

Article

# An Improved Discrete Jaya Algorithm for Shortest Path Problems in Transportation-Related Processes

Ren Wang<sup>1</sup>, Mengchu Zhou<sup>1,2,\*</sup> , Jinglin Wang<sup>1</sup> and Kaizhou Gao<sup>1</sup> 

<sup>1</sup> The Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China; ren\_112358@163.com (R.W.); xiaobeijiujiu@126.com (J.W.); gaokaizh@aliyun.com (K.G.)

<sup>2</sup> Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA

\* Correspondence: zhou@njit.edu

**Abstract:** Shortest path problems are encountered in many engineering applications, e.g., intelligent transportation, robot path planning, and smart logistics. The environmental changes as sensed and transmitted via the Internet of Things make the shortest path change frequently, thus posing ever-increasing difficulty for traditional methods to meet the real-time requirements of many applications. Therefore, developing more efficient solutions has become particularly important. This paper presents an improved discrete Jaya algorithm (IDJaya) to solve the shortest path problem. A local search operation is applied to expand the scope of solution exploration and improve solution quality. The time complexity of IDJaya is analyzed. Experiments are carried out on seven real road networks and dense graphs in transportation-related processes. IDJaya is compared with the Dijkstra and ant colony optimization (ACO) algorithms. The results verify the superiority of the IDJaya over its peers. It can thus be well utilized to meet real-time application requirements.

**Keywords:** shortest path problem (SPP); Jaya algorithm; route planning



**Citation:** Wang, R.; Zhou, M.; Wang, J.; Gao, K. An Improved Discrete Jaya Algorithm for Shortest Path Problems in Transportation-Related Processes. *Processes* **2023**, *11*, 2447. <https://doi.org/10.3390/pr11082447>

Academic Editor: Olympia Roeva

Received: 28 July 2023

Revised: 8 August 2023

Accepted: 10 August 2023

Published: 14 August 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The shortest path problem (SPP) is among the most fundamental ones in graph theory. It is concerned with finding the shortest path from a specified origin to a destination in a given graph or network. In general, each edge's distance, time, and price is called cost. This paper uses "cost" to indicate the distance, time, and other physical attributes of a network. It is applied in a variety of domains (both as a stand-alone model and as a subproblem in complex problems), such as transportation [1–3], biological networks [4], social networks [5], circuit board layout [6], and robotic search/navigation processes [7–13]. In recent years, the rapid development of sensing and Internet of Things (IoT) technology has allowed us to obtain environmental changes rapidly [14,15]. Since such changes impact SPP solutions, it becomes increasingly important for us to find SPP solutions efficiently. Efforts are required to develop faster algorithms than traditional and existing ones to solve SPP.

Many classical algorithms and a series of their variants are proposed, e.g., Dijkstra [16–18], Bellman–Ford [19,20], Floyd [21], and A-Star [22,23], to solve SPP. There are some other ways to solve SPP, such as simple heuristics, meta-heuristics, and neural networks. Otte and Correll (2013) [24] propose a parallel framework of the shortest path planning algorithm based on single-query sampling. Zhang, et al. (2014) [25] propose a novel parameter-free minimum resource neural network framework to solve shortest path problems for various graph types. Jan, et al. (2014) [26] present a method based on Delaunay triangulation, improved Dijkstra algorithm (DA), and Fermat points to construct a new graph that can obtain near-shortest paths in a short computational time. Lei, et al. (2015) [27] propose a parallel field-programmable gate array implementation to solve SPP. An extended systolic array priority queue is proposed to allow large-scale priority queue processing. Wang,

et al. (2016) [28] suggest an improved A-star algorithm to solve SPP with position-based learning effects. Huang, et al. (2017) [29] adopt a position-based pruning strategy to reduce the number of node pairs. An algorithm based on point-to-point shortest path calculation is presented to improve efficiency. Li, et al. (2018) [30] designed an efficient shortest-path algorithm that compresses an original network into a small one that preserves the graph properties used to calculate a path. Yang, et al. (2020) [31] present an ant colony algorithm to solve the shortest path problem. Krauss, et al. (2020) [32] propose three different approaches for solving SPP using a quadratic unconstrained binary optimization formulation suitable for development on a quantum annealing machine.

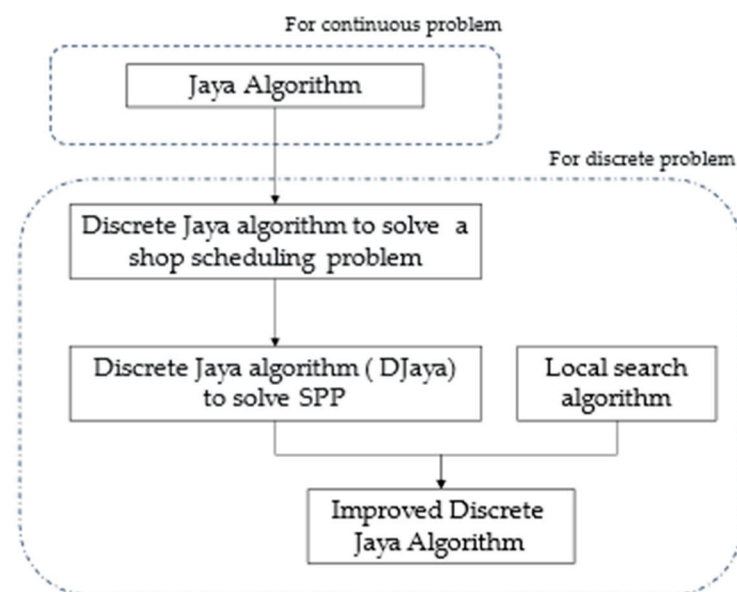
SPP expansions are also widely studied. Examples of other generalizations of SPP are the K-shortest path problem [33], the shortest time path problem with time windows [34], and the traveling salesman problem [35–38]. The application of the SPP extends to the realm of industrial automation. The SPP has been used to optimize and streamline processes, e.g., the design of mechanical components for automation platforms [39], measurement of industrial robot poses [40], and solution of characteristic equations for the elliptical trajectories of industrial robots [41].

Chen, et al. (2013) [42] present a stochastic travel speed model to express the problem of finding the minimal expected time path in stochastic time-dependent road networks. Feng and Korkmaz (2015) [43] propose a new Lagrangian relaxation algorithm to solve a multi-constrained multiple SPP. Talebi, et al. (2017) [44] study online shortest path routing over multi-hop networks. Chemodanov et al. (2018) [45] propose a novel neighborhood method for a constrained SPP. The neighborhood method uses a search space reduction technique and has a theoretical quadratic speed-up that allows it to reach solutions faster than its peers. Li, et al. (2018) [46] present an index-based A-Star algorithm to solve the SPP in a weighted directed acyclic graph with unknown vertex coordinates. Ganganath, et al. (2018) [47] designed a heuristic search algorithm to find solutions to the shortest path on uneven terrains. Su, et al. (2019) [48] propose a multi-stage meta-heuristic algorithm for solving the SPP with must-pass nodes. Kou, et al. (2019) [49] propose two algorithms for a single-constrained SPP. Jie, et al. (2019) [50] establish an information transmission time expectation-variance shortest path model and a fuzzy shortest path critical edge model. An improved ant colony algorithm with traffic congestion factor and random variables is developed to solve the SPP in an uncertain environment. Binh, et al. (2019) [51] suggest an improved evolutionary algorithm that reduces the search space to overcome massive resource consumption in solving clustered shortest-path tree problems. Liu, et al. (2019) [52] propose an optimal path selection algorithm that applies reinforcement learning strategies and designs a new length-first optimal path selection method based on prior knowledge. Xiong, et al. (2020) [53] present an integer programming model for one-to-one pickup and a delivery problem. Two different splitting strategies are proposed to solve the split demand one-to-one pickup and delivery problem with the shortest-path transport along real-life paths. Krauss and McCollum. (2020) [32] designed three different ways to implement the formula of the SPP on a quantum annealing computer. Thanh, et al. (2021) [54] proposed a new method to solve clustered shortest-path tree problems based on a multifactorial evolutionary algorithm. Cosma, et al. (2021) [55] designed a novel genetic algorithm to solve the clustered shortest path tree problem that defines meaningful genetic operators and uses an initial hybrid population.

Although the SPP is studied extensively, researchers face a challenging task when handling large and complex networks in engineering applications. Its various extension problems, such as k-shortest paths, traveling salesman problems, shortest path with time windows, and multiobjective shortest path, are all NP-hard problems. The SPP is frequently invoked when solving these problems. Reducing the running time of the SPP can greatly improve the efficiency of these algorithms. The proposal of a meta-heuristic algorithm to solve the SPP can be regarded as laying the foundation for solutions to other NP-hard problems involving the SPP. In many specific cases, practitioners just need high-quality paths within a very short time instead of the shortest ones but an unacceptably long time

during which any parameter/environmental changes may void an optimal one. In other words, they need the shortest possible time or real-time response to obtain an approximate shortest path. For example, a number of attributes affecting the optimal path in an urban road can change at any time. Such changes require a real-time update of the optimal path. If the guidance is not given quickly, a driver might end up missing an intersection or entering the wrong path. A meta-heuristic algorithm becomes a preferred choice due to its controllable and limited running time and ability to find a high-quality solution [56,57]. The work [58] proposes a simple meta-heuristic algorithm called Jaya to solve continuous optimization problems. Compared with other meta-heuristic algorithms, Jaya is easier to be applied because it does not need to consider complex parameters but just needs to set the number of iterations and population size as its outstanding feature. Jaya is highly competitive in comparison with other heuristic algorithms, e.g., particle swarm optimizer (PSO), genetic algorithm (GA), differential evolution (DE), artificial bee colony (ABC), and teaching-learning-based optimizer (TLBO). It has been extensively used in various fields, such as photovoltaic systems [59], electricity theft detection [60], power transmission [61], unmanned aerial vehicle control [62], manufacturing [63], and scheduling [64,65].

