TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol. 12, No. 12, December 2014, pp. 8175 ~ 8192
DOI: 10.11591/telkomnika.v12i12.6332

The B$^+$-tree-based Method for Nearest Neighbor Queries in Traffic Simulation Systems

Zhu Song*[1], Zhiguang Qin[2], Weiwei Deng[3], Yuping Zhao[4]
School of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu, China
*Corresponding author, e-mail: toni110@163.com[1], zgqin@uestc.edu.cn[2], uestcdengww@hotmail.com[3], zypuestc@live.com[4]

Abstract

Extensive used traffic simulation systems are helpful in planning and controlling the traffic system. In traffic simulation systems, the state of each vehicle is affected by that of nearby vehicles, called neighbors. Nearest neighbor (NN) queries, which are multi 1-dimensional and highly concurrent, largely determine the performance of traffic simulation systems. Majority of existing traffic simulation systems use Linked list based methods to process NN queries. Although simple and effective they are, existing methods are neither scalable nor efficient. In this paper, we propose a B+-tree-based method to improve the efficiency of NN queries by borrowing ideas from methods used in databases. In particular, we create a linked local B+-tree, called LLB+-tree, which is a variation of Original B+-tree, to maintain the index of neighbors of each vehicle. We also build a mathematical model to optimize the parameter setting of LLB+-tree according to multiple parameters for lanes and vehicles. Our theoretical analysis shows that the time complexity of the method is O(logN) under the assumption of randomly distribution of vehicles. Our simulation results show that LLB+-tree can outperform Linked list and Original B+-tree by 64:2% and 12:8%, respectively.

Keywords:

1. Introduction

The traffic simulation system is a mathematical modeling of transportation systems through the application of computer software, which leads to better understanding, planing, designing and optimizing the traffic system. Nearest neighbor (NN) queries play an important role in traffic simulation systems, because each vehicle needs to find nearby vehicles, called neighbors, and determine its state according to neighbors' states.

It is difficult to improve the performance of NN queries because NN queries have the following properties: 1) Multi 1-dimensional cases: if we consider a lane as a 1-dimensional case, then a road with multiple lanes can be seen as infrequent multi 1-dimensional cases. 2) High concurrency: the more vehicles exist in a simulation, the larger number of NN queries occur in each cycle.

Existing traffic simulation systems, Paramics [16], Vissim [17], MITSim [18], SUMO [19] etc, adopt Linked list-based methods to process NN queries. Such methods, which are very easy to create and maintain, index the sequence of vehicles in each lane. However, those methods are not scalable, because they need to traverse the Linked list to find a vehicle. Videlicet, the time complexity of such methods is $O(N)$.

In this paper, we propose a B+-tree-based method, called LLB+-tree (linked local B+-tree) by borrowing ideas from those methods for 1 or 2-dimensional NN queries in databases. In particular, we firstly index all vehicles in each road in a same direction, and implement the bidirectional order of leaf nodes of Original B+-tree. We then maintain links of neighbors for each vehicle in the same lane. We also build a mathematical model to optimize parameters setting for LLB+- tree. Such a model caculates the min value of the expect query length according to numbers of lanes and vehicles.

Our theoretical analysis shows that the time complexity of the LLB+-tree method is $O(\log N)$. The optimal average query length can further improve the performance of the method.

Received May 26, 2014; Revised September 28, 2014; Accepted October 15, 2014

Our simulation results show that LLB+-tree can outperform Linked list and Original B+-tree by 64.2% and 12.8%, respectively.

Overall, the main properties of LLB+-tree are listed as follows:

a) LLB+-tree supports multiple query types, including range query and reverse nearest neighbor (RNN) query.

b) LLB+-tree is efficient in NN queries in traffic simulation systems. The time complexity of LLB+-tree based method is $O(\log N)$.

c) The maintenance overhead of LLB+-tree is acceptable, which is similar to that of the Original B+-tree.

The rest of the paper is organized as follows. Section II gives an overview of the related work. Section III defines NN queries in traffic simulation systems and introduces the data structure of LLB+-tree. Section IV proposes algorithms for managing LLB+-tree. Section V analyzes the time complexity of the method and optimizes parameters. Section VI evaluates the performance of LLB+-tree through both experiments and statistical analysis. Section VII concludes this paper.

2. Relative Work

In this section, we firstly overview methods for NN queries in databases. We then describe methods for NN queries in traffic simulation systems.

2.1. NN Queries in Databases

NN queries, also know as proximity queries, similarity queries or closest point queries, can be divided into two categories: the query for static and moving objects.

NN queries for static objects use index structures, including B-tree, B+-tree, quad-tree and R-tree. Roussopoulos et al. [3] propose an influential method for finding the K-nearest neighbors (KNN) using R-tree; Haibo Hu et al. devise EXO-tree to speed up NN queries [6]; HV Jagadish et al. [7] propose a B+-tree based method, for KNN search in a high-dimensional space; GR Hjaltason et al. [2] devise a general framework and algorithms for performing search based on distance; a randomized algorithm for computing approximate nearest neighbor is proposed by Arya et al. [8]. Ling Hu et al. [9] propose a road network KNN query verification technique to prove the integrity of the query result. Seidl et al. [10] improve a KNN multi-step algorithm which is guaranteed to produce the minimum number of candidates.

NN queries for moving objects mainly use similar index structures, including B+-tree and TPR-tree. Kollios et al. [1] generalize moving objects in a plane, the movements of which are restricted to a number of line segments, as a “1.5-dimensional” case [11]. Jensen et al. [12] develops algorithms for NN queries whose performance is better than TPR-tree. Tao et al. [13] solve the overhead problems in continuous nearest neighbor (CNN) queries. An algorithm which requires only one dataset lookup to deliver a complete predictive result for CNN queries, is devised by Lee et al. [14]. Xie et al. [15] provides a solution which supports different shapes of commonly-used imprecise regions using *u*-bisector. Benetis et al. [11] propose algorithms for responding RNN and NN queries for moving points in plane.

2.2. NN Queries in Traffic Simulation Systems

NN queries in traffic simulation systems aim to find at least 2 neighbors. When the vehicle has no adjacent lane, it only need to find 2 neighbors in the local lane; When the vehicle has one adjacent lane, it needs to find 4 neighbors: 2 in the local lane and 2 in the adjacent lane. When the vehicle has both left and right adjacent lanes, it needs to find additional 2 neighbors in the other adjacent lane.

Although there has minor difference in the definition of neighbors in different simulation systems (e.g., VISSIM [17] do not consider the nearest following vehicle in the local lane as a neighbor), existing traffic simulation systems adopt similar linear methods (Linked list-based methods) for NN queries. Paramics [16], a famous software which supports a simulation over 1 million vehicles, store vehicles currently in linear queues, regardless of lane; While in MITSim [18], a simulator developed by MIT, vehicles are also stored in Linked list in each lane. SUMO [19], an open source, highly portable, microscopic and continuous road traffic simulation package, indexes vehicles in each lane in a linear queue.

Those Linked list-based methods, very cheap in creating and maintaining, are suitable to simulate sparse traffic conditions. However, such methods are not scalable because they need to traverse the Linked list to search a vehicle. Videlicet, the more vehicles in a simulation, the worse performance of those methods.

3. Problem Statement and Data Structures

In this section, we firstly analyze the applicability of traditional methods for 1 and 2-dimensional NN queries in traffic simulation systems. We then propose formal descriptions of such NN queries. At last, we introduce the data structure of B+-tree, double Linked list, LLB+-tree and compare the details of them.

3.1. Applicability Analysis

We use an example to explain why traditional methods for 1 or 2-dimensional NN queries are not suitable for traffic simulation systems.

