ABC-GSX: A hybrid method for solving the Traveling Salesman Problem

Anan Banharnsakun, Tiranee Achalakul, Booncharoen Sirinaovakul
Department of Computer Engineering
King Mongkut's University of Technology Thonburi
Bangkok, Thailand
smilecolon@yahoo.com, tiranee@cpe.kmutt.ac.th, boon@kmutt.ac.th

Abstract— **An optimization problem is a problem of finding the best solution from all possible solutions. In most computer science and mathematical applications, the decision to select the best solution is not polynomially bounded. Heuristics approaches are thus often considered to solve such NP-hard problems. In our work, we focus on developing a heuristic method to solve a combinatorial optimization problem known as the Traveling Salesman Problem or TSP. Our technique implements the Artificial Bee Colony algorithm, which is inspired by the decision making process of the honey bees in finding optimal food sources. We extend the ABC algorithm with Greedy Subtour Crossover to improve the precision. In this hybrid procedure, the exploitation process in the ABC algorithm is improved upon by the Greedy Subtour Crossover method. The new proposed method is called ABC-GSX. We then empirically assess performance of our proposed work using functions from a standard TSP library. Experimental results show improvements in both precision and computational time compared to techniques presented in recent literatures.**

Keywords- Artificial Bee Colony; Swarm Intelligence; Greedy Subtour Crossover; Optimization; Traveling Salesman Problem

I. INTRODUCTION

An optimization problem is a problem of finding the best solution from all feasible solutions. It can be categorized into two major groups: combinatorial and numerical optimization problems. Most of such problems are considered *NP*-hard; it is strongly believed that they cannot be solved to optimality within polynomial computation time.

The Traveling Salesman Problem (TSP) is an example of combinatorial optimization problems known to be NP-complete. It naturally arises as a sub-problem in various application domains such as network communication, transportation, manufacturing and logistics [1-4].

Generally, the TSP states that for one salesman who wants to visit n different cities, his objective would be to find the sequence of tour that minimizes the cost of travel by visiting each city exactly once and finally returns to the starting point.

TSP is so easy to describe and so difficult to solve. In an example of a 16-city problem [5] where Homer's Ulysses attempts to visit the cities described in the Odyssey exactly once, there are 653,837,184,000 distinct routes. To calculate an optimal route requires 92 hours on a powerful 28 MIPS workstation.

Therefore, in solving TSP, we employ an approximation that finds a near-optimal solution in a reasonable amount of time rather than a method that is guaranteed to find the optimal solution in an exponential time.

Metaheuristic [6] is one of many approximation methods widely used to solve practical optimization problems. In recent years, several algorithms employing metaheuristic approaches such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) or Bee Colony Optimization (BCO) were applied to solve TSP.

To balance the exploration of diverse tours and the exploitation of excellent individuals in GA on TSP, Zhao, et al. [7] introduced a new method called Balancing Exploration and Exploitation GA. This method aims to reduce the execution time of GA and improve the quality of the solutions found. Order crossover and greedy crossover are employed during the crossover step and different fitness functions are utilized in each iteration. Greedy mutation is used.

ACO is a well-known algorithm designed for solving TSP. However, it faces the nontrivial difficulty of being trapped in local optima. To reduce this problem, Sarayut and Siriporn [8] proposed ACO-GA and ACO-PSO which are the hybrid methods. These methods perform based on the adjustment of a parameter Q_0. If Q_0 is high, the local search would be geared towards exploitation, and when Q_0 is low, the search is geared towards exploration.

Zhong, et al. [9] proposed a modified method C3DPSO which applied discrete PSO to solve TSP. A new parameter C_3 or the mutation factor was introduced to make the swarm more resilient against premature convergence.

Based on reported results [7,8,9], these algorithms yield solutions with lower precisions on average. Inspired by the decision making capability of bee swarms, Wong, et al. [10] applied BCO to solve TSP: a bee travels from one city to another until it reaches the destination, and a heuristic transition rule calculated from path distance is employed to aid the bee in deciding which city to visit next. Under this rule, a bee tends to continue visiting the city closest to its current city. However, the precisions from BCO in large problems are still low.