We propose to adopt Jaya to solve the SPP for the first time because of its excellent performance and successful engineering optimization problems. The SPP aims to find the shortest path on a given network that requires connecting discrete points in series and finally giving a set of nodes and their order. It is a discrete problem. The original Jaya algorithm is not suitable for solving discrete problems. Gao, et al. (2016) [57] propose a discrete Jaya algorithm by improving the strategy for new solution generation and applying it to manufacturing process scheduling problems. This work proposes an improved discrete Jaya (IDJaya) algorithm for solving the SPP. Figure 1 shows the flowchart of our proposed method. This work contributes to the field of the SPP by (a) developing a discrete version of Jaya to solve it and a local search operator to improve the performance of the discrete Jaya algorithm (DJaya) and (b) comparing the proposed IDJaya with its peers.



**Figure 1.** Proposed method [57,58].

The remainder of this paper is organized as follows. The mathematical model of the SPP is formulated in Section 2. Section 3 introduces the IDJaya to solve the SPP. Section 4 presents the experimental results and their comparisons with their competitive peers. Finally, Section 5 concludes this paper.

### 2. Problem Description and Mathematic Formulation

In this section, we illustrate our concerned SPP. A network is represented by an undirected and connected graph  $G = (V, E)$ , where  $V = \{v_1, v_2, \dots, v_n\}$  represents the set of nodes and  $E = \{e_1, e_2, \dots, e_m\}$  represents the set of edges.  $|V| = n$  and  $|E| = m$  represent the total number of nodes and edges in graph  $G$ , respectively. An edge between nodes  $v_a$  and  $v_b$  is denoted by  $e_{ab}$ . Each edge has a cost  $c_{ab}$  (e.g., representing distance, time, or price) that indicates the cost between nodes  $v_a$  and  $v_b$ .  $p_{od} \in P$  is a path connecting origin ( $v_o$ ) and destination ( $v_d$ ), where  $P$  is the set of paths. The cost of  $p_{od}$  is represented by  $C_{od}$ , which is calculated by summing the costs of each edge in  $p_{od}$ .

Definition 1 (Shortest Path): For a given graph  $G$ , the shortest path from  $v_o$  to  $v_d$  is defined as the path with the smallest  $C_{od}$ .

As shown in Figure 2, there are three alternative paths, i.e.,  $p_{1,3}^1$ : 1-2-3,  $p_{1,3}^2$ : 1-3 and  $p_{1,3}^3$ : 1-4-3. Accordingly,  $C_{1,3}^1 = 7$ ,  $C_{1,3}^2 = 5$  and  $C_{1,3}^3 = 9$ . Here,  $p_{1,3}^2$ : 1-3 is the shortest path in this network. If there is more than one shortest path between  $v_o$  and  $v_d$ ,  $p_{od}$  is one of them.

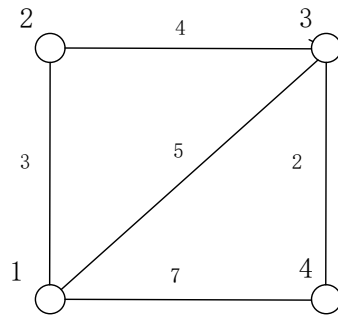


Figure 2. A small network for introducing the SPP.

We formulate the SPP as follows:

$$\text{Min } p_{od} = \sum_{e_{ij} \in p_{od}, i \neq j} c_{ij} \tag{1}$$

subject to:

$$\forall v_k \in V - \{v_o, v_d\}, v_i \in V - \{v_d\}, v_j \in V - \{v_o\}, \sum_{v_i, v_j, v_k \in \{P\}, i \neq k} x_{ik} - \sum_{v_i, v_j, v_k \in \{P\}, j \neq k} x_{kj} = 0 \tag{2}$$

$$c_{ij} = c_{ji}, \forall v_{i,j} \in V \tag{3}$$

$$\sum_{v_i, v_j \in \{P\}, i \neq j} x_{ij} \leq 1 \tag{4}$$

$$x_{ij} \in \{0, 1\}, \forall i, j = \{1, 2, \dots, n\}, i \neq j \tag{5}$$

The objective function (1) serves to minimize path costs. Constraint (2) guarantees network connectivity. (3) restricts that the network is an undirected graph. (4) ensures that each node is accessed at most once, i.e., paths have no loops. (5) defines a binary decision variable.

### 3. Proposed Solution Approach

As networks become large, classic SPP algorithms fail to handle the SPP. A new discrete Jaya algorithm is thus proposed to solve it. Original Jaya is not suitable for solving the SPP because each node is discrete, and Jaya cannot give a solution with discrete values. We extend Jaya by revising a generation strategy of solutions to address this problem such that the proposed IDJaya can access nodes in a network to generate candidate solutions (paths).

### 3.1. Original Jaya Algorithm

Jaya is a novel and simple algorithm. It produces new solutions and constantly tries to find the solutions closer to the optimal solution but away from the worst one. Note that “Jaya” means victory in Sanskrit. Its performance relies on only two parameters, i.e., the number of iterations and population size. It can be viewed as being parameter-free when compared with other popular nature-inspired algorithms such as PSO and ACO. Its simplicity makes it convenient to implement in any programming language, and it is also easy to convert codes between different programming languages. It has the following steps according to [58]:

1. Initializing the population size ( $K$ ), termination condition (i.e., number of iterations), and number of decision variables.
2. Finding the best and worst solutions in population.
3. Generating new solutions via:

$$Y'_{j,k,i} = Y_{j,k,i} + R_1(Y_{j,best,i} - |Y_{j,k,i}|) - R_2(Y_{j,worst,i} - |Y_{j,k,i}|) \tag{6}$$

4. If the new solution is superior to the best solution, then replace the latter with the former.
5. Comparing original solutions with new ones and saving top  $K$  results.
6. Determining whether a pre-given termination condition is met. If so, output results. Otherwise, repeat Steps 2–6.

### 3.2. Discrete Jaya Algorithm for SPP

Jaya is proposed to solve a continuous optimization problem. Since the SSP is discrete, we have to design a new solution generation strategy to make Jaya suitable for the SPP. Therefore, a discrete Jaya algorithm (DJaya) for solving the SPP is proposed. An additional term that can randomly generate adjacent nodes is added to increase the exploration ability of DJaya, called IDJaya. The difference between IDJaya and DJaya is that IDJaya adds a local search algorithm to improve its solution accuracy. Due to the particularity of the SPP, a newly produced node may not be in an original path, and parameters  $\tilde{\varphi}_{j,k,i}/\check{\varphi}_{j,k,i}/\hat{\varphi}_{j,k,i}$  are thus added to ensure the connectivity of the generated paths. IDJaya generates a set of points and builds a path as follows:

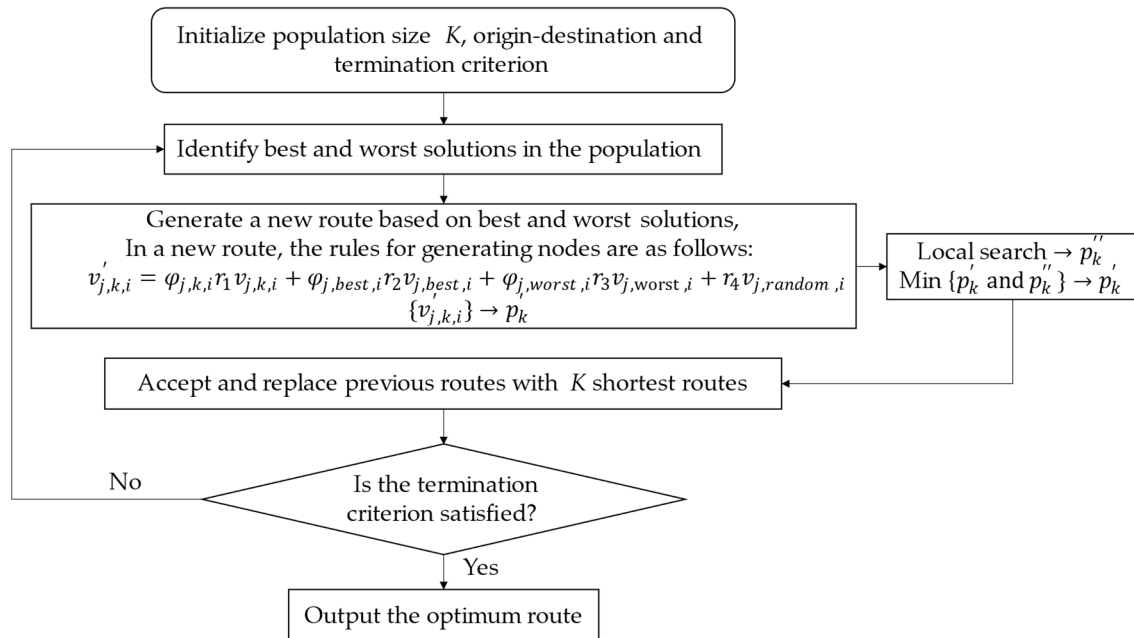
$$v'_{j,k,i} = \tilde{\varphi}_{j,k,i}r_1v_{j,k,i} + \check{\varphi}_{j,best,i}r_2v_{j,best,i} + \hat{\varphi}_{j,worst,i}r_3v_{j,worst,i} + r_4v_{j,random,i} \tag{7}$$

