Abstract—The most famous one for finding the shortest path is the Dijkstra, but it has some limitations. For example, when there are more than one shortest path between the source node and one special node, Dijkstra could find only one of them. Besides, the algorithm is quite complex. This article introduce a conception of coding graph, abstracting the problem of shortest paths and critical paths into the same mathematical model to describe and solve, presents a new algorithm of finding the paths. The algorithm extends first the data structure of the orthogonal list, so that the graph will be stored in the same storage space with the path searching process and result data. Codes for all nodes in the graph starting from the source node, using the rule of getting extremum in the weighing calculation and breadth-first, When accessing recursively the adjacent nodes from the current node, re-estimate the distance of neighboring node and entering edge list, if the distance of current node plus the weight of the edge to the neighboring node is less than (or greater than) the original distance of the neighboring node, then set this value as the new distance of this neighbor node, if the distance of current node plus the weight of the edge to the neighboring node is greater than (or less than) the original distance of the neighboring node, then the edge node will be deleted from the entering edge list, until all the nodes were coded and the coding graph of shortest path (or critical path) is created. For each node in the coding graph, starts searching from entry edge list recursively could get all shortest paths (or critical paths) and distances to the source node. Compared with the existing algorithms, this algorithm is simpler and more understandable, needs only 3n+5e storage unit that is much less than that of Dijkstra (which is n^2+2n). The time complexity O(n+e), which is also lower than the Dijkstra O(n^2).

Index Terms—coding graph, shortest path, critical path, Dijkstra algorithm, extending orthogonal list, time complexity.

I. INTRODUCTION

On the algorithm of shortest path and critical path, there were already many researches and applications. For example, represented as AOE(Activity On Edge Network), through adjusting and distributing the human and other physical resources to short the period of project, the key action which influences the project progress most. It is solving the problem of critical path of AOE network. But in network analysis, traffic plan, communication and computer science, great many problem could be formalized as finding shortest path model[1, 2], that is the problem of weighted graph shortest path of G =<V, E, W>. So, the researches on algorithm for finding shortest path attracted many professionals, and have extensive application area.

After the creative work of Dijkstra, many solutions for finding the shortest path have been presented [1~14]. Eli Olinick[3] and D.K.Smith[4] discussed it in 1966, but they are too simple, incomplete, and lacking time analysis, so have no much practicability. Cherkassky[12] give an theoretic analysis on seventeen algorithms for finding shortest path and give experimental evaluation. F. Benjamin Zhan tested fifteen of them, the result show that three of them are good, they are: TQQ (graph growth with two queues), DKA (the Dijkstra’s algorithm implemented with approximate buckets) and DKD (the Dijkstra’s algorithm implemented with double buckets). The base of TQQ is graph growth theory, fit for computing the shortest path from one single node to all other node; the other two algorithm based on Dijkstra, fit better for computing the shortest path between two nodes[13]. Tan Guozhen[14] analyzed and evaluated more than twenty algorithms, including method of flag setting, modifying, dynamic planning, method based on linear algebra, method of elicitation and two-way elicitation, method based on neural network of fluid. All of these methods are based on Dijkstra, at the cost of time and space to get the improvement and extension, essentially not escape from the limitation of Dijkstra theory. So, the method simpler and more optimized than Dijkstra has not occurred yet.

Time–dependent model is more practical. Hongyan An and Guozhen Tan studied shortest path on time-varying network [15, 16] and got some achievements. But there is no mature algorithm occurred even today. It is still an area needs to do more study.

Traditional algorithm for finding shortest path[17] have shortage as follows: (1) needs to get the earliest and the latest time for every event occurred, calculate earliest and latest starting time for each action, estimate which is key action, the calculation is complex. (2) After the algorithm running, could get to know only which is and which is not key action, but could not get the number of critical path from the source node to the crossing-node. Neither could get each critical path. Fengsheng Xu[18]...
The algorithm is simple, and visual. The storage structure of the algorithm is suitable for the general. The algorithm is seen as a special case of weighted graph, so this algorithm can be used to find the shortest path. The usual directed and undirected graph can be transformed into orthogonal list, abstracting the shortest path and critical path (or critical path tree) and their length could be found. The algorithm has time complexity \(O(n+e)\), optimal than that of Dijkstra (which is \(O(n^2)\)).

