路网中基于地理位置和区域封闭性的最短路径的查询算法

顾明皓[1] 徐 明[1,2]
(上海海事大学信息工程学院 上海 201306)[1] (同济大学电子与信息工程学院 上海 201804)[2]

摘 要 针对大规模城市路网寻找最短路径的问题,提出了一种基于边的聚类树(ECT)和最小封闭格(MCL)的算法来达到路网中快速查询的目的。首先对给定的城市路网进行预处理,即利用封闭格的定义对路网进行划分;其次利用ECT树对划分出的MCL格进行存储;最后利用虚拟路径的思想(两点之间直线距离最短)并结合MCL格的性质和路网的平面性的特点,利用ECT树存储的优势,在查询中大大减少了无用节点的访问数量,降低了时间复杂度,从而达到了快速寻找最短路径的目的。理论分析和仿真实验表明,在大规模的城市路网中ECT树的存储空间相比PCPD算法减少了约45.56%,相比TNR算法减少了24.35%,其在存储方面略优于较为完备的SILC算法。MCL算法在查找过程中的搜索效率比SPB算法快15.6%。实验结果表明基于ECT存储的MCL算法在实际查询过程中能提高查询的效率。

关键词 MCL格,最短路径,虚拟路径,ECT树,预处理
中图法分类号 TP311.13 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.6.038

Shortest Path Searching Algorithm Based on Geographical Coordinates and Closed Attribute in Road Network

GU Ming-hao[1] XU Ming[1,2]
(College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China)[1]
(School of Electronics and Information Engineering, Tongji University, Shanghai 201804, China)[2]

Abstract Concerning the problem of finding the shortest path in the large scale city road network, this paper proposed an algorithm based on the Edge Clustering Tree and Minimal Closed Lattice to achieve the goal of quick searching. Firstly, the city road network is proprocessed, i.e. using the definition of the MCL to classify the road network. Then ETC is used to storage the MCL. Eventually, depending on the idea of the virtual path (the shortest distance between two points is the straight line distance), combining the attribute of MCL planarity of the road network and taking advantage of the ECT will dramatically reduce the visits to dud nodes. So this kind of algorithm really reduces the time complexity and achieves the purpose of quick searching. The theoretical analysis and the simulation experiment demonstrate that the storage space of ECT is 45.56% less than the PCDC algorithm and 24.35% less than TNR. In the aspect of storage, ECT is slightly better than SILC. Besides, the query efficiency of the MCL is 15.6% faster than that of the SPB. The results of the experiment suggest that the MCL algorithm based on ETC storage can improve the query efficiency.

Keywords Minimal closed lattice(MCL), Shortest path, Virtual path, Edge clustering tree(ECT), Preprocessing

1 引言

计算空间网络中两个地理位置之间的最短路径是一个十分经典且有着重要意义的问题。现阶段随着在线地图服务需求的日益增长,在空间路网中查询最短路径的需求量也越来越大,然而现阶段的路网相当复杂,我们在对路网做处理时不仅需要考虑到其规模,还需要考虑到用户的等待时间,一个需要长时间等待的算法是没有意义的,路网中Dijkstra算法[1]是最为经典的,它能够计算出路网中任意两点的最短路径。这个算法简单、准确,但是在大型路网即顶点数量达到百万级时,Dijkstra算法的时间复杂度为$O(n^2)$,其在解决最短路径问题时效率十分低下,甚至可以认为是不可行的,因为该算法要访问从终点一直到起始点的距离范围内所有的点,而当两点之间的距离非常远时,Dijkstra算法几乎把图中所有的点全部遍历了一遍,路网的节点的个数直接决定了算法的复杂度。而当前在许多应用中都要求查询的高效性,因此需要提出更合理的算法来提升效率。

路网中两点之间的最短路径的查询效率在许多应用中特别重要,尤其是在在线地图服务和公路网络的查询应用中(如Google Map、百度地图、丁丁地图等),当大量用户向服务器发

到稿日期:2015-04-19 返修日期:2015-08-11 本文受国家自然科学基金项目(61202370),上海市教委科研创新项目(14YZ110),中国博士后科学基金资助项目(2014M561512)资助。
顾明皓(1994—),男,硕士生,主要研究方向为道路网络、空间数据库,E-mail:929637996@qq.com;徐 明(1977—),男,博士,副教授,CCF会员,主要研究方向为道路网络、水声传感器网络、智能信息处理。

出查询请求时，如果查询算法的效率过低，那么应用的服务质量就会下降，用户便会失去耐心，因此采用高效的查询算法提高大规模城市路网的服务质量是非常有必要的。目前国内外在提高最短路径查询效率的算法上大致可分为4类。第一类算法基于公路的平面性质，如 Philip N 等人[3]提出的算法，其时间复杂度为 $O(n\log n)$。D. Papadias 等人[4]提出了一种时间复杂度和空间复杂度均为 $O(n\log n)$ 的算法。这类算法通过路网的平面性，并将有用的信息适当存储起来，利用这些信息加速查询。第二类对路网进行预处理，利用 R 树索引或者是四叉树索引对路网进行存储，并结合 Dijkstra 算法来提升查询速率。如 INE、IRE[4]算法，Cho H J[5]等人在 INE、IRE 上做出了相应的改进。文献[2]通过对路网中所有的点进行划分，利用四叉树索引对路网进行存储，使得整个查询过程类似于对一片目标区域的不断划分，从而达到加速计算的目的。第三类算法，如文献[6-8]中所提算法，是根据路径的重要程度将路网中的节点按照重要性排序，对重要节点的最短路径进行计算并以较为合理的方式存储起来。但这样的算法往往只能返回最短路径的长度却没有查询的功能，且查询效率也不得而知，因此这些算法似乎也存在着一些局限性。第四类算法是路径一致的点对划分，这类算法简称 PCPD[16,17]。但是该类算法只是在路网节点较小的网络上进行测试，大规模路网的查询效率还不得而知。