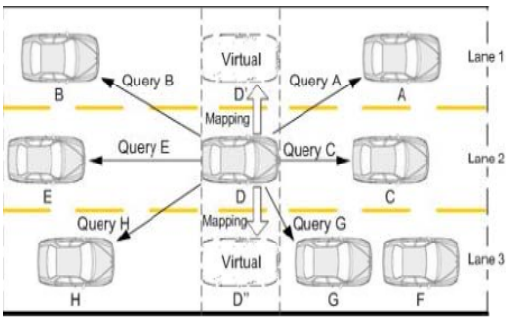

Figure 1. 8 vehicles in a three-lane road segment

Figure 1 is a segment of a road. There are 8 vehicles distributed in 3 lanes: vehicles *A* and *B* in lane 1; vehicles *C*, *D* and *E* in lane 2; vehicles *F*, *G* and *H* in lane 3. When vehicle *D* launchs a NN query, called the initiator, it needs to find 6 neighbors: the nearest leading and following vehicles *C* and *E* in the local lane (lane 2); *A* and *B* in the left adjacent lane (Lane 1); *G* and *H* in the right adjacent lane (lane 3).

If we adopt a method for 1-dimensional NN queries, we consider each lane as a linear space. In order to find the nearest vehicles in each lane, we create virtual initiators *D'* and *D''* with the same displacement of *D* separately in the left and right adjacent lanes. Thus, the method find the 2 nearest vehicles of *D*, *D'* and *D''* in the local, left adjacent and right adjacent lanes, respectively. However, these vehicles may not be the correct neighbors. The 2 nearest vehicles of *D''* in lane 3 are *G* and *F*, while the neighbors are *G* and *H*.

If we use a method for 2-dimensional NN queries, we consider a whole road as a plane. Such a method can find the 6 nearest vehicles of the initiator *D*. While these vehicles may not be the neighbors either. As shown in Figure 1, the 6 nearest vehicles are *A*, *B*, *C*, *F*, *G* and *H*. Vehicle *F* is more closer to *D* than vehicle *E*, but *F* is not a correct neighbor.

3.2. Problem Statement

We denote *V* as the set of all vehicles in a *L*-lanes road, v_i is the ith vehicle. We can further use two properties, the displacement of a vehicle in the road $x|x > 0$ and the lane where the vehicle located $y|1 \leq y \leq L$ to describe each vehicle. That is, v_i can be described in a tuple (x_i, y_i). To find neighbors of v_i, we divide *V* into two sets according to the displacement x. One is the set of leading vehicles: $Pre(v_i) = \{v_k : x(k) > x(i)\}$; The other is the set of following vehicles: $Next(v_i) = \{v_k : x(k) < x(i)\}$. The neighbors of *vi* include the nearest leading and following vehicles in the local and two adjacent lanes.

That is, a NN query contains following 3 steps:

1) We find the neighbors of the vehicle in the local lane. The nearest leading vehicle of *vi* in the local lane is the element with the minimize x in the set $Pre(v_i)$: the vehicle $(min\{x_k\}, y_i)$. The nearest following vehicle of *vi* in the local lane is the element with the maximize x of the set $Next(v_i)$: the vehicle $(max\{x_k\}, y_i)$.

2) If there exists a left adjacent lane, we find the neighbors in that lane. The nearest leading vehicle of v_i in the left adjacent lane is the element with the minimize x in the set $Pre(v_i)$: the vehicle $(min\{x_k\}, y_i - 1)$. The nearest following vehicle of *vi* in the left adjacent lane is the element with the maximize x of the set $Next(v_i)$: the vehicle $(max\{x_k\}, y_i - 1)$.

3) If there exists a right adjacent lane, we find the neighbors in that lane. The nearest leading vehicle of v_i in the right adjacent lane is the element with the minimize x in the set $Pre(v_i)$: the vehicle $(min\{x_k\}, y_i + 1)$. The nearest following vehicle of *vi* in the right adjacent lane is the element with the maximize x of the set $Next(v_i)$: the vehicle $(max\{x_k\}, y_i + 1)$.

3.3. Data Structure of LLB+-tree

The LLB+-tree (Linked local B+-tree), a variation of B+- tree, is a combination of B+-tree and Linked list.

Linked list-based methods index vehicles in each lane, while B+-tree based methods index vehicles in each road. Using an example of a road segment shown in Figure 1, we can build three Linked lists. As shown in Figure 2, each Linked list stores vehicles orderly in a same lane. Figure 3 tells the mapping scheme of LLB+-tree using the same road segment. According to the displacement of vehicles in the road, we can map the distribution of the 8 vehicles into a 1-dimensional queue: $[x_E, x_B, x_H, x_D, x_G, x_C, x_A, x_F]$.

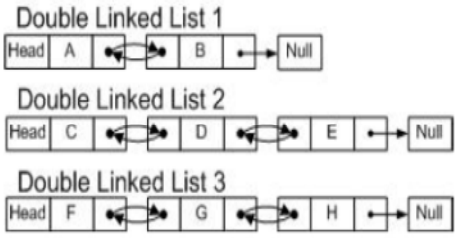

Figure 2. An example of double Linked list

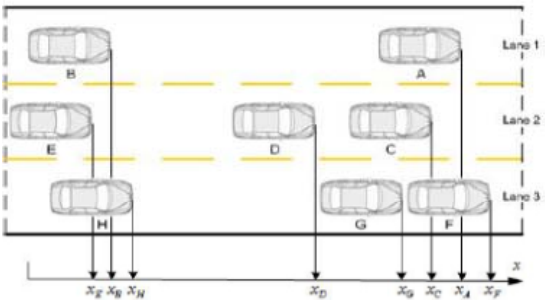

Figure 3. Mapping vehicles into 1-dimensional coordinate

LLB+-tree mainly modifies the structures of internal nodes and leaf nodes. In internal nodes, we implement the bidirectional order of each node to facilitate multi query types. In particular, we create 2 pointers to link the previous and next internal nodes, denote as $PreP$ and $NextP$, respectively. Each internal node has a key values and a + 1 children. The structure of each internal node is printed below, where k_i denotes the $_i$th key value, P_i points the $_i$th child:

$$< P_1, K_1, P_2, K_2, \cdots, P_a, K_a, P_{a+1}, PreP, NextP >$$

In leaf nodes, we maintain the thread of the neighbors of each entity (vehicle). In particular, we create 2 pointers in each entity to link the vehicle's neighbors in the local lane, denote as $\acute{P}rePr_i$ and $NextPr_i$. Thus, each entity has 4 domains: the $_i$th key value k_i, the vehicle data Pr_i, neighbors $\acute{P}rePr_i$ and $NextPr_i$. Each leaf node is formed by entities and two pointers: $PreP$ and $NextP$, which point to the previous and next leaf nodes. The structure of each leaf node is of the form:

$$<< K_1, Pr_1, PrePr_1, NextPr_1 >, \cdots,$$
$$< K_a, Pr_a, PrePr_a, NextPr_a >, PreP, NextP >$$

Using the sample of the distribution of 8 vehicles, the 1-dimensional queue $[x_E, x_B, x_H, x_D, x_G, x_C, x_A, x_F]$, we build a LLB+-tree, shown in Figure 4.

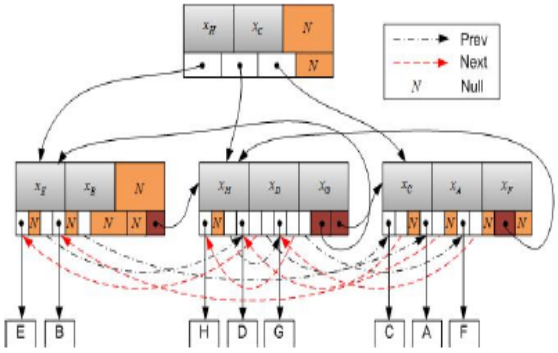

Figure 4. An example of a LLB+-tree

As a combination of B+-tree and Linked list, LLB+-tree inherits a lot of advantages: data points are stored only at the leaf nodes. These leaf nodes are similar to the first (base) level of an index. Internal nodes of B+-tree correspond to the other levels of a multilevel index [20]. B+-tree implementation retains the logarithmic cost properties for operations by key, but gains the advantage of requiring at most 1 access to satisfy a next operation [21]. Just like Linked list, LLB+-tree also maintains the threads of the nearest leading and nearest following neighbors of each vehicle in the local lane, which facilitate NN queries in that lane.

