

文章编号:1001-9081(2010)11-2880-04

## 切换到高一层路网最近四个点的最短路算法

滕 聪

(山东经济学院 统计与数学学院, 济南 250014)

(cong teng\_82@sohu.com)

**摘 要:**针对基于大规模图的最短路问题求解速度慢的问题,提出了一个基于路网等级的求最短路的快速近似算法。该算法首先求出高一层路网到起点的4个最近点和到终点的4个最近点及最短路径,由高一层路网形成的子图 $T$ 再加上这8个最短路径形成图 $T'$ ,在 $T'$ 上求起点到终点的最短路。这种设计使得该算法适合在超大规模图上求解,理论上证明了精度可控,同时预处理数据也是可行的,从而使两点间最短路的求解速度大大提高。在纽约公路网上的测试结果说明了该算法的有效性和合理性。

**关键词:**最短路问题;Dijkstra算法;大规模计算;路网等级;时间复杂度

**中图分类号:** TP301.6; TP18 **文献标志码:** A

## Fast computation for point-to-point shortest path based on four closest nodes in higher level road network

TENG Cong

(School of Statistics and Mathematics, Shandong Economic University, Jinan Shandong 250014, China)

**Abstract:** The point-to-point shortest path computation is one of the hot research topics today. One straight forward application is to find the optimal driving directions. To solve the difficulties in shortest path computation for large scale graph, an efficient approximation algorithm was proposed based on road network hierarchies. Four closest nodes in higher level road network to starting node and four closest nodes to ending node were computed first along with 8 corresponding shortest paths. For subgraph  $T$  which consists of only higher level roads, 8 edges corresponding to the previously computed 8 shortest paths were then added to  $T$  and results in a graph  $T'$ . In graph  $T'$ , search for the shortest path from starting node to ending node, which completed the task. This design demonstrates that the proposed algorithm is suitable to solve large scale problems. An error bound is provided for approximation shortest path. It is also possible to preprocess the data first. In real application, the computational results are quite competitive, which shows that the proposed algorithm is effective.

**Key words:** shortest path problem; Dijkstra algorithm; large scale computation; road network hierarchy; time complexity

### 0 引言

最短路问题,即求一个图的两个点之间的最短路或一个点到其他所有点的最短路。经典的最短路算法由Dijkstra<sup>[1]</sup>给出,其时间复杂度为 $O(m + n \log n)$ <sup>[2]</sup>。很多实际问题可归结为最短路问题,它的一个直接应用是对公路网求两点间的最佳行车路线,也可应用在GPS装置中。在美国人们驾车出门前,一般都习惯先上网搜索最短驾车路线,在<http://www.yahoo.com>或<http://www.google.com>上搜索Maps、Driving directions,输入起点和终点的详细地址后,计算机在2s内会输出详细的行车路线,我国目前还没有这个应用。对大规模的公路网,求两点间的最短路主要用于公众查询,所以计算时间短是首要目的,可以允许有误差。本文提出了一个近似快速算法求公路网中的任两点间的最短路。

最短路问题是近年来国际上的研究热点之一,2006年被美国著名的离散数学与计算机科学研究中心(Center for Discrete Mathematics and Theoretical Computer Science, DIMACS)列为重点问题:9th DIMACS Implementation Challenge: Shortest Paths,其中还给出了美国纽约公路网、加州公路网,及美国全国公路网等所对应的图文件(见<http://www.dis.uniroma1.it/~challenge9>)以方便测试算法。近年来涌现出的求两点间的最短路的精确算法主要分为3类:1) 边带标号的最短路算法(Fast point-to-point shortest path computations with arc-flags)<sup>[3-5]</sup>;2)  $A^*$ 搜索算法( $A^*$  search with landmarks or reach)<sup>[6-8]</sup>;3) 高速路分层算法(Highway hierarchies)<sup>[9-11]</sup>。这几类算法都是在Dijkstra算法的基础上改进的,都适合求大规模图的任两点间的精确最短路,所有这些算法都可以结合双向搜索以减少时间复杂度。求两点间的最短路的首要目的是在用户查询时求解要快,所以上述3类算法都用到了预处理数据,预处理数据即假设对同一个图经常需要多次重复地求任两点间的最短路,所以先求出任两点间的最短路,存储有用信息,以备在用户实际查询两点间的最短路时协助求解,以减少运算和查询时间。虽然预处理数据的时间可以相对较长,但最终存储的预处理信息不能占用太多空间,一般要求为输入图的规模的线性函数,且系数要小。

