文章编号：0253-374X(2012)11-1654-06

DOI:10.3969/j.issn.0253-374x.2012.11.011

基于路网压缩策略的改进 Highway Hierarchical 算法

蔡文学,周　兴,许　靖,钟慧玲

(华南理工大学 经济与贸易学院，广东 广州 510006)

摘要：针对 Highway Hierarchical 算法中存在的路网压缩成环问题、预处理数据存储问题和完整最短路计算问题，采用无环压缩策略、分层存储策略和局部最短路存储策略，对算法进行了改进. 广东省路网测试结果表明，改进后的算法在时间效率上约是原算法的 5 倍，在空间效率上约是原算法的 4 倍.

关键词：压缩路网；Highway Hierarchical 算法；路径规划

中图分类号：U495　　**文献标识码**：A

Improved Highway Hierarchical Algorithm Based on Contracted Network Strategy

CAI Wenxue, *ZHOU Xing*, *XU Jing*, *ZHONG Huiling*

(School of Economics and Commerce, South China University of Technology, Guangzhou 510006, China)

Abstract: In the pretreatment process, Highway Hierarchical (HH) algorithm faces such problems as the compressing of network into a ring road, the storage way of pretreatment data and a complete calculation of the optimal route. Non-cycle compressing, tiered storage and local shortest path storage are introduced to improve the efficiency of the algorithm. The test of the road network of Guangdong Province of China shows that with the improved HH algorithm, the computational efficiency increases by 5 times, and the search space reduces by 4 times.

Key words: contracted network; Highway Hierarchical algorithm; path planning

Dijkstra 算法[1]是计算路网中最短路的经典算法；但在大规模路网中，Dijkstra 算法的效率太低，远远不能满足应用需求. 针对 Dijkstra 算法的不足，出现许多改进算法.

最短路算法的改进主要分如下 3 类：双向搜索策略、目标导向策略和分层压缩策略. Dantzig 中提出双向策略[2]，从起点和终点同时开始搜索最短路，通过减少搜索空间来提高效率，但在最坏的情况下比单向搜索的效率差. 对于目标导向策略，Hart 提出 A * 算法[3]，通过估价函数调整搜索方向，减少搜索空间；Goldberg 提出基于 A * 算法、Landmark 和三角不等式的 ALT 算法[4-6]，通过预处理选择 Landmark 点，大大提高搜索效率. 目标导向策略虽然可以缩小搜索空间，但在大规模路网以及 Internet 环境的多用户请求下，其效率仍然不能满足实际需求. 分层压缩策略的思想是通过预处理来压缩路网，以降低搜索量，提高搜索效率. 对于分层压缩策略，Lipton 提出路网分区理论[7]，将路网分成很多片区，预先计算每个片区边界点之间的最短路，搜索只需在片区边界间进行，降低了搜索量；在此基础上，Schulz 提出 Multi-Level 算法[8]，Schultes 提出 Highway Hierarchical(HH)算法[9]，Muller 提出 Transit-Node 算法[10]. 此外，文献[11]提出考虑交叉路口延误的算法；文献[12]提出基于不同交通信息进行路径选择的算法. 改进过程中，学者们对策略进行了组合. Holzer 在文献[13]中组合了 A * 算法、双向搜索和 Multi-Level 算法；Delling 组合了 ALT 与分层策略[14]. 在改进算法中，主要解决两个方面的内容，一是改进搜索量，二是改进搜索空间. HH 算法作为其中一种改进算法，是近年来处理大规模路网的高效的路径规划算法.

