Solving Word Tile Puzzle using Bee Colony Algorithm

Erum Naz [1], Khaled Al-Dabbas[2], Mahdi Abrishami[3], Lars Mehnen[4], Milan Cvetkovic [5]
University of Applied Science Technikum Wien
Vienna, Austria

***Abstract*—In this paper, an attempt has been made to solve the word tile puzzle with the help of Bee Colony Algorithm, in order to find maximum number of words by moving a tile up, down, right or left. Bee Colony Algorithm is a type of heuristic algorithms and is efficient and better than blind algorithms, in terms of running time and cost of search time. To examine the performance of the implemented algorithm, several experiments were performed with various combinations. The algorithm was evaluated with the help of statistical functions, such as average, maximum and minimum, for hundred and two-hundred iterations. Results show that an increasing number of agents can improve the average number of words found for both number of tested iterations. However, continuous increase in number of steps will not improve the results. Moreover, results of both iterations showed that the overall performance of the algorithm was not much improved by increasing the number of iterations.**

Keywords—slide tile puzzle; artificial bee colony algorithm; swarm intelligence; artificial intelligence; fitness function; loyalty function; word tile puzzle; Bee colony optimization

I. INTRODUCTION

The prime inspiration to design any optimization algorithm is to simulate natural processes. Lots of algorithms have proved their inspiration from natural process such as simulated annealing (SA), genetic algorithms (GA) [1], ant colony optimization (ACO) [2], particle swarm optimization (PSO) and other Swarm Intelligences (SI) [3]. Swarm Intelligence is based on the collective behaviour of individuals in various decentralized systems. These decentralized systems are composed of physical individuals that communicate, cooperate, collaborate, and exchange information and knowledge among themselves to perform some tasks in their environment [4]. A detailed survey to related work is discussed in Section II.

The purpose of this paper is to develop a slide tile game. An attempt has been made to solve the word tile puzzle problem for the most optimum solution by taking an inspiration from natural processes. In the beginning, the board of the tile game is filled with random characters. The tile game is of size n x n, starting from 4 x4 (Figure 1), but in the end larger sizes should be solvable. By moving the tiles (up, down, right, left) a specific situation should be found, where the graph contains a maximum number of words (length 2 to n).

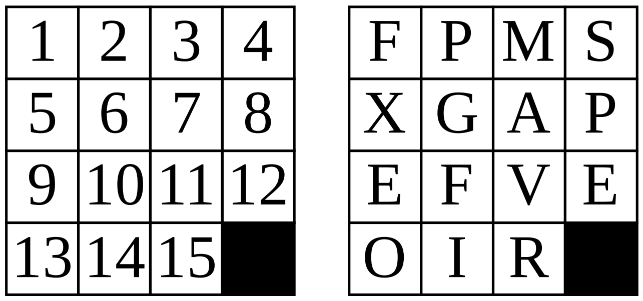

Fig. 1. 15-puzzle slide tile game and 15-word tile puzzle

This paper is organized as follows: Section I gives the brief introduction to the topic, Section II is about related work in the field of string matching, crossword and tile puzzles, Section III elaborates the implemented steps of bee colony algorithm, Section IV describes the statistical results, Section V is discussing the conclusion and finally section VI is future work.

II. LITERATURE REVIEW

Hua [5] implemented blind search and heuristic search algorithms to solve Eight-puzzle problem. In blind search Breadth-first and Depth-first and in heuristic search A* algorithm were implemented to find the optimal solution. The result showed that A* algorithm is more convenient, efficient and better in terms of running speed and cost of search time than blind search algorithms. Genetic algorithm and depth first algorithm was implemented to solve Japanese puzzles. Evaluation and comparisons of performance concluded that depth-first algorithm is faster for small size puzzles and genetic algorithm is better in large size puzzles. However, both methods are slow [6].

One of the recent optimizations algorithms replicating the intelligent natural behavior of honeybee's swarm is artificial bee colony (ABC) [10]. The popularity of ABC increased significantly in last years, the algorithm has been implemented in different field for example in [11] used ABC to solve complex network, while [12] combined Fuzzy logic with ABC in the optimization of parameters for an autonomous mobile robot control.

