# Solving Shortest Common Supersequence Problem Using Artificial Bee Colony Algorithm

Mustafa Monaf Noaman
Yarmouk University
**3** PUBLICATIONS **12** CITATIONS
SEE PROFILE

Ameera Jaradat
Yarmouk University
**30** PUBLICATIONS **159** CITATIONS
SEE PROFILE

# Solving Shortest Common Supersequence Problem Using Artificial Bee Colony Algorithm

Mustafa M. Noaman
Computer Science Department, IT & CS Faculty,
Yarmouk University
Irbid, Jordan
mustafa.monaf@gmail.com

Ameera S. Jaradat
Computer Science Department, IT & CS Faculty,
Yarmouk University
Irbid, Jordan
ameera@yu.edu.jo

## ABSTRACT

The idea behind this work is to solve the Shortest Common Supersequence by using Artificial Bee Colony (ABC) Algorithm. This algorithm is considered as one of the newest nature-inspired swarm-based optimization algorithms and has a promising performance [6]. Shortest Common Supersequence is a classical problem in the field of strings and it is classified as NP-Hard problem [12]. Many algorithms were used to solve this problem, Such as Genetic algorithms [14], Majority Merge algorithm and Ant Colony algorithm [17].

The project uses Artificial Bee Colony Algorithm to provide a scalable solution to the Shortest Common Supersequence problem. The algorithm evaluation showed promising results.

## Categories and Subject Descriptors

I.2.8 Problem Solving, Control Methods, and Search (Heuristic methods)

## General Terms

Algorithms, Performance

## Keywords

Nature Inspired Algorithms, Artificial Bee Colony, Shortest Common Supersequence, Heuristic

## 1. INTRODUCTION

The Shortest Common Supersequence problem (SCS or SCSP) is a classical NP-Hard problem and has applications in DNA analysis [16], artificial intelligence (specifically planning) [18], mechanical engineering [14] and data compression [22]. Given a set of strings L, the SCS problem consists of finding a string of minimal length that is a supersequence of each string in L. The string B is a supersequence of a string A if A can be obtained from B by deleting in B zero or more characters.

For example: Given a set of strings L = {bbbaaa, bbaaab, cbaab, cbaaa} over an alphabet ∑= {a, b, c}. The shortest supersequence for L is cbbbaaab (8 characters) described in Figure 1.

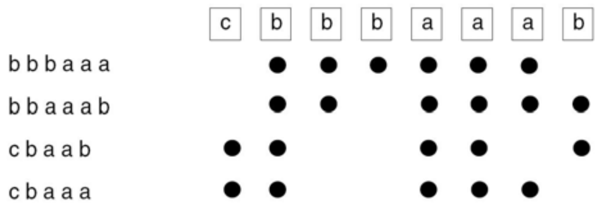

**Figure 1 shows the supersequence cbbbaaab (1st row) contain the characters in the strings that are covered by the supersequence (bullets)**

Many approaches have been proposed for this problem [1, 5, 7], these approaches still need to improve results (ex. large number of generated strings). Finding an efficient solution to the SCS problem is very hard since there is more than one sequence and there sizes may be huge. Also, it is not easy to find in terms of processing time & cost if the number of ∑ (alphabet) is not specified or contain all 26 English language alphabets.

In this paper, we provide a solution to SCS problem using Artificial Bee Colony (ABC) algorithm and evaluate the results with results acquired from other approaches.

The paper is organized as follows: section 2 is background, which provides an under-standing of ABC algorithm and SCSP. Section 3 is devoted to the approach method followed by section 4 which presents the results of applying our approach and the conclusion.

## 2. LITERATURE REVIEW

In this section we will describe Artificial Bee Colony algorithm and the other approaches that were proposed to solve the Shortest Common Supersequence problem.

