Journal of Computer Applications
计算机应用,2015,35(2):572 -577
ISSN 1001-9081
CODEN JYIIDU
2015-02-10
http://www.joca.cn

文章编号:1001-9081(2015)02-0572-06
doi:10.11772/j.issn.1001-9081.2015.02.0572

微观交通仿真系统的近邻查询算法

宋　竹*,秦志光,邓伟伟,赵玉平
(电子科技大学 计算机科学与工程学院,成都 611731)
(*通信作者电子邮箱 toni110@163.com)

摘　要:为解决现有微观交通仿真系统中,由于采用基于链表的方法处理近邻查询导致的查询效率和可扩展性不高的问题,提出了一种基于B+树的近邻查询算法。该算法借鉴了数据库中近邻查询算法的思想并结合了链表结构的优点,在叶子节点维护了其存储的数据单元(即车辆)关系的索引结构,以达到快速查询车辆所在车道的前后车的目的。同时,在假设车辆随机分布的前提下,构建了一个数学模型,根据道路的属性和道路中车辆的数量,计算查询目标车辆的邻近车辆所需的最短平均查询长度,并通过此最短查询长度推导算法参数的最优值。理论分析和对比仿真实验显示,衡量该算法的主要指标:即仿真每个车辆的平均时间消耗,在三类常见的交通参数设置(正常的、较拥堵的和拥堵的)下较链表和B+树分别降低了64.2%和12.8%。仿真结果表明该算法具有良好的可扩展性,更适用于大规模微观交通仿真系统。

关键词:微观交通仿真;B+树;链表;近邻查询;参数优化
中图分类号:TP391.9　　**文献标志码**:A

Nearest neighbor query algorithm in microscopic traffic simulation system

SONG Zhu*, QIN Zhiguang, DENG Weiwei, ZHAO Yuping
(*School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan* 611731, *China*)

Abstract: Since methods based on linked list in existing microscopic traffic simulation systems are not efficient and scalable to process Nearest Neighbor (NN) queries, a variation of B+ tree based method was proposed to resolve these problems. This method combined ideas from NN queries in database with advantages of linked list. By maintaining indices of nearby vehicles of each vehicle in the local lane, the performance of NN queries in that lane could be largely improved. Under the assumption of randomly distribution of vehicles, a mathematical model was also proposed to optimize the parameter setting according to multiple parameters for lanes and the amount of vehicles. This model calculated the minimized average query length of each NN query to optimize the parameter setting. The results of theoretical analysis and simulations showed that in common traffic conditions including sparse, normal and congestion, the main indicator, namely the average simulation time cost of each vehicle, could be reduced by 64.2% and 12.8% compared with linked list and B+ tree respectively. The results prove that the proposed method is suitable for larges-cale microscopic traffic simulation systems.

Key words: microscopic traffic simulation system; B+ tree; linked list; nearest neighbor query; parameter optimization

0　引言

交通仿真系统通过计算机软件并使用数学模型对交通系统的真实行为进行模拟,有利于学习、规划、设计和优化交通系统。在交通仿真系统中,近邻查询扮演了一个重要的角色。因为在每一个仿真周期中,所有车辆都需要查询其邻近车辆的行驶状态来决定其自身的行驶状态。

提高近邻查询效率的难点在于其拥有以下特性:1)多一维空间。如果将每一条车道看作一个一维空间,那么一条道路可以看作一个多一维空间的集合。2)高并发性。仿真的车辆数越多,每个仿真周期的近邻查询数量也越多。

现有的交通仿真系统,包括Paramics[1]、Vissim[2]、MITSim[3]、SUMO[4]等,均采用基于链表的方式处理近邻查询。此类方法为每条车道上的车辆序列建立索引,使得创建和维护的开销非常小。但基于链表方法的可扩展性却非常差,当查询某一车辆时,需要遍历整个链表,其时间复杂度为$O(N)$。

借鉴数据库中处理一维和二维近邻查询算法的思想,本文提出了一种B+树的变种——LLB+树(Local Linked B+树)。首先,通过实现B+树叶子节点的双向排序,使得B+树适用于交通仿真系统中的多一维近邻查询。在此基础上,LLB+树同时在叶子节点维护了每台车辆在其同车道上的近邻车辆的索引。此外,通过构建优化LLB+树参数的数学模型,即可根据道路和车辆的属性,通过计算近邻查询的期望查询长度的最小值来确定LLB+树参数的最优值。

理论分析显示,LLB+树的时间复杂度为$O(\log N)$。通过

收稿日期:2014-09-03;**修回日期**:2014-10-28。
基金项目:国家863计划项目(2011AA010706);国家自然科学基金资助项目(61170041)。
作者简介:宋竹(1983-),男,四川成都人,博士研究生,CCF会员,主要研究方向:数据挖掘、交通仿真;　秦志光(1956-),男,四川隆昌人,教授,博士生导师,博士,主要研究方向:信息安全;　邓伟伟(1990-),男,安徽芜湖人,硕士研究生,主要研究方向:数据挖掘;　赵玉平(1987-),男,四川广元人,硕士,主要研究方向:交通仿真。