4. Algorithms Optimization

In this section, we adopt the replacement of a vehicle in the road as its search key value in a LLB+-tree. we propose three sub-algorithms for the management of the LLB+-tree, including searching, inserting and deleting.

4.1. Searching

This sub-algorithm searches neighbors of a vehicle in the local lane and adjacent lanes. The search key value of vehicle *i* is k_i, the lane of vehicle *i* is y_i. In LLB+-tree, data points are stored only at leaf nodes. Thus, we use a function $GetLeafNode()$ to implement the searching process from root node to the objective leaf node. The sub-algorithm includes two NN queries: the NN queries in the adjacent lane $Q(Adjacent)$ and in the local lane $Q(Local)$. Neighbors in the local lanes indicate the nearest leading vehicle $PreL$ and the following vehicle $NextL$. $PreA$ and $NextA$ are respectively the nearest leading and following vehicles in the adjacent lane.

The sub-algorithm mainly contains following two steps: 1) It searches the leaf node and find the neighbors of a vehicle in the local lanes; 2) It searches related leaf nodes and find the neighbors of a vehicle in the adjacent lane if needed. The pseudo-code of the algorithm is shown in Algorithm 1.

Algorithm 1: Searching for the neighbors of vehicle i in the local and adjacent lanes with K_i, using LLB$^+$-tree

input : K_i, y_i, $Q(Adjacent)$ or $Q(Local)$.
output: $PreA$ and $NextA$; $PreL$ and $NextL$.
$node.n \leftarrow GetLeafNode()$;
while $ture$ **do**
  $\alpha \leftarrow 0$;
  for $Q(Local)$ **do**
    if $K_j = K_i$ **then**
      $PriorL \leftarrow PrePr_j$; $NextL \leftarrow NextPr_j$;
    else
      vehicle not exist;
  for $Q(Adjacent)$ **do**
    search $\forall$ K_l for $y_l = y_i$ in node $n + (-1)^{\alpha+1}\alpha$;
    if $ture$ **then** find $\forall$ K_m where $y_m = y_i$;
    if $|K_m - K_i| < |K_l - K_i|$ *and* $K_i > K_m, K_l$ **then**
    $PriorA \leftarrow PrePr_m$; $NextA \leftarrow Pr_m$;
    else if $|K_m - K_i| < |K_l - K_i|$ *and* $K_i \leq K_m, K_l$ **then**
      $PriorA \leftarrow Pr_m$; $NextA \leftarrow NextPr_m$;
    else if $\nexists$ *node* $n + (-1)^{\alpha+1}\alpha$ **then**
      $PriorA \leftarrow NULL$; $NextA \leftarrow NULL$;
    else
      $\alpha++$;

4.2. Inserting

Algorithm 2: Inserting a Record with K_i in a LLB$^+$-tree

input : K_i, y_i, $Q(Update)$.
$node.n \leftarrow GetLeafNode()$;
while $ture$ *and* $Q(Update)$ **do**
  $\beta \leftarrow 0$;
  create entry $(K_i, Pr_i, PrePr_i, NextPr_i)$;
  for $Q(Update)$ **do**
    search $\forall$ K_l for $y_l = y_i$ in node $n + (-1)^{\beta+1}\beta$;
    if $ture$ **then**
    find $\forall$ K_m where $y_m = y_i$;
    if $|K_m - K_i| < |K_l - K_i|$ *and* $K_i > K_m, K_l$ **then**
    $PrePr_i \leftarrow PrePr_m$; $NextPr_i \leftarrow Pr_m$;
    $PrevPr_m \leftarrow Pr_i$; $Next.PrePr_i \leftarrow Pr_i$;
    else if $|K_m - K_i| < |K_l - K_i|$ *and* $K_i < K_m, K_l$ **then**
      $PrePr_i \leftarrow Pr_m$; $NextPr_i \leftarrow NextPr_m$;
      $NextPr_m \leftarrow Pr_i$; $Pre.NextPr_i \leftarrow Pr_i$;
    else if $\nexists$ *node* $n + (-1)^{\beta+1}\beta$ **then**
      $PrePr_i \leftarrow$ NULL; $NextPr_i \leftarrow$ NULL;
    else
      $\beta++$;
    if *leaf node* n *is not full* **then** insert entry;
    else
      $node.n \leftarrow GetInternalNode()$;
      if *node* n *is not full* **then** insert (K_j, P_j);
      else
        split nodes till no overflow nodes;

This sub-algorithm illustrates the procedure for inserting a record (vehicle) with a search field value k_i in LLB+-tree. Algorithm 2 gives a detailed pseudo-code of the subalgorithm. We denote $Pre.NextPr_i$ and $Next.PrePr_i$ as the pointers $PreP$ and $NextP$ of the nearest following and leading vehicles of i, respectively.
We can further divide this sub-algorithm into two parts: 1) It create entry of vehicle *i* in correct leaf node and updates relative pointers; 2) It updates internal nodes to maintain the right structure of LLB+-tree. When a node is full it will split and when the parent node also be full, the splitting can propagate all way up to create a new level for LLB+-tree.

4.3. Deleting

This algorithm illustrates deleting a record with a search field value k_i from a LLB+-tree. When deleting an entry, we should always remove it from the leaf level. If the entry is in an internal node, we must also remove it from there.

The algorithm contains three parts: 1) It searches internal nodes recursively to find the path according to the search field value *Ki*. When the search field value occur in an internal node, we use a left (or right) entry to replace it; 2) It deletes the entry of the vehicle in correct leaf node and updates relative pointers; 3) It updates the leaf node by merging and redistributing sibling nodes when there exists node underflow; 4) When the merge and redistribute in leaf nodes leads to an underflow of a internal node, the internal node will also merge and redistribute to maintain the structure of the LLB+-tree. We give the pseudo-code of the algorithm in Algorithm 3.

Algorithm 3: Deleting a Record with K_i in a LLB$^+$-tree

```
input: K_i, Q(Delete).
node.n ← GetLeafNode();
while ture and Q(Delete) do
    delete entry (K_i, Pr_i, PrePr_i, NextPr_i);
     Next.PrePr_i ← NextPr_i;
    Pre.NextPr_i ← PrePr_i;
     if node is not underflow then finished;
      else
       redistribute node n with n + 1 or n − 1;
    node.n ← GetInternalNode();
    if internal node n is not underflow then
        finished; else
            redistribute internal nodes;
```

5. Parameters Optimization

In this section, we firstly analyze those parameters that effect on the hit rate of NN queries. We then analyze the time cost of LLB+-tree based method and build a mathematical model to optimize the node size of the LLB+-tree.

5.1. Hit Rate Analysis

In a LLB+-tree, data pointers are stored in leaf nodes. For better understanding, we call vehicles in a leaf node as a “platoon”. Under the assumption of randomly distribution of vehicles, the performance, the expect query length ϵ of a NN query is determined by the hit rate of a query *P* in a platoon. Further, the hit rate *P* is influenced by the average amount of vehicles in a platoon *q*. Thus, we compute the optimized value of *q* to minimize the expect query length ϵ of a NN query.

