論文

MD 木を用いた移動オブジェクト管理

Management of Moving Objects with MD-tree

西川嘉人，金子裕良，阿部　茂

Yoshihito NISHIKAWA, Yasuyoshi KANEKO and Shigeru ABE

The progress in wireless technologies and the advancement of GPS accuracy makes us to get the positions of moving objects easily. Mobile instrument spreads across our social life quickly, so it is important to manage the moving objects data for emerging applications with those technologies. In this paper, we describe an efficient index structure MD-MOtree, which is based on MD-tree, to deal with moving objects data that has frequent insertion and deletion. There are no overlaps and no needs to periodical restructuring of the tree structure by not integrating internal nodes on its delete process. And its combined delete-insert algorithm in each object cut down the processing time.

The good performance of MD-MOtree for managing the moving point objects is shown by simulation experiments.

Keywords: Moving Objects, Index Structure, MD-tree, Frequent Insertion and Deletion

1.はじめに

無線通信技術の発達や GPS 精度の向上により、車や電車、人といった時間によって位置の変化する物体（以下、移動オブジェクトと呼ぶ。）の位置情報取得が容易となっている。それにともない移動オブジェクトの運行の効率化や安全性向上のためのアプリケーションが現実のものとなりつつあり、移動オブジェクト管理に適したデータ構造が研究されている。研究は管理するデータにより２種類に分けられる。一つは移動オブジェクトの履歴・軌跡データを管理するものであり、もう一方は最新の位置データだけを管理するものである。前者の研究としては、データ構造の変化の差分を保持してメモリコストを少なく管理をおこなう Extended PMD-tree[1)]、交通管制を目的としたオンライン解析処理のための Σ-tree[2)]、複数の木を持つことで検索を高速化させる XAT 構造[3)]、移動統計量から分析・予測を行うための管理手法[4)]などがある。

後者の研究では、ノードが管理する領域を時間の関数として未来予測検索を効率的に行うための TPR-tree[5)] や、予測精度を向上させる手法の研究[6)]などがある。これら最新位置データ管理では、時刻を更新する毎に全管理データ数の内、かなりの量の旧データを削除し新データを挿入する必要がある。

本研究ではこの削除と挿入の問題に対して、処理速度、メモリ効率、さらに検索効率に優れたデータ管理構造を提案する。

多くの R-tree[7)]を拡張した木構造や MD-tree[8)]で移動オブジェクト管理を継続して行った場合、その頻繁なデータの挿入と削除によって、データを管理している包括長方形(Bounding Rectangle)の肥大化や重なり(Overlap)が生じ、処理性能が著しく悪化する。一般に

埼玉大学　工学部　電気電子システム工学科
Department of Electrical and Electronics System, Faculty of Engineering, Saitama University, 255 Shimo-Okubo, Sakura-ku, Saitama, Saitama, 338-8570, Japan

（原稿受付日：平成18年 3月31日）

その解決のために木の再構築(Restructure)を行うが、再構築は処理量が大きいために、同時に起こる検索処理要求に対して対応が遅くなる問題があり、改善の余地がある。

本論文では頻繁な挿入と削除が生じる移動オブジェクト管理に対して有効なデータ構造 MD-MOtree を提案する。その特徴を以下に示す。

1）データの挿入と削除の処理が高速。
2）頻繁なデータの挿入と削除による性能劣化がなく、木の再構築の必要がない。
3）点データに対して管理領域に重なりが生じない。

性能劣化がなく、木の再構築の必要がないことから、MD-MOtree は長時間にわたり安定かつ高速に最新情報を管理することが出来る。例えば、多くの車や電車の監視業務においては、最新の位置情報が重要で、しかも速い応答速度が要求される場合が多く、これらにおいて有効である。

MD-MOtree は多次元データ管理構造である MD-tree を拡張したもので、削除アルゴリズムにおいて内部ノードの統合を行わない。これによって頻繁な挿入と削除に対して性能劣化がなく、処理量を低減できる。またオブジェクトごとに削除と挿入を一貫して行うため、木構造にデータを挿入する際の処理量を低減できる。

