Received January 18, 2018, accepted February 13, 2018, date of publication February 21, 2018, date of current version March 16, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2808407

IBAS: Index Based A-Star

YAN LI[1], HONGYAN ZHANG[1], HUAIZHONG ZHU[2,3], JIANWEI LI[2,3], WENJIE YAN[2,3], AND YOUXI WU[2,3], (Member, IEEE)
[1]School of Economics and Management, Hebei University of Technology, Tianjin 300401, China
[2]School of Computer Science and Engineering, Hebei University of Technology, Tianjin 300401, China
[3]Hebei Province Key Laboratory of Big Data Calculation, Tianjin 300401, China

Corresponding author: Youxi Wu (wuc@scse.hebut.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61673159 and Grant 61702157, and in part by the Graduate Student Innovation Program of Hebei Province under Grant CXZZSS2017031.

ABSTRACT The A-star algorithm is an efficient classical algorithm for solving the shortest path problem. The efficiency of the algorithm depends on the evaluation function, which is used to estimate the heuristic value of the shortest path from the current vertex to the target. When the vertex coordinates are known, the heuristic value of the shortest path is usually generated by the distance. In this paper, we present an Index Based A-Star algorithm (IBAS), which aims to solve the shortest path problem in a weighted directed acyclic graph with unknown vertex coordinates. This paper constructs three indexes for each vertex, i.e., the earliest arrival index, reverse earliest arrival index, and latest arrival index. We can compute the lower bound and the upper bound of the shortest distance from the source vertex to the target based on the three indexes and prune the intermediate vertices which are not in shortest path according to the lower and upper bounds. The IBAS algorithm not only makes use of the earliest arrival index to construct the evaluation function of the A-star algorithm but also utilizes the three indexes to prune useless vertices, so as to improve the performance of the algorithm. Compared with the A-star algorithm, the additional time complexity and space complexity of the IBAS algorithm are $O(|V| + |E|)$ and $O(|V|)$, respectively. A real road network and benchmark datasets with large-scale network are selected to verify the performance of IBAS. Experimental results verify the effectiveness of the proposed algorithm.

INDEX TERMS Shortest path, A-star algorithm, pruning strategy, index.

I. INTRODUCTION

The shortest path problem is a classical problem [1], [2], but is also a tough issue especially in large-scale network [3], [4]. It appears in many practical applications, such as transportation networks [5]–[7], isometric feature mapping [8], biological networks analysis [9], subgraph similarity matching [10], pattern mining [11]–[13], RDF clustering [14], and social networks [15]. Motivated by these applications, a variety of shortest path problems have been investigated. Wang *et al.* [16] focused on solving approximate constrained shortest path queries. Yuan *et al.* [17] dealt with k nearest object search on road networks by incorporating social influence. Zhao *et al.* [18] dealt with the problem of an online shortest path computation. Abraham *et al.* [19] investigated the shortest path problem in a high-dimensional space. Sedeno-Noda and Raith [20] considered the bi-objective shortest path problem. Zhu *et al.* [21] constructed an approximate solution to the shortest path in a network. Gao *et al.* [22] focused on a dynamic shortest path algorithm in a hypergraph environment. Nip *et al.* [23] exploited a number of combination problems involving the shortest path. Hong *et al.* [24] studied the shortest path problem for large-scale dynamic graphs. However, the above studies have all focused on setting up effective algorithms for specific problems, but with the growth of the scale of the graphs it is becoming increasingly important to have a fast shortest path solving method for large-scale graphs. A feasible method is to employ a reachable index to quickly identify unreachable vertices in Directed Acyclic Graphs (DAG) [25], and then prune them in order to reduce the scale of the graph [26], [27]. In studies on the reachability query problem, Yildirim *et al.* [28] proposed the GRAIL algorithm, which can reduce the error rate via adding two additional interval labels L_1 and L_2 for each vertex by different depth-first traversal. Cheng *et al.* [29] proposed a k-reach algorithm by using the minimum vertex cover set to solve k-step reachability queries in DAG for any two vertices. Zhou *et al.* [30] proposed the BiRch algorithm, based on the GRAIL algorithm, which further improved the query performance. Our previous studies [31], [32] investigated the number of simple paths and the number of disjoint paths between two vertices by using a special data structure nettree. These studies all concerned k-step reachability based on unweighted graphs. Nevertheless, in practice it is necessary to consider time or cost constraints, thus forming weighted graphs. Existing methods fail to solve this problem.

In addition, the A-star algorithm is an efficient search algorithm, and its evaluation function can be denoted as:

$$f(n) = dis(u, n) + h(n) \tag{1}$$

where $f(n)$ is the evaluation function, $dis(u, n)$ is the shortest path value from the source vertex u to the current vertex n, and $h(n)$ is the heuristic value that estimates the value of the shortest path from the current vertex n to the target. The selection of $h(n)$, which needs to be estimated in advance, significantly influences the efficiency of the A-star algorithm. Although there exist a variety of computing methods, such as the Manhattan distance, diagonal distance, and Euclidean distance, these are all established with known vertex coordinates. In the case with unknown vertex coordinates, the problem of determining $h(n)$ requires urgent attention. In addition, in the process of computing the shortest path, if we can prune some useless vertices in advance this will significantly boost the solving speed of the shortest path problem. The main contributions of this study are threefold:

1) We propose methods for constructing the earliest arrival index, reverse earliest arrival index, and latest arrival index in a weighted DAG.

2) We present an index-based A-star algorithm, IBAS, which uses the earliest arrival index to construct the evaluation function of the A-star algorithm, and also employs the above three indexes to achieve effective pruning.

3) Through extensive experiments, we demonstrate that IBAS can effectively boost the speed of traditional algorithms, and the efficiency of the reachability cost index is high.

The rest of this paper is organized as follows. Section II introduces the related works. Section III presents the detailed construction algorithms of the three indexes, theoretically proves the feasibility of the pruning algorithm, introduces the principle of IBAS through an example, and finally presents the IBAS algorithm. Also the additional time and space complexities are analyzed in this section. Section IV introduces the experimental data and presents the analysis results. Conclusions are discussed in Section V.

II. RELATED WORKS

The shortest path problem is a classical graph theory problem. At present, there are many solving algorithms, among which the most classical ones are Dijkstra algorithm [33], Floyd algorithm [34], and A-star algorithm [35]. The Dijkstra algorithm was put forward by Dijkstra in 1959 and the principle is that it starts from the source vertex and spreads from layer to layer until it reaches the target. Because each time it is necessary for the algorithm to identify all vertices, the time complexity of each identification is $O(|V|)$. Therefore, the time complexity of the Dijkstra algorithm is $O(|V|^2)$. The Floyd algorithm uses dynamic programming to solve the shortest path problem. The shortest path matrix between two vertices is obtained by the weight matrix of a graph, and the time complexity of the algorithm is $O(|V|^3)$. The A-star algorithm is the most effective direct search method for static networks shortest path problem. It adopts the heuristic search strategy to obtain the best position by using the evaluation function, which avoids a lot of useless search path, so as to improve the search efficiency. As mentioned above, the accuracy of the evaluation function has a direct impact on the efficiency of the A-star algorithm.

