

Solving NP-Complete Problem Using ACO Algorithm

Muhammad Asif

Department of Computer Science
NUCES
Islamabad, Pakistan
muhammad.asif@nu.edu.pk

Rauf Baig

Department of Computer Science
NUCES
Islamabad, Pakistan
rauf.baig@nu.edu.pk

Abstract—Recent studies have shown that Evolutionary Algorithms have had reasonable success at providing solutions to those problems that fall in NP-Complete class of algorithms. Ant Colony Optimization (ACO) algorithm is one of the promising field of evolutionary algorithms that gave acceptable solutions to Travelling Salesperson Problem and various Network Routing Optimization problems in polynomial time. These classic computer science problems belong to a NP-Complete class of problems that is amongst some of the most interesting in mathematics, including the Sudoku Puzzle Problem. People have tried to automate solving Sudoku Puzzle Problem using brute force, tabu search. Given the success of ACO algorithm with problems within NP-Complete class of problems, it would be interesting to see how it handles this puzzle. A novel technique is presented as modification to standard ACO algorithm. Moreover, we will compare performance matrix (quality of solution and time complexity) of ACO algorithm with other techniques presented in the past to solve the Sudoku puzzle.

Index Terms—Ant Colony Optimization, Ant System, Sudoku Puzzle, NP-Hard Problems, Tabu Search

I. INTRODUCTION

In the early nineties an algorithm called Ant System (AS) was proposed as a novel heuristic approach for the solution of combinatorial optimization problems. An Ant Colony Optimization algorithm (ACO) is essentially a system based on agents which simulate the natural behavior of ants, including mechanisms of cooperation and adaptation. In the use of this kind of system as a new meta-heuristic was proposed [1] in order to solve combinatorial optimization problems. This new meta-heuristic has been shown to be both robust and versatile – in the sense that it has been successfully applied to a range of different combinatorial optimization problems. Each ant, in ACO algorithm, follows a path that is a candidate solution for the given problem. When an ant follows a path, the amount of pheromone deposited on that path is proportional to the quality of the corresponding candidate solution for the target problem. Higher amount of pheromone on a path indicates there is a high probability that this path will lead to a better solution (or close to the optimum solution). When an ant has to choose between two or more paths, the path(s) with a higher amount of pheromone have a greater probability of being chosen by the ant. As a result, the ants eventually converge to a short path, hopefully the optimum or a near-optimum solution for the target problem, as explained before for the case of natural ants.

In essence, the design of an ACO algorithm involves consideration of many factors. An appropriate representation of the target problem is needed in AS that is an important

parameter in implementing ACO algorithm to solve that problem. This representation of problem allows the ants to incrementally construct or modify solutions through the use of a probabilistic transition rule, based on the amount of pheromone in the trail and on a local, problem-dependent heuristic.

ACO algorithm advances the search mechanism, in finding better solution, by applying legal local modification operators to the intermediate solutions. These legal modifications are apprehension of real world situation corresponding to the problem definition. For example in case of Travelling Salesperson Problem (TSP), we construct (find) solutions by making a move to city; choosing from list of available legal cities. Another element that dictates the search process, in finding an optimum solution, is a problem-dependent heuristic function. This function measures the quality of items that can be added to the current partial solution. In TSP, a path that connects to next city with lower cost will have a higher probability of becoming part of optimum solution. ACO algorithm keeps track of paths that have high fitness values. A path has high fitness value if it is close to solution of target problem and is giving solution in polynomial time. Value of pheromone on such paths should be more as compared to other non-promising paths. Here ACO algorithm provides a rule for pheromone updating, which specifies how to modify the pheromone trail. But in it not true that every time an ant selects the same trail with high pheromone value because this thing will restrict the ants in exploring the search space. And hence could not guarantee an optimal solution. To give a chance to ants in searching new paths, which could have apparently lower fitness values, ACO defines a probabilistic transition rule based on the value of the heuristic function and on the contents of the pheromone trail. In addition, this paper proposed a novel technique that tells how pheromone value for each cell on puzzle should be updated, described in ACO Algorithm to Solve Sudoku section.

Artificial ants have several characteristics similar to real ants, namely:

- Artificial ants have a probabilistic preference for paths with a larger amount of pheromone.
- Shorter paths tend to have larger rates of growth in their amount of pheromone.
- The ants use an indirect communication system based on the amount of pheromone deposited on each path.