### 2.1 Artificial Bee Colony (ABC) Algorithm

ABC is an optimization algorithm based on the intelligent foraging behavior of honey bee swarm, proposed by Dervis Karaboga in 2005. ABC algorithm is motivated by the intelligent behavior of honey bees, and uses only common control parameters such as colony size and maximum cycle number. ABC is an optimization tool which provides a population-based search procedure in which individuals called foods positions are modified by the artificial bees with time and the bee's aim is to discover the places of food sources with high nectar amount and finally chose source with the highest nectar amount among the other resources [4], see the outline of the ABC algorithm in Table 1.

**Table 1 Artificial Bee Colony algorithm Steps**

```
1: Initialize the population of solutions $x_i$, $i = 1, . . . , S_N$
2: Evaluate the population
3: $cycle = 1$
4: repeat
5:     Produce new solutions $v_i$ for the employed bees and evaluate them
6:     Apply the greedy selection process for the employed bees
7:     Calculate the probability values $p_i$ for the solutions $x_i$
8:     Produce the new solutions $v_i$ for the onlookers from the solutions $x_i$ selected depending on $p_i$ and evaluate them
9:     Apply the greedy selection process for the onlooker bees
10:    Determine the abandoned solution for the scout bee, if exists, and replace it with a new randomly produced solution $x_i$
11:    Memorize the best solution achieved so far
12:    $cycle = cycle + 1$
13: until $cycle = MCN$
```

## 2.2 Shortest Common Supersequence Problem Solvers

Shortest Common Supersequence problem (SCS) is a classical NP-Hard problem and has applications in many areas. Pervious researches proposed different algorithms to solve Shortest Common Supersequence problem. The Alphabet algorithm used to solve SCS and has an approximation ratio of $q = |\Sigma|$ with time complexity ($qn$), in practice the algorithm doesn't perform when the alphabet is not fixed [3]. Another solution by using Majority-Merge algorithm (MM) and it does not have any worst-case approximation ratio and performs very well in practice with time complexity O($qkn$) [13]. Greedy and Tournament algorithms used to solve SCS and have $O(k^2n^2)$ time complexity and $O(kn+n^2)$ space complexity [10]. Genetic Algorithm (GA), On the other hand, was very hard to apply for many reasons for example: Changing a good solution slightly will yield an invalid string and most possible strings with reasonable length are invalid. Three versions of GA were proposed to solve SCS: $G_0$, $G_{1/||L||}$, $G_1$, after testing, it turned out there is no basic value which performs best for all test instances, which led to propose another version of GA called $G_v$ . The later allows the basic value to vary between 0 and [14]. Ant Colony Algorithm (ACO) algorithm developed by Michel & Middendorf for the SCS problem they called it (AS-SCSP) showed promising results when compared to the results of other approaches [21].

## 3. PROPOSED APPROACH

This section briefly describes the Artificial Bee Colony algorithm and its modified adaptation for solving the SCS.

### 3.1 ABC-SCS Approach

SCS has a constraint:

- String **S** is called a supersequence of a string **T** if **S** can be obtained from **T** by inserting zero or more symbols.

To hold the SCS constraint we control the generated SCS by:

1. Calculating the number alphabet and their frequency for all of the strings ($L$) to direct the random generation toward the used alphabets & most frequent alphabet.
2. Checking the generated SCS compatibility with each string and give value for it.

We used the following parameters:

1- $L$ (entered strings): the string set, entered by user.
2- $\sum$ (alphabet used): to specify the used alphabets in $L$.
3- $f_i$ (fitness value): numerical representation to measure how much the candidate solution fits to be SCS.
4- *Average* (average fitness): the average of fitness values for all candidate solutions in the current cycle.
5- $x_i$ (number of bees): numerical value entered by user to specify the number of working bees used to find SCS.
6- *MCN* (maximum cycle number): numerical value entered by user to determine the number of cycles that the bees generate candidate solutions.

Our approach first, calculates the frequency of each alphabet used in $L$, and then store frequency for each character occurred in $L$ in Array 1. After that we convert the frequencies to characters in Array 2. Example: L = {APLY, DUHA, LADP}, $\sum$ = {A, D, H, L, P, U, Y}, we store frequency for each character occurred in L in Array 1 (denote the alphabet 26 characters) then we convert the frequencies to character in Array 2 (illustrated in Figure 2). These two steps are to ensure the best way of random string generation.