To calculate *P*, we assume that there are *N* vehicles randomly distributed in *L* lanes. We can use a *q*×*L* matrix *A* to describe possible distributions of *q* vehicles. Each element in the matrix is a possible position for a vehicle. In example, a_{ij} is the $_i$th position in the $_j$th lane.

$$\mathbf{A} = \begin{pmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1L} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{iL} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{q1} & \dots & a_{qj} & \dots & a_{qL} \end{pmatrix}$$

When we query neighbors of vehicle a_{ij} in adjacent lanes, there exist two different situations while the road has at least 3 lanes.

The first situation is, vehicle a_{ij} is in an edge lane of a road (the lane $i = 1$ or $j = L$ when $L \geq 2$). It needs to query neighbors in the only adjacent lane, that are vehicles in $[a_{12}, \cdots, a_{q2}]$ or $[a_{1(L-1)}, \cdots, a_{q(L-1)}]$.

In this case, we calculate P_A, the possibility of finding a neighbor in the adjacent lane of vehicle a_{ij} in the platoon. We can compute the possible distributions of all other $q - 1$ vehicles (except vehicle a_{ij} in the edge lane) that are not distributed in the adjacent lane: $C_{q \cdot L-(q+1)}^{q-1}$. The total possible distributions of $q-1$ vehicles is $C_{q \cdot L-1}^{q-1}$. Thus, we can calculate P_A using the following formula:

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

The second situation is, vehicle *aij* is in a mid lane (the lane $1 < j < L$ when $L > 2$). In this case, vehicle a_{ij} has both left and right adjacent lanes. Therefore, we need to query neighbors in two adjacent lanes, that are vehicles in $[a_{1(j-1)}, \cdots, a_{q(j-1)}]$ and $[a_{1(j+1)}, \cdots, a_{q(j+1)}]$.

We denote P_B as the probability of finding neighbors of vehicle a_{ij} in both adjacent lanes. To calculate P_B, we also define P'_B, the probability of all other $q - 1$ vehicles that are not distributed in those adjacent lanes. When we do not consider distributions of vehicles in one adjacent lane, The distributions of vehicles exist in another adjacent lane is: $C_{q \cdot L-(q+1)}^{q-1}$. According to principle of inclusion-exclusion, the general form of which is shown as follow:

$$\left| \bigcup_{i=1}^{q} A_i \right| = \sum_{k=1}^{q} (-1)^{k+1} \left(\sum_{1 \leq i_1 < \cdots < i_k \leq q} |A_{i_1} \cap \cdots \cap A_{i_k}| \right)$$

P'_B can be given by excluding the overlap distributions $C_{q \cdot L-(2q+1)}^{q-1}$:

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

Thus, we can compute *PB* as follow:

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

In general, we can conclude the hit rate *P* of a NN query in adjacent lanes in 3 cases: 1) When the road has only 1 lane, vehicles don't have to query neighbors in adjacent lanes. 2) When the road has 2 lanes, vehicles only have to query neighbors in one adjacent lane. 3)

When the road has more than 2 lanes, we need to consider two situations: vehicles in edge lanes and in mid lanes. We list the formula of *P* as follow:

$$P = \begin{cases} 1 & \text{if } L = 1 \\ P_A & \text{if } L - 2 \\ \frac{2P_A + (L-2)P_B}{L} & \text{if } L \geq 3 \end{cases}$$

We also summarise the relationship among the hit rate *P*, average number of vehicles in a platoon *q* and number of lanes *L*, which is shown in Figure 5. The observation shows that in common roads conditions, a road with L lanes $1 < L \leq 10$, The rate of convergence of *P* is diminishing with the increasing of *L*.

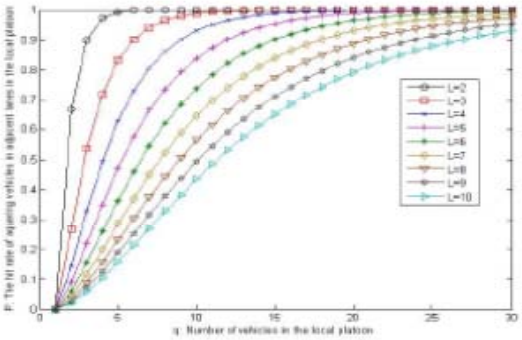

Figure 5. The relationship among *L*, *q* and *P*

5.2. Time Cost Analysis

Being a variation of B+-tree, we can find vehicle *i* in LLB+-tree spending $O(\log_z N)$ time, where *N* is the number of vehicles in the road; *z* is the minimize number of children in each internal node.

When querying neighbors of vehicles *i* in the local lane, we can directly obtain them by pointers $PrePr_i$ and $NextPr_i$. That is, we can find neighbors of vehicle *i* in the local lane also in $O(\log_z N)$ time.

When querying neighbors of vehicle *i* in the adjacent lanes, we use the expect query length ϵ to measure the efficiency of the query. Such a query search neighbors by traversing all vehicles in each platoon. The expect query length of finding a vehicle in a platoon by traversing is $\frac{q}{2}$. The expect query length of finding the object neighbor in the local platoon is: $P \cdot \frac{q}{2}$ and in the next platoon is: $\frac{q}{2} + (1-P)P \cdot \frac{q}{2}$. We can summarize the expect query length ϵ of finding the object vehicle in the *k*th platoon is:

$$\frac{k-1}{2}q + (1-P)^{k-1}P \cdot \frac{q}{2}$$

The number of leaf nodes (platoons) in a LLB+-tree can be express by $\lceil \frac{N}{q} \rceil$. We can describe the expect query length ϵ of finding neighbors in all platoon as follow:

$$\epsilon = \sum_{k=1}^{\lceil \frac{N}{q} \rceil} (1-P)^{k-1} P \cdot \frac{kq}{2}$$

We can simplify the formula using following methods:

$$\epsilon = \frac{\epsilon - (1-P)\epsilon}{P}$$
$$= \sum_{k=1}^{\lceil \frac{N}{q} \rceil} (1-P)^{k-1} \cdot \frac{kq}{2} - \sum_{k=1}^{\lceil \frac{N}{q} \rceil} (1-P)^{k} \cdot \frac{kq}{2}$$
$$= \frac{q}{2} \cdot (\sum_{k=0}^{\lceil \frac{N}{q} \rceil - 1} (1-P)^{k} - (1-P)^{\lceil \frac{N}{q} \rceil} \cdot \lceil \frac{N}{q} \rceil)$$

By using geometric series, we can find:

$$\sum_{k=0}^{\lceil \frac{N}{q} \rceil - 1} (1-P)^{k} = \frac{1-(1-P)^{\lceil \frac{N}{q} \rceil}}{P}$$

Thus, the expect query length ϵ can be simplified as follow:

$$\epsilon = \frac{q}{2} \cdot (\frac{1-(1-P)^{\lceil \frac{N}{q} \rceil}}{P} - (1-P)^{\lceil \frac{N}{q} \rceil} \cdot \lceil \frac{N}{q} \rceil)$$

For a certain road, *L* is a constant, and *P* is rapid convergence to 1 with the increasing of the number of vehicles in the platoon *q*. For a certain *L*, we can get an acceptable *P* with limited *q*. Therefore, *q* is also considered as a constant in this case. Thus, the limitation $\lim_{N\to\infty}(1-P)^{\lceil \frac{N}{q} \rceil} = 0.$, and we can also calculate the limitation of ϵ:

$$\lim_{N\to\infty} \epsilon = \lim_{N\to\infty} \frac{q}{2} \cdot (\frac{1-(1-P)^{\lceil \frac{N}{q} \rceil}}{P} - (1-P)^{\lceil \frac{N}{q} \rceil} \cdot \lceil \frac{N}{q} \rceil) = \frac{q}{2P}$$

As a result, we can also find the neighbor of vehicle i in the adjacent lane in $\log_z N$ time, under the assumption of randomly distribution of vehicles. The relationship among the number of lanes *L*, expect (average) number of vehicles in one platoon *q* and the expect query length of finding neighbors in the adjacent lanes ϵ with an enough large *N* (we adapt *N* = 1000 in this case) is shown in Figure 6. The curve called skyline is a line connecting every points, the minimum value of ϵ of all curves, shows the optimal choice and the variation trend of *q* with the increasing number of lanes *L*.

