Hindawi
Scientific Programming
Volume 2017, Article ID 2056501, 7 pages
https://doi.org/10.1155/2017/2056501

Research Article

Routing Optimization Algorithms Based on Node Compression in Big Data Environment

Lifeng Yang,[1] Liangming Chen,[2] Ningwei Wang,[2] and Zhifang Liao[2]

[1]*School of Continuing Education, Yunnan Open University, Yunnan, China*
[2]*School of Software, Central South University, Hunan, China*

Correspondence should be addressed to Zhifang Liao; zfliao@csu.edu.cn

Received 26 August 2017; Accepted 5 December 2017; Published 26 December 2017

Academic Editor: Wenbing Zhao

Shortest path problem has been a classic issue. Even more so difficulties remain involving large data environment. Current research on shortest path problem mainly focuses on seeking the shortest path from a starting point to the destination, with both vertices already given; but the researches of shortest path on a limited time and limited nodes passing through are few, yet such problem could not be more common in real life. In this paper we propose several time-dependent optimization algorithms for this problem. In regard to traditional backtracking and different node compression methods, we first propose an improved backtracking algorithm for one condition in big data environment and three types of optimization algorithms based on node compression involving large data, in order to realize the path selection from the starting point through a given set of nodes to reach the end within a limited time. Consequently, problems involving different data volume and complexity of network structure can be solved with the appropriate algorithm adopted.

1. Introduction

The single source shortest path problems in graph theory are very typical questions that enjoy wide applications in real life, such as network routing path selection, vehicle navigation, and travel routes. The classic algorithm to solve such problems is Dijkstra's Algorithm [1] proposed by Dijkstra in 1959 and a lot of researchers focus on this research area [2–4]. However, Dijkstra fails to solve problems where routes are required to go from the starting point, pass the specified intermediate node, and finally reach the destination—far more practical problems exemplified as follows:

(1) "Postman problem": the postman starts from the post office, sends letters to residents, and returns home, where we need to find the postman a shortest path within a given time.

(2) "Limited time problem": within a limited time, activities designed for staff members who tracked consent using depth sensors were proposed and they were carefully reminded of noncompliant activities [5], and a collaborative smartphone task model is proposed, which is called Collaboration-Based Intelligent Perception Task Model (CMST) [6].

(3) "Traveler problem": calculate a travel route for the traveler within the specified time, who needs to go from a designated location, pass a designated scenery spot, and visit a given place. The total distance should be the shortest or the total expense should be the lowest [7, 8].

(4) "Compression problem": a new compression method for large data environment is proposed, which can effectively reduce the data compression of single nodes and ensure the quality of data [9]. Due to the large amount of web service data, a data-driven scheme is based on kernel least mean squares (KLMS) algorithm [10]. In order to compress the input to further improve the learning effect, a new QKLMS is based on entropy-guided learning [11].

(5) "Network routing problem": find an efficient routing algorithm to solve the problem of path optimization of wireless sensor network, considering the influences of some practical factors such as the consumption of the energy of the nodes and recovery time of routing [12–14].

(6) "Laguerre neural network" [15]: it intends to propose a novel automatic learning scheme to improve the tracking efficiency while maintaining or improving the data tracking accuracy. A core strategy in the proposed scheme is the design

of Laguerre neural network- (LaNN-) based approximate dynamic programming (ADP).

(7) "Energy of the sensor nodes" [16]: a novel prediction-based data fusion scheme using grey model (GM) and optimally pruned extreme learning machine (OP-ELM) is proposed. The proposed data fusion scheme called GM-OP-ELM uses a dual prediction mechanism to keep the prediction data series at the sink node and sensor node synchronous.

These problems can be summarized as one graph theory problem; that is, in a weighted directed graph, a route goes from a starting point, passes through the designated intermediate node, and reaches a destination. It is required to find valid paths within a specified time, calculate the weight of these paths, and select a path with the lowest weight as the final result.

