An Application of Binary Cuckoo Search Algorithm to Orienteering Problem

Giovano Alberto and Alfian Tan

***Abstract*—This research applies the cuckoo search metaheuristics model to find solutions to the Orienteering Problem (OP). The OP formulation is useful to model a situation in which someone wants to determine an optimal city route that is subject to a specified time constraint. OP can be categorized into NP-Hard Problem which takes a very long time to analytically find the optimal solution as the number of entities involved increases. Therefore, metaheuristics often become an option to deal with this situation. A cuckoo search model based algorithm is developed in this research. An adjustment for discrete combinatorial problem is performed by adopting an idea of binary cuckoo search method. In addition, three types of local search methods are considered to improve the searching performance. This algorithm can eventually find better solutions for some of the 18 cases than two other benchmarked algorithms. Furthermore, experiment on model parameters shows that the worse nest fraction ($P\alpha$) affects the quality of solutions obtained.**

***Index Terms*—Cuckoo search, orienteering problem, NP-Hard problem, discrete optimization.**

I. Introduction

THE world is currently in a pandemic of COVID 19 where every activity is limited. Every country wants to prevent the spread of the virus, especially the virus transmission from the area outside their borders. Most countries in the world make decisions to lockdown their countries from outside parties. This decision has made many people give up their intention to go abroad either for a business trip, visiting family, or a vacation. This is detrimental to many business areas, especially in the tourism sector. Many hotels are empty because no tourist is coming. Many recreational areas are closed due to lockdown and social distancing. Many restaurant owners are forced to close down their business because there are no customers. Tourism planners cannot do their work because they are locked down. However, the demand for the tourism sector will increase significantly when the pandemic ends as many people will realize their plans which previously had to be restrained due to the pandemic. Tourism planners can benefit from this situation by making tour plans that can satisfy their customers. Before the pandemic, the tourism business has already dealt with the strategy to determine a satisfying tour plan. It is sometimes quite difficult and takes a long time because of the demand variations with only a limited resource available.

Good planning before carrying out a tour is important so that customer satisfaction can be maximized. The planning can be performed by determining tour destinations that suit resource availability as well as customer satisfaction. However, determining some locations to be visited is not easy because there are restrictions that cannot be violated, such as travelling time limits. This tourism problem can be modeled as an orienteering problem [1].

The authors are with the Industrial Engineering Department, Parahyangan Catholic University, Bandung 40141, Indonesia e-mail: giovanoalberto08@gmail.com, alfian.tan@unpar.ac.id.

Manuscript received February 24, 2021; accepted June 2, 2021.

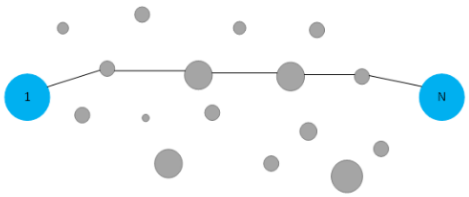

Fig. 1. Illustration of the orienteering problem.

Orienteering problem (OP) is originated from the name of sports activity. Orienteering is an outdoor sport that is usually played in heavily forested areas in which the start and endpoints are specified along with other locations. Other names used for the OP are Selective Traveling Salesperson Problem (STSP) [2], Maximum Collection Problem (MCP) [3], and Bank Robber Problem [4].

For example, there are n alternative nodes with a start and end node at node 1 and node N, respectively. Each alternative node has a score of S_i for node i that is greater than zero, except for the start and end node. An internode movement (node i to node j) requires or has a specific travelling duration and distance. Therefore, OP can be modeled so that there are n nodes in the Euclidean plane (two-dimension), each with a score $S_i \geq 0$ [note that $S_1 = S_n = 0$] and our job is to find a route with a maximum accumulative score across the nodes beginning at 1 and ending at N in which the total length (or duration) cannot be greater than T_{max} [5].