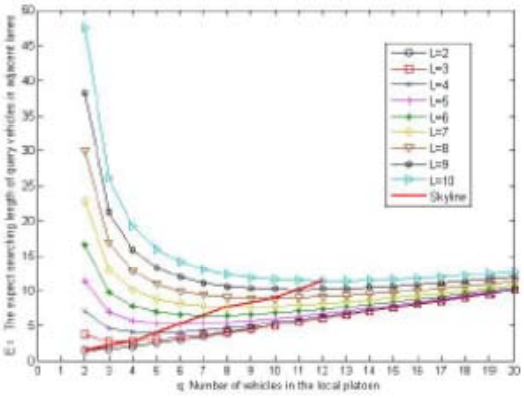

Figure 6. The relationship among *L*, *q* and ϵ

We also analyze the impact of the number of vehicles *N*, our research shows that there exists threshold values of *N* for a certain pair of amount of lanes *L* and average amount of vehicles in a platoon *q*. When *N* is larger than the threshold value, the expect query length ϵ for

NN queries in the adjacent lanes are converging to fixed values. Otherwise, the query length is decreasing with the lower amount of *N*. That is, the threshold line divide *N* into two intervals. The upper interval indicates that the query can be responded in constant time no matter how large the *N* is. While the lower interval shows that the query can be responded much quicker when *N* is smaller than the threshold value. The relationship among *N*, *L* and *ϵ* with the corresponding optimal *q* is shown in Figure 7. The value of thresholds of *N* and the skyline of *q* for each *L* is shown in Table 1. Note that the expect query length *ϵ* in Table 1 do not contain the query length from the root node to the leaf node.

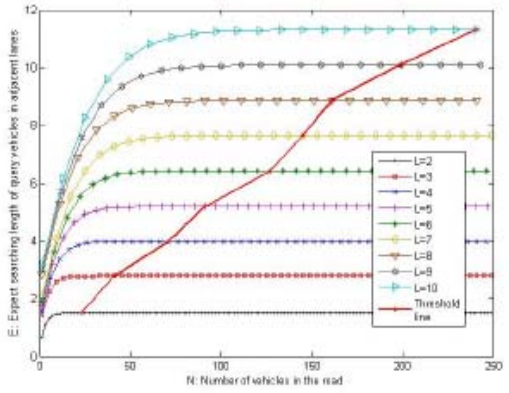

Figure 7. The relationship among *L*, *N* and *ϵ*

Table 1. The Threshold Of N And Skyline Of Q For Minimum ϵ

Number of lanes: L	Optimal number of vehicles in one platoon: q	Expect searing length for a given vehicle x: ϵ	Threshold value of N
2	2	1.5000	23
3	4	2.7809	41
4	5	3.9754	71
5	6	5.1911	91
6	7	6.4176	127
7	8	7.6523	145
8	10	8.8666	161
9	11	10.0884	199
10	12	11.3156	241

5.3. Node Size Optimization

In order to reduce the frequency of the spliting/merging process and maintain the time complexity of the LLB+-tree method, we need to optimize another important parameter of LLB+-tree: the minimize number of children (subtrees) in internal nodes *z*. It is important because *z* determines the value range of *q*. In particular, the value of *q* must larger than the lower bound of the leaf node's size $z-1$ and lower than the upper bound $2z-1$ according to the rule of B+-tree based method.

For each optimized *q*, we need to calculate the optimal value of the corresponding *z*. We can find that for each *q*, there exist multi possible values of *z*. Thus, we need to calculate the optimal value of the corresponding *z* for each optimized *q*. Here we compare the expect query length *ϵ* for each possible *z* using following method:

For a certain *q*, we can find the value range of the corresponding z_i: $\frac{q+1}{2} \leq z_i \leq q+1,\ i \in [1, \lceil \frac{q+1}{2} \rceil + 1]$; For each *zi*, we can find the value range of the possible q_{ij}: $z_i - 1 \leq q_{ij} \leq 2z_i - 1,\ j \in [1, z_i + 1]$. Thus, we can calculate the expect query length of q_{ij} , denoted as $\epsilon(q_{ij})$. By comparing *ϵi*, the average value of the sum of $\epsilon(q_{ij})$ for each *zi*, we can compute out the optimized *zi*, whose *ϵi* has the minimized value. The formula is shown as follow:

$$\epsilon_i = \frac{1}{z_i+1}\sum_{j=1}^{z_i+1}\epsilon(q_{ij})$$
$$= \frac{1}{z_i+1}\sum_{j=1}^{z_i+1}(\sum_{k=1}^{\lceil \frac{N}{q_{ij}} \rceil}(1-P)^{k-1}P\cdot\frac{kq_{ij}}{2}+\log_{z_i}N)$$

The optimal *z* with minimum *ϵ* when *N* = 250 and *N* = 1000 is shown in Table 2. We can see that the the optimal value of *z* is only a little influenced by *N*, and the expect query length *ϵ* as a minor increasing with the increasing of *N*, which shows the strong scalability of LLB+-tree. We also find that the lower the number of *L*, the more obvious the impact of *N*. Nevertheless, there exist load limitations of each road, which is not mentioned in our study. That is, the extreme case that a astronomical number of vehicles congest in a limit *L* road will never happen.

Table 2. The Optimal Z with Minimum ϵ when N = 250 and N = 1000

Number of lanes: L	Optimal value of z: $N = 250$	Expect searing length ϵ: $N = 250$	Optimal value of z: $N = 1000$	Expect searing length ϵ: $N = 1000$
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
6	7	9.6699	7	10.3823
7	8	10.7364	8	11.4031
8	8	11.8136	9	12.4763
9	9	12.9240	9	13.5550
10	10	14.0633	10	14.6654

6. Performance Experiments

In this section, we process simulation experiments to evaluate the performance of Original B+-tree, Linked list and LLB+-tree.

6.1. Experiments Setting

Our experiment is based on a simulation using the backbone network of a section of Chengdu city with 22 roads, the network of which is shown in Figure 8. To facilitate experiments, the simulation network is set to be closed, which means the amount of vehicles is fixed. In our simulation, vehicles have only 4 operations in a simulation cycle: 1) The NN query in the local lane. 2) The NN query in the objective adjacent lane. 3) The operation of leaving a lane. 4) The operation of joining in a lane (the leave and join operations are used both in lanechanging and road-switching process).

(a) Objective section (b) Simulation network

Figure 8. The objective section and corresponding simulation network

There exist three uncorrelated variable parameters in our experiments: the number of vehicles in each road *N*; the number of lanes *L* and the lane-changing rate P_c. Note that P_c does not refers to the rate of successful lane-changes or lane-changing processes, P_c means the average possibility of a vehicle query neighbors in the objective adjacent lane in one simulation cycle. In this paper, we consider the average time cost (response time) of the simulation of each vehicle in one cycle ($T_{response}$) as an indicator to evaluate the performance of different methods. T_i denotes the simulation time cost of each vehicle *i*, and $T_{response}$ is the time cost in average. Thus, the formula of *Tresponse* is described as follow:

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

To reduce the error of the simulation, we adopt the average $T_{response}$ from 100 simulation results. The time cost $T_{response}$ is affected by *N*, *L* and P_c, thus we can describe it in this form: $T_{response} = f(P_c,N,L)$. Due to the difficulty of analyzing the variation of *Tresponse* through three variable parameters simultaneously, we adopt two sets of empirical values (a lower and an upper set of values according to common traffic conditions) for these three parameters: the lane-changing rate $P_c \in \{30\%, 60\%\}$; the number of lanes $L \in \{3, 6\}$, the number of vehicles in each road $N \in \{250, 1000\}$ (both larger than corresponding threshold values). To facilitate experiments, we assume that the number of vehicles in each road are all *N*.