計算機実験では、MD-tree と MD-MOtree の挿入と削除に要する処理時間、メモリ効率(葉ノードのバケット容量にデータが入っている割合)、検索処理量について比較をおこなった。

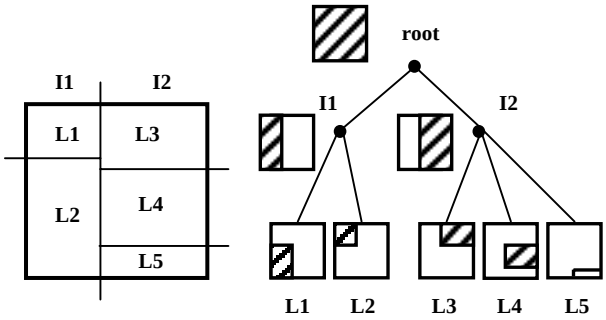
以下、２章で MD-tree と簡易 MD-tree、提案手法 MD-MOtree について述べ、その違いを示す。３章では計算機実験によってこれらの tree の性能評価を行い、特性を比較する。

2.データ構造

本章では、MD-tree、簡易 MD-tree、MD-MOtree に関して、そのアルゴリズムと特性を示す。

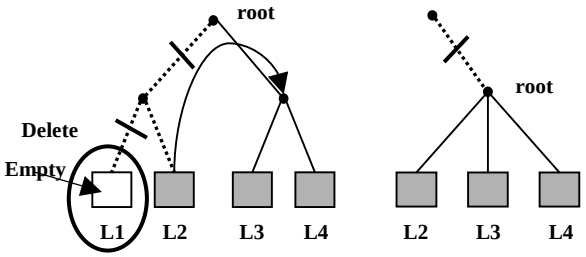
2.1 MD-tree

MD-tree は B-tree を多次元に拡張した 2-3 木で、



a) Region splitting b) Region which nodes have.

Fig.1 An example of MD-tree.



a) Delete empty leaf L1. b) After deletion.

Fig.2 An example of MD-tree delete algorithm.

k-d tree[9]や quad-tree[10]のようにデータ空間を再帰的に分割し、分割された領域を各ノードに対応させて領域内のデータを階層的に管理する(Fig.1)。このとき、各ノードに割り振られた領域を管理領域と呼ぶ。さらに、検索のために各ノードは管理領域内のデータを囲む包括長方形(Bounding Rectangle)を持ち、検索時は根ノード(root)から包括長方形の領域内を検索する。木のレベルは常にバランスし、管理領域を動的に調節することでメモリ効率(storage utilization)を高く保つことができ、最悪時でも 66.7%を保障する。メモリ効率 *S* は、木構造内に蓄積されるデータ数 *Dn*、葉ノード数 *Ln*、バケット容量 *P*(page capacity: １つの葉ノードが所持できるデータ数)とすれば、以下の式で表される。

$$S = \frac{Dn}{Ln \times P}$$

MD-tree は他のデータ構造よりメモリ効率が高く[11]、木構造自体のメモリ使用量が小さい。

MD-tree の木構造が作成された初期状態においては管理領域に重なりはない。MD-tree はデータが削除された場合、その高いメモリ効率を保つために、葉ノードの削除やノード統合を行う(Fig.2)。場合によってはノード統合が上位ノードまで連鎖するため、初期状態

では整えられていた管理領域の秩序が崩れ、管理領域の重なりが生じる。

MD-tree はメモリ効率を高く保てるが、削除時の処理量が大きい。従来の地理情報システムのようにデータ削除が少ない用途では問題ないが、移動オブジェクトのように頻繁にデータの挿入と削除が繰り返される用途では、削除アルゴリズムによって管理領域に重なりが生じて性能が劣化する。

2.2　簡易 MD-tree(EasyMD-tree)

