Solving Sudoku Puzzles using Improved Artificial Bee Colony Algorithm

Jaysonne A. Pacurib
jaysonne_pacurib@yahoo.com

Glaiza Mae M. Seno
glaizaseno@gmail.com

John Paul T. Yusiong
jpyusiong@gmail.com

Division of Natural Sciences and Mathematics
University of the Philippines Visayas Tacloban College
Magsaysay Blvd., Tacloban City, Leyte

Abstract

Sudoku puzzles belong to a set of hard problems called NP-Complete problems. A Sudoku puzzle is a logic-based combinatorial puzzle with rules that are relatively simple. Various algorithms have been applied to solve this combinatorial problem. A relatively new algorithm called Artificial Bee Colony algorithm was developed in 2005. The algorithm mimics the way bees forage for food and has been successfully applied to a wide array of NP-Complete problems. This paper explores the possibility of using an improved variant of the Artificial Bee Colony algorithm in solving Sudoku Puzzles. The results obtained support the conclusion that the algorithm can be used to solve Sudoku Puzzles efficiently and effectively.

1. Introduction

Sudoku puzzles belong to a set of hard problems called NP-Complete problems [12]. It is a logic-based combinatorial puzzle designed by Howard Garnes with rules that are relatively simple. For instance, in a $9\,x\,9$ Sudoku puzzle, the following constraints must be satisfied:

1. Each row of cells contains digits $1-9$ exactly once.
2. Each column of cells contains digits $1-9$ exactly once.
3. Each $3\,x\,3$ block contains digits $1-9$ exactly once.

Given that Sudoku is an NP-Complete problem, it is often impractical to use brute-force trial and error to solve this puzzle since each empty cell can constitute up to nine possible answers. For example, an empty Sudoku grid is calculated to have $6,670,903,752,021,072,936,960$ different combinations [8]. But the rules of Sudoku will add constraints to this equation thus decreasing the number of possible combinations. However, the result is still a huge space to search for one optimal solution. Thus, several researches have been done in an attempt to find efficient methods of obtaining the solution of Sudoku puzzles [4], [6], [7], [8], [9], [10]. In this paper, a new approach in solving Sudoku puzzles is presented by exploring the idea of using the Artificial Bee Colony algorithm.

Proposed in 2005 by Dervis Karaboga, the Artificial Bee Colony (ABC) algorithm [5] is a relatively new population-based optimization algorithm which mimics the behavior of bees when foraging. This foraging behavior of bees has been expanded to build partial solutions in many problem domains including combinatorial optimization problems [1], [2], [3], [11].

The rest of the paper is organized as follows. Section 2 describes in detail our proposed Sudoku solver using the ABC algorithm. The implementation, experiments done and results are discussed in Section 3 while in Section 4, we draw our conclusions.

2. A New Approach to Solving Sudoku Puzzles

In this section, a new approach to solving Sudoku puzzles is described which is based on the improved Artificial Bee Colony algorithm presented in [11]. In this proposed approach, the aforementioned goals of the Sudoku puzzle is accomplished by representing Sudoku puzzles as an instance of combinatorial optimization problems, that is, it is characterized by intrinsically ensuring that the third constraint is not violated while leaving the task of satisfying the other two constraints to the algorithm being employed as discussed in [10]. In other words, the optimization prob-

978-0-7695-3873-0/09 $29.00 © 2009 IEEE

lem is to minimize the number of duplicate digits found on each row and column. The general steps of the ABC algorithm used in this paper are presented.

```
Begin
[01]  Initialize Population
[02]  Evaluate Population
[03]  Set cycle=1
[04]  Repeat
[05]    Produce new solution Vi
        in the neighborhood of Xi
[06]    Evaluate Population
[07]    Apply Greedy Selection process
        between Xi and Vi
[08]    Calculate the probability
        values Pi
[09]    Normalize Pi values into [0,1]
[10]    Produce new solutions Vi from
        Xi depending on the value of Pi
[11]    Evaluate Population
[12]    Apply Greedy Selection process
        between Xi and Vi
[13]    If abandoned solution exists
[14]        Replace abandoned solution
            with new solution
[15]    Memorize best solution
[16]    cycle=cycle+1
[17]  Until (termination condition
      is met)
End
```

2.1. The Sudoku Puzzle

The algorithm will optimize a 9 x 9 Sudoku puzzle with varying degree of difficulty. Initially, the puzzle has fixed values for certain cells which are called "start squares" as shown in Figure 1. These "start squares" cannot be manipulated when initializing the population and when searching for an optimal solution.

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

Figure 1. Sudoku with start squares.

On the other hand, all "non-start squares" are initially filled with random values 1-9 but ensuring that each 3 x 3 block contains values 1-9 exactly once. The idea of first filling up the "non-start squares" with random values from 1 to 9 and satisfying the third constraint leads us to focus on optimizing the puzzle by minimizing duplicate digits found on each row and column.