In our experiments, the simulation platform is a self developed microscopic simulation system called DMTSS. The car-following model we adapted is Pipes model and the platform is running on a Acer Veriton D430 computer with i3 CPU 3.40GHZ and 4GB DDR3 SDRAM.

6.2. Contrast Experiments

In this part, we plan three sets of experiments using relative empirical high and low parameters to evaluate the performance of three methods. 1) Linked list: a typical linear index structure widely adopted in simulation systems. 2) Original B+-tree: B+-tree with bidirectional sorting in leaf nodes. 3) LLB+-tree: a variation of B+-tree proposed in this paper for multi 1-dimensional cases with optimal parameters.

1) The impact of lane change rate: The lane-changing rate P_c, the possibility of a vehicle querying neighbors in adjacent lanes in one simulation cycle, is an important parameter in traffic simulation systems. This set of experiments is to show the performance of three methods with the increasing of P_c using different combinations of empirical high and low parameters $N \in \{250, 1000\}$ and $L \in \{3, 6\}$. The contrast experiments of the impact of *Pc* are shown in Figure 9, 10, 11 and 12.

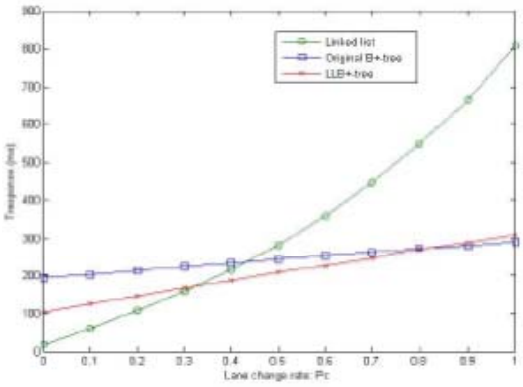

Figure 9. $T_{response}$ with N = 250, L = 3 and variable P_c

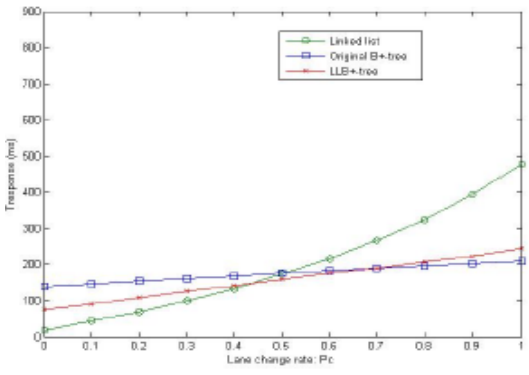

Figure 10. $T^{response}$ with N = 250, L = 6 and variable P_c

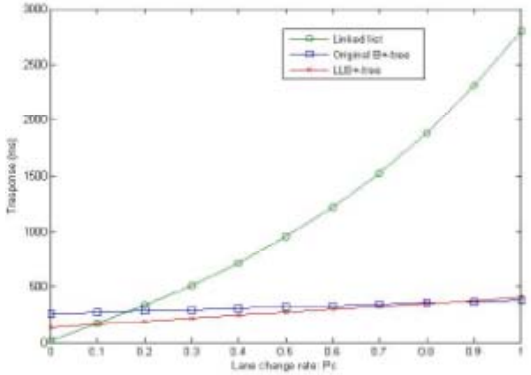

Figure 11. $T_{response}$ with N = 1000, L = 3 and variable P_c

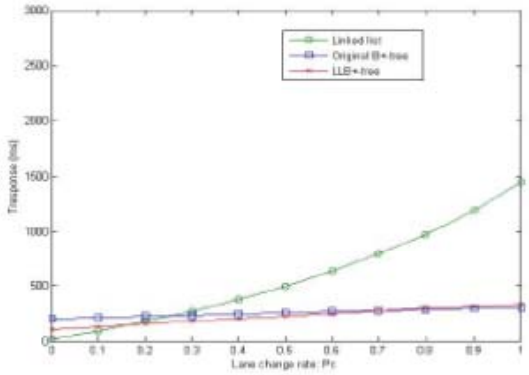

Figure 12. $T_{response}$ with N = 1000, L = 6 and variable P_c

Figure 9 shows the response time $T_{response}$ with low N and low L. In this condition, Linked list has a well performance when the lane-changing rate $P_c \leq 30\%$. When P_c reaches to 50%, the performance of Linked list is getting worse compared with other methods. The performance of LLB+-tree is better than that of Original B+-tree when $Pc \leq 80\%$, after that the *Tresponse* of Original B+-tree is better. Figure 10 shows $T_{response}$ with the low N and high L. In this condition, the advantage of Linked list is more distinctly: the lower P_c, the better performance of Linked list. Besides, the performance of Original B+-tree and that of LLB+-tree are less effected by the increasing P_c, and the performance of LLB+-tree is better than that of Original B+-tree when $P_c \leq 70\%$. Figure 11 shows the *Tresponse* with the high N and low L. Compared with Figure 9, we can see that with the same L, the increasing Pc leads to the worse performance of Linked list compared with that of other two methods. Figure 12 shows $T_{response}$ with the high N and high L. With the same N in Figure 11, Linked list is getting better in high L, because vehicles in each lane are stored in one list, the higher L, the less number of vehicles in one list.

The result of this set of experiments shows that in most conditions of P_c, the performance of LLB+-tree is better than that of Original B+-tree. Only in some very high P_c cases, Original B+-tree is better. Besides, the efficiency of Linked list in some low P_c cases is unsurpassable.

2) The impact of the number of vehicles: We evaluate the performance of three methods with the variation of the number of vehicles N. In a simulation, the number of vehicles N is limited by the length and the number of lanes L of the road. We use similar method to analyze the impact of the number of vehicles N using the combination of the high and low parameters P_c and L. The contrast experiments of the impact of N are shown in Figure 13, 14, 15, and 16.