过去的算法有的通过存储结构，有的依靠空间一致性来实现查询的加速，但似乎还没有算法涉及到基于空间地理位置的查询，基于地理位置的路由在传感器网络上已经取得了很好的研究成果，但在空间路网方面还没有人涉足。就查询最短路径本生而言，不论路网节点有多少，整个路网有多么复杂，都可以直接从地图上直观地得出起点 s 到终点 v 的直线路径，这条路径不论存在与否都是起点到终点的最短路径，而事实上在复杂的路网中这条路径是不存在的，但是可以根据这条路径在整个路网中寻找次优最短路径。本文将根据空间对象的地理位置并通过 R 树索引和四叉树索引来达到快速查询的目的。

路网中的网状结构就几何形状而言，也为我们在计算最短路径时提供了方便，本文研究的难点在于将路网 G 以什么样的方式进行存储，如何根据虚拟路径(起点 s 到终点 v 的直线路径)规划和选择出最短路径。本文提出一种预处理即把路网划分成众多格的形式并用 R 树建立索引。在利用 R 树上的索引信息的同时运用递归的求解方法在每两个邻节点之间求出最短路径，通过求出每一段的最小路径，然后把每一段的路径进行求和，即 $w=\sum_{i=0}^{k} d(v_i, v_{i+1})$，最终快速求解出最短路径的轨迹和长度，从而提升查询效率。

2 封闭格(Closed Lattice,CL)及其性质

2.1 封闭格(CL)的定义

已知图 $G=\{V(G),E(G),\varphi_G\}$。其中 $V(G)=\{v_I \mid i\geqslant 1, i\leqslant n\}$ 是图的顶点，$E(G)=\{e_i \mid v_i\in V(G), d(v_j,v_k), j\neq k\}$ 是图的边。方向向量可在对顶点遍历的过程中建立，具体通过各顶点的经纬度坐标来确定。$\varphi_G(e_i)=\{(v_i,v_j) \mid i\neq j\}$ 即某一条边的两个顶点的集合。$d(v_i,v_k)$ 表示从起点 v_j 到 v_k 的一条路径。$\vartheta(A)$ 表示非空单个元素的总个数。设 $v(G)\in V(G)$，即 $v(G)$ 是所有顶点集合的子集，$e(G)$ 是无向图 G 中所有边的集合的子集。

定义 1(封闭格 $GL(G)$) 它是由顶点组和边组组成的一个集合，且满足：$\forall e_i\in e(G)\ \exists e_j\in e(G), (\varphi_G(e_i)\cap\varphi_G(e_j)\neq\varnothing)\wedge(\vartheta(e(G))=\vartheta(\varphi_G(\bigcup_{k=1}^{\vartheta(e(G))} e_k)))$。那么由顶点集合 $v(G)$ 和边集合所组成的集合被称为封闭格，如图1所示。

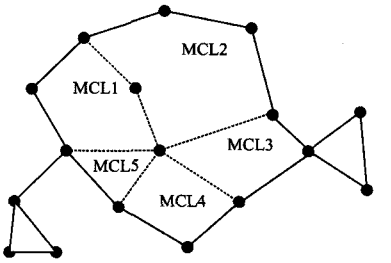

图1 封闭格的例子

图1是由 $L=\{l_1,l_2,l_3,l_4,l_5\}$ 组成的格。它们彼此之间因为有边相邻所以组成一个大的区域，图中已经在不同的格区域内用字母标记，相邻两个格之间用虚线标记它们是两个格的公共边，封闭格有以下4个性质。

性质 1 每一个独立的封闭格 CL 中每一个顶点有且只有两条边穿过这个顶点。即 $\forall l_i\in L\ \forall v_k\in l_i, \exists E=\{e_1,e_2\}$。

性质 2 对于固定的路网，其 L 的集合是固定不变的。

性质 3 任意两个相邻的最小格之间至少存在一条公用的边。即一条公用的边，必定对应着两个最小的相邻的格。

性质 4 $\forall l\in L, v(l)\geqslant 3$，即对于任意一个格，格中的顶点个数至少大于或者等于3。

证明1：(性质3证明)如果一条公用的边存在两个以上的相邻的最小格，那么必定满足最小格 MCL 的性质1，显然在3个格共用一条边时必然满足，$\exists\vec{e_l}(v_i\rightarrow v_j)\in l_i\ \exists\vec{e_j}(v_j\rightarrow v_m), \vec{e_k}(v_j\rightarrow v_n)\in l_k, (\vec{e_i}\times\vec{e_j})\cdot(\vec{e_i}\times\vec{e_k})<0$，那么与最小格的定义相矛盾，于是得证。

证明2：(性质4证明)如果 l_1 是 MCL 格，且格中的顶点个数小于3，那么根据定义1，此时 l_1 在中有 $\vartheta(e(G))\neq\vartheta(\varphi_G(\bigcup_{k=1}^{\vartheta(e(G))} e_k))$ 不满足定义1的条件，这与 l_1 是格相矛盾，于是得证。

定义 2 $B=\{v_i \mid \forall l_i\in CL(v_i)\ \exists l_k\in CL(v_i), e(l_i)\cap e(l_k)=\varnothing\}$。其中 $CL(v_i)$ 表示与顶点 v_i 对应的所有格的集合，$e(l_i)$ 表示在格 l_i 中所有边的集合，边界点的建立是为了能更高效地对最短距离进行查询，特别是对一些不构成格的区域可以直接通过在边界点事先已经存储的最短路径的信息直接向下一个格区域移动，这样就能避开对于非格区域的搜寻进而达到快速查询的目的。图2所示为路网中边界点的选取。图中用空心实线圆圈出的点就是边界点。边界点可以快速地跨过非格区域，进而快速地在格中查询。