To improve upon these results the ABC algorithm [11], an effective algorithm already being applied to several optimization applications [12], is focused. ABC is based on bees' foraging behavior. It was initially designed for numerical optimization problems.

978-1-4244-7376-2/10/$26.00 ©2010 IEEE

In this paper, we extend ABC to the domain of combinatorial problems. TSP is also used in our experiments to validate the performance of our proposed method. To adapt ABC for solving TSP, we introduce Hybrid Artificial Bee Colony and Greedy Subtour Crossover (ABC-GSX).

GSX is applied during the solution update step in ABC, with the additional benefit of improved performance. Based on the assumption that each solution may have the best subtour for a different part of the tour, GSX helps the bees to generate a better candidate solution by combining appropriate subtours. This hybrid method improves the precision of the results.

This paper is organized as follows. Section II presents a brief overview of the ABC Metaheuristic. Section III proposes the application of the hybrid of the ABC Metaheuristic and Greedy Subtour Crossover onto TSP. Section IV outlines parameters of the experiments, summarizes the results and discusses the performance of the proposed algorithm. Section V draws a conclusion.

II. THE ABC METAHEURISTIC

Artificial Bee Colony (ABC) is a metaheuristic in which artificial bees of a colony cooperate in finding good solutions to optimization problems. A characteristic of ABC is that it was inspired by nature, or more precisely by the behavior of honey bees seeking a quality food source.

A. Bees' foraging behavior

Foraging [13] is how honeybees find food sources. In this process, quality food sources are selected based on group decision making by the swarm. Independence and interdependence in collective decision making are important factors in this mechanism.

The bees independently evaluate the quality of different new candidate food sources on their own and the interdependence among them makes them more attentive to candidate food sources discovered and advertised by others. Waggle dances are done by scout bees in the food source selection process to exchange information on new candidate food sources and to recruit unemployed bees to follow them to those sources. Through this kind of information exchanging and learning, the honeybee swarm manages to discover quality food sources.

ABC algorithm takes concepts from this foraging process to discover good solutions in an optimization problem. Essential components in ABC modeled after the foraging process are defined as follows:

- *Food Source*: This component represents a feasible solution in an optimization problem.
- *Fitness Value*: This value represents the profitability of a food source. For simplicity, it is represented as a single quantity associated with the objective function of a feasible solution.
- *Bee Agents*: This component is a set of computational agents. Honey bees in ABC are categorized into three groups: employed bees, onlooker bees and scout bees. The colony is equally separated into employed bees and onlooker bees. Each solution in the search space consists of a set of optimization parameters which represent a food source's location. The number of employed bees is equal to the number of food sources. In other words, there would be one employed bee for each food source.

B. ABC Metaheuristic

Informally, we can summarize the processes of ABC in three major steps: selection of feasible solutions, updating feasible solutions and avoidance of suboptimal solutions.

Selection of feasible solutions is performed by onlooker bees. When the employed bees return to their hive, they share information about candidate solutions they found with the onlooker bees. The onlooker bees select these solutions based on probability. Solutions of higher fitness have a larger chance of being selected by onlooker bees than ones of lower fitness.

Updating feasible solutions is handled by both employed bees and onlooker bees. Solutions previously found by employed bees and onlooker bees in the neighborhood of a newer and a more fit candidate are replaced.

Avoidance of suboptimal solutions is done by reassigning employed bees whose contribution is rejected as low quality to scout bees which randomly search for new solutions.

These three processes of the ABC metaheuristic can be described in pseudo-code as Fig. 1 follows.

```
Procedure ABC_Metaheuristic
Initial_Solutions
While (criterion)
   Update_Feasible_Solutions (Employed bees)
   Select_Feasible_Solutions (Onlooker bees)
   Update_Feasible_Solutions (Onlooker bees)
   Avoid_Sub-Optimal_Solutions (Scout bee)
End while
```

Figure 1. ABC Metaheuristic Procedure.