Figure 1 illustrates the orienteering problem [6]. The blue circle with number 1 is the starting node, while the one with number N is the end point. The gray circles between the start and end point are available alternative nodes that can be visited. The size of a circle represents a proportional score received by taking it as our destination. The black line connecting the starting point, nodes in between, and the end points is defined as a feasible route solution.

According to Vansteenwegen et al. [1], The Orienteering Problem can be formulated as an integer programming problem, as follow:

$$\max \sum_{i=2}^{N-1} \sum_{j=2}^{N} S_i x_{ij} \tag{1}$$

$$\sum_{j=2}^{N} x_{1j} = \sum_{i=1}^{N-1} x_{iN} = 1 \tag{2}$$

$$\sum_{i=1}^{N-1} x_{ik} = \sum_{j=2}^{N} x_{kj} \leq 1, \quad \forall k = 2, \ldots, N-1 \tag{3}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} t_{ij} x_{ij} \leq T_{max} \tag{4}$$

$$2 \leq u_i \leq N, \quad \forall i = 2, \ldots, N \tag{5}$$

$$u_i - u_j + 1 \leq (N-1)(1 - x_{ij}), \quad \forall i, j = 2, \ldots, N,$$
$$x_{ij} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, N \tag{6}$$

A set of control nodes are denoted by N where x_{ij} is a binary decision variable that will be equal to one if someone visits node j immediately after visiting node i; otherwise, it will be equal zero. S_i represents the score for node i. Time required to visit node j immediately after visiting node i is denoted by t_{ij} and maximum total time to complete the trip is denoted by T_{max}. The order of nodes i on the final route is denoted by u_i.

The objective of the OP is to maximize the cumulative score of a route as shown in (1). Equation (2) is used to make sure that a travelling path always begins from the starting point 1 and ends at the end point n. All chosen nodes have to be connected and each node is visited at most once as shown in (3). Equation (4) is required to ensure the resulting route does not violate the time constraint of T_{max}. Equations (5) and (6) ensure that no sub-tour occurs.

The orienteering problem which is classified as an NP-Hard problem [5] can be solved by a metaheuristic method. Metaheuristic method is considered a reasonable choice for finding solutions in terms of the time spent on the solution searching activity as well as the quality of the solution itself [7]. Cuckoo Search Algorithm (CSA) shows a good performance in completing the knapsack problem [8] and produces the best solutions for many variants of the traveling salesman problem [9]. According to Vansteenwegen, et al. [1], OP can be considered as a combination of the Knapsack Problem (KP) and the Travelling Salesperson Problem (TSP). Therefore, we expect that CSA may be able to produce good solutions to the orienteering problem. The cuckoo search model will be used in this research to find solutions for the orienteering problem. The Cuckoo Search model has been argued to have an advantage of using Lévy Flight so that it can explore the solution space more efficiently. As Lévy flights has infinite mean and variance, Cuckoo Search can explore the search space more efficiently than ones with standard Gaussian process [10].

There are three main objectives of this research. First, we would like to develop an algorithm based on Cuckoo Search model to solve the Orienteering Problem. Second, we aim to determine the effect of parameters changes which include the worst nest fraction ($P\alpha$) and Lévy flight movement distance parameter on the quality of solutions found for the orienteering problem. Third, we would like to confirm the potency of the developed algorithm by comparing the result with other works on the application of Ant Colony Optimization algorithm [11] and Firefly Algorithm [12] to the Orienteering Problem. In this research, we also consider some possible modifications needed for discrete combinatorial problem by adopting a binary cuckoo search method. In addition, some improvements on searching activities are also made by applying some local search techniques.

The remainder of this paper is organized as follows. Section 2 presents an overview of CSA. Section 3 contains a CSA framework that is used to solve the OP. Experimental results related to model parameter effect are presented in Section 4, and finally the conclusion is presented in Section 5.