In recent years, some more efficient algorithms from different views have been proposed, such as parallel processing method [36], large graph processing method [37], hardware accelerated method [38], and the pretreatment of the metric backbone based method [39]. Obviously, these methods are with known network topology.

With the emergence of electronic maps such as Google Maps and Baidu Maps, a variety of studies, for example, a quick query based on spatial mashups [40], a new path planning based on historical path caching [41], have been carried out in order to realize the search of the shortest path from the source to the destination on electronic map. Zhang *et al.* [2] realized the search of the shortest path by cloud-based mapping services with path sharing and path caching, without knowing the underlying graph structure or network topology, although only the straight lines for path sharing are considered. Therefore, for the non-spatial graph, the shortest path solving methods still need to be explored.

Similar studies [28]–[32] used the interval index for reachability queries in the unweighted DAG graph, which aimed to determine whether it can be reached in k step from the source vertex to the target. These studies accelerate the identification of the unreachable vertices by constructing a certain number of indexes in advance, so as to improve the query efficiency. This paper establishes three indexes in a weighted DAG, and prunes the intermediate vertice which are not in shortest path by the indexes, thus effectively avoiding unnecessary search, so as to realize the fast computation of the shortest path.

III. METHOD FOR SOLVING THE SHORTEST PATH PROBLEM

This paper presents three indexes: the earliest arrival index $E(u)$, reverse earliest arrival index $R(u)$, and latest arrival index $L(u)$. These three indexes are used to implement an efficient strategy for pruning useless vertices, so as to reduce the scale of DAG. The earliest arrival index is also used to estimate the evaluation function of the A-star algorithm. Finally, a fast search of the shortest path in a large-scale DAG is achieved, and the computational efficiency of the algorithm is improved.

A. CONSTRUCTION OF THREE INDEXES

This paper constructs the earliest arrival index $E(u)$, reverse earliest arrival index $R(u)$, and latest arrival index $L(u)$ for each vertex u.

Suppose $G = (V, E, W)$ is a DAG, where V is a set of vertices, $E \subseteq V * V$ is a set of arcs, and W is a set of arc weights. For any two vertices u and v in G, the shortest path between them is denoted by $dis(u, v)$.

Definition 1: For any two vertices u and v in G, if there exists a path from vertex u to v, then u can reach v, which is denoted by $u \rightarrow v$*. Otherwise, u cannot reach v. If there exists a reachable path between u and v within the distance C, namely the total path distance between u and v is less than or equal to C, then vertex u can reach vertex v within cost C, which is denoted by* $u \underset{C}{\rightarrow} v$*. Otherwise, u cannot reach v within cost C.*

Definition 2: The earliest arrival index $E(u)$ *refers to the minimum cost from the 0 in-degree vertex to vertex u.* $E(u)$ *is equal to 0 if the in-degree of u is 0, otherwise* $E(u)$ *equals the minimum of the sum of all its in-degree vertices* $E(t)$ *and the weights of the corresponding arcs. The method of calculating* $E(u)$ *is as follows:*

$$E(u) = \begin{cases} 0 & \text{the in-degree of } u \text{ is } 0 \\ min\{E(t) + c\} & \text{otherwise} \end{cases} \tag{2}$$

where t is the parent vertex of u, c is the arc weight.

Definition 3: The reverse earliest arrival index $R(u)$ *refers to the reverse minimum cost from the 0 out-degree vertex to vertex u.* $R(u)$ *is equal to 0 if the out-degree of u is 0, otherwise* $R(u)$ *equals the minimum of the sum of all its out-degree vertices* $R(t)$ *and the weights of the corresponding arcs. The method for calculating* $R(u)$ *is as follows:*

$$R(u) = \begin{cases} 0 & \text{the out-degree of } u \text{ is } 0 \\ min\{R(t) + c\} & \text{otherwise} \end{cases} \tag{3}$$

where t is the child vertex of u, c is the arc weight.

Definition 4: The latest arrival index $L(u)$ *refers to the maximum cost from the 0 in-degree vertex to vertex u.* $L(u)$ *is equal to 0 if the in-degree of u is 0, otherwise* $L(u)$ *equals the maximum of the sum of all its in-degree vertices* $L(t)$ *and the weights of the corresponding arcs. The method for calculating* $L(u)$ *is as follows:*

$$L(u) = \begin{cases} 0 & \text{the in-degree of } u \text{ is } 0 \\ max\{L(t) + c\} & \text{otherwise} \end{cases} \tag{4}$$

where t is the parent vertex of u, c is the arc weight.

Fig. 1 is used as an example to illustrate how the three indexes are calculated.

Example 1: The earliest arrival index values of vertices 1, 11, and 20 with 0 in-degree are $E(1) = E(11) = E(20) = 0$*. Therefore, the earliest arrival index value of vertex 2 is* $E(2) = 0+2 = 2$*. Vertex 4 has two parent vertices, vertices 3 and 12. Because their earliest arrival index values are* $E(12) = 4$*,* $E(3) = 4$*, and* $E(12) + 2 = 4 + 2 = 6 < E(3) + 4 = 4 + 4 = 8$*,* $E(4)$ *takes the minimum value of 6. Similarly, vertices with 0 out-degree are 6, 9, and 18, so* $R(6) = R(9) = R(18) = 0$*. Hence, the reverse earliest arrival index value of vertex 8 is* $R(8) = 0 + 3 = 3$*. Vertex 5 has three child vertices, 6, 7, and 10. Because* $R(6) = 0$*,* $R(7) = 5$*,* $R(10) = 8$*, and* $R(6) + 6 = 0 + 6 = 6 < R(7) + 2 = 5 + 2 = 7 < R(10) + 3 = 8 + 3 = 11$*,* $R(5)$ *takes the minimum value of 6. Similarly, the latest arrival index values of vertices 1, 11, and 20 with 0 in-degree are* $L(1) = L(11) = L(20) = 0$*. Hence, the latest arrival index value of vertex 2 is* $L(2) = 0 + 2 = 2$*. For vertex 4, the latest arrival index value of its two parent vertices are* $L(12) = 4$ *and* $L(3) = 4$*, respectively. Because* $L(3)+4 = 4+4 = 8 > L(12) + 2 = 4 + 2 = 6$*,* $L(4)$ *takes the maximum value of 8. The above method can be used to calculate the three index values of all vertices in Fig. 1.*

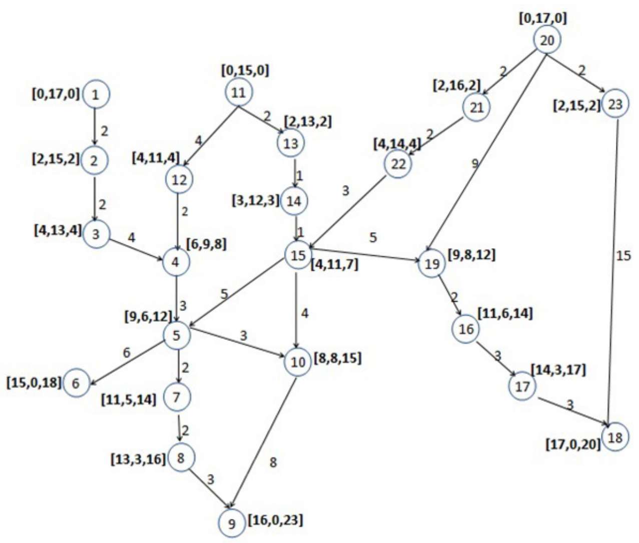


FIGURE 1. Values of the three indexes.