优化近邻查询的平均查询长度可以进一步提升该算法的性能。在对比仿真实验中比较了链表方法、B + 树和 LLB + 树在不同的交通环境参数设置下的查询效率,根据分析各种参数下三种方法的仿真数据以及配对样本非参数检验,结果显示,在主要的评价指标——仿真每个车辆的平均时间消耗上,LLB + 树优于其他两种方法。

LLB + 树的特性可以总结如下:1)支持多种查询,包括范围查询和反向最近邻查询;2)适用于交通仿真系统中的近邻查询,其时间复杂度为 $O(\log N)$;3)其维护开销与 B + 树相当,高于链表结构,但仍处于可接受的范围。

1　相关工作

1.1　数据库中的近邻查询

在数据库领域,近邻查询也被称作相似性查询或最近点查询。传统的一维和二维近邻查询通常使用的索引结构包括 B 树、B + 树、四叉树和 R 树。文献[5]通过 R 树有效地查询目标的 K 个最近邻;文献[6]构建了 EXO 树以提高近邻查询的速度;文献[7]提出了一种基于 B + 树的方法查询高维度空间中目标的 K 个最近邻;文献[8]根据目标间的距离设计了一个通用的近邻查询框架及其相应算法;文献[9]通过一种随机算法查找目标的最近邻;文献[10]提出了一种道路网络中 K 个近邻查询的完整性验证方法;文献[11]在保证产生最少候选目标的条件下,改进了多步 K 近邻查询算法。

这类一维和二维的近邻查询算法不能直接用于微观交通仿真系统,其原因在于微观交通仿真系统中的近邻查询是一个多一维的问题。

1.2　交通仿真系统中的近邻查询

交通仿真系统中的近邻查询最少需要查询 2 个邻近车辆。当发起查询的车辆没有相邻车道时(即当前道路只有 1 条车道),需要查询其在当前车道的 2 个邻近车辆;当发起查询的车辆有一条相邻车道时,需要查询其在当前车道和此相邻车道各 2 个邻近车辆;当发起查询的车辆同时有左右 2 条相邻车道时,则需要查找此 3 条车道上各 2 个邻近车辆。

虽然不同的交通仿真系统对邻近车辆的定义略有不同(例如,VISSIM[2] 不认为当前车道中的最近后继车辆为一个邻近车辆),这些系统都采用类似的线性索引结构,例如基于链表的索引方法来处理近邻查询。著名的大规模交通仿真软件:如支持超过 100 万仿真节点的 Paramics[1] 使用线性队列存储道路上所有的车辆(不考虑车道);麻省理工大学开发的 MITSim[3] 则使用链表存储每条车道上的车辆;开源微观交通仿真软件 SUMO[4] 也使用类似的线性队列存储每条车道上的车辆。

本文将这些线性的存储/索引结构统称为基于链表的方法。此类方法因其非常容易创建和维护,且在仿真较稀疏的交通状况时也有良好的表现,故在现有交通仿真系统中被普遍采用。此类方法缺点在于其可扩展性较差,当查询某一车辆时,需要遍历整个链表;当仿真节点越多,此类方法的性能也越差。

2　适用性分析和问题陈述

本章分析了传统解决一维和二维近邻查询的方法在交通仿真系统中的适用性问题,并形式化地描述了交通仿真系统中的近邻查询问题。

2.1　适用性分析

传统解决一维和二维近邻查询的方法不适用于交通仿真系统,下面将通过一个例子分析其原因。

图 1 是一个包含 8 辆车的 3 车道路段,其中车辆 A 和 B 处于车道 1,车辆 C、D 和 E 处于车道 2,车辆 F、G 和 H 处于车道 3。当车辆 D 发起近邻查询请求时,它需要查找 6 个邻近车辆,分别是:当前车道(车道 2)的最近前驱车辆 C 和最近后继车辆 E;左相邻车道(车道 1)的最近前驱车辆 A 和最近后继车辆 B;右相邻车道(车道 3)的最近前驱车辆 G 和最近后继车辆 H。

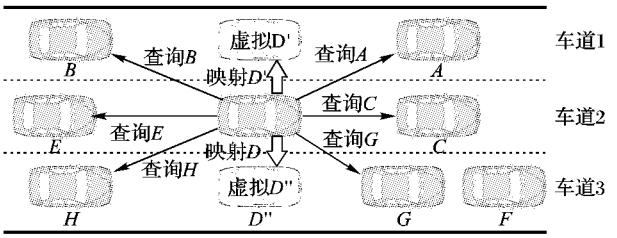

图 1　3 车道的路段

当采用传统的解决一维近邻查询问题的方法时,道路的每一条车道被看作一个线性空间,即根据车道的方向,将每条车道中的车辆顺序以链表方式存储,以图 1 为例,车道 1 的链表顺序为 $A \leftarrow B$,车道 2 为 $C \leftarrow D \leftarrow E$,车道 3 为 $F \leftarrow G \leftarrow H$。为了查找到车辆 D 在不同车道的邻近车辆,在车辆 D 的左右相邻车道分别映射了在车道中有着同样位移属性的虚拟车辆 D' 和 D''。传统的一维方法分别查找 D、D' 和 D'' 的 2 个最近邻作为输出。但是,这些最近邻车辆不一定是车辆 D 的邻近车辆:D'' 的 2 个最近邻车辆为 G 和 F,其中 F 并不是 D 的邻近车辆。