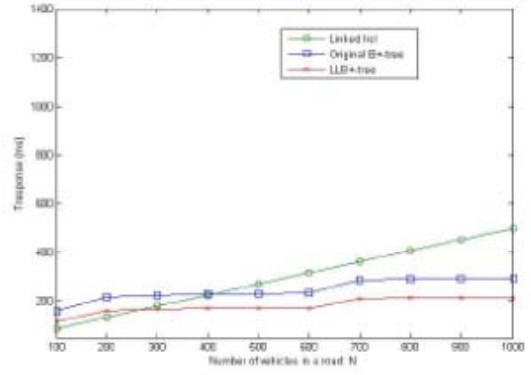

Figure 13. $T_{response}$ with P_c = 30, L = 3 and variable N

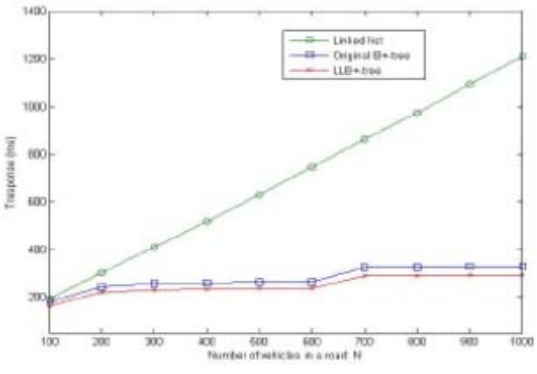

Figure 14. $T_{response}$ with P_c = 60, L = 3 and variable N

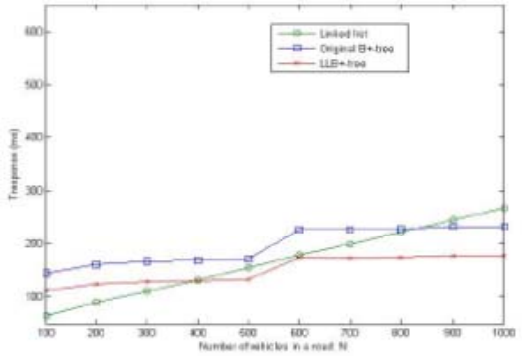

Figure 15. $T_{response}$ with P_c = 30, L = 6 and variable N

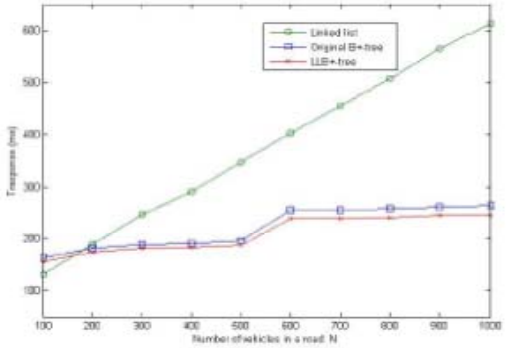

Figure 16. $T_{response}$ with P_c = 60, L = 6 and variable N

Figure 13 shows the $T_{response}$ with the low Pc and low L. We can see that the performance of LLB+-tree is completely better than that of Original B+-tree in this condition. Besides, for majority values of N, LLB+-tree is the most efficient method. Figure 14 shows the $T_{response}$ with the high P_c and low L. Linked list in this case has no superiority compared with other methods, while LLB+-tree is the best method with these parameters. Figure 15 shows the $T_{response}$ with the low P_c and high L. When $N \geq 400$, LLB+-tree is better than Linked list, while Original B+-tree is only better than Linked list when $N \geq 800$. Figure 16 shows the $T_{response}$ with the high P_c and high L. Linked list also depicts the defect of scalability with the increasing N.

The result of experiments on the impact of N shows that the $T_{response}$ of Linked list is a linear growth with the increasing N, while other two methods shows better scalability. With these parameters, LLB+-tree is completely better than Original B+-tree and it is also better than Linked list in most cases.

3) The impact of the number of lanes: In this set of experiments, we try to evaluate the impact of the number of lanes L to $T_{response}$ with the high and low parameters P_c and N. The results are shown in Figure 17, 18, 19 and 20.

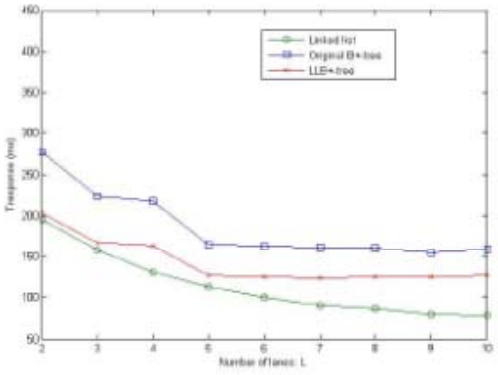

Figure 17. $T_{response}$ with P_c = 30, N = 250 and variable L

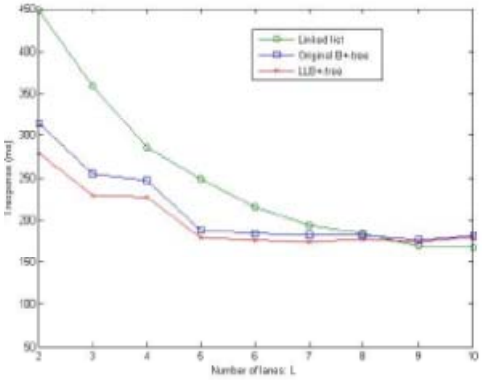

Figure 18. $T_{response}$ with P_c = 60, N = 250 and variable L

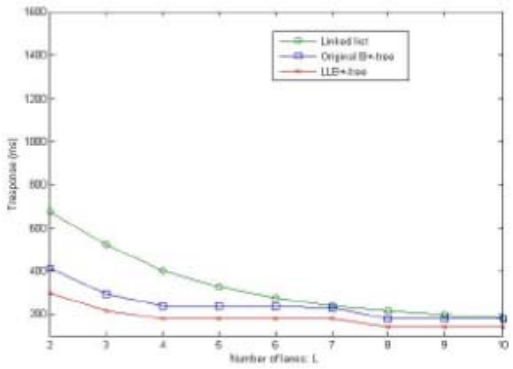

Figure 19. $T_{response}$ with Pc = 30, N = 1000 and variable L

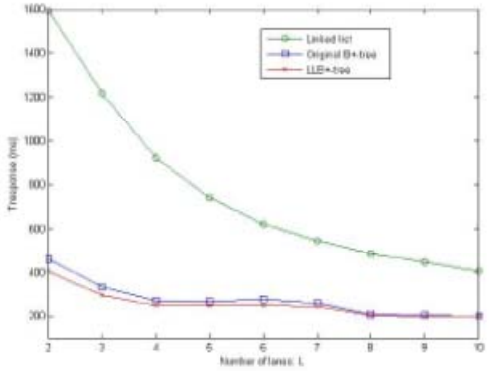

Figure 20. d$T_{response}$ with P_c = 60, N = 1000 and variable L

Figure 17 depict the $T_{response}$ with the low P_c and low N. With these parameters, Linked list is no doubt the best method. Meanwhile, the performance of LLB+-tree is better than that of Original B+-tree. Figure 18 depict the *Tresponse* with the high P_c and low N. With the increasing of L, LLB+-tree and Original B+-tree are gradually approaching to each other, the performance of which are better than that of Linked list when $L \leq 8$. Figure 19 shows the $T_{response}$ with the low P_c and high N. In this case, LLB+-tree has better performance than all other methods. Figure 20 shows the $T_{response}$ with the high P_c and high N. The performance of LLB+-tree and Original B+-tree are gradually approaching to each other with the increasing of L just like that in Figure 18. We conclude the impact of L as follow: there exist cases (low P_c and low N) that fit Linked list most; LLB+-tree generally "controls" Original B+-tree. While in high P_c cases the performance of LLB+-tree is approaching to that of Original B+-tree with the increase of L.

4) Evaluations: Although the contrast experiments of three methods show that LLB+-tree is efficient in most traffic conditions, we evaluate their performance using statistical analysis of simulation data. Such simulation data contain three sets of parameters separately describe congested, normal and sparse traffic conditions, the parameters setting of which are shown in Table 3. For each combination of parameters in a same traffic condition, we take 100 results into account. The basic statistical information of totally 24,300 samples from congested, normal and sparse simulation data are shown in Table 4.

Table 3. Parameters Setting for Common Traffic Conditions

Conditions	Lane change rate: $P_c \in$	Number of vehicle: $N \in$	Number of lanes: $L \in$
Congested	{70%,80%,90%}	{800,900,1000}	{3,4,5}
Normal	{40%,50%,60%}	{500,600,700}	{3,4,5}
Sparse	{10%,20%,30%}	{200,300,400}	{3,4,5}

Table 4. The Basic Statistical Information of Simulation Data

Indicator	Linked list	Original B^+-tree	LLB^+-tree
Sample Size	8100	8100	8100
Avg $T_{response}$	630.8	259.1	226.1
Variance	311545.4	3189.9	5393.1
min $T_{response}$	41	147	90
max $T_{response}$	2245	375	381

The average $T_{response}$, the main indicator to evaluate the performance of different methods, shows that LLB+-tree outperforms Original B+-tree by 12.8%, and the performance of LLB+-tree is increased by 64.2% compared with that of Linked list. The variances indicate that Original B+-tree is more stable than LLB+-tree while the difference is not that obviously. The

result shows that LLB+-tree is better than Original B+-tree when there has not demand strict the range of $T_{response}$.