To solve this kind of problems, we may traverse the whole graph and find a shortest path, although theoretically this traversal algorithm will eventually sort out the optimal solution; however the time complexity remains high. In view of this, this paper proposes a node compression routing algorithm with considered time limits. The study pays attention to node compression and applies useful information obtained in path finding to search conditions, readjusting the order of subnodes and other methods as well. Additionally, the high time complexity in traditional algorithm is improved, offering an effective solution to this type of problem.

2. Problem Description

2.1. Mathematical Model of the Problem. Given a weighted graph $G(V, E)$ where $V = \{1, 2, 3, \ldots, n\}$ is the vertex set, $E = \{e_{ij} = (i, j) \mid i, j \in V,\ i \neq j\}$ is the edge set. d_{ij} $(i, j \in V,\ i \neq j)$ is the weight of vertexes i to j, where $d_{ij} > 0$ and $d_{ij} \neq \infty$; while d_{ij} and d_{ji} may be unequal, $V' = \{1', 2', \ldots, n'\} \in V$. We need to find the sequence $A = \{a_1, a_2, a_3, \ldots, a_n\}$ within a given time, where s is starting point and t is the destination, $s, t \in V$ and s, t do not belong to V', all of the elements in V' must appear in sequence A, making the sum of the weights of all edges of the path formed in sequence A minimal, and loop is not allowed in any path. The mathematical model of the problem is defined as follows.

Under the condition of Time $= t$, solve $\min C = \sum_{i \neq j} d_{ij} \times X_{ij}$, in order to define the starting point s and the destination t and make sure that there's only one in-edge and out-edge on each vertex except the edges of starting point and the destination paths; we make the following constraints:

$$X_{ij} = \begin{cases} 1, & \text{edge } e_{ij} \text{ is along the result path} \\ 0, & \text{edge } e_{ij} \text{ is out of the result path,} \end{cases} \quad (1)$$

where X_{ij} is an integer of 0 or 1, 1 represents edge e_{ij} on the result path, and 0 represents edge e_{ij} out of the result path, and X_{ij} is used to calculate the weight of the resulting path.

$$\sum_{i \neq j} X_{ij} = 1, \quad j \in V', \quad (2)$$

where $i \neq j$ means that the result path cannot contain the edges that the starting node and the end node are the same node, which means the point in the intermediate node set on the result path can only occur once and must occur once.

$$\sum X_{sj} = 1, \quad j \in V,\ j \neq s. \quad (3)$$

The formula defines an edge that begins with the starting nodes which should appear in the result path, and the starting node in the edge cannot be the end node.

$$\sum X_{js} = 0, \quad j \in V,\ j \neq s. \quad (4)$$

The formula restricts that the starting node s can only be the starting node in an edge, and it cannot be any other kind of nodes, such as end node or intermediate nodes.

$$\sum X_{it} = 1, \quad i \in V,\ i \neq t. \quad (5)$$

The formula restricts that the result path must have an edge ended with the end node t, which means the edge cannot start with the end point t.

$$\sum X_{ti} = 0, \quad i \in V,\ i \neq t. \quad (6)$$

The formula restricts that the resulting path cannot contain the edge beginning with the end node t; that is, the end node t can only be used as the final node on the resulting path.

$$\sum_{i,j \in V} X_{ij} = |A|. \quad (7)$$

This formula defines the number of edges on the resulting path which can be the number of nodes minus one; that is, the resulting path cannot appear with unrelated edges and loops.

For the convenience of subsequent description, the following two definitions are given.

Definition 1 (key nodes). The nodes in V' include other must-pass nodes except starting point s and destination t.

Definition 2 (free nodes). All other nodes except the key nodes are included.

2.2. Simple Example. In the weighted graph G shown in Figure 1, four nodes can be found, namely, 0, 1, 2, and 3; therefore $V = \{0, 1, 2, 3\}$, and there are seven edges 0, 1, 2, 3, 4, 5, and 6, so $E = \{0, 1, 2, 3, 4, 5, 6\}$, where the weight of the edge is $\{d_{01} = 1,\ d_{02} = 2,\ d_{03} = 1,\ d_{21} = 3,\ d_{31} = 1,\ d_{23} = 1,\ d_{32} = 1\}$. To find a path from 0 to 1 via vertexes 2 and 3, we have $V' = \{2, 3\}$. Two paths can be found to solve this problem: 0→2→3→1 and 0→3→2→1. Since the weight of edges on the first route is 4, and the weight of the other is 5, the optimal solution should be 0→2→3→1.