我国城市的大路一般形成蜘蛛网般的连通网络,而小路则密密麻麻地穿插其中,假设要求两点之间的最佳驾车路线,当然可用Dijkstra算法或经其改进的上述3类算法<sup>[3,6,9]</sup>求出精确的最短路,但一是求解时间较长,二是求出的最短路线中可能有很多小路,一般不是人们的出行选择,如果两点之间的

www.dis.uniroma1.it/~challenge9)以方便测试算法。近年来涌现出的求两点间的最短路的精确算法主要分为3类:1) 边带标号的最短路算法(Fast point-to-point shortest path computations with arc-flags)<sup>[3-5]</sup>;2)  $A^*$ 搜索算法( $A^*$  search with landmarks or reach)<sup>[6-8]</sup>;3) 高速路分层算法(Highway hierarchies)<sup>[9-11]</sup>。这几类算法都是在Dijkstra算法的基础上改进的,都适合求大规模图的任两点间的精确最短路,所有这些算法都可以结合双向搜索以减少时间复杂度。求两点间的最短路的首要目的是在用户查询时求解要快,所以上述3类算法都用到了预处理数据,预处理数据即假设对同一个图经常需要多次重复地求任两点间的最短路,所以先求出任两点间的最短路,存储有用信息,以备在用户实际查询两点间的最短路时协助求解,以减少运算和查询时间。虽然预处理数据的时间可以相对较长,但最终存储的预处理信息不能占用太多空间,一般要求为输入图的规模的线性函数,且系数要小。

收稿日期:2010-04-19;修回日期:2010-08-02。 基金项目:国家自然科学基金资助项目(61070230)。

作者简介:滕聪(1968-),女,山东济南人,副教授,博士,主要研究方向:计算机大型计算、运筹学、动态系统降维。

行车距离较长,从起点上了大路后,就可以一直走大路,到了终点附近再考虑小路,这种算法比精确最短路算法将大大减少所要搜索的边的数目,从而缩短求解时间并且适合解决超大规模问题。应用这个思路,本文提出了切换到高一层公路最近 4 个点来求两点间最短路的一个近似新算法(Four closest nodes hierarchy algorithm),简称 FourNodeHierarchy 算法。FourNodeHierarchy 算法用到了预处理,存储的预处理信息占用的空间是输入图的点数  $n$  的线性函数且系数较小所以符合要求,在实际求解两点间的最短路时,它转换到了由高一层路网构成的较小规模的图上求解,从而大大提高了求解速度及可处理的图的规模。在美国纽约公路网(<http://www.dis.uniroma1.it/~challenge9>)264 346 个点、733 846 条边的图上测试时,实验效果不错。对随机产生的 100 对点,分别用 FourNodeHierarchy 算法和精确最短路算法求出了相应的 100 个近似最短路(Approximate shortest Path, AP)和 100 个精确最短路 P,结果显示,虽然在距离上所有 100 对近似最短路 AP 大于等于相应的精确最短路 P,但在时间上其中 75 对点即 75% 的近似最短路 AP 的时间长度却严格小于相应的精确最短路 P 的时间长度,即走大路时虽然路程会远,但所花时间常常要短,这符合大多数人的出行选择,也说明了 FourNodeHierarchy 算法的有效性和合理性。在 3.1 节对 FourNodeHierarchy 算法所产生的近似最短路的误差进行了分析,并给出了一个误差上界。算法求一对点的最短路的运行时间为 2 s,连续求解 100 对点的最短路的运行时间一共是 9 s。具体的实验结果见 3.2 节。

关于求公路网中任两点间的最短路,国内在这方面有一些研究,就作者从中国知网上所能查到的文献看,都没有使用预处理数据,而国际上这方面的论文已普遍采用了预处理数据,所以本文提出的 FourNodeHierarchy 算法相比国内最短路算法在求解速度上显然占有优势。根据某种准则切换到高一层公路网的近似算法,国内文献见[12-14],国际文献也有各种近似最短路算法,本文提出了一种不同于这些算法的新算法,即切换到高一层路网最近 4 个点的 FourNodeHierarchy 算法。

## 1 FourNodeHierarchy 算法的描述

先考虑一座城市内的公路网,然后再推广到全国公路网。公路一般有一级公路、二级公路和高速路等,为描述算法方便起见,先设在一座城市内只分为大路和小路,对一般情况的公路网,其后稍加变形即可解决,在 3.1 节详细讨论。