$$\{v'_{j,k,i}\} \rightarrow p'_k \tag{8}$$

$$r_1, r_2, r_3, r_4 \in \{0, 1\} \cap \sum_{j=1}^4 r_j = 1 \tag{9}$$

where  $r_i$  is a binary random number taking 0 or 1 at random and satisfying that only one of them be 1, implying that only one of them is selected to produce a new node;  $\tilde{\varphi}_{j,k,i}$ ,  $\check{\varphi}_{j,k,i}$  and  $\hat{\varphi}_{j,k,i}$  are introduced to prevent IDJaya from being unable to continue running since newly generated nodes are not in the paths. They are used to determine whether the node is in the initial path, best path, and worst path, respectively. They are binary decision variables.  $\tilde{\varphi}_{j,k,i} = 1$  when the node can be found in the initial path, and otherwise zero. So are  $\check{\varphi}_{j,k,i}$  and  $\hat{\varphi}_{j,k,i}$ . For example, a newly generated node is 6. Assuming that the initial path and best path contain node 6, we have:  $\tilde{\varphi}_{j,k,i} = \check{\varphi}_{j,k,i} = 1$  and  $\hat{\varphi}_{j,k,i} = 0$ .  $v'_{j,k,i}$  is a newly generated node and  $v_{j,k,i}$  is an alternative one that is the next node on the selected path.  $v_{j,best,i}$  and  $v_{j,worst,i}$  represent the next node on the best and worst paths, respectively.  $v_{j,random,i}$  includes all nodes accessible by the previous point (except the points that are already in the path). We later give a simple example to show how it works.

Unlike Jaya's solution generation strategy, (7) generates a node in a new path instead of a complete solution. It is constantly called until a new complete path is generated from the origin to the destination. These newly generated nodes build a new path, as shown in (8). The remaining process of the IDJaya is the same as that of the original Jaya. Figure 3 shows the flowchart of IDJaya.



**Figure 3.** IDJaya.

IDJaya requires initial solutions for operation. Initial solutions refer to random paths between an origin to a destination. In order to overcome the difficulty of initial path acquisition in large networks, we propose a random path generation method based on orientation. It always visits a new point in the same direction toward the destination to ensure that a random path can be found quickly.

The idea of our algorithm is based on Jaya. We propose a new solution generation strategy such that Jaya can solve the path problems. A node in a new path is generated when the strategy is invoked. (7) is repeatedly used until the destination is visited and a new path is produced. Its complete procedure is described in pseudo-code of IDJaya with the following steps: (1) set parameters, i.e., population size and the number of iterations; (2) calculate the cost of initial paths and find out the best and worst solutions; (3) select a path in sequence and generate a new point according to Step 6; (4) repeat steps 5–7 until the destination is visited, thus resulting in a new path; (5) use this method to produce corresponding new paths for all initial ones; (6) call the local search algorithm to generate new solutions; (7) compare all paths and select the top  $K$  optimal solutions as the latest path group. (8) repeat the above steps until termination conditions are met, i.e., the maximum number of iterations is reached.; (9) output a path with the lowest cost.  $\{R_k\}$  is a set of paths that is constantly being updated as the algorithm is executed (Algorithm 1).

A local search algorithm is applied to IDJaya to expand the scope of solution exploration and improve the quality of solutions. In the local search algorithm,  $\alpha$  is a randomly selected point in  $p_{od}$ .  $\varepsilon$  represents newly extended points that form a segment. Its steps are as follows: (1) randomly select a node in a path generated by IDJaya; (2) randomly access adjacent nodes from a node selected in the previous step; (3) stop access when a newly accessed node belongs to the initial path or the number of iterations reaches a set value; (4) connect several segments to form a new path; (5) repeat the above steps until the number of paths generated reaches the set value. It ends when the iteration is completed or a newly generated segment returns to the original path. Its time complexity is  $O(n\beta\gamma)$ , and

$T_{\text{Local}} \approx O(n)$  can be obtained.  $T_{\text{Local}}$  is the time complexity of a local search algorithm.  $n$  is the number of nodes.  $\beta$  and  $\gamma$  are two parameters in a local search algorithm. They mean that a local search algorithm is executed to generate at most  $\beta$  paths by exploring no more than  $\gamma$  nodes. Their settings need to prevent the algorithm from wasting too much computing time. As  $\beta$  and  $\gamma$  increase, more paths are generated, and accordingly, more high-quality solutions are likely to be obtained. However, as they increase, so does the running time.  $\beta$  and  $\gamma$  are not sensitive to the size of a network. Their values directly affect the selection range of alternative paths. Larger networks may require more alternative paths to find the optimal solution, but the size of a network does not affect  $\beta$  and  $\gamma$  values. After some experiments, we set  $\beta = 5$  and  $\gamma = 10$  as their proper values (Algorithm 2).

---

**Algorithm 1.** Improved discrete Jaya for shortest path problems (IDJaya)

---

Input: Origin  $O$ , Destination  $D$ , Road network  $G$ , Population size  $K$ , Maximal iteration count  $\hat{I}$

Output: The shortest path

1. Calculate  $\{C_{OD}\}$
  2. Obtain the best value of  $C$  (i.e.,  $(C_{OD})_{\text{best}}$ ) and the worst value of  $C$  (i.e.,  $(C_{OD})_{\text{worst}}$ )
  3. for  $i = 1$  to  $\hat{I}$
  4.     for  $k = 1$  to  $K$
  5.         for  $j = 1$  to  $c$
  6.              $v'_{j,k,i} = \tilde{\varphi}_{j,k,i} r_1 v_{j,k,i} + \check{\varphi}_{j,\text{best},i} r_2 v_{j,\text{best},i} + \hat{\varphi}_{j,\text{worst},i} r_3 v_{j,\text{worst},i} + r_4 v_{j,\text{random},i}$
  7.             end for
  8.              $R'_k \leftarrow \{v'_{j,k,i}\}$
  9.              $P_k \leftarrow \text{Local search}(R'_k)$
  10.         end for
  11.          $\{R_k\} \leftarrow \min\{R_k, R'_k, P_k\}$
  12.         end for
  13.     output  $\min C\{R_k\}$
- 

**Algorithm 2.** Local search algorithm

---

Input: A path generated by IDJaya ( $p_{od}$ )

Output: A set of paths

1.     for  $\beta = 1$  to 5
  2.         Randomly select a node  $\alpha_\beta$  in  $p_{od}$  (except origin and destination)
  3.         Disconnect  $\alpha_\beta$  from the other points in  $p_{od}$
  4.         Let  $\varepsilon_1 = \alpha_\beta$  and  $\text{segment} = \emptyset$
  5.         for  $\gamma = 2$  to 11
  6.             Random access from  $\varepsilon_{\gamma-1}$  to an adjacent node  $\varepsilon_\gamma$  (excluding connections severed in step 3)
  7.             if  $\varepsilon_\gamma \in p_{od}$
  8.                  $P \leftarrow p$
  9.             end if
  10.              $\text{segment} \leftarrow \text{segment} + \varepsilon_\gamma$
  11.         end for
  12.         If  $\varepsilon$  is close to origin than  $\alpha_1$
  13.              $p \leftarrow \text{segment}_{o\varepsilon} + \text{segment} + \text{segment}_{\alpha d}$
  14.         else if
  15.              $p \leftarrow \text{segment}_{o\alpha} + \text{segment} + \text{segment}_{\varepsilon d}$
  16.         end if
  17.          $P \leftarrow p$
  18.         end for
  19.     Output  $P$
- 

**Theorem 1.** The computational complexity of IDJaya is  $O(\hat{I}cK)$ .

**Proof of Theorem 1.** First of all, we summarize the complexity of each step in IDJaya in Table 1.

**Table 1.** IDJaya.

Input: Origin $O$ , Destination $D$ , Road network $G$ , Population size $K$ , Maximal iteration count $\hat{I}$	
Output: The shortest path	
1. Calculate $\{C_{OD}\}$	$O(cK)$
2. Obtain the best value of $f(x)$ (i.e., $(C_{OD})_{\text{best}}$ ) and the worst value of $f(x)$ (i.e., $(C_{OD})_{\text{worst}}$ )	$O(K)$
3. for $i = 1$ to $\hat{I}$	(-)
4. for $k = 1$ to $K$	(-)
5. for $j = 1$ to $c$	(-)
6. $v'_{j,k,i} = \tilde{\varphi}_{j,k,i}r_1v_{j,k,i} + \check{\varphi}_{j,\text{best},i}r_2v_{j,\text{best},i} + \hat{\varphi}_{j,\text{worst},i}r_3v_{j,\text{worst},i} + r_4v_{j,\text{random},i}$	$O(4\hat{I}cK)$
7. end for	(-)
8. $R'_k \leftarrow \{v'_{j,k,i}\}$	$O(\hat{I}cK)$
9. $P_k \leftarrow$ Local search ( $R'_k$ )	$O(n\hat{I}cK)$
10. end for	(-)
11. $\{R_k\} \leftarrow \min\{R_k, R'_k, P_k\}$	$O(2\hat{I}K)$
12. end for	(-)
13. output max $C\{R_k\}$	(-)

Based on Table 1, we formulate the computational complexity of IDJaya as follows:

$$\begin{aligned} T &= O(cK) + O(K) + O(4\hat{I}cK) + O(\hat{I}cK) + O(2\hat{I}K) + O(K) + O(n\hat{I}cK) \\ &= nO(\hat{I}cK) + 2O(\hat{I}K) + O(cK) + 2O(K) \\ &\approx O(\hat{I}cK) \end{aligned}$$

where  $\hat{I}$  is the number of iterations and  $K$  is the population size.  $c$  is the number of nodes on a path. In the worst case, a path passes through all nodes in a network, i.e.,  $c = m$ . On average,  $c \ll m$ .  $n$  is a constant.  $\square$

#### 4. Experimental Results and Discussion

In this section, we illustrate IDJaya via a simple example and then perform two experiments to evaluate the proposed method.