II. Cuckoo Search Algorithm (CSA)

CSA was developed by Yang and Deb [13] in 2009 which was inspired by nature. CSA is based on the life of a cuckoo bird. The three basic rule of CSA developed by Yang and Deb are:

1) Each cuckoo lays one egg at a time and dumps it in a randomly chosen nest.
2) The best nests with high quality of eggs (solutions) will be carried over to the next generations.
3) The number of available host nests is fixed, and a host can discover an alien egg with a probability $P_\alpha \in [0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest to build a completely new nest in a new location.

For simplicity, the last assumption can be approximated by a fraction P_α of the N nests being replaced by new nests (i.e. with new random solutions at new locations). In the implementation point of view, we can use the following simple representations that each egg in a nest represents a solution and each cuckoo can lay only one egg (thus representing one solution). The purpose is to use the new and potentially better solutions (cuckoos) to replace a not-so-good solution in the nests [10]. Based on these three rules, the basic steps of the CS can be summarized on the pseudo-code below [13].

Require: N, P_α
Ensure: best solution
  Objective function $f(x)$, $x = (x_1, \ldots, x_d)^T$
  Generate initial population of n host nests x_i ($i = 1, 2, \ldots, n$)
  while $t < MaxGeneration$ **or** stop criterion **do**
    Get a cuckoo randomly by Lévy flights
    Evaluate its quality / fitness F_i
    Choose a nest among n (say, j) randomly
    if $F_i > F_j$ **then**
      replace j by the new solution
    end if
    A fraction (P_α) of worse nests are abandoned and new ones are built
    Keep the best solutions (or nests with quality solutions)
    Rank the solutions and find the current best
  end while
  Postprocess result and visualization

The number of the nests is denoted by N and the worse nest fraction change is denoted by P_α. Equation (7) and (8) are used to generate new solutions $x(t+1)$ for, say cuckoo

i, a Lévy flight is performed where $\alpha > 0$ is the step size which relates to the scales of the problem of interest. In most cases, we can use $\alpha = O(1)$. The product $\oplus$ means entry-wise multiplications.

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus Levy(\lambda) \tag{7}$$

$$Levy(\lambda) \sim u = t^{\lambda}, \quad (1 < \lambda \leq 3) \tag{8}$$

Lévy flights essentially provide a random walk while their random steps are drawn from a Lévy distribution for large steps which has an infinite variance with an infinite mean. Here the consecutive jumps/steps of a cuckoo essentially form a random walk process that obeys a power-law step-length distribution with a heavy tail [14].

III. Methods

This section will discuss the methods and algorithms used to find solutions for the Orienteering Problem. There are 2 sections covering an adjustment made for discrete optimization and additional local search strategies considered to improve the searching performance.

A. Adjustment for Discrete Optimization

Optimization problems can be classified into two main classes: continuous optimization problems and discrete optimization problems. In continuous optimization problems, the solution is represented by a set of real numbers. However, in discrete optimization problems, the solution is represented by a set of integer values. Discrete binary optimization problems are a sub-class of the discrete optimization problems class in which a solution is represented by a set of bits [15]. CSA operates in continuous search space while on the contrary OP works in a discrete space which specifically deals with binary (0 or 1) integer optimization. A binary optimization problem needs a binary solution. Solutions with real (non-integer) values are not acceptable or considered as illegal solution [15]. An adjustment that can be used is the Binary Solution Representation (BSR) [15] which uses a sigmoid function in (9).

$$S(x_i) = \frac{1}{1 + e^{-x_i}} \tag{9}$$

BSR is used when we are looking for solutions to OP problems. It will convert each non-integer value in a series of solution code into binary representation. The BSR solution is obtained by generating random numbers between 0 to 1 for each node and comparing them with a sigmoid value $S(x_i)$. If the random number obtained is smaller than the value of $S(x_i)$, then the node will be worth 1, otherwise 0. In the context of destination route, a value of 1 means the node/place is visited. According to [15], a pseudo code of the BSR algorithm is shown in Algorithm 2.