C. Problem Representation

In previous literatures, the mapping between the problem and the ABC algorithm was not obviously illustrated, which made it difficult to understand how to map the parameters between the algorithm and the problem. To improve the mapping description, we introduce the use of set and graph theory in this paper.

In this section we give a formal characterization to illustrate how to map a problem in consideration to a representation that ABC metaheuristic can be used to find the solutions.

We consider ABC metaheuristic in terms of (S, f, Ω), where S is the set of candidate food sources, f is the fitness function which assigns a fitness value $f(s)$ to each candidate food source $s \in S$ and Ω is a set of constraints. Food sources belonging to the set $\widetilde{S} \subseteq S$ that satisfy Ω are called feasible food sources. The goal is to find a globally

optimal feasible food source s^*. This consists of finding a solution $s^* \in \tilde{S}$ where $f(s^*) \geq f(s)$ for all $s \in \tilde{S}$.

The terms (S, f, Ω) are mapped to the problem it represents as follows:

Let $G = (V, E)$ be the completely connected graph that represents the problem's search space,

where

- V is a finite set of sub-solutions, ($V = \{v_1, v_2, \ldots, v_{Nv}\}$, N_v being the number of sub-solutions).
- E is a finite set of connections that fully connects V.

A set of dimensions in each candidate food source in ABC is represented as a candidate set of sub-solutions as follows.

$x = \{v_i, v_j, \ldots, v_h, \ldots\}, x \in X$

where

- X is the set of all possible solutions in the graph G,
- The set of candidate food sources $S \subseteq X$ holds,
- $\tilde{X}$ is a set of feasible solutions and $\tilde{X} \subseteq X$ holds,
- S^* is a non-empty set of optimizations, and $S^* \subseteq \tilde{X}$ and $S^* \subseteq S$.

The total weight cost of x determines the fitness value ($f(x)$) which is used to evaluate the candidate solution. The details of the ABC process based on this representation can be described as follows.

First, initial solutions, which form a candidate set of sub-solutions (x) in the graph G, are randomly constructed and assigned to employed bees. After initialization, the artificial bees are subjected to repeated cycles of three major processes: updating feasible solutions, selecting feasible solutions, and avoiding suboptimal solutions.

In the process of updating feasible solutions, employed bees exchange their sub-solutions with neighbors. Old solutions in an employed bee's memory are replaced by new solutions of higher fitness ($f(x)$).

During the selection of feasible solutions, each onlooker bee selects one amongst the proposed solutions. The probability of a solution being selected by an onlooker bee is proportionate to its fitness. After a solution is selected, the bees also update their selected solutions like employed bees do.

In the process of avoiding sub-optimal solutions, all sets of sub-solutions that do not improve the fitness are replaced by new sets of sub-solutions randomly constructed by the scout bees.

A parameter called Maximum Number of Cycles (MCN) determines the number of iterations and is a termination criterion. The processes are repeated until an optimal set of sub-solutions (s^*) is found or the number of iteration equals MCN.

Given this formal characterization, ABC metaheuristic can be applied to a wide variety of interesting optimization problem.

III. MAPPING ABC-GSX METAHEURISTIC TO THE TSP

To illustrate how the algorithm works, we will solve the Traveling Salesman Problem (TSP), a classical optimization problem.

Given a complete undirected graph $G = (V,E)$ with V being the set of cities, E being the set of edges fully connecting the cities V, and each edge $\in E$ being assigned a value d_{ij} that is the cost of the traveling between cities i and j, with $i,j \in V$, the Traveling Salesman Problem is concerned with finding the closed path which visits each of the $n = |V|$ cities of G exactly once with minimal cost. The function of traveling cost $f(v)$ is represented by (1) below:

$$f(v) = \sum_{i=1}^{n-1} d_{v(i)v(i+1)} + d_{v(n)v(1)} \quad (1)$$

where v is a sequence of the tour in a possible closed path.

Fig. 2 describes the steps that need to be taken to solve TSP using ABC-GSX.