1　HH 算法

HH 算法由预处理阶段和搜索阶段两部分组

收稿日期：2011-09-06

基金项目：广东省现代信息服务业发展专项(06120840B0450124 /2)；中央高校基本科研业务费专项

第一作者：蔡文学(1968—)，男，教授，硕士生导师，工学博士，主要研究方向为智能交通，智能物流. E-mail:ctwxcai@scut.edu.cn

通讯作者：周　兴(1986—)，男，工学硕士，主要研究方向为智能交通. E-mail: 542574623@qq.com

成，预处理采用 Contraction 算法[9]，搜索阶段采用 Dijkstra 算法. HH 算法的基本思想是通过预处理原始路网，将路网分层，从原始层到最高层逐层压缩，简化路网；然后，在搜索阶段进行分层搜索，从而达到提高搜索效率的目的. 算法的高效性在于通过耗费预处理的时间降低了搜索时间. 虽然预处理阶段需要耗费大量的时间，但搜索阶段时间却非常短，而算法的效率直接体现在搜索阶段. 采用 Contraction 算法保证效率，Dijkstra 算法保证准确性；HH 算法是目前为止在搜索效率和计算结果合理性上较好的算法.

1.1　Contraction 算法

1.1.1　主要思想

采取路网分层策略，将原始路网作为第 0 层，根据邻域内节点数 H（一般取 $H=40$）计算节点邻域半径，生成最短路树，将每条最短路的中间路段提升等级，初步形成第 1 层路网，中间路段的起点不能在最短路的终点邻域内，终点不能在最短路的起点领域内，如图 1 所示.

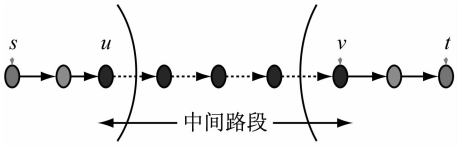

图 1　中间路段计算图

Fig.1　The computation of the central section of highway

去掉第 1 层路网中出入边较少的点，用虚拟边连接未去除的点（图 2），形成第 1 层压缩路网，如此迭代，形成多层压缩路网.

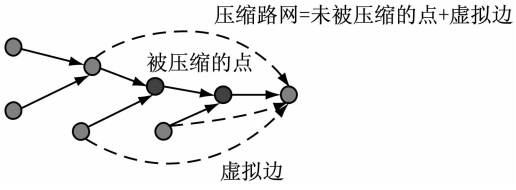

图 2　路网压缩图

Fig.2　Contracted road network

1.1.2　定义

设路网 $G=(V,E)$，假设将路网 G 分 L 层压缩，G 的分层路网为 $G_0, G_1, \cdots, G_{L-1}$，相应的压缩路网为 $G'_0, G'_1, \cdots, G'_{L-1}$，压缩路网也称 Core. V 为路网中节点的集合，v 为节点；E 为路网中边的集合，e 为边. V_l 为第 l 层路网点集；E_l 为第 l 层路网边集. V'_l 为第 l 层压缩路网点集；E'_l 为第 l 层压缩路网边集 $(0\leqslant l<L)$. $r_l(u)$ 为点 u 在第 l 层路网的邻域半径；$N_l(u)$ 为点 u 在第 l 层路网的邻域范围内节点集合，$N_l(u)=\{v\in V'_l \mid d_l(u,v)\leqslant r_l(u)\}$，$d_l(u,v)$ 为点 v 离 u 的距离. Bypassed node 为被压缩的点. B_l 为第 l 层被压缩的点集，$B_l\subseteq V_l$. Shortcut 为压缩点后产生的虚拟边. S_l 为由 shortcut 边构成的边集，如 shortcut 边 $(u,v)\in S_l$，其完整路径 $P=(u,b_1,\ldots,b_k,v)$，$u,v\in V'_l$ 且 $b_i\in B_l, 1<i<k$. Slack 为节点邻域半径与它到父节点距离之差.

1.1.3　算法步骤

Step1　根据路网 $G'_l=(V'_l,E'_l)$ 计算每层道路网络 $G_{l+1}=(V_{l+1},E_{l+1})$.

第一阶段，根据领域内点的个数 H（一般取 $H=40$），用 Dijkstra 算法，求出路网中每个点的邻域点集以及邻域半径.

