交差点での競合車両検索のための車両情報ストリームと高精度道路地図 DB の統合索引

井川 元 [1]　渡辺 陽介 [2]　高田 広章 [3]

近年，自動運転に関する研究が進められている．自動運転車の安全な走行のためには自車両周辺の正確な情報が必要である．自動運転車は高度なセンサを搭載し，ネットワークを通じた情報共有を行う．情報共有の基盤として，高精度な 3 次元地図に車両などの動的な位置情報を組み合わせた「ダイナミックマップ」が提案されている．従来の道路地図における検索要求としては，道路地図上のオブジェクト（コンビニなど）に対する kNN 検索や範囲検索などが存在する．しかし，ダイナミックマップ上では，近辺の危険車両の検索や交差点で自車両と競合する可能性のある車両の検索といった新たな検索要求が現れると考えられる．そこで，本研究では，交差点で自車両と競合する可能性のあるレーンにいる車両の検索を行うシステムを構築し，検索を高速に行うための索引技術を提案する．本研究で提案する索引は，静的情報である道路地図に対する索引と動的情報である車両に対する索引の 2 つで構成される．評価実験では，実データおよび仮想データ（マンハッタンモデル）に対して，索引の構築時間および検索時間を測定する．

1　はじめに

近年，自動運転の研究が注目を浴びている [1]．自動運転が安全に運用されるためには自車両周辺の正確な情報が必要である．そのため自動運転車は高度なセンサを搭載して周辺の情報を取得する．自動運転車のセンサの 1 つとして挙げられるのが LiDAR である．LiDAR は，レーザ光を用いて点群データと呼ばれる 2 次元又は 3 次元の点データを取得する．しかし，自車両の車載センサのみではカバーできる範囲に限界がある．例えば，Velodyne LiDAR 社の VLP-16[2] は，測定距離が約 100m となっている．そのため，仮に時速 60km で車両が走行していた場合，約 6 秒先までかつレーザ光を遮らない範囲しか物体の検知をすることができない．よって，車両同士や道路インフラとのネットワークを通じた情報共有（セルラ V2X[3]，DSRC[4] 等）により視野を補う必要がある．また，センサ情報の共有のみでは不十分で地図との重ね合わせも重要である．例えば，進行方向の先からこちらに向かっている車が存在する時，センサ情報の共有のみではその車両が逆走車なのかそれとも違うレーンを走行しているのかは分からない．しかし，地図データを組み合わせることで，その判断を行うことが可能になる．そこで，センサ情報の共有と地図との重ね合わせの基盤として考えられているのがダイナミックマップである [5][6]．ダイナミックマップは静的情報である高精度 3 次元地図に，渋滞情報や事故による交通規制，車両の位置情報などの動的情報を組み合わせたもの（図 1）となっており，内閣府の戦略的イノベーション創造プログラム・自動走行システム（SIP-adus）[7] における重要分野の 1 つともなっている．静的情報である高精度 3 次元地図 [8][9] は従来のナビゲーション用の地図よりも細かい粒度となっており，交差点間の道路を一本のリンクとするのではなく，車が走行するレーン単位で作成されており，精度も高くなっている．

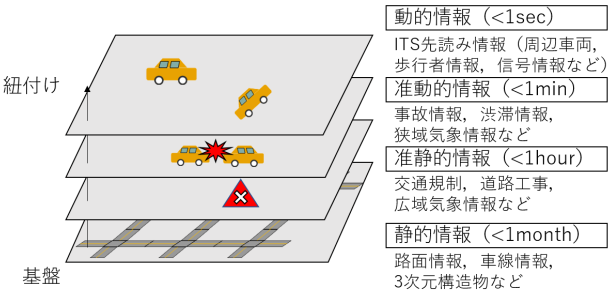

図 1　ダイナミックマップ（文献 [10] を参考に作成）

従来の地図における検索要求として，地図上のオブジェクト（コンビニなど）に対して，検索位置から近い順に k 個のオブジェクトを検索する kNN（k nearest neighbor）検索や指定された範囲内のオブジェクトを検索する範囲検索，指定した距離内のオブジェクトを検索する距離検索などが挙げられる．ダイナミックマップにおいても同様の検索をすることは可能であるが，ダイナミックマップは従来の地図と異なり，より高精度な地図情報（静的情報）と短い間隔で更新される車両の位置情報など（動的情報）が存在する．そのため，ダイナミックマップ上では，従来の地図にはない新しい検索要求が生まれると考えられる．検索例として，走行レーン単位で地図が存在し定期的に車両データが送られてくることにより，近辺の危険車両（逆行，蛇行）の検索や交差点で自車両と競合する可能性のあるレーンにいる車両の検索などが挙げられる．このようなダイナミックマップ上での検索要求は，検索範囲は広くなくとも単純な距離だけで結果が決まらない，応答にリアルタイム性が求められる，静的情報を間に挟んで複数の動的情報を関連付ける等の従来のものと異なる性質が存在する．したがって，新しい検索要求を高速に処理するためには新しい索引が必要になる．

本研究では，ダイナミックマップ上での検索要求として「交差点で自車両と競合する可能性のあるレーンにいる車両の検索」を対象としたシステム及び検索を高速に行うための索引技術を提案する．索引を使わない場合，対象とする検索を実現するにはレーンの接続関係や交差関係を演算する必要があるため，時間がかかってしまう．また，一つ一つの車両で自車両と競合する可能性のあるレーンにいる車両との関連付けを更新し続けるのも困難

[1] 非会員　東海国立大学機構名古屋大学大学院情報学研究科
gengen AT ertl.jp
[2] 正会員　東海国立大学機構名古屋大学未来社会創造機構
watanabe AT coi.nagoya-u.ac.jp
[3] 非会員　東海国立大学機構名古屋大学未来社会創造機構, 大学院情報学研究科
hiro AT ertl.jp