MD-tree はデータ削除時にバケット内のデータ数が兄弟ノードと統合可能となるまで減少すると、葉ノードの削除やノードの統合を行う。しかし移動オブジェクトの場合、削除されるとデータは直ちに挿入されるため、メモリ効率を犠牲にしてノードの削除・統合を行わない方法が考えられる。これを簡易 MD-tree とする。従って簡易 MD-tree の挿入や検索アルゴリズムは MD-tree と同じである。簡易 MD-tree 削除アルゴリズムを以下に示す。

[簡易 MD-tree 削除アルゴリズム]
削除対象データ A
S1.　根ノードより木をたどり A を検索.
S2.　A を葉ノードから削除。終了。

この削除アルゴリズムにより、削除時の複雑で大きな処理はなくなり、管理領域の重なりが生じない。また削除時に葉ノードを削除しないことで、木構造があらかじめ用意されていることになり、挿入時のノード分割による処理の低減にもつながる。

簡易 MD-tree は削除時に木構造を変化させずにそのままにし、メモリ効率を犠牲にするかわりに削除時の演算量を大きく低減する。また、点データの挿入と削除に対して管理領域に重なりが生じない。

2.3　MD-MOtree

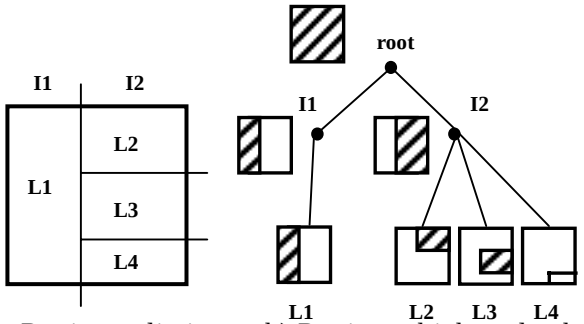
簡易 MD-tree のメモリ効率を改善し、移動オブジェクトのデータの削除と挿入が対になって生じる特性を利用したのが MD-MOtree である。MD-MOtree は、簡易 MD-tree に対して次の二つの変更を加える。

1.　削除時の葉ノード統合（葉ノードレベルのみ）
2.　挿入ルートの短縮

まず削除時の葉ノード統合について述べる。移動オブジェクトは常に同じ場所にとどまることはなく、継続して更新を続けるために簡易 MD-tree では多数の空ノード（データを持たない葉ノード）が生じてメモリ効率が極端に悪化する。空ノードは無意味にメモリを使用するため、削除時に不必要な葉ノード統合を行う(Fig.4)。以下に MD-MOtree 削除アルゴリズムを示す。

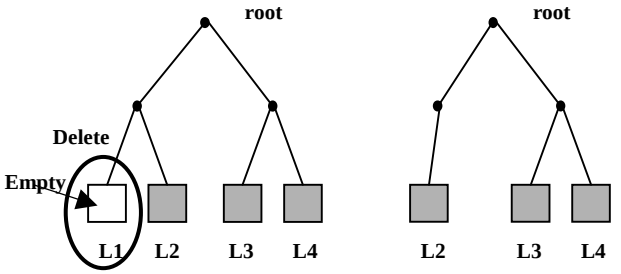
[MD-MOtree 削除アルゴリズム]
削除対象データ A、葉ノードのバケット容量 P、
S1.　根ノードより木をたどり A を検索
S2.　A を葉ノード L から削除
S3.　L に兄弟ノードが存在するとき、
　　leaf_integration へ.
　　L に兄弟ノードが存在しないとき、終了.
◇leaf_integration:
L の親ノード I、
S1.　L の兄弟ノードが 2 つのとき、
　　L と兄弟ノード L1、L2 が持つデータ数の総和が 2P 以下ならば、L、L1、L2 を削除し、L、L1、L2 のデータを、I を根ノードとする部分木から再投入する.終了.
S2.　L の兄弟ノードが 1 つのとき、
　　L と兄弟ノード L1 のデータの和が P 以下ならば、L と L1 を削除し、L と L1 のデータを、I を根ノードとする部分木から再投入する.終了.

MD-MOtree 削除アルゴリズムにより、無駄な葉ノードが存在することを防ぎ、ノード分割によって木が広が



a)　Region splitting.　b) Region which nodes have.