AS, which was first applied to the traveling salesman problem, was recently extended and/or modified both to improve its performance and to apply it to other optimization

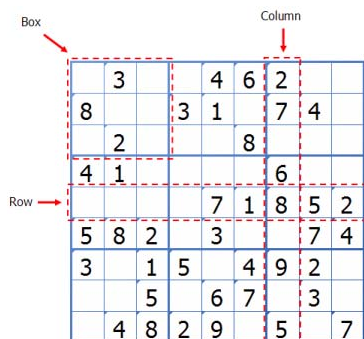
problems. Improved versions of AS include, among others, ACS, MAX-MIN AS, and ASrank. All these algorithms have been applied to the TSP with varying degree of success, but always improving over AS performance. These improved versions, as well as the original AS, have also been applied to a diverse set of optimization problems. Examples are quadratic assignment, vehicle routing, connection oriented and connectionless network routing, sequential ordering, graph coloring, shortest common super sequence, single machine tardiness, multiple knapsack, etc. For many of these problems, the results obtained place the ant system based approaches among the best available heuristics. In this paper we present the Ant Colony Optimization (ACO) meta heuristic, that is the result of an effort to define a common framework for all these versions of AS, to solve Sudoku puzzle. The main motivations for defining the ACO met heuristic are the desire to provide a unitary view of the ongoing research in this growing field, as well as the hope that such a characterization will help to identify the most important aspects of these algorithms and therefore, make the development of new applications easier. And consequently ACO algorithms give us a way of finding better solutions to NP-Complete problems in polynomial time, for which brute force is the only solution.

II. SUDOKU PUZZLE – RULES AND SOLVING TECHNIQUES

To solve the Sudoku, you must fill each of the small blank squares called 'cells' with the correct numbers 1 through 9. To do this, you follow three simple rules:

- Every column must contain all the numbers 1 through 9 in any order.
- Every row must contain all the numbers 1 through 9 in any order.
- Every 3x3 box must contain all the numbers 1 through 9 in any order.

Since each column, row, and box can only contain 9 numbers and must contain all numbers 1 through 9, no number can be repeated within a column, row, or box. These simple rules provide the basis for solving the Sudoku.



	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

Figure 1: Sample Sudoku Puzzle

At the beginning of each game the grid will contain some initial state; some cells are filled with numbers complying Sudoku puzzle. Each puzzle has, normally, a unique solution. For the simple 9x9 version of the game, found alongside the

crossword in every daily paper, there are 6,670,903,752,021,072,936,960 [2] valid configurations for an empty grid and just 1 of these is the solution that you're looking for. Moreover the general case of the problem has been proven to be NP-Complete [3] placing it amongst some of the most interesting problems in computer science.

People have developed different techniques to solve the puzzle manually, i.e., crosshatching, slicing/dicing, penciling. A naive way to start solving the puzzle is by looking at each row or column. Maintain lists of numbers for every row, column, and sub-region on paper that is legal to appear in a row or column or sub-region. After that, numbers are filled in empty cells row-wise or column-wise from available numbers and consequently those numbers are removed from lists.

III. RELATED WORK

There is not much work done in using evolutionary algorithms to automate the process of finding solutions to combinatorial problems such as Sudoku puzzles. The only work qualifying for reference was presented by David Mullaney [4] who used ACO algorithm to solve Sudoku puzzle. He implemented two techniques in attempting to solve puzzles, i.e., tabu search and ACO technique. He tested his techniques on Sudoku puzzles available at [5]. Using tabu search technique, he achieved maximum fitness value of 69, i.e., the technique was able to fill 69 cells of puzzle. For this technique he made hundred iterations of ants. On the other hand using ACO algorithm yielded maximum fitness value of 72. Again, hundred iterations of ants were made. His implementation of ACO algorithm took only 3–7 minutes to complete its execution, but was able to find solutions to only about 20% of the problems that it was tested against. The author did not provide any algorithmic detail of ACO technique that he used in finding solution to Sudoku puzzles. He did not mention how he had mapped Sudoku in ants system. His paper contained only the obtained results and the methodology of solving the problem was missing.

IV. ACO ALGORITHM TO SOLVE SUDOKU

As I mentioned in Introduction section of this paper that ACO algorithms produced reasonably good results in solving a NP-Complete problem, i.e., Travelling Salesman Problem (TSP). This success raises a question, why couldn't other NP-Complete combinatorial problems be solved using ACO algorithm like Sudoku puzzle? The answer to this question lies in the ensuing paragraphs.