当采用传统解决二维近邻查询问题的方法时,整条道路被视为一个平面。传统的二维方法查找平面中车辆 D 的 6 个最近邻作为输出,然而这些最近邻也不一定是车辆 D 的邻近车辆,其中车辆 F 虽然更靠近 D,但其并不是 D 的邻近车辆。

2.2　问题陈述

集合 V 是在一条 L 车道的道路中车辆的集合,v_i 表示集合 V 中的第 i 个车辆。本文使用两个属性描述每个车辆:车辆在道路中的位移 $x(x > 0)$ 和车辆所在的车道数 $y(1 \leqslant y \leqslant L)$。换言之,每个车辆 v_i 可以用一个元组 (x_i, y_i) 表示。根据车辆在道路中的位移 x 的不同,集合 V 可以划分为 2 个子集合:前驱车辆子集合 $Pre(v_i) = \{v_k : x(k) \geqslant x(i)\}$ 和后继车辆子集合 $Next(v_i) = \{v_k : x(k) < x(i)\}$。而 v_i 的邻近车辆则为其当前车道与相邻车道的最近前驱、最近后继车辆。

一个近邻查询可以分为以下 3 步:

1)查找当前车道的邻近车辆。v_i 在当前车道的最近前驱车辆为集合 $Pre(v_i)$ 中 x 值最小的元素:$(\min\{x_k\}, y_i)$;v_i 在当前车道的最近后继车辆为集合 $Next(v_i)$ 中 x 值最大的元素:$(\max\{x_k\}, y_i)$。

2)如果存在左相邻车道,查找该车道的邻近车辆。v_i 在其左相邻车道的最近前驱车辆为集合 $Pre(v_i)$ 中 x 值最小的元素:$(\min\{x_k\}, y_i - 1)$;v_i 在其左相邻车道的最近后继车辆为集合 $Next(v_i)$ 中 x 值最大的元素:$(\max\{x_k\}, y_i - 1)$。

3)如果存在右相邻车道,查找该车道的邻近车辆。v_i 在其右相邻车道的最近前驱车辆为集合 $Pre(v_i)$ 中 x 值最小的元素:$(\min\{x_k\}, y_i + 1)$;v_i 在其右相邻车道的最近后继车辆

为集合 $Next(v_i)$ 中 x 值最大的元素：$(\max\{x_k\}, y_i + 1)$。

3 数据结构与算法

本章比较了 LLB + 树与链表和 B + 树的区别，并介绍了 LLB + 树的数据结构及管理算法。

3.1 LLB + 树的数据结构

LLB + 树是 B + 树的变种，同时结合了 B + 树与链表的优势。

在交通仿真系统中，基于链表的方法为每条车道建立索引，而基于 B + 树的方法则为每条道路建立索引。以图 1 为例，可以建立一组链表结构，其中每条链表根据车道的方向顺序地存储车道中的车辆。即可以将图 1 中 8 个车辆的分布映射为一个一维的队列 $[x_E, x_B, x_H, x_D, x_G, x_C, x_A, x_F]$。

与 B + 树相比，LLB + 树在结构上修改了内部节点与叶子节点。对内部节点的修改主要是实现了节点间的双向排序，用以支持其他的查询类型。在 LLB + 树中的每个内部节点均添加了 2 个指针，分别指向其前驱节点和后续节点。

对于叶子节点，LLB + 树维护了每个实体的邻居的线索。具体来说，LLB + 树为每个车辆建立了 2 个指针指向其当前车道的最近前驱车辆和最近后继车辆。

以图 1 的车辆分布为例，根据车辆的一维队列 $[x_E, x_B, x_H, x_D, x_G, x_C, x_A, x_F]$ 可以生成相应的 LLB + 树，如图 2 所示。

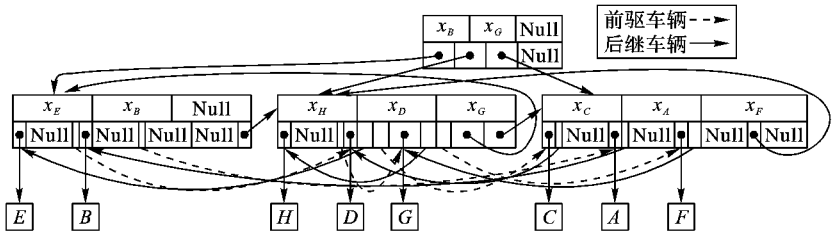

图 2 LLB + 树

LLB + 树结合了 B + 树和链表结构的优点，具体包括：数据指针都存储于叶子节点，构成了底层索引，而内部节点则构成了其他多层的索引[12]。在构建完成 B + 树后，其查询成本为对数级别[13]。在 LLB + 树中同时维护了类似链表的结构，很大程度上提高了查询当前车道的近邻车辆的效率。