假设图 1 是某座城市的模拟交通网络图  $G$ ,边代表公路,点代表交叉路口,粗边代表高一级别的大路,细边代表小路。如果求点  $A$  和  $C$  之间的最佳行车路线,因为距离比较近,所以不管大路小路直接找最短路即可;如果求点  $A$  和  $B$  之间的最佳行车路线,因为距离较远,则从  $A$  直奔大路,然后只走大路到  $B$  点附近,再走小路转到  $B$  点,这样的路线从距离上虽然不一定是最短路,但常常行驶时间会短一些,是大部分人所喜欢和选择的行车路线。这就是 FourNodeHierarchy 算法的思路。

首先预处理数据,对任意一点  $v$ ,求  $v$  到附近大路的 4 条最短路。参看图 2,图 2 是图 1 的局部放大图,对点  $A$ ,用 Dijkstra 算法求  $A$  到大路的 4 条最短路,设终点分别为  $a_1$ 、 $a_2$ 、 $a_3$ 、 $a_4$ 。通过预处理后,任一点  $v$  到附近大路的 4 个最近点及最

短路径都求了出来,标记并存储此信息;同时求出从周围大路到点  $v$  的 4 个最近点及最短路,标记并存储此信息。路网是有向图,很多情况下,点  $v$  到附近大路的 4 个最近点  $a_1$ 、 $a_2$ 、 $a_3$ 、 $a_4$  与附近大路到  $v$  的 4 个最近点  $\bar{a}_1$ 、 $\bar{a}_2$ 、 $\bar{a}_3$ 、 $\bar{a}_4$  并不一样。

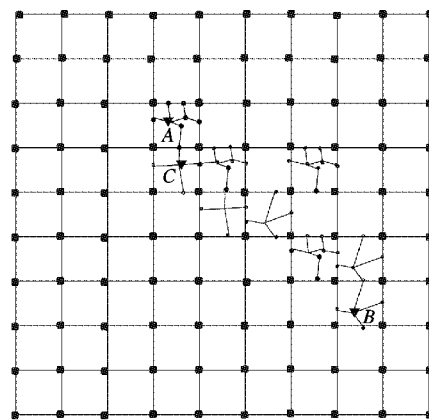


图 1 城市交通模拟路网

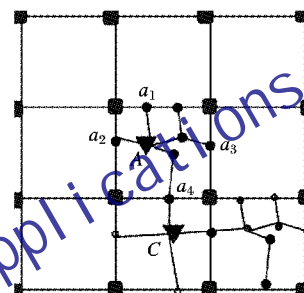


图 2 图 1 局部放大子图

当用户要求两点比如  $A$  和  $B$  之间的最佳行车路线时,计算机则只在城市交通网抽象出的子图  $G_{AB}$  上求解,图  $G_{AB}$  是这样构造的,首先只取大路为边构成的子图,即图 1 中的粗边所形成的网格,再加上点  $A$  及其到附近大路的 4 条最短路,把这 4 条最短路抽象为 4 条边  $Aa_1$ 、 $Aa_2$ 、 $Aa_3$ 、 $Aa_4$ ,同样再加上点  $B$  及附近大路到它的 4 条最短路所抽象成的 4 条边  $b_1B$ 、 $b_2B$ 、 $b_3B$ 、 $b_4B$ ,就构成了图  $G_{AB}$ ,见图 3。在  $G_{AB}$  上求  $A$  到  $B$  的精确最短路,算法的时间复杂度、求解过程中搜索的边比在图  $G$  上要少许多,从而求解速度大大提高,这就解决了大规模公路网求一对点的最优驾车路线问题,也完成了 FourNodeHierarchy 算法的描述。

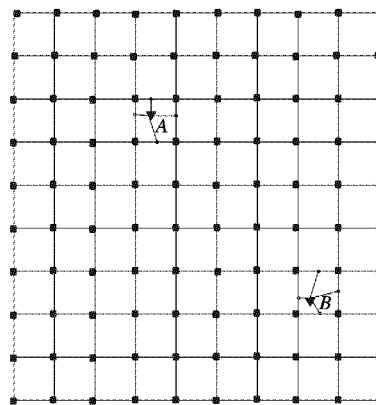


图 3 城市交通网络图导出的图  $G_{AB}$

## 2 FourNodeHierarchy 算法的实现

### 2.1 图的存储及预处理数据

图、图的边及点用下列结构来表示:

```

graph {
    edge * edgeset;
    node * nodeset;
    int endl;
    edge {
        int end2;
        int length;
    }
    node {
        edge * inc; /* Linklist of incident edges */
    }

