HAREKETLİ NESNELERİN İNDEKSLENMESİ

Utku KALAY[1] **Oya KALIPSIZ[2]**

[1,2]Bilgisayar Mühendisliği Bölümü
Elektrik-Elektronik Fakültesi
Yıldız Teknik Üniversitesi, 34349, Beşiktaş, İstanbul

[1]e-posta: utku@ce.yildiz.edu.tr [2]e-posta: kalipsiz@yildiz.edu.tr

Anahtar sözcükler:Uzaysal/Konumsal veri tabanları, İndeksleme, Hareketi Nesneler

ABSTRACT

This paper discusses the recent innovations on indexing of moving objects. There have been many researches on indexing and storage issues of moving objects. This paper specifically discusses the objects that are moving over a network. An innovative index structure based on R-tree, namely MON-tree, has been thought to be a starting point for many developments, such as spatio-temporal query, kNN querying. Although, this paper only discusses the problems on this area, we hope that we will make contributions after we set up the simulation environment for the structures discussed in this paper.

1. GİRİŞ

Uzaysal/Zamansal veri tabanı sistemleri ***(S**patio-**T**emporal **D**atabase **M**anagement **S**ystems, STDBMS)*, konum ve/veya şekil özellikleri zamana bağlı olarak değişen nesnelerin modellenmesi, depolanması, erişimi ve sorgulanması fonksiyonlarını içermektedir. Bu konuda yapılan çalışmalara genel olarak baktığımızda mevcut veri tabanı sistemlerinin alt yapı olarak kullanıldığını görmekteyiz. Kullanıcının veri tipleri tanımlamasına olanak sağlayan nesne-ilişkisel veri tabanları üzerinde son senelerde tanımlanan standartlar, uzaysal/zamansal veri tabanı ve diğer çoklu ortam veri tabanlarına ortak bir zemin oluşturmaktadır.

STDBMS, birçok önemli uygulamalara olanak sağlayan bir teknolojidir. Günümüzde gerek ticari gerekse akademik düzeydeki bu uygulamalara örnek olarak Coğrafi Bilgi Sistemleri ***(G**eographic **I**nformation **S**ystem, GIS)*, çoklu ortam CAD uygulamaları, hareketli nesne takip ve sorgulama sistemleri verilebilir

Bu çalışmada, belirli bir yol haritası üzerindeki hareketli nesnelerin indekslenmesi amacıyla tasarlanan veri yapıları incelenecektir.

Örneğin, belirli bir ağ üzerinde hareket etmekte olan nesneler üzerinde bir çok farklı sorgulamalar yapılabilir. Genellikle sürekli zaman sorguları olarak incelenen bu tip sorgularda sorgu cevap kümesi dinamiktir. Örneğin, "Son 1 saat içerisinde, hastanenin $2km^2$'lik alanı içerisinden geçmiş olan ambulansların bulunması" tipindeki sorgular hareketli nesneler üzerinde sürekli zaman sorgularına örnek olarak verilebilir. Bu sorgunun cevap kümesi belirli zaman aralıklarında geçerli cevap kümelerinden oluşmaktadır[1].

İkinci bölümde konumsal özellik bilgilerinin indekslenmesine olanak sağlayan ve genel bir kabul görmüş olan R-tree veri yapısı anlatılacaktır. Üçüncü bölümde ise hareketli nesnelerin modellenmesi için tasarlanan R-tree tabanlı bir veri yapısı olan MON-tree incelenmiştir. Son bölümde ise, bu makalede ele alınan ve gelecekte planlanan çalışma hedeflerinden bazıları aktarılmıştır.

2. UZAYSAL İNDEKS YAPISI: R-tree

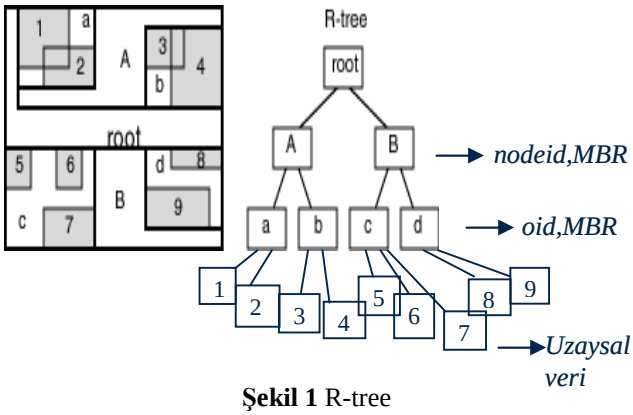
İndeksleme, bilginin depolama kapasitesini en verimli şekilde kullanarak veriye en kısa zamanda erişmeye olanak sağlar. Verinin tek boyutlu sabit veri olması durumunda B-tree, Hashing gibi geleneksel bir çok yöntem vardır.