First of all, it is necessary to generate initial OP solution codes that will represent nests in the cuckoo search. This process creates a x_i value for every node except the starting and the end point because these 2 nodes are compulsorily visited. The original value of x_i ranges from negative infinity to infinity. The value of x_i is an important factor in determining whether a node is visited or not. The higher the value of x_i, the higher the chances of a node being visited. To produce a set of initial solutions x_i, an inverse sigmoid function will be used. A continuous sigmoid value between 0 and 1 will be randomly generated as an input for producing each x_i value. For example, there is a problem with 5 alternative nodes that must start at node 1 and finish at node 2. We will generate a sigmoid value $S(x_3)$ of 0.918 as an input for the inverse sigmoid function to get the value of x_3 (10).

TABLE I
Example of x_i value in BSR

Nodes	**1 (Starting point)**	2 (End point)	3	4	5
x_i	0.0	0.0	2.41	−0.34	−4.29

$$x_i = \ln\left(\frac{S(x_i)}{1 - S(x_i)}\right) \tag{10}$$

Require: Real solution representation x_i
Ensure: x_i'
  for $i = 1$ **to** problem size **do**
    $S(x_i) = \frac{1}{1+e^{-x_i}}$
    if random number $< S(x_i)$ **then**
      $x_i' = 1$
    else
      $x_i' = 0$
    end if
  end for

After doing the same step for the other x_i values, the x_i value for the 5 nodes will be obtained as exampled in Table I. Then x_i will be translated into an OP solution.

Require: N, P_α
Ensure: best solution
  Objective function $f(x)$, $x = (x_1, \ldots, x_d)^T$
  Generate initial population of n host nests x_i $(i = 1, 2, \ldots, n)$
  while $t < MaxGeneration$ **or** stop criterion **do**
    Get a cuckoo randomly by Lévy flights
    Get its binary representation by BSR algorithm
    Evaluate its quality / fitness F_i
    Choose a nest among n (say, j) randomly
    Get its binary representation by BSR algorithm
    if $F_i > F_j$ **then**
      replace j by the new solution
    end if
    Get its binary representation by BSR algorithm
    A fraction (P_α) of worse nests are abandoned and new ones are built
    Get its binary representation by BSR algorithm
    Keep the best solutions (or nests with quality solutions)
    Rank the solutions and find the current best
  end while
  Postprocess result and visualization

The OP solution is a sequence of visited nodes that do not violate any constraints. To translate the solution code into real problem solutions, we first randomly select an available

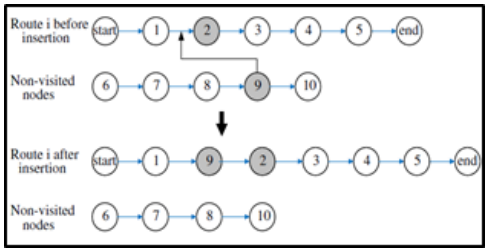

Fig. 2. Insertion process.

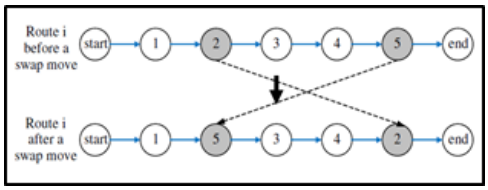

Fig. 3. Swap process.

node then choose another node with the shortest travelling time as a subsequent destination candidate to the previous node. This procedure will be performed until all nodes are included. At this point, we have a node sequence that consists of all nodes. After this, the BSR is applied to decide which nodes to be finally visited and add up each corresponding internode traveling time to check whether the generated route violates the time constraint. The BSR algorithm uses the iterated OP solution code as inputs to probabilistically decide whether a certain node in the node sequence would be visited or not. Finally, we will have a traveling route containing a subset of nodes derived from the initial node sequence. The Binary Cuckoo Search Pseudocode that is used to solve the Orienteering Problem can be shown in Algorithm 3.

The first step of BCS is to initialize the parameters for the algorithm. The BCS algorithm has an advantage of having fewer number of parameters to set than some other algorithms. Parameters in the BCS algorithm include the population size (N) and the worst nest fraction (P_α).

