# ÖZET

"Bilgi çağı" olarak adlandırılan çağımızda, bilginin miktarı ve çeşitliliği her geçen gün artmaktadır. Gelişen Internet teknolojileri sayesinde gün geçtikçe hız kazanan bu artış, bilginin saklanması ve erişiminde yeni ihtiyaçları beraberinde getirmektedir. Hareketli nesneler üzerinde konuma dayalı servisler (LBS) gibi uzlamsal ve zamansal özelliklerin kullanıldığı gerçek zamanlı uygulamalar bu ihtiyaçların sınırlarını belirleyen teknolojilerin başında gelmektedir.

Hareketli nesne veri tabanları, zaman-uzlamsal araştırma konularının veri tabanlarında bir uygulaması olarak düşünülebilir. Bu tez çalışmasında, hareketli nesne veri tabanı sisteminin tasarımında modelleme ve bilgi saklanması/erişimi konularına ağırlık verilmiştir. Konum özelliklerinin sürekli zamanda değiştiği hareketli nesne uygulamalarında konumsal belirsizlik kaçınılmaz bir gerçektir. Bu belirsizliğin gerçeklenebilir bir model ile kontrol altında tutulması ve bunun sorgulamaya olan etkisi sorgu sonuçlarının güvenilirliğinde önemli bir yaklaşımdır. Bu amaçla, bu tez çalışmasında belirsizliğe dayalı model ve bu model üzerine geliştirilen sorgulama algoritmaları üzerinde çalışmalar yapılmıştır. Literatürde bu yöndeki çalışmaların farklı hareket senaryolarına tatbik edilmesi düşüncesiyle, yol-ağı üzerinde hareket eden nesnelerin konumsal belirsizliği modellenmiş ve bu model üzerinde $OAS^N$ zaman-uzlamsal aralık sorgulama algoritması geliştirilmiştir.

Tezdeki ikinci önemli konu erişim yapıları ile ilgilidir. Verinin diskteki organizasyonu ve erişimini düzenleyen "B-tree", "Grid-file", "hashing" gibi standartlaşmış başarılı veri yapıları, uzlamsal ve zamansal özelliklere sahip nesneler için doğrudan kullanılamamaktadır. Bu ihtiyacın neticesinde ortaya çıkan R-tree indis yapısı ve benzerleri, uzlamsal özelliklere sahip nesnelerin diskte saklanması ve erişimine olanak sağlayan başarılı veri yapılarıdır. Diğer yandan, zaman-uzlamsal uygulamalardaki ihtiyaçların çeşitliliğinin bir sonucu olarak bu tip bilginin saklanması ve erişimine yönelik indisleme çalışmaları farklı çalışma kollarında devam etmektedir. R-tree ve benzerlerine dayandırılan veri yapıları da bu çalışma kollarından birisidir ve bu tez çalışmasında hedeflenen çalışma alanıdır.

MON-tree ve Gezinge-2d R-tree yol-ağı üzerinde hareket eden nesneler için geliştirilen erişim yapılarıdır. Bu veri yapıları, hareket ortamını ve nesnelerin geçmiş zaman hareket parçacıklarını bir grup R-tree ile indislemektedir. Diğer yandan MVR-tree ve 3DR-tree gibi yapılar serbest ortamdaki hareketleri indisleyen veri yapılarıdır. Bu tezde gerçeklenen hareketli nesne veri tabanı sorgulama sistemi üzerinde, seçilen bu veri yapılarının gerçeklenmesi ve birbiriyle karşılaştırılması sonucunda zayıf ve kuvvetli yanlarının tespit edilmesi hedeflenmiştir.

Son olarak, erişim yapılarının performansını etkileyen diğer önemli bir organizasyon olan ara bellek tasarımı üzerinde çalışmalar yapılmıştır. Bu amaçla tasarlanan ve gerçeklenen **monpar** isimli ara belleğin, MON-tree veri yapısının performansını arttırdığı, farklı sorgulama karakteristiklerine sahip sorgu setleri ile yapılan deneylerde gösterilmiştir.



**Anahtar Kelimeler**: Hareketli nesne veri tabanları, indisleme, belirsizliğin modellenmesi, ara bellek tasarımı, olasılıksal sorgulama.

## ABSTRACT

The quantity and the variety of data rapidly increase in the“Information Age” which is the name of the current age. With the proliferation of Internet technologies, this increase brings about the new requirements on the data storage and access. The real-time applications using spatial and temporal data, such as Location Based Services over the moving objects, determines the coverage of these requirements.

Moving Object Databases can be thought of as a kind of database application of spatio-temporal applications. This thesis emphasizes on the data modeling and access methods of moving objects. It is an inevitable fact that there exists a location uncertainty in the applications having objects changing their locations continuously. To get more reliable query results, it is a good idea to control the uncertainty with an implementable model on which a probabilistic querying algorithm is built. With this purpose in mind, an uncertainty model and a related probabilistic querying algorithm, namely $OAS^N$, have been proposed. In fact, these models are just the reflections of the proposed models in the literature on the network-based moving scenarios.

Another contribution of the thesis is about the access methods. The immediate usage of access methods like B-tree, Grid-file indexing and hashing methods are not suitable for multidimensional data, such as spatial or spatio-temporal data. R-tree and its derivatives, well-known spatial acess methods in literature arises from this requirement. On the other hand, indexing spatio-temporal data, demanding more robust schemes due to its nature, is an ongoing research area having many branches inside. The spatio-temporal schemes developed based on the idea of R-tree structure take place in one of these subareas and are the main interest of this thesis.

MON-tree and Trajectory-2d R-tree are two schemes indexing the locations of objects moving over a network of road. Those access methods mainly organizes a group of R-tree in order to index the underlying road network and the object movements. Besides, MVR-tree and 3DR-tree are only two examples of those indexing the objects moving in free movement scenarios. All four access methods have been implemented and tested for their query performance over a spatio-temporal framework.

Lastly, the thesis takes a brief look at buffer management. **monpar**, a buffer organization specifically designed for MON-tree, has been implemented in order to increase the disk access performance. Our experiments on buffer organizations are conducted with variety of query sets, each of which has different spatio-temporal characteristics.

**Keywords**: Moving Object Databases, spatio-temporal indexing, uncertainty modelling, spatio-temporal buffer organization, probabilistic querying.

**İÇİNDEKİLER**

## SİMGE LİSTESİ

| | |
|---|---|
| $E$ | $G$ grafındaki kenarları içeren set |
| $h_a$ | $a$ uzlamsal nesnesine ait hilbert değeri |
| $G$ | Yol-ağına ait graf |
| $N$ | $G$ grafındaki düğümleri içeren set |
| $Pi(t)$ | Kısıtlamalı lojik modelde hareketi temsil eden fonksiyon |
| $U_i(t)$ | Zamana bağlı belirsizlik bölgesi |

## KISALTMA LİSTESİ

| | |
|---|---|
| ADT | Abstract Data Type |
| CAD/CAM | Computer-Aided Design/Computer-Aided Manufacturing |
| CBS | Coğrafik Bilgi Sistemleri |
| CCAM | Connectivity Clustered Access Method |
| CRR | Connectivity Residue Ratio |
| FTL | Future Temporal Logic |
| GPS | Global Positioning System |
| HR-tree | Historical R-tree |
| LBS | Location Based Services |
| LRU | Least Recently Used |
| MBB | Minumum Bounding Box |
| MBR | Minumum Bounding Rectangle |
| MON-tree | Moving Objects in Network tree |
| MOST | Moving Object Spatio-temporal |
| MVR-tree | Multi Version R-tree |
| OAS | Olasılıksal Aralık Sorgulama |
| $OAS^N$ | *N* yol-ağı üzerinde olasılıksal sorgulama |
| OLAP | Online Analytical Processing |
| ORDBMS | Object Relational Database Management System |
| OQL | Object Query Language |
| PDF | Probability Density Function |
| SaIL | Spatial Index Library |
| SQL | Structured Query Language |
| UML | Unified Modelling Language |
| UVT | Uzlamsal Veri Tabanı |
| ZVT | Zamansal Veri Tabanı |
| ZUVT | Zaman Uzlamsal Veri Tabanı |

## ŞEKİL LİSTESİ

Sayfa

## ÇİZELGE LİSTESİ

## ÖNSÖZ

Bu tez çalışmasının ortaya çıkmasında tecrübe ve değerli görüşleri ile beni yönlendiren danışman hocam Prof. Dr. Oya KALIPSIZ'a teşekkür ederim.

Çalışmamın her aşamasında destek olan YTÜ Bilgisayar Mühendisliği bölümündeki bütün arkadaşlarıma teşekkür ederim.

Eğitim ve öğretim hayatım boyunca maddi ve manevi desteğini esirgemeyen anneme, babama ve ablama da çok teşekkür ederim.

Son olarak tezi yazmam için bilgisayarını benden esirgemeyen ve kendisiyle atom zerreleri hakkında yaptığımız tartışmalarla düşünsel olarak yeni ufuklar açtığımız kardeşim Ferhat Cüce'ye de teşekkürü bir borç bilirim.

## ÖZET

"Bilgi çağı" olarak adlandırılan çağımızda, bilginin miktarı ve çeşitliliği her geçen gün artmaktadır. Gelişen Internet teknolojileri sayesinde gün geçtikçe hız kazanan bu artış, bilginin saklanması ve erişiminde yeni ihtiyaçları beraberinde getirmektedir. Hareketli nesneler üzerinde konuma dayalı servisler (LBS) gibi uzlamsal ve zamansal özelliklerin kullanıldığı gerçek zamanlı uygulamalar bu ihtiyaçların sınırlarını belirleyen teknolojilerin başında gelmektedir.

Hareketli nesne veri tabanları, zaman-uzlamsal araştırma konularının veri tabanlarında bir uygulaması olarak düşünülebilir. Bu tez çalışmasında, hareketli nesne veri tabanı sisteminin tasarımında modelleme ve bilgi saklanması/erişimi konularına ağırlık verilmiştir. Konum özelliklerinin sürekli zamanda değiştiği hareketli nesne uygulamalarında konumsal belirsizlik kaçınılmaz bir gerçektir. Bu belirsizliğin gerçeklenebilir bir model ile kontrol altında tutulması ve bunun sorgulamaya olan etkisi sorgu sonuçlarının güvenilirliğinde önemli bir yaklaşımdır. Bu amaçla, bu tez çalışmasında belirsizliğe dayalı model ve bu model üzerine geliştirilen sorgulama algoritmaları üzerinde çalışmalar yapılmıştır. Literatürde bu yöndeki çalışmaların farklı hareket senaryolarına tatbik edilmesi düşüncesiyle, yol-ağı üzerinde hareket eden nesnelerin konumsal belirsizliği modellenmiş ve bu model üzerinde $OAS^N$ zaman-uzlamsal aralık sorgulama algoritması geliştirilmiştir.

Tezdeki ikinci önemli konu erişim yapıları ile ilgilidir. Verinin diskteki organizasyonu ve erişimini düzenleyen "B-tree", "Grid-file", "hashing" gibi standartlaşmış başarılı veri yapıları, uzlamsal ve zamansal özelliklere sahip nesneler için doğrudan kullanılamamaktadır. Bu ihtiyacın neticesinde ortaya çıkan R-tree indis yapısı ve benzerleri, uzlamsal özelliklere sahip nesnelerin diskte saklanması ve erişimine olanak sağlayan başarılı veri yapılarıdır. Diğer yandan, zaman-uzlamsal uygulamalardaki ihtiyaçların çeşitliliğinin bir sonucu olarak bu tip bilginin saklanması ve erişimine yönelik indisleme çalışmaları farklı çalışma kollarında devam etmektedir. R-tree ve benzerlerine dayandırılan veri yapıları da bu çalışma kollarından birisidir ve bu tez çalışmasında hedeflenen çalışma alanıdır.

MON-tree ve Gezinge-2d R-tree yol-ağı üzerinde hareket eden nesneler için geliştirilen erişim yapılarıdır. Bu veri yapıları, hareket ortamını ve nesnelerin geçmiş zaman hareket parçacıklarını bir grup R-tree ile indislemektedir. Diğer yandan MVR-tree ve 3DR-tree gibi yapılar serbest ortamdaki hareketleri indisleyen veri yapılarıdır. Bu tezde gerçeklenen hareketli nesne veri tabanı sorgulama sistemi üzerinde, seçilen bu veri yapılarının gerçeklenmesi ve birbiriyle karşılaştırılması sonucunda zayıf ve kuvvetli yanlarının tespit edilmesi hedeflenmiştir.

Son olarak, erişim yapılarının performansını etkileyen diğer önemli bir organizasyon olan ara bellek tasarımı üzerinde çalışmalar yapılmıştır. Bu amaçla tasarlanan ve gerçeklenen **monpar** isimli ara belleğin, MON-tree veri yapısının performansını arttırdığı, farklı sorgulama karakteristiklerine sahip sorgu setleri ile yapılan deneylerde gösterilmiştir.



**Anahtar Kelimeler**: Hareketli nesne veri tabanları, indisleme, belirsizliğin modellenmesi, ara bellek tasarımı, olasılıksal sorgulama.

**ABSTRACT**

The quantity and the variety of data rapidly increase in the"Information Age" which is the name of the current age. With the proliferation of Internet technologies, this increase brings about the new requirements on the data storage and access. The real-time applications using spatial and temporal data, such as Location Based Services over the moving objects, determines the coverage of these requirements.

Moving Object Databases can be thought of as a kind of database application of spatio-temporal applications. This thesis emphasizes on the data modeling and access methods of moving objects. It is an inevitable fact that there exists a location uncertainty in the applications having objects changing their locations continuously. To get more reliable query results, it is a good idea to control the uncertainty with an implementable model on which a probabilistic querying algorithm is built. With this purpose in mind, an uncertainty model and a related probabilistic querying algorithm, namely $OAS^N$, have been proposed. In fact, these models are just the reflections of the proposed models in the literature on the network-based moving scenarios.

Another contribution of the thesis is about the access methods. The immediate usage of access methods like B-tree, Grid-file indexing and hashing methods are not suitable for multidimensional data, such as spatial or spatio-temporal data. R-tree and its derivatives, well-known spatial acess methods in literature arises from this requirement. On the other hand, indexing spatio-temporal data, demanding more robust schemes due to its nature, is an ongoing research area having many branches inside. The spatio-temporal schemes developed based on the idea of R-tree structure take place in one of these subareas and are the main interest of this thesis.

MON-tree and Trajectory-2d R-tree are two schemes indexing the locations of objects moving over a network of road. Those access methods mainly organizes a group of R-tree in order to index the underlying road network and the object movements. Besides, MVR-tree and 3DR-tree are only two examples of those indexing the objects moving in free movement scenarios. All four access methods have been implemented and tested for their query performance over a spatio-temporal framework.

Lastly, the thesis takes a brief look at buffer management. **monpar**, a buffer organization specifically designed for MON-tree, has been implemented in order to increase the disk access performance. Our experiments on buffer organizations are conducted with variety of query sets, each of which has different spatio-temporal characteristics.

**Anahtar Kelimeler**: Moving Object Databases, spatio-temporal indexing, uncertainty modelling, spatio-temporal buffer organization, probabilistic querying.

# 1. GİRİŞ

Veri mühendisliği, geleneksel (alfa sayısal) ve geleneksel-olmayan (çoklu-ortam) veri tipinde bilgi yığınlarının saklanması ve erişilmesinde, yer ve zaman yönünden verimli yöntemleri araştıran bir mühendislik dalıdır. Bu mühendisliğin gelişimi sürecinde karşımıza çıkan uzlamsal (*spatial*) veya zaman-uzlamsal (*spatio-temporal*) tipte veri yığınları, günümüzde birçok ticari uygulamada kullanılmakla beraber, güncel birçok araştırma projesinin ortaya çıkmasına neden olmuştur. Zaman-uzlamsal veri tabanı yönetim sistemleri, zaman-uzlamsal bilginin saklanması ve erişilmesine olanak sağlayan sistemler olup, halen gelişmekte olan kapsamlı bir araştırma konusudur.

Bu tezde, ağ-kısıtlamalı ortama uygun olarak tanımlanan hareket, güncelleme ve belirsizlik modelleri kapsamında üretilen geçmiş zaman konumlarının indislenmesi ve bu indis yardımı ile bazı zaman-uzlamsal sorgulamaların gerçeklenmesi esas alınmıştır. Modelleme kapsamında, ağ-kısıtlamalı hareket ortamları için yeni bir belirsizlik modeli tanımlanmıştır. Tanımlanan belirsizlik modelinde, hareket modeline bağlı olarak *sürekli çizgisel bölge* ve *ayrık çizgisel bölge* olmak üzere iki farklı (zamana bağlı) belirsizlik bölgesi tanımlanmıştır. Daha sonra, yol-ağı üzerinde tanımlanan bu belirsizlik modeli üzerinde, zaman-uzlamsal olasılıksal aralık sorgulama modelleri tanımlanmıştır. Bu modellerden bir kısmı (Kalay ve Kalıpsız, 2005)'de yayınlanmıştır.