```

即图由  $m$  条边组成的数组和  $n$  个点组成的数组构成,所以每个点与其在图的点数组中的下标一一对应,而每条边也与其在图的边数组中的下标一一对应。

在预处理数据时,图的表示中每个点的结构除了所关联的边构成的链表这个域外,再加一个域即标号(label),取0或1,取0表示此点不与粗边相关联,取1表示此点与粗边相关联。对每个点  $v$  运行 Dijkstra 算法,求  $v$  到附近大路的4个最近点、距离及路径。对输入图  $G$ ,改变每条有向边(弧)的方向得到图  $G'$ ,对点  $v$  在  $G'$  上调用 Dijkstra 算法,就求出了  $G$  中附近大路到点  $v$  的4个最近点、距离及路径。用4个长度为  $4n$  的整数数组  $Four\_closest\_node[4n]$ 、 $Four\_node\_dist[4n]$ 、 $Four\_node\_time[4n]$ 、 $Four\_node\_len[4n]$ , 和另一个长度为  $4n$  的数组  $Four\_node\_path[4n]$  来存储经预处理得到的从点  $v$  到附近大路的4个最近点的信息,其中  $n$  为图中点的数目。对每个点  $v$ ,设它在图的点集合中的下标为  $r$ ,它到大路的最近4个点的下标存储在  $Four\_closest\_node$  的第  $4r+1$  到第  $4r+4$  的元素里,相应最短路径的距离、时间和路径中点的数目分别存储在  $Four\_node\_dist$ 、 $Four\_node\_time$ 、 $Four\_node\_len$  的第  $4r+1$  到第  $4r+4$  的元素里,而相应的路径以路径中点的下标来指示,从而可用整数数组的形式存储在  $Four\_node\_path$  的第  $4r+1$  到第  $4r+4$  的元素里。因为每个点到它附近大路的4个最近点不会很远,所以每个最短路径中所含点数不会很多,一般10以内,从而数组  $Four\_node\_path$  所用的存储空间也不会很大。实际在预处理时,可用两个文本文件来存储预处理信息,一个文件存储所有点的4个最近点及相应4个路径的距离、时间和路径上的点数;另一个文本文件存储所有点的4个最短路径。在实际求解两点间的最短路时,只需读取文件并把信息存储在上述数组中。同样的方法可处理附近大路到  $v$  的4个最近点的信息。容易看出,存储的预处理信息所占的空间为点数  $n$  的线性函数,且系数不大。

算法1 FourNodeHierarchy 算法:预处理数据。

对任一点  $v$ ,调用 Dijkstra 算法求得  $v$  到大路的4个最近点和最短路径,将这4个点、路径距离、路径时间及路径所含的点数写入文件  $FourNodeFile\_from.txt$ ,将所对应的4个路径(以路径的点的下标组成的整数数组表示)写入文件  $FourNodePath\_from.txt$ ,同样用两个文件  $FourNodeFile\_to.txt$  和  $FourNodePath\_to.txt$  存储附近大路到点  $v$  的4条最短路,就完成了预处理。

## 2.2 任两点间的最短路

经过预处理数据后,现在可以求任两点间的最佳行车路线。

算法2 FourNodeHierarchy 算法:求任给点对  $u,v$  间的最佳行车路线。

1) 读取文件,存储预处理信息。

2) 构造图  $G$  和  $T$ ,其中  $T$  是由  $G$  中高一级公路组成的  $G$  的子图。

3) 对点  $u$  和  $v$ :

① 若  $u,v$  之间的欧氏距离足够近,则直接在  $G$  上调用 Dijkstra 算法求  $u,v$  之间的最短路  $P$ ;

② 否则,从预处理信息中抽取点  $u$  到附近大路的4条最短路并抽象成4条边加到图  $T$  中,同时从预处理信息中抽取附近大路到点  $v$  的4条最短路并抽象成4条边加到图  $T$  中得到图  $T'$ ,即  $T$  加上8条边得到了图  $T'$ 。

a) 在  $T'$  上调用 Dijkstra 算法求  $u,v$  之间的最短路  $P$ 。

b) 路径  $P$  形如  $u \rightarrow u' \rightarrow \dots \rightarrow v' \rightarrow v$ , 这里  $u'$  为  $u$  到附近大路的4个最近点  $u_1, u_2, u_3, u_4$  之一,  $v'$  为附近大路到点  $v$  的4个最近点  $v_1, v_2, v_3, v_4$  之一。从存储的预处理信息中抽取  $u \rightarrow u'$  和  $v' \rightarrow v$  所对应的实际路径  $P_u$  和  $P_v$ , 加到  $P$  中即得到了  $u,v$  之间的近似最短路  $u \rightarrow P_u \rightarrow u' \rightarrow \dots \rightarrow v' \rightarrow P_v \rightarrow v$ 。