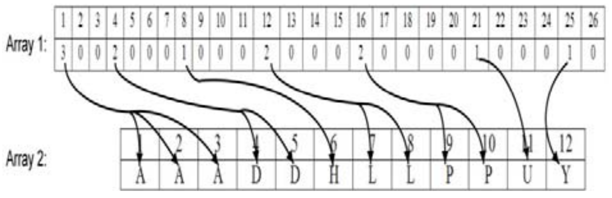

**Figure 2 generating string from alphabet used**

Each worker bee $x_i$ generate candidate SCS by randomly choosing characters from Array 2 in random length and we can see that the most frequent alphabet will be choosing more than less frequent alphabet. Then the onlooker bee calculate fitness value $f_i$ for each candidate SCS by checking each candidate SCS with $L$ one string at a time by checking how much alphabet with the same ordering in $L$ that the candidate SCS have, the more $f_i$ the more the candidate SCS is fitted to be SCS for $L$. We had modified the Merge Algorithm to calculate $f_i$ for the candidate SCS the approach assigns fitness value for each candidate SCS using equation (1) described below:

$$\text{Fitness}\,(S_i) = \frac{\Sigma\,\text{Merge}(S_i, T_1 \ldots T_n)}{\Sigma\,\text{Fitness}\,(S_1 \ldots S_m)} \qquad (1)$$

- $n$: number of entered strings
- $m$: number of generated strings
- $S$: generated string
- $T$: entered string

After assigning final fitness value $f_i$ *for* each worker bee, the onlooker bee needs to calculate *Average* to determine whether the generated string's fitness is good or not. We added a new measure to distinguish good fitness value; this measure helps regenerate new string in the next cycle in case that the fitness is not good in the current one. The measure is simply calculated by finding the average of the finesses of the generated strings over the number of the entered strings, see equation (2).

$$\text{Average} = \frac{\Sigma\,\text{Fitness}\,(S_1 \ldots S_n)}{n} \qquad (2)$$

**Example:** Figure 3 shows that there are 5 worker bees have different fitness values resulted by comparing the generated SCS with entered string "ABCABAA" using heuristic (1) mentioned before, when we calculate the Average if worker bees fitness values the result is 5.6, so we check the fitness values for worker bees and we found that worker bees 1 & 5 have fitness values below average. So in the next cycle the worker bees number 1 & 5 regenerate SCS and compute fitness values.

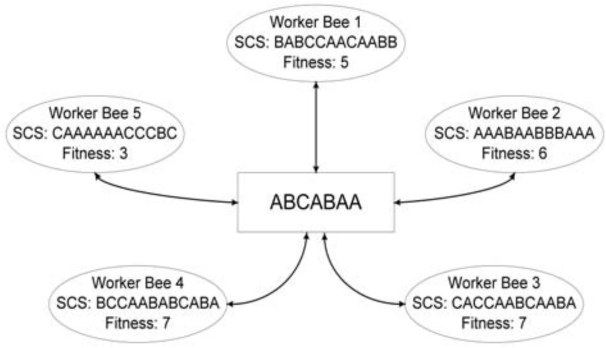

**Figure 3 the results of the 1st cycle of the execution of the program**

Figure 4 shows worker bees in the 2nd cycle of the program execution (illustrated in Figure 6).

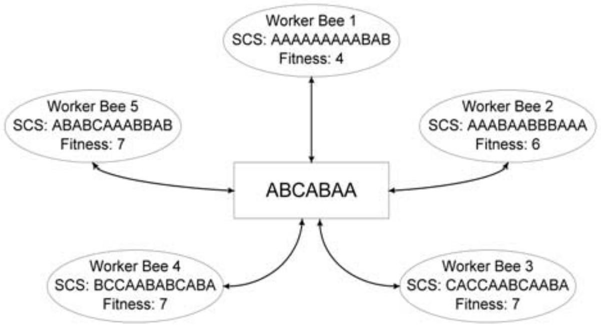

**Figure 4 the results of the 2nd cycle of the execution of the program**

In Table 2 we present the Artificial Bee Colony Shortest Common Supersequence problem Solver (ABC-SCS) pseudocode.