B-tree ağacı tek boyutlu veriyi yaprak düğümlerinde sıralama düşüncesini esas aldığı için, uzaysal konuma sahip olan çok boyutlu verinin saklanması için uygun değildir. Bunun yerine R-tree isimli veri yapısı tasarlanmıştır.

2.1 R-tree:

B-tree ailesinden gelen R-tree ağacı ve türevleri uzaysal nesnelerin indekslenmesinde kullanılır. B-tree'de bilgi, belirli kuralları sağlayan dengeli bir ağaca yerleştirilirken, R-tree'de nesneler benzer kuralları sağlayan dengeli bir ağaç oluştururlar [2].

Ağacın ilk seviyesinde, minumum çevreleyen dikdörtgen *(minumum bounding rectangle, MBR)* adı verilen ve indeksin kullanımında filtreleme görevi yapan dikdörtgenler oluşturulur. Böylece düzenli bir şekilde olmayan nesneler yerine, onları çevreleyen minumum dikdörtgenler üzerinde işlem yaparak, indeks performansı önemli oranda arttırılmış olur. Örneğin Şekil 1'de, 9 adet dikdörtgenin indekslenmesi ile oluşan R-tree ağacı çizilmiştir[3]. Bu örnekte nesneler zaten dikdörtgen olduğu için birinci seviyede bulunan yaprak düğümler için MBR oluşturmaya gerek yoktur. İkinci seviyede, nesneler birbirine en yakın ve en az örtüşme olacak şekilde bir arada gruplandırılır. Bu yaklaşım, en üst seviyedeki köke kadar devam eder. Örneğin, *1 ve 2* nesneleri *a* dikdörtgeninin çocukları, *3 ve 4* nesneleri *b* dikdörtgeninin çocukları olmuştur. Böylece kökten başlayarak herhangi bir yaprak düğüme erişerek filtreleme aşaması tamamlanmaktadır. Bundan sonra yaprak düğümde bulunan gerçek nesne üzerinde işlem yapılabilir.



Şekil 1 R-tree

Örtüşmenin az olması koşulu ile R-tree sorgulamada oldukça başarılı bir ağaçtır[2]. Şekil 1'de görüldüğü gibi ara düğüm MBR'ları (A,B ve a,b,c,d) arasında bir örtüşme yoktur. Bu yüzden kök düğümden başlayan arama ile doğrudan yapraklara erişilir. Sıkışık nesnelerdeki gibi örtüşmenin fazla olduğu durumlarda, bir nesneyi ararken aranan yaprağa doğrudan inmek her zaman mümkün olmaz, bu da performansı olumsuz etkileyen bir durumdur.

R-tree veri yapısının en önemli sorunu, ağaca nesne ekleme/çıkarma işlemlerindeki yüksek maliyettir[2]. Nesne eklemede nesnenin hangi MBR'a dahil olacağı en az "ölü alana" neden olacak şekilde yapılmalıdır. Buna *"least enlargement criterion"* adı verilir. Nesne eklemede, ilgili düğüm doluysa ikiye bölünme *(split)*; nesne çıkarmada, düğümdeki nesne sayısı belirli bir sınır değerin altına düşerse birleşme *(merge)* operasyonu gerekir. Bölme ve birleşme işlemlerinin zaman gereksinimi oldukça yüksektir. Diğer bir problem, R-tree ağacının, nesnelerin ağaca eklenme sırasına göre farklılaşmasıdır. Bu bir dezavantajdır, çünkü verinin geliş sırası rastgeledir ve bu da verimi düşük indeks yapılarının oluşumuna neden olabilir.

Sonuç olarak, bilgi yenilenmesi veya çok sık ekleme/silme gerektirmeyen uygulamalarda R-tree oldukça başarılı bir veri yapısıdır.