3.2 LLB + 树算法

本算法使用车辆在道路上的位移作为 LLB + 树的键值建立索引，并提出了管理 LLB + 树的子算法，包括查询、添加和删除算法。

1）查询算法：解决如何查询车辆在当前道路和相邻道路上的邻近车辆。其中 K_i 为车辆 i 的查询键值，y_i 为车辆 i 所在的车道号。算法伪代码如算法 1 所示。

算法 1　在 LLB + 树中通过键值 K_i 查询车辆 i 的邻近车辆。

①根据 K_i 查找到相应叶子节点；

②根据该叶子节点中车辆 i 的指针 $PrePr_i$ 和 $NextPr_i$ 获得其当前车道的邻近车辆；

③在该叶子节点和其兄弟节点中查找任意车道为 y_{i-1} 和 y_{i+1} 的车辆；

④根据该车辆指针 $PrePr$ 和 $NextPr$，递归查找到车辆 i 的所有邻近车辆。

2）插入算法：根据键值 K_i 在 LLB + 树中新建车辆 i 的索引。其伪代码如算法 2 所示。

算法 2　在 LLB + 树中插入键值为 K_i 的索引。

①在叶子节点建立 $(K_i, Pr_i, PrePr_i, NextPr_i)$；

②查找并更新该叶子节点中 $PrePr_i$ 和 $NextPr_i$ 指针指向的相关叶子节点；

③判断叶子节点和其父节点是否满载，是则分裂节点建立新索引层，直至所有节点都不满载。

3）删除算法：根据键值 K_i 删除其对应的实体。当删除 K_i 对应的实体时，叶子节点和内部节点出现的 K_i 也需要同时删除。算法伪代码如算法 3 所示。

算法 3　删除 LLB + 树中键值为 K_i 的索引。

①递归查找至 K_i 的叶子节点，当 K_i 出现在内部节点中时，使用其左（或右）实体的键值替换 K_i；

②删除叶子节点中 K_i 的实体并更新 $PrePr_i$ 和 $NextPr_i$ 指针，以及其指向的实体的对应指针；

③当节点下溢（实体数低于阈值）时，合并或分裂兄弟叶子节点；

④当合并或分裂兄弟叶子节点引起内部节点下溢时，继续合并或分裂内部节点直至无溢出。

4 参数优化

4.1 查询命中率分析

在 LLB + 树中，所有数据指针都存储于叶子节点。为了便于描述，同一叶子节点中的车辆被称为一个“车辆组”。假设车辆在道路中随机分布，则近邻查询的效率，即近邻查询的期望查询长度 ε，由在一个车辆组中的近邻查询命中率 P（成功查找到目标）决定；而查询命中率 P 则受到一个车辆组中车辆的平均数量 q 影响。本文通过计算 q 的最优值以最小化近邻查询的期望查询长度 ε。

假设有 N 个车辆随机分布在一条单向的 L 条车道的道路。对于 q 个车辆，可以用一个 $q \times L$ 的矩阵描述其在此道路上所有可能的分布。矩阵中每个元素代表一个车辆在道路中可能的位置。例如，a_{ij} 代表第 j 条车道中的第 i 个位置。

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1(L-1)} & a_{1L} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2(L-1)} & a_{2L} \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{i(L-1)} & a_{iL} \\ \vdots & \vdots & & \vdots & & \vdots & \vdots \\ a_{(q-1)1} & a_{(q-1)2} & \cdots & a_{(q-1)j} & \cdots & a_{(q-1)(L-1)} & a_{(q-1)L} \\ a_{q1} & a_{q2} & \cdots & a_{qj} & \cdots & a_{q(L-1)} & a_{qL} \end{pmatrix}$$

根据车辆所在车道的不同，当查询车辆 a_{ij} 在其相邻车道上的邻近车辆时（车道数量大于 2）存在以下两种情况：

第一种情况　当车道数 $L \geqslant 2$，车辆 a_{ij} 处于边缘车道时，需要从 1 个相邻车道中查找 a_{ij} 邻近车辆。此时需要计算车

辆 a_{ij} 在当前车辆组中的近邻查询命中率,记为 P_A。P_A 可由下式所得:

$$P_A = C_{q\cdot L-(q+1)}^{q-1} / C_{q\cdot L-1}^{q-1}$$

第二种情况　当车道数 $L > 2$,且车辆 a_{ij} 处于中间车道时,需要从 2 个相邻车道上的车辆中查找 a_{ij} 的邻近车辆。

设 P_B 为在一个车辆组中查找到 a_{ij} 在相邻车道的邻近车辆的概率,设 P_B' 为其他 $q-1$ 个车辆未分布在相邻车道的可能分布。当不考虑车辆在其中一条相邻车道的分布时,车辆在另一条相邻车道的可能分布为 $C_{q\cdot L-(q+1)}^{q-1}$。根据容斥原理,可以计算出 P_B':

$$P_B' = 2C_{q\cdot L-(q+1)}^{q-1} - C_{q\cdot L-(2q+1)}^{q-1}$$

P_B 可由下式得到:

$$P_B = 1 - P_B' / C_{q\cdot L-1}^{q-1} = 1 - [2C_{q\cdot L-(q+1)}^{q-1} - C_{q\cdot L-(2q+1)}^{q-1}] / C_{q\cdot L-1}^{q-1}$$