The construction algorithm of the three indexes is shown in Algorithm 1.

Algorithm 1 Construction Algorithm of the Three Indexes

Input: DAG adjacency matrix $w[i][j](0 < i, j < n)$;
Output: $E(u)$, $R(u)$, $L(u)$;

```
1:  for u=1 to n do
2:      if in-degree(u)=0 then // Compute E(u) and L(u) according to formula 2 and formula 4, respectively;
3:          $E(u)$=0;
4:          $L(u)$=0;
5:      else
6:          $E(u)$=$min(E(t) + w[t][u])$;
7:          $L(u)$=$max(L(t) + w[t][u])$;
8:      end if
9:      if out-degree(u)=0 then // Compute R(u) according to formula 3;
10:         $R(u)$=0;
11:     else
12:         $R(u)$=$min(R(t) + w[u][t])$;
13:     end if
14: end for
15: return $E(u)$, $R(u)$, $L(u)$
```

In the computation process, it is necessary to ensure that each vertex is marked only once.

B. PRUNING ALGORITHM

The working principle of the pruning algorithm is given as follows. We can calculate the range of the shortest distance $dis(u, v)$ based on the following three theorems. Theorems 1 and 2 give the lower bound of $dis(u, v)$, and Theorem 3 gives the upper bound of $dis(u, v)$. Although we propose two theorems, Theorem 1 and Theorem 2, to calculate the lower bound of $dis(u, v)$, Theorem 5 will show how to apply the two theorems to prune the intermediate vertex t, which must not be on the shortest path from u to v.

Theorem 1: The lower bound of $dis(u, v)$ *can be found as* $E(v) - E(u)$*; that is,* $E(v) - E(u) \leq dis(u, v)$.

Proof: The proof employs reduction to absurdity. Assume that $dis(u, v) < E(v) - E(u)$. Definition 2 implies that the minimum cost from the 0 in-degree vertex to u is $E(u)$. Therefore, the minimum cost of reaching v from the 0 in-degree vertex through u is $E(u) + dis(u, v) < E(u) + E(v) - E(u) = E(v)$. That is, the minimum cost of reaching v through u is less than $E(v)$. However, this is a contradiction of Definition 2. Thus, the proof is complete. ■

Theorem 2: The lower bound of $dis(u, v)$ *can be given as* $R(u) - R(v)$*; that is,* $R(u) - R(v) \leq dis(u, v)$.

Proof: The proof employs reduction to absurdity. Assume that $dis(u, v) < R(u) - R(v)$. Definition 3 implies that the minimum cost from the 0 out-degree vertex to v is $R(v)$. Therefore, the reverse minimum cost of reaching u from the 0 out-degree vertex through v is $R(v) + dis(u, v) < R(v) + R(u) - R(v) = R(u)$. That is, the reverse minimum cost of reaching u through v is less than $R(u)$. However, this is a contradiction of Definition 3. Thus, the proof is complete. ■

Theorem 3: The upper bound of $dis(u, v)$ *should be* $L(v) - L(u)$*; that is,* $dis(u, v) \leq L(v) - L(u)$.

Proof: The proof employs reduction to absurdity. Assume that $dis(u, v) > L(v) - L(u)$. Definition 4 implies that the maximum cost from the 0 in-degree vertex to u is $L(u)$. Therefore, the maximum cost of reaching v from the 0 in-degree vertex through u is $L(u)+dis(u, v) > L(u)+L(v)-L(u) = L(v)$. That is, the cost of reaching v through u is greater than $L(v)$. However, this is a contradiction of Definition 4. Thus, the proof is complete. ■

Theorem 4: If $E(t) = 0$ *and* t *is not* u*, or* $R(t) = 0$ *and* t *is not* v*, then* t *can be pruned.*

Proof: $E(t) = 0$ shows that the in-degree of vertex t is 0. Because t is not u, there is no path pointing to t in the graph, and t can be pruned. Similarly, $R(t) = 0$ means that the out-degree of vertex t is 0 and t is not v, so there is no path t pointing to. The proof is complete. ■

Apparently, for any node t, according to Theorem 1 and Theorem 2, we can calculate four values d_1, d_2, d_3, and d_4 shown in Theorem 5. Therefore, we can draw four circles with centers u and v and with the radius of the four values. If node t locates outside the four circles, which means the lower bound of the distance from u to t or from t to v is greater than the upper bound of the distance from u to v. Therefore vertex t cannot be an intermediate vertex on the shortest path from u to v. Hence node t can be safely removed. Theorem 5 will show the correctness of this strategy.

Theorem 5: Let $d = L(v)-L(u)$*. For a vertex* t *in a graph, let* $d_1 = E(t)-E(u)$*,* $d_2 = E(v)-E(t)$*,* $d_3 = R(u)-R(t)$*, and* $d_4 = R(t) - R(v)$*. If* $d_1 > d$ *or* $d_2 > d$ *or* $d_3 > d$ *or* $d_4 > d$*, then the vertex* t *can be pruned.*

Proof: The proof employs reduction to absurdity. If the shortest path from vertex u to v passes t, then from Theorem 3 we know that $dis(u, t) + dis(t, v) = dis(u, v) \leq d$. However, from Theorems 1 and 2 we know that $d_1 \leq dis(u, t)$, $d_2 \leq dis(t, v)$, $d_3 \leq dis(u, t)$, and $d_4 \leq dis(t, v)$. Therefore, $dis(u, t) + dis(t, v) \geq d_1 + d_2$. If $d_1 > d$ or $d_2 > d$, then it is obvious that $d_1 + d_2 > d$; that is, $dis(u, t) + dis(t, v) > d$. However, this contradicts the fact that $dis(u, t) + dis(t, v) = dis(u, v) \leq d$. Similarly, $dis(u, t) + dis(t, v) \geq d_3 + d_4 > d$ is also a contradiction of the fact. Therefore, if $d_1 > d$ or $d_2 > d$ or $d_3 > d$ or $d_4 > d$, then the shortest path from u to v must not pass t, and the vertex t can be pruned. The proof is complete. ■

Example 2 shows the pruning strategy.

Example 2: The vertices from 20 to 15 in Fig. 1 are used as an example to illustrate the pruning method. From Theorems 1, 2, and 3 we know that $dis(20, 15) \leq L(15) - L(20) = 7 - 0 = 7$*,* $dis(20, 15) \geq E(15) - E(20) = 4 - 0 = 4$*, and* $dis(20, 15) \geq R(20) - R(15) = 17 - 14 = 3$*. Consequently,* $dis(20, 15)$ *ranges from 4 to 7. According to Theorem 4, because the either E value or R value of each vertex 1, 6, 9, 11, and 18 is 0, they can be pruned. In addition, according to Theorem 5 vertex 19 can be pruned, because* $d_1 = E(19) - E(20) = 9 - 0 = 9 > 7$*. However, vertex 14 cannot be pruned, because* $d_1 = 3 < 7$*,* $d_2 = d_4 = 1 < 7$*, and* $d_3 = 5 < 7$*. Similarly, according to Theorem 5 we can further prune vertices 4, 5, 7, 8, 10, 16, and 17.*

Now, we show the pruning algorithm in Algorithm 2.