第二阶段，从任意 $s\in V$ 开始，用 Dijkstra 算法搜索，生成最短路树. Reached node 表示算法的标号点，分两类，一类是永久标号点（settled node），一类是临时标号点（unsettled node）. Settled node 为最短路树中的点. 在搜索过程中，reached node 要么处于 active 状态要么处于 passive 状态. 起点是 active 的，reached node 设为 active 当且仅当其父节点均是 active 的. 若 settled node 点 p 满足规则：

$$s_1<p\wedge p\notin N(s_1)\wedge s_0\notin N(p)\wedge \mid P(s_0,p)\cap N(s_1)\cap N(p)\mid\leqslant 1$$

其中：s_1 为 s_0 的子节点；$\wedge$ 表示“且”；$P(s_0,p)$ 为从 s_0 到 p 的最短路上所有节点

此时，将点 p 设为 passive，如果搜索继续的话，它的后代均设为 passive，算法的终止条件为 unsettled 点中不存在 active 状态.

第三阶段，生成最短路树后，将满足条件的边提升等级. 考察根节点 s 到叶节点 t 的最短路 $(s,\cdots,u,v,\cdots,t)$ 上每条边 (u,v)，如果满足条件

$$u\notin N(t), v\notin N(s)$$

则将边 (u,v) 提升等级，每个点的最短树都经这样处理后，生成了新一层的路网. 为满足提升等级条件，从最短路树的叶子节点反向处理每个节点. 将每个节点的 slack 初始化为其邻域半径，从叶子节点开始反向迭代计算 slack，当时节点存在多个子节点时，取最小的 slack. 在计算过程中，将 slack 小于 0 的边提升等级，当节点进入起点邻域范围内时立即停止计算.

Step2　通过 Step1 生成的新的路网，求出路网中 core 的部分，生成压缩路网

$$G'_1=(V_l\backslash B_l,(E_l\cap(V'_l\times V'_l))\cup S_l).$$

考察路网中每一个点 v，判断它是否该被压缩

掉,如果满足条件

$$\text{shortcuts} \leqslant C[d_{\text{in}}(v)+d_{\text{out}}(v)]$$

即被设为 bypassed 点,其中 shortcuts 表示点被压缩后,它的父节点与子节点所产生的新边的数目,$d_{\text{in}}(v)$,$d_{\text{out}}(v)$分别表示点的入边数与出边数,C 为控制参数(一般取 $C=1$). 每个点经过如此处理后,生成初步简化路网. 复查简化路网,直到不存在 bypass 点为止.

Step3 判断是否达到最高层 L. 若已达最高层,转入 Step4;否则,转入 Step1.

Step4 将每层经处理的路网数据合并,生成虚拟路网

$$\Omega=\bigcup_{i=0}^{L-1} G_i=(V,E\bigcup_{i=0}^{L-1} S_i)$$

点和边在集合 V_l,V'_l,B_l,E_l,E'_l,S_l 中有相关从属信息,预处理阶段结束.

1.2 搜索算法

经过预处理阶段后,生成了分层压缩的路网数据. 基于压缩路网数据,采用双向 Dijkstra 算法进行分层搜索. Contraction 算法与 Djikstra 结合而成的 HH 算法与单纯的 Dijksta 算法的区别在于:前者的搜索阶段是基于经过预处理的压缩路网上进行的,而后者的却是在原始的复杂路网中进行的;前者的搜索量与搜索空间明显小于后者. HH 算法实现的伪代码参见文献[9].

2 HH 算法的问题分析

本文中,T 为搜索过程中临时标记点的 Open 表,P 为永久标记点的 Close 表,outJoint(v) 为点 v 的出边,inJoint(v) 为点 v 的入边,$|A|$为集合 A 的元素个数.

2.1 路网压缩过程的成环问题

在特定路网的局部区域,存在这样的情况:如图 3 所示,B 点只有一条出边 e,C 是 B 的子节点,且 B 恰好是 C 的子节点. 这种情形下,预处理算法压缩路网时,C 点满足了压缩规则,设为 bypass 点;而由于该点的父节点 B 正是其子节点,生成的虚拟边的起点和终点便成了同一点,产生了环路,如图 4 所示.