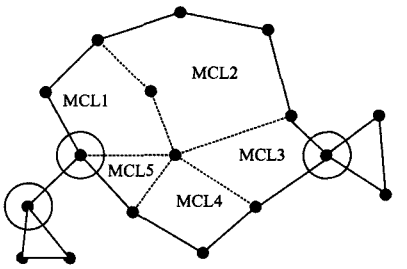

图2 边界点的选取

定义 3 在一张路网图中，对于最小格而言，在格中没有可以再分的格即为最小封闭格(MCL)。

性质 5 如果在一个格中还存在一个小的格，那么小格的某一条边必定夹在这个格的两条边之间。

为了确定最小格，需要引入边的方向向量集合，$\vec{e}=\{\vec{e_i}\mid i\geqslant 0, i\leqslant \vartheta(E)\}$和特定的算法来确定最小格。MCL 构建的基本思想：从图 G 中某一条边 $\varphi_G(e)$出发，查看 $\varphi_G(e)$，并将 G 中相应的顶点所存位置指向一个新建的节点，这个节点需要包含整个顶点组的类别，以及在这个类别中的所有顶点。搜寻边时需要利用广度优先法对边进行遍历，且在搜寻的过程中需要用到边的向量，计算当前向量$\vec{e_0}$与待选边向量$\vec{e_1}$的夹角，选择 $\max(\vec{e_0}\wedge\vec{e_0})$的目的是确定最小格，最小格的建立能够方便我们快速地对路径进行查询和搜索。图 3 展示了以 a 作为起点建立 MCL 格的过程，数字代表寻找边的先后顺序。

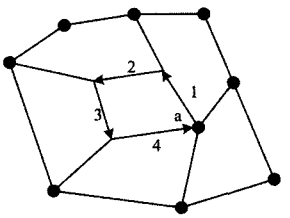

图 3 MCL 格选取的过程

至于寻找最小格的算法的正确性，下面将利用反证法给出相应的证明：已知 l_i 是最小格，假设格$\overrightarrow{e_{l_{\max}}}$不是最小格，那么根据最小格的性质 5 可知 $\exists e_k\in l_j, \exists(e_m,e_n)\in l_i(\varphi_G(e_m)\cap\varphi_G(e_k)\cap\varphi_G(e_n)\neq\emptyset)$，即某一个小格的某一条边必定夹在这个格的两条边之间。而在算法中满足式子：

$degree(\vec{e_i}\wedge\vec{e_j})=\max(degree(\vec{e_i}^{\vec{e_{l1}}}), degree(\vec{e_i}^{\vec{e_{l2}}}),\cdots, degree(\vec{e_i}\wedge\overrightarrow{e_{l_{\max}}}))$

$\overrightarrow{e_{l_{\max}}}$指与某一条边的一个顶点相接的边的最多的边数量，由于某一条边都满足上述的条件，因此可以推出在算法确定的格中不存在某一个子格的边夹在大格的两条边之间，这与已知条件相矛盾，由此算法的正确性得证。

3 路网的存储和相关算法

3.1 基于 R 树的 MCL 格树的建立

在空间路网中，用空间对象的 MBR 来近似表达空间对象，根据地域范围的 MBR 建立 R 树，可以直接对空间中占据一定范围的空间对象进行索引。R 树的建立是为了能在索引中更好地对空间中的数据进行查询，然而 R 树中的对象可以是任意的，只要满足一定的条件就可以在 R 树中存储。

本文在 R 树中存储的是 MCL 格和非格。具体来说就是把路网中的每一个 MCL 格用一个小的矩形框包起来形成 MBR，把那些有公共边的 MCL 格进行聚类，从而形成新的格，并用更大的 MBR 将每一类的格包括起来。而在包围的过程中会形成许多相交的区域，这些小的区域中往往只是两个相邻格之间的公共边的信息，这些信息需要重复地存储，而不是单单只放在一个区域中。在构建矩形框时还需要记录矩形框所包围的区域的范围。为了方便实际操作过程中的计算，在路网中每一个节点都有自己的经纬度坐标，假设结点 v 的经纬度坐标为(x,y)，那么在整个路网中就存在一个关于路网中所有顶点的坐标集合，可形式化表示为：$p=\{(x,y)\mid x(l\_bound, r\_bound), y(b_{bound}, t\_bound)\}$，如图 4 所示，可以清楚地了解怎样构建 MBR 和树的存储结构。根据矩形框的边界可以找出至少 4 个点在矩形框上，而这 4 个点的坐标就能唯一确定并划分一个区域，这样的划分将会在后文介绍的查询操作中起到非常重要的作用。此外划分过程中可能出现许多的相交区域，这些区域的存在也为查询提供了便利，相交区域的出现势必意味着两个区域之间存在着某种联系。

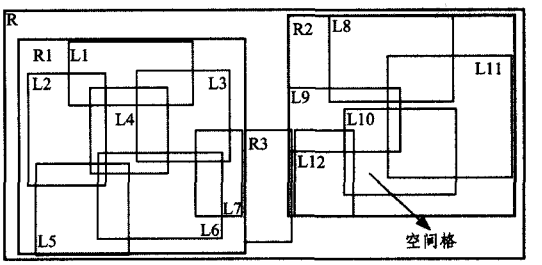

图 4 MCL 格的存储

上述已经阐明了对最小格 MCL 的存储，下面将对 R 树的上层做进一步的说明。R 树的上层依然存储封闭格 CL，但是这里的格不是最小的 MCL，而是很多个 MCL 结合在一起的大格。具体而言就是把两个相邻的格即把两个有公共边的格如果放在一起看作是一个大的格，即对于任意的格满足 $\forall l_i\,\forall l_j(e(l_i)\cap e(l_j)\neq\emptyset)$，那么它们就能构成一个大的格，把这些小格放到一个集合 BCL 中，最后得到的 n 个 BCL 集合中每一个集合都能构成一个大的格，可以把这个大的格用矩形框包起来并存储在 R 树中，同样也需要存储这些大格的区域范围。图 5 展示了 R 树存储的实例。