##### 4.1. A Simple Example

To illustrate how IDJaya works, Figure 4 shows a network example. It is an undirected graph consisting of 11 points and 16 edges. Vertical edges cost 3, horizontal edges cost 4, and hypotenuse edges cost 5. In a simple example, the origin is node "1," and the destination is node "11". Setting  $\hat{I} = 2$  (i.e., the algorithm terminates after two iterations) and  $K = 3$ . There are three random initial paths, namely, "1-2-5-8-11", "1-6-5-2-3-4-9-10-11" and "1-2-3-4-9-10-11". We know that the optimal solution to this case is "1-6-7-11" and  $C_{1-11} = 13$ . Since SPP is a minimization problem, the minimum value of  $C$  is regarded as the best solution, and the maximum value of  $C$  is the worst solution.

As shown in Table 2, alternative paths 1 and 2 are the best and worst solutions, respectively. New paths are generated according to (7) and (8). For example, the first iteration of a new path corresponding to alternative path 1 is generated as follows:

$$\begin{aligned} (1) \quad v'_{1,1,1} &\leftarrow \tilde{\varphi}_{1,1,1}r_1(v_2) + \check{\varphi}_{1,\text{best},1}r_2(v_2) + \hat{\varphi}_{1,\text{worst},1}r_3(v_6) + r_4(v_2, v_6) \\ &\leftarrow r_1(v_2) + r_2(v_2) + r_3(v_6) + r_4(v_2, v_6) \\ &\leftarrow v_6 \end{aligned}$$

$$p'_1:1-6$$

$$\begin{aligned} (2) \quad v'_{2,1,1} &\leftarrow r_3(v_5) + r_4(v_5, v_7) \\ &\leftarrow v_5 \end{aligned}$$

$$p'_1:1-6-5$$



It can be observed from Table 4 that path 3 is the best solution and the first candidate is the worst one. Entering the second iteration, new paths are generated using (7) and (8). Table 5 shows the paths of new production and the corresponding  $C$  in the second iteration. Compare the values of  $C_{1-11}$  in Tables 4 and 5, and obtain the top 3 best solutions shown in Table 6. The second iteration is completed. IDJaya is terminated when  $\hat{l} = 2$ , and the optimal solution is output, i.e., “1-6-7-11”. It is determined that “1-6-7-11” is the optimal solution in this example by comparing the result with DA [16].

**Table 5.** New paths and the corresponding  $C$  during second iteration.

<i>Alternate</i>	<i>Path</i>	$C_{1-11}$
1	1-2-3-4-5-8-11	21
2	1-2-5-8-11	15
3	1-6-7-11	13

**Table 6.** Final paths and the corresponding  $C$  at the end of second iteration.

<i>Alternate</i>	<i>Path</i>	$C_{1-11}$	<i>Status</i>
1	1-6-7-11	13	best
2	1-6-5-8-11	15	worst
3	1-6-7-11	13	

In the above example, it can be observed that the optimal solution is obtained in the first iteration. Although there is a small size network, it can reflect the simplicity of IDJaya. Subsequent computations may yield as good results as we have, but excellent solutions are retained by comparing original solutions with new ones. With the increase in population size and iterations, high-quality solutions can be obtained even if the size of a network continues to grow.

## 4.2. Case Study

### 4.2.1. Experiment Setting

In this section, seven road networks from Macao and seven dense graphs are used to verify the performance of IDJaya. Maps of Macao are given from ArcMap 10.2, and dense ones are generated at random. The sizes of actual instances arrange from 82 nodes and 135 edges to 1281 nodes and 1822 edges. Dense graphs arrange from 82 nodes and 676 edges to 1281 nodes and 151,397 edges. They are generated by ArcMap based on real road networks and with the same nodes as real road networks. All graphs have only non-negative costs. The algorithms devised in this paper are implemented in IDEA 2018.3 (the programming language is Java). We conduct the experiments on a laptop running Microsoft Windows 10 with an Intel/2.5 GHz Core (TM) i5-7300HQ CPU, 8 GB of RAM, and 1 TB of hard disk.

IDJaya is compared with typical Dijkstra [16] and very recent methods based on ant colony optimization (ACO) [31,66,67]. This work selects an improved ACO in [31] for comparison. Similarly, it also uses a swarm intelligence algorithm to solve the SPP, which is the latest and most relevant research to our work. The quality and running time of solutions are used to evaluate the performance of an algorithm. The solution quality is defined as follows:

$$\zeta = 1 - \frac{c - \check{c}}{\check{c}} \quad (10)$$

where  $\zeta$  is the quality of the solutions. The longer the path, the smaller the value.  $c$  and  $\check{c}$  represent the cost of a path and the shortest path, respectively.

DA is parameter-free, IDJaya has two parameters, and ACO has five main parameters. IDJaya has the least number of super-parameters among all the meta-heuristic algorithms. Its two parameters are easy to determine. Increasing them tends to improve the quality of its solutions but takes more calculation time. Increasing the maximal iteration count may

not necessarily improve the quality of solutions since IDJaya may fall into a local/global optimum. The increase in population size can effectively reduce the possibility of IDJaya falling into a local optimum. We set  $K = 50$  and  $\hat{I} = 1000$  in our experiment. The values of these parameters are determined based on some tests; acceptable near-optimal solutions can be obtained without spending a lot of running time. All instances are executed 50 times, and the averages are retained and compared.

#### 4.2.2. Algorithm Performance

Experiments are carried out on real road networks and dense graphs, respectively. We randomly assign connections to generate dense graphs corresponding to real road networks, which have the same number of nodes. IDJaya, DJaya, Dijkstra algorithm (DA), and ACO are compared in terms of running time and accuracy. Note that DA always finds the shortest path as an exact algorithm. The results are reported in Tables 7–10. DJaya is an original discrete Jaya algorithm.

**Table 7.** Running time of all compared algorithms in real networks.

Instances		Running Time (ms)											
<i>n</i>	<i>m</i>	Dijkstra [16]			ACO [31]			DJaya			IDJaya		
		<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>
82	135	52.7	60.4	13.4	75.3	85.0	6.5	22.5	<b>33.7</b>	7.4	23.6	35.9	6.3
203	288	64.3	71.8	15.9	93.9	102.1	7.3	31.2	<b>42.7</b>	9.8	35.9	50.4	6.9
443	621	248.7	279.1	10.7	116.7	127.4	9.8	67.0	<b>87.5</b>	13.6	78.4	96.1	7.9
633	900	494.9	554.8	20.4	129	156.9	25	97.1	<b>126.4</b>	19.9	101.5	134.3	14.1
860	1219	919.5	1000.8	27.5	225.8	257.6	24.4	142.3	<b>180.1</b>	28.6	173.7	205.8	18.4
1037	1470	1408.5	1444	16.5	496.4	553.9	29.8	180.7	<b>219.3</b>	30.1	198.1	253.2	24.7
1281	1822	2127.3	2215.4	37.6	947.8	1059.7	49.9	247.8	<b>307.2</b>	42.3	287.1	359.8	27.2

**Table 8.** Accuracy of all compared algorithms in real networks.

Instances		Accuracy										
<i>n</i>	<i>m</i>	Dijkstra [16]		ACO [31]		DJaya			IDJaya			
		<i>Avg</i>	<i>Max</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Max</i>	<i>Avg</i>	<i>SD</i>	
82	135	100%	100%	100%	-	100%	100%	-	100%	100%	-	
203	288	100%	100%	100%	-	100%	100%	-	100%	100%	-	
443	621	100%	99.1%	97.1%	1.8%	100%	100%	-	100%	100%	-	
633	900	100%	98.2%	95.6%	1.4%	100%	100%	-	100%	100%	-	
860	1219	100%	94.8%	93.5%	1.8%	99.3%	98.6%	0.6%	100%	98.6%	0.6%	
1037	1470	100%	93.0%	90.2%	1.6%	98.9%	98.1%	0.3%	99.3%	98.4%	0.9%	
1281	1822	100%	89.9%	86.6%	1.9%	98.1%	97.6%	0.2%	98.4%	97.9%	0.4%	

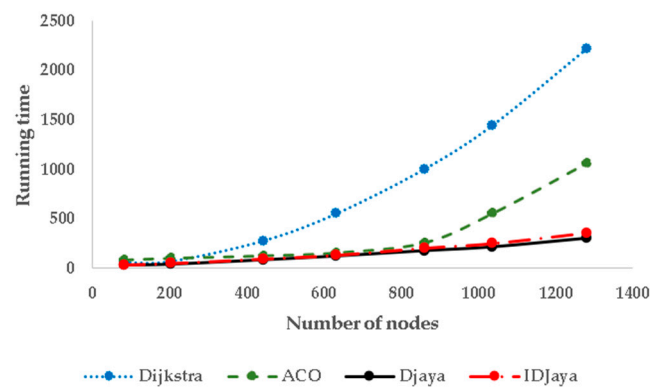
**Table 9.** Running time of all compared algorithms in dense graphs.

