Hindawi
Mobile Information Systems
Volume 2021, Article ID 2963092, 20 pages
https://doi.org/10.1155/2021/2963092

Research Article

The Path Optimization Algorithm of Car Navigation System considering Node Attributes under Time-Invariant Network

Dan-dan Zhu and Jun-qing Sun

Tianjin Key Laboratory of Intelligence Computing and Novel Software Technology, Tianjin University of Technology, Tianjin 300384, China

Correspondence should be addressed to Dan-dan Zhu; ddzhu_0703@163.com

Received 10 January 2020; Revised 10 December 2020; Accepted 18 December 2020; Published 4 January 2021

Academic Editor: Nicola Bicocchi

Vehicle path planning plays a key role in the car navigation system. In actual urban traffic, the time spent at intersections accounts for a large proportion of the total time and cannot be ignored. Therefore, studying the shortest path planning problem considering node attributes has important practical significance. In this article, we study the vehicle path planning problem in time-invariant networks, with the minimum travel time from the starting node to the destination node as the optimization goal (including node time cost). Based on the characteristics of the problem, we construct the mathematical model. We propose a Reverse Order Labeling Algorithm (ROLA) based on the traditional Dijkstra algorithm to solve the problem; the correctness of the proposed algorithm is proved theoretically, and we analyse and give the time complexity of the ROLA and design a calculation example to verify the effectiveness of the algorithm. Finally, through extensive simulation experiments, we compare the performance of the proposed ROLA with several other existing algorithms. The experimental results show that the proposed algorithm has good stability and high efficiency.

1. Introduction

With rapid economic and social development and accelerated urbanization, exhaust pollution, urban congestion, and an increasing number of traffic accidents have become serious urban problems, especially in big cities. Therefore, in order to increase road traffic efficiency, it is necessary to conduct research on the intelligent transportation system (ITS). ITS refers to a comprehensive transportation system formed by integrating modern information technology, sensor technology communication technology, and other high-tech on the basis of a more complete traditional transportation management systems [1]. Its biggest feature is the integration of intelligent systems with the traditional transportation industry, the use of modern communication technology, and intelligent analysis methods such as big data, cloud computing, and artificial intelligence to realize the real-time perception of existing traffic conditions, synchronize optimization, and adjustments. The current situation of urban road congestion is improved, the efficiency of urban road use is maximized, and people's travel safety is ensured [2, 3].

The United States began to focus on the research of Intelligent Transportation Systems in the 1960s [4]. In the following decades of development, the United States created the electronic route guidance system and carried out road system research. In 1991, the Intelligent Transportation Systems Association of America formally proposed the concept of Intelligent Transportation Systems (ITS). Subsequently, on the basis of the vehicle-road integration project, an in-depth study of the vehicle-road collaborative control was carried out [5]. The development of ITS in Japan began in the 1970s, during which a dynamic route guidance system experiment was carried out. Japan established the

Japan Road Traffic Vehicle Intelligent Promotion Association in 1994 to promote ITS. In recent years, Japan has mainly carried out research on vehicle-road coordination, strengthened the application of wireless communication technology in the field of ITS, and carried out research on collaborative safety control technology for traffic objects [6]. Europe began to study road traffic communication technology in the 1970s. In the mid-to-late 1980s, Europe carried out research on the vehicle safety road structure plan and the efficient and safe transportation system plan. The European Road Traffic Communication Technology Application Promotion Organization was established in 1991 to promote cooperation between the European Union and related companies to jointly promote the development of ITS in Europe [7].

Dynamic route guidance system (DRGS), as an indispensable part of the ITS, is one of the more direct and effective means to improve urban traffic conditions [8]. The system uses the global positioning system (GPS), computers, electronic traffic maps, advanced communication technologies to obtain a large number of timely, and accurate traffic network data [9]. Then, the optimal driving path is given between the two fixed nodes in the access network is submitted to the user and displayed by the on-board equipment. The characteristic of this system is based on real-time traffic data, fully taking into account the three elements of traffic, people, vehicles, and roads [10]. While maximizing the "benefits" of drivers, it finally realizes the reasonable distribution of traffic flow in the urban traffic road network. The key to DRGS is the optimal path selection problem, and it is also one of the hot topics currently studied [11].

In the research of path planning model and optimization strategy, the model is mainly constructed with the path geometric mileage or travel time as the target value. Many methods have been proposed and applied in this field. According to the different principles of solving the shortest path, the algorithm is divided into the traditional graph theory, shortest path algorithm and intelligent optimization algorithm [12]. The former includes Dijkstra [13, 14], A* algorithm [15], Floyd algorithm [16], and hierarchical heuristic search algorithm; the intelligent optimization algorithms include simulated annealing algorithm (SA), particle swarm optimization algorithm (PSO), neural network algorithm (NNA), ant colony optimization (ACO), genetic algorithm (GA), and machine learning algorithm. In the next part, we will briefly review the existing path optimization algorithms.

2. Related Works

The shortest path problem (SP) is the core content of graph theory and network optimization research. Many real-world problems can be converted into SP problems [4]. The effective combination of classic graph theory and continuously developed computer data structures and algorithms has resulted in the emergence of new optimal path algorithms [17]. The most commonly used traditional shortest path algorithms are Dijkstra algorithm and A* algorithm. In 1959, E. W. Dijkstra first proposed an algorithm for solving the shortest path between two points in the weighted graph, namely, the Dijkstra algorithm, which can also be used to solve the shortest distance from a specified vertex to the remaining vertices in the graph path. The complexity of the algorithm can be expressed as $O(n^2)$ [18]. In order to solve the problem of automated guided vehicle (AGV) access path planning in the smart garage and overcome the shortcomings of the traditional Dijkstra algorithm such as high time complexity, large search range, and low search efficiency, Liu et al. proposed an improved Dijkstra algorithm [19]. Hart et al. proposed the A* algorithm in 1968, and the literature [20] also gave the main idea of the A* algorithm. This algorithm is widely used in artificial intelligence to complete the search on the map and is widely used in the game scene to find the shortest path between two points. The core idea of the algorithm is to evaluate the quality of each branch in the search process through the function value and then to select the best branch to search. Song and Wang combined particle swarm algorithm and A* algorithm to solve the robot path planning problem [21].

Intelligent algorithms to solve path planning problems have also been extensively studied by scholars. Liu et al. studied the path optimization problem of intelligent driving vehicles and combined the prior reinforcement learning technology with the A* algorithm to search for the shortest path [4]. Meng et al. studied the serial color traveling salesman path planning problem (SCTSP) and proposed a population-based incremental learning algorithm with a 2-opt local search [22]. Xu et al. studied the path optimization problem of the general colored traveling salesman problem (GCTSP) and proposed a Delaunay-triangulation-based variable neighborhood search algorithm to solve the problem [23]. Ge et al. studied the ant colony algorithm and its application in path planning. Aiming at the problem of common ant colony optimization that is easy to fall into a dead end, they proposed an improved ant colony algorithm with the return; according to the characteristics of the urban road network, design A* dynamic limited area search strategy; finally, combined with the A* algorithm, a hierarchical ant colony algorithm based on dynamic area planning is proposed to realize the complementary advantages of the two algorithms [24]. In order to achieve an efficient solution to the vehicle path optimization problem with capacity constraints, Shang et al. proposed an improved simulated annealing algorithm with tempering operation. The characteristics of path optimization under multiple constraints are analyzed, and a simulated annealing framework with simple structure and relatively independent functional modules is constructed, which can facilitate the coupling and nesting of related constraints and their algorithms. On this basis, the acceptance rule of the better solution in the iterative process is changed, and the tempering operation is introduced to achieve a balance between global search and local search; a mandatory random neighborhood transformation strategy is designed to improve quality of new solutions under multiple constraints [25]. Mohammed et al. solved vehicle routing planning problems by using the improved genetic algorithm for optimal solutions [26]. Zhi proposed an improved genetic algorithm and particle swarm

algorithm to solve the optimization problem of vehicle path planning with constraints [27]. Kao et al. proposed a hybrid algorithm based on ACO and PSO for capacitated vehicle routing problems [28]. Ma et al. proposed an improved PSO for simultaneous pickup and delivery vehicle routing problems with time windows under a fuzzy random environment [29]. Zhang et al. proposed an evolutionary scatter search PSO algorithm (ESS-PSO) to solve the vehicle routing problem with time windows (VRPTW) with the objective of minimizing the total travel distance [30]. Wang and Lu proposed a competition-based memetic algorithm (MAC) to solve the capable green vehicle routing problem (CGVRP) [31].

In the route planning of the existing driving navigation system, generally, only the travel distance and time spent on the road section of the vehicle are considered, and the intersection is regarded as a node. The waiting time of the vehicle at the intersection and the time cost of passing the intersection are generally considered either it is ignored or it is attributed to the corresponding weight of the road segment connecting the intersection. However, in urban traffic, because the road section physical distance between two intersections is relatively short, the time for vehicles to pass the road is short. And because of the influence of traffic lights and traffic regulations on the speed of vehicles passing through the intersection, the time spent by the vehicle at the intersection is not negligible relative to the time spent on the road segment. When vehicles go straight, turn left, right, or U-turn through the intersection, the waiting and passing time consumption is also different. So it is impossible to attribute the time spent waiting at the intersection to the weight of the road arc. Therefore, in the route planning of urban traffic, not only the travel time of the driving vehicle on the road but also the time consumed by the vehicle waiting and passing through the intersection must be considered [32].

In this paper, we study the minimum-time path planning problem taking into account the node attribute (weight or time cost) in a deterministic FIFO network. We describe the problem in Section 3, establish a mathematical model in Section 4, and present our optimization algorithm in Section 5. Extensive simulation comparison experiments are carried out on the proposed algorithm with the other four common algorithms in Section 6. Our conclusions are summarized in Section 7.

3. Problem Description

3.1. Problem Definition. The focus of this study is the path-planning problem in a time-invariant deterministic network, taking into account the intersection attribute (time cost). In such a network, we assumed that the travel time along an arc and the waiting time at a node that have different subsequent arcs do not vary with time. We use an abstract road network diagram to illustrate the time consumption of vehicles in the actual road network. As shown in Figure 1, the node represents the intersection, the edge represents the road segment, A is the starting point, and G is the destination. The numbers on the edges represent the travel time of the vehicles on the road represented by this edge. The numbers on the nodes, respectively, represent the waiting time for the vehicle to choose the next different edge at the intersection. If only considering the edge and not the time consumption at the node, the shortest travel time path from point A to point GG is $A \longrightarrow B \longrightarrow E \longrightarrow G$; the time cost is $1 + 2 + 1 = 4$. When the waiting time of the node (intersection) is taken into consideration, in fact, the time cost for the vehicle to walk on the path $A \longrightarrow B \longrightarrow E \longrightarrow G$ is $1 + 3 + 2 + 3 + 1 = 101 + 3 + 2 + 3 + 1 = 10$; but when the vehicle travels on the path $A \longrightarrow C \longrightarrow F \longrightarrow G$, the time cost is $2 + 1 + 1 + 1 + 2 = 72 + 1 + 1 + 1 + 2 = 7$; so in fact, the shortest travel time path when considering the waiting time at the intersection is $A \longrightarrow C \longrightarrow F \longrightarrow G$, which is more in line with the preference of the driver.

3.2. Problem Hypothesis. The directed graph studied in this paper is a time-invariant deterministic "first-in, first-out" (FIFO) network. Before proceeding further, we make the following assumptions:

(1) A vehicle travels on a road network in a certain area of a city at a certain time period. During this time period, the time required for a vehicle to drive on any section of the road network does not depend on the time of departure.

(2) The time required for vehicles to pass through the intersection by turning left, right, straight, or U-turn at any intersection during this period is different, but it does not depend on the time of arrival at the intersection.

(3) The time required for a vehicle to travel on any road section during this period is a fixed constant.

(4) The waiting time and the time required for a vehicle to pass through the intersection in any way (turn left, turn right, go straight, or U-turn) at any intersection is also a fixed constant.

4. Formulation and Analysis of the Mathematical Model

4.1. Formulation of the Mathematical Model. According to the above assumptions, let the weighted directed graph $G = (V, E, \Theta, \Psi)$ composed of four-tuples be the road network graph. The notations and their meanings used in this article are shown in Table 1.