3. HAREKETLİ NESNELERİN İNDEKSLENMESİ: MON-tree

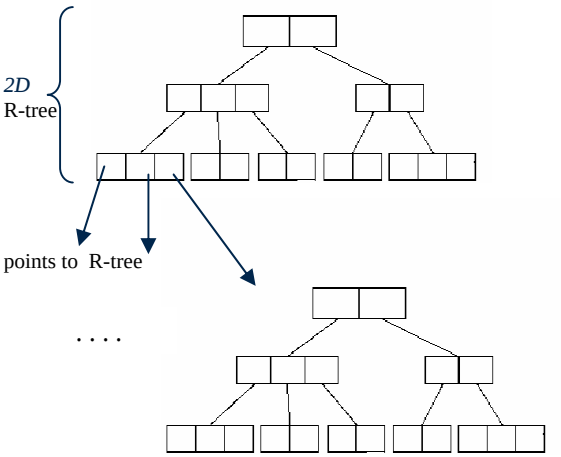
İki boyutlu ortamda hareket halindeki nesneleri, zaman boyutunun eklenmesi ile beraber 2+1=3 boyutlu veri yapısı olarak düşünebiliriz. Üç boyutlu bu veriyi R-tree ile indekslediğimiz zaman indeks performansı oldukça düşük olacaktır. Bunun nedeni, dinamik ortamın gerektirdiği yüksek yenilenme sıklığının, indeksin performansına olumsuz etkisidir. Sonuç olarak, hareketli nesnelerin tutulması için R-tree veri yapısının doğrudan kullanımı uygun değildir [4].

Hareketli nesnelerin indekslenmesi için bir çok veri yapısı tasarlanmıştır. Bu farklılığın başlıca nedeni, hareketli nesnelerin farklı uygulama alanlarının olmasıdır. Örneğin nesnelerin şekli (dikdörtgen, nokta vb gibi), kendi şekillerinin zamanla değişmesi, 3 boyutlu uzayda hareket etmesi aynı problemin farklı uygulamaları veya alt kümeleri olarak ele alınabilir. Bununla beraber sorgu tipleri de probleme ayrı bir boyut getirmektedir. Geçmiş hareketler ile ilgili sorgulamaya olanak veren modeller *(trajectory-based)* ile, gelecek hareketler ile ilgili olan indeks modelleri *(location prediction)* birbirinden farklı yaklaşımları gerektirir. Son olarak, hareketin serbest hareket olması veya bir ağ üzerinde hareket etmesi de problemin diğer önemli bir parametresidir.

3.1 MON-tree *(Moving Object Network-tree)*

Belirli bir ağ üzerinde hareket halinde olan nesnelerin bir indeks yapısında tutulması ve bu yapıya bağlı olarak geçmiş hareketler ile ilgili sorgulamaların gerçekleştirilmesi MON-tree isimli veri yapısı ile gerçekleştirilmiştir. Hareketli nesneler nokta ile modellenmiştir.

Gerçek dünyadaki bir çok uygulamada, nesne hareketleri, belirli bir yol ağı üzerinde gerçekleşir. Şehir içi/arası yollar, tren yolları gibi ağların üzerindeki hareketler bu uygulamalara örnek olarak verilebilir. Bu yol ağları üzerindeki 2 boyutlu nesnelerin indekslenmesi MON-tree isimli iki seviyeli R-tree ile gerçekleştirilebilir. Birinci seviyede yol haritasının indekslenmesi, ikinci seviyede ise her kenar üzerinde hareket eden nesnelerin indekslenmesi gerçekleştirilir[5]. Yol haritası sabit olup, kenar *(edges)* ve köşelerden *(nodes)* oluşan bir grafdır. Basit bir yaklaşım ile bu kenarların her birini çizgi ile temsil edip, 2 boyutlu R-tree ile indeksleyebiliriz. Böylece oluşan 2 boyutlu R-tree'nin her yaprak düğümünde bir çizgi olur. İkinci aşamada bu yaprak düğümlerindeki her çizgiye karşılık gelen bir adet R-tree ile bu çizgiden (kenardan) geçen nesnelerin hangi zaman aralıklarında bu kenarlardan geçtiğini indeksleyebiliriz. Şekil 2'de bu yapı görülmektedir.