Instances		Running Time (ms)											
<i>n</i>	<i>m</i>	Dijkstra [16]			ACO [31]			DJaya			IDJaya		
		<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>
82	1281	62.9	86.6	11.5	77.5	91.7	10.6	24.9	<b>33.5</b>	6.63	20.9	37.1	15.0
203	4105	93.8	108.3	9.14	100.2	113.4	11.7	36.2	<b>45.6</b>	6.58	42.2	59.5	13.1
443	16,724	359.8	382.8	34.5	121.2	141.1	14.5	69.5	<b>92.7</b>	21.62	87.7	102.6	13.5
633	35,891	660.4	732.5	38.8	163.8	186.1	15.5	107.5	<b>137.7</b>	23.24	124.8	150.2	18.7
860	66,012	1268.3	1382.3	57.6	263.2	298.7	19.5	147.9	<b>187.5</b>	30.01	197.7	228.7	22.8
1037	106,324	1907.6	1988.3	56.1	594.8	621.9	27.0	194.3	<b>246.6</b>	39.58	264.5	287.4	14.2
1281	151,397	2606.1	2749.7	82.1	1121.3	1202.9	54.7	280.8	<b>324.4</b>	29.61	339.2	398.1	35.2

**Table 10.** Accuracy of all compared algorithms in dense graphs.

Instances		Accuracy									
<i>n</i>	<i>m</i>	Dijkstra [16]	ACO [31]		DJaya			IDJaya		<i>SD</i>	
		<i>Avg</i>	<i>Max</i>	<i>Avg</i>	<i>SD</i>	<i>Min</i>	<i>Avg</i>	<i>SD</i>	<i>Max</i>		<i>Avg</i>
82	1281	100%	100%	100%	-	100%	100%	-	100%	100%	-
203	4105	100%	100%	98.7%	1.0%	100%	100%	-	100%	100%	-
443	16,724	100%	98.4%	96.8%	1.5%	100%	100%	-	100%	100%	-
633	35,891	100%	97.5%	95.0%	1.1%	99.6%	99.5%	0.1%	100%	100%	-
860	66,012	100%	94.4%	92.4%	1.7%	99.0%	98.0%	0.6%	99.3%	98.5%	0.6%
1037	106,324	100%	91.2%	87.5%	2.3%	97.4%	96.8%	0.4%	98.0%	97.4%	0.3%
1281	151,397	100%	87.4%	82.9%	2.6%	97.0%	96.1%	0.4%	98.5%	97.2%	0.9%

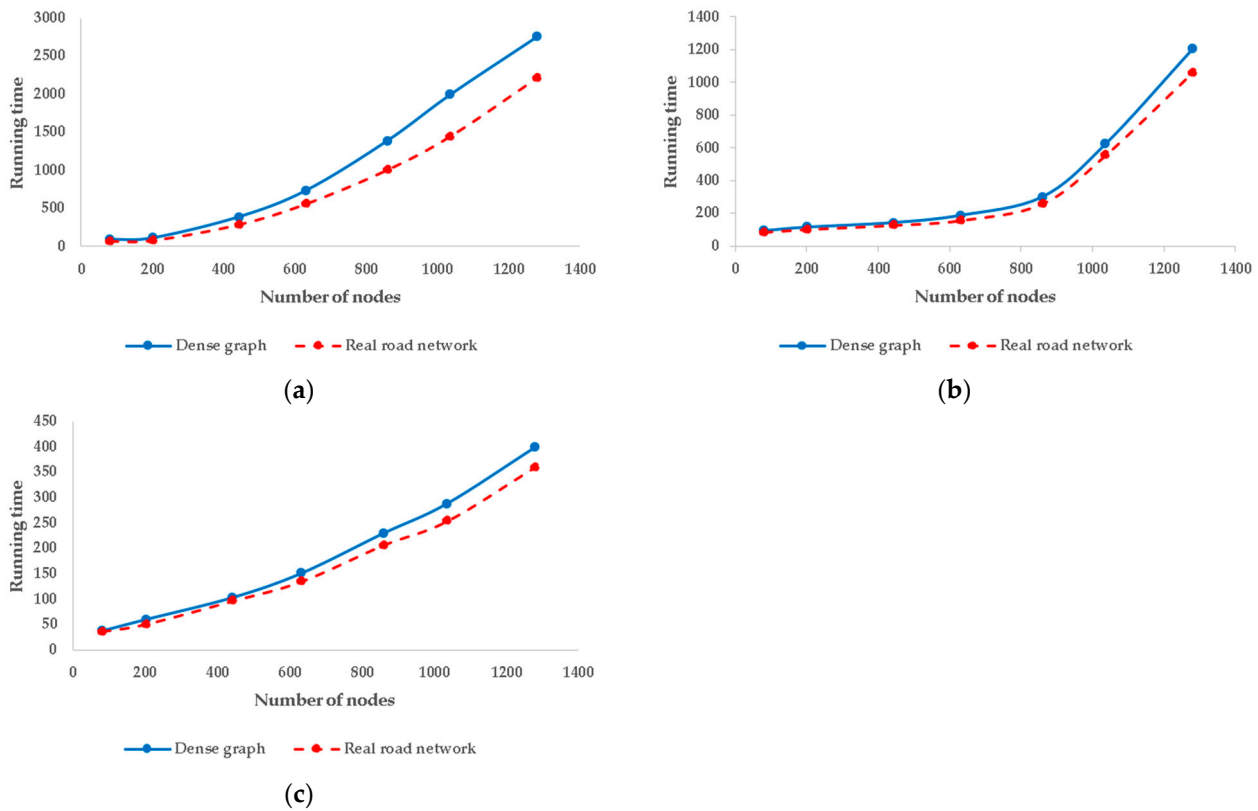
It can be seen from Table 7 that IDJaya and DJaya are always superior to DA in running time, and their advantage is increasing with the expansion of the network scale. The running times of IDJaya and DJaya are half of the DA when there are 82 nodes. When the number of nodes is 1281, their running time is only about one-seventh of the latter. Similarly, compared with ACO, our methods can give solutions of higher accuracy faster. DJaya's running time is slightly less than IDJaya's since IDJaya runs a local search algorithm to increase accuracy. Figure 5 shows that their advantages in terms of calculation time become more and more significant as network size increases. As shown in Table 8, IDJaya and DJaya can always find the shortest paths in the first four instances. They both find the same near-optimal path in the fifth instance. The accuracy of IDJaya becomes more advantageous compared with DJaya and ACO as the network scale increases. Significantly, IDJaya can easily adjust parameters, and users can adjust them according to the SPP's characteristics. Accuracy can be improved by increasing population size and iterations to meet higher accuracy requirements at the cost of longer running time.

**Figure 5.** Average computational time of Dijkstra, ACO, and IDJaya for real road networks.

We use high-density networks in Tables 9 and 10. Similarly, IDJaya takes much less running time than DA and ACO and a bit more than DJaya. The increase in edges has a more significant impact on the calculation time of DA and ACO. The running time spent by IDJaya in dense graphs increases by only a tiny fraction compared to real road networks. Its solution accuracy has decreased but not significantly, falling within an acceptable range. The sixth instance is less accurate than the seventh at the maximum value of accuracy since IDJaya, as a meta-heuristic algorithm, is stuck at a local optimum at running and cannot find a better solution. This locally optimal solution is produced with some randomness. It is possible to obtain a better result in a more complex network.

In Figure 6, the running time of the three algorithms applied to different network types is compared. The increase in the number of edges has little effect on the running time of our algorithm, and DA needs to spend extra time dealing with massive edges. ACO is

less affected than DA. As can be seen from the trend in Figure 6, as the number of nodes increases, the impact of the network density on running time is minimal for IDJaya. The time complexity of IDJaya also supports this result. Its time complexity is  $O(\hat{I}cK)$ , and thus IDJaya has good performance on density graphs.



**Figure 6.** (a) Average computational time of the Dijkstra algorithm; (b) average computational time of ACO; and (c) average computational time of IDJaya in dense graphs.

As shown in Table 11, the main ideas of IDJaya are introduced. Compared to existing heuristic algorithms, IDJaya is simpler and easier to implement since we only need to set two parameters for it. In the process of solving the SPP, just two equations need to be used to generate new solutions. IDJaya’s time complexity is  $O(\hat{I}cK)$ , while the compared approach is  $O(n^2)$ . It can be seen that IDJaya’s time complexity is mainly affected by the maximal iteration count ( $\hat{I}$ ) and population size ( $K$ ). When a network is large, the number of nodes far exceeds  $\hat{I}$  and  $K$ ; thus IDJaya has an advantage in terms of running time.

**Table 11.** Comparison of Different Approaches.

Alternate	IDJaya	Dijkstra [16]	ACO [31]
Main idea	It always tries to move closer to the best solution and move away from the worst solution. Thus, the optimal solution is constantly approached.	Finding the shortest path by accessing and updating the distance of all points from an origin.	Mimicking ants’ foraging behavior. The shorter the path, the more pheromones. Eventually, the whole colony will converge on the optimal path.
Number of parameters	2	-	6
Number of related equations	2	2	5

Table 11. Cont.

