Journal of Southeast University (English Edition) Vol. 27, No. 3, pp. 316 – 321 Sept. 2011 ISSN 1003—7985

Implementation and comparative testing of turn-based algorithm for logit network loading

Gu Cheng Ren Gang

(School of Transportation, Southeast University, Nanjing 210096, China)

Abstract: In order to evaluate the practicality and effectiveness of the turn-based algorithm for logit loading (TALL), the TALL is implemented using C++, and it is compared with a combination of the network-expanding method and the Dial algorithm based on the analysis of algorithm procedures. The TALL uses the arc-labeling shortest path searching, bidirectional star and the deque structure to directly assign the traffic flow, while the Dial algorithm should be used in an expanded network. The test results over realistic networks of eight cities show the superior performance of the TALL algorithm over the combination of the network-expanding method and the Dial algorithm, and the average processing time is reduced by 55.4%. Furthermore, it is found that the operational efficiency of the TALL relates to the original densities of the cities. The average processing time is reduced by 65.1% when the original density is about 14%, but the advantage of the TALL is not obvious with the increase in the original density.

Key words: TALL algorithm; network expanding; deque structure; bidirectional star; arc-labeling shortest path searching

doi: 10.3969/j.issn.1003 – 7985.2011.03.018

Optimization of traffic management, as an effective method to reducing traffic jams, depends on traffic assignment to a large extent. According to the flow aggregation level in equilibration, the traffic assignment algorithms can be classified as route-based or origin-based[1]. In the route-flow-based algorithm, the Dial algorithm[2] which is also named the STOCH algorithm can effectively implement logit-type network loading, i.e. logit traffic assignment in networks with fixed link costs. The Dial method obviates path enumeration and forms a part of various algorithms for finding stochastic user equilibrium (SUE) assignment in networks where congestion plays a role.

In addition, some algorithms also obviate path enumeration and find SUE assignment in networks[3-5]. The Bell method[6] applied a wide-scale matrix whose order is equal to the number of nodes, and, as a result, this method occupies too much memory space on computers for implementation. Akamatsu[7] designed a transition probability matrix and simply weighted the parths in a heuristic way without optimizing the total expected travel cost. Bertsekas et al.[8-9] focused on the original flows which were represented by proportions of traffic arriving at each node from their predecessor links. Yu[1] studied a class of bush-based algorithms (BA) for the user equilibrium (UE) traffic assignment problem, which produced precise solutions by exploiting the acyclicity of UE flows.

However, few algorithms directly deal with turn delays and turn flows, which are of great importance in congested urban road networks. Damberg et al.[10] provided a random assignment method based on column generation for directly solving the path flow, but the high requirement of storage space became the key obstacle of this type of method in application. An indirect method of converting actual turns in the original network into dummy links is widely applicable when turn delays and turn flows exist in the original network. Based on this method, the conventional algorithms such as the Dial algorithm can be used in an expanded network[11-14]. Nevertheless, the network-expanding method has serious disadvantages. Ren and Wang[15] developed a new algorithm: a turn-based algorithm for logit loading (TALL), which differs from the network-expanding method. The TALL algorithm not only agrees with the logit path-choice process but also takes the effect of turn delays on traffic assignment into account.

The purpose of this paper is to evaluate the practicality and effectiveness of the TALL by means of implementing the TALL algorithm using C++. And the TALL algorithm is compared with a combination of the network-expanding method and the Dial algorithm.

1 Algorithm Description

1.1 Deficiencies of network-expanding method

In the network-expanding method[16], real turns in the original network are converted into dummy links, and the conventional traffic assignment algorithms such as the Dial algorithm can be used in the expanded network.

We can discover some of the problems caused by expansion. First, the network-expanding method increases the size of networks too much by adding dummy links and nodes. In Fig. 1, nodes in the expanded network increase from one to seven and links increase from seven to sixteen. The expansion results in redundant computational time and increased memory for large-scale networks. Secondly, the original network topology which is indispensable for route-choice analysis is destroyed. In other words, the topologies of the original network and the expanded network should be maintained at the same time. Thus, the network-expanding method is surely not the best choice for network models with turn delays.