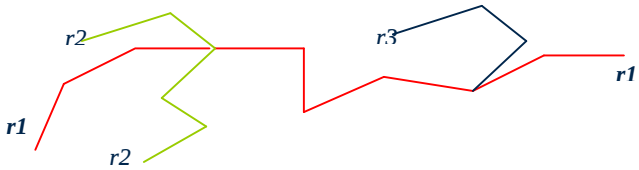


Şekil 2 MON-tree genel yapısı

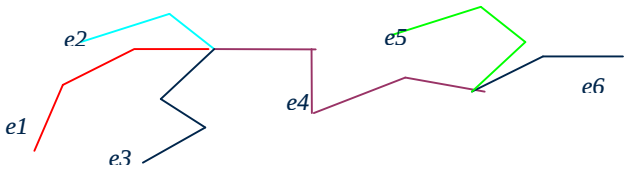
Bu noktada en önemli dezavantaj, ağdaki her kenarın çizgi ile temsil edilmesiyle, 2D R-tree'deki eleman sayısının oldukça fazla olmasıdır. Bu, ikinci seviyedeki R-tree ağaçlarının sayısını da arttırmaktadır. Şekil 2'deki ilk seviyedeki 2 boyutlu R-tree'nin kapasitesini azaltmak için, her kenar yerine, birbirine ucuca bağlı kenarlardan oluşan kenar grupları ağacın yapraklarını oluşturabilir. [6]'da bu düşünceden yola çıkılarak 2 farklı yaklaşım öne sürülmüştür: **Kenar-yönelimli** *(edge oriented)***, rota-yönelimli** *(route oriented).* Kenar-yönelimli modellemede kenarlar, iki düğüm noktası arasındaki ucuca bağlı bir grup çizgiden oluşmaktadır. Bu modelde, düğüm noktaları, ikiden çok kenara sahip kesişim noktaları olarak tanımlanır. Rota-yönelimli modellemede ise kenarlar, bir rotayı temsil eden ucuca bağlı bir grup çizgiden oluşmaktadır.

Buna göre oluşan bir grup çizgi (l_e= (p1,…, pk), p_n = *(x,y))* ile ifade edilirse, her l_e üzerindeki hareketli nesnenin hareket modeli, *T* zaman uzayını *D(G)* konum uzayına taşıyan (f : T → D(G)=E x pos, *pos* ε *(0,1))* fonksiyonu ile ifade edilebilir. Bu fonksiyondaki *pos* değişkeni, nesnenin bulunduğu kenar boyunca konumunu vermektedir. *pos*=0, nesnenin kenarın en başında olması, *pos*=1 nesnenin kenarın sonunda olması demektir. *pos* değişkeni, nesnenin kenarda yol alırken herhangi bir zamanda hangi konumda olduğu, veya hareketine kenarın hangi konumundan başladığı bilgilerini tutmaktadır.

Şekil 3a ve 3b'de, aynı harita üzerinde kenar-yönelimli ve rota yönelimli modellere göre oluşan kenar grupları görülmektedir. e_1,e_4,e_6 kenarları Şekil 3a'da ayrı ayrı 3 parça iken, Şekil 3b'de tek bir rota olarak *(r_1)* temsil edildiği görülmektedir. Böylece, yol haritasında tanımlanan her anayol, rota-yönelimli modeldeki bir rotaya karşılık gelir. Sonuç olarak, ana yollar ve şehirler arası yollar için rota yönelimli model kullanılması daha uygundur. Diğer yandan, şehir içi yolların veya ana yol dışındaki yolların modellenmesinde kenar-yönelimli modelin kullanılması daha uygundur.