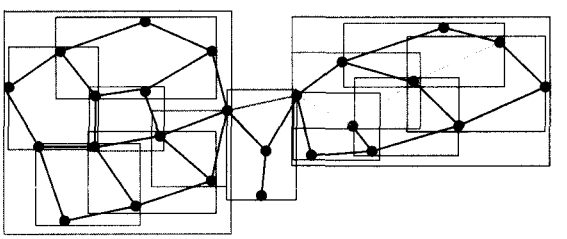

图 5 空间路网存储在 R 树上的一个实例

图 5 中每一个 MCL 格都被一个矩形框包了起来，这个矩形框中不仅存储了 MCL 格，同时还根据顶点信息唯一确定矩形框的位置信息。创建过程详见算法 1。

算法 1

```
1. Assemble_Lattice(L,L'[ ])
2. /* 在函数中输入 MCL 格的集合，它们用链表的形式存储以便操
   作，输出聚类后的大格放在线性表 L'[ ]中 */
3. L0=L
4. whilenot IsEmpty(L)
5. l1=getlattice(L)
6. ei++=[qi++]=e(l1)
7. /* 从格 l1 中取出每一条边存到 ei[ ]数组中，其中 qi 代表 MCL 格
   中边的数量初始状态下 i 为 1 */
8. for i=1 to qi
9. hash_save(T,ei)
10. /* 把 MCL 格存储到其在哈希表中对应的每一条边中去，即在哈
    希表中一些位置上存上当前格 */
11. end for
12. delete(l1,L)
```

```
13. / * 从格表中删除当前的格 * /
14. end while
15. l1 = getlattice(L0)
16. insert(Lt, e(l1))
17. while not IsEmpty(L0)
18. while not IsEmpty(Lt)
19. / * 当 Lt 中没有边时算法结束 * /
20. for i=1 to length(Lt)
21. add (Lt, e(T(Lt(i))))
22. / * 不断地在哈希表中寻找 MCL 格，并不重复地添加边到 Lt，如
    果发现有重复的边出现，就把这条边删除 * /
23. cluster (L'[k], T(Lt(i)))
24. delete (L0, T(Lt(i)))
25. end for
26. end while
27. k++
28. end while
```

在聚类格算法中就是要把有公共边的两个格放在一起作为一个大的格。为了降低算法的时间复杂度和空间复杂度，算法 1 将引入基于边的哈希表。步骤 4－14 执行循环语句直到把所有的 MCL 格全部存储到哈希表中为止。步骤 5－7 先从 MCL 格表中取出靠前的两个格放在 l_1 中，并把该格对应的边存储在数组 e 中，然后步骤 7－13 开始对在哈希表中 l_1 的每一条边对应的位置中添加 l_1。最后当整张表遍历完成后，哈希表中每个位置都能存储到对应的 MCL 格。步骤 15－28对 MCL 格进行聚类，步骤 20－25 中 L_t 存储的是当前聚类格的最外围的边，通过在哈希表中根据边的位置寻找 MCL 格，将 MCL 格存储到 L' 数组中去，而后进一步去重，添加边到 L_t 中，当所有边都重复时，此时 L_t 为空，第一次聚类成功。由于每一次聚类都会把已聚类的格从 L_0 中删去，当所有的格全部存储后，算法结束。

算法 1 的空间复杂度分析：算法 1 在执行过程中最关键的步骤就是不断地在由边信息构建的哈希表中寻找 MCL 格的位置，因此算法 1 的空间复杂度与该表的长度有关。假设路网中对的边数为 E，根据性质 3 可得每一条边最短对应的两个 MCL 格，因此算法的空间复杂度为 $\theta(2E)$。

算法 1 的时间复杂度分析：路网中 MCL 格的个数与算法的时间复杂度有关，对于哈希表的建立，根据算法 1 中的步骤 4－14 可知其时间复杂度为 $\theta(\sum_{i=1}^{\vartheta(L_0)} q_i) \leqslant O(p * q_{\max})$。其中 q_i 代表每一个 MCL 格的边的数量，p 代表总共 MCL 格的个数，根据实际路网的情况，q_i 值的分布往往是相对平均的，因此可以近似认为在存储时算法的复杂度为 $\theta(p * \bar{q})$，在某些路网中当$\bar{q}$的值较小时可近似认为时间复杂度为 $\theta(p)$。在实际聚类过程中需要访问每一个格的每一条边，算法的复杂度随着边数的重复量而增大。根据性质 3，如果每一条边都是公共边，那么算法的复杂度可近似地认为是 $O(2E)$。综上所述，算法 1 的总体的时间复杂度为 $\theta(p * \bar{q} + E)$。

此算法的预处理时间相比于 TNR[6-8] 的预处理时间在边数相对稀疏的路网上有较为明显的优势。

3.2 基于 R 树边索引的建立

基于 R 树的 MCL 格树的建立能够方便我们快速地对空间对象进行定位和索引，但是在实际寻找最短路径 $paht((v_1,v_2),(v_2,v_3),\cdots,(v_{n-1},v_n))$ 的过程中我们还需要引入基于四叉树的边存储，以往算法中四叉树的建立是针对整个路网图来建立，而文中提到的算法使用四叉树对每一个 MCL 格中的每一条边进行存储。如图 6 所示，把 MCL 格的每一条边存储在四叉树的一个分支当中，这样存储可以最大限度地快速对格中的边定位，此外在四叉树中还需要使每一个四叉树的小的分支额外附带该边所占区域的信息，即根据该边上的两个顶点坐标值来确定这条边所处的范围（如顶点 $v_i(x_1,y_1),(x_2,y_2)$，设 $x_1<x_2, y_1<y_2$，则可以确定如下区域 $pos(x,y)=\begin{cases} x\in(x_1,x_2) \\ y\in(y_1,y_2) \end{cases}$，把这样的信息存储到四叉树中）。如图 6 所示，图中用数字来对边所占的区域作出划分。