3. Improved Backtracking Algorithm: IBA

If using the backtracking method to solve this problem, theoretically, we can have the optimal solution and of course other solutions. However, the backtracking method does not effectively use information constructed in the search process or the optimal solution to lay a foundation for optimization condition of the next-step search. In this section, an

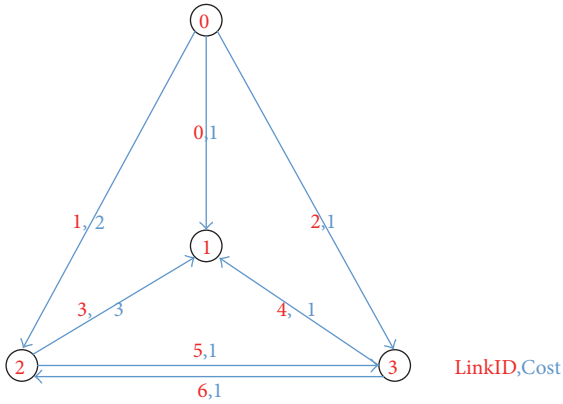

Figure 1: A simple example of the problem.

```
Improved-Backtrack (G)
(1) node=start
(2) while usedtime< t &&
(node!=end && !A′ ∈ nodes)
(3) nodes.add(node)
(4) record information include
route and weigths
(5) for i = 1 to children.length
(6) add search rule
(7) Improvedacktrack (children[i])
(8) if result !=null-B
(9) return result and weight
(10) else
(11) return NA
```

Algorithm 1: The key pseudocode of the improved backtracking algorithm.

improved backtracking method (OPT-Backtrack Algorithm) is proposed based on traditional backtracking method. The new IBA retrieves known information and valid results from the previous search and adds them up to the next search rules before searching from other nodes. In this way, the search method and algorithms can be improved, since existing information and possible results are taken into consideration for a higher search efficiency.

The addition rule in the improved backtracking algorithm is shown below.

Rule 1. If the next node happens to be the destination, yet the current path has not gone through every must-pass node in the node set, the path will track back and begin searching for the next node. This rule avoids the generation of many invalid solutions thus improving the algorithm efficiency.

Rule 2. If the current path weight and the weight of the edge to the next node is greater than or equal to the minimum weight of the available solution, the path will track back and continue searching for the next node. If current path has been found whose current weight and the weight of edge to the next node is no more than the existing weight, then there is no need to search for the next node, because initially the problem is to find the smallest possible weight of the path.

Rule 3. For those nondestination nodes with zero child nodes, we should avoid entering the search. If a node is not destination and has no child nodes, the path shall not continue; therefore, it is not necessary to search at such nodes or rather they can be simply deleted from the graph.

The key pseudocode of the improved backtracking algorithm is shown in Algorithm 1.

4. Node Compression Based Search Algorithm

Although search efficiency can be enhanced by the improved backtracking algorithm to a certain degree, the negative complexity of the improved backtracking method will also increase as scale of the graph and solution domain expand. To reduce algorithm complexity, this paper proposes a new algorithm, node compression based search algorithm: NCSA.

As the scale of graph increases, paths will expand accordingly. The same problem would be finding a path from a start point, reaching an intermediate node halfway and finally the destination. To reduce the algorithm complexity, we may preprocess the graph. The method is to compress the total number of nodes, remove useless nodes and low-value path fragments, and then save the only paths that are necessary to simplify the entire graph; the goal is to compress solution domain and ultimately improve search efficiency.