**Table 2 Artificial Bee Colony Shortest Common Supersequence problem Solver (ABC-SCS)**

```
1: Read Entered Strings
2: Calculate alphabet number & frequencies
3: Random generation new solution for each worker bee
4: for ($cycle = 0$, $cycle < MCN$, $cycle$++)
5:         for-each $ES$ (Entered String)
6:         Calculate the $f_i$ for the solutions produced by
           worker bees
7:         Calculate $Average$
8:          if $f_i < Average$
9:                  then
10:                 Generate new string and calculate $f_i$
11:Store the solution with the highest $f_i$ and shortest length
```

## 4. EXPERIMENTS AND EVALUATION

This section is devoted for testing and evaluating the proposed algorithm results. Then, the results will be compared to other results from applying different approaches (Genetic Algorithm & Majority Merge Algorithm). Finally, we will present the conclusion.

### 4.1 The Relation between ABC-SCS Parameters

We considered two important parameters. The first one is the Number of Bees (# of Bees), and the second one is Maximum Number of Cycles (MCN). We tested ABC SCS problem solver against these parameters to find out the effect of these parameters on the resulted Shortest Common Supersequence (SCS) length.

We used 5 strings {bbbaaa, bbaaab, cbaab, cbaaa, baaab} to calculate the SCS length & runtime; the optimal SCS length is 8. The strings we used are illustrated in Figure 1.

Figure 5 illustrates the effect of increasing the number of bees (20 runs for each value). The SCS length decreases progressively as we increase the number of bees.

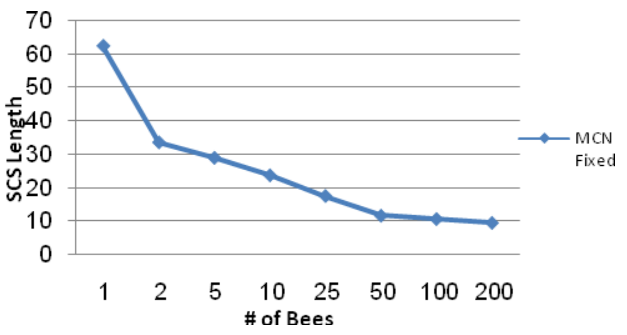

**Figure 5 SCS length change when increasing # of Bees & fixing MCN**

Figure 6 illustrates the effect on increasing the maximum number of cycles MCN (20 runs for each value). The SCS length decreases slightly as MCN increases.

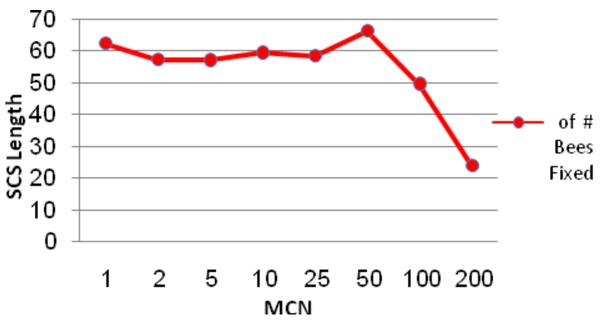

**Figure 6 SCS length change when increasing MCN & fixing # of Bees**

Figure 7 illustrates the effect of increasing the number of bees and MCN (20 runs for each value). The SCS length decreases extensively as we increase both number of bees and MCN.