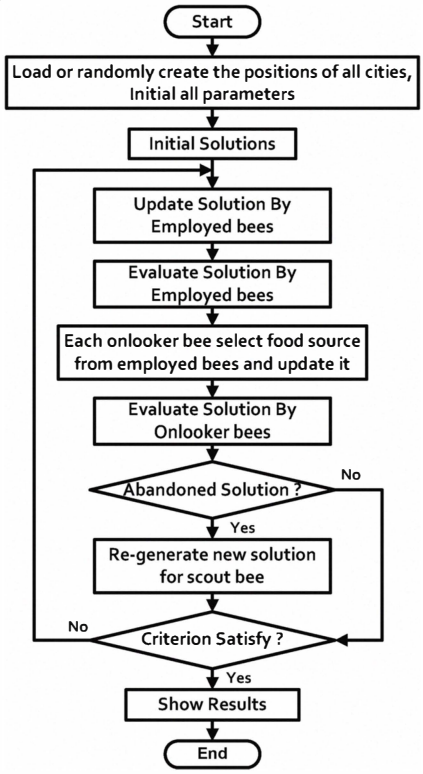

Figure 2. The ABC-GSX algorithm flowchart for TSP.

First, the coordinates of all cities are either randomly generated or loaded from an input file. The cost of traveling between any two cities is determined by their Euclidean distance. A symmetric TSP instance is used in this representation. In symmetric TSP, the traveling cost between two cities is the same in either direction. Next, the initial parameters such as the number of employed bees, onlooker bees and maximum number of iteration (MCN) are set.

New candidate solutions, which are sequences of the tour, are constructed and represented as the food sources. The fitness of each food source is determined by the inverse of its traveling cost. This mapping is shown in Fig. 3 below.

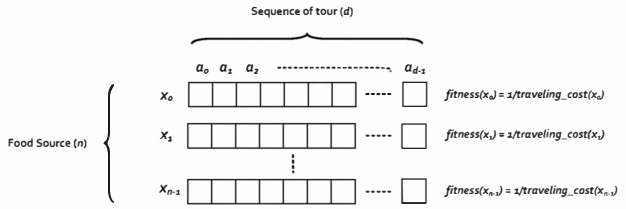

Figure 3. The mapping between the food sources and the tour sequences.

In the second step, the food sources are updated by employed bees. Greedy Subtour Crossover (GSX) [14] is the mechanism used to update the employed bees' old food sources based on their neighboring food sources. This mechanism helps the ABC to generate feasible solutions all the time. The algorithm and an example of GSX are shown in Fig. 4 and 5 respectively.

Note that "⊕" in $x_{new}= a_p \oplus x_{new}$ and $x_{new}=x_{new} \oplus b_q$ is the concatenation operator, and that sentence means to add a_p before x_{new} and to add b_q after x_{new} respectively.

Inputs: the old food source $x_{old} = \{a_0, a_1, \dots, a_{d-1}\}$
the neighboring food source $x_{neighboring} = \{b_0, b_1, \dots, b_{d-1}\}$
Output: The new food source $x_{new} = \{\}$

GSX_Algorithm($x_{old}, x_{neighboring}$){
  $s_a = true$
  $s_b = true$
  Choose city c randomly
  Choose p, where $a_p = c$
  Choose q, where $b_q = c$
  $x_{new} = x_{new} \oplus c$
  do{
    $p = p - 1\ (mod\ d)$
    $q = q + 1\ (mod\ d)$
    if ($s_a = true$){
      if ($a_p \notin x_{new}$){
        $x_{new} = a_p \oplus x_{new}$
      }else{
        $s_a = false$
      }
    }
    if ($s_b = true$){
      if ($b_q \notin x_{new}$){
        $x_{new} = x_{new} \oplus b_q$
      }else{
        $s_b = false$
      }
    }
  }while ($s_a = true$ or $s_b = true$)
  if ($|x_{new}| < d$){
    add the rest of cities to x_{new} in the random order
  }
  return x_{new}
}

Figure 4. Greedy Subtour Crossover method.