A. ACO Representation

In this sub section I described design of ACO algorithm to represent the Sudoku problem. I encapsulated the functionality of an ant in a class called 'Ant'. This class has data members and methods that conform to the ACO algorithm's requirement. Each ant has its own Sudoku board on which it performs algorithmic steps of the ACO algorithm. These functionalities include filling of Sudoku board using pheromone values, find maximum trail on board (row-wise and column-wise) which has unique sequence of numbers, update pheromone[6] value using following formula that

depends upon length of trail found in any row and/or column. In this paper, a novel modification to standard ACO algorithm is proposed described in the following sub section.

$$P(t+1) = \Delta \left(P(t) + \frac{|trail(t)|}{81} \right) \quad (1)$$

P is the level of pheromone, after a given trail update
trail is the length of a given trail
delta is the pheromone decay constant

Then we defined another class Ant System that has instances of Ants class in it, hence representing the complete ant system required by ACO algorithm. We managed separate tabu lists for every row, column, and for every sub-region (3x3 grid) on the Sudoku board. These tabu lists contains all those numbers that are not legal to be appeared in any row, column, or in any sub-region. These tabu lists have greatly reduced the search space for ACO algorithm and made the technique faster by reducing the time complexity of the algorithm. The algorithm then applied the simple rules of Sudoku to ensure that only valid solutions were chosen. Each individual ant operates a tabu search, visiting as many Sudoku board cells as possible, without breaking the rules of the game. Whenever the ant put any number to Sudoku board cell, it also put that number in tabu list so that next time this number was not among legal choices. Once the tabu list contains all of the possible nodes, the ant calculates the amount of pheromone which it should dispense along the trail, and distributes it. The amount of pheromone deposited on each edge in a given path by using formula in Figure 1. Fitness for each solution is given by counting the number of nodes that the ant has traversed, and a complete Sudoku problem will have a fitness of 81.

B. Algorithm

This sub section of the paper presents the technique that is used to solve Sudoku puzzle. I used ACO algorithm together with tabu search in attempting to find solution to Sudoku puzzles. Tabu search accelerates the process of finding solution as it greatly reduces the search space. Each row, column and sub region of puzzle has its own tabu list that contains all the moves that are illegal to perform. In our case, these illegal moves are numbers (1 to 9) that are already present in any particular row or column or sub region in puzzle. As any number put on puzzle cell, was also added to corresponding tabu list so that in future this number will not be among legal choices, hence reducing the search space for ants. I used a novel technique in updating process of pheromone value of the puzzle, i.e., I set a threshold value for pheromone value that any cell of Sudoku puzzle can have. This thing prevents same sequence of numbers to select again and again for any particular row, column or sub region of the puzzle. Pheromone value for the cell is set to zero as reaches to threshold value. Experiments' results showed that this thing reveals significant improvement in goodness of solution as compared to results presented in the past [4]. Following are the implemented algorithmic steps of the proposed ACO:

- Load puzzle in data structure (described in ACO Representation sub section)

- Fill (as maximum as possible) numbers in puzzle cells randomly by set of Ants without breaking rules of game
- Use tabu lists to reduce options of numbers)
- Select those Ants that produced solutions having lesser empty cells and remained on puzzle
- 50 filled cells used as threshold value
- Allow these Ants with better solutions to update Pheromone Table
- Destroy all Ants
- While iterations count less than preset value:-
 - Produce new Ants
 - Ants generate different solutions to puzzle using Pheromone Table as a guidance tool (longer unique sequence of numbers has higher probability to select again)
 - Allow Ants to update selected positions in Pheromone Table
 - Decay selected Pheromone Table positions
 - If Pheromone value for any puzzle cell exceeds from threshold, reset it to zero
 - Evaluate each Ant's Solution
- end while

V. EXPERIMENTS

To test and verify the implemented technique, three experiments were performed. In first experiment 10 ants were run 10 times, starting number of iterations from 10 to 100; results are depicted in Figure 1. Same experiment was performed again with 25 ants; results are showed in Figure 2. Third time the same experiment was performed with 50 ants; results are presented in Figure 3. And then results from all three experiments plotted on graph in Figure 4. Finally, Figure 5 showed the individual fitness values of ants while trying to solve Sudoku puzzles. These trials were performed using 0.15 as the decay constant, which was applied at the end of each pheromone update operation with a 20% probability that pheromone trails will be ignored. Maximum fitness value that is achieved is 76, i.e., proposed ACO algorithm has been able to fill 76 cells of Sudoku puzzle. The time taken for an ant to produce this result was about 1.4 seconds.