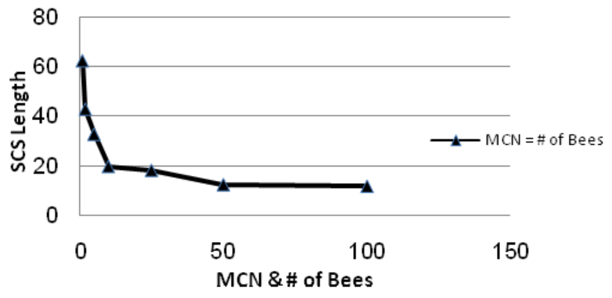

**Figure 7 SCS length change when increasing MCN & number of Bees**

In terms of Runtime, Figure 8 shows the resulted runtime as the number of bees increased (20 runs for each value). The resulted Runtime increases in nearly double each time number of Bees increases.

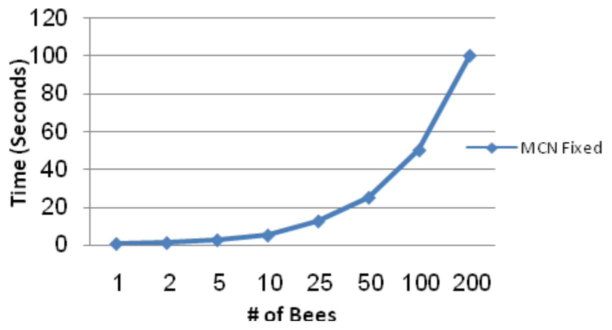

**Figure 8 Runtime values when increasing # of Bees & fixing MCN**

Figure 9 shows the resulted runtime as MCN increased (20 runs for each value). The results show the average and best Runtime have constant value for most of MCN.

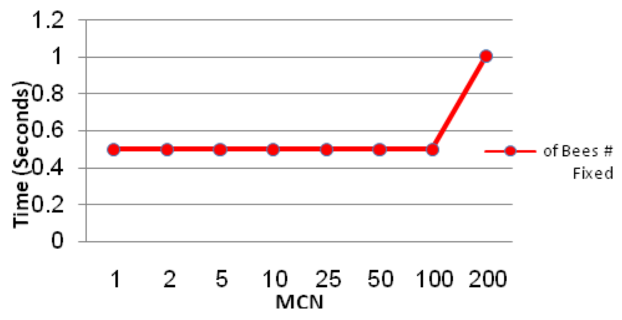

**Figure 9 Runtime change when increasing MCN and fixing number of bees**

Finally we increase both number of Bees and MCN values in the same amount and the results (illustrated in Table 4.13) Figure 10 shows the average and best Runtime (20 runs for each value) increases in large amount in each time both number of Bees & MCN increases.

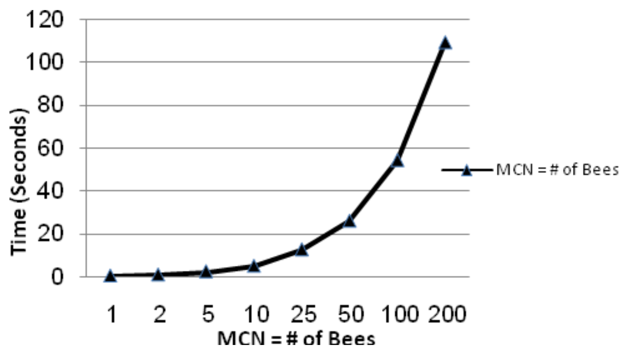

**Figure 10 Runtime values when increasing # of Bees & fixing MCN**

It appears that runtime increases whenever number of Bees increase. On the other hand, as MCN value increases the runtime increases in small amounts.

## 4.2 ABC-SCS Performance

We compared the results of ABC-SCS and the Optimal Solution to check the performance of ABC-SCS. Table 3 shows the results of ABC-SCS implementation with respect of L & ∑.

**Table 3 comparing the results of ABC-SCS with optimal solutions**

| L | ∑ | SCS Length | Optimal Solution |
|---|---|---|---|
| 1 | 1 | 4 | 4 |
| 2 | 2 | 8.5 | 8 |
| 5 | 5 | 13.25 | 12 |
| 10 | 10 | 33 | 26 |
| 20 | 20 | 54 | 46 |

Figure 11 shows the comparison of the results of the ABC-SCS with the optimal solutions.