## 2.3 算法1和算法2中的“调用 Dijkstra 算法”讨论

在上面的算法1和算法2中,有3次用到了“调用 Dijkstra 算法”,除此以外的其他步骤都比较简单明确并易于实现,这里只重点讨论 Dijkstra 算法。算法1中,调用 Dijkstra 算法求  $v$  到大路的4个最近点及距离,直接用经典的 Dijkstra 算法即可,因为  $v$  到附近大路的4个最近点一般距离较近, Dijkstra 算法几步即可结束,使用经处理的由 Dijkstra 算法改进的适合大规模图的最短路算法反而会增加时间复杂度。

对算法2中第①步的“调用 Dijkstra 算法”,因为  $u,v$  距离很近,直接用经典的 Dijkstra 算法几步即可结束。对第a)步中的“调用 Dijkstra 算法”,若大路构成的图  $T$  的规模不大,可以直接用经典的 Dijkstra 算法,若图  $T$  的规模较大,则必须采用经处理的用于应对大规模网络的精确最短路算法。因为求图的一对点之间的最短路应用广泛,引言中提到了近年来主要用于求大规模图的两点间的精确最短路的三类算法。第二类,  $A^*$  搜索算法中每一步都要判断一个函数不等式是否成立,从而增加了时间复杂度,而第三类,高速路分层算法用到的数据结构及写的函数较复杂,不太适合对内存有限制的系统如手机及一些GPS装置等。本文采用第一类算法,即给图的边加标号的方法,结合双向搜索来求导出图  $T'$  中的精确最短路,用的数据结构是优先级队列,并用二叉堆来实现。

## 3 FourNodeHierarchy 算法分析及计算结果

### 3.1 算法分析

FourNodeHierarchy 算法可以继续推广以适应一般情况。根据两点之间的欧氏距离,来决定 Dijkstra 算法是在以二级及以上公路为基础的图上进行,还是在以一级及以上公路为基础的图上进行,或只在高速公路上进行,所以在预处理数据中,需要存储下列信息:每个点  $v$  到二级及以上级别公路的4个最近点及距离,点  $v$  到一级及以上级别公路的4个最近点及距离,点  $v$  到高速公路的4个最近点及距离。二级公路如果占全部公路的  $1/5$ ,则只需在  $1/5$  大小的图上求最短路即可,如果说现在世界上最好的最短路精确算法可以有效处理500万条边的图,则 FourNodeHierarchy 算法可以同样的速度处理2500万条边的图,若两点距离足够远,则只需在一级及以上级别公路组成的图上进行,所处理的图更大。所以算法的有效性是显然的。

关于误差,参看图3,若求  $A$  到  $B$  的最短路,按 FourNodeHierarchy 算法,在抽象出的图  $G_{AB}$  上应该是  $A$  到  $a_4$



或  $a_3$ , 然后沿大路走到  $b_1$  再到  $B$ , 设其路径为  $P'$ , 而在实际的路网中,  $A$  到  $B$  的精确最短路  $P$  最短就是沿斜线从  $A$  到  $B$ , 这时近似地,  $\text{dist}(P') \leq \sqrt{2} \text{dist}(P) \approx 1.414 \text{dist}(P)$ , 事实上对任给三角形, 设  $a, b$  为直边的长度,  $c$  为斜边的长度, 而  $\alpha$  为一个角, 则  $a = c \sin \alpha, b = c \cos \alpha$ , 且  $a^2 + b^2 = c^2, (a+b)^2 = c^2 + 2ab = c^2 + 2c^2 \sin \alpha \cos \alpha = c^2(1 + \sin(2\alpha)) \leq 2c^2$ , 从而  $a+b \leq c\sqrt{2}$ , 所以当两点的欧氏距离不是很近时, 由 FourNodeHierarchy 算法求出的最短路距离小于实际最短路的 1.42 倍, 而当两点的欧氏距离很近时, 它求的是精确最短路且显然误差为零。从而大部分情况下, 由 FourNodeHierarchy 算法求出的最短路小于实际最短路的 1.42 倍, 当然这也取决于算法 2 的 3) 中怎样判断两点的距离足够近。