İndisleme kapsamında ise, geçmiş zaman hareket bilgilerini saklayan veri yapılarının tasarımı ve gerçeklenmesi ve bu yapıların karşılaştırılmaları sonucunda bazı iyileştirme yöntemleri üzerinde çalışılmıştır. Bu yönde bir çalışma için ilk olarak, mevcut yapıların incelenmesi ve gerçeklenmesi yaklaşımında bulunulmuştur. Literatürde, ağ-kısıtlamalı hareket ortamları için geliştirilen, bulabildiğimiz üç farklı erişim yapısı vardır. Bunlar, Pfoser ve Jensen (2003) tarafından geliştirilen ve boyut dönüşümüne dayandırılan Gezinge-2d R-tree, Frentzos (2003) tarafından geliştirilen FNR-tree ve De Almeida ve Güting (2005) tarafından geliştirilen MON-tree isimli erişim yapılarıdır. Bunlardan MON-tree, yapısal olarak FNR-tree'ye benzemekle beraber daha başarılı bir yapı olduğu (De Almeida ve Güting 2005)'de gösterilmiştir. Bununla beraber, MON-tree'nin gerek Gezinge-2d R-tree ve gerek Tao ve Papadias (2001) tarafından serbest hareket ortamı için geliştirilen MVR-tree gibi başarılı diğer erişim yapılarından hangi durumlarda daha başarılı veya zayıf olduğu bu tezde incelenen bir konudur. Diğer yandan Theodoridis vd. (1996) tarafından geliştirilen 3DR-tree, geleneksel R-tree uzlamsal indisinde harekete ait 3 boyutlu gezingeleri doğrudan saklamaya yönelik bir uygulamadır. Bu tez çalışmasında, bahsedilen üç ağ-kısıtlamalı indisten ikisi, MON-tree ve Gezinge-2d R-tree ile

beraber, serbest ortam hareketleri için tanımlanan MVR-tree ve 3DR-tree erişim yapıları gerçeklenmiştir. Performans karşılaştırmaları zaman-uzlamsal sorgulamalar için yapılmıştır.

İndisleme kapsamında ikinci olarak, erişim yapılarının performanslarının belirlenmesinde, Leutenegger ve Lopez'in (1996) ifade ettiği "*sorgu için istenen disk sayfa sayısının, performans için yeterli bir kriter olamayacağı*" fikrinden hareketle, harekeli nesne indis yapılarının performans iyileştirmesi için, ara bellek tasarımı üzerinde çalışılmıştır. Uzlamsal indis yapıları için tanımlanan ara bellek organizasyonlarının (Brinkhoff, 2002a; Kahveci vd., 2000; O'Neil vd., 1993; Sacco, 1997) incelenmesinden sonra, hareketli nesne indisi olan MON-tree'nin erişim performansının arttırılmasını hedefleyen **monpar** isminde yeni bir ara bellek tasarlanmıştır. Bu ara bellek organizasyonundaki sayfa yeniden yerleştirme algoritması, Brinkhoff'un (2002a) geliştirdiği uzlamsal içeriğe dayalı ara belleğe yönelik algoritmanın, MON-tree'nin yapısal özelliklerine uyarlanmasıyla oluşturulmuştur. **monpar'**ın hangi sorgulama koşulları altında LRU (*Least Recently Used*) veya Öncelikli (*Priority*) gibi klasik ara bellek organizasyonlarından daha iyi sonuç verdiği yapılan deneyler ile gösterilmiştir. Bu sonuçlar (Kalay ve Kalıpsız, 2006)'da sunulmuştur.

Önceki paragraflardaki fikirlerin düşünsel düzeyden bilgisayar ortamında gerçek uygulamaya aktarılmasının, bahsedilen fikirlerin akla gelmesinden daha zor olduğu açıktır. Zaman-uzlamsal prototip sistemin gerçeklenmesi için, sistemin gelişim sürecine faydalı olacağı düşünülerek, mümkün olduğunca yüksek seviyeli program geliştirme ortamları tercih edilmiştir. Bu noktadan hareketle sistem, Hadjieleftheriou (web sitesi) tarafından halen geliştirilmekte olan SaIL *(Spatial Index Library)* isimli Java kütüphanesi üzerinde gerçekleştirilmiştir. Gerçekleştirilen zaman-uzlamsal, hareketli nesne sisteminin ileriye yönelik birçok çalışmada deney ortamı olarak kullanılmaya elverişli hale geldiği düşünülmektedir.

Bu bölümde son olarak uzlamsal ve zaman-uzlamsal veri tabanlarının tanımlanması ve bu çalışma konularında yeni yaklaşımlara olan ihtiyaçlar tespit edilecektir.

### 1.1 Zaman-uzlamsal Veri Tabanları

Zaman-uzlamsal veri tabanları, uzlamsal veri tabanlarının yaklaşık 25 senelik tarihsel sürecinin bir neticesi olarak günümüzde gerek akademik gerek ticari olarak önemli bir yere sahiptir. Uzlamsal veri tabanları, karmaşık çok boyutlu uzlamsal veriye, gelişmiş veri yapıları ve algoritmaları ile gerçek zamanda erişmeye imkan sağlayan veri tabanlarıdır. Bu kapsamda,

verinin modellenmesi, kullanımı kolay sorgulama dilinin geliştirilmesi ve kalıcı saklama biriminde yüksek performans erişim yapılarının tasarlanması gibi konular bu çalışma alanındaki önemli konu başlıklarından bazılarıdır. Bu yöndeki çalışmaları tetikleyen gerçek hayattaki uygulamalara, başta Coğrafik Bilgi Sistemleri (CBS) olmak üzere, CAD/CAM (*Computer-Aided Design/Computer-Aided Manufacturing*) uygulamaları, çevrimiçi analitiksel işlem uygulamaları OLAP (*Online Analytical Processing*) ve veri madenciliği örnek olarak gösterilebilir ( Shekhar ve Chawla, 2003).

Zaman-uzlamsal nesne ise, konum ve/veya şekil gibi uzlamsal özellikleri, ayrık veya sürekli zamanda değişen nesnelerdir. Diğer bir ifade ile zaman-uzlamsal nesne, uzlamsal nesnenin, eklenen zamansal özellikler ile geçmiş, anlık ve yakın gelecek durumlarını kapsayan hali olarak nitelendirilebilir. Bu durumda, uzlamsal nesne veri tabanlarında yapılan çalışmalar ve kullanılan yöntemler; zaman-uzlamsal veri tabanı kapsamındaki araştırmalara da yön vermektedir. Uzlamsal nesnenin veri tabanına eklenme zamanı, silinme zamanı, nesnenin herhangi bir konumda bulunma zamanı, zaman-uzlamsal veri tabanında saklanması gereken zamansal özelliklere örnek olarak verilebilir. Zaman-uzlamsal uygulamalara genel olarak, hareketli nesne takibi, trafik kontrol, uçuş yönetimi, hava tahmini gibi konuma-dayalı servisler LBS*(location-based services*) örnek olarak verilebilir (Erwig vd., 1999). Daha kapsamlı uygulamalar Çizelge 1.1'de, 2-boyutlu uzlamsal uzay için listelenmiştir. 3-boyutlu uzayda, benzer örnekler verilebilir. Tablodaki son kolon, modelleme ile ilgili olup, zaman-uzlamsal nesne *(uzlamsal tip, zamansal tip)* olmak üzere 2+1 boyut ile modellenmiştir. Örneğin, *(nokta, nokta),* 2 boyutlu uzayda bir nokta ve zamanda bir nokta ile modellemeyi ifade etmektedir.

Çizelge 1.1'deki son iki örnek, değişimin sürekli zamanda olduğu, diğer örnekler ise değişimin ayrık zamanda olduğu zaman-uzlamsal uygulamalara örneklerdir. Buna göre, zaman-uzlamsal veri tabanlarının bir alt grubu olarak düşünülebilen *hareketli nesne veri tabanları*, özellikleri sürekli zamanda değişen nesnelerin geçmiş zaman bilgilerini, şimdiki zaman bilgilerini ve/veya yakın gelecekteki bilgilerini saklayan ve bu bilgi üzerinde sorgulamaya olanak sağlayan veri tabanları olarak tanımlanabilir (Güting ve Schneider, 2005).

Çizelge 1.1 Zaman-uzlamsal uygulama örnekleri (Güting ve Schneider, 2005)

| **Uygulama** | **Örnek bir olay** | ***Model*** |
|---|---|---|
| Belirli bir zamanda ve yerde olan olaylar | depremler, volkanik patlamalar, uçak kazaları | *(nokta,nokta)* |
| Olaylar serisi | geçen sene meydana gelmiş olan volkanik olaylar | *(nokta, nokta dizisi)* |
| Belirli bir zaman geçerliliğini koruyan olaylar | tarihte bir zaman kurulmuş sonra harab edilmiş olan şehirler | *(nokta, aralık)* |
| Belirli aralıklarda sabit konumlu olaylar | bir ülkenin değişen başkentleri, bir gezginin seyahati boyunca konakladığı yerler. | *(nokta, aralık dizisi)* |
| Belirli bir tarihte ortaya çıkan bölgesel olaylar | uzun süreçte ortaya çıkan bir orman yangını | *(bölge, nokta)* |
| Bölgesel olaylar serisi | uzun süreli bir gözlemde olimpiyatların yapıldığı bölge | *(bölge, nokta dizisi)* |
| Belirli bir zaman geçerliliğini koruyan bölgesel olaylar | trafik kazası sonrası bir süre trafiğe kapalı olan bölge. | *(bölge, aralık)* |
| Belirli aralıklarda sabit konumlu bölgesel olaylar | ülke sınır haritalarının değişimi | *(bölge, aralık dizisi)* |
| Hareketli nesneler | bir odada hareket eden atom zerreleri, bir şehir yol ağında hareket eden araçlar | *hareketli nokta* |
| Boyutları değişen hareketli nesneler | bir orman yangınının gelişimi, bir videoda nesneler | *hareketli bölge* |

Hareketli nesne veri tabanını mevcut veri tabanlarından ayıran birçok farklılık vardır. Bu farklılıkları, bir veri tabanının ana modüllerini oluşturan modelleme, sorgulama ve indislemede görmekteyiz.

### 1.1.1 Modellemede Yeni Yaklaşımlara Olan İhtiyaç

Bu konuda en başta gelen ihtiyaç, dinamik ortamı temsil edebilen yeni tiplerin tanımlanmasıdır. Örneğin, Güting'e (1994) göre, "int", "double", "nokta" gibi tipler için kullanılan modellerin, ortamın dinamik olması ile yeterliliğini yitirmesi kaçınılmazdır. Bilindiği gibi, statik yapıda işleyen geleneksel bir veri tabanında saklanan bir bilgi, kullanıcı tarafından açıkça değiştirilmediği sürece durumunu (değerini) korur. Hareketli nesne veri tabanında ise, saklanan bir nesneye ait konum bilgisi sürekli zamana bağlı olarak değişmektedir. Örneğin, hareketli bir nesnenin $t$ zamanında öklit uzayında (x,y) konumunda olması bilgisi, $t_1 > t$ zamanında geçerliliğini kaybedecektir. Bununla beraber, nesnelerin değişen konum bilgilerinin her anda, açık güncellenmeleri (*explicit update*) için birçok engel vardır. Örneğin, haberleşme kanalının bir kısıtlaması olmadığı varsayılsa bile, bütün hareketli

nesnelerden her anda gelen konum güncelleme bilgisini bir veri tabanının gerçek zamanda işleyebilmesi mümkün değildir (Wolfson vd., 1998). Bu nedenle hareketli nesne konumlarının gerçeklenebilir bir modele uygun olarak veri tabanında saklanması gerekmektedir.

### 1.1.2 Sorgulamada Yeni Yaklaşımlara Olan İhtiyaç

Sorgulamada ise en önemli ihtiyaç, zaman-uzlamsal ortamın gerektirdiği yeni sorgu tiplerinin tanımlanabilmesi ve gerçek zamanda işlenebilmesidir. Wolfson (1998), mevcut *SQL* veya *OQL* gibi alışılmış sorgulama dilleri ile zamana bağlı nesneler üzerinde sorgulama yapılabilirse de; bunun oldukça kullanışsız olacağını belirtmiştir. Çünkü bu dillerin zamansal operatörleri yoktur. Mevcut zamansal olmayan sorgu dilinin, zamansal operatörler ile genişletilip kullanışlı sorgulama dillerinin geliştirilmesine ihtiyaç vardır.

İkinci olarak, zamanın işin içine girmesiyle sorgu çeşitliliği de artmaktadır. Bu farklılaşma en başta, sorgulamanın, nesnelerin geçmiş uzlamsal özellikleri veya yakın gelecek uzlamsal özellikleri üzerine olması şeklinde çeşitlenebilir. Tarihsel veri tabanları (*historical databases*), nesnelerin geçmiş zaman durumları üzerine zaman-uzlamsal sorgulamaya yönelik sistemlerdir. Diğer grup ise, gelecek zamana ait tahmini zamansal/uzlamsal özellikler üzerinde sorgulamaya olanak sağlayan sistemlerdir.

Üçüncü olarak, sorgu sonuçlarının zamana bağımlılığı yeni bir "sürekli sorgulama" (*continuous querying*) tanımını karşımıza çıkarmaktadır. Klasik veri tabanlarında, sürekli zamanda işlenen sorguların, tekrar çalıştırılması tetikleyiciler (*triggers*) ile belirlenen basit kurallara bağlıdır. Örneğin, sorgu ile ilgili özelliklerdeki güncelleştirmeler sorgunun yeniden çalıştırılması için yeterli bir tetikleyici unsurdur. Zaman-uzlamsal sürekli sorgularda ise yeniden çalıştırmanın belirli bir model ile gerçekleşmesi zorunludur; aksi takdirde, sorgunun gereksiz çalıştırılması söz konusu olabilir (Sistla vd.,1997). Örneğin, hareket halindeki bir araçta, sürücünün "*5 km içerisindeki en yakın otelleri*" öğrenmek istediğini düşünelim. Sorgunun, sonuçların sürekli değişmesi nedeniyle sürekli çalıştırılması yerine; sadece sorgu sonuçlarının değişmesinin olası olduğu durumlarda çalıştırılması, bazı stratejilerin geliştirilmesini gerektirmektedir.

Son olarak, diğer önemli bir konu, hareketli nesnelerin modellenmesinin sonucu olarak ortaya çıkan belirsizliğin sorgu sonuçlarına olan yansımasıdır. Wolfson vd.(1999) ve Cheng vd. (2004), tanımladıkları olasılıksal sorgulama modelinde, sorgu sonuçlarını belirli olasılıklar ile ifade ederek, yanlış yakalama (*false hits*) veya ıskalama (*false misses*) problemine çözüm getirmişlerdir. Bu olasılıkların belirlenmesi, sorgunun işlenmesinde konuma bağlı olarak

istatistiksel modellerin geliştirilmesini gerektirmektedir.

### 1.1.3 İndislemede Yeni Yaklaşımlara Olan İhtiyaç

İndis, veri tabanındaki kayıtların diskteki adreslerinin tayin edilmesi amacıyla tasarlanan veri yapısıdır. Bu veri yapısı genel olarak, kaydın içerdiği anahtarı kayıt ile ilişkilendirmekten ibarettir; öyle ki bu küçük boyutlu anahtarlara indis yapısı ile erişim, kayıtları veri tabanı dosyasında aramaktan çok daha verimlidir. *İndis yapısı,* diğer bir ifadeyle *erişim yapısı,* sabit diskteki veri yığınına belirli bir düzen ile erişmeye olanak sağlamaktadır. Büyük bilgi yığınlarının sabit diskteki organizasyonu, yer verimliliği ve erişim performansını belirleyen en önemli etkendir.

Verinin tek boyutlu ve sabit veri olması durumunda B-tree, hesaba dayalı adresleme (*hashing*) ve benzeri birçok klasik yöntem vardır. Diğer yandan, uzlamsal veri gibi çok boyutlu verinin, tek boyutlu bilgi gibi sıralanamaması, yeni erişim yapılarının gelişmesinde temel tetikleyici unsurdur. Uzlamsal erişim yapıları kapsamında, R-tree, Grid-file, uzlamsal-dolgu eğrileri (*space-filling curves*) gibi kendi içinde çeşitlenen birçok farklı indis yapısı tasarlanmıştır (Shekhar ve Chawla, 2003).

Zaman-uzlamsal veri tipindeki bilginin, sabit disk gibi erişimi yavaş olan bir ortamda devamlı güncellenerek saklanması söz konusu olduğunda, depolamada daha karmaşık ve güçlü indis yapılarına olan ihtiyacın önemi anlaşılır. Bu noktada ilk akla gelen problem, sürekli güncellenen (veya belirli bir modele bağlı olarak güncellenen) uzlamsal özelliklerin, sabit diskteki indis yapısında eşzamanlı güncellenmesi ile ortaya çıkan performans problemidir. Bu nedenledir ki; R-tree gibi uzlamsal indis yapıları, zaman-uzlamsal uygulamaların çoğunda doğrudan kullanılamamaktadır. Diğer yandan, R-tree'nin zaman bilgisi ile beraber 2+1-boyutlu zaman-uzlamsal nesneleri indislediğini düşündüğümüzde karşılaşılan diğer bir problem, R-tree'nin zaten mevcut olan ölü alan ve örtüşme probleminin artan boyut sayısıyla beraber daha büyük değerlere çıkmasıdır. Yapılan deneylerde, D>5 boyutlu veriyi indisleyen R-tree'nin erişim performansının, sıralı erişimden daha düşük olduğu gözlenmektedir. Diğer uzlamsal indislerde karşılaşılan benzer problemler yeni indis yapılarına olan ihtiyacı açıklamaktadır.