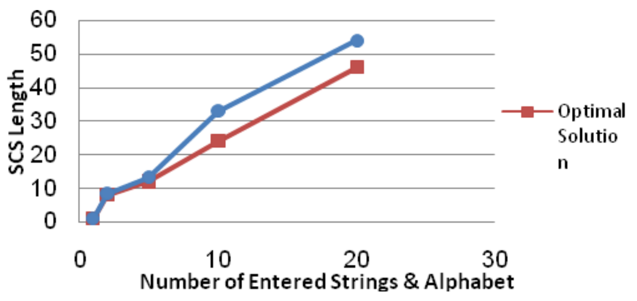

**Figure 11 the difference between ABC-SCS results with optimal solutions**

We understand from Figure 11 that ABC-SCS have excellent results when the number of strings and alphabet are between 1-5 and when they are more than 5 the performance decreases and the algorithm produce good results but not optimal even the results are slightly different but still away from the optimal solution. In practice the ABC-SCS performed well and as in the case of the other proposed algorithms, when the number of alphabet and strings get large the SCS length increases. However, ABC-SCS showed promising results when compared to other algorithms in terms of SCS length & quality.

## 4.3 Results on Special Sets of Strings

The Artificial Bee Colony Shortest Common Supersequence problem solver was tested on different string sets that were obtained from [14]. We compared our results with the results of 5 kinds of solution algorithms used (4 types of Genetic Algorithm: $G_0$, $G_{1/||L||}$, $G_1$ & $G_v$ and Majority Merge Algorithm). We tested Artificial Bee Colony Shortest Common Supersequence (ABC-SCS) problem solver on several instances for which the shortest common supersequence is known. Our test sets were the following:

- $L_1$: 9 strings $a^{40}$, 4 strings $ba^{39}$, 2 strings $bba^{38}$ and 1 string $bbba^{37}$. The optimal solution of length 43 is to first remove the *b*'s, and then the *a*'s of all strings together.
- $L_2$: 9 strings $a^{40}$, 4 strings $b^{13}a^{27}$, 2 strings $b^{26}a^{14}$ and 1 string $b^{39}a$. Similar to $L_1$, it is optimal to first remove all *b*'s and then all *a*'s.
- $L_3$: 8 strings $a^{20}b^{20}$, 8 strings $b^{20}c^{20}$. Here it is optimal to first remove all *a*'s, then the *b*'s and finally the *c*'s.

We tested algorithm with respect to the following parameters:

- Number of Bees = 100
- Maximum Cycle Number = 100

The results for ABC-SCS problem solver for each set illustrated below:

1- The resulted SCS lengths for ABC-SCS implemented on Set $L_1$ were: {46, 52, 47, 43, 45, 46, 47, 50, 48}. So we took the shortest SCS {43} and compare it with results acquired from [14] (Figure 12).

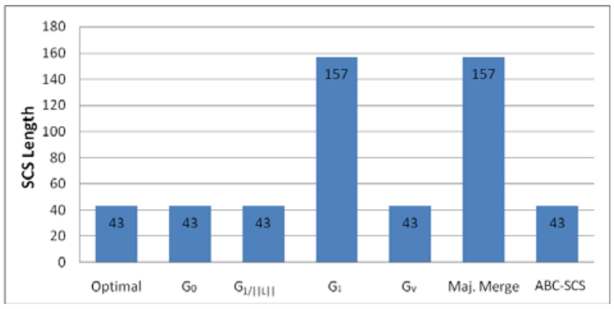

**Figure 12 the resulted SCS lengths of implementing algorithms on set $L_1$**

ABC-SCS produced SCS with length same as optimal SCS length and better than $G_1$ & Majority Merge results.

2- The resulted SCS lengths for ABC-SCS implemented on Set $L_2$ were: {79, 80, 83, 81, 82, 80, 88, 79, 84}. So we took the shortest SCS {79} and compare it with results acquired from [14] (Figure 13).