### 3.2 计算结果

本文分别实现了精确最短路算法和本文的 FourNodeHierarchy 算法, 并在美国纽约公路网上进行了测试比较, 实验环境为 C 语言, Cygwin 下的 GCC 编译器, Intel 双核处理器, CPU 2.2 GHz, 内存 2 GB。纽约公路网有 264 346 个点和 733 846 条边, 测试文件有 3 个, 见 <http://www.dis.uniroma1.it/~challenge9>。

USA-road-d-NY.txt: 边的长度用距离来表示的图文件。

USA-road-t-NY.txt: 边的长度用行车时间来表示的图文件。

USA-road-d-NY-co.txt: 点的坐标文件。

显然可以根据每条边的距离与时间的比值(即速度)来决定大路和小路, 在测试中取的是速度  $\geq 0.5$  的边为大路, 大路构成的图  $T$  有 40 420 条边, 是图  $G$  的边数的 5.5%。对随机产生的 100 对点, 用 FourNodeHierarchy 算法求出了相应的最短路  $AP$ , 同时对原图  $G$  用精确最短路算法求出了 100 条相应的最短路  $P$ , 显然  $d_{AP} \geq d_P$ , 计算结果见图 4。从图 4 可看出, 100 对点所对应的纵坐标都大于等于 1, 但计算结果同时显示 100 对点中只有 10 对点有  $t_{AP} > t_P$  (见图 5), 从图 5 可看出大部分点对所对应的纵坐标小于 1。这说明了 FourNodeHierarchy 算法的有效性和合理性, 即虽然求出的近似最短路  $AP$  的距离长度比精确最短路  $P$  长, 但 90% 的近似最短路  $AP$  的时间长度不大于精确最短路  $P$ , 其中 75% 的近似最短路  $AP$  的时间长度严格小于精确最短路  $P$ 。实现时, 判断两点是否足够近的准则, 用的是物理距离, 即由两点的坐标来计算欧氏距离。

关于 FourNodeHierarchy 算法的运行时间, 结果显示求一对点之间的最短路为 2 s, 而连续求解 100 对点的最短路的运行时间为 9 s。我们同时注意到, 虽然求一对点之间的最短路的运行时间为 2 s, 但 1.5 s 以上为读取并构建图  $G$  和  $T$ , 小于 0.5 s 为求两点间的最短路, 所以当将起点和终点改为在运行程序时在命令行上输入(即 Run-time command), 即在运行程序时, 读取并构建了图之后, 命令行上提示: 请输入起始点及终止点, 用户则输入 2 个点, 计算结果显示, 命令行上 0.5 s 内就给出两点间的最短路径及距离、时间, 这种方式与计算机查询系统是一致的, 因为人们在网上查询两点间的最短路时, 不同的用户输入不同的起点和终点, 而图  $G, T$  等在用户输入点对前就已建好了, 所以在用户查询纽约任两点间的最短路时, 我们的实现会在 0.5 s 内给出结果。从纽约公路网的计算结果可以推知, 对全美公路网中任两点  $u, v$ , FourNodeHierarchy 算法也可在 2 s 内输出最优行车路线。事实上, 若  $u, v$  在不同