Şekil 3a Kenar-yönelimli modelleme

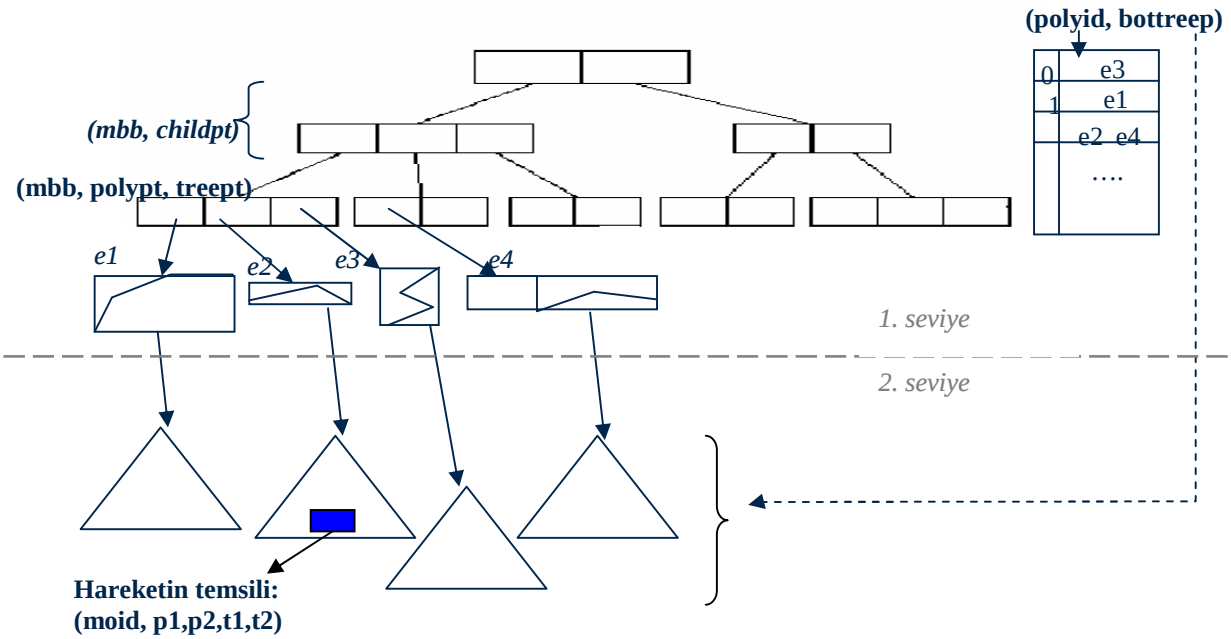


Şekil 3b Rota-yönelimli modelleme

Rota-yönelimli model ile ağaçtaki nesne sayısı azaltmış oluyor; çünkü yaprak düğümlerini rotalar oluşturmaktadır. Diğer yandan, indeks işlemleri azalır; çünkü rota olarak tanımlanan bir anayolda ilerleyen nesnelerin, o anayolu terk etme olasılığı düşük olduğundan indeks yenilenme sıklığı da azalmaktadır. Kenar-yönelimli modelde ise, hem yola karşılık gelen indeks daha büyük, hem de yenilenme sıklığının daha fazla olması kaçınılmazdır.

Yol haritasının indekslenmesi için yapılan bu iyileştirmeler ile beraber MON-tree veri yapısı Şekil 4'de gösterilmiştir[6]. Bu veri yapısı, birinci seviyede bir adet 2-boyutlu R-tree ve bir hash tablosu, ikinci seviyede ise birinci seviyedeki ağacın yapraklarının işaret ettiği 2-boyutlu R-tree ağaçlarından oluşur. MON-tree'yi oluştururken, kenar/rota ekleme, nesne hareketi ekleme olmak üzere iki tür eklemeden bahsedilir. Kenar/rota ekleme ile ağ bilgisini tutan birinci seviyedeki R-tree oluşturulur. Nesne hareketlerinin eklenmesi ile ikinci seviyedeki R-tree'ler oluşturulur. Şekil 4'de, kenar-yönelimli model (Şekil 3a) kullanarak oluşturulan e_1,e_2,e_3,e_4 kenarlarının indeks yapraklarındaki yerleşimi görülmektedir. Her kenardaki nesne hareketlerini tutmak için, *treept* alttaki 2-boyutlu R-tree'ye işaret etmektedir.

Birinci seviyede, R-tree'ye ek olarak ***polyid*** ile organize edilen ve içeriği ***(polyid, bottreept)*** olan bir *hash* tablosu vardır. Uzaysal/zamansal bir sorgunun işlenmesi sırasında, sorgunun uzaysal kısmı ile örtüşen kenarların bulunması üstteki 2-boyutlu R-tree'de gerçekleşir. Diğer taraftan, hareketli nesne ile ilgili bilgilerin indekse eklenmesi için bu bilgilerin ikinci seviyedeki R-tree'ye girilmesi, doğrudan ilgili R-tree'ye ulaşmayı gerektirmektedir. Bu nedenle, doğrudan erişime olanak veren *hash* tablosu kullanılmıştır. Bu yapı, her hareket yenilenmesinde birinci seviyedeki R-tree'ye erişme zorunluluğunu



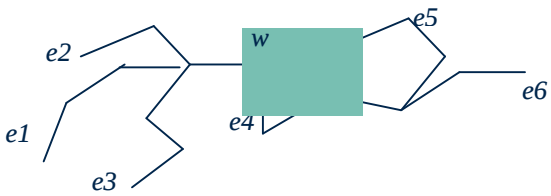
Şekil 4 MON-tree

ortadan kaldırarak performansı önemli ölçüde arttırmaktadır.