4.1. Node Compression Algorithm (NCA). The algorithm is applicable to the following circumstance: If a node is relatively remote which only reaches one other node, that is, a node followed only by one child node, in this case, the search will follow down the only child node route and will repeat this wherever there is such a node during the searching process. What we want to do is to avoid the simple and repeated calculations in this kind of situation.

Solution to this problem is Node Compression Algorithm (NCA). NCA records the paths through the above-mentioned nodes when the algorithm is applied for the first time and will remove the nodes but retain the path information; therefore, when the next search continues at this node, only stored path information will be used to avoid duplicated counting. As a result, the total number of nodes is compressed and reduced, making it easier to search for a better solution.

The process is shown in Figure 2.

In Figure 2, node 1 is followed by the only child node 2, the weight from nodes 1 to 2 is 2, marked as path 1; the compression process means transferring node 1 information to node 2 so that node 2 becomes the direct child node of node 0. If compressed, the weight from nodes 0 to 2 is 3, and path from nodes 0 to 2 is "0 | 1." This means node 1 is removed while the path information from nodes 1 to 2 is retained solely in node 2. When the next search algorithm reaches node 0, information retained in node 2 can be used directly without going back to node 1. So the number of nodes is reduced and the path will not be searched again.

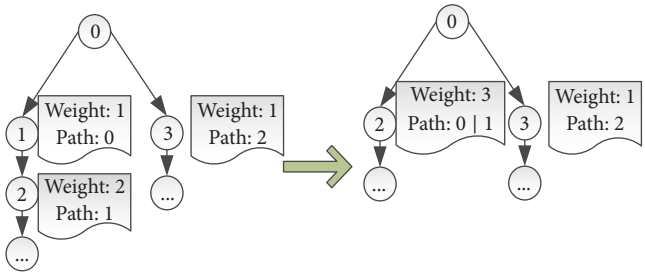

Figure 2: The basic idea of compression search algorithm.

4.2. Complete Compression Algorithm: CCA. Since Node Compression Algorithm (NCA) is used mainly to solve free nodes with only one child node, if such nodes are many in the graph, the algorithm efficiency will be significantly improved. However, if the scale of such nodes is limited, the basic compression algorithm will take less or no effect, which limits the effectiveness of compression search algorithm.

In view of the problem of NCA, this paper proposes a more efficient compression strategy, which compresses all free nodes in the graph to reduce the complexity of the graph, improving the search efficiency.

The problem is finding a noncircle path from the starting node to the destination node while passing through the intermediate node sets so that the weights of the edges on paths are as small as possible. When the reachability of nodes is complex, there will be many more possible paths to reach nodes of one and another. Since the problem requires that intermediate node set V' be passed and, within the set, there are multiple reachable paths between nodes, yet only one path will be selected within the set as one fragment of the final solution, therefore, we should find out all reachable paths while saving the path with the smallest weight. As the search algorithm reaches a corresponding node, the valid path will be retrieved from the stored information while the original nodes on the path can be removed from the graph, reducing useless nodes and repetitive counting. With this compression method, only the starting point, destination, the intermediate node set, and their interconnected path information will remain, simplifying the entire graph to a large extent with excellent compression efficiency.

Just like Figure 1, it can be seen as a simplified graph, and only the starting point, destination, and intermediate node set are preserved. In this way, we can achieve good compression efficiency by selecting the reachable path with the smallest path.

4.3. Improved Complete Compression Algorithm: ICCA. In order to further improve compression efficiency, this section continues to adjust and improve node compression by the three steps.

4.3.1. Adjusting Child Nodes Order by Weight. In the search process, algorithm can be done based on the weight of feasible solutions (see Rule 2 of IBA). First the order of subnodes is sorted according to the weight size from small to large. When algorithm searches the path, subnodes carrying smaller weight are searched with priority so that paths with smaller weight are easily obtained. As a result of this search strategy, other paths with larger weight can be skipped. This certainly reduces unnecessary search processes with greater efficiency.

4.3.2. Adjusting Child Nodes Order by the Sequence of Passing Nodes (from Small to Large). From the perspective of probability, when a new node is inserted into a graph, the more the nodes a path passes, the more likely the repeated path will be generated. Therefore, under the condition of same weight, the nodes with fewer subnodes will be given priority since the paths that follow will make fewer repeated attempts, making it easier to find the solution path.