である．そこで提案する索引は「交差可能性のあるレーングループ」を一つの最小の構成要素とした道路地図に対する索引と，グループごとに分けて行う車両データに対する索引の 2 つで構成される．また，単純な手法との比較を行うことで提案する索引の有効性を示す．本研究の具体的な貢献は以下のとおりである．

- ダイナミックマップにおける静的情報と動的情報を組み合わせた検索．
- 提示した検索を高速に行うための索引の提案．
- 2 つのデータセットを基に性能測定を行い，有効性を確認．

本稿の構成は以下のとおりである．2 節では，動的オブジェクトに対する索引の関連研究について述べる．3 節では，本研究の前提となるデータや検索要求の問題設定について述べる．4 節では本研究で提案する索引手法について述べる．5 節では提案手法の実験結果および考察について述べる．6 節ではまとめと今後の課題を述べる．

2 関連研究

動的オブジェクトに対する kNN 検索や範囲検索のための索引は数多く存在する．Wang ら [11] は道路ネットワークの接続情報を保存したディスク上の Rtree と動的オブジェクトの位置更新を処理するメモリ上のグリッド索引を組み合わせた 2 重構造の索引を提案している．この索引は道路ネットワーク上の動的オブジェクトに対する範囲検索，kNN 検索をサポートしている．また，Zhang ら [12] は Apache Storm[13] の仕組みの上で動作する，動的オブジェクトのための分散索引を提案している．この索引は動的オブジェクトに対する範囲検索，kNN 検索，範囲空間結合検索（ある領域の一定距離内のオブジェクトのペアを検索）に対応したものとなっている．範囲検索，kNN 検索については動的オブジェクトを ID によってグループ化して，グループごとに索引を構築して検索を行う．範囲空間結合検索については，グリッドによるグループ化と，グループごとに索引を構築して検索を行う．ただし，この検索における距離は空間的な距離であり，道路ネットワーク上の距離ではない．

Wang ら，Zhang らのいずれも動的オブジェクトに対する索引であるが，対象とした検索が従来の地図における道路又は空間的距離を考慮した索引となっているため，本研究の道路地図を間に挟んだ複数の動的オブジェクトの検索を行う点と異なる．しかし，更新が頻繁に発生する動的オブジェクトは最初に索引を構築するのではなく，絞り込みを行ってから索引を構築するという考え方は本研究で取り入れている．

3 問題設定

本節では，本研究における車両情報ストリーム及び地図データの前提と，検索要求の定義について述べる．

3.1 データ

車両は，位置センサや通信装置を備えており高精度地図を持っている．そして車両データをサーバに送信しているものとする．車両情報ストリーム上を流れる各タプルに含まれるデータは，車両 ID，現在レーン ID，位置座標，送信時刻の 4 つである．

表 1 利用するデータ

データの種類	保持する要素
レーン L	$LaneId \times Linestring$（レーン ID と位置情報）
レーン間の交差関係 CR	$LaneId \times LaneId$（交差する 2 つのレーン情報の ID を持つ）
レーン間の接続関係 CN	$LaneId \times LaneId$（交差する 2 つのレーン情報の ID を持つ）
交差点 I	$IntersubsectionId \times Polygon$（交差点 ID と位置情報）
交差点への入力レーン IP	$IntersubsectionId \times LaneId$（交差点 ID と交差点への入力となるレーン ID）

地図データは，高精度地図について仕様検討は様々に行われているが，今回は我々の研究グループのフォーマット [9] に基づいたデータから表 1 に示すデータを作成し，使用した．データの種類とそれが保持する要素について説明する．

まずデータの種類について図 2 を例に説明する．レーン L は，各道路のレーン 1 本 1 本について区別された詳細度を持つ線のデータ集合である（青色の線）．レーン間の交差関係 CR は空間的に交差する 2 つのレーン同士の関係の集合を表す．例として，交差関係 {387,391} は，レーン 387 とレーン 391 が，交差点内で競合する関係にあることを意味している．レーン間の接続関係 CN は，通常の走行で移動できるレーン同士の関係の集合を表す．例として接続関係 {1251,393} は，レーン 1251 を走行した先はレーン 393 に繋がっていることを意味する．交差点 I は，2 つ以上の道路が交差する部分の領域（橙色の面）の集合である．交差点への入力レーン IP は，1 ホップ先のレーンが交差点に含まれるレーンの集合である．レーン 1287 は 1 ホップ先のレーン 387 やレーン 395 のレーンが交差点に含まれているため，交差点 32 の入力レーンとなる．交差関係，接続関係，交差点への入力レーンはレーン ID や交差点 ID を保持し，位置情報が必要な時は L, I の各要素が持つ，空間情報（linestring, polygon）を参照する．

次にデータが保持する要素について説明する．$LaneId$, $IntersectionId$ はレーンや交差点を一意に識別する ID である．$Linestring$ は有向線分であり，$\{(x_1, y_1, z_1), ...(x_i, y_i, z_i)\}$ のように始点から終点までの点の集合である．$Polygon$ は面であり $\{(x_1, y_1, z_1), ...(x_i, y_i, z_i), (x_1, y_1, z_1)\}$ のように始点から時計回りに一周して面を表す点の集合である．なお，車両，レーンや交差点の位置情報は 3 次元であるが，空間演算を行うときは簡略化のため 2 次元で計算している．