İkinci seviyedeki 2-boyutlu R-tree hareketin tutulmasını sağlayan ağaçtır. ***moid*** numaralı nesnenin hareketi, *(p1,p2)* konum aralığı ve *(t1,t2)* zaman aralığı ile temsil edilir. *p1* ve *p2*, (0≤p1,p2≤1 olmak üzere) kenar içinde hareketin olduğu konum aralığını, *t1* ve *t2* ise bu hareketin olduğu zaman aralığını ifade etmektedir. Burada tek boyutlu ve zamana bağlı bir hareketin indekslenmesi söz konusu olduğu için 2-boyutlu R-tree kullanılmıştır[6].

MON-tree veri yapısı kullanılarak bir çok sorgulama yapılabilir. Uzaysal/zamansal aralık sorguları, en yakın *k* nesnenin bulunması ***(k Nearest Neighbor, kNN)*** son zamanlarda bu konudaki en yaygın araştırma konularındandır. Aşağıdaki alt bölümde, uzaysal/zamansal aralık sorgularının çalışması üzerinde durulmuştur.

3.1.1 Uzaysal/zamansal sorgulama:

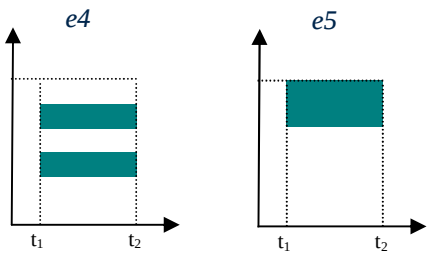


Şekil 5 Uzaysal/zamansal sorgu penceresi

Şekil 5'de şematize edilen "Belirli bir zaman aralığında, *w* bölgesinden geçmiş olan nesnelerin bulunması" tipik bir uzaysal/zamansal sorgudur. Bu sorgunun MON-tree indeks yapısı kullanılarak işlenmesi 3 aşamada gerçekleşmektedir[6].

***w* penceresi ile kesişen *polyid*'lerin bulunması:**
Bu birinci seviyedeki R-tree kullanılarak gerçekleştirilir. Bu ağaçtaki *polyid*'lerin MBB'lerinin hangisinin *w* ile kesiştiği tipik bir R-tree arama operasyonudur.

***w* penceresinin, *polyid*'lerin hangi kesimlerini içine aldığının bulunması:**
Bu gerçek *polyid*'lerin (MBBlerin değil) hangi kısımlarının *w* sorgu penceresi içinde kaldığını bulan operasyondur. Bu işlem ana hafızada yapılıyor. Sonuçta w' = { ($p1_1$, $p1_2$, t1, t2), …, (pn_1, pn_2, t1, t2) } dörtgenler kümesi elde edilir. (pi_1, pi_2), *w* penceresini kesen *i.* kenarının/rotasının, hangi kısımlarının pencereyi kestiğini ifade eder. (t1, t2) ise sorgudaki zamansal aralıktır. Böylece *w* sorgu penceresinden esas *w'* sorgu penceresi elde edilmiştir. *w'* penceresinin içerdiği dörtgenler ayrık ve sıralıdır. Şekil 6, Şekil 5'deki sorgu penceresiyle kesişen e_4 ve e_5 yollarının pencere ile kesiştikleri bölgeleri göstermektedir.



Şekil 6 Sorgu işlemenin ikinci aşamasının sonucu

w' penceresindeki dikdörtgenler ile kesişen nesne hareketlerinin bulunması:
Bu bölümde w' penceresindeki her dikdörtgen ile kesişen hareketler, ikinci seviyedeki R-tree kullanılarak bulunur. Birden çok sorgu penceresi ile kesişen nesnelerin bulunması, bilinen R-tree arama algoritmasında bazı basit değişiklikleri beraberinde getirmektedir.

MON-tree'nin en önemli problemi çizgi gruplarına ait olan MBB'lerin oluşturduğu "ölü alanlar"dır. Bu problem çizgilerin R-tree'de saklanması durumunda kaçınılmaz bir problemdir. Bununla beraber [6]'daki çalışmada, indeks hacmini azaltmak amacıyla tanımlanan kenar-yönelimli/rota-yönelimli gibi modellerin "ölü alan" problemini daha da arttıracağı belirtilmektedir.