Alternate	IDJaya	Dijkstra [16]	ACO [31]
Time complexity	$O(\hat{I}cK)$	$O(n^2)$	$O(n^2)$
Solution quality	Near-optimal solution (over 97%)	Optimal solution	Solution quality is worse than IDJaya

## 5. Conclusions

The SPP is a combinatorial optimization problem widely seen in various engineering applications, especially transportation-related processes. To use the Jaya to solve the SPP, we have to discretize it and improve a generation strategy to make it suitable for the SPP. In this work, we successfully design an improved discrete Jaya algorithm to solve the SPP. We first adjust the discrete Jaya algorithm in [57], and a new solution generation strategy is given to make it suitable for the SPP. Then, we present a local search algorithm to increase the accuracy of this proposed method. The proposed IDJaya has the characteristics of short running time and excellent accuracy validated in many experiments with network sizes ranging from 82 to 1281 nodes. Computational results on seven real road networks and corresponding density graphs demonstrate that the proposed algorithm can yield high-quality solutions for the SPP with up to 1281 nodes and 151,397 edges in a short time. It should be noted that IDJaya performs very well in dense graphs. The results of its time complexity analysis show that increasing the number of edges has little influence on it.

Our future work plans to improve IDJaya further to obtain more accurate results and further explore its performance in large-scale networks [68]. IDJaya has the potential to be extended to solve other path problems, such as the constrained SPP and path planning problems in dynamic and 3-D networks. Its combination with recent deep learning methods [69–73] should be investigated to handle large-scale processes and systems [74–84]. Additionally, we are interested in considering the application of IDJaya to practical scenarios, i.e., industrial robots [39–41], automatic guided vehicle (AGV) [85,86], autonomous driving [87], resource scheduling [88,89] and multi-agent systems [77,90–93].

**Author Contributions:** Conceptualization, algorithm realization, and original draft preparation by R.W.; Result analysis, research supervision, and paper review and editing by M.Z. and K.G.; paper revision by J.W. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data available in a publicly accessible repository.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Nomenclature

The following abbreviations are used in this manuscript:

Symbol	Description
$V$	A set of nodes
$E$	A set of edges
$P$	A set of paths
$v_i$	$i$ -th node
$e_{ij}$	An edge between nodes $v_i$ and $v_j$
$p_{ij}$	A path form node $v_i$ to node $v_j$
$C_{ij}$	Cost of $p_{ij}$
$c_{ij}$	Cost of $e_{ij}$
$\{p\}$	A set of all nodes along that path

$Y_{j,k,i}$	The value of the $j$ -th variable for the $k$ -th candidate in the $i$ -th iteration
$Y_{j,best,i}$	The value of the $j$ -th variable for the <i>best</i> candidate in the $i$ -th iteration
$Y_{j,worst,i}$	The value of the $j$ -th variable for the <i>worst</i> candidate in the $i$ -th iteration
$Y'_{j,k,i}$	The updated value of $Y_{j,k,i}$
$R$	A random number in $[0, 1]$
$v_{j,k,i}$	The $j$ -th node on the $k$ -th path of the $i$ -th iteration
$v_{j,best,i}$	The $j$ -th node on the best path of the $i$ -th iteration
$v_{j,worst,i}$	The $j$ -th node on the worst path of the $i$ -th iteration
$v_{j,random,i}$	The $j$ -th node randomly generated of the $i$ -th iteration
$v'_{j,k,i}$	An updated node of $v_{j,k,i}$
$p'_k$	An updated path of the $k$ -th path
$n$	Number of nodes
$m$	Number of edges
$I$	Number of iterations
$\hat{I}$	Maximal iteration count
$K$	Population size
$c$	Number of nodes on a path with the most nodes
$\zeta$	Quality of solutions.
$x_{ij}$	A binary decision variable. When a path passes through $e_{ij}$ , $x_{ij} = 1$ ; otherwise, $x_{ij} = 0$
$r_i$	A binary decision variable that has one and only one term equal to 1 from $r_1$ to $r_4$
$\tilde{\varphi}_{j,k,i}/\check{\varphi}_{j,k,i}/\hat{\varphi}_{j,k,i}$	A binary decision variable. When a newly generated node is not in the original path, it is zero; otherwise, one

## References

- Owais, M.; Alshehri, A. Pareto Optimal Path Generation Algorithm in Stochastic Transportation Networks. *IEEE Access* **2020**, *8*, 58970–58981. [[CrossRef](#)]
- Jin, Y.; Xu, J.; Wu, S.; Xu, L.; Yang, D. Enabling the Wireless Charging via Bus Network: Route Scheduling for Electric Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 1827–1839. [[CrossRef](#)]
- Wang, R.; Zhou, M.; Gao, K.; Alabdulwahab, A.; Rawa, M.J. Personalized Route Planning System Based on Driver Preference. *Sensors* **2021**, *22*, 11. [[CrossRef](#)] [[PubMed](#)]
- Kuperstein, I.; Grieco, L.; Cohen, D.P.; Thieffry, D.; Zinoviyev, A.; Barillot, E. The shortest path is not the one you know: Application of biological network resources in precision oncology research. *Mutagenesis* **2015**, *30*, 191–204. [[CrossRef](#)]
- Borkar, V.S.; Reiffers-Masson, A. Opinion Shaping in Social Networks Using Reinforcement Learning. *IEEE Trans. Control. Netw. Syst.* **2021**, *9*, 1305–1316. [[CrossRef](#)]
- Rao, C.S.; Tunga, S.; Kumar, A. Analysis of High-Speed Design on a Multilayer PCB Substrate. In Proceedings of the 2021 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT), Bangalore, India, 27–28 August 2021; pp. 713–717. [[CrossRef](#)]
- Gao, Z.; Qin, J.; Wang, S.; Wang, Y. Boundary Gap Based Reactive Navigation in Unknown Environments. *IEEE/CAA J. Autom. Sin.* **2021**, *8*, 468–477. [[CrossRef](#)]
- Huang, L.; Zhou, M.; Hao, K. Non-Dominated Immune-Endocrine Short Feedback Algorithm for Multi-Robot Maritime Patrolling. *IEEE Trans. Intell. Transp. Syst.* **2019**, *21*, 362–373. [[CrossRef](#)]
- Hu, B.; Cao, Z.; Zhou, M. An Efficient RRT-Based Framework for Planning Short and Smooth Wheeled Robot Motion Under Kinodynamic Constraints. *IEEE Trans. Ind. Electron.* **2020**, *68*, 3292–3302. [[CrossRef](#)]
- Li, B.; Chen, B. An Adaptive Rapidly Exploring Random Tree. *IEEE/CAA J. Autom. Sin.* **2021**, *9*, 283–294. [[CrossRef](#)]
- Chen, L.; Hu, Z.; Zhang, F.; Guo, Z.; Jiang, K.; Pan, C.; Ding, W. Remote Wind Farm Path Planning for Patrol Robot Based on the Hybrid Optimization Algorithm. *Processes* **2022**, *10*, 2101. [[CrossRef](#)]
- Zhang, H.; Ge, Y.; Sun, C.; Zeng, H.; Liu, N. Picking Path Planning Method of Dual Rollers Type Safflower Picking Robot Based on Improved Ant Colony Algorithm. *Processes* **2022**, *10*, 1213. [[CrossRef](#)]
- Wang, B.; Wang, J.; Huang, Z.; Zhou, W.; Zheng, X.; Qi, S. Motion Planning of an Inchworm Robot Based on Improved Adaptive PSO. *Processes* **2022**, *10*, 1675. [[CrossRef](#)]
- Li, Z.; Wu, X.; Zhang, S.; Min, L.; Feng, Y.; Hang, Z.; Shi, L. Energy Storage Charging Pile Management Based on Internet of Things Technology for Electric Vehicles. *Processes* **2023**, *11*, 1561. [[CrossRef](#)]
- Karunanidiy, D.; Ramalingam, R.; Dumka, A.; Singh, R.; Alsukayti, I.; Anand, D.; Hamam, H.; Ibrahim, M. An Intelligent Optimized Route-Discovery Model for IoT-Based VANETs. *Processes* **2021**, *9*, 2171. [[CrossRef](#)]
- Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [[CrossRef](#)]
- Zhu, D.-D.; Sun, J.-Q. A New Algorithm Based on Dijkstra for Vehicle Path Planning Considering Intersection Attribute. *IEEE Access* **2021**, *9*, 19761–19775. [[CrossRef](#)]