Given a destination node D in the network, the path from any other node S to D can be defined as $\{i_0, i_1, i_2, \ldots, i_{n-1}, i_n\}$, where $i_0 = S$ and $i_n = D$. To find the shortest path, we minimize the sum of the attribute values from S to D (the sum of the times spent on each road section and the times spent waiting at and passing through intersections). This is what we refer to as the shortest-path problem for a time-invariant deterministic network taking into account node attributes (weight or time cost). To facilitate the solution of this problem, the following assumptions are necessary.

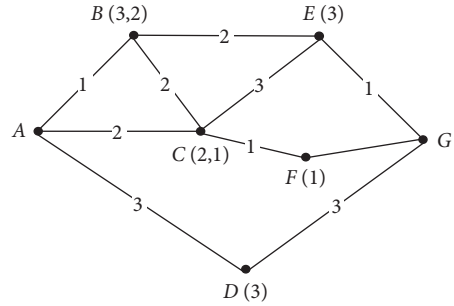

Figure 1: Network graph considering node cost.

Table 1

Notation	Meaning
(i, j)	The arc that connecting node i and node j.
$\phi(i, j)$	The attribute value of the arc segment (i, j), i.e., the time required for the vehicle to traverse the arc segment (i, j). For each arc segment (i, j), $\phi(i, j)$ is a uniquely determined value.
$\theta(i, j, k)$	The attribute value of node j (weight or time cost) represents the time taken for the vehicle to wait at node j (intersection) when the vehicle travels from the road section (i, j) to the road section (j, k). $\theta(i, j, k)$ is also uniquely determined for each node j.
V	The set of nodes (representing a set of intersections in an actual road network).
E	The set of directed arcs that connect nodes (representing road sections in a road network).
$N^+(i)$	The successor nodes set of node $i \in V$, $N^+(i) = \{j \mid (i, j) \in E\}$.
$N^-(i)$	The predecessor nodes set of node i, $N^-(i) = \{j \mid (i, j)t \in nE\}$.
Ψ	The set of each arc attribute values. $\Psi = \{\phi(i, j) \mid i, j \in V, (i, j) \in E\}$.
Θ	The set of each node attribute values in the network. $\Theta = \{\theta(i, j, k) \mid j \in V, i \in N^-(j), \text{and } k \in N^+(j)\}$
$\lambda(S, D)$	The minimum travel time starting from the starting node S to the destination node D.
$\lambda(i, j, k)$	The minimum travel time starting from the first node i of the road section (i, j) and passing through this section, then through intersection j, and the section (j, k) to the destination node D.
$\mu(i, j)$	The minimum travel time from the first node i i of the road section (i, j) passing through this section and through intersection j to the destination node D.

(1) When the vehicle travels along the route $\{i_0, i_1, i_2, \ldots, i_{n-1}, i_n\}$, if it starts from or arrives at node i_k, means that the vehicle at the starting point of the arc (i.e., road section) (i_k, i_{k+1}) at this time.

(2) When the vehicle reaches the endpoint D, we assume that it passes the road section (i_{n-1}, i_n) through the intersection $i_n = D$ reaches the starting point of the road section (i_n, i_{n+1}), where $i_{n+1} = D$, $(i_n, i_{n+1}) = (D, D)$ is a virtual road section, and $\theta(i_{n-1}, D, D) = 0$.

According to the above description, the mathematical model of the shortest-path problem for a time-invariant deterministic network taking into account node time consumption can be defined as

$$\lambda(S, D) = \min_{(i_0, i_1, i_2, \cdots, i_{n-1}, i_n)} \sum_{j=1}^{n} \left[\phi\left(i_{j-1}, i_j\right) + \theta\left(i_{j-1}, i_j, i_{j+1}\right)\right]. \tag{1}$$

4.2. Analysis of the Mathematical Model. Through the analysis of the problem, it is easy to get the following two theorems.

Theorem 1. *Assume that the path $(i_0, i_1, i_2, i_3, \ldots, i_{n-1}, i_n)$ is the optimal solution of the shortest-path problem taking into account node attributes in a time-invariant deterministic network that connects a starting point $i_0 = S$ and an endpoint $i_n = D$. Then, an arbitrary subpath $(i_{j+1}, i_{j+2}, \ldots, i_{j+k-1}, i_{j+k})$ of the optimal path $(i_0, i_1, i_2, i_3, \ldots, i_{n-1}, i_n)$ is also the optimal solution from the starting point i_{j+1} to the endpoint i_{j+k} (i.e., the starting point of the road section (i_{j+k}, i_{j+k+1})). That is, the optimal solution of the shortest-path problem for a time-invariant deterministic network taking into account node attributes has an optimal substructure [33].*

Proof. The theorem can be proved by a cut-and-paste method. Assume that the subpath $(i_{j+1}, i_{j+2}, i_{j+3}, \ldots, i_{j+r-1}, i_{j+r})$ is not the optimal solution of the shortest-path problem for a time-invariant deterministic network taking into account node attributes, starting at i_{j+1}(at the beginning of road section (i_{j+1}, i_{j+2})) and ending at i_{j+r} (at the beginning of the road section (i_{j+r}, i_{j+r+1})). Then, there must exist another subpath $(i_{j+1}, i_{j+2}, i_{j+3}', \ldots, i_{j+s-1}', i_{j+s}')$ where i_{j+s}' represents the starting point of the road section (i_{j+s}', i_{j+s+1}), such that

$$\alpha\left(i_{j+1}, i_{j+2}, i_{j+3}', i_{j+4}', \ldots, i_{j+s}'\right) < \alpha\left(i_{j+1}, i_{j+2}, i_{j+3}, \ldots, i_{j+r}\right). \tag{2}$$

Here,

$$\alpha\left(i_{j+1}, i_{j+2}, i'_{j+3}, i'_{j+4}, \ldots, i'_{j+s}\right) = \sum_{q=1}^{s-1}\left[\phi\left(i'_{j+q}, i_{j+q+1}{}'\right) + \theta\left(i'_{j+q}, i'_{j+q+1}, i'_{j+q+2}\right)\right], \tag{3}$$

where node $i_{j+1}{}'$ represents the starting point of the arc segment (i_{j+1}, i_{j+2}), node $i_{j+2}{}'$ represents the starting point of the arc segment (i_{j+2}, i_{j+3}), node $i_{j+s}{}'$ represents the starting point of the arc segment $(i_{j+s}{}', i_{j+r+1})$, and node $i_{j+s+1}{}'$ indicates the starting point of the arc segment (i_{j+r+1}, i_{j+r+2}); and

$$\alpha\left(i_{j+1}, i_{j+2}, i_{j+3}, \ldots, i_{j+r}\right) = \sum_{q=1}^{r-1}\left[\phi\left(i_{j+q}, i_{j+q+1}\right) + \theta\left(i_{j+q}, i_{j+q+1}, i_{j+q+2}\right)\right], \tag{4}$$

where node i_{j+1} represents the starting point of the arc segment (i_{j+1}, i_{j+2}) and node point i_{j+r} the starting point of the arc segment (i_{j+r}, i_{j+r+1}). Therefore, we have

$$\begin{aligned}\alpha\left(i_0, i_0, i_0, \ldots, i_j\right) + \alpha\left(i_{j+1}, i_{j+2}, i'_{j+3}, i'_{j+4}, \ldots, i'_{j+s}\right) + \alpha\left(i_{j+r+1}, i_{j+r+2}, \ldots, i_n\right) < \alpha\left(i_0, i_0, i_0, \ldots, i_j\right) + \alpha\left(i_{j+1}, i_{j+2}, i_{j+3}, i_{j+4}, \ldots, i_{j+r}\right) \\ + \alpha\left(i_{j+r+1}, i_{j+r+2}, \ldots, i_n\right).\end{aligned} \tag{5}$$

This contradicts the assumption that the path $(i_0, i_1, i_2, i_3, \ldots, i_{n-1}, i_n)$ is the optimal solution of the shortest-path problem from the starting node $i_0 = S$ to the destination node $i_n = D$ for the time-invariant deterministic network taking into account node attributes. Therefore, the assumption is not true, and Theorem 1 is proved. □

Theorem 2. *The shortest-path problem for a time-invariant deterministic network taking into account node attributes has the form of repeating subproblems that can be solved by a recursive algorithm.*

5. Algorithm Design

On the basis of the above analysis, we adopted the dynamic programming method to design an optimal algorithm for solving the shortest-path problem taking into account node attributes.

5.1. Reverse Order Labeling Algorithm for Solving Time-Invariant Network Problems

5.1.1. Basic Formula. Let $\lambda(i, j, k)$ denote the minimum travel time starting from the first node i of the road section (i, j) and passing through this section, then through intersection j and the section (j, k) to the endpoint D. Then, the following recursion formula holds

$$\lambda(i, j, k) = \min_{l \in N^+(k)}\{\phi(i, j) + \theta(i, j, k) + \lambda(j, k, l)\}, \tag{6}$$

where

$$\lambda(i, D, D) = \phi(i, D), \quad \forall i \in N^-(D). \tag{7}$$

From the results in the preceding subsection, it follows that

$$\lambda(i, D) = \min_{j \in N^+(i), k \in N^+(j)}\{\lambda(i, j, k)\}. \tag{8}$$

Let $\mu(i, j)$ denote the minimum travel time from the first node i of the road section (i, j) passing through this section (i, j) and through intersection j to the destination D. Then,

$$\mu(i, j) = \min_{k \in N^+(j)}\{\lambda(i, j, k)\}, \tag{9}$$

$\mu(i, j)$ can also be calculated from the recursion formula as follows:

$$\mu(i, j) = \min_{l \in N^+(j)}\{\phi(i, j) + \theta(i, j, l) + \mu(j, l)\}, \tag{10}$$

where

$$\begin{aligned}&\phi(i, j) \in \Psi,\\ &\theta(i, j, l) \in \Theta,\\ &j \in N^+(i),\\ &l \in N^+(j),\end{aligned} \tag{11}$$

$$\mu(i, D) = \phi(i, D), \quad \forall i \in N^-(D). \tag{12}$$

We record $(j, \pi_\mu(i, j))$ as the road section following the section (i, j) on the minimum-travel-time path via (i, j) to the destination D, where

$$\pi_\mu(i, j) = \arg\min_{l \in N^+(j)}\{\phi(i, j) + \theta(i, j, l) + \mu(j, l)\}. \tag{13}$$

We denote the minimum travel time from the starting point SS to the destination D by $\varphi(S, D)$, and we record $\pi_\varphi(i)$ as the successor node of node i for this path. From the above analysis, it follows that

$$\varphi(S, D) = \arg\min_{j \in N^+(S)}\{\mu(S, j)\}, \tag{14}$$

$$\pi_\varphi(i) = \arg\min_{j \in N^+(i)}\{\mu(i, j)\}. \tag{15}$$

5.1.2. Overview of the Algorithm. To solve the problem in this article, a new improved Dijkstra algorithm was proposed, named Reverse Order Labeling Algorithm (ROLA). It gradually explores the minimum-travel-time paths through one (or several) arcs (i, j) starting from a certain node i (or some nodes) to the destination D. During the execution of the algorithm, a value $\mu(i, j)$ is calculated for the minimum travel time through a given arc (i, j) to the endpoint D, and the arc (i, j) is labeled with a B (black arc). Then, the minimum travel time through all the arcs (k, i) ending at node i to the destination D is calculated. In this process, if any arc that starts at node i has a label B (i.e., the minimum travel time from the arc to the endpoint is known); then, the minimum travel time from the arc (k, i) to the destination D is calculated directly from the recursion formula (5). If an arc starting at node i does not have a label B, then node i is taken as the tree root and arc segments without label B are taken as the search objects. According to the breadth-first search principle, all the arc segments without label B that start from or are connected with node i are searched for, and a breadth-first search tree is constructed. Any arc segment in the tree is labeled with R (red arc segment); then, the minimum travel time through any arc in the tree to the destination D is calculated and the arcs with label R in the tree are relabeled with B. Finally, the minimum travel time to the destination D via the arc (k, i) is calculated. As can be seen from this description, each step of the method modifies the labels of one or more arc segments, with some arc segments without label B acquiring such a label so that the number of arc segments with label B in the network G increases by at least one. Thus, after at most $|V|$ iterations, the minimum-travel-time path through any arc segment starting with each node to the destination can be found. The minimum-travel-time path from the start point S to the destination D is also obtained. In each iteration of the algorithm, the set of arc segments of the directed network graph taking into account node attributes is divided into three subsets: a set of arcs with label B (E_B), a set of arcs with label R (E_R), and the set $E_P = E - (E_B \cup E_R)$ (i.e., the complement of $(E_B \cup E_R)$, a collection of purple arcs). Since it is performed in reverse order, a feature of the reverse order labeling algorithm is that when calculating the shortest path from the current arc segment to the endpoint, it can make full use of the obtained shortest-path data from the subsequent arc segment to the endpoint, which not only exploits all available resources but also saves calculation time. The execution of the algorithm is illustrated below with a specific example.