3.2 検索要求

本研究では，車両が交差点に近づいたとき，衝突の可能性がある他車両の検索を行うことで事故の可能性を低下させるために「交差点で自車両と競合する可能性のあるレーンにいる車両の検索」を対象にシステム及び索引の提案を行っている．より具体的な検索条件の定義を以下に示す．なお，交差点への入力レーンのうちの 1 つに着目したものをベースレーンとする．

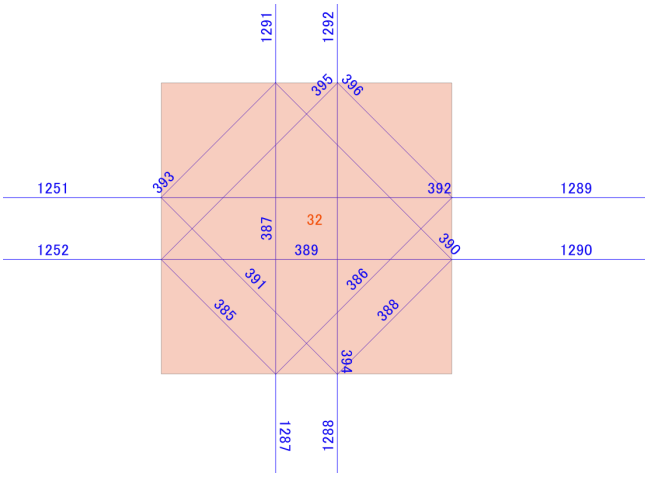

図 2　1 つの交差点におけるレーンと交差点の例

表 2　図 2 について，交差可能性のあるレーングループの一覧

ベースレーン	交差可能性のあるレーングループ
1251	{1251,1287,1290,1292}
1287	{1251,1287,1290,1292}
1290	{1251,1287,1290,1292}
1292	{1251,1287,1290,1292}

定義 1：レーングループ

ベースレーンに対して，交差点への入力レーンのうち，n ホップ先までたどって，交差点領域内で交差するもの同士を交差可能性のあるレーングループと定義する

定義 2：検索要求

車両情報ストリームに対して継続的に実行される連続クエリ Q(i, n, t) を，「車両 i の現在レーンをベースレーンとして n ホップのレーングループを求め，それらのレーン上にいる他車両の中で現在時刻から t 秒前までのデータを検索する」ものとしたとき，Q を「交差点で自車両と競合する可能性のあるレーンにいる車両の検索」と定義する．

定義 1 について図 2 を例として説明する．交差点への入力レーンとして {1251,1287,1290,1292} が存在する．ここで交差点への入力レーンの 1 つである 1251 をベースレーンとする．1251 からホップして交差点領域内にあるレーンは {391,392,393} である．そしてそれらのレーンは，入力レーン 1292 からホップして交差点領域にあるレーンである 394, 395 などと交差する．同様に入力レーン 1290, 1292 についても交差する．よって，ベースレーン 1251 に対する交差可能性のあるレーングループは {1251,1287,1290,1292} の 4 つとなる．これを全ての入力レーンについて考えたものが表 2 である．これにより，例えば自車両がレーン 1287 にいるとき，ベースレーン 1287 に対する交差可能性のあるレーングループを見ることで演算することなく結果を得ることができる．定義 2 について変数である i,n,t はユーザが設定する値とする．

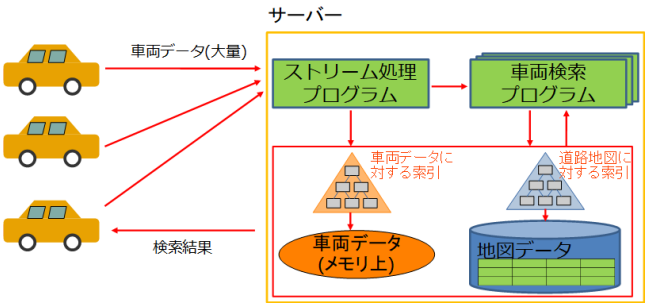

図 3　システムの全体像

4　提案手法

4.1　システムの全体像

本研究の主な貢献は，ダイナミックマップ上での新しい検索要求の提示と索引の提案ではあるが，実際に車両の検索をどのように行っていくかを理解しやすくするため，まずはシステムの全体像について述べる．システムの全体像を図 3 に示す．システムは車両側とサーバ側に分かれる．車両側は前節で述べたデータ（車両 ID，現在レーン ID，現在位置，送信時刻）をサーバ側に送信する．サーバ側ではストリーム処理プログラムと車両検索プログラムを実行する．ストリーム処理プログラムでは，車両データ（ストリームデータ）の処理とクエリの登録を行う．車両データの処理では，車両データをメモリへ保存して，車両データとそのデータへのポインタを車両検索プログラムへ送信する．クエリの登録では，検索内容を決定して車両検索プログラムに内容を送信する．内容は定義 2 の通りでユーザは i,n,t を指定する．なお，メモリの車両データはストリーム処理プログラムがクエリで定義されている時間分だけ確保する．車両検索プログラムでは，ストリーム処理プログラムから受け取ったデータの処理を行う．車両データを受け取ったとき，受け取ったデータを基に車両データに対する索引の更新を行う．登録されたクエリの条件（=車両 ID が i の車両が交差点への入力レーンに存在）を満たすデータだった場合は，索引の更新と検索を行う．

4.2　ナイーブ手法

まず，比較のために「交差点で自車両と競合する可能性のあるレーンにいる車両の検索」をするときに，従来の索引（hash tables，r*tree for lanes）を使用して検索を行う手順について説明する．

4.2.1　hash tables

3.1 節で述べたように地図データは表 1 に示す 5 つのデータが存在する．事前にそれぞれのデータをハッシュをキーとしてグループ化（ハッシュテーブル）してデータへのアクセスを高速化する．そして「自車両と競合する可能性のあるレーンにいる車両の検索」を以下の手順で実行する．