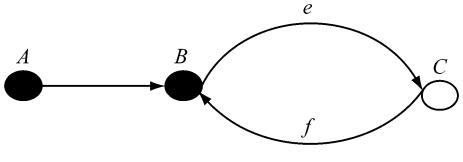

图 3 实际路网

Fig.3 Real road network

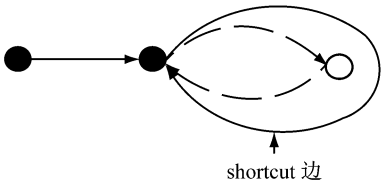

图 4 压缩成环路网

Fig.4 Contracted network with cycle road

搜索算法中部分伪代码为

u=deleteMin($\overleftarrow{T}$);

…;

foreach $e=(u,v)e=(u,v)\in\overrightarrow{E}$

do {…;

if v has been reached then decreaseKey($\overleftarrow{T}$,v,k);

else insert($\overleftarrow{T}$,v,k);

}

搜索到环路时,$u=v$,伪代码的第 5,6 句再次将 v 放入$\overleftarrow{T}$,导致再次执行第 1 句代码. 同一点的多次搜索,增加了搜索时间,带来效率问题.

2.2 完整最短路的计算问题

HH 算法在第一搜索结束后,得到的最短路并非完整最短路,其中包括部分 shortcut 虚拟边. 因此,HH 算法为计算出完整实际最短路,在路网中迭代地对 shortcut 边进行搜索,直到没有 shortcut 边,如图 5 所示.

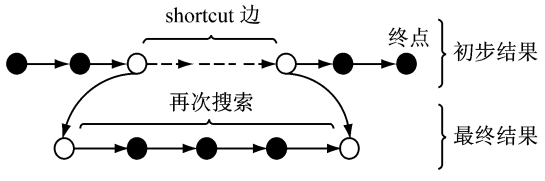

图 5 HH 算法分二次计算完整路径图

Fig.5 Two phases of computation of a full shortest path of HH Algorithm

假设计算起点 s 和终点 t 间最短路方法为 GetPath(s, t),计算完整最短路的伪代码为

//Computer the shortest path $P=(s,\cdots,u,\cdots,t)$ without shortcuts;

u is settled from both directions;

flag←true;

do{

while(flag) do{

flag←flase;

v is parent node of u in P;

if((v, u) is shortcut)then{

```
        flag←true;
        GetPath(v, u);
       }
     }
u←v;
}while(u≠s or u≠t)
```

伪代码中 GetPath(v, u)产生再次搜索时间 Δt,HH 算法总搜索时间为 $t+\Delta t$. 二次迭代搜索增加搜索时间 Δt,带来效率问题.

2.3　预处理数据存储问题

Contraction 算法分层压缩路网后,产生多层虚拟路网数据. HH 算法存储数据的方式是将算法所生成的多层虚拟路网合并,存储单层路网数据,减少了存储空间. 但是,多层压缩路网合并后,增加了路网边集元素个数,增加了搜索空间. 搜索算法部分伪代码为

```
while(T⃗ ∪ T⃖ ≠∅)do{
    …;
    foreach e=(u,v)∈E⃗ do {
       …;
       }
}
```

伪代码中集合 $\vec{E}$包含了集合 E'_l $(0\leqslant l<L)$的所有元素,$|\vec{E}|=\sum_{l=0}^{L-1}|E'_l|$,算法总搜索空间为每层压缩路网搜索空间之和. 因此,分层路网搜索空间大于单层压缩路网搜索空间,带来效率问题.

3　HH 算法的改进

3.1　针对压缩成环的改进

为了避免对同一点的多次搜索,采取无环压缩策略,防止环路的产生. 当压缩过程出现环路时,停止压缩,放弃该点,且不产生 shortcut 边,图 6 所示.