Fig.3　An example of MD-MOtree.



a) Delete empty leaf L1. b) After deletion.

Fig.4 An example of MD-MOtree delete algorithm.

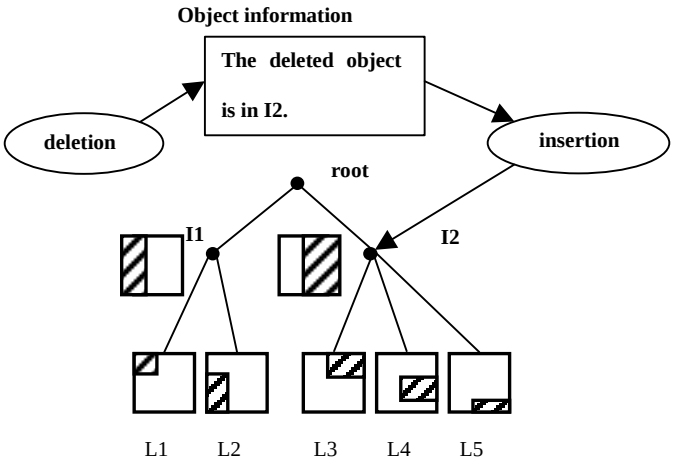


Fig.5 Insertion of MD-MOtree

り続けるのを防ぐ。統合は葉ノードのレベルでのみ行われ、上位ノードに連鎖することはない。葉ノードのレベルでは兄弟ノードのない葉ノードが存在することになるが、木構造のバランスは保たれる。

一定時間毎に全移動オブジェクトの過去位置データを削除し、最新の位置データを挿入することにする。全オブジェクトデータを一括削除、一括挿入する方法もあるが、MD-MOtree では Fig.5 に示すようにオブジェクトごとに削除と挿入を行う。旧データ削除時に対象オブジェクトが所属しているノード番号を取得し、そのノードを根ノードとする部分木から新データを挿入する。以下にそのアルゴリズムを示す。

[MD-MOtree 削除・挿入アルゴリズム]

移動オブジェクト O、O の最新データ(挿入対象データ)A_{new}、O の現在データ A_{now}、O の過去データ A_{old}、

S1. A_{new} が発生したとき、A_{old} に対して MD-MO tree 削除アルゴリズムを適用.

S2. A_{now} が所属するノード N から根に向かって A_{new} が入るべきノード N1 を検索する.

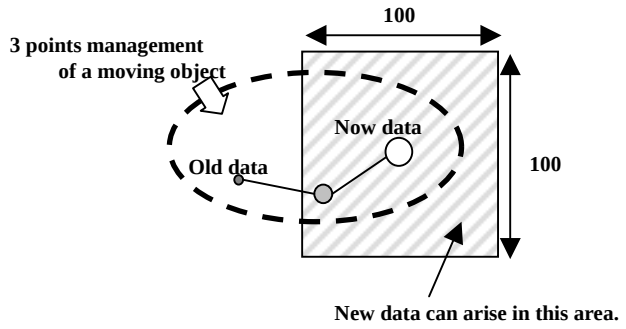


Fig.6 Test data.

S3. N1 を根ノードとする部分木から、A_{new} に対して MD-tree 挿入アルゴリズムを適用.終了.

移動オブジェクトの毎回の移動距離がそれほど大きくないと仮定すれば、発生した最新データは高い確率で現在データと同じ葉ノードによって管理されるため、従来の根ノードから挿入する方法に比べ、挿入時にたどるノード数を減らすことができる。

MD-MOtree は、MD-tree には劣るものの他の木構造と同程度のメモリ効率を実現し [11]、移動オブジェクトの頻繁なデータの挿入と削除に対して、高速かつ安定なデータ構造といえる。点データ管理において管理領域の重なりはなく、極端なメモリ効率の低下もないため、木の再構築を行う必要がない。移動オブジェクトの最新位置を長時間にわたり継続して管理するのに適している。

