Hybrid ABC algorithm for the capacitated vehicle routing problem

Ryo Nagaya*
Kanagawa Institute of Technology
Atsugi-City 243-0292 Japan
s1385003@cce.kanagawa-it.ac.jp

Atsushi Inoie
Kanagawa Institute of Technology
Atsugi-City 243-0292 Japan
inoie@nw.kanagawa-it.ac.jp

ABSTRACT

An effective meta-heuristic algorithm for a capacitated vehicle routing problem (CVRP) is studied. In this paper, we proposed the artificial bee colony (ABC) algorithm combined with simulated annealing (SA), and applied the algorithm to a simple CVRP model. We show the advantage of the proposed algorithm through some numerical experiments.

Keywords

Artificial bee colony algorithm, capacitated vehicle routing problem, simulated annealing

1. INTRODUCTION

The capacitated vehicle routing problem (CVRP) aims to optimize the routes for multiple vehicles departing the depot so that all customers are visited at least once before returning to the depot where each vehicle has a finite capacity. In general, the CVRP is NP hard and obtaining a precise real-time optimum solution is difficult for a large-scale problem. Therefore, heuristic solution methods such as meta-heuristic algorithms are traditionally employed.

Szeto et al. [1] applied the artificial bee colony (ABC) algorithm to CVRP. The ABC algorithm, which was originally proposed by Karaboga [2], is modeled on the behavior of bee swarms. Szeto et al. [1] noted that directly applying the ABC algorithm to CVRP will not yield satisfactory results, and instead applied a partially modified algorithm. In this study, we propose an algorithm based on the ABC algorithm proposed in [2] combined with simulated annealing (SA). Its validity for CVRP is demonstrated by a numerical experiment that compares the results here with those of previous studies.

2. CVRP MODEL

CVRP is a problem where costs are minimized by determining the optimal routes for several vehicles departing from a depot such that all customers are visited at least once before returning to the depot. Consider an undirected graph composed of the vertex set (depot 0 and customers $1, \ldots, n$) $V = \{0, 1, \ldots, n\}$ and of the edge set (routes between customers) $E = \{(i, j) : i, j \in V, i < j\}$. The demand cost $d_i \geq 0$ and service time $s_i \geq 0$ are allocated to each node $i \in V$, and the traveling cost $c_{ij} \geq 0$ is allocated to each of the sides (i, j). We denoted the number of vehicles by m, the limit of load capacity for each vehicle by Q, and the time frame necessary to satisfy the needs of customers by L. For each solution (route) $\boldsymbol{x} \in X$, the traveling cost is expressed as $c(\boldsymbol{x})$, while the penalty costs for the load capacity limit and the time limit are expressed as $q(\boldsymbol{x})$ and $t(\boldsymbol{x})$, respectively. The traveling cost is the total of the costs c_{ij} of the edges (i, j) that are traveled by a vehicle k, while the penalty cost is calculated on the basis of the load capacity limit Q and the time limit L. Therefore, the total cost function is expressed as $z(\boldsymbol{x}) = c(\boldsymbol{x}) + \alpha q(\boldsymbol{x}) + \beta t(\boldsymbol{x})$. The coefficients α and β are parameters that are automatically adjusted to a fixed value δ for each repetition of the algorithm using the rules described in [1].

The solution $\boldsymbol{x}$ to the CVRP shall be expressed as follows. Suppose that n customers are to be visited using m vehicles. Then the solution can be expressed via a vector with a length of $(n + m)$.

To create an initial solution, each customer is randomly allocated to a vehicle. In this case, to ensure that a vehicle respects its load capacity limit, it should go back to the depot once its load surpasses the limit, and another vehicle should begin visiting. When all customers have been allocated, the vehicle returns to the depot, and the course traveled until this time is considered the route (solution). τ initial solutions shall be created.

A neighborhood operator is used to generate a new solution $\tilde{\boldsymbol{x}_i}$ from $\boldsymbol{x}_i$. In this article, we use the following three neighborhood operators, random swaps, reversing a subsequence, random swaps of reversed subsequences which is described in [1][1].

3. ABC ALGORITHM

The ABC algorithm is an evolutionary algorithm originally proposed by Karaboga [2], which is modeled on the behavior of bee swarms as they search for a good quality food source (fitness) among several potential sources (solutions) near the nest. In this algorithm, the solution is searched for by three kinds of artificial bees: Employed bee, onlooker bees and scout bees.