2.2. Description of the ABC Sudoku Solver

Initially, parameter values are defined: maximum number of cycles, c, number of Employed bees, e, number of Onlooker bees, $o\,(o > e)$ and number of Scout bees, s. The first three parameters are user-defined while the fourth is defined by $s = 0.1 * e$.

During the initialization of the population each Employed bee is given a copy of the Sudoku puzzle to be optimized (the food source). The bee will randomly place digits 1-9 on each "non-start square". Afterwards, the population is evaluated then it goes to a cycle until a stopping criterion is met as depicted in Figure 2.

There are two stopping criteria: the maximum number of cycles and the fitness value of 1. If the first is met this means that the algorithm was not able to find the optimal solution to the puzzle thus the best solution obtained so far will be displayed. In contrast, if the second condition is satisfied this means that the optimal solution to the Sudoku puzzle problem has been found.

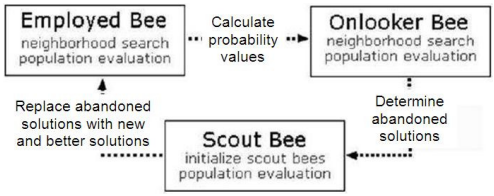

Figure 2. General ABC Flow.

2.3. Neighborhood Search

Given the feasible solution X_i and the randomly chosen neighbor X_k, a random number j is chosen. Once the value of j is known, a duplicate of X_i is then made and labeled as V_i. V_i is similar to X_i except for the value of V_{ij} as illustrated in Figure 3. To find for new value of the cell V_{ij}, the following formula is used:

$$V_{ij} = X_{ij} + rand[0,1] * abs(X_{ij} - X_{kj}) \qquad (1)$$

provided that V_{ij} is not a start square. Also, if the value obtained is greater than 9, the modulo of the value plus one is used as the final value of V_{ij}.

However, if the intrinsic property of the subgrids of the Sudoku puzzles is violated as depicted in Figure 3(right), then a swap operation is performed, that is, the original location of the element V_{ij} is replaced with X_{ij}. Figure 4 shows the final V_i after performing the swap operation.

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

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

Figure 3. V_i (right) is exactly the same as X_i (left) except for the j^{th} element.

The two feasible solutions, X_i and V_i are then evaluated using the evaluation function. The feasible solution with the higher fitness value is kept in the bee's memory.

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

Figure 4. Final V_i after the swap operation.

2.4. Evaluation and Probability Functions

Fitness values determine the quality of the solution space. This is given by:

$$fit_i = \frac{1}{1 + f_i} \tag{2}$$

where fit_i is the fitness of the feasible solution of bee i and f_i represents the penalty value of each feasible solution. Penalty value is the total number of missing values in each row and column. The higher the fitness value, the better the quality of food in that food source. An optimal Sudoku has a penalty value of 0 and a fitness value of 1.

Likewise, the probability function is simply a calculation of a solution's attractiveness to the Onlooker bees. The probability function of a particular feasible solution is defined by:

$$P_i = \frac{fit_i}{\Sigma_{i=1}^{N} fit_i} \tag{3}$$

The value obtained is multiplied by the number of Onlooker bees and the product will determine the number of Onlooker bees that will do the neighborhood search on X_i.

5	7	4	2	**7**	3	9	6	7	2
6	**9**	2	**5**	8	**6**	1	**8**	5	3
3	1	**8**	**4**	9	**1**	**2**	3	4	3
8	**5**	**9**	6	9	3	**8**	**4**	1	2
7	1	6	2	7	4	3	2	**6**	3
4	**2**	**3**	5	8	1	**5**	**7**	9	1
8	2	**5**	**3**	9	**7**	**4**	1	8	1
3	**1**	4	**6**	5	**8**	2	**9**	5	1
9	7	6	2	**1**	4	7	6	3	2
2	4	2	4	4	3	1	1	1	40

Figure 5. The penalty value of the Sudoku.

Now, in determining the abandoned solution, an Employed bee with the worst fitness is paired to a randomly generated scout bee. If the scout bee has a higher fitness than the Employed bee, the Employed bee is replaced.

3. Experiments and Results

The goal is to use the improved Artificial Bee Colony algorithm to solve Sudoku puzzles. Several Sudoku puzzles as shown in Figure 6 were used to determine the effectiveness of the ABC-based Sudoku solver which include the well-known Hard Sudoku puzzle, the Al Escargot used in [7]. In the experiments, the following parameters are defined: $e = 100$, $o = 200$ and $c = 100,000$.