4. İLERİYE YÖNELİK ÇALIŞMALAR:

MON-tree veri yapısının performans analizi, hareketli nesne üreteçleri tarafından üretilen hareket bilgileri ile gerçekleştirilebilir. [7]'de bir ağ üzerinde hareket eden nesnelerin üretilmesine olanak tanıyan bir sistem anlatılmıştır. [7]'deki Hareketli Nesne Üreteci, grafik ara yüzü olan bir simülasyon ortamı olup Java ortamında yazılmıştır. PostgreSQL veri tabanında saklanan yol haritası ile igili tablolar kullanılarak MON-tree oluşturulması ve üreteçten gelen hareketlerin MON-tree'ye eklenmesi bundan sonraki yapılacak çalışmalar arasındadır. Bu indeks yapısı ile yukarıda tanımlanan uzaysal/zamansal sorgular üzerinde başarılı sonuçlar elde edileceği ümidindeyiz.

Ağacın yapraklarındaki "ölü alanların" , örtüşmeye neden olarak erişim performansını düşüreceği bilinen bir problemdir. Bu nedenle, MON-tree'nin birinci seviyesindeki R-tree yerine, çizgilerin indekslenmesi için daha uygun veri yapıları kullanılabilir. Buna örnek olarak [8]'de tasarlanan PMR Quad-tree örnek olarak verilebilir. Alan parçalanmasına dayalı olarak geliştirilen Quad-tree tabanlı veri yapılarının ağın indekslenmesinde daha verimli sonuçlar vereceği düşüncesindeyiz.

Bu makalede üzerinde durulmayan önemli bir konu kNN tipindeki sorgulardır. Serbest hareket eden nesneler üzerinde, hem uzaysal hem konumsal kNN sorguları için bazı algoritmalar geliştirilmiştir. Bu algoritmalar, hareketli nesnelerin ağ üzerinde olması durumunda geçerliliğini büyük oranda yitirmektedir[9]. Bu konuda, [9]'daki Ağ-Voronoi diyagramları kullanarak yapılan deneyler oldukça başarılı sonuçlar vermiştir. Bu çalışmadan yola çıkarak kurduğumuz deney düzeneğinde kNN tipindeki sorgular için yeni yaklaşımlar elde edileceği ümidindeyiz.

5. SONUÇ:

Bu makalede hareketli nesnelerin ikincil hafızada saklanması için [6]'da tasarlanan veri yapısı açıklanmıştır. Bu veri yapısına temel teşkil eden R-tree veri yapısının temel özellikleri tartışılmıştır. Bununla beraber MON-tree indeks yapısı üzerinde erişim, arama ve ekleme operasyonları ve bunların verimi açıklanmıştır. Bu veri yapısı kullanılarak uzaysal/zamansal bir sorgunun işlenmesindeki aşamalar incelenmiştir. Bu veri yapısı ve diğer bazı makalelerden yola çıkılarak gelecekte yapılması hedeflenen çalışmalar aktarılmıştır.

6. KAYNAKLAR

[1] P. Sistla, O. Wolfson, S. Chamberlain, S. Dao, "Modeling and Querying Moving Objects", Proceedings of the Thirteenth International Conference on Data Engineering (ICDE13), Birmingham, UK, Apr. 1997, pp.422-432.

[2] Antonin Guttman: "R-trees: A Dynamic Index Structure for Spatial Searching", Proceedings of the ACM SIGMOD, 1984

[3] Oracle Spatial User Guide and Manual, *Oracle 9i Documentation*

[4] Theodoridis, Y.,Vazirgiannis, M., Sellis, T. K. "Spatio-Temporal Indexing for Large Multimedia Applications" In International Conference on Multimedia Computing and Systems (1996), pp. 441--448.

[5] Frentzos, E. "Indexing Objects Moving on Fixed Networks", *8th ISSTD, 2003*

[6] V.T. de Almeida and R.H. Güting, "Indexing the Trajectories of Moving Objects in Networks" Fernuniversität Hagen, Informatik-Report 309, March 2004.

[7] Thomas Brinkhoff. "A Framework for Generating Network-Based Moving Objects" GeoInformatica 6(2): 153-180 (2002)

[8] J. Tayeb, O. Ulusoy, and O. Wolfson, "A Quadtree Based Dynamic Attribute Indexing Method," Computer Journal, vol. 41, no. 3, pp 185-200, 1998.

[9] Kolahdouzan M., Shahabi C., "Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases" Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004