Author in [7] used the natural behaviours of the real honey bee to develop the artificial bee colony algorithm and resolve numerical optimization problems. In the Artificial Bee Colony algorithm (ABC), there are three types of bees, scout bees, employed bees and onlooker bees. The half of the total bees are initially scout bees. For each scout bee, a new food source position is produced. After producing new food source position, these scout bees become employed bees. Then these employed bees try to improve food sources by interacting with each other. The onlooker bees wait for the food sources positions by the employed bees in the hive. Employed bees share position information about the food sources, each onlooker bee picks up one of the food source positions and tries to improve the food source position.

Like ABC, the Bee Colony Optimization (BCO) algorithm, follows the way how honey bees in real-world nature look for food source, which makes this algorithm more effective, compared to lots of other stochastic random-search algorithms. So far, BCO is implemented in various real-life optimization problems, such as vehicle routing problem [8], the routing and wavelength assignment in all-optical networks, the traffic sensor location problems on highways, the static scheduling of independent tasks on homogeneous multiprocessor systems and disruption management in public transit [9].

III. Methodology

This section elaborates the step by step approach to solve word tile puzzle by using bee colony algorithm [8].

A. Step 1: Initial Input

The Bee Colony Algorithm starts with the initial inputs. These inputs are:

1) Agents: an entity that performs the activity – Artificial bees.

2) Number of Steps per Agents: number of shifts movements by each agents or distance covered during one trip.

3) Number of Iterations: number of trips made by each agents.

The number of agents, represent the number of employed artificial bees, looking for food source. In this case, the agents will look for the best state value of the board, which is the maximum number of words found in rows and columns. The number of steps implies how far the agents can go during one iteration (In Bee Colony Algorithm, the number of steps indicate, how far the bees can travel). The final input value is the number of iterations, which demonstrates the number of trips per each run of the program (The number of trips that each employed bee will perform in an assigned working period). It has to be taken into account that the output value of implemented program will highly depend on our initial inputs.

B. Step 2: Defining Initial State

In this step, the dimensions of the board are defined and initial position of the blank tile (starting point for agents/bees) is marked (tile in the last row and the last column in this case); random letters (alphabets) are assigned to the initial state of the board. The value of the initial state is calculated, by using the fitness function. Fitness Function counts the number of all existing words in the board by matching the input string to the imported dictionary. Word could be two letter long or maximum to the dimension of the board. Fitness function is calculated after each movement of the agent in the board. It is allowed, that more than one word is found in a row or a column and there can be a word inside another word. Word matching is from left to right and top to bottom (diagonal matching is not allowed).

In order to reduce the computational cost and improve performance, two constraints are applied to the dictionary. The first constraint is to eliminate the words longer than the dimension of the board and the second constraint omits the words which were not the part of randomly generated board in the initial state.

C. Step 3: Agents movement and state update

In this step, the movement of agents is assigned randomly and new state per each agent is generated.

Agents can move up, down, right and left inside the board. The last movement of the agents and their directions are stored to avoid a backward movement. The idea behind not letting agents to move backwards is to avoid unnecessary movements which will be not only time consuming, but they will also take some memory computation. Agent cannot move diagonally and beyond the walls of the board.

After defining initial input, states and keeping constraints in consideration, agents are ready to move and start searching for higher value state. After each step of each agent, the value state is calculated (using fitness function). If an agent reaches higher value of Fitness Function (number of words) than value of existing best state/states will be stored by updating archive of best states and previous best state/states will be deleted. If an agent reach equivalent value of Fitness Function as value of existing best solution or solutions its state will be added to other states with the same value in the archive.

D. Step 5: Agents' Loyalty Function

In this step, agent's loyalty is determined by using probability. The idea of considering loyalty comes from the natural processes inspiration. Studying the behaviour of bees for finding nectar, it has been observed that the successful bees in finding food sources will go back to the hive, performing the "Waggle dance", in order to convince other bees to get the same route as them and go for the same food source.

This behaviour in this algorithm, has been defined by loyalty function. After each step of each agent per each iteration, the state value of that specific agent will be calculated and compared to the state values of other agents. Certainly, there will be agents with higher state values, and the loyalty function will make other agents to decide, if they want to keep on their own route or they would rather change their routes, which show that there might be higher possibility of finding better state values.