4.3.3. Removing Child Nodes with Larger Weight. This strategy is only applicable to high-complexity graphs. After compression, the remaining nodes will connect one and another to form paths; complexity of the graph might be still high. There would be the case where one path might be an effective solution but the nodes it passes carry excessive weight, so the path will not be considered the final solution. In this case, removing large weight nodes will lower the graph complexity and improve search efficiency. In addition, it will save time and figure out a better solution with a lower weight path.

By analysis, the spatial complexity of IBA is $O(n)$, while the spatial complexity of NCA, CCA, and ICCA is $O(n^2)$, where n is the total number of nodes in the graph. ICCA can quickly select the shortest paths according to the weights of nodes and the nodes with smaller weights and delete the nodes with larger weights from the compression of large networks efficiently.

5. Experimental Analysis

5.1. Data Description and Analysis. Without loss of generality, experiment data are from the cases of *2016 Huawei Software Elite Competition*; these quoted examples are based on the network topological graph of Huawei's network routers, switches, and other network elements when Huawei established its own network facilities.

5.1.1. Problem Description. Given a weighted graph $G = (V, E)$, V is the vertex set, E is the directed edge set, and each directed edge contains the weight. For a given vertex s, t, and a subset V' of V, find a nonringing directed path P from s to t within a given time so that P passes through all vertices in V' (the order of passing is not required), making the total weight of all directed edges on path P as small as possible.

5.1.2. Data Description. (1) All weights in the graph are integers within $[1, 20]$.

(2) The starting point of any directed edge is not destination.

(3) The number of directed edges connecting vertex A to vertex B may be more than one, whose weight may or may not be the same.

(4) The total number of vertices of the directed graph will not exceed 600, and the number of each vertex out-degree

(the number of directed edges with these points as the starting point) does not exceed 8.

(5) The number of elements in V' does not exceed 50.

(6) The nonringing directed path P starts from s to t, where P is a directed connected path consisting of a series of directed edges from s to t, with no repeated path allowed.

(7) The weight of a path is the sum of all weights on the directed edges of the path.

5.1.3. Data Format. (1) In the graph, each line contains the following information:

{LinkID, SourceID, DestinationID, Cost},

where LinkID is index of directed edge, SourceID is index of the starting vertex of the directed edge, DestinationID is the index of destination vertex of the directed edge, Cost is the weight of the directed edge. The index of vertex and that of directed edge are numbered from 0 (not necessarily continuous, but the case ensures that the index does not repeat).

(2) Path information includes

{SourceID, DestinationID, IncludingSet},

where SourceID is the starting point of the path, DestinationID is the destination of the path, and IncludingSet represents the must-pass vertex set V', and different vertex indexes are segmented with "|."

5.1.4. Experiment Environment. Windows 7 64-bit operating system, with Intel core i5 processor, jre1.6, 32-bit java virtual machine, up to 4 G memory, is used.

5.2. Experiment Methods and Result Analysis

5.2.1. IBA, NCA, and CCA Comparison. To verify backtracking method and IBA, NCA, and CCA algorithms, four sets of experiments will be conducted with the solution time limited to 10 seconds. From Experiments 1–4, the total number of nodes and edges in the graph will be gradually increased, while the number of intermediate nodes will be kept unchanged. Experiment results will be compared by the weight of final path result and time spent.

Experiment 1. Total nodes are 10; must-pass nodes are 3; edges are 39.

Figure 3 shows the experimental result from Experiment 1 and it presents the fact that IBA has higher efficiency than the backtracking method. Efficiency difference is not remarkably obvious in NCA and CCA because the compression process takes time and also the efficiency becomes even less obvious if the complexity of the graph is low.

Experiment 2. Total nodes are 20; must-pass nodes are 5; edges are 55.

