>
<tr>
<td>Puzzle 4</td>
<td>0.77</td>
</tr>
<tr>
<td>Puzzle 5</td>
<td>0</td>
</tr>
<tr>
<td>Puzzle 6</td>
<td>4.03</td>
</tr>
<tr>
<td>Puzzle 7</td>
<td>1.1</td>
</tr>
<tr>
<td>Puzzle 8</td>
<td>6.8</td>
</tr>
<tr>
<td>Puzzle 9</td>
<td>1.72</td>
</tr>
<tr>
<td>Puzzle 10</td>
<td>0</td>
</tr>
<tr>
<td>Puzzle 11</td>
<td>1092.11</td>
</tr>
</tbody>
</table>

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

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

**Puzzle 7**

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

**Puzzle 8**

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>-</th>
<th>-</th>
<th>4</th>
<th>-</th>
<th>-</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>-</td>
<td>4</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>9</td>
<td>5</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>-</td>
<td>9</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
</tbody>
</table>

Puzzle 10

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>3</th>
<th>-</th>
<th>4</th>
<th>6</th>
<th>2</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>1</td>
<td>5</td>
<td>-</td>
<td>4</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>-</td>
<td>5</td>
<td>-</td>
<td>6</td>
<td>7</td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>2</td>
<td>9</td>
<td>-</td>
<td>5</td>
<td>-</td>
<td>7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>3</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>7</td>
<td>-</td>
<td>9</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 4 | - | - | - | - | - | - | - | -