(5) Take the maximum advantage of the graph formed before changing, by re-encoding on the changed shortest path tree (or critical path tree) it could be expanded to the shortest path (or critical path) of the fast dynamic algorithm.

### II. BASIC CONCEPTS AND CHARACTERS

#### A. Definition of terms

**Definition 1.** For a given weighted graph \(G = (V, E, W)\) with \(n\) nodes, where node set \(V = \{V_i | i=0, 1, 2, \ldots, n-1\}\), edge set \(E = \{E(i, j) | i, j=0, 1, 2, \ldots, n-1\}\), weight set \(W = \{w(i, j) | i, j=0, 1, 2, \ldots, n-1\}\), the coding graph \(G\) corresponding source node \(V_0\) (denoted by \(G(S, V_0)\)) could be defined constructively as:

\[
G(S, V_0) = \langle PV, E', W' \rangle
\]

Where \(PV = \{PV_i = (V_i, L_i) | i=0, 1, 2, \ldots, n-1\}\), \(L_i\) as PV scalar quantity of nodes in \(G(S, V_0)\), its value is equal to the length of the shortest path between \(V_i\) and \(V_0\) (if no path, take \(\infty\), which will be called isolated node); \(E' \subseteq E\), and \(E'=\{all\ edges\ of\ shortest\ path\}\), \(W'\) is the set of weights corresponding to elements in \(E'\), i.e., \(W' \subseteq W\).

For example, Figure 1(B) is coding graph \(G(S, V_0)\) of \(G_1\), corresponds to source node \(V_0\) in Figure 1(A), while Figure 1(C) is the expansion of orthogonal list storage representations of \(G_1(S, V_0)\).
Clearly, in the representation of expanded orthogonal list G1 (S, V0) for each node Vi, search in opposite direction starting from entry-edge, until arrive the source node V0, could find all shortest path from V0 to V1, and Li in PVi = (Vi, Li) is the shortest path from V0 to Vi.

Completely similar, let G = <V, E, W> is a n-node acyclic AOE network, in the above constructed definitions, if in PVi = (Vi, Li) the node Li is the longest path value from V0 to V1 (if no path, take 0 value), then G1 (S, V0) is the coding graph to solve the critical path, same way, all the critical path, the key activities and length can be obtained from the source to either meeting-node.

Definition 3. Set G = <V, E, W> is a n-nodes weighted graph (or acyclic AOE network), G′ = G (S, V0) is the shortest path code graph of G corresponds to source node V0, in G′ shortest path tree (or critical path tree) of node s is the biggest sub-graph T(s) = <PV, E, W> of G′, shortest path code graph of G corresponds to source node V0, T(s) is the shortest path from V0 to V1, if L1 is the shortest path from V0 to V1, L2 is the shortest path from Vk to Vi, if L pass through the node Vk, then according to the Dijkstra algorithm, you can find the shortest path and the length, suppose the path (V0, …, Vf, V1) is the shortest path, and length L, taking PVj = (Vj, L) as the node of entry-edge list; if no path, take (Vj, L) = (∞, ∞), set a table header field of the entry-edge empty.

For the unique, two ways to prove, first of all, the unique of L1, guaranteed by the Dijkstra algorithm, the shortest path length is unique. The proof is as following: without considering the edge-node order of edge-entry list, entry-edge list is unique for any node Vi, that is, the set of edge-node is unique. In fact, suppose it is not unique, then exists at least two groups foreword-order-set of nodes that are not completely the same. Say they are \{Vp1\} and \{Vp2\}, and \{Vp1\} ≠ \{Vp2\}, but Vj not belongs to \{Vp1\}, then foreword node set V = Vj\{Vp1\} is the preceding node set of Vi, this is contradict with the assumed condition that \{Vp1\} is of all the nodes Vi of the shortest path. So it is unique.