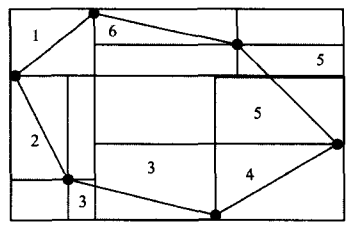

图 6　运用四叉树存储 MCL 格的边

在图 6 中每一条边都被存储起来并被标记了范围，在具体的查询过程中可以根据在树中存储的范围信息协同工作，根据地理位置划分的区域 pos 不断减小四叉树的搜索范围，从而达到快速查询的目的（其中相同数字代表在四叉树中存储同一条边）。

3.3 基于四叉树边索引的建立

前文已经对 MCL 格的存储和每个 MCL 格对应边的存储的过程做了具体的介绍，为了达到快速查询的目的，本文提出边的聚类树（ECT）。ECT 需要将两种存储方式有机地结合在一起，即在 R 树的空间对象上用四叉树进一步对空间对象进行划分和存储（因为空间对象存储的是 MCL 格）。最终四叉树的叶子节点将存储 MCL 格的每一条边。由此所有的边都被存储了起来，并且这些边都是经过分类的，把相同类的边组合在一起就形成了 MCL 格，如图 7 所示。图 7 展示的是 ECT 树的基本结构。

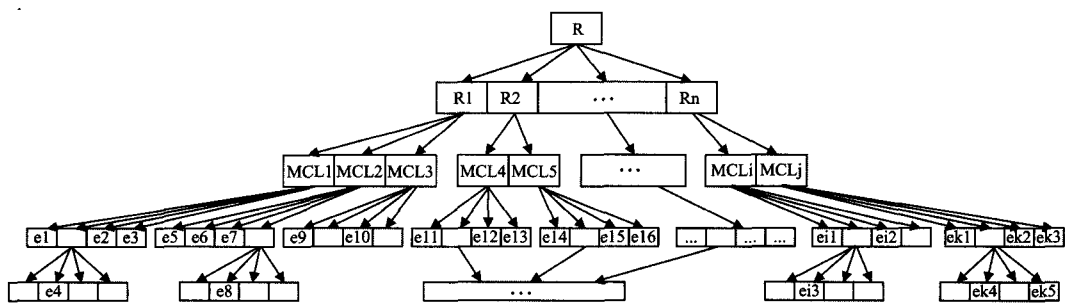

图 7　R 树与四叉树结合后存储路网的示意图

至此前期的预处理工作已经全部完成，且预处理的空间复杂度为 $O(p*\bar{q}+E)$，其中 p 代表 MCL 格的个数，$\bar{q}$ 为所有 MCL 格中的平均边数。而 SILC 算法将空间中不同种类的点及所对应的邻边信息使用四叉树存储，最后用四叉树的二进制形式 Morton Block[14] 方式压缩存储空间使得最终的复杂度为 $O(n^{1.5})$。相比之下，在某些情况下本文提出的 ETC 树所用的存储空间要少一些。

3.4 MCL 格空间索引最短路径算法

对于给定的出发点 $v_s(x_s,y_s)$ 和终点 $v_t(x_t,y_t)$，这里的坐标是根据路网中节点的经纬度而引进的。由于这里给出了两节点之间的确切的坐标位置，那么可以确定从 v_s 到 v_t 的直线路径和这条直线路径的方向 $\vec{e}_s$，这条直线路径无论是否存在，它都会为我们提供最短路径的大致方向，这条路径的给出在很大程度上方便我们快速对最短路径做出选择和判断。具体的最短路径查询算法如下。

首先在 R 树中找到含有起点 v_s 的格的集合 $l=\{l_0,l_1,\cdots,l_k\}$，在确定格的集合后需要进一步落实到具体在哪一个格上，根据虚拟直线路径 s' 即从 v_s 到 v_t 的直线路径和每一个 MCL 格所占区域来判断这条虚拟路径究竟穿过了哪个格。一旦虚拟直线路径 s' 与 MCL 格相交的区域最大，满足 $pos_{virtual}\cap pos_{MCL}=\max(pos)$，那么就可以确定虚拟路径 s' 穿过的格。紧接着在四叉树中进一步寻找这条虚拟路径在四叉树中究竟穿过了哪一条边。具体的方法就是在四叉树中从最顶层开始根据虚拟路径 s' 与四叉树的某一分支相交区域最大，缩小范围后递归向下直至寻找叶子节点，这样的查找方式能够大大减小对边的遍历从而达到快速确定 MCL 格中与 s' 相交的边 e。在确定边之后还需要确定一个点 v_{temp}，这个点将携带从顶点 v_s 到顶点 v_t 过程中其中一个 MCL 格的最短路径，此外这个点亦是下一次搜索的起点。关于这个点的确定，需要根据 $\varphi(e)$ 来确定中点 v_m、边的左顶点 v_l、边的右顶点 v_r。已知起点 v_s，可以唯一确定出 4 个向量 $\vec{e}_l(v_s\to v_l)$，$\vec{e}_m(v_s\to v_m)$，$\vec{e}_r(v_s\to v_r)$，$\vec{e}$（起点到终点的向量）且若满足 $(\vec{e}_m\times\vec{e})\cdot(\vec{e}_x\times\vec{e})<0$，那么就取 $\vec{e}_x$ 上的顶点，这里只需要把 $\vec{e}_l$ 或者 $\vec{e}_m$ 向量中的一个代入进行尝试就能确定出目标点 v_{temp}。至此算法就在格中确定了两个顶点 v_s 和 v_{temp}，接下来可以利用前文提到的格的性质 1 来快速确定出从 v_s 到 v_{temp} 的最短路径。由性质 1，在一个格内每个顶点有且仅有两条路可以选择。根据事先已在 R 树中存储的 MCL 格的总权重 W 可以选择其中一条路，$w-\sum_{i=s}^{temp-1}l_k(w(v_i\to v_{i+1}))<\sum_{i=s}^{temp-1}l_k(w(v_i\to v_{i+1}))$，那么这条路就是确定的最短路径，否则就是格中的另外一条路。另外由于路网的规则性和平面性的特点，还可以用向量的方式一定程度上减小搜索量。在起点 v_s 所在的格内对应于 v_s 存在着两条边，设其中一条以 v_s 为起点的边的向量为 $\vec{e}_1$，另一条为 $\vec{e}_1$，取 $\min(|\frac{\vec{e}_1\cdot\vec{e}}{|\vec{e}_1||\vec{e}|}|,|\frac{\vec{e}_2\cdot\vec{e}}{|\vec{e}_2||\vec{e}|}|)$，即沿着与虚拟路径 s' 夹角较小的边出发。到此为止，第一个点在格中的过程已经完成了，接下来需要把 v_{temp} 作为新的起点通过递归调用的方式依次确定下一个格和下一个格中的最短路径，直至找到终点 v_t。