18. Edmonds, N.; Breuer, A.; Gregor, D.; Lumsdaine, A. Single-Source Shortest Paths with the Parallel Boost Graph Library. *Shortest Path Problem* **2009**, *74*, 219–248. [[CrossRef](#)]
19. Floyd, R.W. Algorithm 97: Shortest path. *Commun. ACM* **1962**, *5*, 345.S. [[CrossRef](#)]
20. Wang, S.; Liu, B.; Liu, W.; Hu, C.; Tang, Y.; Yang, J. Research on the Shortest Path for Crossing Desert Based on Floyd Algorithm. In Proceedings of the 2021 IEEE 3rd International Conference on Frontiers Technology of Information and Computer (ICFTIC), Greenville, SC, USA, 12–14 November 2021.
21. Dreyfus, S.E. An Appraisal of Some Shortest-Path Algorithms. *Oper. Res.* **1969**, *17*, 395–412. [[CrossRef](#)]
22. Hart, P.E.; Nilsson, N.J.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [[CrossRef](#)]
23. Tang, G.; Tang, C.; Claramunt, C.; Hu, X.; Zhou, P. Geometric A-Star Algorithm: An Improved A-Star Algorithm for AGV Path Planning in a Port Environment. *IEEE Access* **2021**, *9*, 59196–59210. [[CrossRef](#)]
24. Otte, M.; Correll, N. C-Forest: Parallel Shortest Path Planning with Superlinear Speedup. *IEEE Trans. Robot.* **2013**, *29*, 798–806. [[CrossRef](#)]
25. Zhang, J.; Zhao, X.; He, X. A Minimum Resource Neural Network Framework for Solving Multiconstraint Shortest Path Problems. *IEEE Trans. Neural Netw. Learn. Syst.* **2014**, *25*, 1566–1582. [[CrossRef](#)] [[PubMed](#)]
26. Jan, G.E.; Sun, C.-C.; Tsai, W.C.; Lin, T.-H. An  $O(n \log n)$  Shortest Path Algorithm Based on Delaunay Triangulation. *IEEE/ASME Trans. Mechatron.* **2013**, *19*, 660–666. [[CrossRef](#)]
27. Lei, G.; Dou, Y.; Li, R.; Xia, F. An FPGA Implementation for Solving the Large Single-Source-Shortest-Path Problem. *IEEE Trans. Circuits Syst. II Express Briefs* **2015**, *63*, 473–477. [[CrossRef](#)]
28. Wang, Y.; Li, X.; Ruiz, R. An Exact Algorithm for the Shortest Path Problem with Position-Based Learning Effects. *IEEE Trans. Syst. Man Cybern. Syst.* **2016**, *47*, 3037–3049. [[CrossRef](#)]
29. Huang, W.; Zhang, Y.; Shang, Z.; Yu, J.X. To Meet or Not to Meet: Finding the Shortest Paths in Road Networks. *IEEE Trans. Knowl. Data Eng.* **2017**, *30*, 772–785. [[CrossRef](#)]
30. Li, Z.; Ji, L.; Huang, R.; Liu, S. Improving centralized path calculation based on graph compression. *China Commun.* **2018**, *15*, 120–124. [[CrossRef](#)]
31. Yang, L.; Li, D.; Tan, R. Research on the Shortest Path Solution Method of Interval Valued Neutrosophic Graphs Based on the Ant Colony Algorithm. *IEEE Access* **2020**, *8*, 88717–88728. [[CrossRef](#)]
32. Krauss, T.; McCollum, J. Solving the Network Shortest Path Problem on a Quantum Annealer. *IEEE Trans. Quantum Eng.* **2020**, *1*, 1–12. [[CrossRef](#)]
33. Liu, H.; Jin, C.; Yang, B.; Zhou, A. Finding Top-k Shortest Paths with Diversity. *IEEE Trans. Knowl. Data Eng.* **2017**, *30*, 488–502. [[CrossRef](#)]
34. Sidoti, D.; Avvari, G.V.; Mishra, M.; Zhang, L.; Nadella, B.K.; Peak, J.E.; Hansen, J.A.; Pattipati, K.R. A Multiobjective Path-Planning Algorithm with Time Windows for Asset Routing in a Dynamic Weather-Impacted Environment. *IEEE Trans. Syst. Man Cybern. Syst.* **2016**, *47*, 3256–3271. [[CrossRef](#)]
35. Meng, X.; Li, J.; Zhou, M.; Dai, X.; Dou, J. Population-Based Incremental Learning Algorithm for a Serial Colored Traveling Salesman Problem. *IEEE Trans. Syst. Man Cybern. Syst.* **2016**, *48*, 277–288. [[CrossRef](#)]
36. Xu, X.; Li, J.; Zhou, M. Delaunay-Triangulation-Based Variable Neighborhood Search to Solve Large-Scale General Colored Traveling Salesman Problems. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 1583–1593. [[CrossRef](#)]
37. Li, J.; Zhou, M.; Sun, Q.; Dai, X.; Yu, X. Colored Traveling Salesman Problem. *IEEE Trans. Cybern.* **2014**, *45*, 2390–2401. [[CrossRef](#)] [[PubMed](#)]
38. Xu, X.; Li, J.; Zhou, M. Bi-Objective Colored Traveling Salesman Problems. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 6326–6336. [[CrossRef](#)]
39. Blatnický, M.; Dižo, J.; Sága, M.; Gerlici, J.; Kuba, E. Design of a Mechanical Part of an Automated Platform for Oblique Manipulation. *Appl. Sci.* **2020**, *10*, 8467. [[CrossRef](#)]
40. Kuric, I.; Tlach, V.; Ságová, Z.; Císar, M.; Gritsuk, I. Measurement of industrial robot pose repeatability. *MATEC Web Conf.* **2018**, *244*, 01015. [[CrossRef](#)]
41. Božek, P.; Ivandić, Ž.; Lozhkin, A.; Lyalin, V.; Tarasov, V. Solutions to the characteristic equation for industrial robot's elliptic trajectories. *Teh. Vjesn. Tech. Gaz.* **2016**, *23*, 1017–1023. [[CrossRef](#)]
42. Chen, B.Y.; Lam, W.H.K.; Li, Q.; Sumalee, A.; Yan, K. Shortest Path Finding Problem in Stochastic Time-Dependent Road Networks with Stochastic First-In-First-Out Property. *IEEE Trans. Intell. Transp. Syst.* **2013**, *14*, 1907–1917. [[CrossRef](#)]
43. Feng, G.; Korkmaz, T. Finding Multi-Constrained Multiple Shortest Paths. *IEEE Trans. Comput.* **2014**, *64*, 2559–2572. [[CrossRef](#)]
44. Talebi, M.S.; Zou, Z.; Combes, R.; Proutiere, A.; Johansson, M. Stochastic Online Shortest Path Routing: The Value of Feedback. *IEEE Trans. Autom. Control* **2017**, *63*, 915–930. [[CrossRef](#)]
45. Chemoanov, D.; Esposito, F.; Calyam, P.; Sukhov, A. A Constrained Shortest Path Scheme for Virtual Network Service Management. *IEEE Trans. Netw. Serv. Manag.* **2018**, *16*, 127–142. [[CrossRef](#)]
46. Li, Y.; Zhang, H.; Zhu, H.; Li, J.; Yan, W.; Wu, Y. IBAS: Index Based A-Star. *IEEE Access* **2018**, *6*, 11707–11715. [[CrossRef](#)]
47. Ganganath, N.; Cheng, C.-T.; Fernando, T.; Iu, H.H.C.; Tse, C.K. Shortest Path Planning for Energy-Constrained Mobile Platforms Navigating on Uneven Terrains. *IEEE Trans. Ind. Inform.* **2018**, *14*, 4264–4272. [[CrossRef](#)]