3.計算機実験

本章では MD-tree、簡易 MD-tree、MD-MOtree にそれぞれの挿入と削除に要する時間、検索時間について性能比較を行う。

3.1 移動オブジェクトデータの３点管理

本研究は車や電車、人の位置管理を行うようなアプリケーションへの利用を考慮し、その進行方向や速度変化を捉えるために最新データ３つによる３点管理を行う(Fig.6)。TPR-tree は１点とその位置の速度ベクトルを管理し、これによって単純な予測を行っている。３点管理を行うことで移動オブジェクトの速度変化、

方向変化などの高度な情報を持たせることができる。

3.2　テストデータ

移動オブジェクトを時間によって移動する点データで表現し、移動オブジェクトは二次元の管理領域を出ないものとする。３点管理より一定時間毎に 1/3 のデータが入れ替わる。移動オブジェクトの移動は慣性を持たせたランダム移動とし、一回の移動距離は現在位置を中心とする 100×100 の正方形内部とした(Fig.6)。挿入・削除・最近接検索実験において全体領域面積 B は、

$$B = 100 \times 100 \times \text{オブジェクト数}$$

とし、オブジェクト数の変化に対して領域を広げ、密度を一定とした（例えばオブジェクト数１万のとき、B=10000×10000)。データ構造で管理するのは移動オブジェクトデータのみで固定データは扱わない。バケット容量は 20 とする。また、図中の MD-tree with restructure は 50 回の挿入と削除ごとに木の再構築を行う MD-tree、MD-MOtree (leaf integration only)は挿入ルートの短縮は行わず、葉ノードの統合のみの MD-MOtree である。プログラムは Visual C++で作成、使用した CPU は Athlon™4 1.2GHz である。

3.3　挿入と削除の繰り返しによる性能変化

移動オブジェクト数を 10 万としたとき、挿入と削除を繰り返すことによる性能の変化を Fig.7 に示す。このとき全体領域面積は B である。MD-tree を再構築せずに挿入と削除を繰り返すと徐々に性能が劣化、挿入と削除の処理時間が増大する。これは複雑に管理領域が重なり合うことが原因である。簡易 MD-tree と MD-MOtree は繰り返される挿入と削除においても安定した処理を行うことができ、継続的な管理が可能である。

3.4　挿入・削除実験

全体領域面積 B とし、移動オブジェクトの数を 1000～100000 まで変化させた。移動オブジェクトすべてを挿入・削除するのに要した時間(CPU Time)とメモリ効率をそれぞれ Fig.8、Fig.9 に示す。CPU time は各オブジェクト数について 1000 回挿入・削除し、平均をとった。また、MD-tree は回数を重ねる度に性能が悪化してしまうため、挿入・削除 50 回ごとに木の再構築を行っている。

Fig.8 より、簡易 MD-tree、MD-MOtree の CPU time は MD-tree と比べ、オブジェクト数が多くなるほど有効であるといえ、オブジェクト数 100000 の時には 1/3 以下の処理時間となる。MD-MOtree(leaf integration only)と MD-MOtree を比べると、挿入・削除アルゴリズムによって 20%程度 CPU time が短縮されたことがわかる。Fig.9 より、簡易 MD-tree のメモリ効率は約 40%と MD-tree に比べ大きく悪化しているが、MD-MOtree は約 67%と、15%程度の低下にとどまっている。これは固定データにおける R-tree と同等程度で、実用上問題ない値である [11]。Table.1 に移動オブジェクト数 10000（３点管理よりデータ数 30000）における各木構造の構成ノード数、葉ノード数、木のレベルを示した。MD-tree は他に比べ小さい木構造を構成している。また、MD-MOtree は簡易 MD-tree に比べ、葉ノード数が 45%程度減少した。

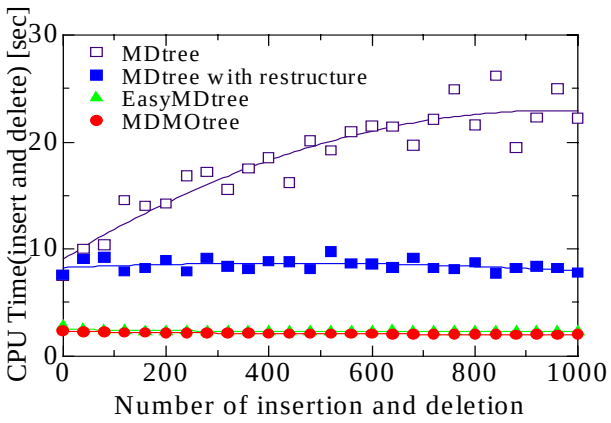


Fig.7　Performance of　repetitive insert and delete.