We intercept a section of the network to illustrate two situations that occur during the calculation of the reverse order labeling method, as shown in Figures 2 and 3, where the black solid arc (i, j) indicates that the minimum travel time $\mu(i, j)$ from the arc (i, j) to the endpoint has been found, i.e., the arc (i, j) has label B. Assuming that the current scanning node is node 2, it is necessary to compute the minimum travel time $\mu(i, j)$ from the arc (1, 2) (the purple dashed line) to the endpoint. In Figure 2, since the arc segments $(2, j)(\forall j \in N^+(2))$ are already arcs with label B, they can be calculated directly using the recursion formula (5). In Figure 3, when calculating the minimum travel time $\mu(1,2)$ of the arc (1, 2) to the endpoint, because the arcs (2, 3) and (2, 4) starting with node 2 are not labeled B, we need to establish a breadth-first search tree taking node 2 as the root node. Then, all the arcs without label B (the purple dashed lines) that start from or are connected with node 2 are placed on the search tree and labeled R label (red dash dotted lines), as shown in Figure 4. The minimum travel times $\mu(2,3)$, $\mu(2,4)$, and $\mu(3,4)$ of the arc segments (2, 3), (2, 4), and (3, 4) to the endpoint in the tree are calculated, and finally the minimum travel time $\mu(1,2)$ from the arc (1, 2) to the endpoint is calculated.

5.1.3. Specific Steps of the Algorithm. Given a directed graph $G = (V, E, \Theta, \Psi)$, node set V, edge set E, node attribute value set Θ, edge attribute value set Ψ, and endpoint D as follows:

Step 0. BEGIN. Let $E_B = \emptyset$, $E_R = \emptyset$, $V_R = \{v | v \in D\}$; for any $i \in V$, let $N_n^+(i) = N^+(i)$, $N_n^-(i) = N^-(i)$, $n = 1$.

Step 1. Check whether $E_B = E$ holds. If it does, then go to Step 5; otherwise, go to Step 2.

Step 2. Take one $v \in V_R$, and check whether $N_n^+(v) = \emptyset$ holds. (1) If it does, then go to Step 3. (2) If it does not hold, i.e., there exists at least one $j \in N^+(v)$, make $(v, j) \notin E_B$ (this means that the arc (v, j) does not have label B); then follow the steps below:

Step 2.1. Taking the node vv as the tree root on the directed graph $G = (V, E, \Theta, \Psi)$, take the arc segment without label B as the search object, construct a breadth-first search tree according to the breadth-first search principle, and select all arc segments (i, l) such that $l \in N_n^+(i)$ and (i, l) is not in the breadth-first search tree (i.e., (i, l) has neither label B nor label R); insert (i, l) into the breadth-first search tree and into the stack E_R(i.e., label the arc segment (i, l) with R), and delete l from $l \in N_n^+(i)$, until the search process is no longer able to find an arc segment that has neither a label B nor a label R.

Step 2.2. Take the arcs (i, l) from the stack E_R one by one, put each node from the node set $N_n^-(l) - V_R$ into the queue V_R (the node is labeled R), and for each arc (i, l); then calculate

$$\mu(i,l) = \min_{j \in N^+(l)} \{\phi(i,l) + \theta(i,l,j) + \mu(l,j)\}$$

and record

$$\pi_\mu(i,l) = \operatorname{argmin}_{j \in N^+(l)} \{\phi(i,l) + \theta(i,l,j) + \mu(l,j)\}$$

$$\gamma((i,l)) = (l, \pi_\mu(i,l))$$

Insert the arc segment (i, l) into E_B and search for node i from $N_n^-(l)$ and delete it. Repeat these operations until the stack E_R is empty, then go to Step 3.

Step 3. Check whether $N_n^-(v) = \emptyset$ holds. (1) If it does, then delete v from V_R and go to Step 4. (2) Otherwise, for arbitrary $i \in N_n^-(v)$, if $i \notin V_R$, insert i into V_R, insert (i, v) into E_B, delete v from $N_n^+(i)$, delete i from $N_n^-(v)$, delete v from V_R; then calculate

$$\mu(i,v) = \min_{j \in N^+(v)} \{\phi(i,v) + \theta(i,v,j) + \mu(v,j)\}$$

and record

$$\pi_\mu(i,v) = \operatorname{argmin}_{j \in N^+(v)} \{\phi(i,v) + \theta(i,v,j) + \mu(v,j)\}$$

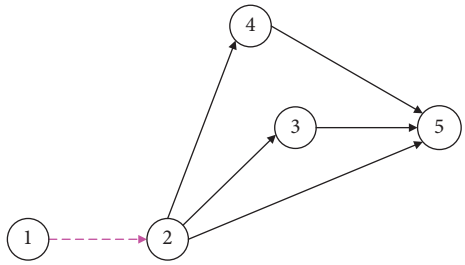

Figure 2: First case.

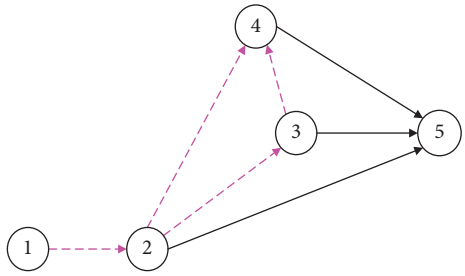

Figure 3: Second case.

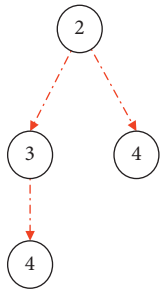

Figure 4: Breadth-first search tree.

$\gamma((i,v)) = (v, \pi_\mu(i,v))$.

Go to Step 4.

Step 4. Let $n = n+1$, and go to Step 1.

Step5. Calculate

$\varphi(S,D) = \text{argmin}_{j\in N^+(i)}\{\mu(S,j)\}$

from which the minimum travel time from the start point S to the endpoint D is obtained.

Step 6. Construct the optimal path from the start node S to the destination D by $\pi_\mu(i,v)$,

$\pi_\varphi(S) = \text{argmin}_{j\in N^+(i)}\{\mu(S,j)\}$ and $\gamma((i,v))$ END.

The steps for constructing the breadth-first search tree algorithm in the algorithm subroutines in Steps 2.1 and 2.2 are as follows:

Step 0. BEGIN. Let $E_Y = \{(v,v)\}$.

Step 1. Check whether $E_Y = \emptyset$ holds. (1) If it does, then go to Step 3; (2) Otherwise, take an arc segment $(i,j) \in E_Y$ If $N_n^+(j)$ is not empty, take out $k \in N_n^+(j)$ sequentially, insert arc segments (j,k) into the stack E_R in turn, and record $F(j,k) = (i,j)$, until $N_n^+(j)$ is empty. Go to Step 2.

Step 2. Remove (i,j) from E_Y, and go to Step 1.

Step 3. Output the collection stack E_R END.

Whether or not the set E_Y is empty controls whether or not the loop ends. $F(j,k) = (i,j)$ indicates that the subarc of the arc (i,j) on the breadth-first search tree is (j,k).

5.1.4. Modular Flow Chart of ROLA. To give a clearer explanation of the execution process of the algorithm, this section presents the pseudocode of the ROLA in modules corresponding to the specific steps of the algorithm.

The execution of the ROLA involves the construction of the breadth-first search tree, the data structure of the queue, and the stack. The breadth-first search algorithm is one of the basic graph search algorithms. Both the Dijkstra single-source shortest path algorithm and the Prim minimum spanning tree algorithm use a similar idea to breadth-first search and belong to the class of blind search methods, which systematically expand and examine all the nodes of a graph to obtain their results. Stacks and queues are two linear data structures that are widely used in programming: a stack operates according to the rule "last-in, first-out" (insert $(L, n+1, x)$ and delete (L, n)), whereas a queue operates according to the rule "first-in, first-out" (insert $(L, n+1, x)$ and delete $(L, 1)$). The algorithm performs enqueue and dequeue operations on the nodes during its execution. Because the enqueue operation starts from the endpoint, nodes close to the endpoint are placed in the queue first, and based on the "first-in, first-out" rule, nodes near the end will be dequeued first and processed first, which matches the reverse order basis of the reverse order labeling method. When constructing the breadth-first search tree, beginning with the root v, the non-B-labeled arc segments that are searched for are pushed into the stack until the search switches to the B-labeled arc segments. After this search has been completed, the pop-up stack operation is performed on the arc according to the "last-in, first-out" rule, which ensures that the minimum travel times from the arcs nearest the endpoint are calculated first, from near-to-far, and only then, the minimum travel times from arcs far from the endpoint are calculated, which is again in accordance with the reverse order labeling method.

The ROLA composed of four modules as follows: the breadth-first search tree construction process (Module 1), the arc pushing and popping stack process (Module 2), the node enqueue and dequeue process (Module 3), and the optimal path output (Module 4). In order to understand the relationship between each module more clearly and intuitively, we use pseudo to describe. The pseudo of module 1 to module 4 is shown in the Algorithms 1–4.

It can be seen from Algorithm 5 that each module has an interface connected with one or more of the other modules. So the modules are not isolated but closely related. After initialization, Module 1 or Module 4 is selected for execution by checking whether the condition $E_B = E$ holds. Module 1 is directly connected to Module 2, indicating that Module 2

(1) BFS(G, v)
% Given, $G = (V, E, \Theta, \Psi)$ node v as the root of the tree, to construct a breadth-first search tree using the arcs without the B label.
(2) *for each* $(i,l) \notin E_B \cup E_R$ *do*
(3) $Color \leftarrow [(i,l)] \leftarrow red$
(4) *end*

Algorithm 1: Breadth-first search tree (Module 1).

(1) *Push* (*stack* E_R, (i,l));
% Select each arc (i, l) that satisfies the condition $l \in N_n^+(i)$ and is not on the breadth-first search tree, insert it into the breadth-first search tree, and insert such arcs (i, l) into the stack E_R in turn (the arc E_R is marked as R)
(2) *do*
(3) $E_B = E_R \cup \{(i,l)\}$;
(4) $N_n^+(i) = N_n^+(i) - \{l\}$;
% Remove node l from the set of successor nodes of node i
(5) *Until* $(i,l) \notin E_B \&\& (i,l) \notin E_R$;
% Breadth-first search tree construction completed
(6) *Pop*(*stack*, E_R);
(7) *While* $E_R = \emptyset$
% when the optimal path from all arcs on the tree to the destination has not been obtained, proceed as follows
(8) *do*
(9) $Color \leftarrow [(i,l)] \leftarrow Black$;
(10) $E_R = E_R - \{(i,l)\}$;
(11) $\mu(i,l) = \min_{j \in N^+(l)} \{\phi(i,l) + \theta(i,l,j) + \mu(l,j)\}$
(12) $\pi_\mu(i,l) = \operatorname{argmin}_{j \in N^+(l)} \{\phi(i,l) + \theta(i,l,j) + \mu(l,j)\}$
(13) $\gamma((i,l)) = ((l, \pi_\mu(i,l))$
% Calculate the minimum time from each arc segment on the breadth-first search tree to the destination in the stacking order and record the path.
(14) *then*
(15) $N_n^-(l) = N_n^-(l) - \{i\}$
% Delete node i from the predecessor node of node l
(16) *End*

Algorithm 2: Arc stack operation (Module 2).