B. Local search

Local search is a method that can be used to help solve and improve the quality of a solution resulted in certain optimization conditions. Local search works by moving from one solution to another in the search space. The solutions initially obtained from the CSA are improved by local search methods. There are 3 local search methods considered in this research as explained below.

- Insertion: this method will choose randomly and add an unvisited node, if it exists, to a random position between the starting node and the ending node of the current tour (see Fig. 2).
- Swap: this method will swap a pair of random nodes existing on a tour, except the starting and the ending nodes (see Fig. 3).
- 2-opt: this method will change the order of nodes by randomly selecting 2 nodes, then reverses the order of the 2 selected nodes (see Fig. 4).

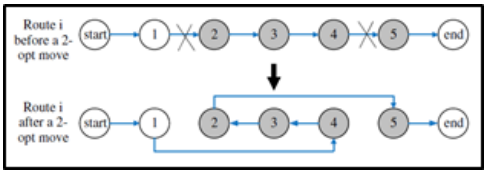

Fig. 4. 2-opt process.

In this research, local search is applied in every iteration. If the total score is higher or the total distance of the new solution is lower, then it will replace the initial solution generated by CSA. The improvement strategy employed in this research is divided into 2 parts which include insertion only and a combined insertion-swap-2-opt technique. The first part is conducted to find a higher score solution, while the second step tries to decrease the total time or increase the total score of a solution. An increase in score can occur if a previously unvisited node becomes a visited one after going through the insertion procedure. Then a reduction in the total time can occur if the swap or the 2-opt process results in a shorter path.

If the solution resulting from the local search is better than the best solution or the best nest in a certain iteration, then the best solution will be updated. If the best nest for a certain iteration is better than the global best solution throughout all iterations, the global-best nest will be updated.

IV. Results and Discussions

The Binary CSA algorithm with 3 local search methods is coded in Java with NetBeans IDE 8.2. All experiments are run using Fujitsu Lifebook LH531 Intel-Core i5 1333 MHz laptop with 4 GB RAM. All calculations use real floating-point precision without rounding or cutting values. The total time of the final route and CPU time (in second) are rounded to three decimal digits. The results of computational experiments are presented in Tables IV through VI.

Three already available hypothetical cases are used to implement the algorithm. These include cases with 32, 21, and 33 nodes with 18, 11, and 20 time-limit variances, respectively. Each case varies in terms of duration (T_{max}). These 3 set of cases can be obtained from www.mech.kuleuven.be [16]. There are similarities in time bound characteristics in each problem set so that only some time limits are selected. In each case, there are six time limits (T_{max}) selected to represent the variations in the time constraint. The selected cases consist of two low-level time limits, two moderate time limits, and two high-level time limits. Table II summarizes some chosen test case set for our experiment.

In this research, we also examine the effect of some algorithm parameters including the worse nest fraction (P_α) and the parameters on Lévy flight which is the distance of levy movement to find a solution (λ). There are three parameter levels used for P_α (i.e. 0.1; 0.5; and 0.9) and λ (i.e. 1, 2, and 3). The values for the other parameters are set to 40 for the number of nests and 1 for α. The stopping criteria used is the completion of 10.000 iterations or a condition in which the best results have not changed for 100 consecutive iterations.

TABLE II
SUMMARIZES CHOSEN TEST CASE SET

	Set 1	Set 2	Set 3
T_{max}	5	15	15
	10	20	20
	40	27	55
	46	30	60
	80	40	105
	85	45	110

TABLE III
ANOVA MULTIFACTOR RESULTS

	T_{max}	P_α	λ	$P_\alpha^*\lambda$
Set 1	5	-	-	-
	10	-	-	-
	40	+	-	-
	46	+	-	+
	80	-	-	-
	85	-	-	-
Set 2	15	-	-	-
	20	+	-	-
	27	-	-	-
	30	+	-	-
	40	+	-	-
	45	+	-	-
Set 3	-	-	-	
	+	-	-	
	+	-	-	
	+	-	+	
	+	-	-	
	-	-	-	

Stopping criteria can impact the result of the algorithm. The determination of the value and condition of the stopping criteria is based on some benchmarked algorithms which are Ant Colony Optimization (ACO) [11] and Modified Firefly Algorithm (MFA) [12]. In ACO [11], the stopping criterion is when the best solution does not change during 100 consecutive iterations while the application of MFA in [12] set their stopping criterion to the max iterations of 10,000. In our experiment, we will run the algorithm for 5 replications for every parameter combination value.