Received 2011-04-19.

Biographies: Gu Cheng (1984—), male, graduate; Ren Gang (corresponding author), male, doctor, researcher, rengang@seu.edu.cn.

Foundation items: National Science and Technology Action Program for Road Traffic Safety (No. 2009BAG13A05), the National Natural Science Foundation of China (No. 51078086).

Citation: Gu Cheng, Ren Gang. Implementation and comparative testing of turn-based algorithm for logit network loading[J]. Journal of Southeast University (English Edition), 2011, 27(3): 316 – 321. [doi: 10.3969/j.issn.1003 – 7985.2011.03.018]

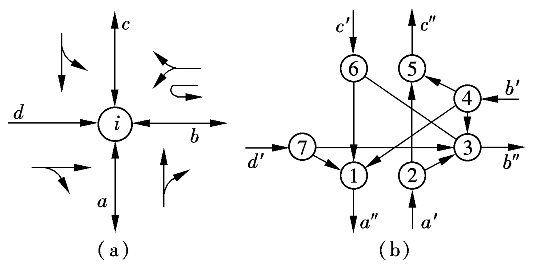


Fig. 1 Illustration of network-expanding. (a) A 4-leg intersection with link b allowing U-turn; (b) The corresponding expanded network

1.2 TALL algorithm

1.2.1 Notations

Given a transport network $G = (N, A)$ defined by the node set $N = \{i\}$, the directed link set $A = \{a = (i, j)\} \subseteq N \times N$, the turn set $T = \{a \to b\} \subseteq A \times A$ (here let $a \to b$ denote the turning movement from link a to link b), the trip origin set $R = \{r\} \subseteq N$, the trip destination set $S = \{s\} \subseteq N$ and the origin-destination (O-D) pair set $W = \{r\text{-}s\} \subseteq R \times S$. For each link $a \in A$, let $F(a)$ be the set of links adjacent to a and $B(a)$ the set of links adjacent from a. If $a = (i, j)$, $F(a)$ is actually the set of the outgoing links of node j, and $B(a)$ is the set of the incoming links of node i. Namely, for each link $a = (i, j) \in A$, let $F(a) = \{(j, m) \in A: m \in N\}$ and $B(a) = \{(n, i) \in A: n \in N\}$.

1.2.2 Concept of node-to-link path

In order to exploit turn delays to assign traffic flows, a new concept of the node-to-link path is given, which is a little different from the regular node-to-node path. For a directed graph G, a path from a node r to a link a is the one that starts from r, goes by intermediate nodes, and finally arrives at i and j in succession. In fact, a node-to-link path differs from a node-to-node path in nothing but in that its last node must be the head of this link and its second to last node must be the tail of this link.

Subsequently, we present the concepts of efficient turn and efficient path. Given an O-D pair r-s, for each link a, let $r(a)$ represent the shortest path cost from the origin node r to link a. The algorithm defines a turn $a \to b$ as "efficient" if $r(a) < r(b)$; that is, an efficient turn takes the traveler further away from the origin. Thus, a path between r-s is an "efficient" one if it only includes efficient turns. Note that a path is also regarded as a series of turns as well as a series of links or nodes. Furthermore, the costs of each path in networks with turn delays should include the delays of all the turns on that path as well as the costs of all the links on the path, and, hence, it can be expressed as

$$c_k^{rs} = \sum_a t_a \delta_{a,k}^{rs} + \sum_{a \to b} d_{ab} \phi_{ab,k}^{rs} \qquad \forall\, r\text{-}s, k \tag{1}$$

where c_k^{rs} is the travel cost of path k between the O-D pair from r to s; t_a is the travel cost (or referred to as generalized cost) of link a; d_{ab} is the delay of turn $a \to b$; $\delta_{a,k}^{rs}$ is the indicator variable of path-link incidence, satisfying $\delta_{a,k}^{rs} = 1$ if link a is on path k between the O-D pair r-s, otherwise $\delta_{a,k}^{rs} = 0$; $\phi_{ab,k}^{rs}$ is the indicator variable of path-turn incidence, satisfying $\phi_{ab,k}^{rs} = 1$ if turn $a \to b$ is on path k between the O-D pair r-s, otherwise $\phi_{ab,k}^{rs} = 0$.