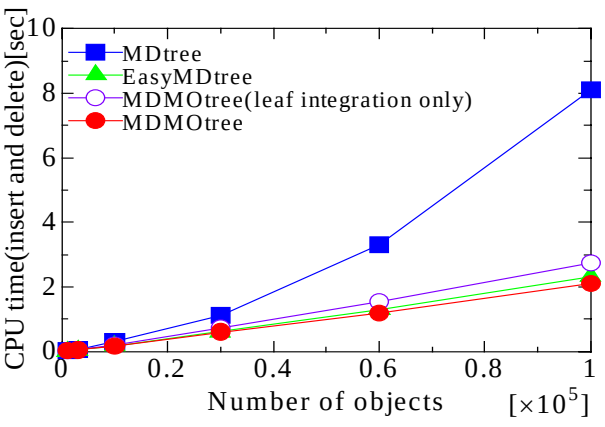


Fig.8　Insertion and deletion performance.

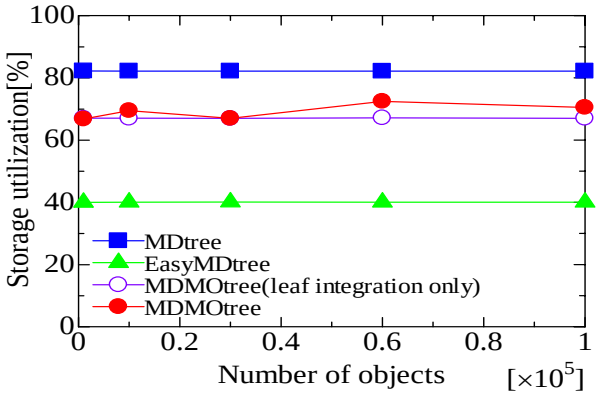


Fig.9 Storage Utilization.

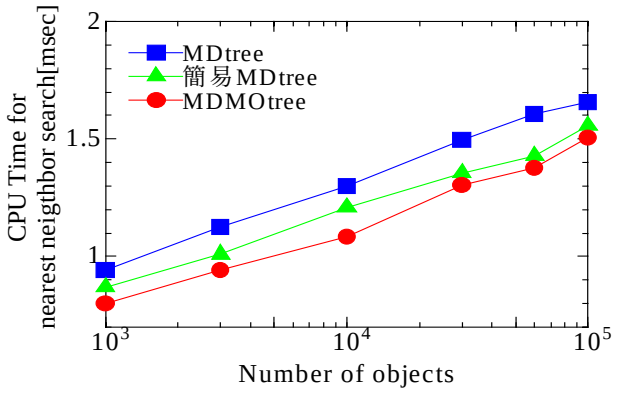


Fig.10 Nearest neighbor search performance.

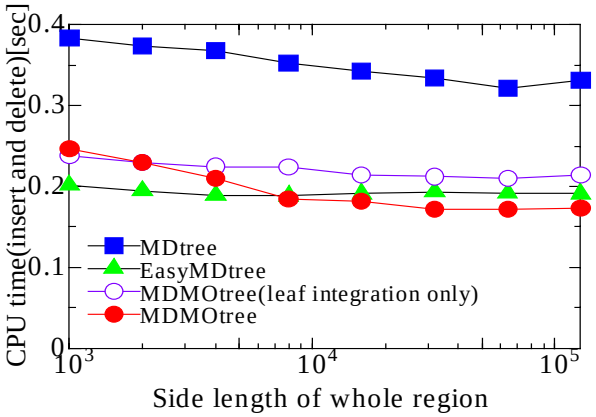


Fig.11 CPU time of insertion and deletion with changing the size of region.