(1) DeQueue(V_R)
% Select each arc (i, l) that satisfies the condition $l \in N_n^+(i)$ and is not on the breadth-first search tree, insert it into the breadth-first search tree, and insert such arcs (i, l) into the stack E_R in turn (the arc E_R is marked as R)
(2) *for* $v \in V_R \&\& N_n^+(v) = \emptyset$ do % take a node v from the queue V_R
(3) *if* $N_n^-(v) = \emptyset$ *then*
(4) $V_R = V_R - \{v\}$;
% Determine whether node v is isolated, if it is, delete node v from the queue V_R.
(5) *else*
(6) *for each* $i \in N_n^-(v)$;
% Perform the following operations for each predecessor node i of the node v.
(7) if $i \notin V_R$ do
(8) *EnQueue* (V_R, i);
(9) $V_R = V_R \cup \{i\}$; % insert node i into the queue V_R
(10) $E_B = E_B \cup \{(i,v)\}$;
% insert (i, v) into the set E_B and mark it as a black arc.
(11) $\mu(i,v) = \min_{j \in N^+(v)} \{\phi(i,v) + \theta(i,v,j) + \mu(v,j)\}$
(12) $\pi_\mu(i,v) = \operatorname{argmin}_{j \in N^+(v)} \{\phi(i,v) + \theta(i,v,j) + \mu(v,j)\}$
(13) $\gamma((i,v)) = ((v, \pi_\mu(i,v))$
% Calculate the optimal path from all arc segments (i, v) with v as the end node to the destination.
(14) *then*

Algorithm 3: Continued.

(15) $N_n^+(i) = N_n^+(i) - \{v\}$; % *Remove v from* $N_n^+(i)$.
(16) $N_n^-(v) = N_n^-(v) - \{i\}$; % *Remove i from.*
(17) $V_R = V_R - \{v\}$;% *Remove v from the queue* V_R
(18) *Update* E_B
(19) *End*

Algorithm 3: Node queue operation (Module 3).

(1) Output $(\cdot)$
(2) $\varphi(S, D) = \min_{j \in N^+(S)}\{\mu(S, j)\}$;
% obtain the minimum time from start node S to destination D.
(3) $\pi_\varphi(S) = \text{argmin}_{j \in N^+(i)}\{\mu(S, j)\}$;
% obtain the subsequent nodes of the node S on the optimal path.
(4) $\pi_\mu(i, v) = \text{argmin}_{j \in N^+(v)}\{\phi(i, v) + \theta(i, v, j) + \mu(v, j)\}$;
% obtain the subsequent arc (v, j) of arc (i, v) on the optimal path from arc (i, v) to destination D.
(5) $\gamma((i, v)) = ((v, \pi_\mu(i, v))$;
% record the subsequent arc of arc (i, v) on the optimal path as $(v, \pi_\mu(i, v))$
(6) $(S, \pi_\varphi(S) \longrightarrow \gamma(S, \pi_\varphi(S)) \longrightarrow \gamma(\gamma((S, \pi_\varphi(S)))), \ldots, (D, D)$;
% obtain the best path from start S to destination D.
(7) end

Algorithm 4: Optimal path result output (Module 4).

is executed after Module 1 has been completed. Similarly, Module 2 is directly connected to Module 3. After Module 3 has been executed, by checking whether or not $E_B = E$ holds, the algorithm decides to go to the next iteration or to exit the program after executing Module 4.

5.1.5. Correctness of the Algorithm and Proof of Time Complexity. (1) Correctness of the Algorithm

Proposition 1. *Before the n-th loop execution of the algorithm, if node i already exists in the queue* V_R*, it cannot be inserted into* V_R *again during the n-th loop execution; i.e., during the execution of the algorithm, it is impossible for the same node to appear more than once in the queue* V_R*.*

PROOF. This proposition follows clearly from the way in which the algorithm is executed. □

Proposition 2. *If a node* $i \in V_R$ *is deleted from* V_R *in the n-th loop execution of the algorithm, then* $N_n^+(i) = \emptyset$ *must hold after the n-th loop execution; so, for arbitrary* $m > n$*, node i cannot be inserted into* V_R *again during the m-th loop execution.*

Proof. For any node $i \in V_R$, if node i is deleted from the queue V_R during the n-th loop execution of the algorithm, then after the Step 2 and Step 3 operations in the n-th loop execution, $N_n^+(i) = \emptyset$ a must hold. For any $m > n$, before the m-th loop execution, $N_n^+(i) = \emptyset$ also holds; and during the m-th loop execution, before a certain node j can be inserted into the queue V_R, $N_n^+(j) = \emptyset$ a must hold. Therefore, during the m-th loop execution of the algorithm, node i cannot be inserted into the queue V_R, and the proposition is proved.

Proposition 3. *After the end of a certain loop execution of the algorithm, if* $E_B \neq E$ *holds, then* $V_R \neq \emptyset$ *must hold.*

Proof. After the end of a certain loop execution of the algorithm, if $E_B \neq E$ holds, then there must exist a certain arc segment (i, j) without label B, and $k \in N^+(j)$, $(j, k) \in E_B$, and it is obvious that $N_n^+(i) = \emptyset$ and $N^+(j) = \emptyset$ hold. It is then known from Proposition 2 that node i is either in the queue V_R or is not in the queue and has never entered it. If it is assumed that node i is not in the queue V_R and has never entered it, then, because the arc segment (j, k) has a label B, node i must have been inserted into V_R during the procedure of inserting the arc (j, k) into the set E_B, which is contradictory; hence the assumption is not true. Therefore, if $E_B \neq E$ holds, $V_R \neq \emptyset$ must hold, and Proposition 3 is proved.

From Propositions 1–3, it is easy to derive the following theorem.

Theorem 3. *During the execution of the algorithm, any node* $i \in V$ *enters and exits the queue* V_R *at most one time, so there can be no more than* $|V|$ *loops from Step 1 to Step 4 of the algorithm.*

Proposition 4. *The ROLA has the following loop invariance: before the start of the n-th loop, if all the arcs in the set* E_B *have label B, then all the arcs in* E_B *still have label B before the start of the* $(n + 1)$*-th loop. The proof of this is omitted here.*

From Theorem 4 and Propositions 3, and 4, the following theorem can be obtained.

```
Initialization: G = (V, E, Θ, Ψ), n = 1, E_R = ∅, V_R = {v|vt = nD}, E_B = ∅, for ∀i ∈ V, N_n^+(i) = N^+(i), N_n^-(i) = N^-(i)
Result: A Sequence Π: = {(S, i), (i, j), ..., (D, D)}
Main procedure
(1) if E_B ≠ E
(2)   for ∀v ∈ V_R
(3)   if N_n^+(v) = ∅
      % if N_n^+(v) = ∅ directly performs node vv queue operation (i.e. Module 3)
(4)     execute Module 3;
(5)     if E_B ≠ E
(6)      execute Module 1;
(7)       else
(8)         execute Module 4;
(9)     else
(10)        execute Module 1;
            % perform the breadth-first search tree construction operation with v as the root node
(11)        execute Module 2;
            % perform push and pop operations on the arcs on the breadth-first search tree constructed
(12)        execute Module 3
            % when all the arcs in the stack E_R are popped out of the stack, perform the queue operation of node v
(13)          if E_B ≠ E
(14)          execute Module 1;
                % if E_B ≠ E, enter the next loop of the algorithm, first perform the operation of building breadth-first search tree
(15)        else
(16)          execute Module 4;
              % if E_B = E, the algorithm loop termination condition is met, execute the algorithm's route output operation
(17) else execute Module 4;
(18) end.
```

Algorithm 5: Relationship between modules.

Theorem 4. *During the $|V||V|$ loops (at most) from Step 1 to Step 4 of the ROLA, the condition $E_B = E$ holds; i.e., for any arc segment $(i, j) \in E$, the minimum travel time $\mu(i, j)$ though the arc segment (i, j) to the endpoint D can be obtained.*

Proposition 5. *For any arc segment $(i, j) \in E$, during the $|V|$ loops (at most) from Step 1 to Step 4 of the ROLA, the arc segment (i, j) is put no more than once into the breadth-first search tree that is constructed so that it goes into and out of the stack E_R at most once.*

Proof. If an arc (i, j) appears in the breadth-first search tree in Step 2 of a certain loop, then, after the end of that loop, the arc (i, j) must have label B (because it has already been inserted into the set E_B). During each subsequent loop execution, any arc segment (i, j) inserted into the breadth-first search tree does not have label B before it is inserted, so the arc segment (i, j) cannot be inserted into the breadth-first search tree again during any of the subsequent loop executions. Thus Proposition 5 is proved.

(2) Time Complexity of the Algorithm

It can be seen from the description of the algorithm that the difficulty in analyzing its time complexity lies in the loops from Step 1 to Step 4, of which there $|V|$ at most. The time complexity cost in the execution of the algorithm consists mainly of the following three components:

(1) The time cost of all nodes that are put into or are removed from the queue V_R

(2) The time cost of constructing a breadth-first search tree and inserting and removing arcs from the stack E_R

(3) The time cost of calculating the minimum travel time $\mu(i, j)$ via all arcs (i, j) to the endpoint D

Let

$$N = \max_i \left\{ \left| N^+(i) \right|, \left| N^-(i) \right| \right\}. \tag{16}$$

Theorem 5. *From the beginning to the end of the algorithm, during the construction of the breadth-first search tree, including the operations on all nodes that are put into or taken out of the queue V_R and on all arcs that are put into or taken out of the stack E_R, the total complexity cost in computation time is $O(|V| + N \cdot |E|)$.*

Proof. The proof of the theorem is divided into three steps.

In the first step, we first prove that, during the process from the beginning to the end of the algorithm, the computation time complexity cost of all arcs that are put into or taken out of the stack E_R (i.e., into or out of the breadth-first search tree) is $O(N \cdot |E|)$. As can be seen from Proposition 5, any arc segment $(i, j) \in E$ is put into and taken out of the breadth-first search tree at most once, i.e., into and out of the stack E_R at most once. According to the operation of the algorithm, for each (i, j) that is put into and taken out of the stack E_R, the time cost of entering E_R is $O(1)$, so for all the arcs (i, j) that are

put into and taken out of E_R, the total cost of entering E_R is $O(|E|)$. For each arc (i, j) that is put into and taken out of the stack E_R, there is an operation of removing (i, j) from the stack E_R and inserting it into the stack E_B, and the time complexity of this part of the operation is $O(1)$. There is also the operation of searching for node i in the set $N_n^-(j)$ and deleting it from this set, the time complexity cost of which is $O(N)$; hence the time complexity of removing the arc (i, j) from the stack E_R is $O(N)$. Therefore, for all the arcs (i, j) that are put into and taken out of the stack E_R, the total time cost when they are moved out of E_R is $O(N \cdot |E|)$. In summary, the total time cost of all the arcs (i, j) that are put into and taken out of the stack E_R is $O(N \cdot |E|)$.

In the second step, we prove that, for all the nodes i that are put into and taken out of the queue V_R , the time cost when they are removed from V_R is $O(|V|)$. From Theorem 3, for an arbitrary node $i \in V$, in the execution of the algorithm from beginning to end, node i enters and exits the queue V_R at most once. According to the operation of the algorithm, for each node i that is put into and taken out of V_R, the time cost when it is taken out of V_R is $O(1)$. Therefore, for all nodes i that are put into and taken out of the queue V_R , the total time cost when they are removed from V_R is $O(|V|)$.