It should be noted that equation $\phi_{ab,k}^{rs} = \delta_{a,k}^{rs}\,\delta_{b,k}^{rs}$ always holds because turn $a \to b$ belongs to path k between the O-D pair r-s if and only if two adjacent links constitute this turn.

1.2.3 Algorithm steps

For a complete logit-type network loading, the algorithm executes the following steps:

Step 1 Initialization

1) For each origin r, add a dummy node r' and a dummy link $\gamma = (r', r)$ with $t_\gamma = 0$, and then add a dummy turn $\gamma \to b$ with $d_{\gamma b} = 0$ for each $b \in F(\gamma)$. For each destination s, add a dummy node s' and a dummy link $\zeta = (s, s')$ with $t_\zeta = 0$, and then add a dummy turn $b \to \zeta$ with $d_{b\zeta} = 0$ for each $b \in B(\zeta)$.

2) Set $x_a = 0$ for each link a, and set $y_{ab} = 0$ for each turn $a \to b$, where x_a is the flow on link a; y_{ab} is the flow on turn $a \to b$.

Note that the dummy nodes, links and turns added in step 1 are used to map the node-oriented origins and destinations to certain links, because the proposed algorithm is link-oriented other than the traditional node-oriented. In fact, dummy link γ and ζ just act as origin r and destination s, respectively, from a link-oriented perspective. However, the dummy elements have a small size proportional to the number of the trip origins and destinations, and actually are not indispensable to the algorithm.

Step 2 Preliminaries for current origin r

1) Compute the shortest path costs from r' to all the links. This yields $r(a)$ for each a, especially $r(\gamma) = 0$ and $r(\zeta) = \infty$, $\forall \zeta$.

2) Compute the likelihood L_{ab} for each turn $a \to b$, where

$$L_{ab} = \begin{cases} e^{-\theta(d_{ab} + t_b)} & r(a) < r(b) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Step 3 Forward pass to compute turn weights

Process links in an ascending order of $r(a)$, starting with γ. Namely, for each link a, compute the turn weight w_{ab} for each $b \in F(a)$, where

$$w_{ab} = \begin{cases} L_{ab} & a = \gamma \\ L_{ab} \sum w_{ca} & \text{otherwise} \end{cases} \tag{3}$$

Step 4 Backward pass to assign turn flows

Process links in a descending order of $r(b)$, starting with the most distant link. Namely, for each link b, compute the link flow due to r, x_b^r, and the turn flow due to r, y_{ab}^r, for each $a \in B(b)$, where

$$x_b^r = \begin{cases} q_{rs} & b \in \xi \\ \sum\limits_{c \in F(b)} y_{bc}^r & \text{otherwise} \end{cases} \tag{4}$$

$$y_{ab}^r = x_b^r \frac{w_{ab}}{\sum\limits_{c \in B(b)} w_{cb}} \tag{5}$$

where x_b^r is the flow on link b due to origin r; q_{rs} is the travel demand between the O-D pair r-s; and y_{ab}^r is the flow on turn $a \to b$ due to origin r.

Step 5 Flow accumulation

1) Set $x_a = x_a + x_a^r$ for each link a, and set $y_a b = y_a b +$

$y_a b^r$ for each turn $a \to b$.

2) If r is the last origin, stop; otherwise, let r be the next origin and go to step 1.

2 Algorithm Implementation

Detailed implementation of the network-expanding method, mainly data structures and algorithmic steps, is rare in existing literature. In this part, the network expanding procedure, bidirectional star and shortest path searching for both the TALL algorithm and the combination of the network-expanding method and the Dial algorithm are introduced in detail. Some codes of C + + style are to be demonstrated when necessary.