Theorem 2. Set G = <V, E, W> as a directed graph, in the coding graph G (S, V0), L as a shortest path from V0 to V1, if L pass through the node V1, and L′ = L\{V1\}, then L1 is the shortest path from V0 to V1, L2 is the shortest path from V1 to Vi.

Proof: Without loss of generality, assume L1 is not the shortest path from V0 to V1, but Qi is. Then, there exists at least one a different node Vj, and length of Qi is less then length of Li, obviously L′ = Qj\{V1\} is a path from V0 to V1, pass through V1, the length of which is less then that of Li, this is contradict with the assumed condition that L is a shortest path from V0 to V1.

Theorem 3. In coding graph G (S, V0), search backward starting from the entry-edge list of PVi, if L′ = ∞, then no path exists, otherwise, at the most through n-1 (n is the number of nodes in graph G) steps could surely arrive the source node V0, and the sequence of nodes obtained V0, V1, V2, …, Vm, V0 is a shortest path from V0 to V1.

Proof: if the conclusion does not true, then in the sequence of searching path, there is at least one heavy node, such as Vk. In other words, exists at least one ring in the sequence of the search path. Assuming the node sequence ring is:
\[ V_i, V_k, V_1, \ldots, V_n, \ldots, V_0 \]

Obviously, the distance \([L_0, L_j]\) of the first two preceding nodes \(\{V_i, V_j\}\) of the repeated nodes \(V_k\), satisfying: \(L_0 + w_{ik} > L_0 + w_{jk}\), according to the definition of coding graph, the length of node \(V_i\) could only be \(L_0 + w_{ik}\), thus, \(V_i\) could not be the preceding node of \(V_k\), this is a contradiction, so, at most through \(n-1\) step could surely arrive the source node \(V_0\), referred to the coding graph \(G(S, V_0)\) and Theorem 2, the conclusion is proved to be true.

Theorem 4. Set \(G' = G(S, V_0)\) is the coding graph of \(G\) corresponds to source node \(V_0\). \(T(V_i)\) and \(T(V_j)\) are the two shortest path tree of \(G'\), corresponds to node \(V_i\) and \(V_j\). If \(V_j \in T(V_i)\), then \(T(V_j) \subseteq T(V_i)\). Otherwise, if \(T(V_i)\) is a shortest path tree in \(T(V_j)\), then \(T(V_j)\) is also a shortest path tree of \(G' = G(S, V_0)\).

Referred to the definition of shortest path tree, Theorem 4 be true.

Set \(G = G(S, V_0)\) as the n-node acyclic AOE network, then in the critical path coding graph \(G' = G(S, V_0)\), there are Theorem 1 ~ Theorem 4 similarly.

### III. CODING GRAPH

#### A. Data structures

Orthogonal list of traditional graph stored method be extended to make as long as the node on the graph \(G\) to modify certain fields, delete some edge node of entry-edge list, it generates the expansion of orthogonal list storage of coding graph \(G(S, V_0)\), the storage structure of graph \(G\) designed as follows:

1. **Node table header**: header from the node to all the order structure (vector) is stored in order to randomly access any node in or out side list.

   ```
   typedef struct Vertex //G (S, V0) node table structure
   {   int L;   //the path length
       struct Ede *firstin;//entry-edge table header pointer
       struct Ede *firstout;//out-edge table header pointer
     } * VERTEX;
   ```

2. **Edge table**: composed by the edges which representing adjacency relation between nodes in the graph.

   ```
   typedef struct Ede //structure of nodes in edge table
   {   int tailvex; //the subscript of starting node of edge
       int headvex; //the subscript of ending node of edge
       int weight; //weight of edge
       struct Ede *headlink; //pointer of entry-edge table
       struct Ede *taillink; //pointer of out-edge table
     } * PEDE;
   ```

#### B. Construction Algorithm

Construction algorithm of coding graph \(G(S, V_0)\) (coding graph construction, CGC for simple) is a recursive algorithm, first of all to establish orthogonal list storage structure graph, introducing queue, on the graph \(G\) to breadth-first coding, establish distance values of the node, modify entry-edge list to generate the orthogonal list storage representation of shortest path (or critical path) coding graph.