In this study, the effect of 2 parameters (i.e. P_α and λ) including their interaction is statistically tested using ANOVA multifactor with a significance level of 5%. The performance of the algorithm is measured by the objective function value of a solution that can be found.

The result of the ANOVA multifactor test is shown in Table III. The "+" sign indicates that the correspondent parameter affects the quality of solutions of the algorithm while the "-" sign indicates that a parameter does not affect the performance. The results show that P_α affects the algorithm performance in finding solutions for 10 instances, while parameter λ does not give any effect. However, the interaction between parameter P_α and λ affects the performance in finding solutions for 2 instances.

TABLE IV
COMPARISON OF RESULTS ON PROBLEM SET 1

T_{max}	Heuristic methods		CSA	CSA vs Heuristic methods	
	MFA	ACO		MFA	ACO
5	10	10	10		
10	15	15	15		
40	150	155	155	+	
46	175	175	175		
80	275	280	270	-	-
85	285	285	275	-	-

TABLE V
COMPARISON OF RESULTS ON PROBLEM SET 2

T_{max}	Heuristic methods		CSA	CSA vs Heuristic methods	
	MFA	ACO		MFA	ACO
15	120	120	120		
20	200	200	200		
27	230	230	230		
30	265	265	265		
40	395	395	395		
45	450	450	450		

Tables IV through VI show the best results from our algorithm compared to ACO and MFA in solving Orienteering problems. The "+" sign indicates that our algorithm finds a better solution while the "-" sign indicates the opposite. The blank cells indicate that our algorithm produces the same result as the metaheuristics approach in comparison.

Based on the 3 sets of 18 instances of orienteering problem, 14 cases show that our algorithm is able to find the same quality of solutions or even better than the best results previously obtained by the 2 benchmarked algorithms. However, in 4 problem scenarios, the algorithm fails to find solutions that are at least as good as the benchmarked algorithm. The problems involve cases with high time constraints for set 1 and 3. According to these results, we can observe that the Binary CSA based algorithm developed in this research has a difficulty in producing solutions for problem with high time constraints. This condition may be expected because of a higher route combination complexity as there is more flexibility in the time constraint. There are more number of nodes that can be considered to be part of the routing sequence which results in a higher number of feasible route alternatives. This bigger solution space may associate with a higher difficulty in finding the best solution.

Some possible solutions would be to increase the number of nest, try to use a more suitable P_α parameter which has been proved to significantly affect the quality of solutions, or even look back at the encoding method that bridges the continuous and discrete problem in order to make sure an effective encoding which can further help an effective solution space exploration. In the relatively larger time constraint cases in set 2, Binary CSA algorithm can find a good solution, this is estimated because the number of nodes in set 2 is less than set 1 and 3, so that the solution space will be smaller or less complex.

TABLE VI
COMPARISON OF RESULTS ON PROBLEM SET 3

T_{max}	Heuristic methods		CSA	CSA vs Heuristic methods	
	MFA	ACO		MFA	ACO
15	170	170	170		
20	190	200	200	+	
55	520	550	550	+	
60	580	580	580		
105	800	800	770	-	-
110	800	800	770	-	-

Binary CSA based algorithm developed in this research manages to produce the best solutions for 14 out of 18 cases. This algorithm can be considered quite reliable to solve orienteering problem, but obviously it still needs some improvements to solve cases with a higher time limit which represent a more complex scenario.