Algorithm 2 Pruning Algorithm

```
Input: Indexes $E(u)$, $R(u)$, $L(u)$, vertices $u$ and $v$;
Output: pruned;
1:  $d = L(u) - L(v)$;
2:  for $t$=1 to $|V|$ do
3:      if ($E(t) = 0$ and $t != u$) or ($R(t) = 0$ and $t != v$) then
4:          $pruned(t)$=true; //Prune $t$ according to Theorem 4;
5:      end if
6:      if $E(t) - E(u) > d$ or $E(v) - E(t) > d$ or $R(u) - R(t) > d$ or $R(t) - R(v) > d$ then
7:          $pruned(t)$=true; //Prune $t$ according to Theorem 5;
8:      end if
9:  end for
10: return pruned
```

C. IBAS ALGORITHM

Theorem 6: Let the shortest path from u to r be $dis(u, r)$*. For the vertex t in a graph, if* $(r, t) \in W$*, then let the weight of* (r, t) *be w. If* $dis(u, r)+w+d_2 > d$ *or* $dis(u, r)+w+d_4 > d$*, then the vertex t can be pruned.*

Proof: The proof employs reduction to absurdity. From Theorem 1 we know that $d_2 \leq dis(t, v)$. Suppose the shortest path from vertex u to v passes t. From Theorem 3, we know that $dis(u, r) + w + d_2 \leq dis(u, r) + w + dis(t, v) = dis(u, v) \leq d$. Therefore, if $dis(u, r) + w + d_2 > d$ or $dis(u, r)+w+d_4 > d$, then the shortest path from u to v definitely does not pass t, and t can be pruned. The proof is complete. ■

Illustrative examples are shown in Example 3 and Example 4.

Example 3: Fig. 1 shows an example to illustrate the computation process of the IBAS algorithm. In the computation of the shortest path from vertex 15 to 6, $L(u)$*,* $E(u)$*, and* $R(u)$ *are used to calculate the path distance constraint values* $d = L(6)-L(15) = 18-7 = 11$*,* $E(6)-E(15) = 15-4 = 11$*, and* $R(15) - R(6) = 11 - 0 = 11$*. Consequently,* $dis(15, 6)$ *is 11. According to algorithm 2, twelve vertices can be pruned, such as vertices 1, 2, 3, 9, 12, and so on. Based on the above pruning result, the IBAS algorithm dynamically prunes vertices 10, 19, and 7 according to Theorem 6 and obtain the shortest path sequence* $15 \rightarrow 5 \rightarrow 6$*, whose distance is 11. In this computation process, it is only necessary to update 10 vertices. However, when computing the shortest path on the original network, the shortest path sequence is still* $15 \rightarrow 5 \rightarrow 6$*, the path distance is also 11, and 18 vertices need to be updated in the computation process. Thus, it is clear that the IBAS algorithm can effectively reduce the computational complexity and improve the computational efficiency.*

Example 4: The vertices from 23 to 9 in Fig. 1 are selected to illustrate the case which there is no path between them. Vertex 18 and other vertices are pruned according to Theorem 4. After pruning vertex 18, the out-degree of vertex 23 is 0. Therefore, IBAS can know that there is no path from 23 to 9 without further calculation.

After applying the pruning algorithm in DAG, Algorithm 3 is used to compute the shortest path.

Algorithm 3 IBAS Algorithm

Input: DAG adjacency matrix $w[i][j](0 < i, j < n)$, vertices u and v;

Output: the shortest path $dis(u, v)$;

```
1: Call algorithm 1 to achieve $E(u)$, $R(u)$, $L(u)$;
2: Call algorithm 2 to realize static pruning, so as to reduce the scale of DAG;
3: Put $u$ into OPEN table;
4: while (OPEN!=NULL) do
5:     Get vertex $r$ from OPEN table;
6:     Get vertex $t$ which is the neighbor of $r$;
7:     if $pruned(t)$ =false then
8:         if $dis(u, r) + w[r][t] + E(v) - E(t) > d$ or $dis(u, r) + w[r][t] + R(t) - R(v) > d$ then
9:             $pruned(t)$=true;
10:        else
11:            Calculate $dis(u, v)$ according to A-star algorithm;
12:        end if
13:    end if
14: end while
15: return the shortest path $dis(u, v)$;
```

D. COMPLEXITY ANALYSIS

Because the three indexes $E(u)$, $R(u)$, and $L(u)$ are constructed for each vertex in the graph, the additional space complexity of the IBAS algorithm is $O(|V|)$. In the process of construction of $E(u)$, it is necessary to compute each edge, so the time complexity of the construction of $E(u)$ is $O(|V| + |E|)$, and the time complexities of the construction of other two indexes are the same as $E(u)$. Algorithm 2 conducts the pruning strategy for all vertices, so its time complexity is $O(|V|)$. The time complexity of the third line of IBAS is same as that of A-star, since the time complexity of Theorem 6 is $O(1)$. Thus, compared with A-star, the additional time complexity of the IBAS algorithm is $O(|V|+|E|)$.

IV. EXPERIMENTAL DATA AND RESULTS ANALYSIS

In order to verify the effectiveness of the proposed algorithm, a real road network data set and benchmark data sets are selected. We also contrast the IBAS algorithm with the Dijkstra algorithm without a pruning strategy, the Floyd algorithm without a pruning strategy, the A-star algorithm using $E(u)$ to represent its h function, the Dijkstra algorithm with a pruning strategy according to Theorem 5 (i.e., index-based Dijkstra algorithm, IBD), and the Floyd algorithm with a pruning strategy according to Theorem 5 (i.e., index-based Floyd algorithm, IBF). In order to further verify the effect of dynamic pruning, we also perform the IBAS-S algorithm with only static pruning, which does not use the if-then structure of line 8 of IBAS. To further verify the performance of the three indexes, we propose a new algorithm, named IBAS-M, which uses both the three indexes and the two index intervals, $L_u[x, y]$ and $R_u[x, y]$, proposed in[28] and [30] to prune the vertices. Since [28] and [30] focused on reachability query in unweighted DAG, we *set al*l arc weights as 1 to get the corresponding unweighted DAG at first. Then we can obtain the two intervals in the new DAG according to the pre-order and post-order traversal strategy.

All of the above algorithms are written in VC++6.0. All experiments are conducted on a laptop with an Intel (R) Core i7-5500U, a 2.40-GHZ CPU, 4.00 GB of RAM, and Windows 7 SP1 operating system.

A. EXPERIMENTAL DATA DESCRIPTION

1) REAL ROAD NETWORK DATA

In this paper, the Xi'an Yanta District road network from [42] is used to demonstrate the performance of the proposed

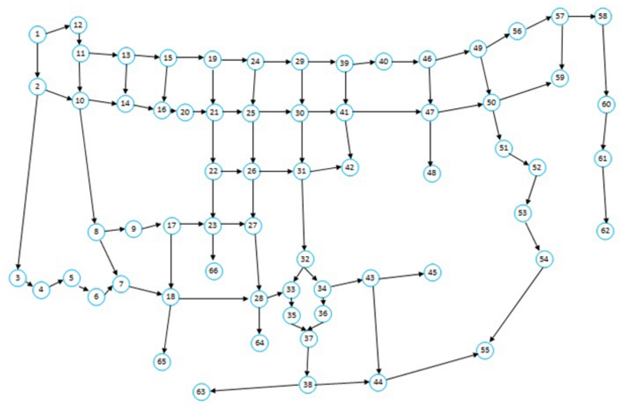


FIGURE 2. A real road network.