Start from source node \(V_0\), coding its adjacent node one by one, then starting from the adjacent coding sequence of their adjacent nodes, and to "first be encoded node’s adjacent node" before "last encoded node’s adjacent node" to be encoded, until the end of the recursive encoding, construction algorithm of shortest path (or critical path) encoding graph \(G(S, V_0)\) described as follows:

1. **Initialization**: Initialize the queue to be empty, set \(L_0\) of the source node \(PV_0\) to 0, for all other nodes initialize scalar \(L\) as \(\infty\) (or \(L\) initialize 0).

2. **Take the source \(PV_0\) into the queue**.

3. **If the queue is not empty, then take out the first node \(PV_i\), transfer \(i\) of node \(PV_i\) to the adjacent node, and take all the adjacent nodes with which the value of \(L\) has been modified into the queue. Delete non-shortest edge path (or critical path) edge node in the entry-edge list. Without loss of generality, assume current out-queue node is \(PV_i\), and of \(PV_i\) the one-way adjacent nodes are \((V_j1, V_j2), \ldots, (V_jn, Ljm)\), and out edge are \((i, j1), (i, j2), \ldots, (i, jk)\), the weight for corresponded adjacent edge are \(w_{ij1}, w_{ij2}, \ldots, w_{ijk}\) for any adjacent nods \((Vjm, Ljm)\) (\(m=1, 2, 3, \ldots, k\)) , make the mapping of transfer weight and take the minimum value (or maximum) \(\xi\):

   \[
   \xi (PV_i) = PV_jm = \begin{cases} 
   (Vjm, L_i+w_{ijm}) & \text{if } Ljm > L_i + w_{ijm} \\
   (Vjm, Ljm) & \text{or } Ljm < L_i + w_{ijm} \\
   \text{others} & 
   \end{cases}
   \]

   and, when \(Ljm < L_i + w_{ijm}\) (or \(Ljm > L_i + w_{ijm}\)), delete edge node \(E(i, jm)\) from entry edge list; when \(Ljm > L_i + w_{ijm}\) (or \(Ljm < L_i + w_{ijm}\)), take the node \(PV_jm\) into queue.

4. **Repeat step 3 until the queue becomes empty**.

#### C. Properties

The paper discussed only the propertis of shortest path coding graph, of critical path coding graph is similar.

Theorem 5. Set \(G' = G(S, V_0)\) as the coding graph of \(G\), \(T(V_i)\) is the shortest path tree corresponding with the node \(V_i\) in \(G'\). Then, if the distance value \(L_i\) of nodes \(V_i\) changes, then the graph \(G\), if and only if the new coding graph created from re-encoding the code graph \(T(V_i)\) constructs the code graph of \(G\). In other words, when and only when Access \(T(V_i)\) changes the value of distance and entry edge list of the corresponding node, the new code graph of \(G\) after changing will be constructed.

Proof: According to the definition of coding graph and shortest path tree, it is obviously by recursively derive that all nodes of \(T(V_i)\) need to re-encoding. If not, assume in graph \(G\), there exists at least one node \(V_k\) not belongs to \(T(V_i)\), of which the value of distance or entry edge list changed, then in \(G' = G(S, V_0)\), the shortest path from \(V_k\) to source node must pass through \(V_i\), according to definition of \(T(V_i)\), \(V_k\) shall belongs to \(T(V_i)\), that conflict with the assumption, therefore, only the node in \(T(V_i)\) need to be re-encoded.

By Theorem 5 that, when distance value \(L_i\) of nodes \(V_i\) changes, need only re-encoding the shortest path tree \(T(V_i)\) that rooted at node \(V_i\), to construct the new coding...
graph. This makes the existing graphs to be applied in the maximal scale, reduce time complexity. About re-encoding $T(V_j)$ have the following conclusions.