1) Loyalty of Agent: To compute the loyalty of agents, following probability functions have been used [9].

$$P_n = e^{-\{(O_{max} - O_n)/k\}} \quad (1)$$

P_n: Probability of agent n to stay loyal

O_{max}:	MAX (O_n) =1 , O_{max} is maximal normalized value of fitness functions of all agents
O_n:	$(C_n – C_{min})/(C_{max} - C_{min})$ C_n is normalized value of fitness function for agent n C_{min} is minimum value of all fitness functions C_{max} is maximum value of all fitness functions
k:	number of current step in current iteration

The methodology is based on the roulette game; a number, which is between 0 and 1, should randomly be chosen and if the value of this random number is less than P_n, then agent n is considered as loyal, otherwise, it is counted as disloyal and change its route for the agents with better state values.

Assigning new state to disloyal agents: The next step, will be for disloyal agents to choose a state of all possible loyal agents' states. However, there is the possibility that there might be more than one loyal agent, whose state, is a nominee of being chosen as the higher value state. Or there can be more than one disloyal agent, which can make a choice out of different states of nominate loyal agents. In this algorithm, probability functions in (2) has been implemented that compute the probability of being chosen as the higher state value agent by disloyal agents:

$$P_A = \frac{O_r}{\sum_{r=1}^{R} O_r} \tag{2}$$

P_A	Probability of loyal agents to be chosen by disloyal agent
r:	Loyal agent
R:	Total number of Loyal Agents
O_r :	Normalized Fitness Function value of loyal agent
$\sum_{r=1}^{R} O_r$:	Sum of normalize Fitness Function values of all loyal agents

In this formula, the normalized fitness function value of each loyal agent is divided by the sum of fitness function values of all loyal agents. Finally, to each disloyal agent, a state of a loyal agent would be assigned, based on the outcomes of the probability functions.

E. Step 6: Best Value State (Optimal Solution)

After the last iteration is done, the best value state is printed from archive of the best state or states that is/are archived during the whole search process, and the existing words in the board will be presented.

IV. Results

In order to shape our results, number of experiments were performed. Implemented solution is tested for number of steps, number of agents and number of iteration. To determine the effect of each variable, they were tested separately. To achieve this, variables other than test variables were kept constant and results were calculated, using fitness function, by changing the value of test variables.

A. Evaluation measurements

Basic functions of statistics such as average, maximum and minimum are used to calculate the results.

1) Average: average number of words found per 30 testing experiments for defined number of agents (2, 4, 8, 12, 16, 20, 30) and number of steps (10, 20, 30, 40) with fixed number of iteration.

2) Maximum: the highest value of the state of the board per 30 testing experiments with fixed number of iterations.

3) Minimum: the lowest value of the state of the board per 30 testing experiments with fixed number of iterations.

The main dictionary consisting of 22,353 words was used for testing, however it is reduced to 1,359 words after passing through the dictionary reduction function. 2, 640 experiments were performed including 100 and 200 iterations, 2, 4, 8, 12, 16, 20 and 30 agents and 10, 20, 30 and 40 steps. Detailed results are presented below.

B. Results for 100 iterations

1) Varying number of steps for 100 iterations

To examine the effect of the number of steps, 660 experiments were executed, in which the number of steps were changed while keeping the number of agents and the number of iterations constant. The number of steps started from 10 extending to 20, 30 and 40.

Table I shows the test results for 100 iterations, varying number of steps, starting from 10 and then 20, 30 and 40 for 2, 4, 8, 12 and 16 agents.

TABLE I. Varying Number of Steps for 100 Iterations

Varying number of steps for 100 iterations		**Number of steps**			
		10	*20*	*30*	*40*
2 Agents	Average number of words	27.57	28.5	28.57	28.93
	Maximum number of words	33.00	32.00	32.00	32.00
	Minimum number of words	23.00	24.00	25.00	25.00
4 Agents	Average number of words	27.7	29.93	30.30	30.23
	Maximum number of words	31.00	34.00	36.00	36.00
	Minimum number of words	24.00	26.00	26.00	25.00
8 Agents	Average number of words	29.13	30.97	30.50	30.47
	Maximum number of words	33.00	34.00	35.00	35.00
	Minimum number of words	25.00	28.00	27.00	27.00
12 Agents	Average number of words	28.53	31.17	31.43	31.90
	Maximum number of words	35.00	35.00	38.00	36.00
	Minimum number of words	23.00	27.00	27.00	27.00
16 Agents	Average number of words	29.27	32.13	32.57	32.10
	Maximum number of words	34.00	36.00	36.00	40.00
	Minimum number of words	26.00	28.00	29.00	27.00

There is an observable increase in average, maximum and minimum number of words when increasing number of steps from 10 to 20. However, after first 20 steps, there is no noticeable improvement and minimum number of words found, did not progress.