的城市, 则只需在高速路网上求  $u, v$  的最近 4 个点及  $u, v$  间的最短路, 图的规模比一座城市内由二级公路构成的图大不了多少。

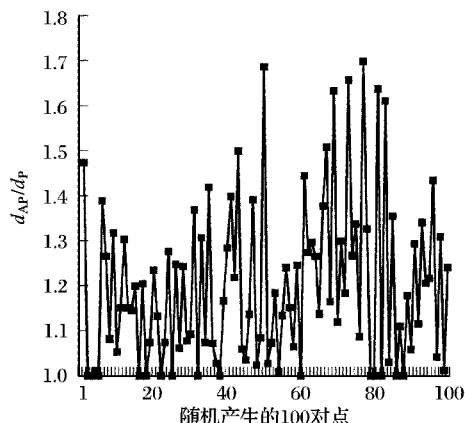


图 4 最短路距离的比较

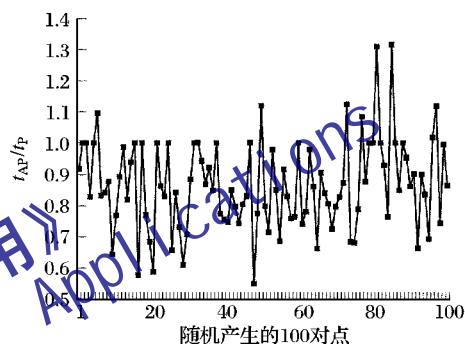


图 5 最短路所费时间的比较

FourNodeHierarchy 算法中取了 4 个最近点, 在实际计算中, 若误差结果不满意, 可以改为求 6 个最近点或 8 个最近点等。在实际路网中, 对任一点  $v$ , 大部分情况下有 4 条大路包围着它(见图 1), 直观上看取 4 个最近点相对更合适。我们分别实现了 2 个最近点、6 个最近点和 8 个最近点的情况, 并在纽约公路网上进行了测试, 实验结果显示 2 个最近点的情况舍近求远出现的机会相对较多, 而 6 个或 8 个最近点的情况比 4 个最近点的情况在最短路的距离及时间上改进不大。但求解时间几乎都没什么变化, 事实上在图  $G$  及大路构成的图  $T$  构建好之后, 取 4 个最近点给  $T$  加 8 条边与取 8 个最近点给  $T$  加 16 条边所构成的图的规模相差无几, 但存储的预处理信息量将加倍, 当输入图是超大规模时, 这会影响运行时间。所以在实际应用时, 需要看输入路网的状况来决定取几个最近点, 对一般的城市路网, 4 个点较好。

## 4 结语

算法的设计及实验结果均显示了本文提出的 FourNodeHierarchy 最短路算法是有效和合理的, 并适合大规模公路网。程序在纽约公路网上的运行时间也说明可以建立网络查询系统, 任何人只要上网, 输入中国国内的任意起始点及终止点, 计算机会在 2 s 内给出最佳行车路线及驾驶时间, 从而对这项国外常用的日常查询服务在我国也得以实现。这个算法的数据结构较简单, 且是转换到较小规模的图上求解问题, 这使得它适宜应用到内存小的设备上(如手机、GPS 装置, 以及街头的查询电路板等), 从而可直接应用到实际中以便人们的生活。

(下转第 3001 页)

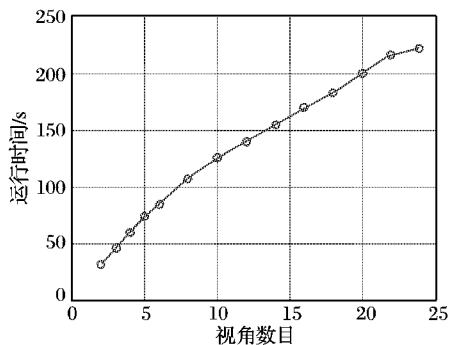


图 9 本文可见外壳算法性能

## 5 结语

理论分析和实验结果表明,本文提出的快速三维模型重建算法和视角数目呈线性关系。对于以增加视角数目来得到更精确的三维模型的可见外壳生成方法而言,本文算法能比文献[4-6]更快速地重建三维模型。另外,本文算法可以通过增加平行三维平面的个数获得更密集的三维模型重建效果,对于密集高精度三维模型的重建提供了很好的基础。

本文提出的方法易于硬件移植,实现三维模型重建实时系统。借助对场景的实时重建,可以方便地实现目标跟踪和动作识别等。近年来,较多的二维识别问题正逐渐向三维扩展,如三维人脸识别、基于三维重建的故障识别等,可见三维模型重建有非常好的应用前景。

### 参考文献:

- [1] 章毓晋. 图像工程——图像理解: 下册[M]. 2版. 北京: 清华大学出版社, 2007: 60-72.
- [2] LAURENTINI A. The visual hull concept for silhouette-based image understanding [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1994, 16(2): 150-162.
- [3] MATUSIK W, BUEHLER C, RASKAR R, et al. Image-based visual hulls [EB/OL]. [2009-12-12]. <http://people.csail.mit.edu/wojciech/IBVH/ibvh.pdf>