1. 現在レーン ID から交差可能性のあるレーングループを求める
2. メモリ上の車両データから現在レーン ID が交差可能性のあるレーングループに含まれるものを見つける

Algorithm 1 交差可能性のあるレーングループを演算

Input: laneid
Output: laneid_group
/* initialize */
current_laneid（レーン ID を保持）
next_laneid（current_laneid の次レーンのレーン ID を保持）
1: next_laneid_queue = ϕ
2: cross_laneid_queue = ϕ
3: laneid = EnQueue(next_laneid_queue)
4: **while** next_laneid_queue != ϕ **do**
5: current_laneid = DeQueue(next_laneid_queue)
6: **if** IsInputLane(current_laneid) **then**
7: next_laneid = NextLane(current_laneid)
8: **if** IsCovered(next_laneid) **then**
9: CrossLane(next_laneid) = EnQueue(cross_laneid_queue)
10: next_laneid = EnQueue(next_laneid_queue)
11: **end if**
12: **end if**
13: **end while**
14: **while** current_laneid = DeQueue(cross_laneid_queue) **do**
15: **while** NOT IsCovered(current_laneid) **do**
16: current_laneid = PreviousLane(current_laneid)
17: **end while**
18: **if** current_laneid != laneid **then**
19: current_lane = EnQueue(laneid_group)
20: **end if**
21: **end while**

3. 現在時刻から t(s) 前までの車両データを見つける

手順 (1) の交差可能性のレーン ID を見つけるアルゴリズムを Algorithm1 に示す．また，手順内で呼び出されている関数を次のように定義する．次レーンの ID を返す関数を $NextLane(lid1) = \{lid2|(lid1, lid2) \in CR\}$ とする．前レーンの ID を返す関数を $PreviousLane(lid1) = \{lid2|(lid2, lid1) \in CR\}$ とする．交差するレーンの ID を返す関数を $CrossLane(lid1) = \{lid2|(lid1, lid2) \in CR\}$ とする．入力レーンであれば真を返す関数を $IsInputLane(lid) = \exists(interid, lid) \in IP \Rightarrow True$ とする．レーンが交差点に含まれていたら真を返す関数を $IsCovered(lid) = \exists(interid, poly) \in I \wedge (lid, ls) \in L \wedge Covered(ls, poly) \Rightarrow True$ とする．なお，$Covered(ls, poly)$ は空間演算を行うことで ls が $poly$ に含まれていたら真を返す関数とする．

4.2.2 r*tree for lanes

こちらはテーブルの 1 つであるレーン L について，レーン 1 つをエントリとした R*tree を構築して，hash tables における手順 (1) を実行する前に，現在位置を中心とした範囲検索を行う．それによってレーン L のデータを現在位置近くのレーンのみに絞り込むことで，レーン L へのアクセス時間を削減する．それ以外の部分は hash tables と検索する手順は同様である．

4.3 索引構造

先に述べた 2 つのナイーブ手法はいずれも検索を実行するときに交差可能性のあるレーングループを演算する必要があり，大規模なデータ処理には向いていない．そこで本研究では，「交差点で自車両と競合する可能性のあるレーンにいる車両の検索」に適した索引構造を提案する．

提案する索引は 2 つで構成される．1 つは道路地図に対する索引で，もう 1 つが車両データに対する索引である．まず，索引を 2 つの構造に分けたポイントについて説明する．索引は新しいデータの挿入や削除が発生すると，索引の更新処理を行う必要がある．そのため，絞り込みを行っていない車両データに対して，例えば現在レーン ID をキーとして索引を構築したとき，短い間隔（1 秒以内）で，車両の台数分だけ索引の更新処理が必要となってしまい，コストが大きい．また，道路地図に対する索引を構築して，交差可能性のあるレーングループを高速に見つけることが出来ても，メモリ上にある大量の車両データの中から該当するものを探す必要がある．そこで，道路地図に対する索引で車両データをグループ分けすることで，全車両データに対して索引を構築することなく，必要な車両データへのアクセスを高速化している．

次に，それぞれの索引について詳細を述べる．道路地図に対する索引では，交差可能性のあるレーングループを事前に計算して空間索引を構築する（図 4）．空間索引には R*tree を利用する．R*tree に挿入するエントリの内容は，ベースレーン，交差可能性のあるレーングループ及び交差可能性のあるレーングループを包含する MBR である．全ての交差点においてベースレーンとそれに対する交差可能性のあるレーングループを Algorithm1 により演算する．しかし，そのままでは表 2 のように交差可能性のあるレーングループが同じ (=MBR が同じ) エントリが複数存在してしまうため，無駄が生じる．よって，交差可能性のあるレーングループが同じものについては，ベースレーンをまとめて，エントリを構築する．これによって，自車両の現在レーン ID と現在位置があれば，現在位置を含むエントリを R*tree によって検索した後 (複数存在する可能性あり)，現在レーン ID を含むベースレーンをもつエントリを見つけることで（エントリを一意に特定)，現在レーン ID に対応する交差可能性のあるレーングループを見つけることができる．

図 2 の交差点を例にすると，ベースレーンと交差可能性のあるレーングループを演算によって表 2 のように求める．次に，交差可能性のあるレーングループが同じベースレーンを図 5 のようにまとめる．ベースレーン {1251,1287,1290,1292}，交差可能性のあるレーングループ {1251,1287,1290,1292}，交差可能性のあるレーングループを包含する MBR をエントリとして挿入する．