IBAS algorithm. The network contains 66 vertices without coordinates, as shown in Fig. 2. Limited to the length, we omit the distance of each path. The path distance can be downloaded from the Internet, and here is taken as the arc weight, thus forming a weighted DAG. In this DAG, we ignore the vertex coordinates, and only preserve the path distance between each vertex.

We extract the vertex pairs (19,37), (10,42), (8,38), (31,55), (19,42), (10,50), (13,48), (19,51), (21,45), (24,33), (24,52), and (11,28) from Fig. 2 to verify the algorithm performance.

2) BENCHMARK DATA

To further verify the performance of the proposed algorithm, we use the medium and large-scale data provided by http://algs4.cs.princeton.edu/44sp/, which is shown in Table 1.

TABLE 1. Testing datasets.

Name	Number of vertices	Number of edges
mediumEWD	250	2546
1000EWG	1000	16866
10000EWG	10000	123462
largeEWG	one million	15172126

We randomly generate 10,000 vertex pairs to verify the algorithm performance.

B. EXPERIMENTAL RESULTS

1) REAL DATA EXPERIMENTAL RESULTS

Tables 2 and 3 compare the number of pruned vertices and indexes construction time.

To facilitate analysis, in this paper, two parameters, the number of updated vertices and the number of loops of each algorithm, are selected. The two parameters in Dijkstra, Floyd, IBD, and IBF refer to the number of vertices checked in solving the problem and the number of loops in which at least a vertex is checked, respectively. As we know, the time complexity of A-star, IBAS-S, IBAS-M, and IBAS depends on the number loops of line 4 which is the number of vertices put into the OPEN table. Therefore, both two parameters in A-star, IBAS-S, IBAS-M, and IBAS are the same. Hence,

TABLE 2. Comparison of number of pruned vertices.

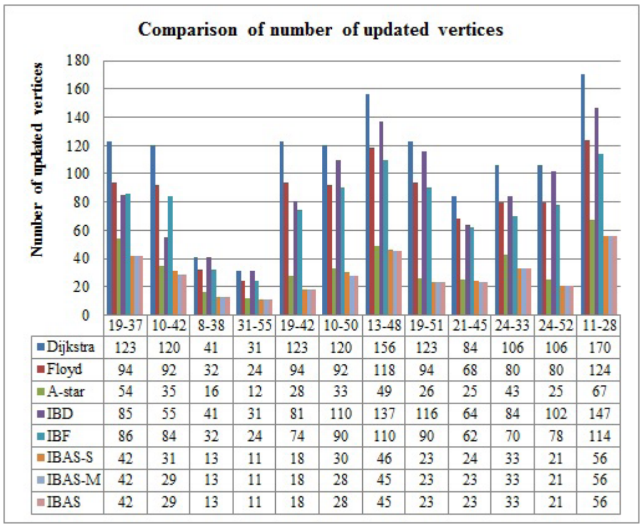
Source vertex	Target vertex	IBAS-M	IBAS
19	37	23	15
10	42	39	26
8	38	43	12
31	55	52	16
19	42	50	39
10	50	34	22
13	48	28	17
19	51	33	22
21	45	44	31
24	33	37	15
24	52	38	24
11	28	21	15

TABLE 3. Comparison of indexes construction time (ms).

Indexes construction time for IBAS-M	Indexes construction time for IBAS
6.56	3.56

TABLE 4. Comparison of the average running time (ms).

Dijkstra	Floyd	A-star	IBD	IBF	IBAS-S	IBAS-M	IBAS
120.6	116.6	42.3	106.5	99.4	38.7	37.6	35.8



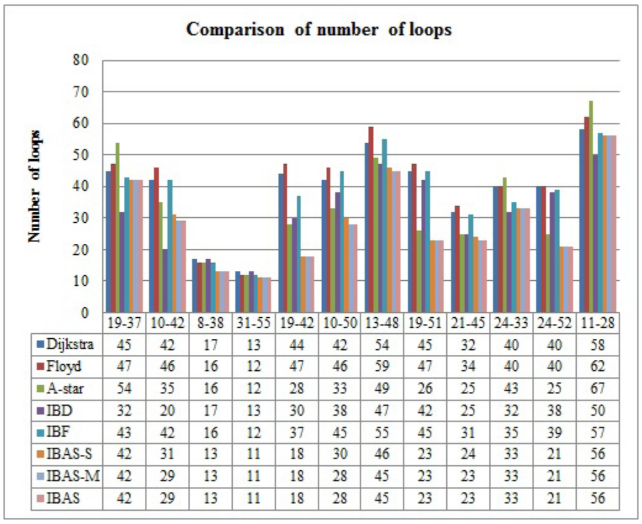
	19-37	10-42	8-38	31-55	19-42	10-50	13-48	19-51	21-45	24-33	24-52	11-28
🟦 Dijkstra	123	120	41	31	123	120	156	123	84	106	106	170
🟥 Floyd	94	92	32	24	94	92	118	94	68	80	80	124
🟩 A-star	54	35	16	12	28	33	49	26	25	43	25	67
🟪 IBD	85	55	41	31	81	110	137	116	64	84	102	147
🟦 IBF	86	84	32	24	74	90	110	90	62	70	78	114
🟧 IBAS-S	42	31	13	11	18	30	46	23	24	33	21	56
🟦 IBAS-M	42	29	13	11	18	28	45	23	23	33	21	56
🟥 IBAS	42	29	13	11	18	28	45	23	23	33	21	56

FIGURE 3. Comparison of number of updated vertices.

the two parameters are comparable in the similar algorithms, otherwise they are not comparable, since the principles of Dijkstra, Floyd, and A-star are different. Apparently, the less number of updated vertices is, the more effective the algorithm is. Figs. 3 and 4 illustrate the number of updated vertices and the number of loops of each algorithm, respectively. Fig. 5 and Table 4 demonstrate the running time and the average running time of each algorithm, respectively.