但是如果在搜寻的过程中遇到边界点 B，且终点与起点不在同一个大格区域中时，可以通过边界点跳到下一个格区域。即在边界区域查看相邻的格，根据虚拟路径确定格后，在确定的格中再调用搜索算法即可（详见算法 2）。

引理（直线穿过路途中子格对应的最短路径的和路径也是最短路径） 对于给定的有权路网 $G=\{V(G),E(G),\varphi_G\}$ 和权重函数 $w:E\to R$。设 $p=\{v_a,v_b,\cdots,v_{k_m}\}$ 为从顶点 v_a 到 v_{k_m} 的一条格中的最短路径。对于任意的 i 和 j 满足 $a\leqslant i\leqslant j\leqslant k$，$p_{ij}$ 就是格中的一条边，设 $l=\{l_0,l_1,\cdots,l_n\}$ 为从顶点 v_s 到顶点 v_t 直线所要经过的 MCL 格的集合。那么就有：

$$\sum_{m=0}^{n}\sum_{i=a,j=i+1}^{k_m}\min(l_m(w(p_{ij})))$$

证明：已知从 v_0 到 v_k 的直线路径 p_{0k} 经过 m 个格，则有最短路径 $w(p_{0k})=\sum_{i=1}^{m}\min(w(l_i))$，现假设存在一条从 v_0 到 v_k 的直线路径 p'_{ok}，且 $w(p_{0k})>w(p'_{0k})$，那么一定存在 $w(l_i')<w(l_i)$，这与每个格中的路径都是最短路径相矛盾，证毕。

算法 2

```
1. shortest_path_qRtree(R, v_s, v_t, path)
2. /* R 代表 R 树索引，v_s 是目标中给出的起点，v_t 是终点，path 用来返回从起点到终点的最短路径 */
3. Vertex v_temp = v_s, v_temp2, v_m
4. /* v_temp 是临时的节点，该节点负责递归调用时起到新起点的作用，v_m 是两个确定点的终点，v_temp2 是在求解过程中 MCL 格中产生的中间节点，它进一步为求出格中的最短路径起到重要作用 */
5. Lattice [ ]L, L_t
6. /* [ ]L 为格的数组用以存放于某一定点相关的格的集合 L_t 代表某一具体的已经确定的格 */
7. Edge e
8. /* e 代表格中虚拟路径穿过的边 */
9. if (v_temp ≠ v_t)
10. L = getadjacent_L_s(R, v_temp)
11. /* 找出包含 v_temp 的相邻的 MCL 格 */
12. pos_max = pos_virtual(v_temp, v_t) ∩ pos_MCL(L[i++])
13. while not (lastone(L[i]))
14. if (pos_virtual(v_temp, v_t) ∩ pos_MCL(L[i]) > pos_max))
15. pos_max = pos_virtual(v_temp, v_t) ∩ pos_MCL(L[i++])
16. end if
17. end while
18. L_t = determine(pos_max)
19. /* 利用 R 树存储的信息，通过判断式子初始时 i=0 从第一个格开始判断直至找到满足条件的格区域把它放入 L_t 中 */
20. e = get_e_from_qtree(L_t · qtree, v_temp, v_t)
21. /* 利用递归算法和路网的平面性的特点在 L_t 格的四叉树中寻找边，即不断深入搜索直到找到最后一个叶子节点 */
22. v_m = getmidvertex(e)
23. getvector(e_l, e_m, e_r, e, v_m, v_temp, φ(e))
24. /* 根据出发点坐标 v_temp，中点坐标 v_m，边 e 上的两个坐标可确定 e_l, e_m, e_r 向量 */
25. if ((e_n × e) · (e_l × e) < 0)
26. v_temp2 = getvertex(e_l)
27. else
28. v_temp2 = getvertex(e_r)
29. end if
30. /* 找出在边 e 上的两个顶点中靠近虚拟路径 s' 与边 e 交点的哪一个顶点 */
31. save_path_in_MCL(v_temp, v_temp2, path)
32. /* 把格中的最短路径存储在 path 中 */
```

33. $v_s = v_{temp2}$

34. shortest_path_qRtree(R, v_s, v_t, path)

35. /* 进行递归调用直到找到终点为止 */

36. end if

图 8 所示为从起点 A 到起点 B 的查询过程图。

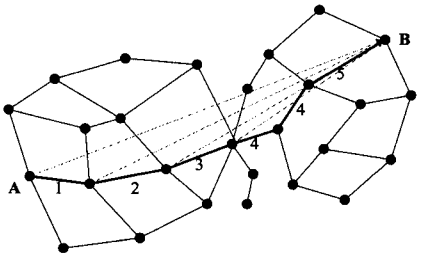

图 8 最短路径查询过程图

图 8 中相同数字代表在同一个 MCL 格内的最短路径中，不同的数字代表在不同的 MCL 格内搜寻最短路径，数字的大小代表搜索过程中的先后顺序，这些由 MCL 格按顺序得到的边连在一起，就构成了从起点 A 到终点 B 的最短按路径(图中已用粗线标出)。