車両データに対する索引では，先に述べた道路地図に対する索引を利用する．各エントリごとに車両情報リストを作成し，エントリに車両情報リストへのポインタを追加する．車両情報リストは，対応するエントリの交差可能性のあるレーングループを現在レーン ID とする車両データへのポインタを保持する (図 6)．そのため，車両検索プログラムは車両データとそのポインタをスト

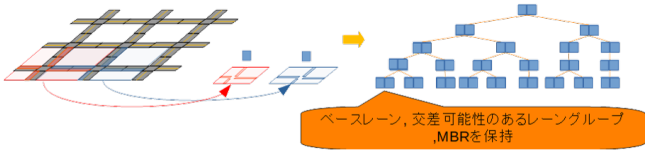

図 4　道路地図に対する索引の全体像

ベースレーン	交差可能性のあるレーングループ
1251	{1251,1287,1290,1292}
1287	{1251,1287,1290,1292}
1290	{1251,1287,1290,1292}
1292	{1251,1287,1290,1292}

ベースレーン	交差可能性のあるレーングループ
{1251,1287,1290 ,1292}	{1251,1287,1290,1292}

図 5　ベースレーンをまとめる例

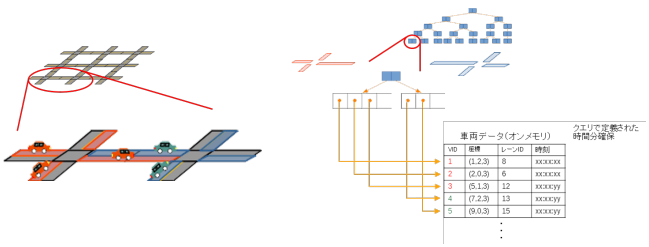

図 6　車両データに対する索引の全体像

リーム処理プログラムから受け取ったあと，車両データの現在レーン ID を含む交差可能性のあるレーングループを持つエントリを探して，エントリの車両情報リストの後ろに車両データへのポインタを追加する．

以上により，検索を実行する際，現在レーン ID を含むベースレーンを持つ葉ノードを探し，ノードの車両情報リストを見ることで，交差可能性のあるレーンにいる車両のみを車両データの中から見ることができる．車両情報リストの後ろが最新時刻のものとなっているため，後ろから順に読み取っていくことで，最新時刻から近い順に t(s) 前までの車両データを得ることができる．

5　評価実験

本研究で提案した道路地図に対する索引について以下の 5 点について評価を行った．

- 索引の構築時間
- 挿入時間
- 削除時間
- 検索時間
- スループット

また，車両データに対する索引の評価として，車両情報リストへのポインタを道路地図に対する索引から得てから，車両データを取得するまでの時間を測定した．

5.1　実験環境

実験 PC の性能を表 3 に示す．また，本実験では，2 つの OSS を利用している．1 つは，地図データを保存しているデータベースの PostgreSQL（PostgreSQL 12.3, PostGIS 3.0）である．もう 1 つが Boost C++ である．Boost C++（ver1.74）には，Rtree

表 3　PC の性能

OS	Windows 10 Home
CPU	Intel(R) Core(TM) i7-7700HQ
DB version	PostgreSQL 12.3，PostGIS 3.0
ディスク	HDD 3TB （SATA 5,400 回転/分）
メモリ	8GB DDR4 SDRAM（4GB × 2）

に関する機能が備わっている [14]．実験時には，地図データを PostgreSQL から読み込み，Boost C++ の機能を使って索引を構築している．

5.2　実験データ

本研究では，実際に存在する地図に対して機能するか及びデータ数が増加したときにどのような影響があるかという点を調べるために 2 種類のデータセットを用いた．1 つ（高蔵寺データ）は愛知県春日井市にある高蔵寺ニュータウンという場所の地図データ（図 7）及び PTV Vissim[15] という交通シミュレータを用いて生成した車両データである．地図データの交差点数は 195，レーン数は 3,306 存在する．もう一つ（仮想データ）はマンハッタンモデルという道路モデルを用いた地図データと交差点間のレーンにランダムで生成された車両データである．マンハッタンモデルはグリッド状道路モデルである．本実験では，車線幅 3.5m，道路幅 16.5m，交差点間 200m，片側 1 車線の道路で構成されている（参考 [16]）．交差点数が 10x10 の例を図 8 に示す．実験時には交差点数を変化させて（10x10，20x20，30x30，40x40，50x50）測定を行う．

図 7　高蔵寺データ

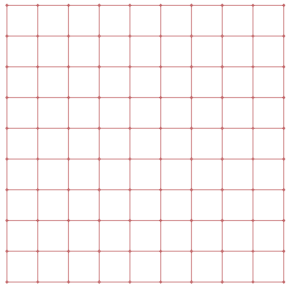

図 8　仮想データ

5.3　実験結果

5.3.1　索引の構築時間

2 つのデータセットに対して道路地図に対する索引の構築時間の測定を行った．構築時間は交差可能性のあるレーングループの演算時間，交差可能性のあるレーングループを包含する MBR の演算時間及び R*tree の構築時間の合計である．なお，100 回測定を行い平均を取っている．高蔵寺データの結果を表 4 に示す．また，仮想データのグラフを図 9 に示す．

仮想データの実験結果から交差点数と構築時間はほぼ比例の関係にあることが分かった．また，高蔵寺データは仮想データと比較して，構築時間が大きくなっている．これは，高蔵寺データは，道路の疎密に偏りがあるため，索引の構築時にノードの分割処理が多くなってしまったためだと考えられる．

5.3.2 挿入時間

索引の構築後の状態から，エントリの挿入を行ったときの時間を測定した．挿入するエントリの MBR の位置は地図全体からランダムに決定して，大きさは 200m × 200m の正方形とする．なお，100 回測定を行っている．挿入時間についてはそれぞれのデータ値の差が大きいため，箱ひげ図によりグラフ化した．なお，×が平均値を表し，データの値が 第三四分位数 ＋四分位範囲 × 1.5 となるものについては別途丸の点で表示している．高蔵寺データ，仮想データの箱ひげ図を図 10，11 に示す．