In addition, the fitness $f(\boldsymbol{x}_i)$ of the solution , $\boldsymbol{x}_i$, in ABC algorithm is given by $f(\boldsymbol{x}_i) = 1/z(\boldsymbol{x}_i)$.

3.1 Modification by Szeto et al. [1]

*Corresponding author.

[1]Szeto et al. [1] observed with there numerical experiments that the combination of these three neighborhood operators yield the most promising results.

.

BICT 2014, December 01-03, Boston, United States

DOI 10.4108/icst.bict.2014.257837

In [1], the ABC algorithm has been improved. At the update of the solution $\boldsymbol{x}_i$, the neighborhood solution with the largest fitness is expressed as $\hat{\boldsymbol{x}}$. If the fitness of $\hat{\boldsymbol{x}}$ is larger than the current solution $\boldsymbol{x}$, and $\boldsymbol{x}_j$ has the following two characteristics, then $\hat{\boldsymbol{x}}$ supersedes the current neighborhood solution $\boldsymbol{x}$: 1) $\boldsymbol{x}_j$ is the least improved of all the solutions obtained after repeated searches. 2) $\boldsymbol{x}_j$ is a worse solution than $\hat{\boldsymbol{x}}$.

In addition, a solution $\boldsymbol{x}$ that is not updated after a certain number of times is replaced with the neighborhood solution $\tilde{\boldsymbol{x}}$ of the solution.

4. PROPOSED ALGORITHM

The modification in section 3.1 [1] is intended to make the bees work together; that is, the bees collectively search for the best solutions by means of mutual communication of the information on solutions. Consider the case where the algorithm converges to a local solution. Then, the solution has to be replaced with a randomly created solution if we expect further improvement of the solution. We therefore consider the ABC algorithm based on the algorithm proposed by Karaboga [2]. We further try to combine the algorithm with simulated annealing (SA).

The proposed algorithm (called *ABC+SA algorithm*) is as follows:

ABC+SA algorithm.

Step 1. Generate randomly the initial solution $\boldsymbol{x}_i, i = 1, ..., \tau$, and assign each employed bee to an initial solution. Set the number θ of onlooker bees. Set ϕ_i, $\gamma_i \in (0, 1)$ and T.

Step 2. For each $\boldsymbol{x}_i, i = 1, 2, ..., \tau$, compute the fitness value $f(\boldsymbol{x}_i)$.

Step 3. Set $v = 0$.

Step 4. Repeat

a. For each $\boldsymbol{x}_i,\ i = 1, 2, \ldots, \tau$
   i. compare the fitness $f(\boldsymbol{x}_i)$ of solution $\boldsymbol{x}_i$ and the fitness $f(\tilde{\boldsymbol{x}}_i)$ of the neighborhood $\tilde{\boldsymbol{x}}_i$ of $\boldsymbol{x}_i$.
   ii. $\Delta f(\boldsymbol{x}_i) = f(\tilde{\boldsymbol{x}}_i) - f(\boldsymbol{x}_i)$. If $\Delta f(\boldsymbol{x}_i) > 0$, then $\boldsymbol{x}_i = \tilde{\boldsymbol{x}}_i$. Otherwise, generate a uniform random number r in [0,1). If $\exp(\frac{-\Delta f(\boldsymbol{x}_i)}{\phi_i}) > r$ then $\boldsymbol{x}_i = \tilde{\boldsymbol{x}}$. Otherwise, $\boldsymbol{x}_i$ is not updated.
   iii. $\phi_i = \phi_i \times \gamma_i$.

b. Set $G_i = \emptyset, i = 1, 2, ..., \tau$.

c. For each $j = 1, 2, \ldots, \theta$,
   i. Select a solution $\boldsymbol{x}_i$ using the fitness-based roulette wheel selection method where the probability of choosing the solution $\boldsymbol{x}_i$ is p_i.
   ii. Generate the neighborhood $\tilde{\boldsymbol{x}}$ of $\boldsymbol{x}_i$.
   iii. $G_i = G_i \cup \{\tilde{\boldsymbol{x}}\}$.

d. For each $\boldsymbol{x}_i$ and $G_i \neq \emptyset$
   i. Set $\tilde{\boldsymbol{x}} \in \arg\max_{\boldsymbol{x} \in G_i} f(\boldsymbol{x})$.
   ii. $\Delta f(\boldsymbol{x}) = f(\tilde{\boldsymbol{x}}) - f(\boldsymbol{x})$. If $\Delta f(\boldsymbol{x}_i) > 0$, then $\boldsymbol{x}_i = \tilde{\boldsymbol{x}}$. Otherwise, generate a uniform random number r in $[0, 1)$. If $\exp(\frac{-\Delta f(\boldsymbol{x}_i)}{\phi_i}) > r$, then $\boldsymbol{x}_i = \tilde{\boldsymbol{x}}$. Otherwise, $\phi_i = \phi_i \times \gamma_i$ and $\boldsymbol{x}_i$ is not updated.

e. For each solution $\boldsymbol{x}_i$,
   i. If $\phi_i < T$, then replace $\boldsymbol{x}_i$ with a randomly generated solution.