Theorem 6. Set $T(V_j)$ as the shortest path tree rooted at $V_j$ in $G' = G(S, V_0)$, then,

1. If the distance value $L'_{j}$ of $V_j$ changed to be smaller (i.e., $L'_{j} < L_j$), then for the internal node $T1 = \{PV \}$ edge node including both the edge of $T(V_j)$ and $E(i,j)$ in the entry edge list, need only modify the distance value, that is for any $V_f \in E(T(V_j))$, change the distance value of node $V_k$ into $L_k = L_k + L'_{j} - L_j$. While on the border node $T2 = \{PV \}$ edge node including non-$T(V_j)$ edge in entry edge list {, in addition to modify the distance value, but also remove all the non-$T(V_j)$ edge nodes in the entry edge list. The resulting graph is the code graph of changed graph $G$.

2. If the distance value $L'_{j}$ changed bigger, (i.e., $L'_{j} > L_j$), then starting from the node $V_j$, re-encoding recursively for $T(V_j)$, we must consider the convex of $T(V_j)$, the resulting graph is the coding graph of changed graph $G$.

Proof: Without loss of generality, as long as proof the adjacent node, because the proof can be repeated recursively for all nodes. Suppose node $V_i$ is the adjacent node of $V_j$, and both of them belong to $T2$, and $E(i,j)$ and $E(f,k)$ are two edges in the entry edge list of $V_i$, where $E(f,k)$ does not belong to $T(V_j)$ (that is, $V_f$ is a node of $G(S, V_0)$, but not $T(V_j)$), because the shortest path value $L_k$ from $V_k$ to $V_i$ via $V_f$ is less than the original value $L$ (the shortest path via node $V_i$), so, $V_i$ is no longer the preceding node of $V_k$, $E(f,k)$ must be removed from entry edge list. For node $V_k$ on $T1$, from the definition of $T(V_j)$ could know clearly that is enough only modify the distance value. Then according to Theorem 5, Theorem 6 (1) is proved to be true.

As for Theorem 6 (2), similarly if the adjacent node on $V_j$ can be proved is enough. Suppose $V_k$ is the adjacent node of $V_j$, if $V_k$ has another one entry edge $E(f,k)$ not belonging to $T(V_j)$, and $V_k \in E(T(V_j))$, because the distance value $L'_{j}$ of the tree’s root node $V_j$ getting bigger, so, when $L_k+ w_{f,k} < L_{j} + w_{f,j}$, must add the edge $E(f,k)$ into the entry edge list of $V_k$, and remove $E(i,j)$. In other words, when re-encoding $T(V_j)$ recursively, it is necessary to consider the convex of $T(V_j)$, then according to Theorem 5, Theorem 6 (2) proved.

The dynamic shortest path algorithm could be simplified as code graph re-construction algorithm when single edge weight changed. From Theorem 6 could get the following inference.

Inference 1. Set $E(i,j)$ as a directed edge of code graph $G(S, V_0)$, if weight $w'_{ij}$ changed to be bigger (i.e $w'_{ij} > w_{ij}$), then

1. When $E(i,j)$ is the only entry edge of node $V_j$ in $G$, then the distance value $L'$ of root node $V_j$ of shortest path tree $T(V_j)$ become bigger, the code graph constructed from re-encoding $T(V_j)$ becomes new weight of $E(i,j)$ to transfer to $V_j$ ($L'_{j} = L_{j} + w'_{ij}$), therefore, $E(i,j)$ could not be edge node of the entry edge list, so could only add the distance value $L_j$ of node $V_j$ with the new weight of $E(i,j)$ to transfer to $V_j$ ($L'_{j} = L_{j} + w'_{ij}$), i.e., the distance value of root node $V_j$ of $T(V_j)$ becomes smaller, according to Theorem 6 (1), Inference 2 (1) was true.