Güçlü indis yapılarına olan yeni ihtiyaçlar, zaman-uzlamsal sorgu çeşitliliğinin bir sonucu olarak da ortaya çıkmaktadır. Çünkü indis yapıları, bazı sorgu tiplerini desteklerken, diğerlerinde kullanılamamaktadır. Bütün bunların yanında, zaman-uzlamsal verinin tipi, hareket ortamının özellikleri, zamanın gerçek veya işlem zamanı olarak ele alınması gibi diğer

farklı parametreler ile geliştirilecek indis yapısının özellikleri belirlenir (Güting ve Schneider, 2005).

Diğer önemli bir konu, zaman-uzlamsal uygulamalar ile ortaya çıkan belirsizliğin, indis yapılarına olan etkisidir. Sorgulamada, belirsizliğin indis yapısına entegrasyonu, yüksek seviyelerde budamalara (*pruning*) olanak tanıması ölçüsünde başarılı olacaktır. Örneğin, Cheng vd. (2004) nesnelere ait belirsizlik bilgisinin zaman-uzlamsal indisin yapısal özelliklerini bozmadan yardımcı bir bilgi olarak indis düğümlerine eklenmesine dayanan bir model geliştirmiştir. Böylece, indisin yaprak düğümlerine ulaşmadan, üst seviyelerdeki yardımcı belirsizlik bilgisi sayesinde yapılan budamalar ile olasılıksal sorgu performansı iyileştirilmiş olur.

## 1.2 Tezin İçeriği

Hareketli nesne veri tabanlarında, sırasıyla modelleme ve indisleme konuları Bölüm 2 ve Bölüm 3'de ana başlıkları ile incelenmektedir. Bu bölümler literatürde yapılan en son çalışmalardan örnekler içermektedir. Bölüm 4, bu tez çalışmasında geliştirilen hareketli nesne sisteminin tasarımında kullanılan modeller, geliştirilen sorgulama algoritmaları ve gerçeklenmesi hedeflenen erişim yapılarının belirlenmesi gibi konu başlıklarını içermektedir. Bölüm 4'de, son olarak, disk sayfası içeriğine bağlı olarak geliştirilen ara bellek organizasyonlarının tasarımı incelenmiştir. Bölüm 5 ise, sistemin gerçeklenmesinden sonra erişim yapılarının ve ara bellek performansının ölçülmesine yönelik simülasyon sonuçlarının değerlendirilmesini kapsamaktadır.

# 2. HAREKETLİ NESNELERİN MODELLENMESİ

Bu bölüm hareketli nesnelerin modellenmesi konu kapsamında zamansal-uzlamsal veri entegrasyonu ve literatürde karşılaşılan modelleme çalışmalarından örnekler içermektedir. Son olarak, hareket ortamlarının sınıflandırılması ve bu ortamlar için uygulanabilecek modeller üzerinde durulmaktadır.

## 2.1 Zamansal Uzlamsal Entegrasyonda Karşılaşılan Problemler

Yakın geçmişe kadar, Uzlamsal Veri Tabanında (UVT) ve Zamansal Veri Tabanında (ZVT) modelleme konusundaki araştırmaların çoğunlukla birbirinden bağımsız olduğu görülmektedir. Zaman-uzlamsal sistemin modellenmesinde ise bu çalışmaların ortak bir çizgide buluşması hedeflenmektedir. Bu bölümde bu ihtiyacın nedenleri ile beraber, bahsedilen entegrasyonda karşılaşılan problemlere değinilmiştir.

Güting (1994) UVT'de nesne geometrilerini soyut veri tipleri, ADT (*Abstract Data Type*) ile temsil etmiştir. Bu tipler ilişkisel, nesne-yönelimli, nesne-ilişkisel veya diğer veri tabanı veri modellerinde bir özellik olarak doğrudan kullanılabilir. Örneğin, Nesne-İlişkisel Veri Tabanı Sistemleri, ORDBMS (*Object Relational Database Management System*), uzlamsal veri yapıları ve bu yapılar üzerindeki sorguları yüksek performans ve başarımda gerçekleştirmek için uygun sistemlerdir. Genel olarak, bu sistemler uzlamsal veriyi ayrı bir modülde sorgular. Ticari uygulamalarda bu modüller, *Oracle Spatial Data Cartridge, Illustra Spatial Data Blade, ESRI Spatial Data Engine, IBM Spatial Extender* gibi isimler almaktadır. Diğer taraftan, ZVT, nesnelerin anlık bilgileri yanında nesnenin tarihçesini de tutmak için modeller kullanır. Bunun için iki farklı yaklaşım vardır: Geçerli zaman modeli (*valid time*), işlem zaman modeli (*transaction time*). Geçerli zaman modeli, "gerçek dünyanın" geçmiş durumlarını saklamayı esas tutarken, işlem zaman modeli ise "veri tabanının" geçmiş durumlarını saklamaktadır.

Zaman-uzlamsal veri tabanları (ZUVT), dolayısıyla hareketli nesne veritabanları, konum ve zamanın entegrasyonuna, veri tabanının birçok modülünde (sorgulama dili, indisleme, optimizasyon gibi) ihtiyaç duymaktadır (Erwig vd., 1999). Çünkü ZUVT, sadece konum veya sadece zaman aralığı değil, bunlarla beraber hız ve ivme gibi zaman-uzlamsal kavramları da kullanmaktadır. Bu nedenledir ki; UVT-ZVT entegrasyonunda, vektör çarpımı şeklinde her iki tarafın özeliklerinin doğrudan birleştirilmesi yeterli olmayacaktır.

### 2.1.1 ZVT, Zaman-Uzlamsal Uygulamalarda Niçin Yeterli Değildir?

Zamansal veri tabanının, "uzlamsal birleştirme ve indisleme tekniklerini sağladıktan sonra, uzlamsal nesneleri diğer nesneler gibi tutması" fikri basit olmakla beraber, bazı sakıncaları olan bir yaklaşımdır. Öncelikle, bu yaklaşım Çizelge1.1'de son iki satırdaki hareketli nesne uygulamaları için yetersiz kalacaktır. Çünkü bu uygulamalar, geometrilerin veya konumların sürekli zamanda değişiminin modellenmesini gerektirmektedir ki; ZVT, sürekli zamandaki hareketlerin saklanabilmesi için yeterli değildir. İkinci olarak, zaman-uzlamsal topluluk sorgularında *(spatiotemporal aggregate queries),* verim oldukça düşük olacaktır. Bu tip sorguların, ZVT' deki gibi zamana bağlı satırlar (*fact*) üzerinden çalışması yerine, UVT'deki gibi zamana bağlı ADT ile modellenen nesne üzerinden yapılması daha verimlidir. Son olarak, çok sık yenileme gerektiren uygulamalarda yer verimliliği düşük olacaktır (Erwig vd., 1999). Çünkü ZVT' de değişikliğin olduğu satırların bir kopyasının alınması ve saklanması gerekmektedir. ZVT'nin uzlamsal nesneler gibi, temsilleri yer kaplayan nesneleri tutması ve her güncellemede bu nesnelerin kopyalarının alınması, önemli bir yer kaybına neden olur.

### 2.1.2 UVT, Zaman-Uzlamsal Uygulamalarda Niçin Yeterli Değildir?

Uzlamsal nesnelerin tarihçesinin, UVT veri tabanında tutulması oldukça zor olduğundan, zaman-uzlamsal uygulamalar için yeterli değildir. Bunun esas nedeni, UVT'nin geçerli zamanda *(valid time)* uzlamsal nesneleri tutması ve nesne yenilendiği zaman eski durumlarının silinmesidir. Bununla beraber, ZUVT'nin gelişiminde, UVT'deki nesne sınıflarının verimli organizasyonları ile tanımlanan yeni tip ve fonksiyonların hedeflendiğini görmekteyiz. Erwig vd. (1999), bu noktada esas ihtiyacın, kavramsal seviyedeki modellemeden çok, zaman ve uzlamsal entegrasyonun başarıyla sağlandığı indisleme yöntemleri ile beraber diğer bazı gelişmiş sorgulama dillerinin işlenmesi ve optimizasyon yöntemlerinin geliştirilmesi olduğunu belirtmiştir.

Özet olarak, ZUVT'dan beklenen, uzlamsal nesnelerin şimdiki zaman ve geçmiş zaman (belki gelecek zamana ait tahmini) durumlarını saklaması ve sorgulayabilmesidir. Bunun için, ZVT temelli yeni modellerin yetersizliği ve UVT'nin tek başına yetersizliği gösterilmiştir. Bununla beraber, "geometrileri zamanla değişen nesnelerin" ve hareketli nesnelerin modellenmesi için, UVT ve ZVT modellerinden yola çıkarak daha entegre modellere ihtiyaç duyulmaktadır.

## 2.2 Literatürdeki Hareketli Nesne Modellerine Genel Bakış

Bu bölümde, Erwig vd. (1998) tarafından tanımlanan "ADT ile modelleme", Grumbach vd.

(1997) tarafından tanımlanan "kısıtlamalı lojik model" ve Sistla vd. (1997) tarafından tanımlanan "zamansal lojik model" olmak üzere birbirinden tamamıyla farklı üç model hakkında bilgi verilecektir. Zamansal lojik modeli esas alan MOST modeli, bu tezde gerçeklenen hareketli nesne sisteminde kullanılan model ve algoritmalar için örnek alınmıştır. ADT ile modelleme ve kısıtlamalı lojik model literatürdeki diğer çalışmalara örnek olması amacıyla incelenmiştir.

### 2.2.1 ADT (*Abstract Data Type*) ile Modelleme

Bu yaklaşımda üç boyutlu dinamik nesneler, iki uzlamsal ve bir zaman boyutu ile temsil edilmişlerdir ve bu nesnelerin zaman ve uzlamsal davranışları soyut veri tipleri ile modellenmiştir. Buna göre, başarılı sorgu işleme için uygun tip *(sort)* ve fonksiyonların tasarımı ve gerçeklenmesi gerekmektedir. Tasarımda uygun tip ve operatörlerden oluşan kullanışlı imzalar *(signature)* hedeflenirken, bu tiplerin tanım uzayları *(carrier set)* ve operatörlere ait fonksiyonların tanımlanması gerekmektedir. Bu kendi içinde cebirsel bir yapıdır ve tasarlanması için soyut modelleme ve ayrık (*discrete*) modelleme olmak üzere iki farklı soyutlama seviyelerinden bahsedilmiştir. Soyut model, sonsuz setler kullanarak modellemenin kavramsal seviyede sıkışmasına neden olur. Örneğin, soyut modellemede bir gezinge, her an için gerçek değerini saklamaktadır. Bununla beraber, sonsuz setlerin sonlu kümede temsilini düşünmeden yapılan modelleme her yönden kolaydır, fakat gerçeklenmesi zordur, belki imkansızdır. Ayrık model ise, doğrusal (veya doğrusal olmayan) yaklaşımlar yardımı ile sonlu setler ve bunlar üzerinde gerçeklenebilir operatörler tasarlamayı hedeflemektedir. Erwig vd. (1998), her iki seviyede modellemenin gereğine değinmiştir; çünkü soyut model, sonlu ortamda temsil edilmenin getireceği karışıklığa girmeden kavramsal olarak doğru tip ve operatörün seçilmesine olanak sağlarken; ayrık model, kavramsal seviyedeki tip ve operatörlerin sonlu kümeler ile temsil edilmesine olanak sağlar. Güting vd. (1998) modelinin oldukça sistematik ve biçimsel bu stratejisi zaman-uzlamsal literatürde önemli bir yer edinmiştir. Koubarakis vd. (2003) tarafından geliştirilen CHOROCHRONOS isimli proje kapsamında, Erwig vd.'de (1998) geliştirilen zaman-uzlamsal soyut modeli ve Forlizzi vd.'de (2000) bu modelin ayrık model ile tamamlanması ve gerçeklenmesi, bu konudaki çalışmalarında önemli adım taşları olması yönünden örnek olarak verilebilir.

Erwig vd. (1999) modelinde gerçekleştirilen kapsamlı ve biçimsel cebirsel sistemden küçük bir örnek verilebilir. Erwig vd.'de (1999) *mpoint* ve *mregion* olmak üzere iki temel zaman-uzlamsal tipten bahsetmiştir. Bunlar, zamandan iki boyutlu uzaya eşleştirmedir. *time*, geçerli

zaman olmak üzere, Çizelge 2.1'de bu tiplerin ve bazı operatörlerin imzası gösterilmiştir.

Çizelge 2.1 ADT ile modellemede bazı tip ve operatörler

| ***Örnek Tipler*** | ***Örnek operatörler*** | |
|---|---|---|
| *mpoint* = *time* → *point*<br>*mregion* = *time* → *region* | *mpoint* × *mpoint* → *mreal* | **mdistance** |
| | *mpoint* × *mregion* → *mpoint* | **visits** |
| | *mpoint* → *line* | **trajectory** |
| | *mregion* → *region* | **traversed** |
| | *point* × *region* → *bool* | **inside** |
| | *line* → *real* | **length** |

**mdistance,** iki hareketli nesne arasındaki mesafeyi ifade etmek için zamanla değişen reel sayı tipi (*mreal)* ile modellenmiştir. **trajectory**, operatörü ise, zaman-uzlamsal bir nesnenin uzlamsal koordinatlardaki izdüşümü olan gezinge bilgisini vermektedir.

Örneğin, **uçuşlar** (**id**: *string*, **from**: *string*, **to**: *string*, **route**: *mpoint*) ilişkisel bir veri tabanındaki uçuş tablosu olmak üzere, Çizelge 2.2 ADT fonksiyonları ile geliştirilmiş SQL cümle örneklerini içermektedir.

Çizelge 2.2 Örnek ADT fonksiyonlarını kullanan SQL cümleleri

| ***Sorgu*** | ***SQL*** |
|---|---|
| "İstanbul'dan kalkan uçaklardan, 5000km yol almış olan uçuşları bulunuz | SELECT id<br>FROM flights<br>WHERE from = "IST" AND<br>**length** (**trajectory** (route)) > 5000 km |
| Birbirine 500 metreden az mesafe kalacak kadar yaklaşmış olan uçuşları bulunuz | SELECT A.id, B.id<br>FROM flights A, flights B<br>WHERE A.id <> B.id AND<br>**minvalue** (**mdistance** (A.route,B.route))<.5 km |

ADT ile modelleme üzerinde benzer sorgular geliştirilebilir. Bu yönde çalışma sadece Güting vd. (1994,1998,1999,2000) ile sınırlı değildir. Sistla vd. (1997) benzer yönde modellemeler geliştirmiştir.

Özet olarak, ADT ve gerekli operatörler ile ZUVT'da modelleme, ADT'lerin ve operatörlerin çeşitli uygulamalara göre şekillenebilmesi, kullanıcı seviyesinde yeni gelişmelere (yeni tip ve operatörlere) açık olması ve verimli algoritmalarla gerçeklenebilmesi yönünden oldukça başarılıdır. Bu tür modellemenin önündeki en önemli engel, ayrık zaman modellemenin gerektirdiği kısıtlamalardır. Konum bilgisinin bu yolla saklanması, sürekli sorgulama gibi bazı sorgular için verimsizdir. Diğer taraftan, ADT ile modellemenin diğer bir zorluğu boyut

bağımlılığıdır. Tanımlanan ve gerçeklenen operatörler, farklı boyutlu nesneler üzerinde de geçerliliğini korumalıdır. Son olarak, Su vd.'de (2001), ADT ile modellemenin esnek bir yapı olmadığı ve bazı zamansal, uzlamsal olay kombinasyonlarının ifade edilemeyeceği tartışılmıştır. Sonuç olarak, zaman ve uzlamsal entegrasyonun ADT seviyesinde kaldığı ve sorgulama seviyesinde (örneğin, SQL'in, ADT ile entegrasyonunun) istenen başarıyı veremeyeceği ifade edilmiştir (Su vd., 2001).

### 2.2.2 Kısıtlamalı Lojik Veri Modeli

$O_2$ veri tabanı üzerinde gerçeklenen ve Grumbach vd. (1997) tarafından geliştirilen DEDALE, kısıtlı lojik veri modelinin başarılı bir uygulamasıdır. Geometrik verinin lineer-kısıtlı soyutlanmasından yola çıkarak, zaman-uzlamsal nesneler için soyut seviyede tanımlanan sonsuz ilişkisel tabloların ayrık modelde matematiksel formüller ile temsil edilmesi esas alınmıştır. Bu iki seviyeli yaklaşım yönüyle, ADT ile modellemeye benzemektedir.

Bu model, uzlamsal bir sorgulama örneği ile kısaca açıklanabilir. Örneğin, Çizelge 2.3'de **ülke** ve **şehir** uzlamsal tablolarında **alan** özelliği, ***Region*** tipindeki ADT ile **konum** ise ***Point*** tipindeki ADT ile modellenmiştir. **ülke** uzlamsal tablosunun, soyut seviyedeki sonsuz elemanlı ilişkisel tablo karşılığı **ülke** (**isim**: *string,* **x**: *real,* **y**: *real*) olacaktır. Her şehrin 1 satırla ifade edilebildiği **şehir** tablosunun soyut modelde karşılığı ise tek elemanlı **şehir** (**isim**: *string,* **nüfus:** *int,* **x**: *real,* **y**: *real*) olacaktır. Tamamıyla ilişkisel modele dayalı **ülke** ve **şehir** tablolar üzerinde, Çizelge 2.3'deki uzlamsal sorguyu, SQL ile çözebiliriz.