Fig. 2, shows the graphical overview of results obtained by varying number of steps for 100 iterations.

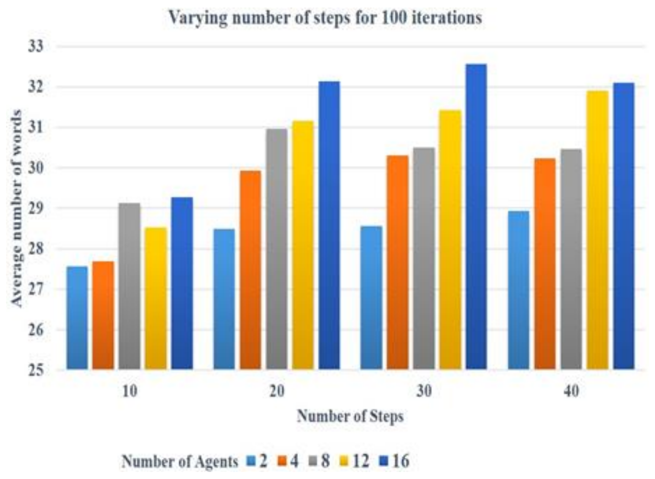

Fig. 2. Varying number of steps for 100 iterations

2) *Varying number of Agents for 100 iterations*

To examine the effect of the number of agents, 660 experiments were executed, in which the number of agents were changed while keeping the number of steps and the number of iterations constant. The number of agents started from 2, extending to 4, 8, 12, and 16.

Table II, shows the test results for 100 iterations, varying number of agents, starting from 2, 4, 8, 12 and 16 for 10, 20, 30 and 40 steps.

It was noticed that increasing the number of agents improves the average number of words. However, the maximum and the minimum number of words were not much effected.

TABLE II. VARYING NUMBER OF AGENT FOR 100 ITERATION

Varying number of Agent for 100 iteration		Number of Agents			
		2	*4*	*8*	*12*
10 Steps	Average number of words	27.57	28.5	28.57	28.93
	Maximum number of words	33.00	32.00	32.00	32.00
	Minimum number of words	23.00	24.00	25.00	25.00
20 Steps	Average number of words	27.7	29.93	30.30	30.23
	Maximum number of words	31.00	34.00	36.00	36.00
	Minimum number of words	24.00	26.00	26.00	25.00
30 Steps	Average number of words	29.13	30.97	30.50	30.47
	Maximum number of words	33.00	34.00	35.00	35.00
	Minimum number of words	25.00	28.00	27.00	27.00
40 Steps	Average number of words	28.53	31.17	31.43	31.90
	Maximum number of words	35.00	35.00	38.00	36.00
	Minimum number of words	23.00	27.00	27.00	27.00

Fig. 3, shows the graphical overview of results obtained by varying number of Agents for 100 iterations.

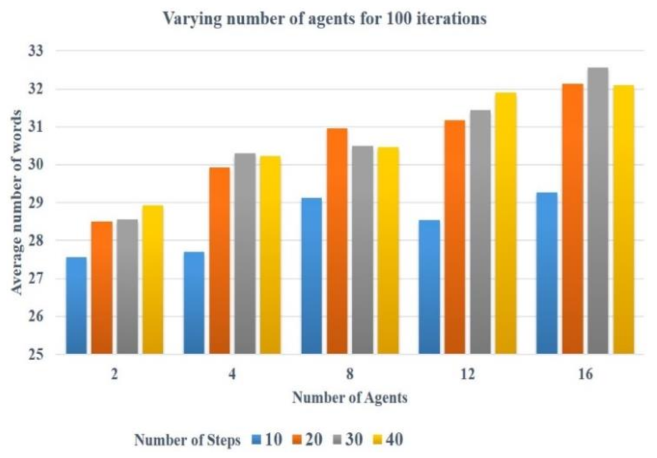

Fig. 3. Varying number of agents for 100 iterations

C. *Results for 200 iterations*

1) *Varying number of steps for 200 iterations*

Table III shows the varying number of steps for 200 iterations by keeping agents constant. The number of steps started from 10, extending to 20, 30 and 40.