In the coding graph $G(S, V_0)$, the node $V_j$ has more than one entry edge, that means where are more than one shortest paths from $V_0$ to $V_j$; because the weight $w_{ij}$ of $E(i,j)$ becomes smaller. Therefore, the length value from $V_0$ via $V_i$ to $V_j$ becomes smaller, this path is the only shortest path from $V_0$ to $V_j$, and as for the definition of coding graph, node $V_j$ has only entry edge list for single edge node, so all other entry edge nodes must be removed from the entry edge list of $V_j$, retain only $E(i,j)$ of which the weight value becomes smaller, then according to Theorem 6(1), inference 2(2) is true.

IV. PATH ALGORITHM

A. Static Path Algorithm

After construction of Coding graph $G(S, V_0)$ it is easy to find all the shortest path (or critical path) and the length from the source node $V_0$ to other node $V_k (k=1, 2,$
... n-1) in the graph G, obviously in coding graph \( G(S, V_0) \), if \( l_i \) in node \( P(V_i, L_i) \) is a non-0 finite value, then it is the length of the shortest path from \( V_i \) to the source node \( V_0 \), otherwise no path exists. Thus, solving the shortest path (or critical path) algorithm, described as follows:

1. Create the expanded representation of orthogonal list of graph \( G \)
2. According to the shortest path (or critical path) construction algorithm for coding graph, construct the expanded representation of orthogonal list of coding graph \( G(S, V_0) \).
3. In the expanded representation of orthogonal list of coding graph \( G(S, V_0) \), for any node \( P(V_i, L_i) \) if the factor \( L_i \) is a non-0 finite value, then it is the shortest path (or critical path) length from source node \( V_0 \) to \( V_i \), while the tailvex value of entry-edge list node shall be the subscripts of the preceding node of \( V_i \) in the shortest path (or critical path). By recursive searching for each node in entry-edge could find all shortest path (or critical path). Otherwise, no path exists.

B. Dynamic Path Algorithm

When the network environment changes, i.e., in the weighted graph \( G=\langle V, E, W \rangle \) some of the edge weights changed, dynamic algorithm of the path can be reduced to the path algorithm of single weight change.

Obviously, the key of dynamic algorithm is the dynamic construction of coded graph, however, when the edge weights change, the removed edge node might re-enter into the list again, so, when constructing code graphs, if simply do delete operation, the deleted edge nodes need to be re-searched, resulting decreased efficiency of the algorithm. To solve this problem, add a flag field into the edge list structure, change the "delete" operation for the edge node from entry edge list into the operation “validate the delete flag”; and the operation for the edge node “add into entry edge list” change to the operation “invalidate the delete flag” in the entry edge list. Therefore, in the constructing of coding graph, just do the operation of setting delete flag field into "on(validate)/off(invalidate)", remain the original entry edge list not changed.

This article only gives a dynamic shortest path algorithm, the critical path of the dynamic algorithm completely similar.

a) The expansion of edge list structure

Expansion of edge list structure is as follows:

```c
typedef struct Ede {
    int tailvex; //subscript of the edge starting point
    int headvex; //subscript of the edge ending point
    int weight; //weight of edge
    struct Ede * headlink; //entry edge list pointer field
    struct Ede * taillink; //out edge list pointer field
} Ede;
```

b) The dynamic shortest path algorithm.

Set \( G = (V, E, W) \) as a code graph of weighted graph \( G = (V, E, W) \), weight value of the edge \( E(i, j) \) on node \( V_i \), \( V_j \) changed from \( w_{ij} \) into \( w'_{ij} \), according to the character of code graph, for single edge weighting change, shortest path algorithm is described as follows:

1. If \( E(i, j) \) is not an edge of \( G(S, V_0) \), then turn to step 3.
2. If \( w_{ij}' \) become bigger (\( w_{ij}' > w_{ij} \)), when \( E(i, j) \) is the only edge node in the entry edge list of node \( V_j \), turn to step 6. Otherwise, the entry edge list of \( V_j \) turn on deleted flag of \( E(i, j) \) in the entry edge list, and then turn to step 7.
3. If \( w_{ij}' \) become smaller (\( w_{ij}' < w_{ij} \)), then turn to step 8.
4. If \( w_{ij}' \) become smaller (\( w_{ij}' < w_{ij} \)), when \( L_i + w_{ij}' > L_j \), turn to step 8, otherwise turn to step 5.
5. If \( L_i + w_{ij}' < L_j \) in the entry edge list of node \( V_j \) turn on deleted flag for all other edge nodes, turn off only the deleted flag for \( E(i, j) \), then turn to step 6. If \( L_i + w_{ij}' = L_j \) then turn off the deleted flag for \( E(i, j) \) and turn to step 7.
6. Transfer the distance value \( L_i \) of node \( V_i \) and the weight value \( w_{ij}' \) of edge \( E(i, j) \) to node \( V_j \), which is \( L_i = L_i + w_{ij}' \); according to Theorem 6, calling CGC algorithm processing the distance value \( L \) and entry edge list on shortest path tree \( T(V_j) \), construct the coding graph \( G(S, V_0, G) \) with \( G \) changed.

7. Starting from any code \( V_i \) in the graph \( G(S, V_0) \), began with entry edge list do reverse-search recursively to find all shortest paths and their lengths from source node to any node in the graph, then turn to Step 9.
8. Coding graph \( G(S, V_0) \) did not change, no changes in the shortest path.
9. Algorithm end.

V. ALGORITHM ANALYSIS

A. The consumption of storage space

Set \( G = (V, E, W) \) is a n-node directed weighted graph, Dijkstra algorithm adjacency matrix representation, need to provide auxiliary array dist \([n]\), each element of the array includes two word Section: len field is distance from the source node \( V_0 \) to other, per field value is the order number from \( V_0 \) to the previous node; it needs a total of \( n^2+2n \) basic memory units. In this paper, graph \( G \) is stored with the expansion of orthogonal list, header array needs 3n storage units, edge table needs 5e (e is the number of edges in graph \( G \)) memory units, Total requirement is therefore 3n+5e basic memory units, apparently, in the dynamic algorithm when single edge weight changing requires 3n+6e basic memory units, apparently, save more storage space compared with traditional method.

B. Time Complexity

In the construction algorithm of coding graph \( G(S, V_0) \), each node in \( G \) get into the queue at most one time, so the number of nodes into the queue is not greater than \( n \). When node \( V_i \) goes out of the queue, the internal cycling number of dealing with its neighboring nodes is equal to the out-degree \( d_i \) of node \( V_i \). As the total time
complexity accessing to all nodes of the adjacent node is \(O(d_0+d_1+d_2+\ldots+d_{n-1})=O(e)\), so the time complexity of structural coding graph \(G(S, V_0)\) is \(O(n^2)\), superior to the traditional method of time complexity \(O(n^2)\).

VI. ALGORITHM APPLICATION

Set \(G_2=(V, E, W)\) as 9-node weighted directional graph, shown as Figure 2 (A) below. Without loss of generality, to solve the shortest path and the length from the node \(V_0\) to other node, steps as following:

1. Create the expansion of orthogonal list representation of \(G_2\), shown as Figure 2(B). Field \(L\) be set value 0 in the table header (node \(V_0\)) or \(\infty\) (other node).

2. According to the construction algorithm of the shortest path coding graph, the orthogonal list representation of construction coding graph \(G_2(S, V_0)\), shown as Figure 2(C), obviously is the sub-graph of \(G_2(S, V_0)\) in Figure 2(B) that deleted \(E(1, 4), E(4, 7)\) and \(E(6, 8)\) from the entry-edge list of node \(V_4, V_7, V_8\).