2.1 Network expanding procedure

2.1.1 Key steps

Step 1 Establish dummy nodes

Split each node up into some dummy nodes and update the head node or the tail node of the links concerned. The corresponding relationship between the original nodes and the nodes in the expanded network is shown as follows:

```
for( i = 1;  i < = n;  i + + )
    {  din = GetInArcs( i, inArcs) ;
       dout = GetOutArcs( i, outArcs) ;
        if( din = = 0)
       {
            relation[ i] [ 0]  =  + + num;
       }
       else if ( dout = = 0)//din! = 0
       {
             + + num;
            for( count1 = 0; count1 < din;  ount1 + + )
            {
                relation[ i] [ count1]  = num;
            }
            }
       else//din! = 0,  dout! = 0
       {
            for( count1 = 0; count1 < din; count1 + + )
            relation[ i] [ count1]  =  + + num;
       for( count2 = 0; count2 < dout; count2 + + )
          relation[ i] [ din + count2]  =  + + num;
       }
    }
```

Step 2 Establish dummy links

Add dummy links for each turn, as shown in Fig. 2, where d_{in} and d_{out} are backward and forward adjacent nodes.

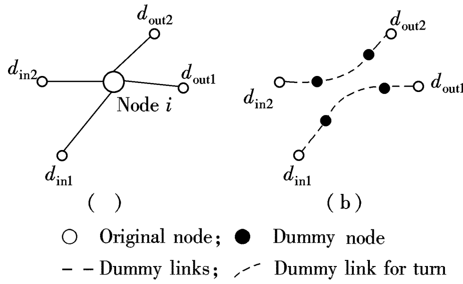


Fig. 2 Illustration of converting original node into dummy link for turn. (a) Node i and adjacent nodes in origin network; (b) Expanded network with corresponding dummy links for turn

2.1.2 Topological relationship analysis

Tab. 1 lists the numbers of nodes, links and turns in networks used for different methods. From Tab. 1, we can find that the network-expanding method completely destroys the topological relationship of the origin network. And the network used in the TALL algorithm maintains the same topological relationship.

Tab. 1 Comparison of topological relationship

Network	Node number	Link number	Turn number
Original network	n	m	t
Expanded network	$n + \sum_{1}^{n} [d_{in}(i) + d_{out}(i)]$	$m + \sum_{i=1}^{n} [d_{in}(i) \times d_{out}(i)]$	N/A
Network used in TALL	$n + o + d$	$m + o + d$	$t + \sum_{r \in R} d_{out}(r) + \sum_{s \in S} d_{in}(s)$

Note: n, m, t, o and d mean the number of nodes, links, turns, origins and destinations in the original network, respectively. Usually, o and d are much less than n.

In regular transportation networks, the in-degree and the out-degree of each node usually approximate to three, which means $d_{in} \approx d_{out} \approx 3$. As a result, the node number and the link number in the expanded network increase to $7n$ and $m + 9n$, while those in the network used in the TALL algorithm are much smaller.

2.2 Bidirectional star

Because forward and backward adjacency relationships of nodes and links are necessary to the Dial algorithm and the TALL algorithm, the expanded network is represented by a node-based bidirectional star, which consists of a forward star (i. e. an adjacent list stored in an array structure) and a backward star, while the original network is represented by a link-based bidirectional star which incorporates turning movement.

2.2.1 Node-based bidirectional star for the Dial algorithm

In the node-based star, the nodes are stored in the array and a single list is established to record adjacent nodes for each node. The forward star is established as follows:

Step 1 Read a pair of O-D from the O-D list.

Step 2 If the current node i is the destination, insert a dummy arc.

Step 3 If the current node is the origin or any other node but not the destination, insert a dummy link from the origin to the destination.

Step 4 If the last O-D pair is read, stop; otherwise, go back to step 1.

The complementation code is shown as follows:

```
ifile≫tfnode≫ttnode≫tcost;
if(tfnode! =cnode)//cnode = current node
{
    for(i = cnode; c < tfnode; i + + )
  {
    If(nlist[i]. dflag! = 0)//cnode is destination
        {
            llist[a]. fnode = i;
        llist[a]. tnode = n + o + nlist[i]. dflag;
        }
        cnode = tfnode;
        llist[a]. fnode = tfnode;
        llist[a]. tnode = ttnode;
  }
    }
    else
    {
    llist[a]. fnode = tfnode;
    llist[a]. tnode = ttnode;
    }
}
```

When the forward star is established, the corresponding backward star is formed at the same time. A reverse cycle of forward search can be used to form the list.