TABLE III. VARYING NUMBER OF STEPS FOR 200 ITERATIONS

Varying number of Steps for 200 iterations		Number of steps			
		10	*20*	*30*	*40*
2 Agents	Average number of words	28.30	29.50	29.90	30.40
	Maximum number of words	34.00	35.00	35.00	34.00
	Minimum number of words	23.00	26.00	26.00	27.00
4 Agents	Average number of words	28.50	31.10	30.80	31.60
	Maximum number of words	32.00	35.00	35.00	36.00
	Minimum number of words	25.00	27.00	27.00	28.00
8 Agents	Average number of words	30.20	31.40	32.00	32.20
	Maximum number of words	34.00	35.00	36.00	36.00
	Minimum number of words	26.00	26.00	29.00	29.00
12 Agents	Average number of words	30.40	31.90	32.30	32.60
	Maximum number of words	36.00	36.00	37.00	37.00
	Minimum number of words	25.00	29.00	27.00	29.00
16 Agents	Average number of words	29.80	32.40	33.50	33.20
	Maximum number of words	36.00	36.00	37.00	37.00
	Minimum number of words	25.00	27.00	27.00	30.00

Like 100 iterations, with 200 iterations, an observable increase in average number of words was found, when increasing number of steps from 10 to 20. Afterwards, there is no noticeable improvement for average number of words. However, for maximum and minimum number of words, there is a variation in results as compared to 100 iterations. Minimum number of words improved, when moving from 30 to 40 steps which was not the case in 100 iterations.

Fig. 4, shows the graphical overview of results obtained by varying number of steps for 200 iterations.

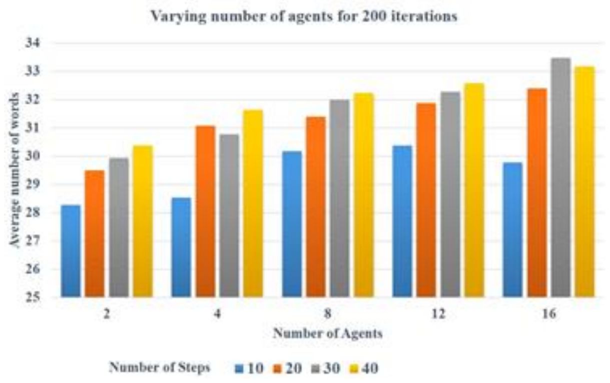

Fig. 4. Varying number of steps for 200 iterations

2) Varying number of Agents for 200 iterations

Table IV is presenting the results for varying number of agents for 200 iterations by keeping steps constant. The number of agents started from 2, extending to 4, 8, 12, and 16.

TABLE IV. VARYING NUMBER OF AGENT FOR 200 ITERATION

Varying number of Agent for 200 iteration		**Number of Agents**			
		2	*4*	*8*	*12*
10 Steps	Average number of words	28.30	28.50	30.20	30.40
	Maximum number of words	34.00	32.00	34.00	36.00
	Minimum number of words	23.00	25.00	26.00	25.00
20 Steps	Average number of words	29.50	31.10	31.40	31.90
	Maximum number of words	35.00	35.00	35.00	36.00
	Minimum number of words	26.00	27.00	26.00	29.00
30 Steps	Average number of words	29.90	30.80	32.00	32.30
	Maximum number of words	35.00	35.00	36.00	37.00
	Minimum number of words	26.00	27.00	29.00	27.00
40 Steps	Average number of words	30.40	31.60	32.20	32.60
	Maximum number of words	34.00	36.00	36.00	37.00
	Minimum number of words	27.00	28.00	29.00	29.00

Here again, like 100 iterations, with 200 iterations, increasing the number of agents, improve the average number of words and increase in maximum number of words. However, the behaviour of minimum number of words is mixed.

Fig. 5, shows the graphical overview of results obtained by varying number of Agents for 200 iterations.