Table.1 Number of node and leaf. And the level of each tree. Where the number of data = 30000(object = 10000), and page capacity = 20.

	Number of node	Number of leaf	Level of tree
MD-tree	1347	1829	9
EasyMD-tree	2804	3805	10
MD-MOtree	2693	2135	10

3.5 最近接検索実験

最近接点検索はランダムに検索点を発生させ、検索点から最も近いデータを発見するまでに要する時間を出力とした。Fig.10 より、簡易 MD-tree と MD-MOtree は MD-tree に比べ、検索時間が 5~15%短い。これは簡易 MD-tree と MD-MOtree に管理領域に重なりがなく、検索時にたどるノード数を減少できたためである。MD-MOtree は簡易 MD-tree に比べ、葉ノードが少ないため検索時間が短くなる。

3.6 管理領域面積変化実験

Fig.11 に移動オブジェクト数 10000 として全体の管理領域の大きさを変えた（移動オブジェクトの面積あたりの密度を変えた）場合の CPU time を示す。MD-MOtree に注目すると、面積が大きくなるにつれて MD-MOtree(leaf integration only)との差が大きくなっている。これは、移動距離に対して葉ノードの面積が大きいほど挿入・削除アルゴリズムが効果的になることを示している。

4.まとめ

MD-tree を拡張し、削除アルゴリズムを変更、部分木からのデータ挿入をおこなうことで、移動オブジェクト管理において有効な MD-MOtree を提案した。MD-MOtree が移動点オブジェクトを管理した場合の特長は以下のとおりである。

1）データの挿入と削除の処理が高速。
2）頻繁なデータの挿入と削除による性能劣化がなく、木の再構築の必要がない。
3）点データに対して管理領域に重なりが生じない。

MD-MOtree は MD-tree に比べ、メモリ効率は 15%程度悪化するものの、繰り返される大量のデータの挿入・削除に対して安定で、かつオブジェクト数 10 万では挿入と削除に要する時間を 1/3 以下にできた。

参考文献

1) 出来原裕順,中村泰明,移動オブジェクトを管理す

る時空間データ構造,電学論 C, 122-6, pp1052-1059,2002

2) 岸浩志,田名部淳,河野博之,Σ-tree による時空間 OLAP 技術の交通データへの適用,Data Engineering WorkShop 2002,C2-13,2002
3) 王軼群,野沢博,土方嘉徳,中谷美江,西田正吾,移動オブジェクトを対象とした時空間データ管理の一手法,電学論 C,123-6,pp1155-1165,2003
4) 塚本祐一,石川佳治,北川博之,移動軌跡ストリームからの移動統計量推定のための動的ヒストグラム構築手法について, Data Engineering WorkShop 2004, 7-C-01,2004
5) Simonas Saltenis, Chiristian S.Jensen, Scott T.Leutenegger, Mario A.Lopez, Indexing the Positions of Continuously Moving Objects, ACM SIGMOD, pp.331-342, 2000
6) Y.Tao, J.Sun, D.Papadias, Selectivity Estimation for Predictive Spatio-Temporal Queries, ACM Conf. ICDE, 2003
7) A. Guttman, R-tree: A dynamic index structure for special searching, in Proc. ACM SIGMOD, pp.49-59, 1984
8) 中村泰明, 阿部茂, 大沢裕, 坂内正夫, 多次元データの平衡木による管理-MD 木の提案-, 信学論(D), J71-D, 9, pp.1745-1752, 1988
9) J.L.Bentley, Multidimensional binary search trees used for associative searching, Common. ACM, 18, 9, pp.509-517, 1975
10) R.A.Finkel, J.L.Bentley, Quad trees: A data structure for retrieval on composite keys, Acta Inform. Vol.4, no.1, pp.1-9, 1974
11) Yasuaki Nakamura, Shigeru Age, Yutaka Osawa, Masao Sakauti, A Balanced Hierarchical Data Structure for Multidimensional Data with Highly Efficient Dynamic Characteristics, IEEE, vol.5, no.4, pp.682-693, 1993