算法 2 的空间复杂度分析：算法 2 在运行过程中，实际上是对 ECT 树的查询操作，在运算过程中没有引入额外的辅助空间，因此该算法的空间复杂度可近似地认为是 ECT 树的空间复杂度。ECT 树的实质就是对边聚类形成 MCL 格并将其存储，因此树中实际需要存储的是 MCL 格的信息和边的信息。其空间复杂度可近似地认为是 $\theta(P+E)$，其中 p 为 MCL 格的个数。

算法 2 的时间复杂度分析：其在最短路径的搜索速度上相对传统算法而言有着显著的提高。传统的 Dijkstra 算法和一些基于新的数据结构的算法如文献[11-13]的复杂度仍然是以 n 为数量级的，所以以上的算法也并不是十分的迅速。然而本文提出的 MCL 格空间索引最短路径算法(算法 2)的时间复杂度与路网中的节点数无关，与起点到终点两点之间的实际欧氏距离无关，与路网中的边数无关。算法 2 的时间复杂度只取决于路网中从起点到终点虚拟路径所经过的 MCL 格的个数 m 和每个顶点对应的 MCL 格的个数 r_i，以及虚拟路径所经过某一个 MCL 格中的边数 q_i。

对于大城市的规则的路网而言，算法 2 还是非常有效的，时间复杂度是 $\theta(\sum_{i=1}^{m}(\log_4 q_i + r_i))$，而根据实际路网的情况，$q_i$ 值和 r_i 值的分布往往是相对平均的，所以以 $\bar{q}$ 和 $\bar{r}$ 来代替每一个 q_i 和 r_i，于是可近似认为算法时间复杂度为 $O(m*(\log \bar{q} + \bar{r}))$，相比文献[13]中的 $O(k)$ 时间复杂度而言有一定程度的提高，k 的大小与起点和终点连接的边数有关，这其中蕴含着大量无用的边，而本文的算法巧妙地把虚拟路径和 MCL 格相互结合在一起，一定程度上减少了 k 的数量，从而使得算法的时间复杂度得以减少。

4 路网的存储和相关算法

本节将通过实验验证 MCL 格空间查找算法的预处理的存储花费和查询等各方面的指标。

本文所有实验均在一台装有 windows 7、配备 3.2GHz CPU 和 8GB 内存的 PC 机上运行，采用的编程语言为 C++。

表 1 给出了仿真实验路网的节点数、边数，实验数据来源于文献[9]，本文将与 Dijkstra[1]、SLIC[2,15]、SPB 树[13]、网络分割法[12]、ALT[18]、PCPD[16,17]、TNR[4]、椭圆算法[11]对比来检测 ECT 存储方式在空间上的效率和 MCL 格算法在时间上的效率。

表 1 实验数据集

名称	对应区域	点数	边数
CA	California and Nevada	1890815	4657742
E-US	Eastern US	3598623	8778114
W-US	Western US	6262104	15248146
C-US	Central US	14081816	34292496
US	United state	23947347	58333344

实验中为了验证算法对于城市路网的普遍适用性，我们对多个不同的城市进行测试，在测试的过程中对每个城市路网生成 8 个小的数据集：$R_1, R_2, R_3, \cdots, R_8$，其中每个子区域的定点数不断增加。

4.1 存储所需的花费

首先，比较存储过程中即预处理过程中的花费。图 9 示出各最短路径查询算法的存储空间随着顶点数量的关系。由于 PCPD 算法和 ALT 算法在预处理中将会占用大量的存储空间(即以牺牲存储空间的方式，达到提高查询效率的目的)，这样的算法显然有一定的局限性，所以在这里不给出 PCPD 算法和 ALT 算法的量化的结果，只给出在存储方面表现较优的算法的比较，如 SILC 算法和 SPB 算法。

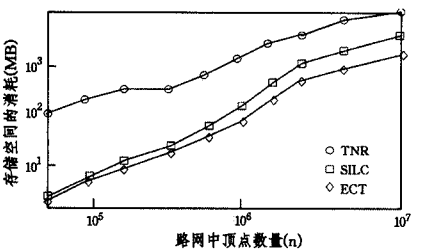

图 9 CA 区所占的存储空间

从图中可以看出，在顶点数量较多的情况下 SILC 算法相对而言存储空间的开销较小，本文提出的 ECT 存储方式在 CA 这条路段上的测试结果显示算法在搜寻中所用的存储空间比 TNR 小 24.35%，比 PCDC 算法小 45.56%，比 SILC 小 8.5%。并且它们的存储量都随着路网规模的增大而增大。

4.2 存储所需的花费

图 10 显示了利用 MCL 格的方法在数据集 US 上进行最短路径查询的效率，通过对比可以发现 SPB 树、网络分割法、椭圆算法、MCL 算法都比 Dijkstra 算法的效率高出很多。随着查询点数量的增加，查询时间也增加，这是因为 SPB 算法的复杂度为 $O(k)$，k 为两点之间最短路径包含的条数，MCL 的时间复杂度为 $O(m*(\log \bar{q} + \bar{r}))$，由于篇幅有限，这里只以 US 地区为例。

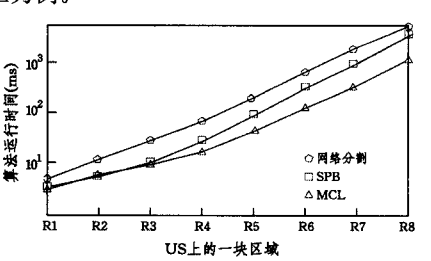

图 10 不同算法查询效率对比

(下转第 247 页)