In the third step, we prove that, for each node i that is put into and taken out of the queue V_R, the time cost when it is inserted into V_R is $O(|E|)$. For any node ii that is put into and taken out of the queue V_R, two situations can occur: in one, node i is the predecessor node of a node v that is about to be moved out of V_R (i.e., $i \in N_n^-(v)$); in the other, node i is the predecessor node of the endpoint l of the arc (j, l) in the breadth-first search tree that has been constructed. In the former situation, when node i is inserted into the queue V_R, all the nodes in the set $N_n^-(v)$ that are not included in the queue V_R are inserted into V_R, which is equivalent to beginning from node v, searching for all the arcs (i, v) that satisfy the conditions $i \in N_n^-(v)$ and $i \notin V_R$, and then inserting node i into the set V_R. Therefore, from the beginning to the end of the algorithm, the time cost of all the nodes that are inserted into the queue V_R, because they are predecessor nodes of a certain node that was removed from V_R, is $O(|E|)$. In the second situation, when the node i is inserted into the queue V_R, all the nodes that are in the set $N_n^-(l)$ and do not belong to V_R are inserted into V_R. This is equivalent to starting from node l, scanning and searching for all the arcs (i, l) that satisfy the conditions $i \in N_n^-(l)$ and $i \notin V_R$, and inserting node ii into the set V_R; then $N_n^-(l) = \varnothing$ holds. Therefore, in this loop or later loops, when we search for an arc (k, l) that has the same endpoint as the arc segment (j, l) from the stack E_R, because $N_n^-(l) = \emptyset$, the arc segment (i, l) that was previously scanned and searched for no longer needs to be searched for. By Proposition 5, any arc segment $(j, l) \in E$ is inserted into and removed from the breadth-first search tree at most once in the $|V|$ iterations at most of the algorithm. Therefore, for any node $l \in V$, each arc (i, l) that satisfies the condition $i \in N_n^-(l)$ needs at most one scan to search for node i and inserted it into V_R. Hence the time complexity of all those nodes that are inserted into V_R because they are predecessors of the endpoint of the arc (j, l) in the breadth-first search tree is $O(|E|)$.

In summary, from the beginning to the end of the algorithm, during the construction of the breadth-first search tree, including the operations on all nodes that are put into and taken out of the queue V_R and all arcs that are put into and taken out of the stack E_R, the total computation time complexity cost is $O(|V| + N \cdot |E|)$. Thus Theorem 5 is proved.

Theorem 6. *Using ROLA, the complexity cost of calculating the minimum time for all arcs in E to reach the destination D is* $O(N \cdot |E|)$, *and*

$$\mu(i, j) = \min_{l \in N^+(j)} \{\phi(i, j) + \theta(i, j, l) + \mu(j, l)\}, \tag{17}$$

where

$$\begin{aligned} &\phi(i, j) \in \Psi, \\ &\theta(i, j, l) \in \Theta, \\ &j \in N^+(i), \\ &l \in N^+(j), \end{aligned} \tag{18}$$

$$\mu(i, D) = \varphi(i, D), \quad \forall i \in N^-(D). \tag{19}$$

Proof. For arbitrary $(i, j) \in E$ in the process of calculating $\mu(i, j)$ according to the above recursion formula, it is necessary to calculate $|N^+(j)|$ times the following expression and then to perform $|N^+(j)| - 1$ comparison operations.

$$\phi(i, j) + \theta(i, j, l) + \mu(j, l). \tag{20}$$

In each calculation using this expression, there are two addition operations, so the time complexity cost of calculating $\mu(i, j)$ for one arc segment (i, j) is $O(N)$. Therefore, the complexity cost of calculating the minimum time through all arcs in E to the endpoint D is $O(N \cdot |E|)$, where $N = \max\{|N^+(i)|, |N^-(i)|\}$. Thus Theorem 6 is proved. The following theorem is apparent from Theorems 5 and 6. □

Theorem 7. *The time complexity of the ROLA in a time-invariant network is* $O(|V| + N \cdot |E|)$.

It can be seen that the time complexity of the ROLA is a polynomial in the number of nodes (|V|), the number of edges (|E|), the maximum number of predecessors, and the number of subsequent nodes (N) in the network. The time complexity of the algorithm is low.

5.2. Calculation Example. A simple directed graph is used to illustrate the execution process of the algorithm. As shown in Figure 5, the directed graph G is composed of 11 nodes and 18 edges. The arrows indicate the direction of travel on each road section. Node 11 is a given starting point and node 1111 is the given destination. The travel time values $\phi(i, j)$ on the arcs (i, j) are shown in Table 2, and the waiting time values $\theta(i, j, k)$ of the arc (i, j) selected different arcs (j, k) at the nodes j are shown in Table 3.

Our goal is to calculate the minimum travel time path from each arc segment to the destination and the minimum

travel time path from the starting node 11 to the destination 1111 in the network. When the intersection attribute (weight or time cost) is not considered, the minimum time path from the starting node 11 to the endpoint 1111 is $1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 10 \longrightarrow 11$ obtained by the Dijkstra algorithm, and the travel time is 11.

The Reverse Order Labeling Algorithm given in this paper is used to solve the minimum time path from node 11 to node 11 in Figure 5 considering the waiting time of the node. The specific calculation process is omitted here. After 9 iterations of calculation, the data in Table 4 are obtained when the algorithm terminates. Including the minimum time path of each arc in the directed graph to the end and the travel time value. Let

$$\varphi(1, 11) = \min_{j \in N^+(1)} \{\mu(1, j)\} = \mu(1, 2) = 16. \quad (21)$$

According to $\gamma(i, j)$, the minimum expected time path from the starting point 11 to the destination 11 is $1 \longrightarrow 2 \longrightarrow 5 \longrightarrow 9 \longrightarrow 11$, and the minimum travel time is 16.

6. Simulation Comparison Experiment

6.1. Actual Urban Network Simulation of ROLA. This section presents an example of the use of the ROLA to solve a path planning problem in an actual urban transportation network. We implement the algorithm in MATLAB on an Intel Core i3-550 processor at 3.20 GHz with 4 GB memory on a Windows 10 32-bit operating system. The data of the passing time for each road section and the waiting times for different turns at the intersections in the selected area are randomly generated by the simulation software.

6.1.1. Map Selection and Graphics. To provide a realistic example of the shortest-path optimization problem in an actual road network taking account of the waiting times at intersections, the road traffic map of Tianjin was selected as the research object, shown by the red lines in Figure 6.

The roads and their intersections in the selected area are represented by the 39 edges and 25 nodes in the map, as shown in Figure 7, where the arrows indicate the directions of travel on the road sections. The travel time $\phi(i, j)$ on each edge and the time spent $\theta(i, j, k)$ on each node are all uniformly distributed with U [1, 10]. The goal of the optimization is to obtain the minimum-time path from N1 to N25 in the directed graph.

The diagram of the simulated road network as drawn in MATLAB according to the relationship between nodes and edges is shown in Figure 8.

6.1.2. Simulation Program Design. The experimental program consists of four sections: two data input sections, the algorithm section (the most important part), and a framework for data processing, data transmission, and display of the results. A simple menu interface is displayed in the main window of MATLAB. To facilitate the functions of calling input and calculation using the menu, the functions of each section are written as independent.m files, which are listed together with their functions in Table 5. An error handling function is integrated. For example, when the start and endpoints of the shortest path are input, the program will judge whether the start point and the endpoint are the same. If they are equal, an error message will be output. The connections between them are illustrated in Figure 9.

6.1.3. ROLA Simulation Result. The data needed in the algorithm is represented by the adjacency matrix. Because MATLAB language itself is used for matrix calculation and has inherent advantages for matrix calculation. Use arc_data (i, j) represent the travel time of any arc (i, j), and node_data (i, j, k) represent the waiting time of arc (i, j) tuning arc (j, k). Since we are studying a directed graph, in general: arc_data $(i, j) \neq$ arc_data (j, i) and node_data $(i, j, k) \neq$ node_data (k, j, i), if there is no path from i to j, then arc_data $(i, j) = \infty$ and node_data $(i, j, k) = \infty$. For a network with n nodes, arc_data (i, j) is a $n * n$ two-dimensional matrix, and node_data (i, j, k) is a $n * n * n$ three-dimensional matrix.

The simulation procedure is as follows: start MATLAB (version 7.11.0 R2010b), put all.m files into the MATLAB search path, enter the "menu" in the MATLAB interface, and output the main menu. Enter "1" to input the adjacency matrix of the arc. The program prompts success and outputs the menu again. Enter "2" to input the node adjacency matrix. The program then prompts success and outputs the menu again. Enter "3" to view the data and output the array of arc_data and node_data. Enter "4" to find the path, obtain the minimum-time path from the given arc to the destination and the minimum-time path from the starting node to the destination, and arbitrarily select two target arcs for verification.

Because there are 39 edges in the network, the fourth step is performed 39 times to find the minimum-time paths and values for all the arcs to the endpoint. It is easy to find the subsequent arcs on the optimal path from each arc to the endpoint. The minimum-time path from the starting node N1 to the destination node N25 is $N1 \longrightarrow N2 \longrightarrow N7 \longrightarrow N12 \longrightarrow N13 \longrightarrow N14 \longrightarrow N15 \longrightarrow N20 \longrightarrow N23 \longrightarrow N25$. The travel time is 67.8, and the CPU running time of the simulation is 0.0238s. The optimal simulation route is shown in Figure 10 by the thick gray arrows and the corresponding optimal route on the map is shown in Figure 11.

By calling the traditional Dijkstra algorithm in the MATLAB simulation environment, we easily obtain the minimum-time path from the starting point N1 (do not consider node attribute) to the destination N25 (do not consider node attribute) is $N1 \longrightarrow N2 \longrightarrow N7 \longrightarrow N12 \longrightarrow N13 \longrightarrow N14 \longrightarrow N15 \longrightarrow N20 \longrightarrow N23 \longrightarrow N25$, with travel time 45.6.

It can be seen from the simulation results for the above two cases that the same optimal paths (under the coincidence) are obtained but with different values, which

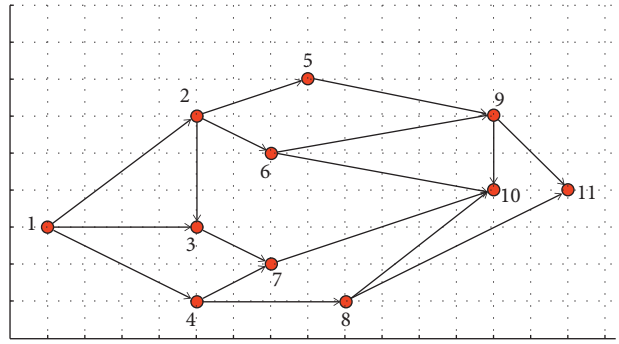

Figure 5: Directed graph representing a time-invariant network.

Table 2: Edge attribute value.

$\phi(1,2)$	$\phi(1,3)$	$\phi(1,4)$	$\phi(2,5)$	$\phi(2,6)$	$\phi(3,2)$
2	4	3	2	2	2
$\phi(3,7)$	$\phi(4,7)$	$\phi(4,8)$	$\phi(5,9)$	$\phi(6,9)$	$\phi(6,10)$
3	4	5	2	3	2
$\phi(7,10)$	$\phi(8,10)$	$\phi(8,11)$	$\phi(9,11)$	$\phi(10,9)$	$\phi(10,11)$
4	5	5	6	4	5

Table 3: Node attribute value.

$\theta(1,2,5)$	$\theta(1,2,6)$	$\theta(1,3,2)$	$\theta(1,3,7)$	$\theta(1,4,7)$	$\theta(1,4,8)$
1	3	1	2	3	2
$\theta(2,5,9)$	$\theta(2,6,9)$	$\theta(2,6,10)$	$\theta(3,2,5)$	$\theta(3,2,6)$	$\theta(3,7,10)$
1	3	3	2	1	4
$\theta(4,7,10)$	$\theta(4,8,10)$	$\theta(4,8,11)$	$\theta(5,9,11)$	$\theta(6,9,11)$	$\theta(6,10,9)$
2	2	3	2	2	3
$\theta(6,10,11)$	$\theta(7,10,9)$	$\theta(7,10,11)$	$\theta(8,10,9)$	$\theta(8,10,11)$	$\theta(10,9,11)$
4	4	3	3	2	1

Table 4: Algorithm execution result.