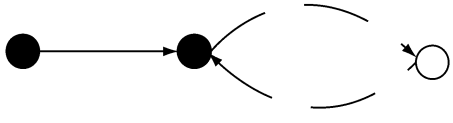

图 6　无环压缩路网

Fig.6　Contracted network without cycle road

预处理算法中部分伪代码改为

```
foreach v∈V_l{
  if v meets to the bypassability criterion then {
     set v to be bypassed node;
     delete(E_l,outJoint(v)),
     delete(E_l,inJoint(v));
     if(shortcuts is a circle)then{
        delete(V_l,v)
      }
      else{
        insert(E_l,shortcuts);
       }
   }
   …;
}
```

伪代码中增加了成环判断条件,即解决成环问题,又减少了搜索空间.

3.2　针对计算完整最短路时需重新搜索 shortcut 段的改进

为了避免产生搜索 shortcut 段的时间 Δt,采取局部最短路存储策略,以空间换时间. 储存 shortcut 边所包含的边,计算完整最短路时,直接将储存的边替代 shortcut 边即可,不必对 shortcut 边再搜索一次,节约搜索时间,提高算法的效率,如图 7 所示.

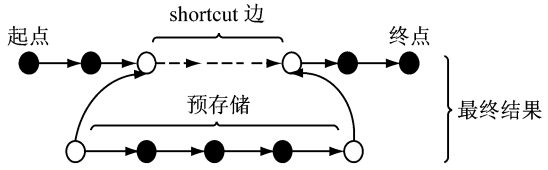

图 7　改进 HH 算法一次性计算完整路径图

Fig.7　A complete computation of full shortest path with the improved HH Algorithm

伪代码为

```
//Computer the shortest path P=(s,…,u,…,t)
without shortcuts;
u is settled from both directions
do{
    v is parent node of u in P;
    if((v, u) is shortcut )then{
        (v, u) ←(v,…, u);
    }
    u←v;
}while(u≠s or u≠t)
```

3.3　针对预处理阶段数据存储的改进

采取分层存储策略,不将所有层路网数据合并,而是采用多张表对路网数据分层存储,以空间换时间. 每层节点的出边与入边只存储当层的边,core 点的出边与入边只存储其它 core 点连接的边,bypass

点的出边与入边存储与它连接的所边. 压缩路网边集元素个数 $|E_l'|=\sum_{l=0}^{L-1}|E_l'|$，缩小搜索空间，提高效率.

3.4 搜索阶段的改进

文献[3]研究表明，A * 算法的效率要高于Dijkstra算法，结合改进后的 Contraction 算法，用A * 算法进行搜索. 算法伪代码如下：

输入：起点 s 和终点 t.

输出：最短路的距离 d.

$d=\infty$;

Insert($\overrightarrow{T}$,s,(0,0,$\overrightarrow{r_0}(s)$));Insert($\overleftarrow{T}$,t,(0,0,$\overleftarrow{r_0}(t)$));

While($\overrightarrow{T}\cup\overleftarrow{T}\neq\Phi$)do{

select direction↔ such that $\overrightarrow{T}\neq\varnothing$;

u=deleteMin($f(x)$), $x\in\overrightarrow{T}$; 其中

$f(x)=\delta(x)+w(x,y)$

if u has been settled from both directions

$d=\min(d,\overrightarrow{\delta}(u)+\overleftarrow{\delta}(u))$;

if gap(u)$\neq\infty$ then gap′=gap(u) ,

else gap′=$\overleftrightarrow{r}_{l(u)}(u)$;

foreach $e=(u,v)\in\overrightarrow{E}(u)$ do {

for ($l=l(u)$, gap=gap′;$w(e)$>gap; l++,

gap=$\overleftrightarrow{r}_{l(u)}(u)$; //go “upwards”

if $l(e)$<l then continue; //Restriction1

if $u\in V_l'\wedge v\in B_l$ then continue; //Restriction 2

$k=(\delta(u)+w(e),l,\text{gap}-w(e))$;

if v has been reached then decreaseKey($\overleftrightarrow{T}$,v,k) by $f(v)$; else insert($\overleftrightarrow{T}$,v,k);

}

}

return d;

其中第5行和第13行为算法的主要改进部分. $f(x)=\delta(x)+w(x,y)$为点 x 的估价函数，$\delta(x)$为当前离起点的距离，$w(x,y)$为点 x 与终点的欧几里德距离.