交差点数が大きくなるほど，挿入時間は増加している．また，挿入時間について，平均から大きくずれた値が生じるときがある．さらにそれは交差点数が増加するほどに多く見られる．挿入時間が大きくなってしまう原因として，エントリを挿入したときに R*tree の更新時に大きな木の作りかえが必要になるからだと考えられる．また，この作りかえが必要になる可能性は，木が大きくなっていくほど増加していくと考えられる．

5.3.3 削除時間

索引の構築後の状態から，エントリの削除を行ったときの時間を測定した．削除するエントリは，構築された索引にあるエントリの中からランダムに決定する．なお，100 回測定を行い平均をとっている．高蔵寺データの結果を表 5 に示す．また，仮想データのグラフを図 12 に示す．

交差点数が大きくなるほど，削除時間が増加していることが分かった．また，索引の構築時間と同様に高蔵寺データは仮想データよりも削除時間が大きくなっている．

5.3.4 検索時間

道路地図に対する索引を用いて交差可能性のあるレーングループを検索するのにかかる時間を測定した．なお，100 回測定を行い平均を取っている．高蔵寺データ，仮想データの結果図 13，図 14 に示す．

仮想データの結果から，交差点数が増加しても，検索時間に大きな違いが見られないことが分かった．また，高蔵寺データ，仮想データともに提案手法，hash tables，r*tree for lanes の順に検索時間が大きくなっていることが分かった．r*tree for lanes が hash tables よりも検索時間が大きくなってしまったのは，ハッシュテーブルによる検索のほうが高速で，範囲検索によるレーンの絞り込みがボトルネックになってしまったためだと考えられる．

5.3.5 スループット

道路地図に対する索引を用いて交差可能性のあるレーングループを検索を 10,000 回逐次実行することによってスループット（1 秒当たりの検索処理数）を測定した．高蔵寺データ，仮想データの結果を図 15，図 16 に示す．

結果から交差点数が増加するとスループットが減少するが，ある程度交差点数が大きくなると，スループットの低下が小さくなっていることが分かる．また，高蔵寺データ，仮想データともに提案手法のスループットが一番大きいことが分かる．

5.3.6 車両データに対する索引の評価

車両情報リストへのポインタを得てから，車両情報リストを経由して車両データを取得するためにかかる時間を評価した．な

表 4 高蔵寺データの索引構築時間 (ms)

提案手法	2112.803

表 5 高蔵寺データの削除時間 (μs)

提案手法	52.46

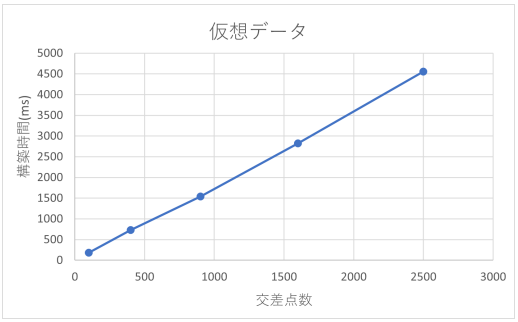

図 9 仮想データの索引構築時間

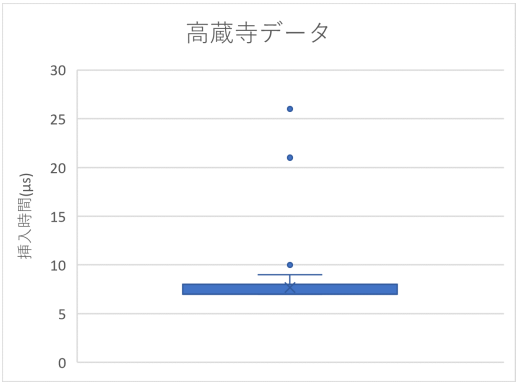

図 10 高蔵寺データの挿入時間

お，測定に利用したデータは仮想データの交差点数 10x10 のものである．10,000 台分の車両データを利用しており，車両データへのポインタを車両情報リストに追加すると同時に車両情報リストに存在する車両データへのポインタから車両データを検索することにより測定した．結果を図 17 に示す．横軸が取得した車両台数であり，縦軸が取得にかかった時間 (ns) である．

結果から，取得にかかる時間は取得する車両台数に比例していることが分かる．また，車両情報リストの性質から，地図全体の交差点数が増加しても，車両情報リストに格納される車両データへのポインタの数が増えるわけではない点，交差可能性のあるレーングループにいる車両数には限界がある点から車両の取得時間のオーダーについて，大きな問題にはならないと考えられる．

5.4 考察

既存の R*tree と異なる要素のある道路地図に対する索引の構築時間及び検索時間の計算複雑度について述べる．なお，木構造の索引はデータ数を N とすると検索時間は木の高さに依存して，$O(logN)$ となる．構築時間は，問題設定をした 5 つのデータから交差可能性のあるレーングループおよびレーング