2.2.2 Link-based bidirectional star for TALL algorithm

Based on a node-based star, a link-based star is established as follows:

```
for(i = 1; i < = N; i + + )
{
    $k_1$ = GetOutArcs(i, outArcs);//adjacent node of i
    for(j = 0; j < $k_1$; j + + )
    {
        a = outArcs[j];//list of adjacent nodes of j
        llist[a]. fptr = u;
        $k_2$ = GetOutArcs(llist[a]. tnode, inArcs);
        for(k = 0; k < $k_2$; k + + )
        {
            tlist[u]. farc = a;
            tlist[u]. tarc = inArcs[k];
            u + + ;
        }
    }
}
```

2.3 Shortest path searching

Shortest path searching is a core procedure in the network loading algorithm. A label-correcting shortest path algorithm using the deque structure is implemented in the Dial algorithm, and an arc-labeling version is used in the TALL.

2.3.1 Deque manipulation for the Dial algorithm

Deque Q can be viewed as a combination of stack S and queue Q'. Elements insertion and deletion follow two principles: 1) When a node is added for the first time, list it at the bottom of queue Q'. If a node is added for the second time, list it at the top of stack S and mark the value as -1. 2) When S is empty, delete the top element in Q', otherwise delete the top element of S.

The practical steps are shown as follows:

Step 1 Initialization

For each node, set the minimum cost at infinity and set deque empty.

Step 2 Seek the minimum cost

At the beginning of a cycle, regard the current node i as the top element of the deque. Then seek the adjacent nodes j of i, and choose the minimum cost of link (i, j).

Step 3 Add j into the deque.

1) If j is added into the deque for the first time, list j at the bottom of queue Q' and let top $=j$.

2) If the marked value of j is -1, then let top $=j$ directly.

Step 4 If the top element is the last node, stop; otherwise, go back to step 2.

The deuqe structure improves the efficiency of the shortest path algorithm due to the following advantages: 1) Avoid enumeration because the shortest path searching starts from one node instead of from all of them; 2) Only the nodes which do not exist in stack S are to be scanned, so repeated calculation will be avoided; 3) Nodes are stored in order and using the last-in-first-out (LIFO) principle, the search efficiency can be improved.

2.3.2 Deque manipulation for TALL algorithm

Based on the deque structure used in the Dial algorithm, we take turn delays into consideration. From this point of view, the minimum cost should include link costs and turn delays. The pseudocode form of deque manipulation used in the TALL algorithm is shown as follows:

```
While (Deque is not empty)
{//Initialization
 ⋮
//Seek the minimum cost
For (access all adjacent links of a)
{
  If (SPTlist[b]. dist > SPTlist[a]. dist + llist[b]. cost + tlist[u]. cost), then (SPTlist[b]. dist = SPTlist[a]. dist + llist[b]. cost + tlist[u]. cost
  If b is added into deque for the first time, then bottom = b and top = b; if b is not added for the first time, then top = b
}
```

3 Performance Testing

3.1 Data set and testing environment

The performance of the TALL algorithm is evaluated in comparison with the Dial algorithm over a set of real networks which contain the number of nodes, links and turns, as shown in Tab. 2. The data set of two Chinese networks and six popular networks available on the website (http://www.bgu.ac.il/~bargera/tntp/) are used for testing.

This evaluation is performed as follows: For each network, a logit network loading is implemented by the Dial algorithm with the expanded network, and the other loading is implemented by applying the TALL algorithm with the original network. To achieve better time efficiency and guarantee the comparability, we choose sub-algorithms and data structures carefully as mentioned in section 2.

In addition, the evaluation is performed under user mode of Windows XP, and the platform is a work station with a 2.93 GHz Inter(R) E7500 processor and 2 GB memory.