总之,根据不同的车道数量,相邻车道中的近邻查询命中率 P 可以表示为:

$$P = \begin{cases} 1, & L = 1 \\ P_A, & \text{相邻车道数} = 1 \\ [2P_A + (L-2)P_B]/L, & \text{相邻车道数} = 2 \end{cases}$$

本文分析了查询命中率 P、一个车辆组的平均车辆数 q 和车道数 L 的关系,如图 3 所示。结果显示,在常见的道路中,即车道数 $1 < L \leqslant 5$ 时,查询命中率 P 的收敛速度随车道数 L 的增加而降低。

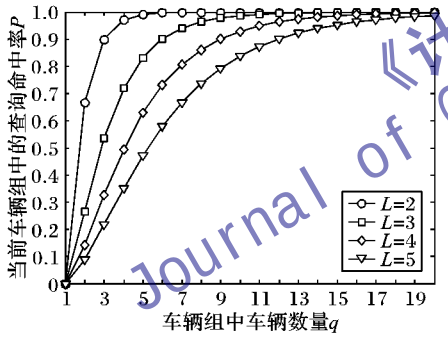

图 3　P、q 和 L 的关系

4.2　时间复杂度分析

与 B + 树相同,LLB + 树的查询时间复杂度为 $O(\log N)$,其中:N 为道路中车辆的数量;z 是 LLB + 树中每个内部节点的最小孩子数。

当查询当前车道中车辆 i 的邻近车辆时,可以通过指针 $PrePr_i$ 和 $NextPr_i$ 直接获得。故查询当前车道中车辆 i 的邻近车辆的时间复杂度为 $O(\log N)$。

当查询相邻车道中车辆 i 的邻近车辆时,本文使用期望查询长度 ε 衡量查询的效率。LLB + 树需要遍历一个叶子节点(一个车辆组)以查找目标车辆的邻近车辆。在一个车辆组中找到邻近车辆的期望查询长度为 $q/2$;在当前车辆组中找到邻近车辆的期望查询长度为 $P(q/2)$;在第 k 个车辆组中找到邻近车辆的期望查询长度为:

$$0.5(k-1)q + (1-P)^{k-1}P \times 0.5q$$

LLB + 树中的叶子节点数量,即车辆组的数量为 $\lceil N/q \rceil$,则在整个 LLB + 树中查找到邻近车辆所需的期望查询长度 ε 为:

$$\varepsilon = \sum_{k=1}^{\lceil N/q \rceil} (1-P)^{k-1}P \times 0.5kq$$

使用错位相减法可得:

$$\varepsilon = 0.5q\left(\sum_{k=0}^{\lceil N/q \rceil}(1-P)^k - (1-P)^{\lceil N/q \rceil}\cdot\lceil N/q \rceil\right)$$

根据等比数列求和公式,可将期望查询长度 ε 简化为:

$$\varepsilon = 0.5q\cdot([1-(1-P)^{\lceil N/q \rceil}]P^{-1} - (1-P)^{\lceil N/q \rceil}\cdot\lceil N/q \rceil)$$

对于某一确定道路,其车道数 L 为常数,查询命中率 P 随着一个车辆组中车辆的数量 q 的增长趋近于 1。对于任意一个 L,可以通过有限的 q 得到一个“足够大”的 P。因此这里 q 也为常数,而 $(1-P)^{\lceil N/q \rceil}$ 的极限 $\lim_{N\to\infty}(1-P)^{\lceil N/q \rceil} = 0$,故可计算 ε 的极限值:

$$\lim_{N\to\infty}\varepsilon = q/2P$$

即在车辆随机分布的假设下,查询相邻车道中车辆 i 的邻近车辆的时间复杂度同样为 $O(\log N)$。车道数 L、一个车辆组内的平均车辆数 q 和在相邻车道查询邻近车辆的期望查询长度 ε 的关系在 N 足够大的情况下($N = 1000$)如图 4 所示。