4 算法效率测试分析

对广东省路网(节点数约37万个，路段数约81万条)分别用 HH 算法(Contraction+Dijkstra)、改进后的 HH 算法(改进的 Contration + A *)、Contraction 算法与 A * 算法的组合算法(Contr+A *)和改进后的 Contration 算法与 Dijkstra 算法的组合算法(改进 Contr+Dijks)以及 A * 算法进行了测试.

算法效率评价的依据为算法的合理性、算法的时间效率和空间效率. 测试方案为在广东省路网上按路径长度随机抽取 600 对点进行测试，将这 600 对点按 0～50，51～100，101～150，151～200，…，501～550，550 km 以上分成 12 组. 测试结果如图 8、表 1、图 9 及表 2 所示.

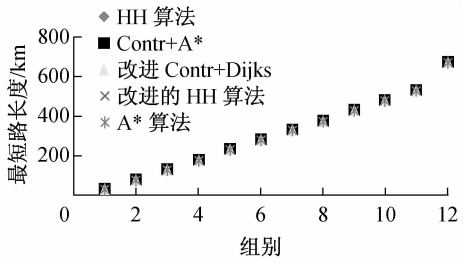

图 8 搜索路径长度对比

Fig.8 Comparison of search distances

表 1 搜索时间对比

Tab.1 Comparison of search time (unit: s)

取值范围	搜索时间/s				
	改进的 HH 算法	Contr+A *	改进 Contr+Dijks	HH 算法	A * 算法
最小值	0.009	0.013	0.039	0.068	0.101
平均值	0.126	0.147	0.425	0.634	4.453
最大值	0.297	0.348	1.027	1.439	11.263

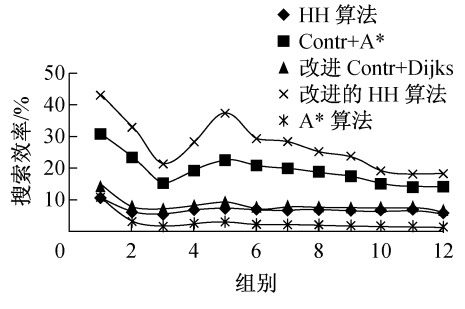

图 9 搜索空间对比

Fig.9 The comparison of search spaces

(1)算法合理性上，从图 8 知，5 种算法的长度较为接近，说明算法搜索结果的路径大致相同.

(2)算法时间效率上，通过表 2 可知，Contr+A * 算法和改进 Contr+Dijks 算法分别对 HH 算法有所改进，但改进的 HH 算法(改进 Contr+A *)的时间效率最高，约是 HH 的 5 倍；且根据最值与平均值的差距，可知改进 HH 算法效率较稳定. 另外，由表 2 知，在 0～50 km 范围内，5 种算法效率均较

高，且相差不大，但随着距离的增大，5 种算法效率存在较大的差距，如在 301～350 km 范围内，改进后的 HH 算法效率为 0.0980 s，HH 算法的为 0.5401 s，A＊算法的为 3.2433 s. 表明改进后的 HH 算法在起点与终点相距较大时，仍然能保持较高的效率；这也表明了改进的 HH 算法在大规模路网中使用的优势性.

表 2　按起、终点直线距离分类搜索时间对比

Tab. 2　Comparison of search time by straight-line distance of start node and end node(unit: s)

长度/km	搜索时间/s				
	改进的 HH 算法	HH 算法	Contr＋A＊	改进 Contr＋Dijks	A＊算法
0～50	0.008 6	0.068 3	0.013 2	0.039 5	0.101 3
51～100	0.033 6	0.189 2	0.042 1	0.156 7	0.710 3
101～150	0.110 8	0.447 2	0.125 1	0.324 3	3.389 2
151～200	0.098 0	0.454 8	0.114 2	0.316 3	3.233 6
201～250	0.112 7	0.439 5	0.129 5	0.305 9	3.877 1
251～300	0.104 3	0.514 9	0.124 2	0.348 4	3.875 6
301～350	0.098 0	0.540 1	0.117 1	0.330 7	3.243 3
351～400	0.100 8	0.583 0	0.120 1	0.366 0	3.406 7
401～450	0.135 3	0.779 7	0.159 9	0.543 2	4.736 6
451～500	0.189 3	1.024 5	0.216 7	0.597 4	7.355 6
501～550	0.219 5	1.128 0	0.251 1	0.752 3	8.244 3
＞550	0.297 1	1.438 7	0.347 6	1.027 4	11.264 0