V. CONCLUSIONS

The algorithm applied in this research shows equal performance among ACO and MFA. Combination of CSA with local search can improve the solution obtained. Based on the result, it is known that changes in parameters P_α affect the solutions produced by CSA for Orienteering Problem. CSA can achieve the same quality of solutions as Ant Colony Optimization in 14 cases and has not yet reached the best performances in 4 cases. Meanwhile, when compared to MFA, CSA can provide a better solution in 3 cases, giving the same value in 11 cases and not yet achieving the best value in 4 cases. The four cases that have not been reached their optimal values are those with high time limits.

Binary CSA is a reliable algorithm to solve orienteering problem and can still be improved to produce a better solution. Some weaknesses that need to be considered are increase the number of nest, use a more suitable P_α parameter which has been proved to significantly affect the quality of solutions, CSA which is a continuous algorithm is significantly influenced by the encoding process so that future research should be emphasized on trying other encoding techniques, combining CSA with other algorithms in generating solutions, and conducting tests in determining stopping criteria.

REFERENCES

[1] P. Vansteenwegen, W. Souffriau, and D. Van Oudheusden, "The orienteering problem: A survey," *European Journal of Operational Research*, vol. 209, no. 1, pp. 1–10, 2011.

[2] G. Laporte, M. Desrochers, and Y. Nobert, "Two exact algorithms for the distance-constrained vehicle routing problem," *Networks*, vol. 14, no. 1, pp. 161–172, 1984.

[3] S. Kataoka and S. Morito, "An algorithm for single constraint maximum collection problem," *Journal of the Operations Research Society of Japan*, vol. 31, no. 4, pp. 515–531, 1988.

[4] E. Arkin, J. Mitchell, and G. Narasimhan, "Resource-constrained geometric network optimization," in *Proceedings of the fourteenth annual symposium on Computational geometry*, 1998, pp. 307–316.

[5] B. Golden, L. Levy, and R. Vohra, "The orienteering problem," *Naval Research Logistics (NRL)*, vol. 34, no. 3, pp. 307–318, 1987.

[6] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "A survey on algorithmic approaches for solving tourist trip design problems," *Journal of Heuristics*, vol. 20, no. 3, pp. 291–328, 2014.

[7] S. Desale, A. Rasool, S. Andhale, and P. Rane, "Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey," *Int. J. Comput. Eng. Res. Trends*, vol. 351, no. 5, pp. 2349–7084, 2015.

[8] K. Bhattacharjee and S. Sarmah, "A binary cuckoo search algorithm for knapsack problems," in *International Conference on Industrial Engineering and Operations Management (IEOM)*, 2015, pp. 1–5.

[9] D. Gupta, "Solving tsp using various meta-heuristic algorithms," *International Journal of Recent Contributions from Engineering, Science & IT (iJES)*, vol. 1, no. 2, pp. 22–26, 2013.

[10] X.-S. Yang and S. Deb, "Cuckoo search: recent advances and applications," *Neural Computing and Applications*, vol. 24, no. 1, pp. 169–174, 2014.

[11] Y.-C. Liang and A. Smith, "An ant colony approach to the orienteering problem," *Journal of the Chinese Institute of Industrial Engineers*, vol. 23, no. 5, pp. 403–414, 2006.

[12] R. Pramudi, "Penerapan modified firefly algorithm pada orienteering problem," *Skripsi Jurusan Teknik Industri Universitas Katolik Parahyangan*, 2018.

[13] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *World congress on nature & biologically inspired computing (NaBIC)*, 2009, pp. 210–214.

[14] ——, "Engineering optimisation by cuckoo search," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1, no. 4, pp. 330–343, 2010.

[15] A. Gherboudj, A. Layeb, and S. Chikhi, "Solving 0-1 knapsack problems by a discrete binary version of cuckoo search algorithm," *International Journal of Bio-Inspired Computation*, vol. 4, no. 4, pp. 229–236, 2012.

[16] T. Tsiligirides, "Heuristic methods applied to orienteering," *Journal of the Operational Research Society*, vol. 35, no. 9, pp. 797–809, 1984.