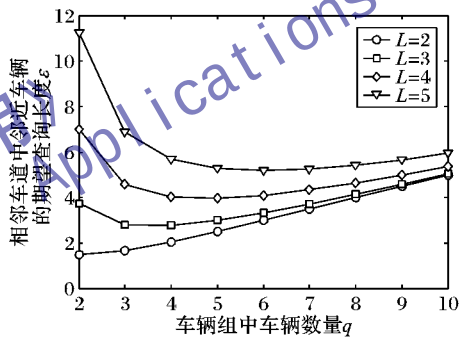

图 4　ε、q 和 L 的关系

对车辆数 N 的分析显示,对于每一确定的车道数 L 和一个车辆组中车辆的平均数量 q,N 存在一个阈值。当 N 的取值大于该阈值时,在相邻车道查询邻近车辆的期望查询长度收敛为常数值;当 N 的取值小于该阈值,期望查询长度会随着 N 的减少而减小。N 的阈值和不同 L 对应的最优 q 值如表 1 所示,其中的期望查询长度 ε 并未包含从根节点到叶子节点的查询长度。

表 1　最优 q 值和 N 的阈值

车道数 L	车辆组数量最优值 q	期望查询长度 ε	N 的阈值
2	2	1.5000	23
3	4	2.7809	41
4	5	3.9754	71
5	6	5.1911	91

4.3　节点大小优化

节点大小优化的意义在于降低 LLB + 树中为维护其时间复杂度而分裂、合并内部及叶子节点的频率。节点大小的优化即优化 LLB + 树的重要参数:内部节点的最小孩子数 z(最小子树)。而 z 决定了 q 的取值范围,根据 B + 树的规则:q 的取值必须在叶子节点的取值范围内,即大于取值下限 $z-1$ 且小于取值上限 $2z-1$。

对于每个最优的 q,需要计算相应 z 的最优取值,而对于每个 q,存在多个可能的 z。本文通过构建数学模型比较每个

z 的可能取值所对应的期望查询长度 ε，该模型思想如下：

对于某一确定的 q，其对应 z_i 的取值范围为：$0.5(q+1) \leqslant z_i \leqslant q+1 (i \in [1, \lceil (q+1)/2 \rceil + 1])$；对于每一个 z_i，对应 q_{ij} 的可能取值范围为：$z_i - 1 \leqslant q_{ij} \leqslant 2z_i - 1 (j \in [1, z_i + 1])$。即可计算 q_{ij} 的期望查询长度，记作 $\varepsilon(q_{ij})$。通过比较不同 z_i 对应的 $\varepsilon(q_{ij})$ 之和的平均值，可以找出 ε_i 的最小值，其所对应的 z_i 即最优的 z，计算方法如下所示：

$$\varepsilon_i = \frac{1}{z_i+1}\sum_{j=1}^{z_i+1}\varepsilon(q_{ij}) = \frac{1}{z_i+1}\sum_{j=1}^{z_i+1}\left(\sum_{k=1}^{\lceil N/q_{ij} \rceil}(1-P)^{k-1}P \cdot \frac{kq_{ij}}{2} + \log_{z_i} N\right)$$

表 2 列举了在 $N = 250$ 和 $N = 1000$ 时最优的 z 值及其所对应的最小 ε 值。由表 2 可见，N 的取值对 z 的最优值影响不大，体现了 LLB + 树良好的可扩展性。

表 2 最优 z 值及对应最小 ε 值

车道数 L	$N = 250$		$N = 1000$	
	z 的最优值	期望查询长度 ε	z 的最优值	期望查询长度 ε
2	3	6.9622	3	8.2241
3	5	7.0266	5	7.8879
4	5	7.7810	6	8.6024
5	6	8.6694	6	9.4431

5 仿真实验

本章通过仿真实验，分别对比了 B + 树、链表及 LLB + 树在拥堵交通状况下近邻查询的性能。

5.1 实验环境

仿真实验使用的道路网络被设为一个封闭空间，即在仿真过程中，仿真车辆的总数不变。

仿真实验中存在 3 个影响仿真结果的变量：道路中的车辆数 N、车道数 L 以及换道率 P_c。其中换道率 P_c 是指每个仿真周期中平均每辆车查询其相邻车道邻近车辆的概率（即换道意图），而不是完成换道的概率。本文将一个仿真周期中仿真每个车辆的平均时间消耗 $T_{response}$ 作为衡量指标。T_i 为仿真车辆 i 的时间消耗，$T_{response}$ 为平均时间消耗，可由下式所得：

$$T_{response} = \frac{1}{N}\sum_{i=1}^{N} T_i$$

对每个 $T_{response}$ 仿真 100 次并取均值以减少实验误差。$T_{response}$ 是由变量 N、L 和 P_c 共同决定的，因此需要同时分析 3 个变量对 $T_{response}$ 的影响。根据常见交通状况，定义了交通状况拥堵、较拥堵及正常时的参数设置。拥堵时参数设为：$P_c \in \{70\%, 80\%, 90\%\}$，$N \in \{800, 900, 1000\}$；较拥堵时参数设为：$P_c \in \{40\%, 50\%, 60\%\}$，$N \in \{500, 600, 700, 800\}$；正常时参数设为：$P_c \in \{10\%, 20\%, 30\%, 40\%\}$，$N \in \{200, 300, 400, 500\}$。

仿真实验中选取了常见的车道数 $L = 3$，交通拥堵时换道率经验值 $P_c = 60\%$，交通拥堵时道路中车辆数经验值 $N = 1000$。出于简化实验的需要，仿真中道路的车辆总数设为 N。