$\mu(10,11) = 5, \gamma((10,11)) = (11,11)$	$\mu(10,9) = 11, \gamma((10,9)) = (9,11)$
$\mu(9,11) = 6, \gamma((9,11)) = (11,11)$	$\mu(8,11) = 5, \gamma((8,11)) = (11,11)$
$\mu(8,10) = 12, \gamma((8,10)) = (10,11)$	$\mu(7,10) = 12, \gamma((7,10)) = (10,11)$
$\mu(6,10) = 10, \gamma((6,10)) = (10,11)$	$\mu(6,9) = 11, \gamma((6,9)) = (9,11)$
$\mu(5,9) = 10, \gamma((5,9)) = (9,11)$	$\mu(4,8) = 12, \gamma((4,8)) = (8,11)$
$\mu(4,7) = 18, \gamma((4,7)) = (7,10)$	$\mu(3,7) = 19, \gamma((3,7)) = (7,10)$
$\mu(3,2) = 17, \gamma((3,2)) = (2,5)$	$\mu(2,6) = 15, \gamma((2,6)) = (6,10)$
$\mu(2,5) = 13, \gamma((2,5)) = (5,9)$	$\mu(1,4) = 17, \gamma((1,4)) = (4,8)$
$\mu(1,3) = 22, \gamma((1,3)) = (3,7)$	$\mu(1,2) = 16, \gamma((1,2)) = (2,5)$

depending on whether or not node attributes are taken into account.

6.2. Comparative Simulation Experiment

6.2.1. Test Environment and Parameter Settings. In this section, in order to measure the performance of the algorithm ROLA proposed in this paper and the more widely used algorithms [34], we have conducted extensive simulation experiments, and the implemented algorithms include PSO, ACO, GA, NNA, and OPABRL [4]. Table 6 gives the parameter settings of each algorithm.

The simulation process will be carried out on the MATLAB platform (same as above). Under the condition of selecting the same starting node and destination node, we compare the differences among the path obtained by different algorithms. The main measurement indicators include (1) algorithm convergence speed; (2) algorithm running time.

We select 10 groups of networks with different numbers of nodes and edges. The grouping is shown in Table 7. The number of network nodes, respectively, are $50, 100, \ldots, 500$, and the number of corresponding arc segments, respectively, are $100, 250, \ldots, 1450$. Regarding the composition of each group of the network, similar to Figure 7, each node is connected to at least one other node and at most six other nodes. Starting from the first node, each node is numbered from near to far, and the first node is the starting node and the last node is the destination. Same as before, for the network with n nodes,arc_data(i, j) is an $n * n$ two-dimensional matrix, and node_data(i, j, k) is

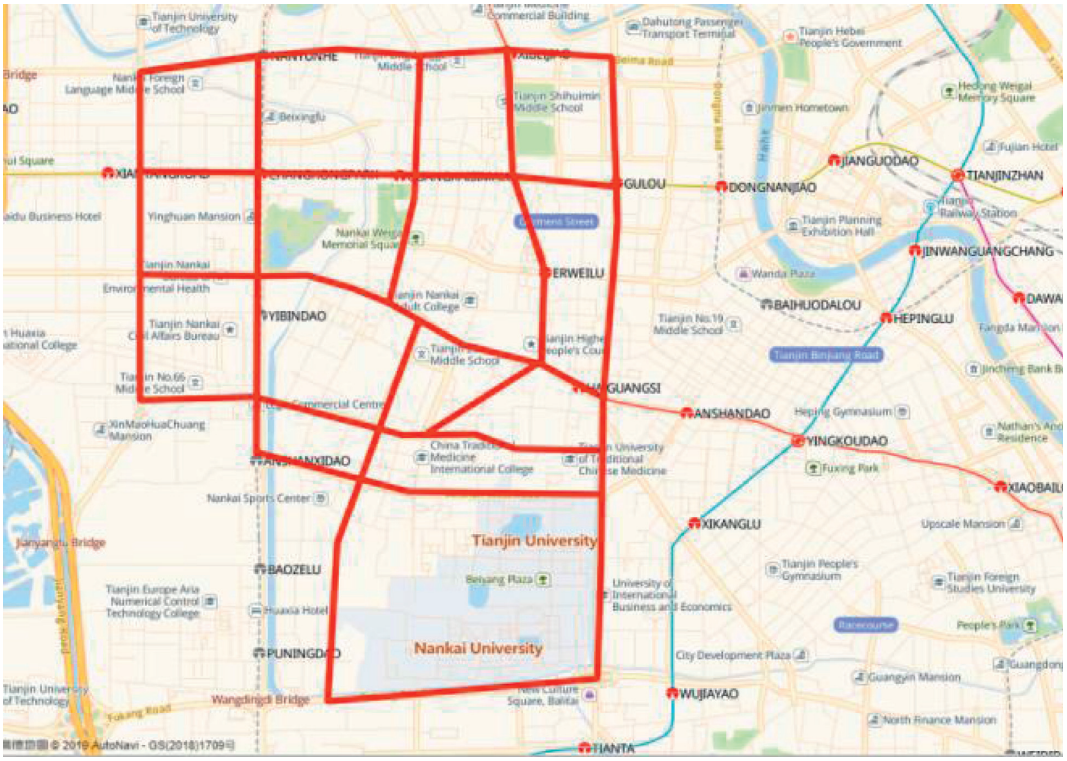

Figure 6: Target road network.

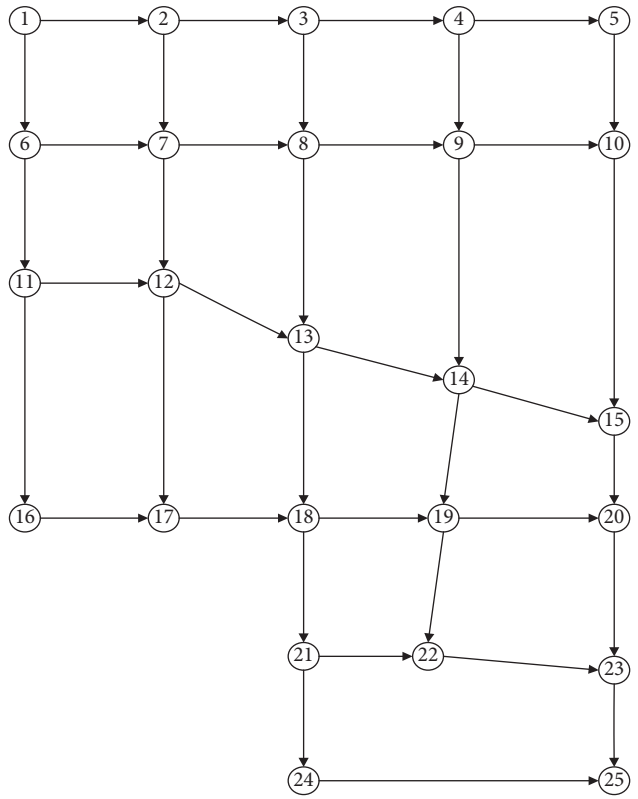

Figure 7: Graphical representation of the selected road network.

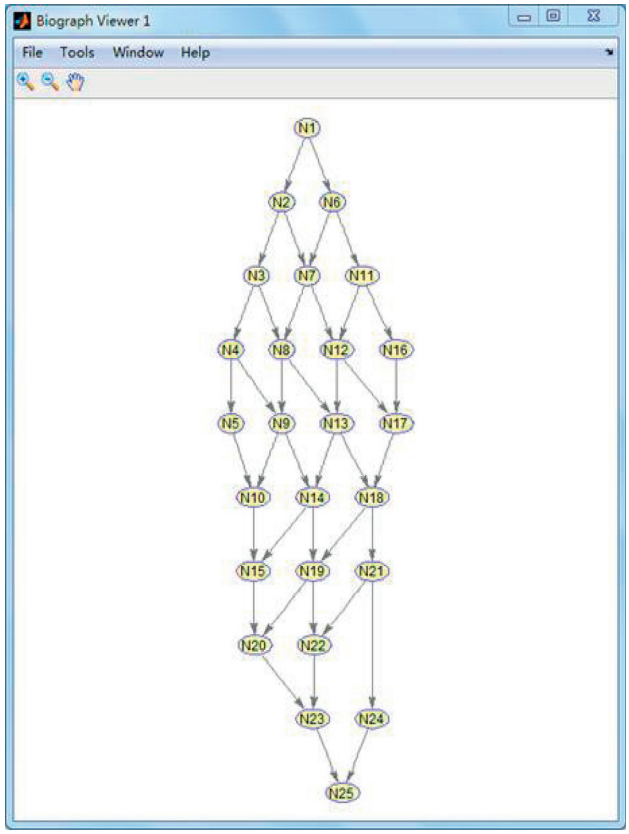

Figure 8: Road network simulation diagram.

Table 5: List of M files and their functions.

menu.m	Displays the menu and calls other functions. It is the entrance to the program.
ROLA.m	Performs the ROLA operation on the data input and returns the result. It is the core of the program.
input_arc.m	Calls the arc travel time data to generate the adjacency matrix and return it.
input_node.m	Calls the node travel time data to generate the adjacency matrix and return it.

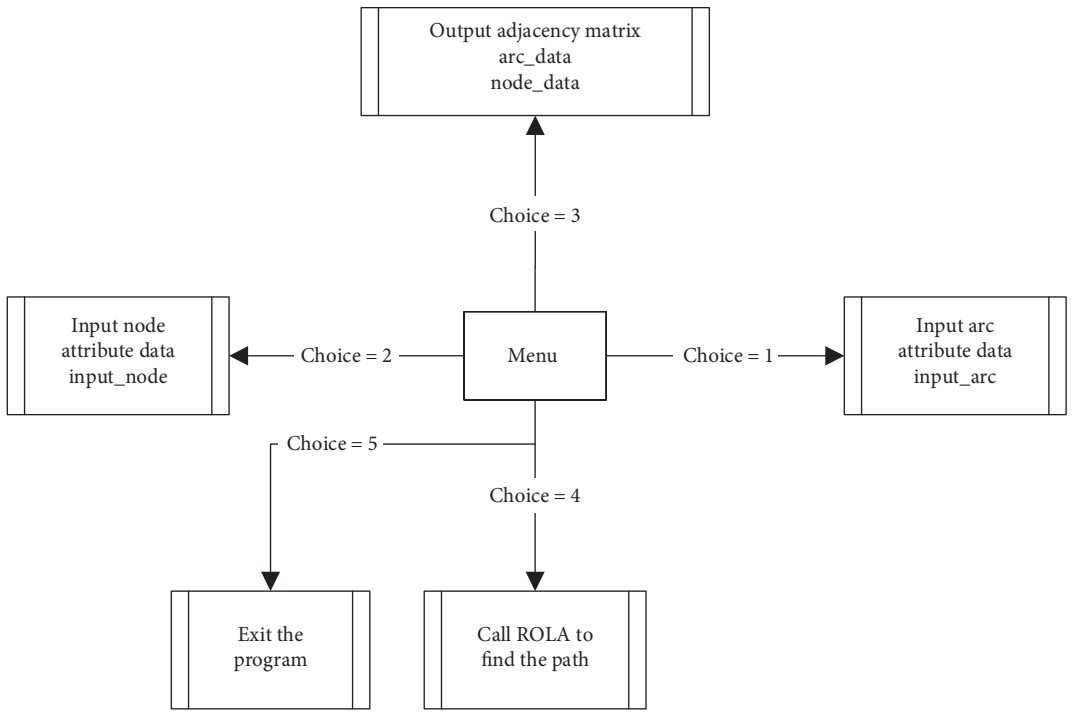

Figure 9: Call relationship of each section of the program.