For better understanding the performance of these methods, we analyze the total possible frequency distributions of $T_{response}$ of different methods from simulation data with all possible combination of parameters: $N \in${100, 200, 300, 400, 500, 600, 700, 800, 900, 1000}, $P_c \in${0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%} and $L \in${2, 3, 4, 5, 6, 7, 8, 9, 10}. The distribution of Linked list is shown in Figure 21 with Original B+-tree shown in Figure 22 and LLB+-tree shown in Figure 23. Note that in this case we haven't consider the weight (possibility) of each combination of parameters that would occur in real cases. The figure is only to display the range of possible $T_{response}$ for different methods (e.g., a condition with N = 100, L = 10 and Pc = 100% is extremely infrequent in real cases).

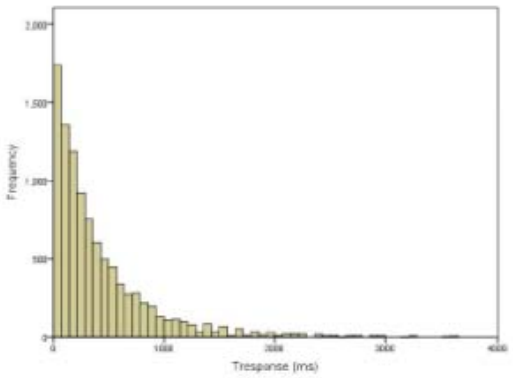

Figure 21. Range of $T_{response}$ with Linked list

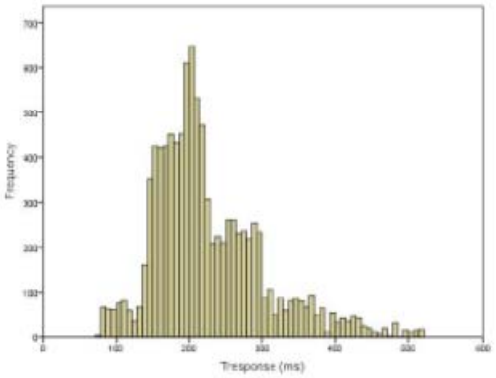

Figure 22. Range of $T_{response}$ with Original B+-tree

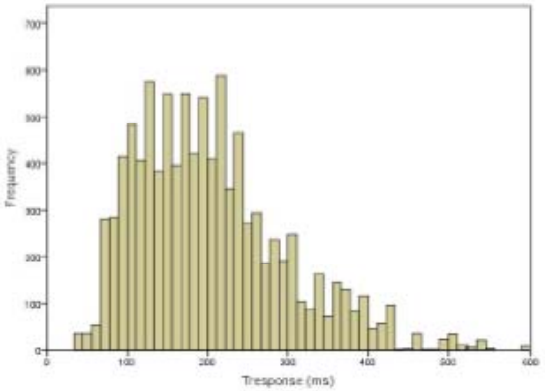

Figure 23. Range of $T_{response}$ with LLB+-tree

We can see that the distribution of $T_{response}$ with Linked list is scattered from 1ms to 3643ms, while Original B+-tree scattered from 71ms to 521ms, LLB+-tree scattered from 33ms to 597ms.

In collusion, although Linked list-based methods are suitable in some cases (e.g., in some sparse traffic conditions), the distribution of $T_{response}$ prove that they are not suitable for large-scale simulations. According to simulation experiments and statistical analysis, the result shows that LLB+-tree is more suitable in most traffic conditions.

6. Conclusion

In this paper, we design and implement the LLB+-tree for NN queries in traffic simulation systems. Such a LLB+-tree is suitable to multi 1-dimensional and highly concurrent NN queries. We then propose three sub-algorithms to manage a LLB+-tree and build mathematical models to optimize parameters settings. We also propose a theoretical analysis to estimate the time complexity of the method. The result of simulation experiments and statistical analysis show that LLB+-tree is efficient for NN queries in traffic simulation systems. In particular, LLB+-tree can save 64.2% and 12.8% response time respectively compared with Linked list and Original B+-tree.

References

[1] Kollios G, Gunopulos D, Tsotras VJ. Nearest neighbor queries in a mobile environment. Spatio-Temporal Database Management. Springer Berlin Heidelberg, 1999: 119-134.
[2] Hjaltason GR, Samet H. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems (TODS)*. 2003; 28(4): 517-580.
[3] Roussopoulos N, Kelley S, Vincent F. Nearest neighbor queries. *ACM sigmod record. ACM.* 1995; 24(2): 71-79.
[4] Katayama N, Satoh S. The SR-tree: An index structure for high dimensional nearest neighbor queries. *ACM SIGMOD Record. ACM.* 1997; 26(2): 369-380.
[5] White DA, Jain R. *Similarity indexing with the SS-tree. Data Engineering.* Proceedings of the Twelfth International Conference on. IEEE. 1996: 516-523.
[6] Hu H, Lee DL. Range nearest-neighbor query. Knowledge and Data Engineering. *IEEE Transactions on*. 2006; 18(1): 78-91.
[7] Jagadish HV, Ooi BC, Tan KL, et al. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS).* 2005; 30(2): 364-397.
[8] Arya S, Mount DM. Approximate nearest neighbor queries in fixed dimensions. Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms. *Society for Industrial and Applied Mathematics*. 1993: 271-280.
[9] Hu L, Jing Y, Ku WS, et al. *Enforcing k nearest neighbor query integrity on road networks*. Proceedings of the 20th International Conference on Advances in Geographic Information Systems. ACM, 2012: 422-425.
[10] Seidl T, Kriegel H P. Optimal multi-step k-nearest neighbour search[C]//ACM SIGMOD Record. ACM, 1998; 27(2): 154-165.
[11] Benetis R, Jensen CS, Karciauskas G, et al. *Nearest neighbor and reverse nearest neighbor queries for moving objects. Database Engineering and Applications Symposium.* Proceedings. International. IEEE. 2002: 44-53.
[12] Jensen CS, Lin D, Ooi BC. *Query and update efficient B+-tree based indexing of moving objects.* Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004: 768-779.
[13] Tao Y, Papadias D, Shen Q. Continuous nearest neighbour search[C]//Proceedings of the 28th international conference on Very Large Data Bases. VLDB Endowment. 2002: 287-298.
[14] Lee KCK, Leong HV, Zhou J, et al. *An efficient algorithm for predictive continuous nearest neighbor query processing and result maintenance*. Proceedings of the 6th international conference on Mobile data management. ACM, 2005: 178-182.
[15] Xie X, Yiu M L, Cheng R, et al. Trajectory Possible Nearest Neighbor Queries over Imprecise Location Data. 2012.
[16] Cameron GDB, Duncan GID. PARAMICS‐Parallel microscopic simulation of road traffic. *The Journal of Supercomputing*. 1996; 10(1): 25-53.
[17] Fellendorf M. *VISSIM: A microscopic simulation tool to evaluate actuated signal control including bus priority.* 64th Institute of Transportation Engineers Annual Meeting. 1994: 1-9.
[18] Yang Q, Koutsopoulos HN. A microscopic traffic simulator for evaluation of dynamic traffic management systems. *Transportation Research Part C: Emerging Technologies*. 1996; 4(3): 113-129.
[19] Krajzewicz D, Bonert M, Wagner P. The open source traffic simulation package SUMO. *RoboCup 2006 Infrastructure Simulation Competition.* 2006; 1: 1-5.
[20] Elmasri R. Fundamentals of database system. Pearson Education India, 2008.
[21] Comer D. Ubiquitous B-tree. ACM Computing Surveys (CSUR). 1979; 11(2): 121-137.