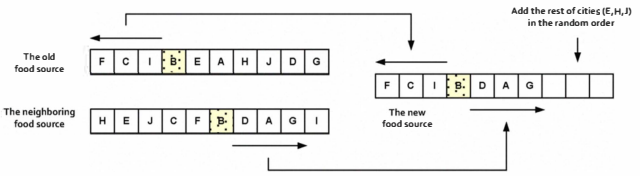

Figure 5. Example of Greedy Subtour Crossover method.

Suppose that we have x_{old}={F,C,I,B,E,A,H,J,D,G} and $x_{neighboring}$={H,E,J,C,F,B,D,A,G,I}. Start by choosing one city at random. In this example we will pick B. Then we derive p=3 and q=5 from the facts a_3 =B and b_5 =B respectively. Now x_{new}={B}.

Next, alternate picking cities from x_{old} and $x_{neighboring}$. Start with a_2 (city I) because p=3-1=2 and continue with b_6=D because q=5+1=6. x_{new} becomes {I,B,D}.

Similarly, add a_1 (city C) and b_7 (city A) and add a_0 (city F) and b_8 (city G). x_{new} becomes {F,C,I,B,D,A,G}. Now the next city becomes a_9=G but since G has already appeared in x_{new}, we cannot add any more cities from x_{old}.

Instead we try to add cities from $x_{neighboring}$. The next city would be b_9=I, but I is already visited in x_{new}. Thus we cannot add cities from $x_{neighboring}$ either.

Then, we add the rest of the cities, i.e. E,H and J , to x_{new} in random order. Finally, x_{new} becomes {F,C,I,B,D,A,G,H,E,J}.

After this updating process is completed, local search based on the 2opt method [15] is also used to improve x_{new}. The 2opt method is shown in Fig. 6 below.

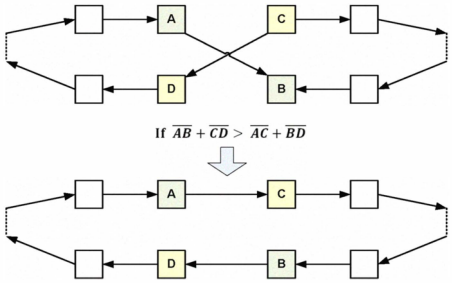

Figure 6. The 2opt method.

In the 2opt method, every pair of edges in the tour sequence is compared to one another. From the example in Fig. 6, if the sum of the traveling cost from A to B and from C to D exceeds the sum of the traveling cost from A to C and from B to D, we remove the two former edges (A to B and C to D) and add the two latter edges (A to C and B to D). This gives us the new tour sequence shown in the lower half of Fig. 6.

An old food source (x_{old}) in an employed bee's memory will be replaced by a new food source (x_{new}) if the new solution has better fitness ($f(x)$).

In the third step, the employed bees share new tour sequences that they have found with the onlooker bees, who

then select the tour sequences of higher fitness and update those sequences based on the same methods used by the employed bees including GSX and 2opt.

In the last step, tour sequences whose fitness has not improved after a certain period are abandoned and replaced by new tour sequences constructed by the scout bee.

Steps 2 to 4 are repeated until the number of iteration reaches the MCN.

IV. Experiment Setting and Results

To verify the validity of our ABC-GSX algorithm on TSP, we benchmarked our results against those reported in [8,9,10,17] and instances from the TSP library [16]. All experiments in this paper were performed on a PC with a 2.83-GHz Intel Core 2 Quad CPU and 4 GB of memory.

A fair comparison in terms of computational efficiency would be difficult. However, we tried to compare our proposed method to other aforementioned methods based on the same parameter settings (e.g. the number of populations and the number of iterations). We thus conducted two separated experiments.

In the first experiment, we compared our proposed method to PSO [17] and C3DPSO [9]. The number of employed and onlooker bees were 15 and the MCN was $N * 10$ where N was the number of cities, matching the parameters used in the experiment on C3DPSO reported in [9].

In the second experiment, we compared our proposed method to ACO-PSO [8] and BCO [10]. The number of employed and onlooker bees were 50. The MCN was 2000. Note that the number of populations used by ACO-PSO and BCO experiments reported in [8,10] were $N * 2$ and N respectively, for the number of iterations shown in Table I.