其中车辆数 N 的经验值计算如下：设道路长度 5 km，使用拥堵时的平均时速 $\bar{v} = 30$ km/h，通过保守的安全距离公式计算平均车头间距，可获得 $N \approx 1000$；而换道率的经验值，即换道意图，因取值范围为 0% ~ 100%，且随着拥堵程度的增加，换道意图也会增加，故选取 $P_c = 60\%$ 为参数。

实验使用的仿真平台是一款名为 DMTSS 的自主开发的微观交通仿真系统，采用 Pipes 跟驰模型，运行于 i3 3.40 GHz CPU，4 GB 内存的台式电脑。

5.2 对比实验

本文使用交通拥堵时参数的经验值，通过 3 组实验对比以下方法的性能：1）链表：现有仿真系统中采用的典型线性索引结构。2）B + 树：数据库领域常用的索引结构（叶子节点的双向排序）。3）LLB + 树：本文提出的适用于多一维空间的索引结构及参数优化模型。

1）换道率对性能的影响：换道率 P_c，即车辆查询相邻车道中邻近车辆的概率，是影响交通仿真系统性能的重要参数。本实验取参数 $N = 1000$ 时，衡量 P_c 的变化对三种方法的性能的影响，结果如图 5 所示。

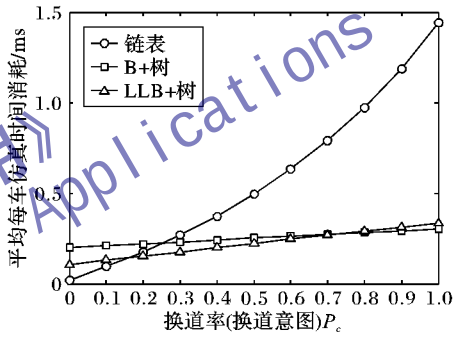

图 5 $N = 1000$，P_c 变化时的效率曲线

从图 5 可以看出，当 $P_c \leqslant 80\%$ 时，LLB + 树优于 B + 树；当 $P_c > 20$ 时，P_c 的增加使得链表的性能大幅下降，其对链表的影响远远大于对 B + 树和 LLB + 树的影响。

2）车辆数对性能的影响：道路中车辆的数量 N 是决定仿真规模的重要参数，对仿真的性能有很大的影响。具体在仿真实验中，N 的上限是由道路的长度和车道数 L 共同决定的。通过使用类似的方法，分析当 $P_c = 60\%$ 时，N 的变化对三种方法性能带来的影响，结果如图 6 所示。

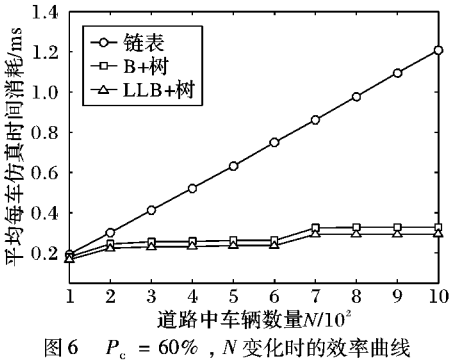

图 6 $P_c = 60\%$，N 变化时的效率曲线

由图 6 可以看出，链表的性能随着 N 的增加呈线性增长，而 B + 树和 LLB + 树受 N 的影响很小。此时三种方法中 LLB + 树最优。

3）车道数对性能的影响：本实验为验证车道数的多少对仿真性能的影响，参数取值 $P_c = 60\%$ 且 $N = 1000$，结果如图

7 所示。

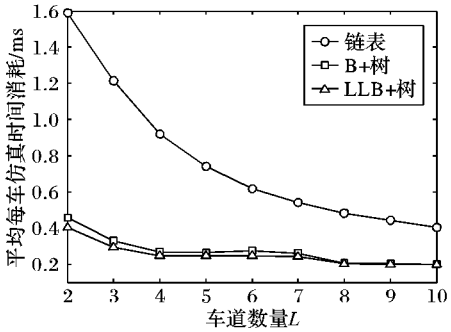

图 7　P_c = 60% 且 N = 1 000,L 变化时的效率曲线

从图 7 可以看出,随着 L 的增加,LLB + 树和 B + 树的效率曲线逐渐逼近,且 LLB + 树和 B + 树均优于链表。

4)评估分析:通过该仿真实验,可以得出 LLB + 树适用于仿真拥堵的交通状况的结论。本文使用类似的方法进行了多组仿真实验,选用多组不同的参数模拟了较拥堵以及正常的交通状况,结果显示在大部分参数设置下,LLB + 树查询效率均优于 B + 树和链表。对所有仿真数据进行统计分析可更全面地评估各种方法在不同交通参数设置下的性能。通过组合拥堵、较拥堵和正常的交通参数设置,获取了各 8 100 条仿真数据,对这些数据的统计分析如表 3 所示。