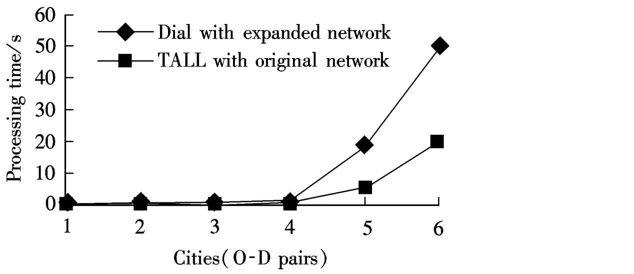
Tab. 2 Characteristics of test network

Network	Number of nodes	Number of links	Number of turns	O-D pair	Original density of O-D pairs/%
Sioux Falls	24	76	254	24 × 24	100
Zhenjiang	252	814	2 810	65 × 65	25. 8
Anaheim	416	914	2 646	38 × 38	9. 1
Zhengzhou	317	1 076	3 862	100 × 100	31. 5
Winnipeg	1 052	2 848	8 434	147 × 147	14. 0
Chicago sketch	933	2 950	13 116	387 × 387	41. 5
Chicago regional	12 982	39 018	135 302	1 790 × 1 790	13. 8
Philadelphia	13 389	40 003	136 483	1 525 × 1 525	11. 4

3. 2 Results and analysis

3. 2. 1 Evaluation of execution time

In this evaluation, we obtain the average execution times required for a logit network loading by two solutions, i. e. the TALL algorithm on the original network and the Dial algorithm with network expanding by a great deal of testing. From Fig. 3, it is apparent that the TALL algorithm has better performance in execution time.



1— Sioux Falls (24); 2—Anaheim (38); 3—Zhenjiang (65)
4— Zhengzhou (100); 5—Winnipeg (147); 6— Chicago sketch (387)

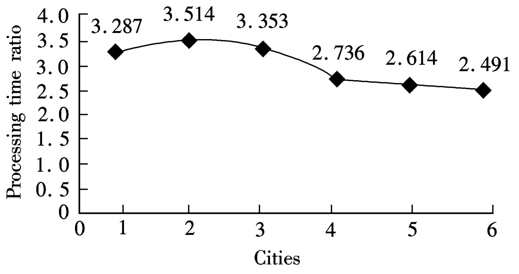
Fig. 3 Comparison of execution time

Meanwhile, the results of tests on large-scale networks such as Chicago regional and Philadelphia show that the processing time ratios of the two algorithms are 3. 287 and 3. 514, which indicates that the TALL algorithm is effective for large-scale networks.

3. 2. 2 Analysis of original density

For comparison, we calculate the ratio of execution times of the Dial algorithm and the expanded network to that of the TALL algorithm, as shown in Fig. 4. Since all the figures of the ratio exceed one, they demonstrate the superiority of the TALL algorithm over the Dial algorithm/expanded network combination on eight different networks of various scales from small to large. Meanwhile, the TALL algorithm also has good performance in large scales of networks. From Fig. 4, the execution time ratios (combination of Dial and network expanding/TALL) are 3. 287 and 3. 514 when O-D pairs are 1 525 and 1 790. This is because wide networks require more time to expand original networks.

Moreover, we find that the advantage of the TALL algorithm is not obvious with the increase in the original density. This is because, in that case, additional execution times for converting the node-oriented origins and destinations to the corresponding links will increase much more, and more time will be spent for calculation. According to the data in Tab. 2, it is clear that the ratio reaches a peak of 3. 514 when the original density is 13. 8%, and it has a better expansibility when the original density is about 14%.



1—Philadelphia; 2—Chicago region; 3—Winnipeg
4— Zhenjiang; 5— Zhengzhou; 6— Chicago sketch

Fig. 4 Processing time ratio of two solutions

4 Conclusion

The TALL algorithm deals with turn delays and turn flows in an explicit and direct way and avoids the disadvantages of the conventional method. The testing results on some real networks confirm the superior performance of the TALL algorithm, in terms of time efficiency, over the combination of the network-expanding method and the Dial algorithm. Furthermore, the TALL algorithm has good performance on large-scale networks and better performance when the original density is about 14%.