- [4] FRANCO J-S, BOYER E. Exact polyhedral visual hulls [C]// Proceedings of the Fourteenth British Machine Vision Conference. London: BMVA Press, 2003: 329-338.
- [5] FRANCO J-S. Efficient polyhedral modeling from silhouettes [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2009, 31(3): 414-427.
- [6] LAZEBNIK S, FURUKAWA Y, PONCE J. Projective visual hulls [J]. International Journal of Computer Vision, 2007, 74(2): 137-165.
- [7] KIECK H, HEIDRICH W, VOGELGSANG C. Shape from contours and multiple stereo — A hierarchical, mesh-based approach [C]// Proceedings of the First Canadian Conference on Computer and Robot Vision. Washington, DC: IEEE, 2004: 76-83.
- [8] VATTI B R. A generic solution to polygon clipping [J]. Communications of the ACM, 1992, 35(7): 56-63.
- [9] GREINER G, HORMANN K. Efficient clipping of arbitrary polygons [J]. ACM Transactions on Graphics, 1998, 17(2): 71-83.
- [10] MARTINEZ F, RUEDA A J, FEITO F R. A new algorithm for computing Boolean operations on polygons [J]. Computer & Geosciences, 2009, 35(6): 1177-1185.
- [11] MURTA A. General polygon clipper library [EB/OL]. [2009-12-25]. <http://www.cs.man.ac.uk/~tobczalan/software/>.
- [12] de BERG M, OTFRIED CHEONG, van KREVELD M, et al. Computational geometry: algorithms and applications [M]. Third edition. Berlin: Springer, 2008: 20-29.
- [13] KIECK H, KEDEM Z M, USELTON S P. Optimal surface reconstruction from planar contours [J]. Communications of the ACM, 1977, 20(2): 693-702.
- [14] MEYERS D, SKINNER S, SLOAN K. Surfaces from contours [J]. ACM Transactions on Graphics, 1992, 11(3): 228-258.
- [15] CONG G, PARVIN B. An algebraic solution to surface recovery from cross-sectional contours [J]. Graphical Models and Image Processing, 1999, 61(4): 222-243.
- [16] 纪凤欣, 欧宗瑛, 秦绪佳. 基于 Delaunay 三角剖分的层析图像离散数据表面重建算法 [J]. 工程图学学报, 2001, 22(2): 53-58.

(上接第 2883 页)

### 参考文献:

- [1] DIJKSTRA E. A note on two problems in connexion with graphs [J]. Numerische Mathematik, 1959, 1(1): 267-271.
- [2] FREDMAN M L, TARJAN R E. Fibonacci heaps and their uses in improved network optimization algorithms [J]. Journal of the Association for Computing Machinery, 1987, 34(3): 596-615.
- [3] KÖHLER E, MÖHRING R, SCHILLING H. Fast point-to-point shortest path computation with arc-flags [EB/OL]. [2009-12-12]. <http://www.math.tu-berlin.de/coga/people/schilling/pub/20061113-DIMACS.pdf>.
- [4] LAUTHER U. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background [EB/OL]. [2009-12-12]. <https://gor.uni-paderborn.de/Members/AC06/LAUTHER.PDF>
- [5] LAUTHER U. Slow preprocessing of graphs for extremely fast shortest path calculations [C]// Lecture at the Workshop on Computational Integer Programming. ZIB: [s. n.], 1997.
- [6] GOLDBERG A V, HARRELSON C. Computing the shortest path: A\* search meets graph theory [C]// Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). Vancouver: SIAM, 2005: 156-165.
- [7] GOLDBERG A V, WERNECK R F. Computing point-to-point shortest paths from external memory [C]// Proceedings of the 7th Work-

shop on Algorithm Engineering and Experiment (ALENEX). Vancouver: SIAM, 2005: 26-40.

- [8] GOLDBERG A V, KAPLAN H, WERNECK R F. Reach for A\*: Efficient point-to-point shortest path algorithms [C]// ALENEX: proceedings of the 8th Workshop on Algorithm Engineering and Experiment. Vancouver: SIAM, 2006.
- [9] SANDERS P, SCHULTES D. Highway hierarchies hasten exact shortest path queries [C]// Proceedings of the 13th Annual European Symposium. Berlin: Springer, 2005: 568-579.
- [10] SANDERS P, SCHULTERS D. Engineering highway hierarchies [C]// Proceedings of the 14th Annual European Symposium on Algorithms. Berlin: Springer, 2006: 804-816.
- [11] KNOPP S, SANDERS P, SCHULTES D, et al. Fast computation of distance tables using highway hierarchies [R]. Karlsruhe: University of Karlsruhe, Faculty of Informatics, 2006.
- [12] 李清泉, 郑年波, 徐敬海, 等. 一种基于道路网络层次拓扑结构的分层路径规划算法 [J]. 中国图象图形学报, 2007, 7(12): 1280-1285.
- [13] 李锴, 钟耳顺, 曾志明, 等. 基于网络分层拓扑结构的最优路径算法 [J]. 中国图象图形学报, 2006, 11(7): 1004-1009.
- [14] 高松, 陆峰. 一种基于路网等级启发式策略的路径搜索算法 [J]. 地球信息科学学报, 2009, 11(2): 151-156.