[11] Yang Jie, Ji Duo, Cai Dong-feng, et al. Keyword Extraction in Multi-Document Based on TextRank Technology[C]// Proc. of NCIRCS'2008(The 2nd Vol.). 2008 (in Chinese)
杨洁,季铎,蔡东风,等. 基于 TextRank 的多文档关键词抽取技术[C]// 第四届全国信息检索与内容安全学术会议论文集(上). 2008

[12] Li Peng, Wang Bin, Shi Zhi-wei, et al. Tag-TextRank: A Webpage Keyword Extraction Method Based on Tags[J]. Journal of Computer Research and Development, 2012, 49(11): 2344-2351 (in Chinese)
李鹏,王斌,石志伟,等. Tag-TextRank:一种基于 Tag 的网页关键词抽取方法[J]. 计算机研究与发展,2012,49(11):2344-2351

[13] Luhn H P. The Automatic Creation of Literature Abstracts [J]. IBM Journal of Research and Development, 1958, 2(8): 159-165

[14] Baxendale P E. Machine-made Index for Technical Literature-an Experiment[J]. IBM Journal of Research and Development, 1958, 2(4): 354-361

[15] Salton G, Wong A, Yan C S. A Vector Space Model for Automatic Indexing[J]. Communication of the ACM, 1995(18): 613-620

(上接第 193 页)

由图 10 可知在查询较优的已有算法中,MCL 算法的查询效率比网络划分算法高 15.26%,在节点数量较低情况 MCL 算法几乎与 SPB 算法相当,在节点数量增多时 MCL 算法比 SPB 算法效率高 15.65%。由此可见,MCL 算法适用于大规模的路网,尤其能在节点数量多的网络突显优势。图 11 以 W-US 数据集进行研究,显示的是在空间路网中节点的边数与查询效率之间的关系。

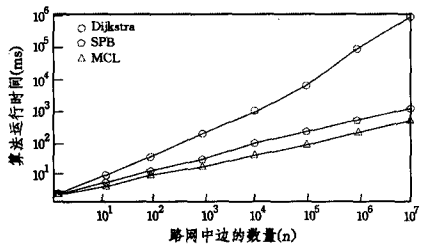

图 11 路网中的边数与查询效率的关系

从图中可以发现边的数量并不影响 MCL 算法的效率。

结束语 本文提出了一种基于虚拟路径和 MCL 格思想的查询算法,结合 ECT 树,达到了快速查询的目的,是查询效率高,所花的存储空间少。利用该算法能够实现快速查询的目的,特别是在大规模的城市路网中查询的高效性尤为显著。在此之前学者们所提出的算法的复杂度都是依赖城市路网节点的规模,而本算法的时间复杂度只与两点之间虚拟路径所经过的 MCL 格有关,因此省去了很多不需要扫描的节点,从而提升了算法的效率。下一步的工作就是如何在 KNN 查询中应用本算法。

参 考 文 献

[1] Dijkstra E W. A note on two Problems in connectionwithgraphs [J]. Numerische Mathematik, 1959, 1(1): 269-271

[2] Samet H, Sankaranarayanan J, Alborzi H. Sealable network distance browsing in spatial databases [C]// The 8th SIGMOD ACM Conference. 2008: 43-54

[3] Klein P N, Mozes S, Weimann O. Shortest Paths in direeted Planar graphs with negative lengths: Aline-space time algorithm [J]. ACM Transactions on Algorithms, 2010, 6(2): 1-13

[4] Papadias D, Zhang J, Mamoulis N, et al. Query processing in spatial network databases [C]// The 3rd VLDB. 2003: 802-813

[5] Cho H J, Chung C W. An efficient and scalable approach to CNN queries in a road network[C]// The 5th VLDB. 2005: 865-876

[6] Sanders P, Sehultes D. Highway hierarchieshasten exact shortest path queries[C]// 13th Annual European Symposium2005. Palma de Mallorca, Spain, October 2005: 568-579

[7] Geisberger R, Sanders P, Sehultes D, et al. Con-Traction hierarchies[C]// Faster and Simpler Hierarchical Routing in Road Networks. WEA, 2008: 319-333

[8] Bast H, Funke S, Matijevie D. Transit: ultrafast shortest-Path queries with linear-time Preprocessing[C]// The 9th DIMACS Implementation Challenge. 2006: 175-192

[9] Deng Ding-xiong. Shortest Path Queries on Road Networks based on Pre-Computation Techniques [D]. Shanghai: FuDan University, 2011

[10] Lee K C K, Lee W C, Zheng Bai-hua, et al. ROAD: A New Spatial Object Search Framework for Road Networks [J]. IEEE Transactions Onknowledge and Data Engineering, 2012, 24(3): 547-560

[11] Wang Shi-ming. Research and Realization of the Shortest Path Algorithm within Topical Urban Road Network[D]. Shandong: Shandong University, 2012

[12] Lu Zhao, Shi Jun. Design and Implementation of parallel shortest path search algorithm[J]. Computer Engineering and applications, 2010, 46(3): 69-71(in Chinese)
卢照,师军. 并行最短路径搜所算法的设计与实现[J]. 计算机工程与应用,2010,46(3):69-71

[13] Deng Ding-xiong. Shortest Path Query for Road Network Based on SPB Tree[J]. Computer Engineering, 2011, 37(22): 56-63

[14] Gargantini I. An Effective Way to Represent Quadtrees [J]. Communication of ACM, 1982, 25(12): 905-910

[15] Sankaranarayanan J, Alborzi H, Samet H. Efficient query Processing on spatial networks[C]// GIS'05. 2005: 200-209

[16] Sankaranarayanan J, Samet H. Distance oracles for spatial network[J]. IEEE Computer Society, 2009, 9: 652-663

[17] Sankaranarayanan J, Samet H, Alborzi H. Path oracles for Spatial networks[J]. PVLDB, 2009, 2: 1210-1221

[18] Goldberg A V, Harrelson C. Computing the shortest Path: A * search meets graph theory [C]// Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms. 2005: 156-165