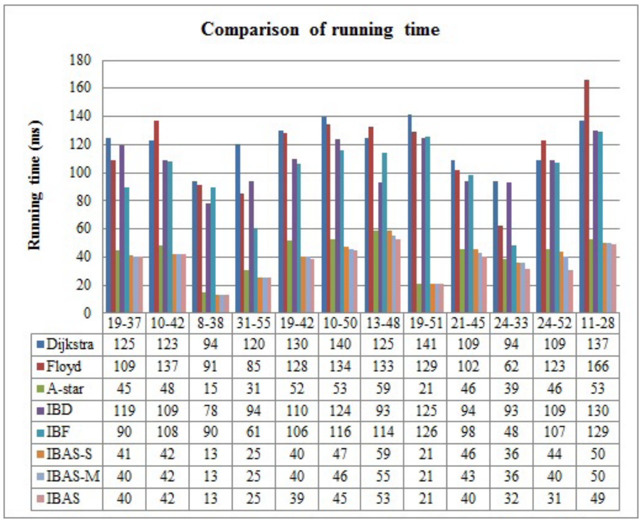
2) BENCHMARK DATA EXPERIMENTAL RESULTS

Table 5 lists the construction time and size of the indexes on the benchmark data. The construction time is based on the average value of 50 computations. It can be seen that the index size grows in line with the number of vertices in the graph,



	19-37	10-42	8-38	31-55	19-42	10-50	13-48	19-51	21-45	24-33	24-52	11-28
Dijkstra	45	42	17	13	44	42	54	45	32	40	40	58
Floyd	47	46	16	12	47	46	59	47	34	40	40	62
A-star	54	35	16	12	28	33	49	26	25	43	25	67
IBD	32	20	17	13	30	38	47	42	25	32	38	50
IBF	43	42	16	12	37	45	55	45	31	35	39	57
IBAS-S	42	31	13	11	18	30	46	23	24	33	21	56
IBAS-M	42	29	13	11	18	28	45	23	23	33	21	56
IBAS	42	29	13	11	18	28	45	23	23	33	21	56

FIGURE 4. Comparison of number of loops.



	19-37	10-42	8-38	31-55	19-42	10-50	13-48	19-51	21-45	24-33	24-52	11-28
Dijkstra	125	123	94	120	130	140	125	141	109	94	109	137
Floyd	109	137	91	85	128	134	133	129	102	62	123	166
A-star	45	48	15	31	52	53	59	21	46	39	46	53
IBD	119	109	78	94	110	124	93	125	94	93	109	130
IBF	90	108	90	61	106	116	114	126	98	48	107	129
IBAS-S	41	42	13	25	40	47	59	21	46	36	44	50
IBAS-M	40	42	13	25	40	46	55	21	43	36	40	50
IBAS	40	42	13	25	39	45	53	21	40	32	31	49

FIGURE 5. Comparison of running time (ms).

TABLE 5. Indexes construction time and size.

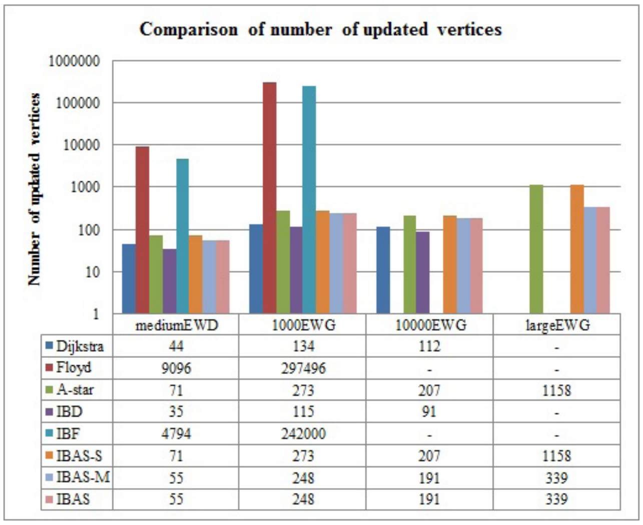
Name	Indexes construction time (ms)	Indexes size (M)
mediumEWD	89.33	0.003
1000EWG	596.2	0.011
10000EWG	3858	0.114
largeEWG	20074	11.44

and the construction time increases as the scale of the data increases.

Figs. 6, 7, and 8 respectively show the number of updated vertices, number of loops, and running time of each algorithm when computing the shortest path in each graph. The vertical coordinates of Figs. 6, 7, and 8 are transformed to log-polar coordinates, and "-" indicates that we do not have the results, as the corresponding algorithm failed due to running out of memory.

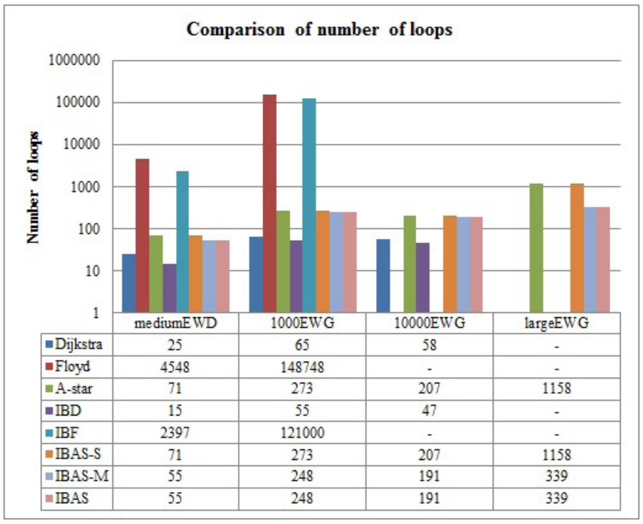
C. EXPERIMENTAL RESULTS ANALYSIS

1) Algorithms with a pruning strategy perform better than those without one in terms of the number of updated vertices,



	mediumEWD	1000EWG	10000EWG	largeEWG
Dijkstra	44	134	112	-
Floyd	9096	297496	-	-
A-star	71	273	207	1158
IBD	35	115	91	-
IBF	4794	242000	-	-
IBAS-S	71	273	207	1158
IBAS-M	55	248	191	339
IBAS	55	248	191	339

FIGURE 6. Comparison of number of updated vertices for benchmark data.

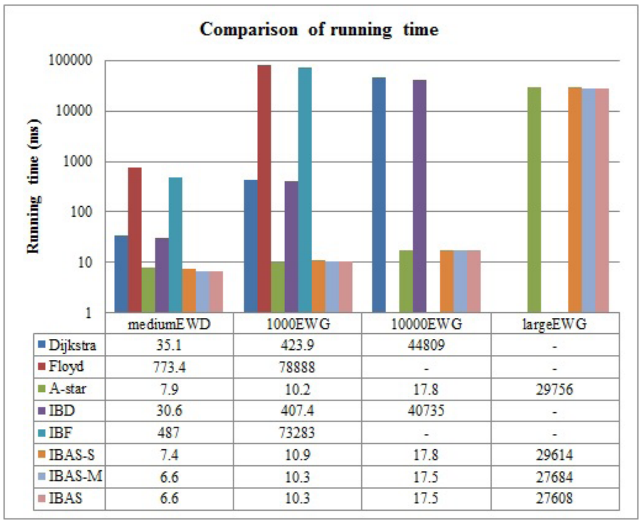


	mediumEWD	1000EWG	10000EWG	largeEWG
Dijkstra	25	65	58	-
Floyd	4548	148748	-	-
A-star	71	273	207	1158
IBD	15	55	47	-
IBF	2397	121000	-	-
IBAS-S	71	273	207	1158
IBAS-M	55	248	191	339
IBAS	55	248	191	339

FIGURE 7. Comparison of number of loops for benchmark data.

number of loops, and running time. On the one hand, from Table 2 we can see that an average of 20 vertices are pruned from the 66 vertices of the real road network in the process of the shortest path computation, and the pruning effect is beneficial. On the other hand, the algorithms IBD, IBF, and IBAS with indexes improve the Dijkstra, Floyd, and A-star algorithms. From the benchmark data experimental results, it is easy to see that the performances of the Dijkstra, Floyd, and A-star algorithms have been improved following the index construction. In 1000EWG, the IBD algorithm reduces the running time of Dijkstra from 423.9 ms to 407.4 ms, because the number of loops is reduced from 65 to 55, which leads to the number of updated vertices being reduced from 134 to 115. Exactly the same results are achieved on many instances in Figs. 3, 4, 5, 6, 7, and 8, which validate the effectiveness of the index method.