Çizelge 2.3 Uzlamsal bir sorgu için soyut model üzerinde SQL cümlesi

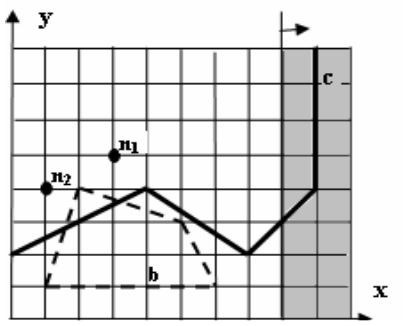
| **ülke** (**isim**: *string*, **alan**: *Region*) → **ülke** (**isim**: *string,* **x**: *real,* **y**: *real*)<br>**şehir** (**isim**: *string,* **nüfus**: *int,* **konum:** *Point*) → **şehir** (**isim**: *string,* **nüfus:** *int,* **x**: *real,* **y**: *real*) | |
|---|---|
| ***Sorgu*** | ***SQL*** |
| Türkiye'deki şehirlerin toplam popülasyonunu bulunuz | SELECT SUM(s.nüfus)<br>FROM ülke AS u, şehir AS s<br>WHERE u.x = s.x AND u.y = s.y AND<br>u.name="Türkiye" |

Çizelge 2.3'deki uzlamsal sorguda, istenen ülkedeki şehirlerin bulunması, şehrin bulunduğu noktanın ülke sınırları içindeki bir nokta ile *join* olmasıyla bulunmaktadır. Bu geleneksel *join* yönteminden farklı değildir; ancak bu işlem soyut modelde, sonsuz bir set üzerinde yapılmaktadır.

Soyut seviyedeki model; ayrık seviyede, atomik kısıtlar üzerinden lojik formüller ile temsil edilebilir. k-boyutlu uzayda tanımlanan nesneye ait lojik formülün genel ifadesi,

$$\left\{\left(x_1, x_2 ... x_k\right) \in \Re^k \mid \theta\left(x_1, x_2, ..... x_k\right)\right\} \tag{2.1}$$

olur. Örneğin, Şekil 2.1'de, k=2 boyutlu uzayda 2 nokta, 1 çizgi ve bir bölgeye ait, *sembolik ilişki* adı verilen lojik kısıtlamalar görülmektedir.




Noktalar **n1, n2**:
*x=1 ∧ y=4 ∨ x=3 ∧ y=5*
Çizgi **c**:
*(-x+2y = 4 ∧ x≥0 ∧ x≤4) ∨*
*(2x+3y=20 ∧ x≥4 ∧ x≤7) ∨*
*(x-y = 5 ∧ x≥7 ∧ x≤9) ∨*
*(x=9 ∧ y≥4 ∧ y≤8)*
Bölge **b**:
*(y≥1 ∧ 3x-y≥2 ∧ x+3y≤14 ∧ 2x+y≤13)*

Şekil 2.1 Örnek uzlamsal nesneler ve sembolik ilişkiler ile modellenmesi (Güting ve Schneider, 2005)

Bu kısıtlamalar yardımı ile modellenen uzlamsal nesneler üzerinde, seçme (*selection*), kesişim (*intersection*) ve izdüşümü (*projection*) gibi fonksiyonlar, sorgunun getirdiği kısıtlamanın ortamdaki kısıtlamalara eklenmesi ve gerekli sadeleştirmeler ile kolayca gerçeklenir. Örneğin, Şekil 2.1'de koyu bölge ile gösterilen $\sigma_{\underline{x \geq 8}}$ sorgusunda, noktalar ve bölge tamamıyla elenir, **c** çizgisinin ise bir kısmı olan $\left(x - y = 5 \wedge x \geq 8 \wedge x \leq 9\right) \vee \left(x = 9 \wedge y \geq 4 \wedge y \leq 8\right)$ ifadesi sorgu sonucundaki nesne olur.

Uzlamsal nesneler için tanımlanan kısıtlamalı model yaklaşımı; zamanın, uzlamsal koordinatlara ek bir boyut olarak doğrudan eklenmesi ile zaman-uzlamsal nesneler için de yapılabilir. Bu fikirden hareketle, Su vd.'de (2001) hareketli nesneler için kısıtlama modelinin genel ifadesi aşağıdaki gibi tanımlanmıştır:

m:= zaman aralıklarının sayısı olmak üzere,

$$a_1 < a_2 < ..... < a_m \tag{2.2}$$

ve n:= uzlamsal boyut sayısı olmak üzere, her *i=1,2,...n* uzlamsal boyutu için

$$p_i(t) = \bigvee_{j=0}^{m} (x_i = b_{ij} t + c_{ij} \wedge a_j \leq t \leq a_{j+1}) \tag{2.3}$$

hareketi temsil eden fonksiyon olmaktadır. (2.3)'deki, $b_{ij}$, $c_{ij}$ reel katsayılar olup, her ***j*** ($1 \leq j \leq m$ olmak üzere) ve her ***i*** ($1 \leq i \leq n$ olmak üzere) için, $b_{ij-1}a_j + c_{ij-1} = b_{ij}a_j + c_{ij}$ eşitliği sağlanmaktadır. Bu eşitlik, $p_i(t)$ fonksiyonunun sürekliliği için gerekli bir ifadedir. Böylece; bir hareketli nesnenin hareket ettiği uzlamsal uzayın her koordinatındaki hareketi, zamana bağlı doğrusal bir fonksiyon ile ifade edilmiştir. Örneğin,

**uçuşlar** (**id**: *string*, **from**: *string*, **to**: *string*, **position**: *mpoint*)

tablosunu ele alalım (Güting ve Schneider, 2005). Burada, **position** özelliğinin, 3 boyutlu uzayda hareketli bir uçağın konumu olduğunu düşünürsek; Şekil 2.2'deki kısıtlamalı lojik formüller, bu nesnenin her boyuttaki örnek hareketinden bir parçadır.

$x_1=2t-40 \ \wedge 0 \leq t \leq 21 \ \vee \ x_1=2 \ \wedge 21 \leq t \leq 22 \ \vee \ x_1=.5t-9 \ \wedge 22 \leq t \leq 47$
$x_2=-t+23 \ \wedge 0 \leq t \leq 21 \ \vee \ x_2=-t+23 \ \wedge 21 \leq t \leq 22 \ \vee \ x_2=1 \ \wedge 22 \leq t \leq 47$
$x_3=30 \ \wedge 0 \leq t \leq 21 \ \vee \ x_3=-5t+135 \ \wedge 21 \leq t \leq 22 \ \vee \ x_3=-t+47 \ \wedge 22 \leq t \leq 47$

Şekil 2.2 Üç boyutlu uzlamsalda hareket eden uçağın kısıtlamalı lojik formüller ile hareketinin modellenmesi

Kısıtlamalı modelin kendi içinde tutarlı ve basit lojik ifadeleri hareketli ortamlar için bazı sorgularda yetersiz kalacağı için, yön ve hız gibi özelliklerin kısıtlamalar ile ifade edilmesi de gerekmektedir (Su vd., 2001). Böylece, hız, diferansiyel matematikten faydalanarak; yön ise birim vektörler kullanılarak lojik olarak ifade edilmiştir. Örneğin; yukarıdaki doğrusal ifadelerin ***t***, zamanına göre türevleri hıza ait lojik formülleri verecektir. Bu iyileştirmelerle beraber, Vazirgiannis ve Wolfson (2001) kısıtlamalı modelin geleneksel veri tiplerini hiç kullanmadan, "kısıtlama seti" olarak tek tip kullanmasının oldukça radikal bir yaklaşım olduğundan ve yeni açılımlara ve tip tanımlamalarına olan ihtiyaçtan bahsetmiştir.

Bunların yanında kısıtlamalı modelin en önemli avantajı, nesnelerin dinamik özelliklerinin matematiksel ifade edilebilmesine olanak sağlamasıdır. Özellikle, bu ifadelerin sorgulamada kullanılması, ortamın boyutundan bağımsızdır. Bu modelin zamansal ve uzlamsal entegrasyonu sağlaması ve bu model üzerine kurulan sorgulamada başarımın arttırılması yönünde aktif olarak çalışmalar devam etmektedir.

### 2.2.3 MOST (*Moving Object Spatio-Temporal*) Modeli

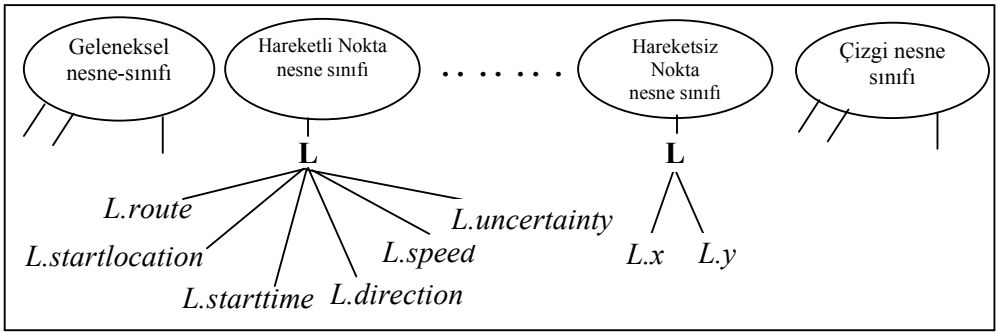
Wolfson vd.'de (1998) ortaya atılan bu modelin ana fikir, hareketli nesneye ait dinamik özellik olarak, sürekli zamanda değişen nesne konumları yerine "hareket vektörünün" kullanılmasıdır. Böylece, yüksek frekansta güncellenmeyi gerektiren konum bilgisinin yerini,

daha az sıklıkta güncellemenin yeterli olacağı hareket vektörü alır. Diğer bir ifade ile hareket vektörü, nesneye ait bir üst seviyedeki soyutlamadır ve daha çok bilgi taşır. Bu da, güncelleme sıklığında azalmaya olanak sağlar.

MOST modeli, dinamik nesnelere ait **anlık** ve **yakın gelecek hakkında** sorgulamayı hedefleyen bir modeldir. Örneğin, MOST modeli için geliştirilen, FTL (***F***uture ***T***emporal ***L***ogic) adı verilen zamansal sorgulama dili, "*Bundan sonraki 10 dakika içinde havaalanının 30 km yarıçaplı bölgesine girecek olan uçakların bulunması*" gibi sorguları hedef almaktadır.

MOST modeli; veri modeli, belirsizlik ve konum yenileme modeli olarak iki ana başlıkta incelenebilir.

#### *2.2.3.1 MOST Veri Modeli*




Şekil 2.3 MOST veri modelinde tanımlanan nesne sınıfları

Veri modeli, nesne-sınıflarından (*object-classes*) oluşan bir veri tabanı üzerinde Şekil 2.3'de gösterilmiştir. Nesne sınıfı, bir grup özellik içeren veri yapılarıdır ve herhangi bir nesne-ilişkisel veri tabanında gerçeklenebilir. MOST veri modelinde nesne-sınıfları, geleneksel, uzlamsal, zamansal ve zaman-uzlamsal veri tipleri için tanımlanabilir. Örneğin, durağan nokta nesne sınıfı konum özelliği, **L**, *L.x* ve *L.y* olmak üzere iki alt özellikten oluşurken; hareketli nokta nesne sınıfına ait konum "dinamik özelliği", **L**, 6 özellikten oluşmaktadır. Diğer bir ifade ile **L,** hareket vektörünü temsil eden dinamik özellikler setidir. Wolfson vd.'de (1998) tanımlanan bu özellikler Çizelge 2.4'deki tabloda gösterilmiştir.

Çizelge 2.4 Hareket vektöründeki dinamik özellikler

| | |
|---|---|
| *L.route* | Hareketli noktanın önceden tanımlanmış rotası* |
| *L.startlocation* | Nesnenin rotası üzerindeki, en son güncellenme konumudur. Diğer bir ifade ile *L.starttime* zamanında nesnenin konumudur. |
| *L.starttime* | en son güncellenme zamanıdır. |
| *L.direction* | nesnenin hareket yönüdür. |
| *L.speed* | nesnenin gelecek zamandaki konumlarını tahmin eden hareket fonksiyonudur. Diğer bir ifade ile nesnenin *L.starttime* zamanından itibaren, *L.startlocation* konumuna olan uzaklığının ***t***'ye bağlı fonksiyonudur. |
| *L.uncertainty* | Belirsizliğin, yani gerçek konumdan sapma miktarı için sınır değerin, ***t***'ye bağlı fonksiyonudur. |

#### *2.2.3.2 Belirsizlik ve Konum Yenileme Modeli*

Nesne hızının sabit olamaması nedeniyle, veri tabanında saklanan dinamik özellikleri kullanarak hesaplanan konum bilgisi, gerçek hayat konum bilgisinden farklı olur. Bu farka "konumsal sapma" (*deviation*) adı verilir. Nesnenin konumsal sapması, o andaki **gerçek konumu** ile veri tabanında **modellenen konumu** arasındaki fark alınarak her an için hesaplanabilir (Wolfson vd., 1999). Nesnenin gerçek konumu, GPS (***G***lobal ***P***ositioning ***S***ystem) sayesinde her zaman dilimi için öğrenilebilir. Diğer yandan veri tabanına gönderilen en son güncellenme raporu sayesinde, veri tabanının nesnenin o andaki konumu hakkında vereceği bilgi de doğrusal ara değerlendirme ile bulunabilir. (Örneğin; *L.speed* sabit olduğu varsayımı ile veri tabanındaki modellenen konum, *"L.startlocation + v*t"* olur.*)*

Hareketli nesne, konum yenileme raporunu, konumsal sapma miktarı *L.uncertainty* değerini aştığı zaman veya yön ve/veya rota değiştirdiği zaman üretip, veri tabanına iletir. Veri tabanına gönderilen konum yenileme raporu, hareketli nesneye ait olan konum özelliği **L**'nin, alt-özelliklerinin hepsi veya bir kısmının yeni değerlerini içermektedir (en azından *L.startlocation* ve *L.speed* alt-özelliklerini içermelidir).

Güncellenmenin gerçeklenmesi belirsizlik modeli ile doğrudan ilişkilidir. Çünkü belirsizlik modeli, *L.uncertainty* sınır değerini belirlemektedir. Belirsizliğin azalması, güncellenmenin daha sık olması; diğer bir ifade ile güncellenme maliyetinin artmasıyla sağlanır. "Güncellenme aralığı", ardışık güncellenme zamanları arasındaki zaman aralığı olmak üzere, *L.uncertainty* belirsizlik değeri Wolfson vd.'de (1999) 3 farklı şekilde modellenmiştir:

- bütün güncellenme aralıklarında aynı olup sabit bir değer olabilir.
- güncellenme aralığı boyunca sabit olmak üzere, her güncellenmede yeniden hesaplanır.

---

* Yol-haritasındaki hareketler için rota, önceden belirli ardışık çizgilerdir. Serbest hareketler için, rota sonsuz bir çizgi olup; yenilenme zamanlarında sadece açısı değişir.

- güncellenme aralığı boyunca zamana bağlı olarak değişir.

Gözü-kapalı tahmin yönetimi *(dead-reckoning policy)* adı altında, yukarıdaki belirsizlik modellerine bağlı olarak "hız gözü-kapalı-tahmin modeli (*speed dead-reckoning*)", "uyarlanır gözü-kapalı-tahmin modeli (*adaptive dead-reckoning*)" ve "sökme-algılamalı gözü-kapalı-tahmin modeli (*disconnection detection dead-reckoning*)" isimlerinde üç farklı güncellenme yönetim modelleri önerilmiştir. Bu modeller yenileme maliyeti ile belirsizlik arasındaki dengelemeyi sağlarlar. Ek 1, bu modeller hakkındaki data detaylı bilgi içermektedir.

#### *2.2.3.3 FTL (Future Temporal Logic) Sorgulama Dili*

Daha önce açıklandığı gibi MOST modeli üzerinde dinamik nesnelere ait **yakın gelecek hakkında** sorgulama, FTL adı verilen zamansal sorgulama dili ile gerçeklenmiştir (Wolfson vd., 1998). Bu ihtiyacın esas nedeni SQL ve OQL gibi düzenli sorgulama dilleri ile tipik zamansal olayların (*until*, *eventually* gibi) ifade edilememesidir.

FTL'deki zamansal operatörlerin hepsi (*Eventually, Eventually_Within, Always* gibi), 2 temel operatörden (*Until*, *Nexttime*) türetilmiştir. FTL derleyici modülü ve hareket modeli mevcut bir ilişkisel veri tabanı sistemi üzerinde geliştirilebilir. Çizelge 2.5'de 2 adet FTL sorgu örneği verilmiştir (Sistla vd., 1997).

Çizelge 2.5 FTL'de örnek sorgulama

| ***Sorgu*** | ***FTL*** |
|---|---|
| **P** poligon bölgesine girinceye kadar aralarındaki mesafe 5km'den az olan ***o*** ve ***n*** nesnelerini bulunuz | **RETRIEVE** o,n<br>**WHERE** DIST(o,n) < 5 *UNTIL*<br>(*INSIDE*(o,P) and *INSIDE*(n,P)) |
| Üç birim zamana kadar **P** poligon bölgesine giren ve bu bölgede 2 birim zaman kadar kalan ***o*** nesnelerini bulunuz" | **RETRIEVE** o<br>**WHERE** *EVENTUALLY_WITHIN*_3<br>((*INSIDE*(o,P) and *ALWAYS_FOR*_2(o,P)) |

MOST modeli içerdiği belirsizlik modeli sayesinde olasılıksal sorgulamaya da olanak sağlar. Olasılıksal sorgulamada, bir $\mathbf{o_i}$ nesnesinin sorgudaki belirtimi (*predicate*) sağlamasının olasılığı, $\mathbf{p_i}$ bulunur. Buna göre, sorgu sonuçları $\mathbf{(o_i, p_i)}$ ikilileridir.