48. Su, Z.; Zhang, J.; Lu, Z. A Multi-Stage Metaheuristic Algorithm for Shortest Simple Path Problem with Must-Pass Nodes. *IEEE Access* **2019**, *7*, 52142–52154. [[CrossRef](#)]
49. Kou, C.; Hu, D.; Yuan, J.; Ai, W. Bisection and Exact Algorithms Based on the Lagrangian Dual for a Single-Constrained Shortest Path Problem. *IEEE/ACM Trans. Netw.* **2019**, *28*, 224–233. [[CrossRef](#)]
50. Jie, K.W.; Zhao, G.C.; Sun, X.J. The Shortest Path Problem and Its Critical Edge in Uncertain Environment. *IEEE Access* **2019**, *7*, 154414–154423. [[CrossRef](#)]
51. Binh, H.T.T.; Thanh, P.D.; Thang, T.B. New approach to solving the clustered shortest-path tree problem based on reducing the search space of evolutionary algorithm. *Knowl. Based Syst.* **2019**, *180*, 12–25. [[CrossRef](#)]
52. Liu, X.-H.; Zhang, D.-G.; Yan, H.-R.; Cui, Y.-Y.; Chen, L. A New Algorithm of the Best Path Selection Based on Machine Learning. *IEEE Access* **2019**, *7*, 126913–126928. [[CrossRef](#)]
53. Xiong, J.; Qi, X.; Fu, Z.; Zha, W. Split Demand One-to-One Pickup and Delivery Problems with the Shortest-Path Transport Along Real-Life Paths. *IEEE Access* **2020**, *8*, 150539–150554. [[CrossRef](#)]
54. Thanh, P.D.; Binh, H.T.T.; Trung, T.B. An efficient strategy for using multifactorial optimization to solve the clustered shortest path tree problem. *Appl. Intell.* **2020**, *50*, 1233–1258. [[CrossRef](#)]
55. Cosma, O.; Pop, P.C.; Zelina, I. An Effective Genetic Algorithm for Solving the Clustered Shortest-Path Tree Problem. *IEEE Access* **2021**, *9*, 15570–15591. [[CrossRef](#)]
56. Guo, J.; Gao, K.; Wang, C.; Sang, H.; Li, J.; Duan, P. Discrete Jaya Algorithm for Solving Flexible Job Shop Rescheduling Problem. In Proceedings of the 2017 29th Chinese Control And Decision Conference (CCDC), Chongqing, China, 28–30 May 2017; pp. 6010–6015. [[CrossRef](#)]
57. Gao, K.; Sadollah, A.; Zhang, Y.; Su, R.; Li, K.G.J. Discrete Jaya Algorithm for Flexible Job Shop Scheduling Problem with New Job Insertion. In Proceedings of the 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand, 13–15 November 2016; pp. 1–5. [[CrossRef](#)]
58. Rao, R.V. Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems. *Int. J. Ind. Eng. Comput.* **2016**, *7*, 19–34. [[CrossRef](#)]
59. Motamarri, R.; Bhookya, N. JAYA Algorithm Based on Lévy Flight for Global MPPT Under Partial Shading in Photovoltaic System. *IEEE J. Emerg. Sel. Top. Power Electron.* **2020**, *9*, 4979–4991. [[CrossRef](#)]
60. Mujeeb, S.; Javaid, N.; Ahmed, A.; Gulfam, S.M.; Qasim, U.; Shafiq, M.; Choi, J.-G. Electricity Theft Detection with Automatic Labeling and Enhanced RUSBoost Classification Using Differential Evolution and Jaya Algorithm. *IEEE Access* **2021**, *9*, 128521–128539. [[CrossRef](#)]
61. Elgebaly, A.E.; Taha, I.B.M.; Azmy, A.M.; El-Ghany, H.A.A. Optimal Design and Control of SSSCs for TLs Considering Technical and Economic Indices Using GA and SAMPE-JAYA Algorithms. *IEEE Access* **2021**, *9*, 38907–38919. [[CrossRef](#)]
62. Le, T.-L.; Quynh, N.V.; Long, N.K.; Hong, S.K. Multilayer Interval Type-2 Fuzzy Controller Design for Quadcopter Unmanned Aerial Vehicles Using Jaya Algorithm. *IEEE Access* **2020**, *8*, 181246–181257. [[CrossRef](#)]
63. Zhang, X.; Mu, R.; Chen, K.; Yang, Y.; Chen, Y. Intelligent Hough Transform with Jaya to Detect the Diameter of Red-Hot Circular Workpiece. *IEEE Sens. J.* **2020**, *21*, 560–567. [[CrossRef](#)]
64. Gao, K.; Yang, F.; Zhou, M.; Pan, Q.; Suganthan, P.N. Flexible Job-Shop Rescheduling for New Job Insertion by Using Discrete Jaya Algorithm. *IEEE Trans. Cybern.* **2018**, *49*, 1944–1955. [[CrossRef](#)]
65. Gao, K.; Zhang, Y.; Su, R.; Yang, F.; Suganthan, P.N.; Zhou, M. Solving Traffic Signal Scheduling Problems in Heterogeneous Traffic Network by Using Meta-Heuristics. *IEEE Trans. Intell. Transp. Syst.* **2018**, *20*, 3272–3282. [[CrossRef](#)]
66. Xiang, X.; Tian, Y.; Zhang, X.; Xiao, J.; Jin, Y. A Pairwise Proximity Learning-Based Ant Colony Algorithm for Dynamic Vehicle Routing Problems. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 5275–5286. [[CrossRef](#)]
67. Luan, W.; Liu, G.; Jiang, C.; Zhou, M. MPTR: A Maximal-Marginal-Relevance-Based Personalized Trip Recommendation Method. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 3461–3474. [[CrossRef](#)]
68. Goldberg, A.V.; Johnson, D.S.; Demetrescu, C. (Eds.) *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, ser; DIMACS Book; AMS: Providence, RI, USA, 2009; Volume 74.
69. Liu, H.; Chatterjee, I.; Zhou, M.; Lu, X.S.; Abusorrah, A. Aspect-Based Sentiment Analysis: A Survey of Deep Learning Methods. *IEEE Trans. Comput. Soc. Syst.* **2020**, *7*, 1358–1375. [[CrossRef](#)]
70. Zhang, P.; Huang, W.; Chen, Y.; Zhou, M.; Al-Turki, Y. A Novel Deep-Learning-Based QoS Prediction Model for Service Recommendation Utilizing Multi-Stage Multi-Scale Feature Fusion with Individual Evaluations. *IEEE Trans. Autom. Sci. Eng.* **2023**, 1–14. [[CrossRef](#)]
71. Zhang, Z.; Liu, H.; Zhou, M.; Wang, J. Solving Dynamic Traveling Salesman Problems with Deep Reinforcement Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *34*, 2119–2132. [[CrossRef](#)]
72. Liu, G.; Zhang, R.; Wang, Y.; Man, R. Road Scene Recognition of Forklift AGV Equipment Based on Deep Learning. *Processes* **2021**, *9*, 1955. [[CrossRef](#)]
73. Huang, X.; Zeng, X.; Wu, Q.; Lu, Y.; Huang, X.; Zheng, H. Face Verification Based on Deep Learning for Person Tracking in Hazardous Goods Factories. *Processes* **2022**, *10*, 380. [[CrossRef](#)]
74. Liu, Z.; Wei, H.; Wang, H.; Li, H.; Wang, H. Integrated Task Allocation and Path Coordination for Large-Scale Robot Networks with Uncertainties. *IEEE Trans. Autom. Sci. Eng.* **2021**, *19*, 2750–2761. [[CrossRef](#)]

75. Li, C.; He, W.; Yao, H.; Mai, T.; Wang, J.; Guo, S. Knowledge Graph Aided Network Representation and Routing Algorithm for LEO Satellite Networks. *IEEE Trans. Veh. Technol.* **2022**, *72*, 5195–5207. [[CrossRef](#)]
76. Zhang, C.; Zhang, S.; Zou, X.; Yu, S.; Yu, J.J.Q. Toward Large-Scale Graph-Based Traffic Forecasting: A Data-Driven Network Partitioning Approach. *IEEE Internet Things J.* **2022**, *10*, 4506–4519. [[CrossRef](#)]
77. Liu, Z.; Zuo, X.; Zhou, M.; Guan, W.; Al-Turki, Y. Electric Vehicle Routing Problem with Variable Vehicle Speed and Soft Time Windows for Perishable Product Delivery. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 1–13. [[CrossRef](#)]
78. Wang, K.; An, J.; Zhou, M.; Shi, Z.; Shi, X.; Kang, Q. Minority-Weighted Graph Neural Network for Imbalanced Node Classification in Social Networks of Internet of People. *IEEE Internet Things J.* **2022**, *10*, 330–340. [[CrossRef](#)]
79. Zhao, Q.; Li, G.; Cai, J.; Zhou, M.; Feng, L. A Tutorial on Internet of Behaviors: Concept, Architecture, Technology, Applications, and Challenges. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 1227–1260. [[CrossRef](#)]
80. Zhao, Z.; Yang, Z.; Li, C.; Zeng, Q.; Guan, W.; Zhou, M. Dual Feature Interaction-Based Graph Convolutional Network. *IEEE Trans. Knowl. Data Eng.* **2022**, *35*, 9019–9030. [[CrossRef](#)]
81. Zhou, Y.; Kou, Y.; Zhou, M. Bilevel Memetic Search Approach to the Soft-Clustered Vehicle Routing Problem. *Transp. Sci.* **2023**, *57*, 701–716. [[CrossRef](#)]
82. Zhou, Y.; Wang, G.; Zhou, M. Detecting  $k$ -Vertex Cuts in Sparse Networks via a Fast Local Search Approach. *IEEE Trans. Comput. Soc. Syst.* **2023**, 1–10. [[CrossRef](#)]
83. Zhou, Y.; Xu, W.; Zhou, M.; Fu, Z.H. Bi-Trajectory Hybrid Search to Solve Bottleneck-Minimized Colored Traveling Salesman Problems. *IEEE Trans. Autom. Sci. Eng.* **2023**, 1–11. [[CrossRef](#)]
84. Zhou, Y.; Xu, W.; Fu, Z.H.; Zhou, M. Multi-Neighborhood Simulated Annealing-Based Iterated Local Search for Colored Traveling Salesman Problems. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 16072–16082. [[CrossRef](#)]
85. Wu, N.; Zhou, M. Modeling and deadlock avoidance of automated manufacturing systems with multiple automated guided vehicles. *IEEE Trans. Syst. Man Cybern. Part B* **2005**, *35*, 1193–1202. [[CrossRef](#)]
86. Zhen, L.; Li, H. A literature review of smart warehouse operations management. *Front. Eng. Manag.* **2022**, *9*, 31–55. [[CrossRef](#)]
87. Claussmann, L.; Revilloud, M.; Gruyer, D.; Glaser, S. A Review of Motion Planning for Highway Autonomous Driving. *IEEE Trans. Intell. Transp. Syst.* **2019**, *21*, 1826–1848. [[CrossRef](#)]
88. Huang, B.; Zhou, M.; Lu, X.S.; Abusorrah, A. Scheduling of Resource Allocation Systems with Timed Petri Nets: A Survey. *ACM Comput. Surv.* **2023**, *55*, 230. [[CrossRef](#)]
89. Yuan, H.; Zhou, M. Profit-Maximized Collaborative Computation Offloading and Resource Allocation in Distributed Cloud and Edge Computing Systems. *IEEE Trans. Autom. Sci. Eng.* **2020**, *18*, 1277–1287. [[CrossRef](#)]
90. Zhang, J.; Lu, Y.; Wu, Y.; Wang, C.; Zang, D.; Abusorrah, A.; Zhou, M. PSO-Based Sparse Source Location in Large-Scale Environments with a UAV Swarm. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 5249–5258. [[CrossRef](#)]
91. Han, S.; Zhu, K.; Zhou, M.; Liu, X. Joint Deployment Optimization and Flight Trajectory Planning for UAV Assisted IoT Data Collection: A Bilevel Optimization Approach. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 21492–21504. [[CrossRef](#)]
92. Huang, L.; Zhou, M.; Hao, K.; Han, H. Multirobot Cooperative Patrolling Strategy for Moving Objects. *IEEE Trans. Syst. Man Cybern. Syst.* **2022**, *53*, 2995–3007. [[CrossRef](#)]
93. Jin, L.; Liang, S.; Luo, X.; Zhou, M. Distributed and Time-Delayed-Winner-Take-All Network for Competitive Coordination of Multiple Robots. *IEEE Trans. Cybern.* **2023**, *53*, 641–652. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.