2) IBAS employs more effective index strategy than IBAS-M. From Table 2, we can see that IBAS-M can prune



	mediumEWD	1000EWG	10000EWG	largeEWG
Dijkstra	35.1	423.9	44809	-
Floyd	773.4	78888	-	-
A-star	7.9	10.2	17.8	29756
IBD	30.6	407.4	40735	-
IBF	487	73283	-	-
IBAS-S	7.4	10.9	17.8	29614
IBAS-M	6.6	10.3	17.5	27684
IBAS	6.6	10.3	17.5	27608

FIGURE 8. Comparison of running time for benchmark data (ms).

more vertices than IBAS. For example IBAS-M can prune 34 vertices for the instance from 10 to 50 while IBAS only prunes 22 vertices. It seems that IBAS-M is better than IBAS. But from Figs. 3 and 4, we see that the number of updated vertices and the number of loops are the same for the two algorithms. Therefore the running time of two algorithms is almost the same. For the instance from 10 to 50, IBAS-M prunes 12 vertices more than IBAS which are 3, 4, 5, 6, 11, 12, 13, 15, 19, 24, 29, and 39. From Fig. 2, we see that all these 12 vertices are in the branch of the path from 10 to 50. Therefore, these 12 vertices are useless vertices for solving the shortest path problem. Moreover, from Table 3, IBAS-M costs more time to create the two index intervals. Hence, IBAS which does not prune the useless vertices has better performance than IBAS-M.

3) The IBAS-S and IBAS algorithms are not only fast, but also suitable for solving large-scale network shortest path problems. Compared with IBD and IBF, the IBAS-S and IBAS algorithms can effectively shorten the running time and improve the computational efficiency. From the data in Fig. 8, taking 1000EWG as an example, the running time of IBAS-S and IBAS are 10.9 ms and 10.3 ms, respectively, while the running time of the IBD algorithm is 407.4 ms, and the IBF algorithm requires 73283 ms to complete the computation. Thus, from this comparison it is not difficult to see that IBAS-S and IBAS significantly shorten the running time and improve the computational efficiency. More importantly, the Dijkstra and Floyd algorithms cannot solve the shortest path problem in a large-scale network, while IBAS-S and IBAS can.

4) The performance of the IBAS algorithm is the best. In Fig. 6, the results for largeEWG show that IBAS-S and A-star have the same number of updated vertices, namely 1158, and the number of updated vertices for IBAS is only 339, which is clearly lower. Because the path between the vertices in the experiment is sparse and the path distance is large, the pruning effect cannot be achieved by the static pruning strategy, while the dynamic pruning strategy can achieve better pruning results. Therefore, in the shortest path computation the combination of these two pruning strategies, namely the IBAS algorithm, can achieve more efficient pruning.

V. CONCLUSION

In this paper, an Index-Based A-Star algorithm, IBAS, is proposed to solve the shortest path problem for weighted DAG with unknown vertex coordinates. The algorithm constructs the earliest arrival index, reverse earliest index, and latest arrival index for each vertex. The earliest arrival index is used to construct the evaluation function of the A-star algorithm, and all three indexes are used to prune useless vertices in the process of computing the shortest path, so as to improve the performance of the algorithm. The results of comparative experiments for the Dijkstra, Floyd, A-star, IBD, IBF, IBAS-S, and IBAS algorithms verify the effectiveness of the proposed pruning strategy, and then further verify the efficiency of the IBAS algorithm, which can be applied to solving the shortest path problem in large-scale networks.

This paper focuses on solving the shortest path problem in a weigthed DAG which is inspired by the indexes for reachability queries in an unweighted DAG. More effective algorithms in a weighted DAG and more valuable problems in a weighted directed graph will be studied in the future.

REFERENCES

[1] C. Sommer, "Shortest-path queries in static networks," *ACM Comput. Surv.*, vol. 46, no. 4, p. 45, 2014.
[2] D. Zhang, Y. Liu, A. Liu, X. Mao, and Q. Li, "Efficient path query processing through cloud-based mapping services," *IEEE Access*, vol. 5, pp. 12963–12973, 2017.
[3] T. Akiba, Y. Iwata, and Y. Yoshida, "Fast exact shortest-path distance queries on large networks by pruned landmark labeling," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 349–360.
[4] A. D. Zhu, H. Ma, X. Xiao, S. Luo, Y. Tang, and S. Zhou, "Shortest path and distance queries on road networks: Towards bridging theory and practice," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 857–868.
[5] Y. Sun, X. Yu, R. Bie, and H. Song, "Discovering time-dependent shortest path on traffic graph for drivers towards green driving," *J. Netw. Comput. Appl.*, vol. 83, pp. 204–212, Apr. 2017.
[6] D. Delling and G. Nannicini, "Core routing on dynamic time-dependent road networks," *INFORMS J. Comput.*, vol. 24, no. 2, pp. 187–201, 2012.
[7] D. Quercia, R. Schifanella, and L. M. Aiello, "The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city," in *Proc. 25th ACM Conf. Hypertext Soc. Media*, 2014, pp. 116–125.
[8] C.-H. Chen, "A semi-supervised feature selection method using a non-parametric technique with pairwise instance constraints," *J. Inf. Sci.*, vol. 39, no. 3, pp. 359–371, 2013.
[9] I. Kuperstein, L. Grieco, D. P. A. Cohen, D. Thieffry, A. Zinovyev, and E. Barillot, "The shortest path is not the one you know: Application of biological network resources in precision oncology research," *Mutagenesis*, vol. 30, no. 2, pp. 191–204, 2015.
[10] Y. Yuan, G. Wang, J. Y. Xu, and L. Chen, "Efficient distributed subgraph similarity matching," *VLDB J.*, vol. 24, no. 3, pp. 369–394, 2015.
[11] J. Han, Z. Li, and L. A. Tang, "Mining moving object, trajectory and traffic data," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2010, pp. 485–486.
[12] Y. Wu, Y. Tong, X. Zhu, and X. Wu, "NOSEP: Nonoverlapping sequence pattern mining with gap constraints," *IEEE Trans. Cybern.*, to be published, doi: 10.1109/TCYB.2017.2750691.
[13] Y.-X. Wu, L.-L. Wang, J.-D. Ren, W. Ding, and X.-D. Wu, "Mining sequential patterns with periodic wildcard gaps," *Appl. Intell.*, vol. 41, no. 1, pp. 99–116, 2014.