Each problem instance in each experiment ran for 100 times.

The numerical results from the first and second experiments are shown in Table II and III respectively. The problem size ranged from 51 to 100 cities in the first experiment and from 51 to 318 cities in the second experiment. The "Optimal" figures represent the best known optimal tour cost in each problem instance. The "Best", the "Average", and the "Err" figures show, respectively, the best result, the average result, and the relative error obtained from each algorithm. The relative error is calculated using

Error = [(Average - Optimal) / Optimal] x 100 %

Note that lower "Best" and "Error" values indicate better solutions.

TABLE I. Number of iterations used in ABC-GSX, ACO-PSO and BCO algorithms

Problem	Number of iteration usage		
	ABC-GSX	ACO-PSO	BCO
EIL51	2000	n/a	50000
BERLIN52	2000	2000	n/a
EIL76	2000	n/a	50000
KROA100	2000	3500	50000
KROB100	2000	n/a	50000
CH150	2000	4000	n/a
KROB200	2000	n/a	50000
LIN318	2000	n/a	50000

"n/a" stands for "not available".

It can be drawn that using ABC-GSX on TSP has produced, on average, nearly optimal results in each problem instance. Our proposed approach outperformed all other aforementioned approaches. Average and best-case solution qualities of ABC-GSX were always as good as or better than of those obtained with other approaches.

Maximum relative error never exceeded 2% except for the LIN318 problem instance and average relative error was less than 0.8%. Average improvements on the average results of ABC-GSX method compared to PSO, C3DPSO, ACO-PSO, and BCO across all experiments were 2.54%, 0.84%, 1.87%, and 2.87% respectively.

More importantly, ABC-GSX managed to find globally optimal solutions on most problem instances. ABC-GSX also converged substantially faster with a much smaller number of iterations needed when we focus on the number of iteration usage setting in Table I. Average improvement on the number of iteration usage for ABC-GSX compared to ACO-PSO and BCO method were 16% and 96% respectively.

TABLE II. Results of the first experiment

Problem	Optimal	ABC-GSX			PSO [17]			C3DPSO [9]		
		Best	*Average*	*Err (%)*	*Best*	*Average*	*Err (%)*	*Best*	*Average*	*Err (%)*
EIL51	426	426	430.01	0.9413	427	436.97	2.5751	427	433.64	1.7930
BERLIN52	7542	7542	7544.27	0.0301	7542	7832.05	3.8458	7542	7598.76	0.7526
ST70	675	675	679.81	0.7126	675	697.56	3.3422	n/a	n/a	n/a
EIL76	538	538	550.99	2.4145	546	560.42	4.1673	540	551.72	2.5502
PR76	108159	108159	108658.00	0.4614	108280	112288.08	3.8176	n/a	n/a	n/a
KROA100	21282	21282	21324.30	0.1988	n/a	n/a	n/a	21296	21689.30	1.9138

"n/a" stands for "not available".

TABLE III. RESULTS OF THE SECOND EXPERIMENT

Problem	Optimal	ABC-GSX			ACO-PSO [8]			BCO [10]		
		Best	*Average*	*Err (%)*	*Best*	*Average*	*Err (%)*	*Best*	*Average*	*Err (%)*
EIL51	426	426	426.68	0.1596	n/a	n/a	n/a	428	429.62	0.8497
BERLIN52	7542	7542	7542.00	0	7544	7544.26	0.0299	n/a	n/a	n/a
EIL76	538	538	541.93	0.7305	n/a	n/a	n/a	539	548.81	2.0092
KROA100	21282	21282	21282.00	0	22238	22239.69	4.5000	21762	22011.97	3.4299
KROB100	22141	22141	22147.71	0.0303	n/a	n/a	n/a	22636	22827.37	3.0999
CH150	6528	6533	6603.93	1.1631	6689	6689.24	2.4699	n/a	n/a	n/a
KROB200	29437	29438	29916.10	1.6275	n/a	n/a	n/a	30349	31309.19	6.3599
LIN318	42029	42215	43142.60	2.6496	n/a	n/a	n/a	44685	45202.19	7.5500

"n/a" stands for "not available".