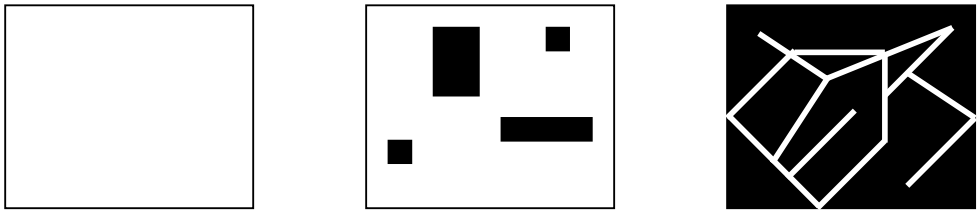
Sonuç olarak, MOST modeli, hareket vektörünü "dinamik özellikler" ile modeller. Dinamik özellikler ise, kendileri veri tabanında güncellenmeden, nesnenin ilerleyen zamandaki konum bilgisini belirli bir doğruluk garantisi ile (belirsizlik sınır değeri içinde) hesaplamaya olanak

sağlar. Bununla beraber, konum güncellenme maliyeti ile belirsizlik arasıdaki dengenin sağlanması için çeşitli maliyet modelleri geliştirilmiştir. MOST modeli, mevcut veri tabanı üzerinde, tasarsız *(ad-hoc)* olarak, yakın gelecek hakkında zaman-uzlamsal uygulama geliştirmeyi hedeflemektedir. Zamansal operatörler esas alınarak geliştirilen FTL lojiği ile yakın gelecek üzerinde birçok kullanışlı sorgu tanımlanmıştır. Dinamik özellikler ile tanımlanan hareket modeli, gerçek hayatta birçok hareketli sistemler için başarılı bir model olabilir.

## 2.3 Hareket Ortamının Modellenmesi

Hareketin bulunduğu ortamın modellenmesi ise, hareket ortamının hareket üzerindeki kısıtlamalarının ortaya çıkardığı **hareketin senaryoları** ile belirlenir. Pfoser ve Jensen (2003), farklı hareketli nesne uygulamalarını 3 ana hareket senaryosu ile sınıflandırmaktadır:

- kısıtlamasız hareket
- kısıtlamalı hareket
- ağ kısıtlamalı hareket



a) b) c)

Şekil 2.4 Hareket senaryoları *a)* kısıtlamasız *b)* kısıtlamalı *c)* ağ kısıtlamalı

Şekil 2.4'de gösterilen hareket senaryolarında, beyaz bölge hareketin serbest olduğu, siyah bölge ise herhangi bir hareketin gerçekleşmediği alanları temsil etmektedir. Buna göre kısıtlamasız hareket bütün uygulamalı alan boyunca hiçbir engele (kısıtlamaya) maruz kalmadan gerçekleşmektedir. Denizdeki gemi hareketi veya bir odada hareket eden atom zerreleri bu tip senaryo modeline örnek olarak verilebilir.

Kısıtlamalı hareket modelinde ise, hareketi kısıtlayan bazı uzlamsal altyapı nesnelerinin varlığı söz konusudur. Örneğin; bina, göl vb. gibi nesnelerin bulunduğu alanda gerçekleşen bir yaya hareketi tipik bir kısıtlamalı hareket senaryosudur.

Son olarak, ağ kısıtlamalı senaryolarda, hareketler belirli bir ağ üzerinde gerçekleşmektedir.

Bu tip senaryo, kısıtlamalı hareket senaryosunun özel bir hali olup, nesne konumlarının belirlenmesinde 2 boyutlu öklit koordinat sistemi referans alınmaz. Bunun yerine, çalışma uzayı olarak üzerinde hareketin gerçekleştiği ağ referans alınır. Örneğin, "E-1 isimli yolunda 12. km." bilgisi, bir nesnenin konumunu belirlemede yeterlidir. Bu nedenledir ki, ağ kısıtlamalı hareketlerin çalışma uzayı, 2 boyutlu öklit uzayından daha basit olarak, literatürde 1,5 boyutlu uzay olarak tanımlanmıştır. Bununla beraber, bu tip senaryolarda yolun genişliği gibi bilgiler de ihmal edilebilir; çünkü bu bilgi, ağ referansı yerine ancak 2 boyutlu öklit referans sistemi ile belirlenebilir. Bu gibi özellikleri ile ağ kısıtlamalı hareket senaryosu, diğer senaryolardan farklılık göstermektedir. Ağ-kısıtlamalı ortamlardaki yol-ağı, ihtiyaca göre kenarlar ve köşelerden oluşan bir ağ yapısı (graf yapısı) ile modellenebilir.

## 2.4 Sonuç

Bu bölüm, hareketli nesne veri tabanlarında nesnelerin, hareketin, konum belirsizliğinin ve hareket ortamının modellenmesi konusunda yapılan çalışmalardan örnekler içermektedir. Bu modellerden, ADT ile modelleme, uzlamsal modeller üzerine bina edilen yaklaşımlar için bir örnek iken; MOST modeli, geleceğe ait zamansal bir lojik olup, zamansal yargılamanın uzlamsal boyutlarda kullanımına dayanan bir model örneğidir. Kısıtlamalı lojik model ise dinamik ortamın matematiksel ifadeler ile temsil edilmesi esasına dayanan bir modeldir.

# 3. HAREKETLİ NESNE VERİ TABANLARINDA ERİŞİM YAPILARI

Veritabanları büyük bilgi yığınlarının saklanmasını ve hızlı erişimini hedef almaktadır. Hem bilgi yığınlarını saklamak hem de hızlı erişim birçok sorunu beraberinde getirmektedir. Bunlardan en başta geleni; veri tabanının ihtiyaç duyulan kısmı için ana hafızanın yetersiz kalmasıdır. Bu yüzden ikincil hafızaya erişim zamanı, sorgu işlenmesinde sistem performansını belirleyen önemli bir parametredir. Erişim yöntemleri, ikincil hafızada yer alan bilgiye erişim yükünü en aza indirgemeyi hedefleyen veri yapılarıdır. Genel olarak baktığımızda, erişim yöntemleri indisleme (*indexing*) veya hesaba dayalı adresleme (*hashing*) yöntemleri ile gerçeklenmektedir. Bu tez çalışmasında indislemeye dayalı geliştirilen yöntemler esas alınmıştır.

Statik ve tek boyutlu bilginin saklandığı geleneksel veri tabanlarındaki indisleme ihtiyacını düşündüğümüzde, hareketli nesne veri tabanlarında bu tip veri yapılarının ne kadar kritik olduğu daha iyi anlaşılabilir. Çünkü böyle bir veri tabanının dinamik yapısı, indisleme ve diğer veri tabanı ana modüllerinin bu dinamik ortama ayak uydurmasını gerekli kılmaktadır. Örneğin, son senelerde etkin bir araştırma konusu olan hareketli nesne veri tabanları, veri tabanı alanında birçok yeni gelişmelere zemin hazırlamıştır. Büyük bir şehirdeki bütün araçların konumlarını, hareketli nesne veri tabanında sakladığımızı düşünelim. Böyle bir sistem üzerindeki zaman-uzlamsal bir sorgular için bütün nesnelerin konumlarının test edilmesi gerçeklenebilir bir çözüm değildir. Bununla beraber, indis olarak geleneksel veya uzlamsal indis yapılarını doğrudan kullanmak da etkin bir çözüm olmayacaktır; çünkü belirli bir model üzerine konum bilgisi yenilenen nesneler için indisin devamlı güncellenmesi ve güncellenme işleminin maliyetinin yüksek olması bu yaklaşımı da yetersiz kılmaktadır. O zaman dinamik ortama uyumluluk, hareketli nesnelere ait indislerden beklenen temel özelliklerin başında gelmektedir.

Bir indis veri yapısından beklenen, belirli bir arama anahtarı üzerinden, dosyadaki kayıt veya kayıtlara, hızlı erişmeye olanak sağlamasıdır. Hızlı erişim ile kastedilen, disk tabanlı bir indis yapısı için disk erişim sayısıdır; ana hafıza tabanlı bir indis için ise çalışma zamanıdır. Sorgu (join veya normal tipte) işlenmesinde, indis üzerinden arama yapılması veri tabanındaki bütün kayıtlara erişilmeyi engeller. Örneğin, Bayer ve McCreight'in (1972) tasarladığı, dengeli ağaç yapısındaki B-tree indis'inin çalışma zamanı $O(\log_k N)$'dir. Bu çalışma zaman ile anlatılmak istenen, disk ortamında bulunan k-order bir B-tree ağacı ile ağacın yüksekliği kadar adımda istenen kayıt veya kayıtları içeren disk sayfalarına erişilebilmesidir. **N** indislenen eleman sayısı olmak üzere ağacın yüksekliği $\log_k N$'dir ve ağaç her seviyede vereceği kararda bundan

sonraki alt ağacın sadece 1/K'lık kısmı ile ilgilenmektedir.

B-tree gibi ağaç tabanlı indis yapıları zamansal veya uzlamsal veri için indis yapısı olarak doğrudan kullanılamasa da, bu gibi çok boyutlu veri için güçlü indis yapılarına bir temel teşkil edebilir. Örneğin, çok-boyutlu uzlamsal erişim yöntemleri (**S**patial **A**ccess **M**ethods, SAM) adı altında birçok geometrik indis yapıları tasarlanmıştır. Bunlardan Guttman'ın (1984) tasarladığı R-tree, B-tree'nin çok-boyutlu genelleştirilmiş bir versiyonu olarak düşünülebilir. Diğer yandan, zamansal veri için daha çok "ayrık zamanla değişen" tipte veriler için indis yapıları tasarlanmıştır. Gerek uzlamsal gerek zamansal indis yapıları sürekli zamanda zaman-uzlamsal indis yapıları için bir esin kaynağı olabilir.

Bu tez çalışmasında incelenen ve geliştirilen indis yapıları daha çok uzlamsal erişim yöntemleri esas alınarak tasarlanan yapılar olduğu için bu bölümde öncelikle, Bölüm 4.1'de uzlamsal erişim yöntemlerinden bahsedilecektir. Bölüm 4.2'de ise, üzerinde araştırma yapılan hareketli nesne indis yapıları incelenecektir. R-tree uzlamsal indis yapısı, Bölüm 4.2'de incelenecek olan hareketli nesne indisleri için bir temel yapı taşıdır. "Hilbert packing R-tree" ise, önceden belirli olan yol-ağının indislenmesi için seçilen indis yapısıdır. Zaman-uzlamsal erişim yöntemleri R-tree tabanlı yöntemler ile sınırlı olmadığı gibi, indisleme ile de sınırlı değildir. Hesaba dayalı adresleme, ayarlanabilir hücre-tabanlı yapılanma gibi indisleme dışındaki yaklaşımlar bu kapsamdaki diğer konu başlıklarıdır.

## 3.1 Uzlamsal İndislemede Temel Yaklaşımlar

İndislemede esas yaklaşım bilginin sıralanması olduğu için, indislenecek olan bilginin sıralanabilir olması önemli bir sorundur. Örneğin, uzlamsal bilgiyi düşündüğümüzde, tek boyuta göre sıralamanın konumsal bir sıralama olamayacağı açıktır; çünkü bilginin diğer boyutlardaki değerleri için de başka bir sıralama gerekecektir. O zaman, uzlamsal bilginin indislenmesinde çok boyutlu konumsal sıralama önemli bir problemdir (Shekhar ve Chawla, 2003). Bununla beraber, sıralamak yerine, birbirine konumsal olarak yakın nesneler aynı veya yakın disk sayfalarına yerleştirilebilir. Daha sonra, bu disk sayfaları B-tree'ye benzer hiyerarşik bir ağaç yapısında organize edilebilir. Bu ana fikirler ışığında, literatürde uzlamsal indislemede üç farklı yöntem göze çarpmaktadır:

- Konumsal sıralama ve sonrasında tek boyutlu indisleme
- Nesneye dayalı parçalanma yapıları *(Data decomposition)*
- Nesneye dayalı olmayan parçalanma yapıları *(Space decomposition)*

**Konumsal sıralama** ile çok boyutlu bilgi bütün boyutları göz önüne alınarak tek boyuta indirgenerek sıralanmış olur. Konumsal sıralama için en başta gelen yöntemler, Faloutsos ve Roseman'ın (1989) tasarladığı "Z-order" ve "Hilbert-order" yöntemleridir. Bu yöntemlerde nesnelerin, örneğin, tek-boyutlu Z-değerleri belirlenir. Bu değerler, nesnelerin birbirine yakınlığı için bir ölçü olmaktadır. Sonuçta, tek boyuta indirgenen nesnelerin sıralanması için Z-değerleri üzerinden B-tree indisleme doğrudan kullanılabilir.

Diğer iki yöntemdeki ana fikir ise, çalışma uzayının düzenli parçalanması ve bu parçalar ile nesneleri konumsal özelliklere göre ilişkilendirmektir. Örneğin, yakın olan nesnelerin aynı disk sayfasında tutulması ve çalışma uzayının, ağacın kökünden itibaren yapraklara kadar iç içe birbirini kapsayarak parçalanması bu yöntemlerde ortak özelliklerdir. **Nesneye dayalı parçalanma yapılarında,** uzayın parçalanmasında kriter nesnelerin konumsal özelliklerdir. Bu yöntemle oluşturulan veri yapılarının en yaygın olanı, Guttman'ın (1984) tasarladığı R-tree'dir. **Nesneye dayalı olmayan parçalanma yapılarında** ise uzayın, nesne konumsal özelliklerinden bağımsız olarak daha düzenli parçalanması esas alınmıştır. Örneğin, Nievergelt vd.'ın (1984) *grid-file* yapısı ve Bentley'ın (1990) *kd-tree* yapısı bu gruptaki birçok erişim yapısına esin kaynağı olmuş indis yapılarıdır.

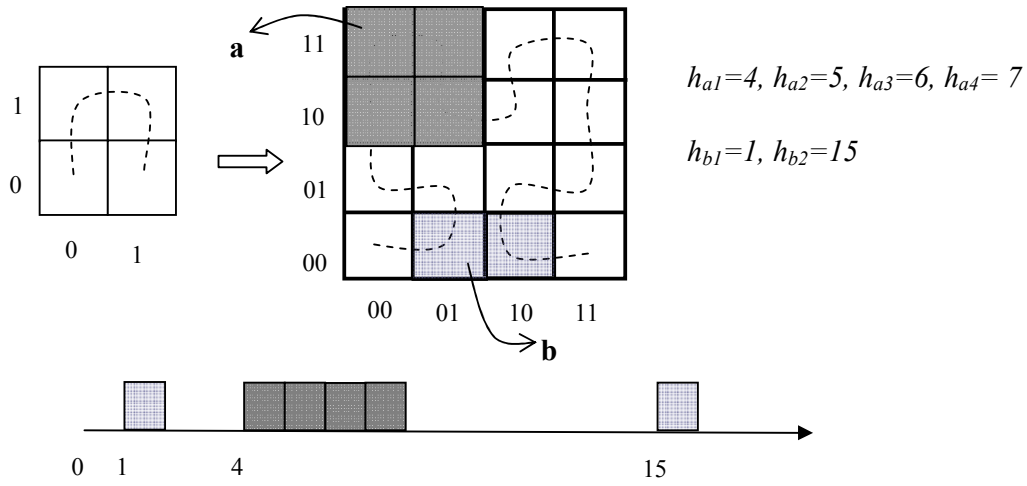
Yukarıda açıklanan farklı yaklaşım gruplarının her birinde (özellikle son ikisinde) son 15-20 sene içerisinde dinamik* özellikte onlarca farklı güçlü indis yapıları tasarlanmıştır. Bu tezde geliştirilen modeller ve uygulamalarda, Hilbert-order ve R-tree tabanlı erişim yöntemleri kullanıldığı için, bu yapıların ana özellikleri incelenecektir.

### 3.1.1 Hilbert Sıralaması

Faloutsos ve Roseman'ın (1989) tanımladığı Hilbert sıralaması, bir çeşit uzlamsal-dolgu eğrisidir (*space-filling curve*). Bu eğri, 1-e-1 eşleme ile birbirine yakın olan nesneleri, tek boyutlu bir eksen üzerinde sıralamakta kullanılır. Örneğin, Şekil 3.1'de, sırasıyla 4(2x2) ve 16(4x4) eşit parçaya ayrılan çalışma uzayının sıralanması ve 4x4 grid yapıdaki **a** ve **b** nesnelerinin *hilbert* değerlerine göre tek boyutlu eksende sıralanması gösterilmiştir. **a** nesnesi tek bir hilbert değer aralığı ile ($h_a$ = 4-7) ifade edilebilirken; **b**, iki farklı hilbert değeri ile ($h_{b1}$= 1, $h_{b2}$= 15) ifade edilebilmektedir. Herhangi bloğun hilbert değerinin bulunması algoritması, –x ve –y koordinatlarının dönüşümlü birleştirilmesinin (*interleaving*) ardından

* Dinamik ile kastedilen, ağacın karakteristik özelliklerini bozmadan genişleyebilmesi, ekleme/çıkarma işlemleri ile ağacın dengesinin bozulmamasıdır. Yoksa, sürekli zamanda özellikleri değişen dinamik ortamlara uygun olması demek değildir.

bazı yerleştirme kuralları ile elde edilir.