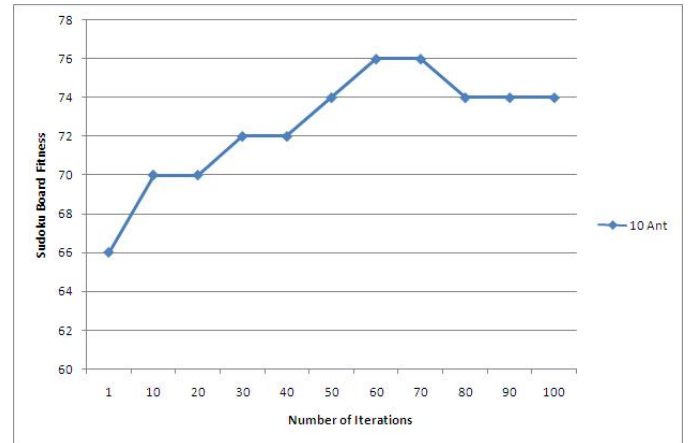


Figure 2: 10 Ants, 10 Runs, 100 Max Iterations

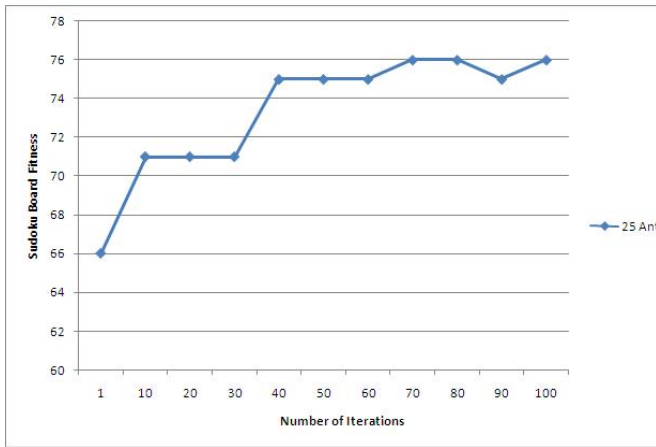


Figure 3: 25 Ants, 10 Runs, 100 Max Iterations

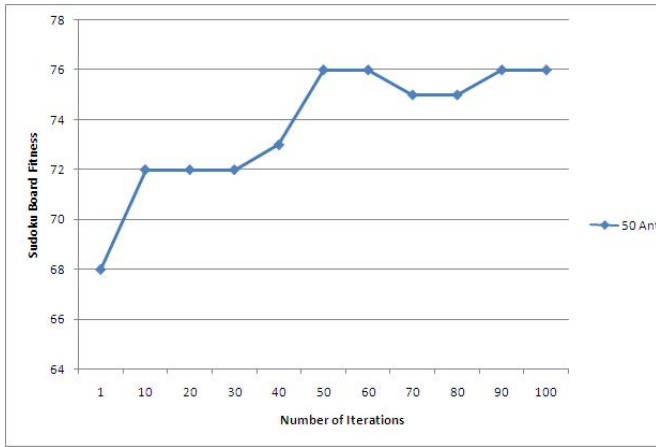


Figure 4: 50 Ants, 10 Runs, 100 Max Iterations

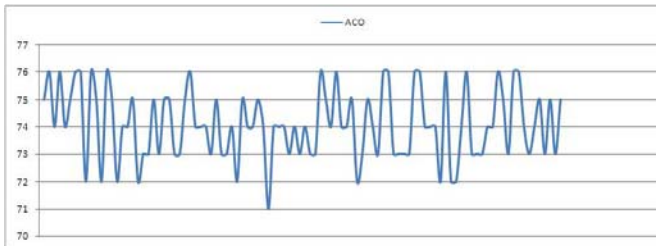


Figure 5: Distribution of Individual Fitness, across a 100 run execution

VI. CONCLUSION

As we can see from the results presented in the above section that they are much better than those presented in the past, although the same ACO algorithm was used. Maximum number of cells that filled in Sudoku puzzle was 72. ACO algorithm, with novel modification in process of updating pheromone values proposed in this paper, gave maximum of 76 filled Sudoku cells which is a significant improvement (while 81 filled cells is solution to Sudoku). This thing supports the novel idea of updating pheromone values in standard ACO algorithm, hence has improved the quality of solution by giving us solution close to optimum in polynomial time. Results have proved that usage of evolutionary algorithms (ACO) in solving NP-Complete problems can give solutions much closer to the optimum. Most important thing is

that this technique gives results in polynomial time which is extremely difficult in such combinatorial problems.

REFERENCES

- [1] M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant Algorithms for Discrete Optimization," *ACM Trans. Artificial Life*, vol. 5, pp. 137–172, April 1999.
- [2] B. Felgenhauer, F. Jarvis. (2005) Enumerating possible Sudoku grids. [Online]. Available: <http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf>
- [3] T. Yato. (2003) Complexity and completeness of finding another solution and its application to puzzles. [Online]. Available: <http://www-imai.is.s.u-tokyo.ac.jp/~yato/data2/MasterThesis.ps>
- [4] D. Mullaney. Using ant systems to solve Sudoku problems. [Online]. Available: <http://ncra.ucd.ie/COMP30290/crc2006/mullaney.pdf>
- [5] G. Stertenbrink. Sudoku webpage. [Online]. Available: <http://magictour.free.fr/top95>
- [6] S. Gilmour, M. Dras. (2005) Understanding the pheromone system within ant colony optimization. [Online]. Available: <http://www.ics.mq.edu.au/~gilmour/publications/gilmour2005b.pdf>