V. CONCLUSIONS

In this work, a hybrid method combining Artificial Bee Colony and Greedy Subtour Crossover (ABC-GSX) was proposed. The mapping of parameters of this algorithm for the combinatorial optimization problem domain was addressed. Traveling Salesman Problem, a classical optimization problem, was chosen to evaluate the effectiveness of this mapping and our proposed method. In this method, the exploitation process in the ABC algorithm is improved by combining GSX. The results indicated that our hybrid method yielded more effective results for TSP, with an average relative error below 0.8%.

ACKNOWLEDGMENT

This work is supported by the Thailand Research Fund through the Royal Golden Jubilee Ph.D. Program under Grant No. PHD/0038/2552.

REFERENCES

[1] J.R.L. Fournier , S. Pierre, "Assigning cells to switches in mobile networks using an ant colony optimization heuristic," Computer Communications., vol. 28, pp. 65-73, 2005.

[2] J.E. Bella, P.R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," Advanced Engineering Informatics., vol. 18, pp. 41–48, 2004.

[3] T.P. Bagchi, J.N.D. Gupta, C. Sriskandarajah, "A review of TSP based approaches for flowshop scheduling," European Journal of Operational Research., pp. 816–854, 2006.

[4] C.A. Silvaa, J.M.C. Sousaa, T.A. Runkler, "Rescheduling and optimization of logistic processes using GA and ACO," Engineering Applications of Artificial Intelligence., vol. 21, pp. 343–352, 2008.

[5] M. Grötschel, M. Padberg, "Ulysses 2000: In Search of Optimal Solutions to Hard Combinatorial Problems," Technical Report, New York University Stern School of Business, 1993.

[6] X.S. Yang, Introduction to computational mathematics. New Jersey: World Scientific Publishing, 2008.

[7] G. Zhao, W. Luo, H. Nie, C. Li, "A Genetic Algorithm Balancing Exploration and Exploitation for the Travelling Salesman Problem," in Proceedings of the 2008 Fourth International Conference on Natural Computation, 2008, pp. 505-509.

[8] S. Nonsiri, S. Supratid, "Modifying Ant Colony Optimization," IEEE Conference on Soft Computing in Industrial Applications, 2008, pp. 95-100.

[9] W.-L. Zhong, J. Zhang, W.-N. Chen, "A Novel Discrete Particle Swarm Optimization to Solve Traveling Salesman Problem," in Proc. IEEE Int. Conf. Evol. Comput. (CEC), 2007, pp. 3283-3287.

[10] L.-P. Wong, M.Y. Hean Low, C.S. Chong, "A Bee Colony Optimization Algorithm for Traveling Salesman Problem," Second Asia International Conference on Modelling & Simulation, 2008, pp. 818-823.

[11] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Erciyes University, Engineering Faculty, Computer Engineering Department, Turkey, Technical Report-TR06, 2005.

[12] D. Karaboga, B. Akay, "A Survey: Algorithms Simulating Bee Swarm Intelligence," Artificial Intelligence Review., vol. 31, pp. 68-85, 2009.

[13] J.C. Biesmeijer, T.D. Seeley, "The use of waggle dance information by honey bees throughout their foraging careers," Behav. Ecol. Sociobiol., vol. 59, pp. 133–142, 2005.

[14] H. Sengoku and I. Yoshihara, "A Fast TSP Solver using GA on JAVA," in Proc. of 3rd International Symposium on Artificial Life and Robotics, vol. 1, 1998, pp. 283-288.

[15] G.A. Croes, ''A method for solving traveling salesman problems,'' Operations Research., vol. 6, pp. 791-812, 1958.

[16] G. Reinelt, "TSPLIB—A traveling salesman problem library," ORSA Journal on Computing., vol. 3, no. 4, pp. 376–384, 1991.

[17] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu, Q.X. Wang, "Particle swarm optimization-based algorithms for TSP and generalized TSP," Information Processing Letters., vol 103, pp. 169-176, 2007.