Şekil 3.1 Hilbert sıralaması (Shekhar ve Chawla,2003)

Çalışma uzayında bulunan nesnelerin hilbert değerleri birbirinden farklı ve sıralanmış olduğu için, B+-tree gibi aralık sorgulama için verimli geleneksel indis yapılarına yerleştirilebilir. Sorgulamada; örneğin aralık uzlamsal sorgulamada, uzlamsal sorgu bölgesinin hilbert değerleri, B+-tree ağacında sıralı saklanan hilbert değerleri ile karşılaştırılır. Eşitliğin sağlandığı hilbert değerlerine sahip nesneler, aralık sorgusu ile kesişenler olacaktır.

### 3.1.2 R-tree

Bilindiği gibi, Bayer ve McCreight'in (1972) B-tree indisleme yöntemi, çok seviyeli bir indisleme olup, dinamik ve dengeli bir ağaçtır. B-tree'de ekleme/çıkarma işlemleri ile ağacın dengesinin korunması ikiye bölme(*split*) ve birleştirme (*merge*) işlemleri ile gerçekleştirilir. B-tree'nin yer verimliliği ise, en kötü durumda %50'dir. Bu özelliklerin uzlamsal ortama taşınması, diğer bir ifade ile B-tree'nin çok boyutta genelleştirilmesi R-tree ağacı için motivasyon kaynağı olmuştur.

Guttman'ın (1984) R-tree yönteminde ise, her çeşit geometrik nesne indislenebilir; bununla beraber, ağaçta saklanan veri **region** tipindeki, d-boyutlu uzlamsal nesneleri çevreleyen minimum d-boyutlu dörtgenlerdir. Bu dörtgenlere MBR *(**M**inimum **B**ounding **R**ectangles)* adı verilir. MBR'ların kenarları ortamdaki koordinat sistemine paralel olmalıdır. Bu yapısal düzenlemeler sayesinde ağaç, hiyerarşik ve iç içe düzenli bir yapı kazanabilmektedir. R-

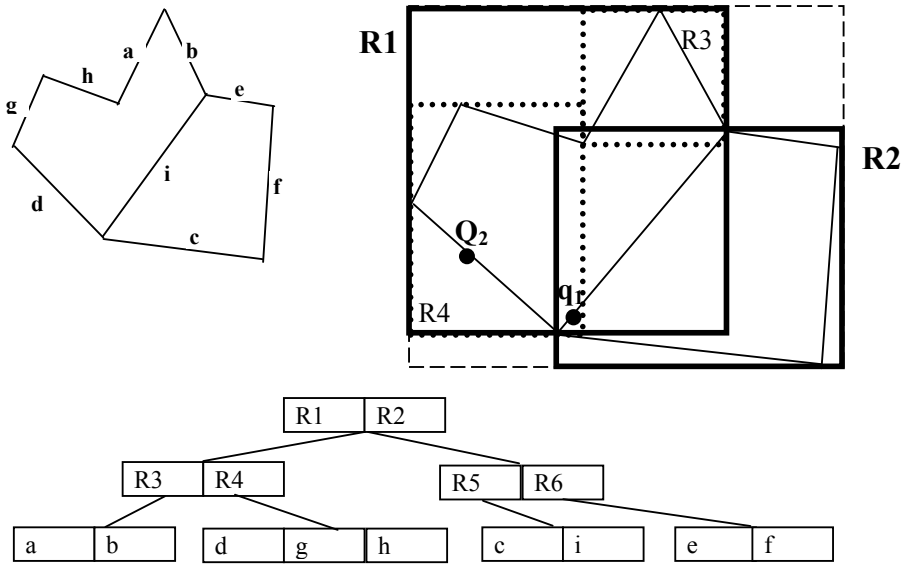
tree'de iç düğümlerde yer alan MBR'lar bir alt seviyedeki düğümleri kapsamaktadır. Yaprak düğümlerde ise ortamdaki uzlamsal nesnelere işaretçiler bulunmaktadır. R-tree'de iç içe MBR'lardan oluşan uzlamsal parçalama ile başarılı bir filtreleme sağlanmaktadır. Böylece bir sorgu için, R-tree ile filtrelenerek erişilen yaprak düğümlerdeki nesneler tasfiye (*refining*) aşamasıyla, yani nesnenin gerçek uzlamsal şekli analiz edilerek, sorgu koşuluna uygunluğu kontrol edilir.

#### *3.1.2.1 Tanım*

Bir R-tree, Guttman'da (1984), *(m, M)* parametre ikilisi ile tanımlanmıştır. Buna göre,

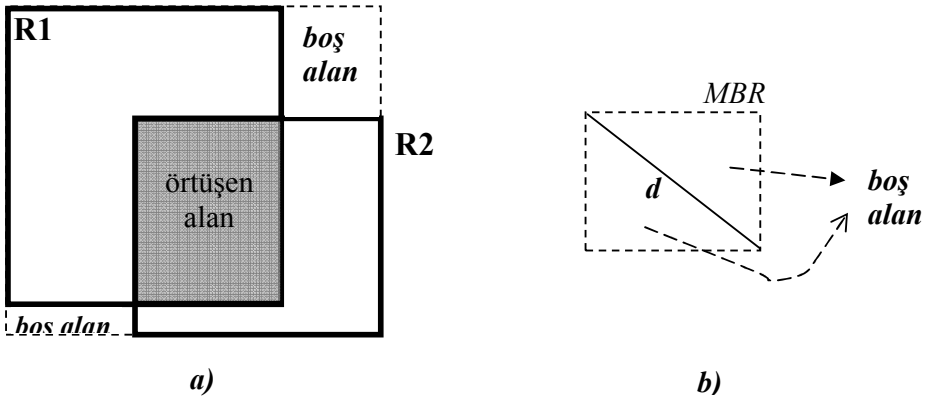
- Ağaçtaki her düğüm, m ≤ (M/2) olmak üzere, en az *m*, en fazla *M* indis kaydı tutar. Kök düğümü ise, (yaprak düğüm olmaması kaydıyla) en az iki indis kaydı tutar.
- Yaprak düğümlerdeki her indis kaydı *(I, id)* bilgi ikilisini tutar. *I, id* ile işaret edilen ve uzlamsal nesneyi çevreleyen MBR'dır.
- Yaprak olmayan iç düğümlerdeki her indis kaydı, *(I, child-id)* bilgi ikilisini tutar. *I, child-id* ile işaret edilen bir alt seviyedeki çocuk düğümündeki nesneleri çevreleyen MBR'dır.

Samet'ten (1995) alınan örnek (Şekil 3.2), 9 adet çizginin, *(m,M)=(2,3)* olan bir R-tree'ye yerleştirilmesini göstermektedir.




Şekil 3.2 Dokuz adet çizginin nesneye dayalı parçalanması ve oluşan R-tree yapısı (Samet, 1995)

Şekil 3.2'de, kesikli çizgi ile gösterilen çalışma bölgesi, birbiri ile kesişen R1 ve R2 MBR'larına ayrılmıştır. Şekilde karışıklık olmaması için sadece R1'in iç içe parçalanması gösterilmiştir. Buna göre, R1, noktalı çizgi ile gösterilen R3 ve R4 MBR'larını içermektedir. Son olarak R3, 'a' ve 'b' nesnelerini çevrelerken; R4, 'd', 'g' ve 'h' nesnelerini çevrelemektedir. R-tree kurallarına uygun olarak farklı ağaç yapıları oluşturulabilir. Bu farklılık, ikiye bölme işleminde, nesneleri gruplandırmadaki farklı yöntemlerden kaynaklanır. Örneğin *quadratic_split* ve *linear_split*, özgün R-tree veri yapısında tanımlanan ikiye bölme algoritmalarıdır. Sellis'in (1987) R+-tree yapısı ve Beckmann vd.'nin (1990) R*-tree yapısı önemli R-tree çeşitlemelerinden bazılarıdır. Bunlardaki farklı ikiye bölme algoritmaları, daha başarılı uzay parçalamaları ile daha verimli ağaçlara olanak sağlar.




Şekil 3.3 Örtüşen alan ve boş alanlar

### *3.1.2.2 Performans Analizi*

Çoğu araştırma projelerinde olduğu gibi bu tezde kullanılan R-tree gerçeklemelerinde de, her düğüm, bir disk sayfasına *(disk page)* karşılık gelmektedir. Bu performans için de arzu edilen bir kriterdir. R-tree indisinin performansını belirleyen diğer iki önemli parametre, ağaçtaki **toplam örtüşen alan miktarı** ve **toplam boş alan miktarıdır**. Şekil 3.2'deki ağacın kök seviyesindeki bu değerler, Şekil 3.3'de gösterilmiştir. Benzer şekilde, 'd' çizgi nesnesine ait MBR'ın oluşturduğu boş alan Şekil 3.3b'de gösterilmiştir. **Örtüşen alan** miktarının performansa etkisi, ağacın aramada seçiciliği (filtreleme özelliği) ile ters orantılıdır (Hadjieleftheriou vd., 2006). Örneğin, R-tree çalışma uzayının ayrık olarak parçalanmaması nedeniyle, MBR'ların kesişmeleri kaçınılmazdır. Bu yüzdendir ki; bir nesne birden çok MBR'ın kapsam alanına girebilir; bununla beraber, sadece bir disk sayfasında saklanır. Örneğin Şekil 3.2'de, *i* çizgi nesnesi R5 MBR'ı ile ilişkilendirilmiştir; fakat aynı zamanda R1

MBR'ının da içindedir ve R3 ve R4 MBR'ı ile kesişmektedir. Bu durumda, uzaydaki herhangi bir noktanın hangi MBR'a ait olduğunun bulunması en kötü durumda bütün ağacın araştırılmasını gerekli kılabilir. Örneğin, Şekil 3.2'deki $\mathbf{q_1}$ noktasının hangi çizgide olduğunu bulan bir sorguyu düşünelim. R-tree üzerinde uzlamsal nokta ve aralık sorguları yukarıdan aşağıya *(top-down)* rekürsif olarak işlendiğine göre, erişilen indis disk sayfaları 5 adettir. Bununla beraber, $\mathbf{Q_2}$ noktasının 'd' çizgisi üzerinde olduğu ise 3 erişimde anlaşılmaktadır. **Boş alan** miktarı ise, filtrelemenin her seviyesinde yanlış yakalama (*false hit*) sayısını doğru orantılı etkileyen değerdir.

N: indislenen kayıt sayısı olmak üzere, (m,N) özellikteki bir R-tree'nin maksimum seviye sayısı $\lfloor \log_m N \rfloor$-1'dir. R-tree uygulamalarının çoğunda **m** değeri yüksek tutularak ağacın derinliği küçük tutulur ve genişlemenin yatay olması sağlanır. Örneğin, 100 milyon uzlamsal nesne için oluşturulan bir R-tree'de **m**=100 seçilirse, derinlik 5 olur.

#### *3.1.2.3 Nesne Ekleme/Çıkarma*

R-tree'ye olan yeni kayıt ekleme/çıkarma işlemi ise, yapısal olarak B-tree nesne eklemesine benzerdir. Ekleme yapılacak yaprak düğümün bulunmasının ardından, eklemenin yapraktan – gerekiyorsa- köke kadar rekürsif olarak yapılması işlemlerini içerir. Eklemenin yapılacağı yaprak düğüm, kökten itibaren her seviyede en az alan artışına (*least enlargement criterion*) neden olacak şekilde belirlenir. Eklemenin yapılacağı düğümde yer kalmaması durumunda, ikiye bölünme *(split)* operasyonu ile düğümdeki nesneler iki gruba ayrılır. Bu gruplanma için ise –daha önce bahsedildiği gibi- farklı yaklaşımlar (*quadratic_split*, *linear_split, Rstar_split* gibi*)* vardır. Yaprak düğüme ekleme ile ortaya çıkan değişiklikler bir üst düğüme yansıtılır. Ağaçtaki bir **D** düğümüne yapılacak eklemenin, üst seviyede bir değişiklik gerektirmesi, ekleme ile **D**'nin ikiye bölünmesi veya **D**'yi çevreleyen MBR'ın değişmesi ile olur. Diğer yandan, nesne çıkarmada ise, düğümdeki nesne sayısı *m* değerinin altın düşerse birleşme *(merge)* işlemi gerektirir. İkiye bölme ve birleşme işlemleri, zaman gereksinimi oldukça yüksek işlemlerdir (Guttman, 1984).

### 3.1.3 Paketlenmiş (*Hilbert Packed* ) R-tree:

Kamel ve Faloutsos (1994), önceden bilinen veri setinin, indise daha kontrollü yerleştirilmesi ile %100 yer verimli R-tree yapıları elde etmiştir. Hilbert Packed R-tree bu sınıftaki bir indis yapısıdır. Bunun için, uzlamsal nesne seti, merkez noktalarının hilbert değerlerine göre sıralanıp, bu sıraya göre R-tree'ye yerleştirilir. Yapılan deneylerde, hilbert packing R-tree

gibi, önceden belirli set ile –çevrimdışı- oluşturulan R-tree'lerin, aynı set üzerinden oluşturulan orijinal R-tree'den daha iyi sonuçlar verdiği gösterilmiştir.

### 3.1.4 Sonuç

R-tree indisi, geleneksel B-tree ağacından esinlenerek geliştirilen uzlamsal bir indistir. B-tree'nin ikiye bölme (*split*) ve birleşme (*merge*) gibi yapısal işlemlerinin uzlamsal olarak uygulanmasıyla beraber; B-tree'den en önemli farklılığı, ağaçtaki örtüşmelerdir. Bu problem nedeniyle, indiste aramada disk erişim sayısı her zaman ağacın yüksekliği kadar olamayacaktır. Örtüşme probleminin azaltılmasını hedefleyen ve R-tree çeşitlemeleri olarak adlandırabileceğimiz R+-tree, R*-tree gibi yapılar, daha yüksek performans sağlayan erişim yapılarıdır. Diğer yandan, Hilbert Packed R-tree, önceden bilinen veri setinin, indise daha kontrollü yerleştirilmesi ile %100 yer verimli bir erişim yapısına olanak sağlamaktadır.

R-tree disk erişiminde bir alt sınır garanti etmese de, ortalama performansı oldukça başarılı olup, birçok çok-boyutlu indisleme araştırma ortamında kullanılmaktadır. Örneğin, 3DR-tree (Theodoridis vd., 1996), HR-tree (Nascimento ve Silva, 1998), TPR-tree (Tao ve Papadias, 2002), R-tree üzerine geliştirilmiş tanınmış zaman-uzlamsal indislerden sadece birkaçıdır. Bunların da ötesinde, R-tree, veri ambarları, veri madenciliğinde ve OLAP işlemlerinde hız kazanmak için de kullanılmaktadır.

## 3.2 Hareketli Nesne Erişim Yapıları

Mobil teknolojilerin son senelerde yaygın kullanımının, her alanda olduğu gibi, veri tabanı araştırma konularına olan etkileri göz ardı edilemez. Geleneksel erişim yapılarının, dinamik ortamın getirdiği yeni uygulamalar için yetersiz kalması, yeni erişim yöntemlerinin tasarlanmasını tetikleyen temel unsurdur. Bu yönde, günümüzde yeni açılımlarla devam eden birçok çalışma yapılmaktadır.

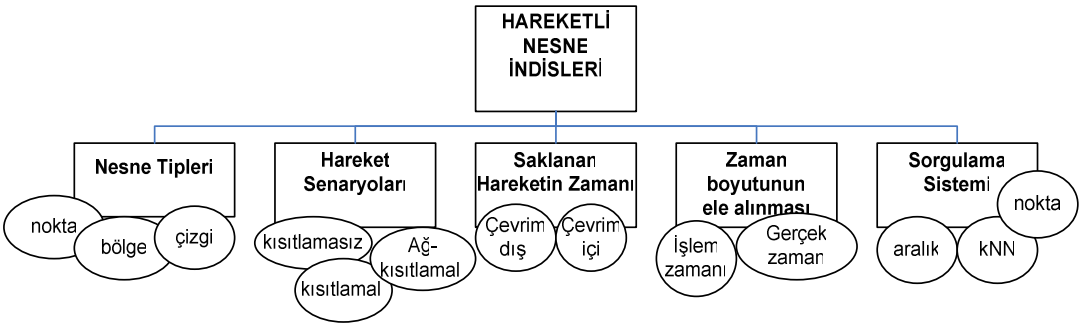
Bu bölümde, ilk olarak hareketli nesne ortamlarında indislemenin, çeşitli gereksinimlere göre çeşitlenmesi (Bölüm 3.2.1) ve literatürde bu probleme olan yaklaşımlar hakkında bilgi (Bölüm 3.2.2) verilecektir. Bu temel bilgilerden sonra, Bölüm 3.2.3 ve Bölüm 3.2.4 ise farklı hareket senaryolarında tanımlanan hareketli nesne indislerine örnekler içermektedir.

### 3.2.1 Hareketli Nesne Erişim Yapıları için Farklı Gereksinimler

Hareketli nesne veri tabanlarında, çok farklı erişim yapıları tasarlanabilir. Bu farklılaşmaya neden olan ana faktörleri Güting ve Schneider (2005), şu şekilde sıralamıştır:

1. Hareketli nesneler farklı tipte olabilir. (nokta, bölge, çizgi gibi)
2. Hareket çok farklı ortamlarda gerçekleşebilir. Örneğin; ortam, 2-boyutlu, 3-boyutlu olabileceği gibi; hareket bir yol-ağı üzerinde veya kısıtlamasız bir ortam üzerinde gerçekleşiyor olabilir.
3. Saklanan ve sorgulanan hareket, geçmiş zaman *(çevrim dışı-offline)* veya o andaki ve/veya yakın gelecekteki *(çevrim içi- online)* hareket olabilir.
4. Zaman boyutunun ele alınmasında, değişik yaklaşımlar olabilir. Örneğin; "zaman" boyutunun temsili için, veri tabanı sistemine bağlı olan "işlem zamanı" (*transaction time*) kullanılabilir veya "geçerli zamana" (*valid time*) dayalı bir yaklaşım esas tutulabilir.
5. Hareketli nesneler üzerinde farklı sorgulamalar tasarlanabilir. Örneğin; zaman-uzlamsal veya en yakın *k* nesnenin bulunması gibi sorgulamalar amaçlanabilir.