Figure 4 shows the experimental result from Experiment 2 and it presents the fact that IBA, NCA, and CCA have a greater efficiency than backtracking method. Efficiency

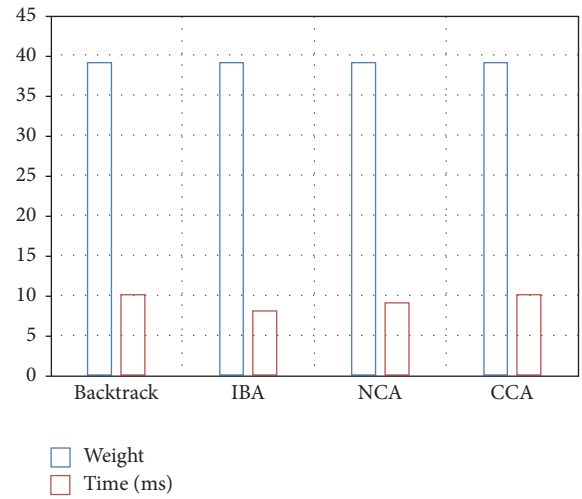

Figure 3: Experimental results of Experiment 1.

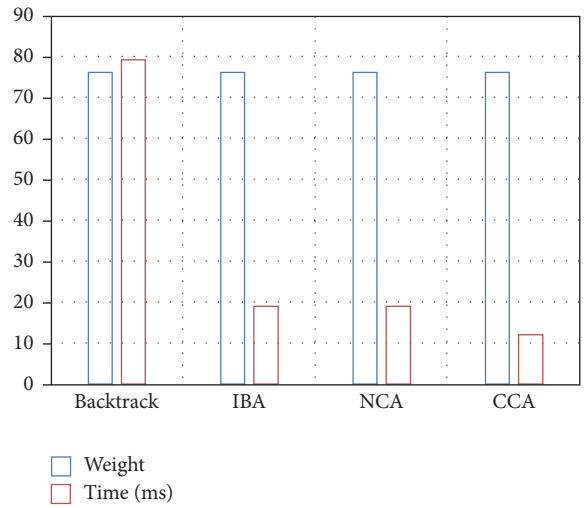

Figure 4: Experimental results of Experiment 2.

of CCA is the highest while IBA and NCA have a similar efficiency because of few remote nodes.

Experiment 3. Total nodes are 30; must-pass nodes are 10; edges are 135.

Figure 5 shows the experimental result from Experiment 3 and it presents the fact that the superiority of CCA proves obvious as graph complexity gradually improves.

Experiment 4. Total nodes are 40; must-pass nodes are 10; edges are 229.

Figure 6 shows the experimental result from Experiment 4 and it presents the fact that backtracking method indicates low efficiency if complexity of the graph is even higher; in contrast, CCA efficiency performs reasonably well.

Experiment results have shown that IBA has a higher efficiency than backtracking method judged by either weights

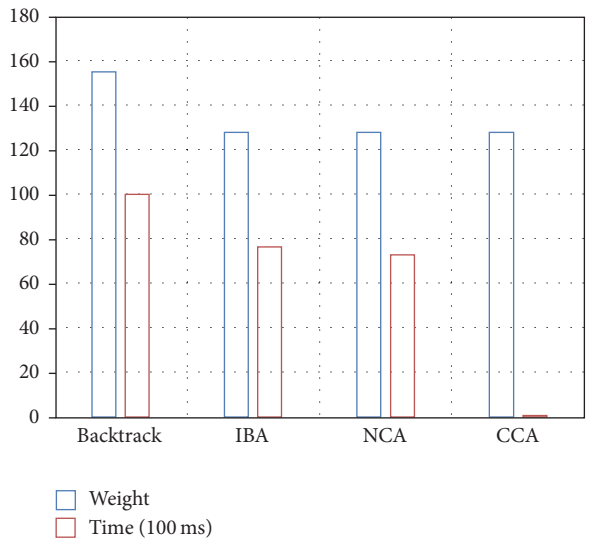

Figure 5: Experimental results of Experiment 3.