(3)算法空间效率上，搜索空间效率是用最短路上点个数与搜索点个数之比来衡量的，比率越大，效率越高；由图 9 可知，改进的 HH 算法的空间效率最高，约是 HH 的 4 倍.

综上，无论在时间效率上还是空间效率上，改进的 HH 算法都是最优的.

5　结语

本文针对改进大规模路网路径规划效率问题，运用无环压缩策略、分层存储策略及局部最短路存储策略，对 HH 算法进行了改进. 改进后的 HH 算法不仅减少了搜索空间，也降低了搜索量. 实验结果表明，改进后的 HH 算法效率大大提高.

参考文献：

[1] Dijkstra E W. A note on two problems in conexiom with graphs [J]. Numerische Mathematik, 1959: 269.

[2] Dantzig G B. Linear programming and extensions [M]. Princeton: Princeton University Press, 1962.

[3] Hart P E, Nilsson N J, Raphael B. A formal basis for the heuristic determination of minimum cost paths [J]. IEEE Transactions on System Science and Cybernetics, 1968, 4(2): 100.

[4] Goldberg A V, Harrelson C. Computing the shortest path: A＊ meets graph theory[R]. [s.l.]: Microsoft Research, 2004.

[5] Goldberg A V, Harrelson C. Computing the shortest path: A＊ meets graph theory [C]//16th ACM- SIAM Symposium on Discrete Algorithms, Vancouver: ACM and SIAM, 2005: 156-165.

[6] Goldberg A V, Werneck R F. Computing point-to-point shortest paths from external memory [C]//Workshop on Algorithm Engineering and Experiments (ALENEX), British Columbia: SIAM, 2005: 26-40.

[7] Lipton R J, Tarjan R E. A separator theorem for planar graphs [J]. SIAM Journal on Applied Mathematics, 1979, 36(2): 177.

[8] Schulz F, Wagner D, Zaroliagis C D. Using multi-level graphs for timetable information [C]//Workshop on Algorithm Engineering and Experiments (ALENEX), San Francisico: Springer, 2002: 43-59.

[9] Schultes D. Fast and exact shortest path queries using highway hierarchies[D]. [S.l.] Universitat des Saarlandes, 2005.

[10] Muller K. Design and implementation of an efficient hierarchical speed-up technique for computation of exact shortest paths in graphs[D]. [S.l.]: Universitat Karlsruhe (TH), 2006.

[11] 刘灿齐. 车流在交叉口分流向延误的最短路径及算法[J]. 同济大学学报：自然科学版，2002，30(1)：52.
LIU Canqi. Shortest path including delay of each flow at Intersection and its algorithm[J]. Journal of Tongji University: Natural Science, 2002, 30(1): 52.

[12] 徐天东，孙立军，郝媛. 不同交通信息下网络交通动态路径选择行为[J]. 同济大学学报：自然科学版，2009，37(8)：1029.
XU Tiandong, SUN Lijun, HAO Yuan. Influence of different traffic information on drivers' dynamic route choice behavior in urban road network level[J]. Journal of Tongji University: Natural Science, 2009, 37(8): 1029.

[13] Holzer M, Schulz F, Willhalm T. Combining speed-up techniques for shortest-path computations [C]//3rd International Workshop on Experimental and Efficient Algorithms (WEA), Angrados Reis: Springer, 2004: 269-284.

[14] Delling D, Sanders P, Schultes D, et al. Highway hierarchies star[C]//9th DIMACS Implementatio Challenge, Rhode Island: American Mathematical Society, 2006: 568-579.