Yukarıdaki maddelerde verilen örnekler arttırılabilir. Şekil 3.4'deki diyagram, hareketli nesnelerde farklılaşmayı ana hatlarıyla göstermektedir.




Şekil 3.4 Hareketli Nesnelerin çeşitlenmesi için ele alınan parametreler

**Nesne tipinin,** uzlamsal indis performansına etkisi kaçınılmazdır. Benzer şekilde; hareket eden nesnelerin tipinin, hareketli nesne indisinin verimini etkileyeceği söylenebilir. Hareketli nesne indisi, saklanacak veri tipine bağlı olarak tasarlanmaktadır. Hareketli noktaların indislenmesi; noktaların konumlarının indislenmesi olarak düşünülebileceği gibi, çizgi ile modellenen gezingelerin indislenmesi olarak da ele alınabilir. Örneğin, Nascimento vd. (1999), hareketli noktaların belirli zamanlardaki konum bilgilerini saklayan yapılar üzerine bir çalışmadır. Diğer yandan, hareketli noktanın, belirli zamanlarda konumlarının tutulması ile bu konumlar arası gezingelerin oluşması ve bunların saklanması ile STR-tree (*Spatio-temporal R-tree*) ve TB-tree (*Trajectory bundle tree*) gibi önemli indis yapıları da mevcuttur (Pfoser vd., 2000). Diğer yandan, Tao ve Papadias (2001), farklı nesne tipleri için, hatta zamanla

nesne tipinin değiştiği, MV3R-tree hareketli nesne indisini tasarlamıştır. İleriki bölümlerde incelenecek olan, MVR-tree bu yapının bir kısmıdır.

Bölüm 2.3'de anlatıldığı gibi, hareketin bulunduğu ortamın, hareket üzerindeki kısıtlamalarına **hareketin senaryosu** adı verilmektedir. Örneğin, kısıtlamasız hareket senaryoları için TB-tree (Pfoser vd., 2000), MV3R-tree(Tao ve Papadias, 2001), NSI (*Native Space Indexing*) ve PSI (*Parametric Space Indexing*) (Porkaew vd., 2001) gibi birçok veri erişim yapısı tasarlanmıştır. Denizdeki gemi hareketi veya bir odadaki atom zerrelerinin hareketi, kısıtlamasız hareket senaryo modeline örnek olarak verilebilir. Pfoser ve Jensen (2001) çalışması ise, kısıtlamalı hareket senaryolarına yönelik geliştirilen erişim yapısına örnek olarak verilebilir. Son olarak, hareketin belirli bir ağ üzerinde gerçekleştiği ağ-kısıtlamalı senaryolar için, boyut indirgenmesine dayalı Gezinge-2d R-tree (Pfoser ve Jensen, 2003), FNR-tree (Frentzos, 2003) ve MON-tree (De Almeida ve Güting, 2005) erişim yapıları örnek olarak gösterilebilir. Bu çalışmaların ortak özelliklerine baktığımızda, hareket senaryosuna bağlı olarak temsil edilen hareket modelinin, tasarlanan erişim yapısına verimli bir entegrasyonunun hedef alındığı gözlenmektedir. Örneğin, Bölüm 2.3'de 1,5 boyutlu uzay olarak tanımlanan ağ-kısıtlamalı hareket senaryosunda nesne konumları, 2 boyutlu öklit koordinat sistemi yerine, ağdaki yollar referans alınarak saklanmıştır. Bu yaklaşım, hareket senaryosuna bağlı tanımlanan MON-tree ve Gezinge-2d R-tree gibi erişim yöntemlerinin esasını oluşturmaktadır.

**Saklanan nesne hareketlerinin zamanı da,** indis tasarlamada önemli bir kriterdir. Hadjieleftheriou (2006), literatürdekine benzer bir yaklaşımla, bu noktayı esas alan çalışmaları iki ana gruba ayırmaktadır. Bunlardan birincisi, tarihsel veri tabanı olarak düşünülebilen geçmiş zaman hareketlerinin tutulduğu çevrimdışı *(off-line)* erişim yapılarıdır. Çevrimdışı erişim yapıları, nesnelerin eklenme ve çıkarılma zamanlarının önceden bilinmesi esasına dayanarak geliştirilir ve daha çok bilgi taşıdıkları için, çevrimiçi yapılara göre genel olarak daha verimlidirler. Örneğin, bu tezde incelenen MON-tree, çevrimdışı bir erişim yapısıdır. Diğer yandan, paketlemeli (*packing*) erişim yapıları bu sınıfa dahil edilebilir. Diğer grup çalışmalar ise anlık ve/veya yakın gelecekteki nesne konumlarının tahmin edilmesi üzerine kurulan çevrimiçi *(on-line)* erişim yapılarıdır. Bu yapılarda "şimdiki zaman" (*now*), zamana bağlı sürekli artan bir değişken olarak saklanır. Çevrimiçi erişim metoduna verilebilecek en bilinen örnek Tao ve Papadias'ın (2002) TPR-tree (*Time Parameterized R-tree*) indis yapısıdır. Bununla beraber, çevrimiçi ve çevrimdışı bilgiye erişimi beraber saklayan ve sorgulayan erişim yapıları da mevcuttur (Meng ve Ding, 2003; Pelanis vd., 2006).

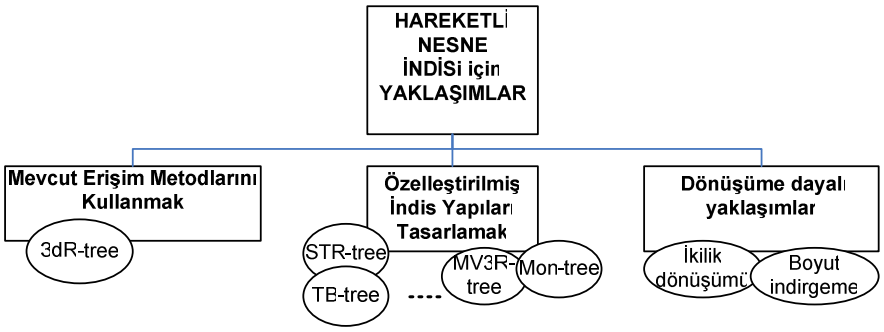
**Zaman boyutunun ele alınması** konusunda**,** genel olarak veri tabanlarına baktığımızda, iki türlü zaman tanımı yapılmaktadır: İşlem zamanı(*transaction time*) ve geçerli zaman(*valid time*). Geçerli zaman, herhangi bir olayın –değişikliğin- gerçek zamanda meydana gelme zamanını ifade ederken; işlem zamanı bu olayın –değişikliğin- veri tabanında kaydedildiği zamanı ifade etmektedir. 3DR-tree (Theodoridis vd., 1996), STR-tree ve TB-tree (Pfoser vd., 2000) geçerli zaman bilgisini destekleyen erişim yapıları için örnek gösterilebilir. Diğer yandan, HR-tree (*Historical R-tree*)(Nascimento ve Silva, 1998) ve MVR-tree (*Multi-version R-tree*) (Tao ve Papadis, 2001) gibi çok-versiyon yapıları ise işlem zamanını esas alan indis yapılarıdır. İşlem zamanı ve geçerli zaman destekli indis yapılarının farkını Tao ve Papadis (2001) şöyle açıklamıştır. İşlem zamanı destekli indis yapılarında, nesneler indise eklenirken önceki versiyonları da indiste korunur ve yeni versiyonlu nesne indiste saklanır. Burada, indiste tutulan zaman bilgisi indise ekleme ve çıkarma zamanlarıdır. Diğer bir ifade ile hareketli bir nesne MVR-tree işlem zamanlı indise, $t_1$ zamanında ilk yerleştirileceği zaman, $[t_1, \infty)$ olarak açık uçlu olarak yerleştirilir. İndisteki bu bilgi, işlem zamanı bilgisidir; çünkü "gerçekte" nesnenin sonsuza kadar yaşamayacağı bellidir. Bir sonraki güncellenme zamanında, örneğin $t_2$; $[t_1,t_2]$ zamanında canlı olan ilk versiyon, $t_2$'de ölü olarak saklanırken yeni oluşan versiyon $[t_2, \infty)$'da canlı olarak indiste saklanır. Bu tip versiyon destekli indislerde, aynı nesnenin farklı versiyonları için ayrı kopyaların tutulması nedeniyle yer verimliliğinde düşme söz konusudur. Diğer yandan, geçerli zamana bağlı indislemede nesnelerin indise eklenmesi geçerli zaman üzerinden olur. Örneğin, 3DR-tree'ye, nesnenin bir sonraki güncellenmenin zamanını bilememekten kaynaklanan $[t_1, \infty)$ gibi, açık-uçlu bir ekleme, düğümler arasında büyük oranda örtüşmelere sebep olacağından desteklenen bir ekleme değildir. Bu yüzden 3DR-tree geçerli zaman bir indistir ve eklemelerde nesnenin geçerli zaman aralıkları olan $[t_1,t_2]$ kullanılır.

Son olarak, kullanılan **sorguların ve çeşitliliğinin** erişim yapısının başarımında etkisi oldukça fazladır. Örneğin, zaman-kesit (*timeslice*) sorgularda (anlık zaman sorguları), HR-tree indisi başarılı iken; uzun aralık sorgularında 3DR-tree daha başarılıdır. MVR-tree ise, gerek kısa aralık ve uzun aralık, gerekse zaman-kesit –anlık- sorgu tiplerindeki başarım hedef alınarak tasarlanmıştır. MON-tree'nin ise şu andaki tasarımında, zaman-uzlamsal aralık sorgularını hedeflediğini görmekteyiz.

### 3.2.2 Hareketli Nesne Erişim Yapısındaki Temel Yaklaşımlar

Önceki altbölümde (Bölüm 3.2.1) farklı özellik gruplarındaki erişim yapılarının tasarımına genel olarak baktığımızda, Pfoser ve Jensen'de (2003) bahsedilen üç farklı ana düşünce ile

karşılaşmaktayız (Şekil 3.5).




Şekil 3.5 Hareketli Nesne İndisi için Yaklaşımlar

Bunlardan **birincisi**, mevcut erişim yöntemlerinin kullanılmasıdır ki; bu çoğu zaman ya mümkün olmamakta veya sorgu işlemede başarısız sonuçlar sergilemektedir. Örneğin, R-tree ve türevleri standart haline gelmiş yüksek performans sağlayan bir erişim yapısıdır. Ancak, hareketli nesneleri temsil ettiğimiz gezingenin depolanması için, R-tree'nin doğrudan kullanımı ile ölü alan artışındaki fazlalık, sorgulama performansında istenen verimin düşmesine neden olmaktadır. Bu sonuç, ileride aktarılan bu tez çalışmasının sonuçlarında da doğrulanmıştır.

**İkinci yaklaşım**, çalışma uzayına ve veri yapısına uygun yeni erişim yöntemlerinin geliştirilmesidir. Bu tip özelleştirilmiş erişim yöntemleri, belirli uygulamalar için yüksek başarım sağlayabilirler. Ancak, bu veri yapısının mevcut ticari veri tabanlarına entegrasyonu oldukça uzun bir süreç gerektirmektedir. Örneğin, R-tree gibi kabul görmüş bir veri tabanının ticari veri tabanlarında kullanımı ancak 12 sene sonunda gerçekleşmiştir. İşte bu yüzden, bu yaklaşım, yani yeni bir erişim metodunun keşfedilmesi, ancak uzun süreli bir yatırım olarak ele alınabilir. Bu yönde yapılmış olan çalışmalara TB-tree, MVR-tree, HR tree, MON-tree örnek olarak verilebilir.

**Son yaklaşımda** ise, birinci yaklaşıma benzer olarak mevcut erişim yapılarının kullanılması esas alınır. Ancak, bu kullanım, çalışma ortamında boyut indirgenmesi gibi bazı dönüşümlerden sonra gerçekleştirilir. Böylece, mevcut ticari veri tabanları üzerinde hareketli nesnelerin verimli olarak tutulması ve erişimi gerçeklenmiş olur. Örneğin, Kollios vd. (1999), hareket modeline uygulanan ikilik dönüşümü *(duality transformation)* ile yeni referans eksenleri üzerinde elde edilen nesnelerin R-tree gibi mevcut erişim yapıları ile daha verimli indislendiğini göstermiştir. Diğer bir örnek, ağ kısıtlamalı senaryolara yönelik olarak tasarlanan ve ağın indirgenmesini esas alan Pfoser ve Jensen'in (2003) çalışmasıdır. Bu

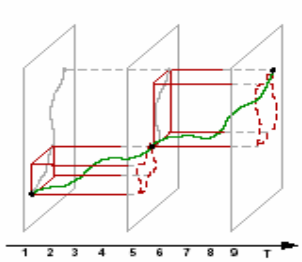
çalışma, 2 boyutlu ağın 1 boyutlu aralık tipindeki bilgiye ve (2+1)-boyutlu konumsal hareketin *(x,y,t)*, (1+1) boyuta *(x,t)* indirgenmesi esasına dayanır. Bu yapının detayları, Gezinge-2d R-tree başlığı altında ileriki bölümlerde açıklanmıştır.

### 3.2.3 Kısıtlamasız Model İçin Tasarlanan Bazı Erişim Yapıları

3DR-tree, zaman-uzlamsal hareketin indislenmesinde oldukça basit bir yaklaşım olmakla beraber, sonradan geliştirilen güçlü indis yapılarıyla karşılaştırma amacıyla birçok araştırmada kullanılmıştır. Hareketli nesne indisleme probleminin kritik noktalarının belirlenmesine ışık tutması açısından, önemli olduğu düşünülerek bu bölümde ilk olarak incelenecektir. 3DR-tree'de bahsi geçen problemlere yönelik çözüm yaklaşımları üzerinden diğer indis yapılarının temel özelliklerine geçilecektir.

#### *3.2.3.1 3DR-tree*

İki boyutlu uzaydaki dinamik nesnelerin ikincil hafızada saklanması için, zaman boyutunun konumsal boyutlarla beraber düşünülmesi oldukça kolay ve yer verimliliği bakımından da oldukça verimli bir indis yapısı karşımıza çıkarmaktadır. Bu yapı Theodoridis vd.'nin (1996) 3DR-tree yapısıdır ve 3-boyutlu MBB'ların *(Minimum Bounding Box)* 1 adet R-tree'de saklanması esasına dayanır. Her dinamik kayda karşılık indiste bir MBB kaydı tutulabileceği gibi; her konum bilgisi yenileme ile yeni bir MBB indise eklenebilir. Birinci durumda tahmin edileceği gibi, ölü alan ve örtüşme problemi oldukça fazladır. Bu yüzden tercih edilen ikinci durum, Şekil 3.6'da görülmektedir.




Şekil 3.6 Ardışık güncellenme zamanları ile oluşan MBBler

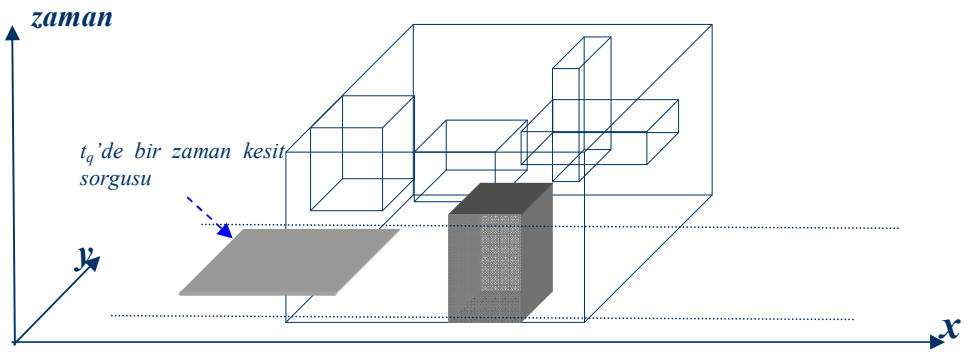
Nesne hareketinin güncellenme noktalarının, en az MBR ölü alan oluşturacak şekilde belirlenmesi Hadjieleftheriou vd. (2006) çalışmasında incelenmiştir. 3DR-tree'nin yer verimliliği yüksek ve yoğun bir indis olması özelliği, zaman boyutunun konumsal boyutlarla direk entegrasyonu ve kayıtların indiste tekrar etmemesi ile sağlanır. İleride, versiyona dayalı

kayıt tekrarının olduğu indis yapıları incelenecektir ki; bu tekrarlama nedeniyle yer verimliliğinde düşmeden bahsedilecektir.

*3DR-tree'de Zaman-kesit Sorgulama:* 3DR-tree'nin anlık-zamanda (veya zaman-kesit sorgulama) zaman-uzlamsal sorgu işlemede performansı düşüktür. Bunun iki ana nedeni vardır:

- ölü alandan kaynaklanan gereksiz düğüm erişimlerinin fazla olması (*false hits*);
- anlık-zaman sorgularının ağaçtaki kayıt sayısına bağımlılığı.