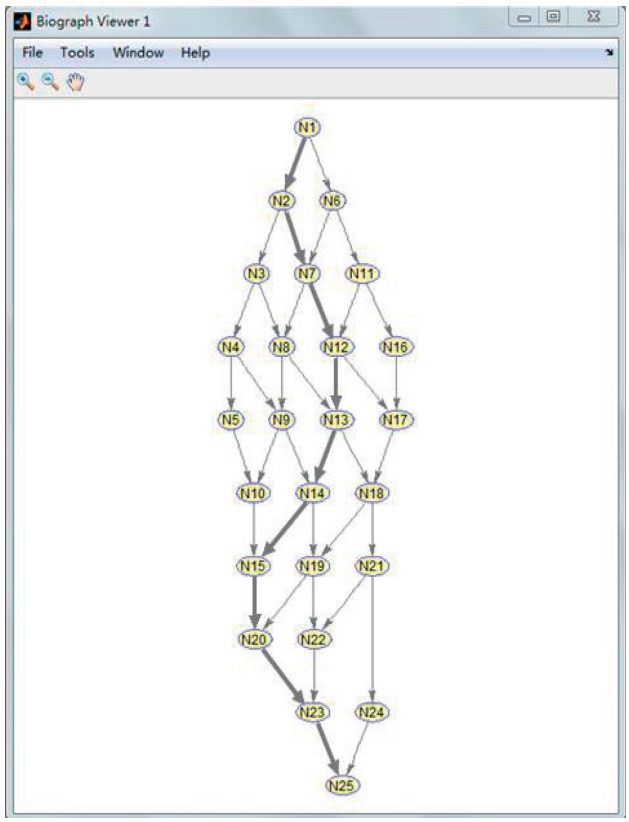

Figure 10: Optimal route simulation.

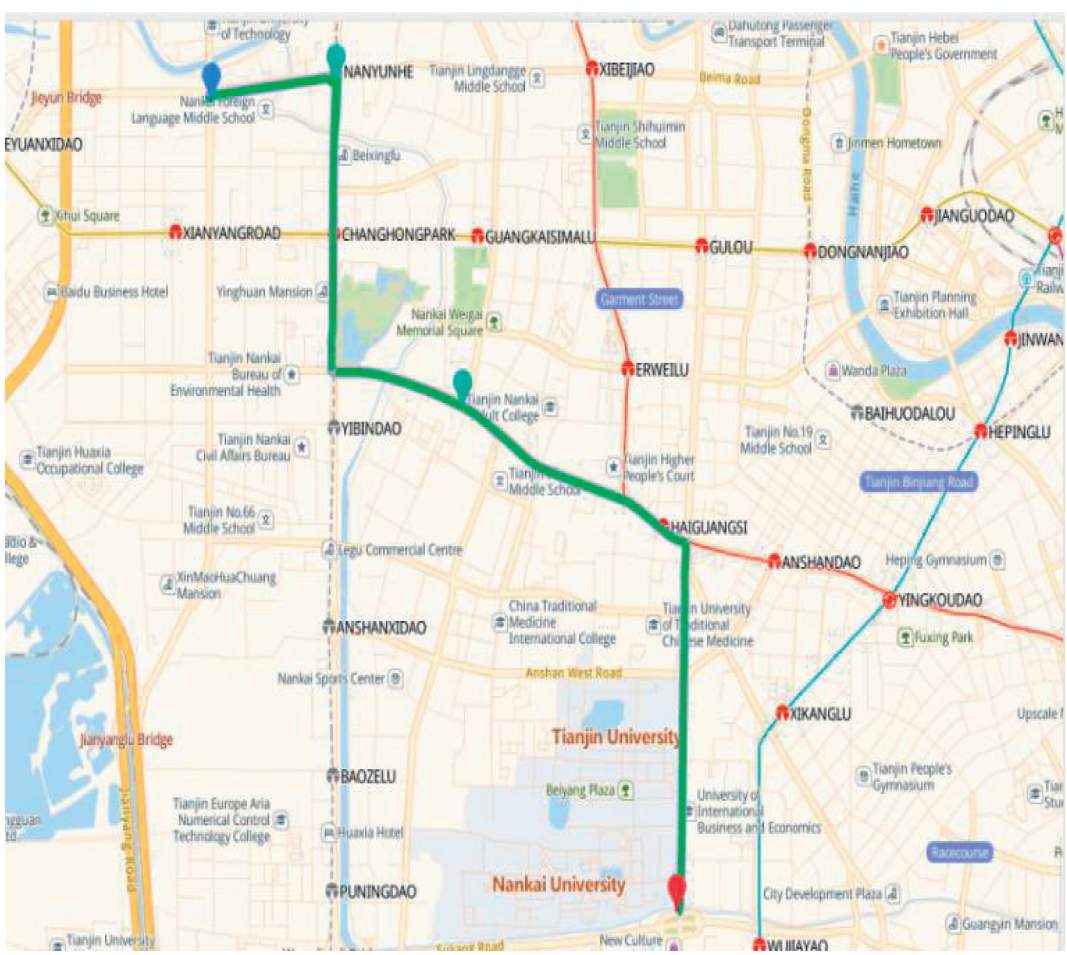

Figure 11: Optimal route on the map.

Table 6: Parameters setting.

Algorithm	Parameters	Value
ACO	Ant number m	80
	Heuristic factor a	1
	Pheromone volatility ρ	0.3
GA	Population size M	100
	Cross probability P_1	0.6
	Mutation probability p_2	0.01
NNA	Network layers	3
	Number of input nodes	6
	Number of output nodes	1
	Number of hidden layer nodes	12
PSO	Particle swarm size	80
	Learning factor c_1	1.5
	Learning factor c_2	2.0
	Inertia weight parameter ω	0.9
OPABRL	Learning factor α	0.7
	Discount factor θ	1
	Greedy strategy ε	0.5

Table 7: Grouping of networks selected.

	NODE_COUNT	ARC_COUNT
GROUP1	50	100
GROUP2	100	250
GROUP3	150	400
GROUP4	200	550
GROUP5	250	700
GROUP6	300	850
GROUP7	350	1000
GROUP8	400	1150
GROUP9	450	1300
GROUP10	500	1450

Table 8: Statistics of experimental results.

Statistical indicators	Method	NODE 50	NODE 100	NODE 150	NODE 200	NODE 250	NODE 300	NODE 350	NODE 400	NODE 450	NODE 500
Iterations	ROLA	34	87	104	155	202	267	349	452	495	522
	PSO	43	93	136	203	301	399	452	469	475	479
	ACO	51	102	165	255	342	415	455	487	492	513
	GA	60	105	134	234	331	408	464	501	512	527
	NNA	57	110	173	265	358	422	456	478	501	526
	OPABRL	39	92	125	187	258	312	341	363	371	390
Path length	ROLA	97	303	514	675	816	1023	1225	1421	1732	2003
	PSO	106	308	522	692	824	1035	1243	1436	1732	2014
	ACO	112	310	527	679	828	1044	1218	1421	1747	2005
	GA	117	310	532	712	833	1029	1228	1431	1729	2011
	NNA	114	314	533	704	819	1023	1237	1440	1744	2025
	OPABRL	105	306	514	683	816	1031	1234	1426	1753	2047
Running time (ms)	ROLA	8	62	181	369	708	914	1251	1625	1844	1893
	PSO	14	190	442	1004	1213	1567	1617	1639	1652	1666
	ACO	21	168	607	1298	1591	1760	1792	1828	1834	1829
	GA	30	153	524	1158	1539	1822	1910	1926	1934	1927
	NNA	154	273	573	1067	1330	1548	1667	1746	1762	1769
	OPABRL	11	92	381	887	991	1162	1159	1213	1231	1242

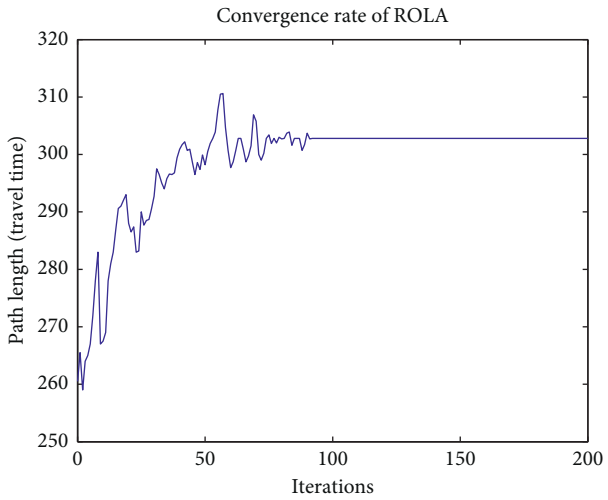

Figure 12: Convergence rate of ROLA (NODE_COUNT = 100).

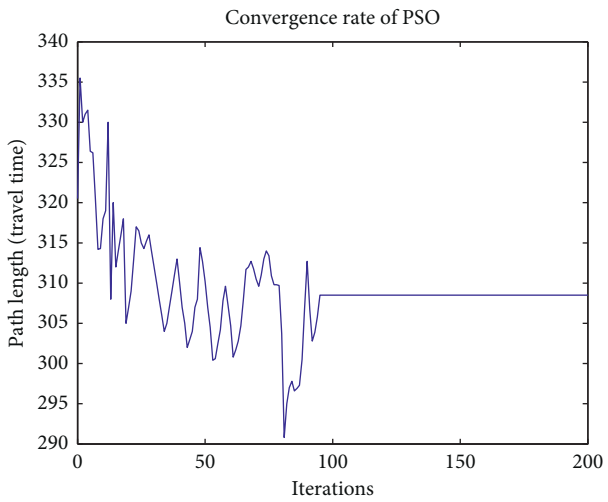

Figure 13: Convergence rate of PSO (NODE_COUNT = 100).

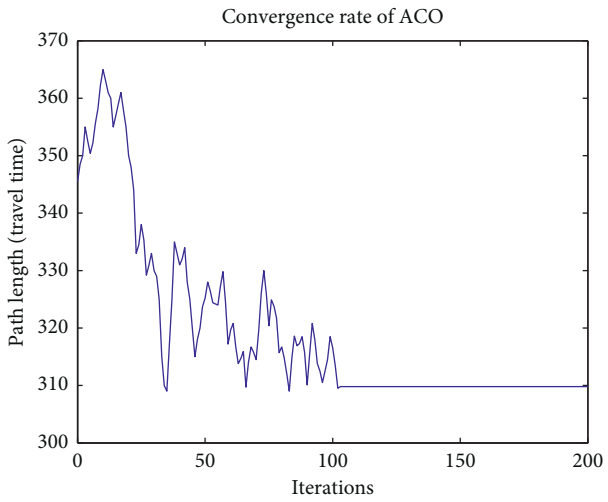

Figure 14: Convergence rate of ACO (NODE_COUNT = 100).

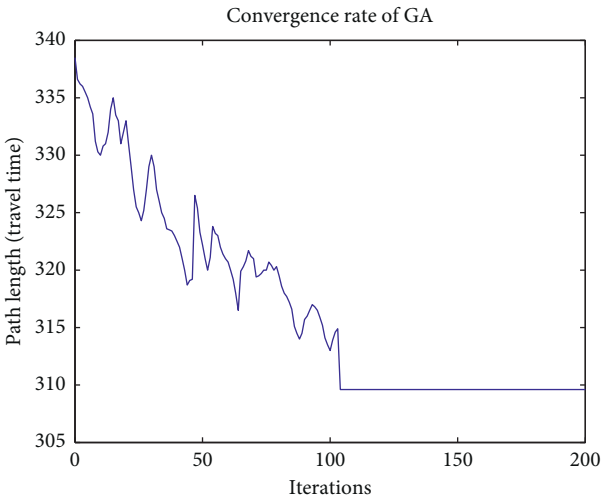

Figure 15: Convergence rate of GA (NODE_COUNT = 100).

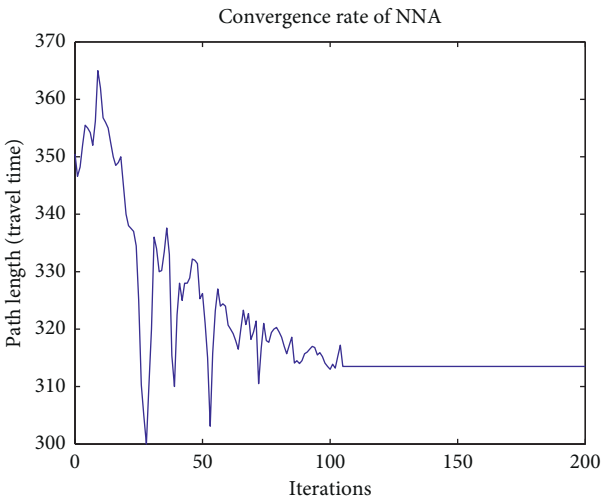

Figure 16: Convergence rate of NNA (NODE_COUNT = 100).