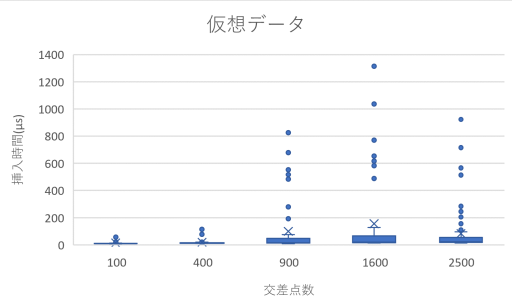

図 11　仮想データの挿入時間

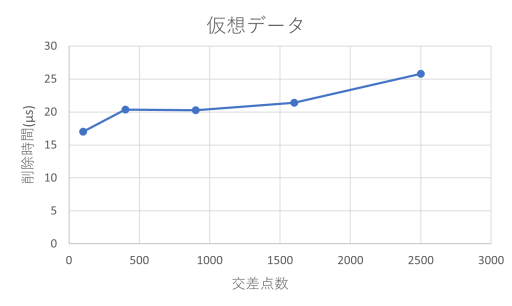

図 12　仮想データの削除時間

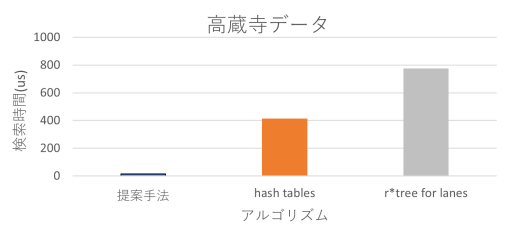

図 13　高蔵寺データの検索時間

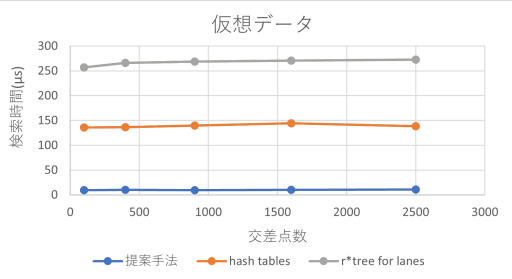

図 14　仮想データの検索時間

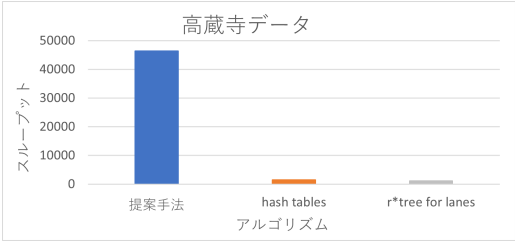

図 15　高蔵寺データのスループット

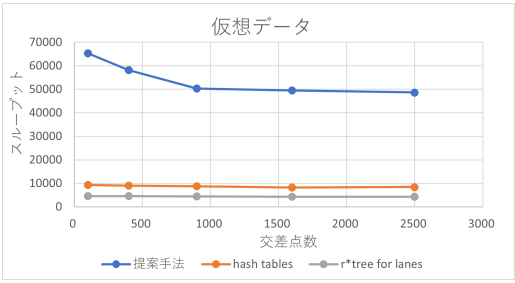

図 16　仮想データのスループット

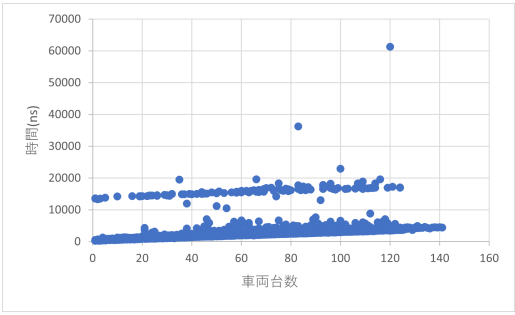

図 17　車両情報リストを利用した車両データの検索時間

ループの MBR を求める演算時間と R*tree の構築時間の和である．まず，交差可能性のあるレーングループを求める演算を考える．交差可能性のあるレーングループの演算は Algorithm1 を交差点への入力レーンの数だけ繰り返す．Algorithm1 の計算複雑度について，5 つのデータはそれぞれ数は異なるが，地図が大きくなれば（交差点数が増えれば），それに比例して大きくなる．そこで，交差点数を N として，交差点数を基準に考える．$NextLane, PreviousLane, CrossLane, IsInputLane$ 関数は Btree をはった一つのデータにアクセスするため $O(logN)$ となる．$IsCovered$ 関数は，Btree をはったデータへのアクセスと GiST を用いた空間検索を行うため，$O(logN + logN) = O(logN)$ である．4-13 行，14-21 行や 15-20 行にループが存在するが，これらはいずれも高々交差点内に存在するレーン数繰り返すだけであるため，定数と考えて良い．したがって，このアルゴリズム全体は $O(NlogN)$ となる．次に MBR を求める演算であるが，これは交差可能性のあるレーングループの ID をレーンデータから読み込み，最小，最大座標を求める処理を行う．こちらも交差可能性のあるレーングループ数だけ繰り返すため $O(NlogN)$ となる．最後に R*tree の構築時間について，挿入をデータ数分だけ繰り返す．挿入処理は，挿入すべき葉ノードの探索と葉ノードのエントリ数がすでに最大であった場合ノードの分割を行う．ノードの分割にかかる時間は文献 [17] より最大エントリ数を M とすると $O(MlogM)$ である．したがって，$O(\sum_{k=1}^{N}(logk + MlogM)) = O(logN! + NMlogM)$ となる．以上より，道路地図に対する索引の構築時間は $O(NlogN)$ である．実験結果からは比例の関係に見えるが，これは交差点数が 100 とある程度大きい値から始まっているためだと考えられる．

検索時間は R*tree の探索時間と現在レーン ID をベースレーンに含むエントリの探索時間の和である．R*tree の検索時間は，木の高さに依存して $O(logN)$ である．現在レーン ID をベースレーンに含むエントリの探索は，R*tree の探索によって得られた結果を n とすると線形探索により $O(n)$ となる．しかし，道路構造から R*tree の探索によって得られるエントリ数には限りがある．そのため定数と考えて良く，全体として $O(logN)$ が得られる．検索時間とスループットの結果を見ると検索時間は値が小さく分かりづらいが，スループットに関しては，その傾向が見られるため，測定結果は理論的にも妥当であると考えられる．