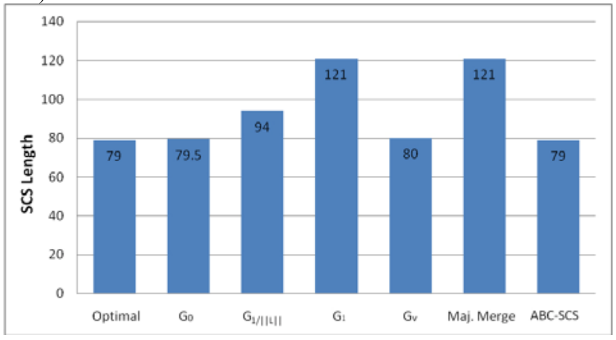

**Figure 13 the resulted SCS lengths of implementing algorithms on set $L_2$**

ABC-SCS produced SCS with length same as optimal SCS length and better than $G_0$, $G_{1/\|L\|}$, $G_1$, $G_v$ & Majority Merge results

3- The resulted SCS lengths for ABC-SCS implemented on Set $L_3$ were: {62, 64, 62, 63, 64, 65, 63, 65, 66}. So we took the shortest SCS {62} and compare it with results acquired from [14] (Figure 14).

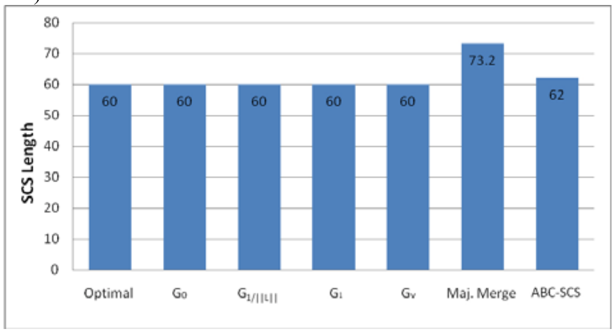

**Figure 14 the resulted SCS lengths of implementing algorithms on set $L_3$**

ABC-SCS produced SCS with length lager than the optimal SCS length by 2 characters and better than Majority Merge result but worse than $G_0$, $G_{1/\|L\|}$, $G_1$, $G_v$ results.

The Final results of implementing the algorithms on special sets of strings mentioned before illustrated in Table 4.

**Table 4 the final resulted SCS lengths of implementing algorithms on special sets of strings ($L_1$, $L_2$, $L_3$)**

| | Optimal Solution | $G_0$ | $G_{1/\|L\|}$ | $G_1$ | $G_v$ | Maj. Merge | ABC |
|---|---|---|---|---|---|---|---|
| $L_1$ | 43 | 43 | 43 | 157 | 43 | 157 | 43 |
| $L_2$ | 79 | 79.5 | 94 | 121 | 80 | 121 | 79 |
| $L_3$ | 60 | 60 | 60 | 60 | 60 | 73.2 | 62 |

## 5. CONCLUSION

In this paper we proposed the Artificial Bee Colony (ABC) Algorithm as a solver for the Shortest Common Supersequence problem. We compared the results obtained by applying Artificial Bee Colony (ABC) Algorithm with the results obtained from applying other approaches that were proposed for solving the SCSP. The Artificial Bee Colony (ABC) Algorithm provides a scalable solution and promising results.

## 6. REFERENCES

[1] Adil, B., Lale, Ö., and Pınar, T. 2007. *Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem*. I-Tech Education and Publishing, ISBN 978-3-902613-09-7, (Dec, 2007).

[2] Andreas, W. 2003. The *Shortest Common Supersequence Problem*. DOI= http://www.update.uu.se/~shikaree/Westling/

[3] Barone, P., Bonizzoni P., Vedova, G.D., and Mauri, G. 2001. *An approximation algorithm for the shortest common supersequence problem: an experimental analysis*. Symposium on Applied Computing. Proceedings of the 2001 ACM symposium on applied computing, 56-60.