表 3　仿真数据的统计信息

算法	样本数量	平均 $T_{response}$/μs	方差	最小 $T_{response}$/μs	最大 $T_{response}$/μs
链表	8 100	630.8	311 545.4	41	2 245
B + 树	8 100	259.1	3 189.9	147	375
LLB + 树	8 100	226.1	5 393.1	90	81

可见,对衡量算法效率的主要指标 $T_{response}$,即仿真每个车辆的平均时间消耗(包含创建和维护数据结构的时间消耗),LLB + 树在平均查询速度上比 B + 树提高了 12.8%,比链表方法提高了 64.2%;从方差项可以看出,在稳定性上 LLB + 树较 B + 树有下降,但仍然优于传统的链表结构。

为了验证方差的差异以及 LLB + 树与 B + 树的查询性能,使用 2 组 8 100 条数据进行两配对样本非参数检验(Wilcoxon 检验),判断同样参数设置下使用 LLB + 树的查询时间是否较 B + 树更低。Wilcoxon 检验结果如表 4 所示。

表 4　Wilcoxon 检验结果

查询时间关系	数量	秩均值	秩和
LLB + 树 < B + 树	6 700	4 558.25	30 540 275
LLB + 树 > B + 树	1 340	1 358.77	1 820 752
LLB + 树 = B + 树	60		

从表 4 可以看出,使用 LLB + 树后,8 100 条数据中有 6 700 条数据的查询时间缩短,1 340 条数据的查询时间增加。Z 统计量为 -21.8,相伴概率低于 0.01,低于显著性水平 0.05,因此拒绝零假设,认为 LLB + 树和 B + 树的查询效率存在明显差异,即 LLB + 树优于 B + 树。

6　结语

目前大部分微观交通仿真系统都采用基于链表的方法处理近邻查询问题。此类方法虽然简单有效,但其查询效率和可扩展性都不高,这在很大程度上限制了大规模微观交通仿真的研究与应用。

与传统的基于链表的方法及改进的 B + 树比较,LLB + 树在常见的交通参数设置下能有效降低查询时间消耗,更适用于大规模交通仿真环境。

本文方法的不足之处在于,参数优化模型和理论推导方法需要假设车辆在道路上的随机分布,并不适用与其他的分布情况。因此,微观交通仿真中近邻查询算法的改进和优化仍然需要更多的研究与探索。

参考文献:

[1] CAMERON G D B, DUNCAN G I D. PARAMICS — Parallel microscopic simulation of road traffic [J]. The Journal of Supercomputing, 1996, 10(1): 25 - 53.

[2] FELLENDORF M. VISSIM: a microscopic simulation tool to evaluate actuated signal control including bus priority [C]// Proceedings of the 64th Institute of Transportation Engineers Annual Meeting. [S.l.]: ITE, 1994: 1 - 9.

[3] YANG Q, KOUTSOPOULOS H N. A microscopic traffic simulator for evaluation of dynamic traffic management systems [J]. Transportation Research Part C: Emerging Technologies, 1996, 4(3): 113 - 129.

[4] KRAJZEWICZ D, BONERT M, WAGNER P. The open source traffic simulation package SUMO [C]// RoboCup 2006: Proceedings of the 2006 Infrastructure Simulation Competition. [S.l.]: RoboCup Federation, 2006, 1: 1 - 5.

[5] ROUSSOPOULOS N, KELLEY S, VINCENT F. Nearest neighbor queries [C]// SIGMOD '95: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1995: 71 - 79.

[6] HU H, LEE D L. Range nearestn-eighbor query [J]. IEEE Transactions on Knowledge and Data Engineering, 2006, 18(1): 78 - 91.

[7] JAGADISH H V, OOI B C, TAN K-L, *et al.* iDistance: an adaptive B^+-tree based indexing method for nearest neighbor search [J]. ACM Transactions on Database Systems, 2005, 30(2): 364 - 397.

[8] HJALTASON G R, SAMET H. Index-driven similarity search in metric spaces (survey article) [J]. ACM Transactions on Database Systems, 2003, 28(4): 517 - 580.

[9] ARYA S, MOUNT D M. Approximate nearest neighbor queries in fixed dimensions [C]// SODA '93: Proceedings of the Fourth Annual ACMS-IAM Symposium on Discrete algorithms. Philadelphia: Society for Industrial and Applied Mathematics, 1993: 271 - 280.

[10] HU L, JING Y, KU W-S, *et al.* Enforcing k nearest neighbor query integrity on road networks [C]// SIGSPATIAL '12: Proceedings of the 20th International Conference on Advances in Geographic Information Systems. New York: ACM, 2012: 422 - 425.

[11] SEIDL T, KRIEGEL H-P. Optimal multi-step k-nearest neighbor search [C]// SIGMOD '98: Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1998: 154 - 165.

[12] ELMASRI R, NAVATHE S B. Fundamentals of database systems [M]. 5th ed. Upper Saddle River: Pearson Education, 2008: 652 - 660.

[13] COMER D. Ubiquitous B-tree [J]. ACM Computing Surveys, 1979, 11(2): 121 - 137.