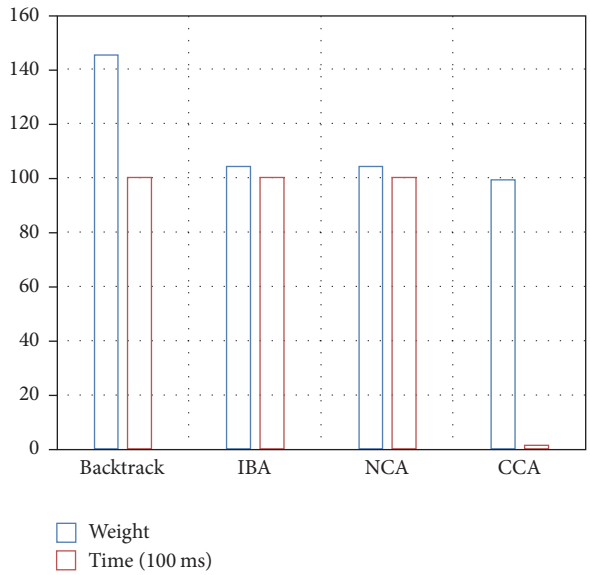

Figure 6: Experimental results of Experiment 4.

or search time. NCA shows only a slight advantage over IBA because remote nodes in the graph are very limited. In particular, judging from all dimensions, CCA has proved significant quality in searching the results with superior efficiency to other algorithms, indicating the effectiveness of CCA in solving such problems.

5.2.2. CCA and ICCA Comparison. It is observed from the previous four experiments that the respective efficiency of backtracking method, IBA, and NCA decreases drastically as the sum of nodes increases. Therefore, there is no research value to add up more nodes to the graph. This section continues to compare between CCA and ICCA.

Experiment environment will remain the same as those of Experiments 1–4; experiment will gradually increase total

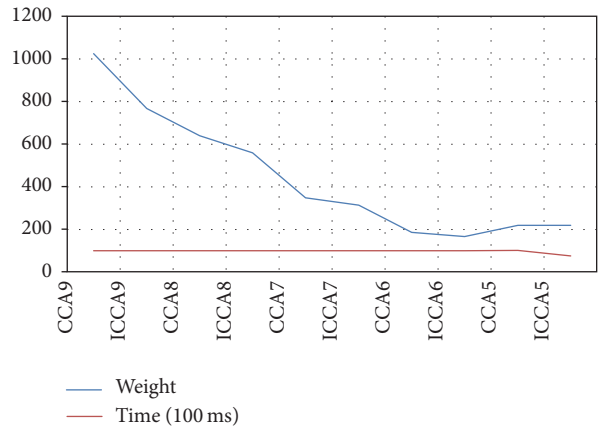

Figure 7: Experimental results of Experiments 5–9.

nodes and edges, while the size of intermediate nodes set will also increase. Comparison will be based on the following five experiments.

Experiment 5. Total nodes are 60, must-pass nodes are 10, and edges are 285.

Experiment 6. Total nodes are 100, must-pass nodes are 15, and edges are 516.

Experiment 7. Total nodes are 200, must-pass nodes are 20, and edges are 997.

Experiment 8. Total nodes are 400, must-pass nodes are 28, and edges are 2178.

Experiment 9. Total nodes are 600, must-pass nodes are 50, and edges are 3418.

Figure 7 shows the experimental results which have indicated that compared to CCA, ICCA obtains better solutions. Therefore, the improved strategy in Section 4.3 is proved to be effective.

6. Conclusion

Problems like postman problem, traveler problem, bus line design, network routing problem, and other similar cases can be abstracted as the path finding graph model as discussed in this study. IBA and NCA are applicable to medium-sized problems. NCA is recommended to solve graphs that contain many remote nodes, while CCA and ICCA are more efficient in dealing with large-scale problems with great algorithm complexity. Additionally, ICCA is able to promote search efficiency when subnodes are readjusted.