スループットの実験結果から，提案手法の索引を利用することで交差可能性のあるレーングループの検索だけでも hash tables よりも 5 倍以上処理できていることが分かる．

しかし，提案手法では，道路地図に対する索引を事前に構築する必要があり，車両データが送られてくるたびに，その車両データが（現在レーン ID が交差可能性のあるレーングループに含まれる）エントリを検索する必要がある．ただ，索引を構築する必要がある点に関しては，索引の構築を行うのは最初の 1 回のみで，地図の更新があった場合はエントリの挿入をするのみでよく，さらに地図の更新頻度は高くない（SIP-adus の想定では一ヶ月以内の更新頻度（図 1））ことや規模が大きくなっても $O(NlogN)$ であることから，大きな問題にはならないと考えられる．車両データが送られてくるたびに，エントリを検索する必要がある点に関しては，高蔵寺データの場合 1 秒あたり 48,000 台分の車両データが送られてくると限界になってしまうため，分散処理を行うなどの対処が必要になる..

6 まとめと今後の課題

本研究では，ダイナミックマップ上で考えられる新しい検索要求の 1 つである「自車両と競合する可能性のあるレーンにいる車両の検索」を提示した．そして，検索を実行するためのシステムの構成及び検索を高速化するための索引手法を提案した．

今後の課題として，大別して 2 つのものが存在する．1 つ目が車両データに対する索引の洗練である．索引は必要なデータのみを効率よく見つけることが望ましい．しかし，提案手法では，現状車両データへのポインタを配列の後ろに追加していっているだけであるため，後ろから順に指定した時間内でなくなるまで読み取っていく必要がある．ただ，車両データに対する索引に関して，より効率的な索引を追加しようとすると，車両データの追加時に索引の更新処理にかかる時間が大きくなってしまうため，検証をしっかり行う必要がある．

2 つ目は，継続的な動作に関する検証である．本実験では，検索を 1 回行うだけであったり，短時間で決められたデータ量を送信していた．しかし，実際に運用しようと考えると，長時間動作させ続けても同様のスループットが得られるのか，といった点や過負荷があった場合にどうするのかといった点についても考えていく必要がある．

謝辞

本研究の一部は，JST，OPERA，JPMJOP1612 の支援を受けたものである．

参考文献

[1] 青木 啓二. 自動運転車の開発動向と技術課題 : 2020 年の自動化実現を目指して. 情報管理, Vol. 60, No. 4, pp. 229–239, 2017.
[2] “全方位レーザー LiDAR イメージングユニット”. https://www.argocorp.com/cam/special/Velodyne/VLP-16.html.
[3] Manuel Gonzalez-Martín, Miguel Sepulcre, Rafael Molina-Masegosa, and Javier Gozalvez. Analytical models of the performance of c-v2x mode 4 vehicular communications. *IEEE Transactions on Vehicular Technology*, Vol. 68, No. 2, pp. 1155–1166, 2018.
[4] 柳内洋一, 太刀川喜久男, 中村順一, 坂本敏幸. Etc を支える無線通信方式 dsrc. 映像情報メディア学会誌, Vol. 63, No. 2, pp. 179–184, 2009.
[5] 小山 浩，柴田 泰秀. “自動走行におけるダイナミックマップ整備”. 「システム/制御/情報」, Vol. 60, No. 11, pp. 463–468, 2016.
[6] 渡辺 陽介，高木 健太郎，手嶋 茂晴，二宮 芳樹，佐藤 健哉，高田 広章. “協調型運転支援のための交通社会ダイナミックマップの提案”. 第 7 回データ工学と情報マネジメントに関するフォーラム (DEIM2015)，F6-6, 2015.
[7] 自動運転（システムとサービスの拡張）sip-adus. https://www.sip-adus.go.jp/.
[8] “HERE HD Live Map — HERE - HERE Technologies”. https://www.here.com/products/automotive/hd-maps.
[9] “名古屋大学 COI 高精度地図フォーマット仕様書”. http://www.nces.i.nagoya-u.ac.jp/dm2/COImap_20170906.pdf.
[10] “SIP 自動走行システム”. https://www8.cao.go.jp/cstp/tyousakai/juyoukadai/system/2kai/shiryo3-1.pdf.
[11] Wang Haojun, Zimmermann Rogger. Location-based query processing on moving objects in road networks. In *Proc. VLDB*, pp. 321–332. Citeseer, 2007.
[12] Feng Zhang, Ye Zheng, Dengping Xu, Zhenhong Du, Yingzhi Wang, Renyi Liu, Xinyue Ye. “Real-Time Spatial Queries for Moving Objects Using Storm Topology”. *ISPRS International Journal of Geo-Infromation*, Vol. 5, No. 10, p. 178, 2016.
[13] Apache storm. https://storm.apache.org/.
[14] “Chapter1 Geometry - 1.74.0”. https://www.boost.org/doc/libs/1_74_0/libs/geometry/doc/html/index.html.
[15] “PTV Vissim: Software for multimodat trafic simulation”. https://www.ptvgroup.com/ja/ソリューション/製品/ptv-vissim/.
[16] “ITS 通信シミュレーション評価シナリオ（Ver.1.2）− ITS シミュレータ利用促進検討委員会−”. http://www.jari.or.jp/tabid/259/pdid/7985/Default.aspx.
[17] N. Beckmann, H.P. Kriegel, R. Schneider and B. Seeger. The R*-tree: an Efficient and Robust Method for Points and Rectangles. *Proceedings ACM SIGMOD Conference on Management of Data*, pp. 322–331, 1990.