3. In the expansion of orthogonal list representation of shortest path coding graph \(G_2(S, V_0)\), search from entry-edge of \(V_i\) \((i=1, 2, \ldots, 8)\), finish at the source node \(V_0\), could find all shortest path from \(V_0\) to all node \(V_i\) \((i=1, 2, \ldots, 8)\), information shown in Table 1. where has two shortest paths from \(V_0\) to \(V_4: V_0, V_2, V_4\) and \(V_0, V_3, V_4\), the length is 5; from \(V_5\) to \(V_6\) has two shortest paths: \(V_0, V_2, V_4, V_6\) and \(V_0, V_3, V_4, V_6\), with length 14.

VII. CONCLUSION

The article breakthrough traditional thinking of the solution path algorithm, by introducing the concept of coded graph, abstract the shortest path and critical path issues as the same mathematical model to describe and solve. Present a new path algorithm based on coding graph, and find the dynamic algorithm to solve shortest path when single edge weight changed. Compared with the existing shortest path algorithm is not only simple and intuitive, but also need storage space only \(3n+5e\) (dynamic algorithm is \(3n+6e\) basic memory units, less then in traditional method \(n^2+2n\). Time complexity reduced to \(O(n+e)\). That could efficiently, simply find the shortest path and length from any node to all other node in the graph, has a good adaptability.

In this paper, graph-based coding algorithm is the expansion of [21] for solving the critical path algorithm. More detailed study for the character of the coding graph, the expansion applied to time-dependent dynamic complex network shortest path algorithm, that is, the dynamic algorithm in complex case like add and remove nodes will be the subject of further study.

ACKNOWLEDGMENT

I would like to thank all those who helped in the preparation of this paper. In particular, I am grateful to Prof. HuJun Bao for his constructive suggestions. It is supported by the Open Project Program of the State Key Lab of CAD&CG (A1015), Zhejiang University.

REFERENCES

achieved inventive and creative Award by Technological Information System of Unit Nation (China Bureau), now study instruct at shenzhen Polytechnic. Typical works includes:


His research interests computer graphics and algorithm analysis.

![Weighted directed graph G2](image)

(A) Weighted directed graph G2

![Expansion of orthogonal list representation of G2](image)

(B) Expansion of orthogonal list representation of G2

![Expansion of orthogonal list representation of coding graph G2(S, V0)](image)

(C) Expansion of orthogonal list representation of coding graph G2(S, V0)

Figure 2. Expansion of orthogonal list representation of graph G2 and coding graph G2(S, V0).

Table 1. Information of coding Graph G2(S, V0)

<table>
<thead>
<tr>
<th>Source node</th>
<th>Node V_i</th>
<th>Node of entry-edge list of V_i</th>
<th>Length of the shortest path from V_0 to V_i</th>
<th>Shortest path from V_0 to V_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_0</td>
<td>E(0, 1)</td>
<td></td>
<td>6</td>
<td>(V_0, V_1)</td>
</tr>
<tr>
<td></td>
<td>E(0, 2)</td>
<td></td>
<td>4</td>
<td>(V_0, V_2)</td>
</tr>
<tr>
<td></td>
<td>E(0, 3)</td>
<td></td>
<td>2</td>
<td>(V_0, V_3)</td>
</tr>
<tr>
<td>V_1</td>
<td>E(2, 4), E(3, 4)</td>
<td></td>
<td>5</td>
<td>(V_0, V_2, V_3, V_4)</td>
</tr>
<tr>
<td></td>
<td>E(3, 5)</td>
<td></td>
<td>4</td>
<td>(V_0, V_3, V_4)</td>
</tr>
<tr>
<td>V_2</td>
<td>E(4, 6)</td>
<td></td>
<td>14</td>
<td>(V_0, V_3, V_4, V_5)</td>
</tr>
<tr>
<td></td>
<td>E(5, 7)</td>
<td></td>
<td>8</td>
<td>(V_0, V_3, V_4, V_6)</td>
</tr>
<tr>
<td>V_3</td>
<td>E(7, 8)</td>
<td></td>
<td>12</td>
<td>(V_0, V_3, V_4, V_6)</td>
</tr>
</tbody>
</table>