References

[1] Yu N. A class of bush-based algorithms for the traffic assignment problem [J]. *Transportation Research Part B*, 2010, **44**(1): 73 - 89.

[2] Dial R B. A probabilistic multipath traffic assignment algorithm which obviates path enumeration [J]. *Transportation Research*, 1971, **5**(2): 83 - 111.

[3] Powell W B, Sheffi Y. The convergence of equilibrium algorithms with predetermined step sizes [J]. *Transportation Science*, 1982, **16**(1): 45 - 55.

[4] Patriksson M. *The traffic assignment problem*: *models and methods* [M]. Utrecht, The Netherlands: V. S. P. Intl Science, 1994.

[5] Leurent F M. Curbing the computational difficulty of the logit equilibrium assignment model [J]. *Transportation Research Part B*, 1997, **31**(4): 315 - 326.

[6] Bell M G H. Alternative to Dial's logit assignment algorithm [J]. *Transportation Research Part B*, 1995, **29** (4): 287 - 295.

[7] Akamatsu T. Cyclic flows, Marcov process and stochastic traffic assignment [J]. *Transportation Research Part B*, 1996, **30**(5): 369 - 386.

[8] Bertsekas D P, Gafni E M, Gallager R G. Second derivative algorithms for minimum delay distributed routing in networks [J]. *IEEE Transactions on Communications*, 1984, **32**(8):

911 − 919.

[9] Bar-Gera H. Origin-based algorithm for the traffic assignment problem [J]. *Transportation Science*, 2002, **36**(4): 398 − 417.

[10] Damberg O, Lundgren J T, Patriksson M. An algorithm for the stochastic user equilibrium problem [J]. *Transportation Research Part B*, 1996, **30**(2): 115 − 131.

[11] Liu H S, Fu Y. Stochastic user equilibrium assignment model based on travel trait [J]. *China Journal of Highway and Transport*, 2004, **17**(4): 93 − 118. (in Chinese)

[12] Meneguzzer C. An equilibrium route choice model with explicit treatment of the effect of intersections [J]. *Transportation Research Part B*, 1995, **29**(5): 329 − 356.

[13] Wang F Y, Pan F Q, Zhang L X, et al. Optimal path algorithm of road network with traffic restriction [J]. *Journal of Traffic and Transportation Engineering*, 2005, **5**(1): 92 − 95. (in Chinese)

[14] Ren Gang, Wang Wei, Deng Wei. Shortest path problem with turn penalties and prohibitions and its solutions [J]. *Journal of Southeast University: Natural Science Edition*, 2004, **34**(1): 104 − 108. (in Chinese)

[15] Ren Gang, Wang Wei. A turn-based algorithm for solving the logit type network loading problem [C]//*Proceedings of* 2009 *IEEE International Joint Conference on Computational Sciences and Optimization*. Nanjing, China, 2009: 89 − 94.

[16] Ren Gang. *Traffic assignment models and algorithms with traffic management* [M]. Nanjing: Southeast University Press, 2007. (in Chinese)

基于转向的 Logit 网络分配算法实现与比较测试

顾 程　任 刚

(东南大学交通学院,南京 210096)

摘要:为了评价基于转向的 Logit 网络分配算法(TALL)的实用性和高效性,在分析 TALL 算法过程的基础上,运用 C ++ 实现了 TALL 算法,并与传统的 Dial 算法 + 网络扩展法进行比较测试. TALL 算法运用弧标号最短路径搜索、双向星形表和 Deque 结构,直接对道路网络进行流量分配,而不像 Dial 算法要在扩展后的路网上应用. 在实际 8 个不同大小城市的路网比较测试结果表明:TALL 算法在时间效率方面表现明显优于 Dial 算法 + 网络拓展法,平均运行时间减少 55.4% ;TALL 算法运算效率与城市路网起点密度有很大关系. 当起点密度在 14% 左右时,平均运算时间减少 65.1% ,但随着起点密度的增加,TALL 算法的优势变得不明显.

关键词:TALL 算法; 网络拓展; Deque 结构; 双向星形表; 弧标号最短路搜索

中图分类号:U491.1