As the size of problem becomes larger, CCA and ICCA may not be able to search the whole solution space completely with the optimal solution within a given time. In this case, the compression idea will be integrated into heuristic algorithms such as genetic algorithm and ant colony algorithm to expect a far more efficient search algorithm so as to resolve routing problems with larger scales.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[2] D.-Y. Zhang, W.-L. Wu, and C.-F. Ouyang, "Top-k shortest-path query on RDF graphs," *Tien Tzu Hsueh Pao/Acta Electronica Sinica*, vol. 43, no. 8, pp. 1531–1537, 2015.

[3] H. Y. Cao, Y. Yuan, and Z. Q. Liu, "Routing algorithm for WSNs based on residual energy of node and the maximum angle," *Transducer & Microsystem Technologies*, 2015.

[4] L.-Y. Feng, L.-W. Yuan, W. Luo, R.-C. Li, and Z.-Y. Yu, "Geometric algebra-based algorithm for solving nodes constrained shortest path," *Tien Tzu Hsueh Pao/Acta Electronica Sinica*, vol. 42, no. 5, pp. 846–851, 2014.

[5] W. Zhao, R. Lun, C. Gordon et al., "A human-centered activity tracking system: toward a healthier workplace," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 3, pp. 343–355, 2017.

[6] T. Li, Y. Liu, L. Gao, and A. Liu, "A cooperative-based model for smart-sensing tasks in fog computing," *IEEE Access*, vol. 5, pp. 21296–21311, 2017.

[7] Y.-H. Qi, Y.-G. Cai, H. Cai, Y.-L. Tang, and W.-X. Lv, "Chaotic Hybrid Discrete Bat AIgorithm for TraveIing SaIesman ProbIem," *Acta Electronica Sinica*, vol. 44, no. 10, pp. 2543–2547, 2016.

[8] Y. Z. Wang, Y. Chen, and J.-S. Zhang, "Novel Fruit Fly Algorithm Based on Learning and Memory for Solving Traveling Salmesman Problem," *Journal of Chinese Computer Systems*, vol. 37, no. 12, pp. 2722–2726, 2016.

[9] C. Yang, X. Zhang, C. Zhong et al., "A spatiotemporal compression based approach for efficient big data processing on Cloud," *Journal of Computer and System Sciences*, vol. 80, no. 8, pp. 1563–1583, 2014.

[10] X. Luo, J. Liu, D. D. Zhang, and X. Chang, "A large-scale web QoS prediction scheme for the Industrial Internet of Things based on a kernel machine learning algorithm," *Computer Networks*, vol. 101, pp. 81–89, 2016.

[11] X. Luo, J. Deng, J. Liu, W. Wang, X. Ban, and J. Wang, "A quantized kernel least mean square scheme with entropy-guided learning for intelligent data analysis," *China Communications*, vol. 14, no. 7, pp. 127–136, 2017.

[12] A. Fernández-Fernández, C. Cervelló-Pastor, and L. Ochoa-Aday, "Improved Energy-Aware Routing Algorithm in Software-Defined Networks," in *Proceedings of the 41st IEEE Conference on Local Computer Networks, LCN 2016*, pp. 196–199, UAE, November 2016.

[13] N. Li, J.-F. Martínez, and V. H. Díaz, "The balanced cross-layer design routing algorithm in wireless sensor networks using fuzzy logic," *Sensors*, vol. 15, no. 8, pp. 19541–19559, 2015.

[14] L. Lei, W. F. Li, and H. J. Wang, "Path optimization of wireless sensor network based on genetic algorithm," *Journal of University of Electronic Science & Technology of China*, vol. 38, no. 2, pp. 227–230, 2009.

[15] X. Luo, Y. Lv, M. Zhou, W. Wang, and W. Zhao, "A laguerre neural network-based ADP learning scheme with its application to tracking control in the Internet of Things," *Personal and Ubiquitous Computing*, vol. 20, no. 3, pp. 361–372, 2016.

[16] X. Luo and X. Chang, "A novel data fusion scheme using grey model and extreme learning machine in wireless sensor networks," *International Journal of Control, Automation, and Systems*, vol. 13, no. 5, 2015.