Table 1: Comparison Results of ABC, ABC+SA and SA

	Best known	ABC(Szeto)		ABC+SA		SA	
Problem		Best	Average	Best	Average	Best	Average
vrpnc1	524.61	524.61	526.23	**524.61**	525.43	524.61	529.96
vrpnc2	835.26	**836.74**	842.97	837.59	840.17	843.78	855.42
vrpnc3	826.14	831.16	834.61	**829.48**	836.94	862.46	874.73
vrpnc4	1028.42	**1031.69**	1061.8	1034.01	1040.65	1035.15	1057.04
vrpnc5	1291.29	1320.24	1331.24	**1298.84**	1306.34	1307.81	1325.73
vrpnc6	555.43	555.43	558.09	**555.43**	556.1	555.43	560.27
vrpnc7	909.68	909.68	916.58	**909.68**	922.26	910.63	943.49
vrpnc8	865.94	865.94	876.12	**865.94**	870.55	865.94	868.53
vrpnc9	1162.55	1170.25	1192.64	**1166.4**	1179.03	1170.53	1189.75
vrpnc10	1395.85	1415.23	1434.05	**1401.83**	1413.62	1403.12	1431.46
vrpnc11	1042.11	**1049.91**	1055.41	1056.27	1075.49	1067.25	1087.92

Table 2: Results of large-scale problems

n	ABC(Szeto)		ABC+SA	
	Best	Average	Best	Average
1000	3406.38	3407.95	**3298.22**	3314.21
5000	17317.42	17328.07	**17229.16**	17235.89

f. $v = v + 1$.

g. If v exceeds a stopping limit V, then stop algorithm. Otherwise, go back to Step 4-a.

5. NUMERICAL EXPERIMENTS

In this section, we show the performance of the ABC+SA algorithm by the comparison with the ABC algorithm proposed by [1]. We use the standard benchmark sets [3] in our numerical experiments. Our simulation program is programmed in C++ Language and is performed in a personal computer with an Intel Core i7-3770 3.40 GHz CPU and 16 GB memory.

Let denote $\boldsymbol{x}_{i,j}$ and $\bar{Q}(\boldsymbol{x}_{i,j})$ by the route and the total load of vehicle j in solution $\boldsymbol{x}_i$ respectively. We consider the following penalty cost functions: $q(\boldsymbol{x}) = \sum_{j=1}^{m} \max\{0, \bar{Q}(\boldsymbol{x}_{i,j}) - Q\}$ and $t(\boldsymbol{x}) = 0$ where m is the total number of active vehicle and Q is the capacity of each vehicle.

We set parameter values as follows: $\tau = \theta = 25$, $\alpha = 0.1$, $\beta = 0$, $\delta = 0.001$, $\phi_i = 1$, $\gamma_i = 0.8,\ i = 1, \ldots, \tau$, $T = 0.001$ and $V = 32$. We repeat the simulation 20 times where each simulation takes five minutes. In the neighborhood generation procedure in these algorithms, three neighborhood operators introduced in Section 2 are randomly chosen. Table 1 shows the comparison results of ABC, ABC+SA and SA algorithm. From the result, we confirm that the proposed algorithm outperforms the algorithm by [1] in 8 out of 11 problems

We further apply ABC+SA algorithm to larger-scale problems (see Table 2). We consider two problems whose numbers n of customers are respectively 1000 and 5000 where the location of each customer is randomly determined. We set the parameter values as follows: $Q = 500$, $s_i = 0,\ i = 1, \ldots, n$, $L = \infty$ and the range of each demand cost is $[10,\ 50]$. We repeat the simulation 5 times where each simulation takes 60 minutes.

6. REFERENCES

[1] W.Y. Szeto, Yongzhong Wu, and Sin C. Ho. An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research*, 215(1):126–135, November 2011.

[2] D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Erciyes University, October 2005.

[3] Benchmark problem. `http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/`.