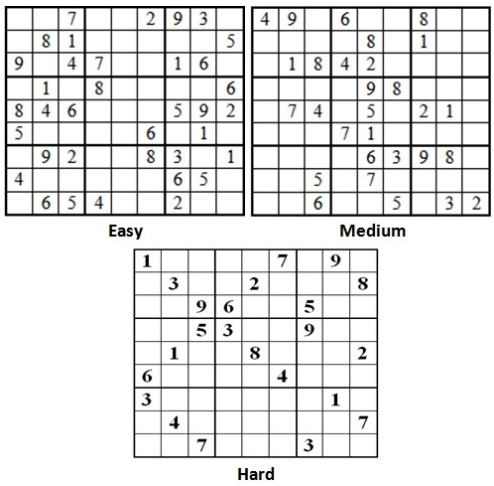

Figure 6. Sudoku puzzles used as Test Data.

For each puzzle, the experiments were repeated thirty (30) times and each experiment uses a different randomly generated initial population. Tables 1-2 show the results obtained from our experiments. The values in the tables are the average values over 30 trials. It is clearly shown in Tables 1-2 that the difficulty level of the Sudoku affects the ability of the ABC-based Sudoku solver to solve puzzles, that is, as the difficulty level increases, the number of cycles needed

Table 1. Average Number of Cycles Needed to Solve the Sudoku Puzzles.

Level of Difficulty	Easy	Medium	Hard
Mean	213.90	663.63	6243.87
Median	213.50	623.00	4249.00
Minimum	58	210	784
Maximum	523	1872	22145
Std. Dev.	98.31	424.77	5184.43

Table 2. Average Time (in seconds) to Solve the Sudoku Puzzles.

Level of Difficulty	Easy	Medium	Hard
Mean	4.10	17.90	267.50
Median	4.00	12.50	176.00
Minimum	1.00	3.00	35.00
Maximum	10.00	73.00	1213.00
Std. Dev.	1.92	18.76	263.64

to find the optimal solution of the puzzle and the time it takes to solve the Sudoku increase as well. Nonetheless, the proposed Sudoku solver is able to find the optimal solution of the puzzle all the time. Furthermore, we compared the results obtained for the Hard Sudoku puzzle with the results presented in [7]. Table 3 shows the comparison between the ABC-based and the GA-based Sudoku solvers. Results indicate that the ABC-based Sudoku solver outperforms the GA-based Sudoku solver.

Table 3. Number of Cycles Needed by ABC and GA in Solving Al Escargot

Algorithm	ABC	GA
Mean	6,243.87	1,238,749.00
Minimum	784	7220
Maximum	22,145	5,901,680

4. Conclusion

This paper introduces a new approach in solving Sudoku puzzles using the improved Artificial Bee Colony algorithm. Experiment results show that the algorithm has always found the optimal solution to all the Sudoku Puzzles tested.

References

[1] Basturk, B. and Karaboga, D., An Artificial Bee Colony (ABC) Algorithm for Numeric Function Optimization. IEEE Swarm Intelligence Symposium 2006, vol. 8, Issue 1, 2006, pp. 687-697.

[2] Baykasoglu, A., Ozbakor, L., and Tapkan, P., Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem, I-Tech Education and Publishing, Vienna, Austria, 2007, pp. 113-144.

[3] Chong, C. S., Low M. Y. H., Sivakumar, A. I., and Gay, K. L., A Bee Colony Optimization Algorithm to Job Shop Scheduling, Proceedings of the 37th Winter Simulation Conference, 2006, pp. 1954-1961.

[4] Crawford, B., Aranda, M., Castro, C., and Monfroy, E., Using Constraint Programming to solve Sudoku Puzzles. ICCIT Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology, Vol. 2, 2008, pp. 926-931.

[5] Karaboga, D., An Idea Based on Honey Bee Swarm for Numerical Optimization, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey, October, 2005.

[6] Lewis, R., Metaheuristics can Solve Sudoku Puzzles, Journal of Heuristics, vol. 13, no. 4, 2007, pp. 387-401.

[7] Mantere, T. and Koljonen, J., Solving and rating Sudoku puzzles with Genetic Algorithms, IEEE Congress on Evolutionary Computation, 2007, pp. 1382-1389.

[8] Moon, T. K., and Gunther, J. H., Multiple Constraint Satisfaction by Belief Propagation: An Example of Using Sudoku, IEEE Mountain Workshop on Adaptive and Learning Systems, 2006, pp. 122-126.

[9] Mullaney, D., Using Ant Systems to Solve Sudoku Problems, Springer Publishing Company, 2006.

[10] Perez, M. and Marwala, T., Stochastic Optimization Approaches for Solving Sudoku, arXiv:0805.0697v1, 2008.

[11] Quan, H., and Shi, X., On the Analysis of Performance of the Improved Artificial-Bee-Colony Algorithm, Proceedings of the 2008 Fourth International Conference on Natural Computation, vol. 7, 2007, pp. 654-658.

[12] Yato, T. and Seta, T., Complexity and Completeness of Finding Another Solution and Its Application to Puzzles, IEICE Trans. Fundamentals, vol. E86-A, no. 5, 2003, pp. 1052-1060.