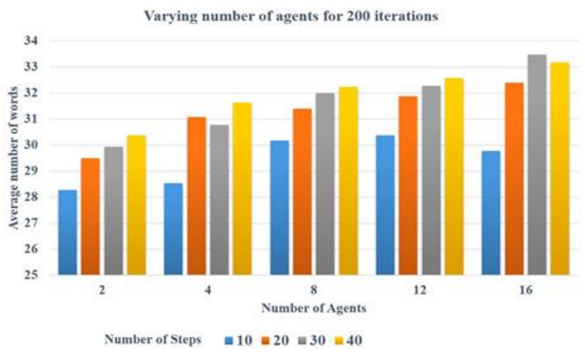

Fig. 5. Varying number of agents for 200 iterations

V. DISCUSSION/ANALYSIS

The performance of Artificial Bee Colony Algorithm is measured by executing larger number of experiments with multiple combinations. A huge variation in results was witnessed due to random behaviour of agents (bees). Average, maximum and minimum number of words found increased, while increasing number of steps from 10 to 20 for both 100 and 200 iterations. However, there is no significant change noticed by increasing steps from 20 to 30 or onwards. Experiments with 20 and 30 agents were also performed, due to high insignificance, those results are not presented in this paper. For 100 iterations, best result is found for 16 agents and 30 steps (average number of words=32.57), following with 20 steps and 16 agents (average number of words=32.13), whereas for 200 iterations best result found was for 16 agents and 30 steps (average number of words=33.50) followed by 16 agents and 40 steps (average number of words=33.20).

By comparing results for 100 and 200 iterations, it is observed that overall performance of the algorithm is not much improved by increasing number of iterations. However, the behaviour of algorithm for 100 and 200 iterations is almost the same for average number of words, whereas for maximum and minimum number of words, it is diverse.

Fig. 6, shows the overall improvements of best result for 100 iterations. 100% of improvements are achieved in 96 iterations. The graph shows that for first 16 iterations, there is almost 70% improvement in results and the remaining 30% improvement covers 80 iterations.

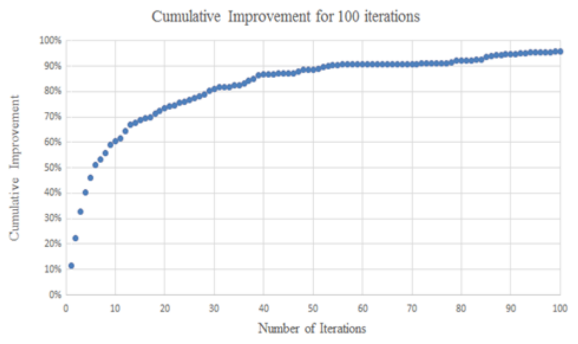

Fig. 6. Cumulative improvement for best result of 100 iterations

Fig.7, shows the overall improvements of best result for 200 iterations. 100% of improvements are achieved in 194 iterations. The graph shows that for first 16 iterations there is almost 61% improvement in results and the remaining 39% improvement covers 178 iterations.

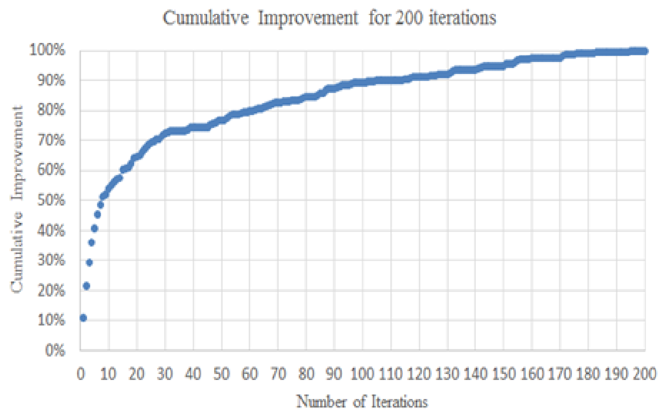

Fig. 7. Cumulative improvement for best result of 200 iterations

Furthermore, in this experiment the time needed to achieve the solution was not taken into consideration as nowadays modern computer have different computing power, and the goal was to achieve the best result regardless of time. Although, the computation time for best results (only) were calculated and it is analysed that time to compute results for 200 iterations is much higher as compared to 100 iterations. Moreover, the improvement in overall results for 200 iterations are not that high. Table V, shows the exact time for best results.

TABLE V. TIME CONSUMPTION

Time Consumption		**Number of steps**		
		20	*30*	*40*
16 Agents	100 Iteration	1:43	2:24	3:04
	200 Iterations	3:56	5:01	6:29

Fig. 8, shows the bar chart for time consumption for best results.