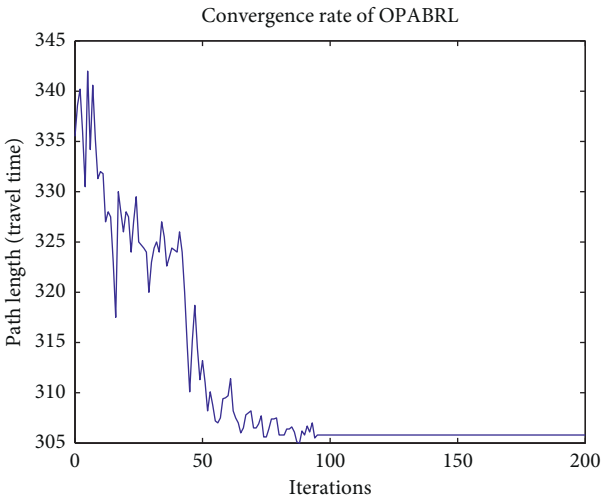

Figure 17: Convergence rate of OPABRL (NODE_COUNT = 100).

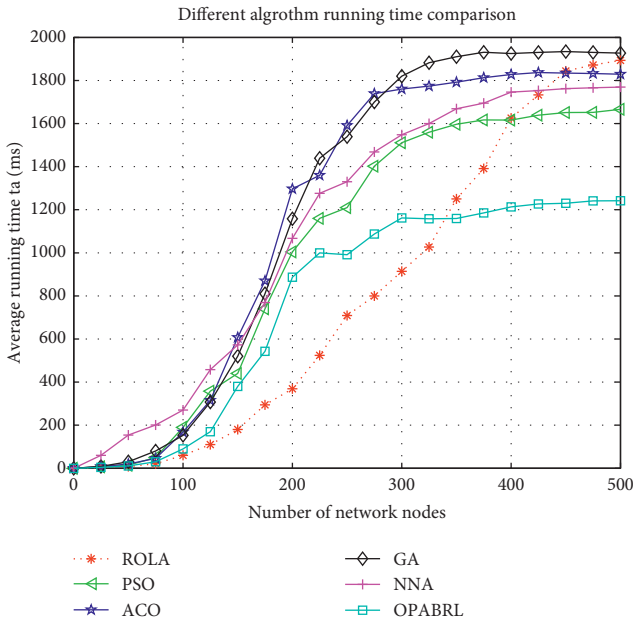

Figure 18: Running time of different algorithms.

a $n * n * n$ three-dimensional matrix. The value of travel time $\phi(i, j)$ for each arc is uniformly distributed with $U[1, 5]$, the value of time spent $\theta(i, j, k)$ for each node are uniformly distributed with $U(1, 5)$. In the experiment, we need to call each algorithm to find the minimum time path from the starting node to the destination in each group of the network, each algorithm runs ten times for each group of networks, and the average value is taken as the statistics. The total number of runs of all algorithms is 500.

6.2.2. Simulation Experiment Results and Analysis. The average iterations, path length change, running time of the proposed ROLA, and the other five comparison algorithms running ten times in each network size are shown in Table 8.

The convergence rate of these algorithms is shown in Figures 12–17 (here, we give the experimental results of the network with 100 nodes) where the x-axis represents the number of iterations and y-axis represents the shortest path length (travel time) change in the algorithm process. From the experiment result, we observed that, although there are fluctuations at the beginning, all the implemented algorithms can reach convergence after multiple iterations. The number of iterations for the proposed ROLA to reach convergence is significantly less than that of GA, NNA, and ACO, we also found there is some instability after the ant colony algorithm converges. Even though the ROLA fluctuates greatly at the beginning, under the same experimental environment, the algorithm can converge to the shortest path result earlier and has a better stability.

Regarding the computational effort required by the different algorithms, Figure 18 shows the average computational times (in milliseconds) for all considered algorithms for each instance size. In view of the results, the following points can be highlighted: (1) with the increase of network nodes, the running time of the other five algorithms increasing faster and faster, when the network nodes reach a certain number, the running time changes very little. However, the growth rate of running time changes for the ROLA changes very little, when the network nodes reach a certain number, and it also tends to be stable. (2) When network nodes are less than 350, the ROLA outperforms all the other algorithms. (3) When network nodes more than 350, the OPABRL outperforms all the other algorithms. Therefore, the running time performance of the proposed ROLA significantly outperforms other algorithms in the small to medium scale network.

Through all experiments, it can be concluded that the ROLA in this paper is easy to implement and shows good practicability and high efficiency under certain conditions.

7. Conclusion

In view of the importance of intersections in the actual road network, this paper studies the minimum time path optimization problem in time-invariant networks considering node attributes (weight or time cost). Based on the Dijkstra algorithm, we propose a Reverse Order Labeling Algorithm (ROLA) to solve the problem. We theoretically proved the correctness of the ROLA , analyzed, and gave its time complexity. Through large-scale simulation experiments, the convergence efficiency and calculation speed of the ROLA and the PSO, GA, ACO, NNA, and OPABRL algorithms are compared. The experimental results show that the algorithm proposed (ROLA) in this paper has certain advantages in the running time and search efficiency under certain conditions.

In the future, on the basis of the research in this article, we will try to extend the research results of this article to more realistic stochastic time-varying networks and conduct research on the path planning of stochastic time-varying networks considering node attributes.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request (ddzhu_0703@163.com).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research work was supported by the National Nature Science Foundation under Grant 71501141 and the Science and Technology Ministry's Special Foundation of Developing Important and Large Equipment under Grant 2014YQ120351.

References

[1] B. W. Yang, "Research status and development trend analysis of intelligent transportation system," *China Plant Engineering*, vol. 2, pp. 121-122, 2019.

[2] Q. T. Geng, *Study on the Key Technology of Intelligent Transportation System Based on the Theory of Image Recognition*, Jilin University, Changchun, China, 2016.

[3] B. Wei, *Modeling and Simulation Research on Vehicular Ad Hoc Networks of Intelligent Transportation System Based on Internet of Things*, Lanzhou Jiaotong University, Lanzhou, China, 2017.

[4] X.-H. Liu, D.-G. Zhang, H.-R. Yan, Y.-Y. Cui, and L. Chen, "A new algorithm of the best path selection based on machine learning," *IEEE Access*, vol. 7, pp. 126913–126928, 2019.

[5] H. Zhang and X. Lu, "Vehicle communication network in intelligent transportation system based on Internet of Things," *Computer Communications*, vol. 160, no. 160, pp. 799–806, 2020.

[6] Y. S. Chen, *Research on Decision-Making Method of Driving Behavior in Urban Complex Traffic Situation of Intelligent Automobile*, Jilin University, Changchun, China, 2019.

[7] F. Zong, M. Zeng, W. Zhong, and F. Lu, "Hybrid path selection modeling by considering habits and traffic conditions," *IEEE Access*, vol. 7, pp. 43781–43794, 2019.

[8] F. Hong, *Research on Comprehensive Cognition of Traffic Vehicles and Virtual Test for Intelligent Vehicle*, Jilin University, Changchun, China, 2018.

[9] B. Hou, Z. He, H. Zhou, and J. Wang, "Integrated design and accuracy analysis of star sensor and gyro on the same benchmark for satellite attitude determination system," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 1074–1080, 2019.

[10] J. Wang, Y. Sun, Z. Zhang, and S. Gao, "Solving multitrip pickup and delivery problem with time windows and manpower planning using multiobjective algorithms," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 4, pp. 1134–1153, 2020.

[11] H. Geng, *Research on Path Optimization of Distributed Dynamic Route Guidance System of Intelligent Transportation Syste*, Lanzhou Jiaotong University, Lanzhou, China, 2016.

[12] P. Liu, *Research on Modeling and Optimization Algorithm of Dynamic Route Guidance System in Intelligent Transportation System*, Jilin University, Changchun, China, 2017.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[14] Y. L. Chen, L. Y. Zhuang, L. B. Zhu et al., "Research on path planning of parking system based on the improved Dijkstra algorithm," *Modern Manufacturing Engineering*, vol. 8, pp. 63–67, 2017.

[15] Y. W. Liu and H. Q. Wu, "Path planning based on theoretical shortest distance variable weight A* algorithm," *Computer Measurement and Control*, vol. 26, no. 4, pp. 175–178, 2018.

[16] X. F. Zuo and W. J. Shen, "Improved algorithm about mutishortest path problem based on Floyd algorithm," *Computer Science*, vol. 44, no. 5, pp. 232–267, 2017.

[17] K. Heesu, H. K. Sang, J. Maro et al., "A study on path optimization method of an unmanned surface vehicle under environmental loads using genetic algorithm," *Ocean Engineering*, vol. 142, pp. 616–624, 2017.

[18] S. X. Wang and Z. X. Wu, "Improved Dijkstra shortest path algorithm and its application," *Computer Science*, vol. 39, no. 5, pp. 223–228, 2012.

[19] Y. L. Liu, Y. Li, J. L. Wu et al., "Route planning of expressway emergency evacuation based on improved Dijkstra algorithm," *Transport Research*, vol. 2, no. 6, pp. 54–66, 2016.

[20] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[21] Y. Song and Z. M. Wang, "Path planning simulation based on improved A* algorithm," *Journal of Changchun University of Technology*, vol. 40, no. 2, pp. 138–141, 2019.

[22] X. Meng, J. Li, M. Zhou, X. Dai, and J. Dou, "Population-based incremental learning algorithm for a serial colored traveling salesman problem," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 2, pp. 277–288, 2018.

[23] X. P. Xu, J. Li, and M. C. Zhou, "Delaunay-triangulation-based variable neighborhood search to solve large-scale general colored traveling salesman problems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, pp. 1–11, 2020.

[24] Y. F. Ge, T. Chen, X. Y. Kong et al., "Application of improved ant colony algorithm in car navigation," *Control Engineering of China*, vol. 23, no. 1, pp. 133–137, 2016.

[25] Z. Y. Shang, J. N. Gu, and J. P. Wang, "An improved simulated annealing algorithm for the capacitated vehicle routing problem," *Computer Integrated Manufacturing Systems*, vol. 13, 2020.

[26] M. A. Mohammed, M. K. Abd Ghani, R. I. Hamed, S. A. Mostafa, M. S. Ahmad, and D. A. Ibrahim, "Solving vehicle routing problem by using improved genetic algorithm for optimal solution," *Journal of Computational Science*, vol. 21, pp. 255–262, 2017.

[27] W. Y. Zhi, "Improved GA with particle Swarm's evolutionary strategy for solving constrained optimization problems," *Control and Decision*, vol. 5, 2012.

[28] Y. Kao, M.-H. Chen, Y.-T. Huang et al., "A hybrid algorithm based on ACO and PSO for capacitated vehicle routing problems," *Mathematical Problems in Engineering*, vol. 2012, Article ID 726564, 17 pages, 2012.

[29] Y. F. Ma, F. Yan, K. Kang et al., "An improved particle swarm optimization for simultaneous pickup and delivery vehicle routing problems with time windows under a fuzzy random environment," *Operations Research and Management Science*, vol. 27, no. 12, pp. 73–83, 2018.

[30] J. Zhang, F. Yang, X. Weng et al., "An evolutionary scatter search particle swarm optimization algorithm for the vehicle routing problem with time windows," *IEEE Access*, vol. 6, pp. 63468–63485, 2018.

[31] L. Wang and J. Lu, "A memetic algorithm with competition for the capacitated green vehicle routing problem," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 2, pp. 516–526, 2019.

[32] S. H. Li, J. Q. Sun, and Z. Yang, "Key technologies of car navigation service recommendation system based on context awareness," in *Proceedings of the 31st Chinese Control Conference*, pp. 7298–7303, Hefei, Anhui, July 2012.

[33] R. Jiang, *Least Travel Time Paths in Stochastic and Time-Varying Transportation Networks Considering the Time Consumption of Nodes*, Tianjin University of Technology, Tianjin, China, 2016.

[34] P. A. Mohammad, H. A. Mohammad, and B. Mehdi, "Application of GA, PSO, and ACO algorithms to path planning of autonomous underwater vehicles," *Journal of Marine Science and Application*, vol. 11, no. 3, pp. 492–496, 2012.