[4] Dervis, K. 2010. *Artificial bee colony algorithm*. Scholarpedia. 5(3):6915. DOI= http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm

[5] Dervis, K., and Bahriye, A. 2009. *A comparative study of Artificial Bee Colony algorithm*. Applied Mathematics and Computation, 214, 108–132.

[6] Dervis, K. 2005. *An Idea Based on Honey Bee swarm for Numerical Optimization*. Technical Report-TR06, (Oct, 2005). DOI= http://mf.erciyes.edu.tr/abc/pub/tr06_2005.pdf

[7] Douglas, C.F. 1993. *String Merging: Finding a Shortest Common Supersequence*, (Dec, 1993). DOI= http://www.pfr.com/dfrick/misc/SCS_Paper.pdf

[8] Hubbell, E.A., Morris, M.S., and Winkler, J.L. 1996. *Computer-aided engineering system for design of sequence arrays and lithographic masks*. US Patent, no-5571639.

[9] Haluk, G., M.Cengiz, T., and Ilhan, K. 2010. *Application of Artificial Bees Colony algorithm in an Automatic Voltage Regulator (AVR) System*. International Journal on TPE (Technical and Physical Problems of Engineering) IJTPE, Vol.1, No.3, (Sep, 2010).

[10] Irving, R.W., and Fraser, C. 1993. *On the Worst-Case Behavior of Some Approximation Algorithms for the Shortest Common Supersequence of k Strings*. Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching, 63-73.

[11] Jaysonne, A.P., and Glaiza, M.S. 2009. *Solving Sudoku Puzzles using Improved Artificial Bee Colony Algorithm*. Fourth International Conference on Innovative Computing, Information and Control.

[12] Jeff, E. 1999. *Lecture 21: NP-Hard Problems*. University of Illinois at Urbana-Champaign. DOI= http://theory.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/21-nphard.pdf

[13] Jiang, T., and Li, M. 1995. *On the approximation of shortest common supersequences and longest common subsequences*. SIAM Journal of Computing, 24(5):1122-1139.

[14] Jürgen, B., and Martin, M. 1996. *Searching for Shortest Common Supersequences by Means of a Heuristic-Based Genetic Algorithm. 2nd Nordic Workshop on Genetic Algorithms and Their Applications*. Finish Artificial Intelligence Society, 105-114.

[15] Kang, N., and Hon, W.L. 2006. *Towards a better solution to the shortest common supersequence problem: the deposition and reduction algorithm*. Symposium of Computations in Bioinformatics and Bioscience (SCBB06) with the International Multi-Symposiums on Computer and Computational Sciences (IMSCCS|06), (Jun, 2006).

[16] Marco, D., and Thomas, S. 2004. *Ant Colony Optimization. Library of Congress Cataloging-in-Publication Data*, ISBN 0-262-04219-3.

[17] Marco, D., Eric, B., and Guy, T. 1999. *Swarm Intelligence - From Natural to Artificial Systems*. Oxford University Press, ISBN 0-19-513158-4.

[18] Ming, L., Qiang, Y., and David E.F. 1992. *Theory and algorithms for plan merging*. Artificial Intelligence, 57(2-3), 143-181.

[19] Surendra, P.B., and Thirupathi, R. 2010. *Implementation of Artificial Bee Colony (ABC) Algorithm on Garlic Expert Advisory System*. International Journal of Computer Science and Research, Vol. 1, Issue 1.

[20] Ning, K., Choi, K.P., Leong, H.W., and Zhang, L. 2005. *A Post Processing Method for Optimizing Synthesis Strategy for Oligonucleotide Microarrays*. Nucleic Acids Research, 33:e144.

[21] Ren, M., and Martin, M. 1998. *An island model based Ant System with lookahead for the shortest supersequence problem*. Fifth International Conference on Parallel Problem Solving from Nature, vol.-1498 of Lecture Notes in Computer Science, 692-701.

[22] Vadim, G.T. 1990. *Complexity of common subsequence and supersequence problems and related problems*. Cybernetics 25, 565-580.