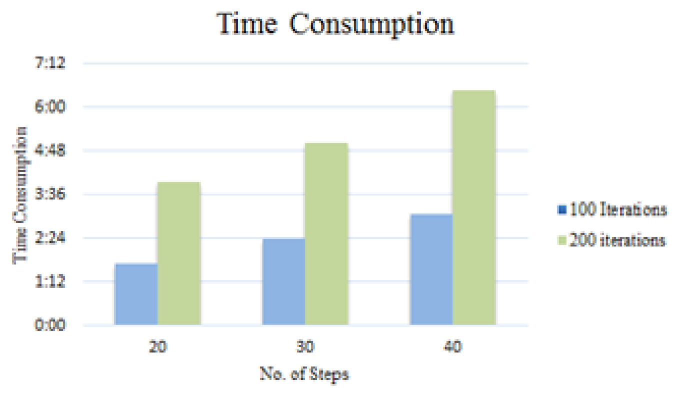

Fig. 8. Cumulative improvement for best result of 200 iterations

VI. CONCLUSION

A wide range of algorithms are inspired by natural processes proved to be successful in solving complicated optimization problems. Bee colony is considered as a class of swarm intelligence technique, where the corporation between different gents increase the efficiency and increase the probability of achieving better results, which cannot be achieved by individual agents. In this paper, word tile puzzle has been analysed using one of the heuristic algorithm named as Bee Colony Algorithm because of the way that artificial bees can communicate with each other and exchange information, making this method, not only fast, but also statistically optimal. Results showed that best solution could be achieved by increasing number of agents, nevertheless results are not improving with increasing number of iterations and steps continually. Furthermore, huge amount of time is required to run high number of iterations and results which has very nominal effect on results.

VII. FUTURE WORK

In future, other heuristic and blind algorithms can be implemented together for the word tile puzzle to compare the efficiency of each algorithm.

ACKNOWLEDGMENT

We would like to express our deep sense of gratitude to our mentor Dr. Lars Mehnen for guiding and encouraging us and one of our colleague Mr. Jakub Stok for his support and help.

REFERENCES

[1] J. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[2] W. Gao, "Study on Immunized Ant Colony Optimization," in Natural Computation, 2007. ICNC 2007. Third International Conference, Haikou, 24-27 Aug. 2007.

[3] D. Teodorović, "BEE COLONY OPTIMIZATION: RECENT DEVELOPMENTS AND APPLICATIONS," "Mircea cel Batran" Naval Academy Scientific Bulletin, vol. XVIII, no. 2, pp. 225-235, 2015.

[4] D. Teodorović, "Bee Colony Optimization (BCO)," in Innovations in Swarm Intelligence, Springer Berlin Heidelberg, 2009, pp. pp 39-60.

[5] R. Shi, "Searching Algorithms Implementation and Comparison of Eight-puzzle Problem," in International Conference on Computer Science and Network Technology , Harbin, 24-26 Dec. 2011.

[6] W. Wiggers, "A Comparison of a Genetic Algorithm and a Depth First Search Algorithm Applied to Japanese Nonograms," in Twente Student Confer D. Karaboga, "D. Karaboga, An Idea Based on Honey Bee Swarm for Numerical Optimization," Technical Report – TR06, Turkey, 2005.ence on IT, Jun. 2004.

[7] Š. Milica and T. Dušan, Computational Intelligence in traffic, University of Belgrade - Faculty of Transport, 2012.

[8] M. Nikolić and D. Teodorović, "Empirical study of the Bee Colony Optimization (BCO) algorithm," Expert Systems with Applications, vol. 40, no. 11, p. 4609–4620, September 2013.

[9] M. Nikolić and D. Teodorović, "Empirical study of the Bee Colony Optimization (BCO) algorithm," Expert Systems with Applications, vol. 40, no. 11, p. 4609–4620, September 2013.

[10] B. B. D. Karaboga, "On the performance of artificial bee colony (ABC) algorithm," Applied Soft Computing, vol. 8, no. 1, pp. 687-697, 2008.

[11] D. D. Magdalena Metlicka, "Complex Network based Adaptive Artificial Bee Colony algorithm," in IEEE Congress on Evolutionary Computation (CEC), 2016.

[12] Leticia Amador-Angulo, "A Generalized Type-2 Fuzzy Logic System for the dynamic adaptation the parameters in a Bee Colony Optimization algorithm applied in an autonomous mobile robot control," in IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2016.