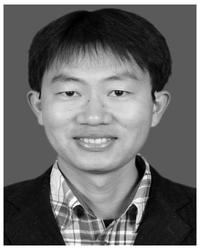
[14] H. Khosravi-Farsani, M. Nematbakhsh, and G. Lausen, "SRank: Shortest paths as distance between nodes of a graph with application to RDF clustering," *J. Inf. Sci.*, vol. 39, no. 2, pp. 198–210, 2013.
[15] R. Alireza and R. M. Mohammad, "Sampling social networks using shortest paths," *Phys. A, Stat. Mech. Appl.*, vol. 424, pp. 254–268, Apr. 2015.
[16] S. Wang, X. Xiao, Y. Yang, and W. Lin, "Effective indexing for approximate constrained shortest path queries on large road networks," in *Proc. VLDB Endowment*, 2016, vol. 10. no. 2, pp. 61–72.
[17] Y. Yuan, X. Lian, L. Chen, Y. Sun, and G. Wang, "RSkNN: kNN search on road networks by incorporating social influence," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 6, pp. 1575–1588, Jun. 2016.
[18] L. H. U, H. J. Zhao, M. L. Yiu, Y. Li, and Z. Gong, "Towards online shortest path computation," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 1012–1025, Apr. 2014.
[19] I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck, "Highway dimension and provably efficient shortest path algorithms," *J. ACM*, vol. 63, no. 5, p. 41, 2016.
[20] A. Sedeño-Noda and A. Raith, "A Dijkstra-like method computing all extreme supported non-dominated solutions of the biobjective shortest path problem," *Comput. Oper. Res.*, vol. 57, pp. 83–94, May 2015.
[21] C. J. Zhu, K.-Y. Lam, and S. Han, "Approximate path searching for supporting shortest path queries on road networks," *Inf. Sci.*, vol. 325, pp. 409–428, Dec. 2015.
[22] J. Gao, Q. Zhao, W. Ren, A. Swami, R. Ramanathan, and A. Bar-Noy, "Dynamic shortest path algorithms for hypergraphs," *IEEE/ACM Trans. Netw.*, vol. 23, no. 6, pp. 1805–1817, Dec. 2015.
[23] K. Nip, Z. Wang, and W. Xing, "A study on several combination problems of classic shop scheduling and shortest path," *Theor. Comput. Sci.*, vol. 22, pp. 175–187, Nov. 2016.
[24] J. Hong, K. Park, Y. Han, M. K. Rasel, D. Vonvou, and Y. K. Lee, "Disk-based shortest path discovery using distance index over large dynamic graphs," *Inf. Sci.*, vols. 382–383, pp. 382–383, 2017.
[25] A. Fariha, C. F. Ahmed, C. K.-S. Leung, S. M. Abdullah, and L. Cao, "Mining frequent patterns from human interactions in meetings using directed acyclic graphs," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining*, 2013, pp. 38–49.
[26] R. Jin, N. Ruan, J. Y. Xu, and S. Dey, "SCARAB: Scaling reachability computation on large graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 169–180.
[27] L. Zou, K. Xu, J. X. Yu, L. Chen, Y. Xiao, and D. Zhao, "Efficient processing of label-constraint reachability queries in large graphs," *Inf. Syst.*, vol. 40, pp. 47–66, 2014.
[28] H. Yildirim, V. Chaoji, and M. J. Zaki, "Grail: Scalable reachability index for large graphs," *Proc. VLDB Endowment J.*, vol. 3, nos. 1–2, pp. 276–284, 2010.
[29] J. Cheng, Z. Shang, H. Wang, J. X. Yu, and H. Cheng, "Efficient processing of k-hop reachability queries," *Very Large Data Bases J.*, vol. 23, no. 2, pp. 227–252, 2014.
[30] J. F. Zhou, W. Chen, Z.-Y. Chen, and C. P. Fei, "BiRch: A bidirectional search algorithm for k-step reachability queries," *J. Commun.*, vol. 36, no. 8, pp. 50–60, 2015.
[31] Y. Li, L. Sun, H. Z. Zhu, and Y. X. Wu, "A nettree for simple paths with length constraint and the longest path in directed acyclic graphs," *Chin. J. Comput.*, vol. 35, no. 10, pp. 2194–2203, 2012.
[32] Y. Li, Y. Wu, C. Huang, Z. Zhang, and Z. Zeng, "Nettree for maximum disjoint paths with length constraint in DAG," *J. Commun.*, vol. 36, no. 8, pp. 38–49, 2015.
[33] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
[34] R. Bellman, "On a routing problem," *Quart. Appl. Math.*, vol. 16, no. 1, pp. 87–90, 1958.
[35] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
[36] U. Meyer, P. Sanders, and B. Raphael, "Δ-stepping: A parallelizable shortest path algorithm," *J. Algorithms*, vol. 49, no. 1, pp. 114–152, 2003.
[37] G. Malewicz *et al.*, "Pregel: A system for large-scale graph processing," in *Proc. SIGMOD*, 2010, pp. 135–146.
[38] D. Delling, A. V. Goldberg, A. Nowatzyk, and R. F. Werneck, "PHAST: Hardware-accelerated shortest path trees," *J. Parallel Distrib. Comput.*, vol. 73, no. 7, pp. 940–952, 2013.
[39] V. Kalavri, T. Simas, and D. Logothetis, "The shortest path is not always a straight line: Leveraging semi-metricity in graph analysis," *VLDB Endowment*, vol. 9, no. 9, pp. 672–683, 2016.
[40] D. Zhang, C.-Y. Chow, A. Liu, X. Zhang, Q. Ding, and Q. Li, "Efficient evaluation of shortest travel-time path queries through spatial mashups," *GeoInformatica*, vol. 22, no. 1, pp. 3–26, 2017.
[41] Y. Zhang, Y.-L. Hsueh, W.-C. Lee, and Y.-H. Jhang, "Efficient cache-supported path planning on roads," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 4, pp. 951–964, Apr. 2016.
[42] Z. T. Duan *et al.*, "A K-th shortest path set algorithm for urban traffic network," *J. Transp. Syst. Eng. Inf. Technol.*, vol. 14, no. 3, pp. 194–200, 2014.

YAN LI received the Ph.D. degree in management science and engineering from Tianjin University, Tianjin, China. She is currently an Associate Professor with the Hebei University of Technology, Tianjin. Her current research interests include supply chain management and data mining.

HONGYAN ZHANG is currently pursuing the master's degree with the School of Economics and Management, Hebei University and Technology, Tianjin, China. Her research interests include supply chain management and data mining.

HUAIZHONG ZHU received the master's degree in computer science and engineering from the Hebei University of Technology, Tianjin, China. He is currently an Assistant Professor with the Hebei University of Technology. His current research interests include data mining and machine learning.

JIANWEI LI received the Ph.D. degree in theory and new technology of electrical engineering from the Hebei University of Technology, Tianjin, China. He is currently a Professor with the Hebei University of Technology. His current research interests include bioinformatics and data mining.



WENJIE YAN received the Ph.D. degree from the School of Computer Science and Engineering, Harbin Institute of Technology, Harbin, China. He is currently an Assistant Professor with the Hebei University of Technology, Tianjin. His current research interests include data mining and intelligent recommendation.

YOUXI WU (M'17) received the Ph.D. degree in theory and new technology of electrical engineering from the Hebei University of Technology, Tianjin, China. He is currently a Ph.D. Supervisor and a Professor with the Hebei University of Technology. His current research interests include data mining and machine learning. He is a Senior Member of CCF.