*Birinci problem*; Şekil 3.7'de örnek bir MBB ile gösterilmiştir. Şekilde, yatay karesel sorgu bölgesi, bir düğüm MBB'si ile konumsal olarak kesişirken, bu MBB içindeki bir nesne MBB'si ile (koyu bölge) zamansal olarak kesiştiği görülmektedir. Bu olay, MBB'nin zamansal boyuttan kaynaklanan hacim fazlalıklarının sebep olduğu ölü alandan kaynaklanan bir örtüşmedir. Bu durum, zaman-uzlamsal anlık sorgunun, zamansal olarak düğüm MBB'si ile kesişmesi olasılığını doğrudan arttırarak gereksiz erişimler ile performansın düşmesine sebep olmasına bir örnektir. Diğer yandan; MBB kaydının hacmi, zamanla doğru orantılı olarak arttığı için, uzun ömürlü kayıtlar için (ya da büyük güncellenme periyotlu kayıtlar için) daha çok ölü alanlara neden olup, diğer indis kayıtlarına karşılık gelen MBB'lerle (veya kendisinin önceki MBB'leri ile) örtüşmeyi arttıracaktır.




Şekil 3.7 3DR-tree'de bir zaman-kesit sorgusu (Tao ve Papadias, 2003)

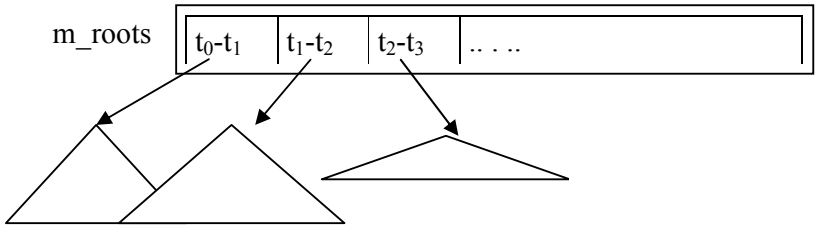
3DR-tree'nin zaman-kesit (*timestamp*) veya kısa aralık (*short interval*) sorgularındaki performans kaybının ikinci nedeni şöyle açıklanabilir; anlık sorgu sadece bir *t* zamanı ile ilgili bir sorgudur ve sorgulamanın verimi için indisin sadece bu *t* zamanındaki kayıtlar üzerinde konumsal analiz yapması istenir. Fakat maalesef, 3DR-tree bütün zamanlardaki kayıtları tek bir R-tree'de tuttuğu için kısa aralık sorgularının maliyeti bütün R-tree'deki tutulan kayıt

sayısına bağlı kalmakta ve oldukça büyük olmaktadır. Sonuç olarak; anlık sorgulardaki bu kaybın, 3DR-tree'nin yer verimliliği yönünden başarılı olmasının negatif bir yansıması olduğunu söyleyebiliriz. Bununla beraber, kayıt versiyonlarının kopyalanmasıyla yer kaybı olmadığı için, uzun-aralık sorgularında başarım artmaktadır.

*3DR-tree'nin Getirdiği Problemlere Çözüm Önerileri:* Zaman boyutunu konumsal özelliklerden ayrıştırarak, 3DR-tree yerine, 2D-Rtree kullanarak zaman boyutundan kaynaklanan örtüşmeden kurtulmuş oluruz. Bu 2D-Rtree kayıtlarında (indis ve veri kayıtlarında) nesnenin konumsal 2-boyutlu MBR'ı ve bu nesnenin canlı olduğu zaman aralık bilgisi tutulabilir. R-tree'nin, nesnelerin zamansal özelliklerinden bağımsız olarak sadece konumsal özelliklerine bağlı kalarak oluşturulması ölü alan artışı ve örtüşme problemlerine karşı kuvvetli bir çözümdür. Ne var ki; bu basit yaklaşım aşağıdaki soruları aklımıza getirmektedir:

- R-tree yapısından dışladığımız zaman boyutu, nasıl temsil edilecektir?
- Konumsal özelliklerle oluşturulan 2DR-tree kayıtların o andaki konum bilgilerini tutabilir; fakat geçmiş konum bilgileri nasıl saklanacaktır?

3DR-tree'de bu sorunlar yoktu; çünkü zaman boyutu zaten indisin yapısal özelliklerini belirliyordu ve nesnelerin bütün geçmiş zaman boyunca bulunduğu konumlara karşılık büyük hacimli birden çok MBB'ler kullanılıyordu. Yukarıdaki sorulara literatürdeki en belirgin cevap; zaman boyutunun, R-tree kayıtları düzeyinde bir boyut olmaktan çıkarıp R-tree ağacı seviyesinde bir boyut olarak düşünülmesidir. Örneğin, zaman ilerledikçe oluşturulan R-tree versiyonları ile geliştirilen bir yapıyı düşünelim. Şekil 3.8'de belirli zaman aralıklarında aktif olan bir gurup R-tree versiyon serisi gösterilmiştir.




Şekil 3.8 Bir grup R-tree versiyonları

Kumar vd.'de (1998) kısmi süreklilik (*partial persistent*) olarak adlandırılan bu fikre göre, kısa ömürlü R-tree ağacı (*ephemeral R-tree*) kayıtların gelişimiyle (ekleme, çıkarma, konum/şekil değiştirme) beraber, belirli kurallar çerçevesinde gelişim gösterir. Bu gelişim

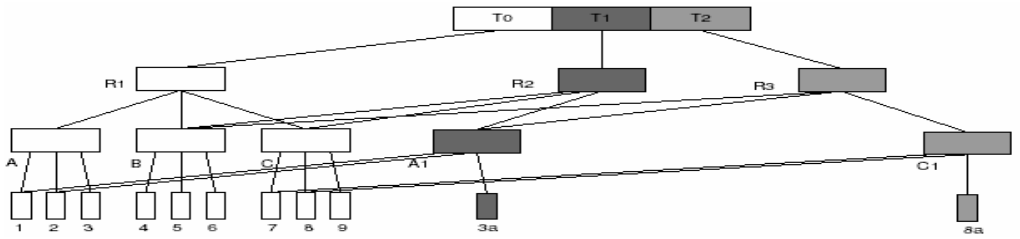
yeni ağaçların oluşması ve bu yeni ağaçların eski ağaçların düğümlerini mümkün olduğu oranda ortak kullanma esasına dayanır. Ağacın bu tarzdaki zamana bağlı gelişimi için, literatürde 2 farklı organizasyon (ve bunların türevleri) dikkat çekmektedir: HR-tree (*Historical R-tree*) ve MVR-tree (*Multi Version R-tree*). Bu organizasyonların ortak noktaları aşağıdaki gibidir:

- kayıt yenileme (ekleme, çıkarma, konum/şekil değiştirme) işleminin geçmiş zamanlar üzerinde değil de; sadece bulunduğu zamana karşılık gelen anlık R-tree üzerinde yapılmasıdır. Zaten; kısmi süreklilik ismini de buradan almaktadır.
- Ağaçların gelişimi ile beraber bazı kayıtların kopyaları indisin farklı zaman dilimlerinde tekrar eder. Kaçınılmaz olan bu fazlalık *(redundancy)* yer verimliliğini düşürmektedir.
- Ağaçlarda tutulan kayıt bilgisi, nesnenin *x-y* uzayında yansıması olan 2-boyutlu *MBR* ve bu nesnenin işlem zamanı *(transactional time*) olarak yaşam sınırlarını belirleyen [t1-t2) bilgi parçalarından ibarettir.

Sonuç olarak; 3DR-tree bütün "zamanı", tamamıyla sadece 1 R-tree'de tutarken; HR-tree ve MVR-tree gibi yapılar zaman boyutunu birbirinden ayrık zamansal aralıklara parçalar; öyle ki, her aralıktaki olaylar bir adet R-tree'de saklanır.

#### *3.2.3.2 HR-tree (Historical R-tree) ve MVR-tree (Multi Version R-tree)*

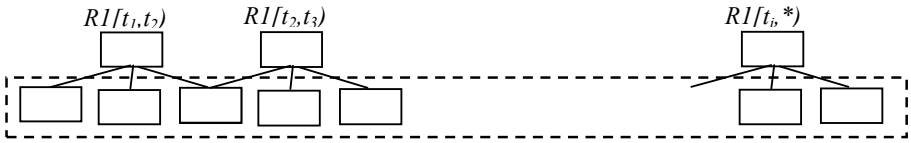
Nascimento ve Silva'nın (1998) tasarladığı HR-tree, örtüşme (*overlapping*) tekniği adı verilen ve kayıtlarında bir değişiklik olmayan düğümlerin R-tree'ler arasında ortak kullanımına dayanır. HR-tree'de birbirini izleyen her zaman kesitine *(time-instant*) bir R-tree karşılık gelmekle beraber ortaya çıkabilecek çok büyük hacim probleminin getireceği yük, bahsi geçen örtüşme tekniği ile bir miktar hafifletilebilir. Şekil 3.9'daki örnek HR-tree'de T1 anlık-zamanında sadece A yaprağındaki 3.dinamik kayıt konum değiştirmekte; bu yüzden T1 anlık-zamanındaki R-tree, sadece A (A→A1 oluyor) ve A'nın köke kadar olan yolundaki bütün düğümleri yeniler (R1→R2 oluyor) ve diğer düğümleri (B ve C) ortak olarak kullanır. T2 anlık-zamanında 8.kayıt konum değiştirdiği için ait olduğu C (C→C1 oluyor) ve C'den köke kadar olan yoldaki düğümler yenilenirken (R2→R3); diğer düğümler bir önceki T1 anlık zamanındaki düğümlerdir.




Şekil 3.9 Örnek bir HR-tree

HR-tree'nin en belirgin özellikleri gerçeklenmesindeki kolaylığın yanında, zaman-kesit ve kısa-zaman-aralık sorgularındaki başarımının yüksekliğidir. Fakat bu yöndeki mükemmellik, her zaman olduğu gibi karşılıksız elde edilmemektedir. Kolaylıkla tahmin edileceği gibi HR-tree'nin en önemli dezavantajı indisin kapladığı alanın çok fazla olmasıdır. Hatta çoğu pratik uygulamada örtüşme *(overlapping*) tekniğinden hedeflenen sonucun alınamaması ile her zaman dilimine bir R-tree karşılık gelmesi ve bunun sonucunda uzun-aralık sorgularının performansının düşmesidir (Tao ve Papadias, 2001).

Tao ve Papadias (2001), MVR-tree adını verdikleri çok-versiyon (*multi-version*) tekniğinde, R-tree'de saklanan kayıtların zamana bağlı gelişimini yönlü ve çevrimsiz bir grafta tutmaktadır. Bu yönüyle HR-tree'ye benzemektedir; fakat her bir anlık-zamanda R-tree'ler oluşması yerine, anlık R-tree'nin zamansal gelişiminin bütün adımları, yönlü ve çevrimsiz bir grafın içine gömülmüştür. Burada kastedilen, anlık ağacın gelişimi için harcanan yer miktarı yapılan güncellenme sayısına lineer orantılı olmasıdır. Şekil 3.8'deki gibi *m_roots* adı verilen lineer bir dizi ile grafın farklı zaman aralıklarındaki bölümlerine erişilebilir. Tahmin edilebileceği gibi *m_roots;* belirli zaman aralıklarına karşılık gelen ağaçların köklerine erişimi sağlayan işaretçileri tutmaktadır. Şekil 3.10 MVR-tree'nin genel şemasını göstermektedir.




Şekil 3.10 MVR-tree yapısal şeması

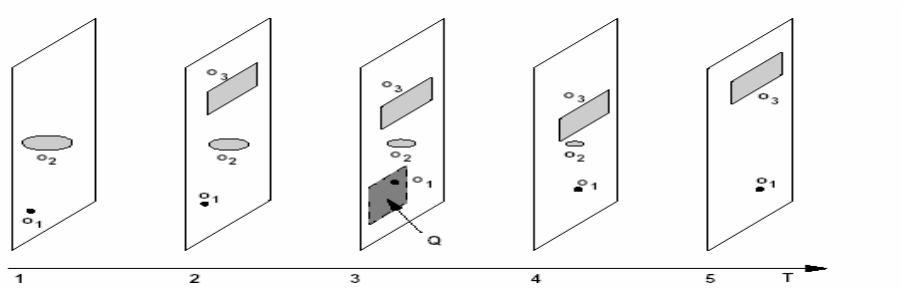
*MVR-tree'de Dinamik Nesne Ekleme/Çıkarma:* MVR-tree'de yapılacak işlemler aşağıdan-yukarıya (*bottom-up*) doğru olmaktadır. İşlem bir ekleme ise; ekleme yapılacak olan yaprak düğüm bulunduktan sonra; yeni kayıt, konumsal *MBR* özellikleri ile ve $[t_{current},\infty)$ zamansal özelliği ile beraber bu yaprak düğüme eklenir. İşlem eğer bir silme ise, yaprak düğüm

bulunduktan sonra, bu düğüm içindeki silinecek dinamik kayıt belirlenir ve bu kaydın zamansal özelliğindeki sonlanma zamanı $t_{current}$ olur. ($[t_{insert\_time},\infty)$ bilgisi $[t_{insert\_time}, t_{current}]$ ile değiştirilir.) Burada silme, lojik bir silmedir; yoksa kayıt gerçekten silinmemektedir. Çünkü bu yaklaşım, tarihsel bilgi tutan veri tabanı (*historical database*) olmasının bir gereğidir.

Kayıt ekleme ve silmede yapılan işlemin neden olduğu değişiklik yaprak düğüm ile sınırlı kalıyorsa bu *yapısal-olmayan-işlem* olarak adlandırılır; diğer işlemler (mesela en az yeni bir yaprak düğüm oluşumuna neden olanlar) *yapısal-işlem* adını almaktadır. Bir kayıt ekleme işleminin yapısal-işlem olması için, ekleme yapılacak olan yaprak düğümdeki eleman sayısının $\boldsymbol{B}$ (düğüm kapasitesi) olması gerekir ki; buna **düğüm taşması** (*node overflow)* denir. Kayıt silme işleminin yapısal-işlem olması için ise, silme sonunda geride $\boldsymbol{B*P_v}$ 'den daha az adet canlı kayıt kalması gerekmektedir ki, buna da **zayıf versiyon aşağı taşması** (*weak version underflow*) adı verilir. Burada $\boldsymbol{P_v}$; bir ağacın bir düğümünün canlı veya ölü olmasını belirleyen bir parametre olup, ($0<\boldsymbol{P_v}<0.5$) özelliğindedir. Buradan hareketle; herhangi bir düğümdeki canlı kayıt sayısının $\boldsymbol{B*P_v}$ 'den aşağıya düşmesi bu düğümün bundan sonra ölü olmasını; bir daha bir değişiklik yapılamaması anlamına gelmektedir. Bu yaklaşım genel anlamda birbirine yakın zamanlarda yaşayan kayıtların birbirine yakın tutulmasına olanak sağlayarak; ağacın I/O performansını önemli ölçüde iyileştiren bir tasarım başarısı olarak düşünülebilir. Bu noktada, Şekil 3.7'deki 3DR-tree düğümünü tekrar göz önüne alalım. Buradaki ölü alan probleminin temel nedenlerinden birisi de; düğümün herhangi bir zamanda tutulabileceği minimum canlı kayıt sayısını belirleyen; $\boldsymbol{P_v}$ gibi bir parametrenin olmamasıdır. Çünkü eğer Şekil 3.7'de düğümün $t_q$ sorgu zamanında bir adet canlı kayıt tutmasına izin verilmeseydi; bu kadar ölü alan olmayacak ve 3DR-tree'nin performansı düşmeyecekti.

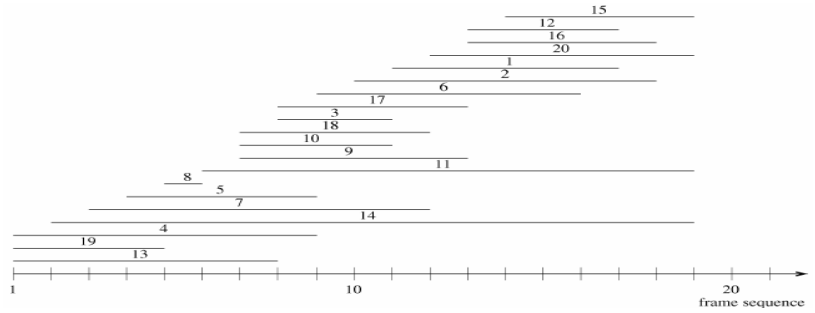
Ağaçtaki *yapısal-işlemler (node overflow*, *weak version underflow)* özel durum olup ikiye-parçalanma (*split*) ve birleşme (*merge*) gibi özel işlemler ile ağacın genel karakteristiklerinin korunması sağlanır. Bu işlemlerin detaylı algoritmaları, Tao ve Papadias'da (2001) açıklanmaktadır.

*Örnek bir MVR-tree:* Kollios vd.'den (2001) alınan örnek bir uygulama ile MVR-tree daha iyi anlaşılabilir. Şekil 3.11'de zaman şeridine asılmış sinema levhalarında, nesnelerin konumsal özellikleri gösterilmiştir. İşte böyle düzenli aralıklı ayrık zamanda alınan örneklerle oluşturulan tarihsel veri yığınının indislenmesi MVR-tree ile gerçekleştirilebilir. Şekildeki zaman-uzlamsal nesnelerin ömürleri her levhaya karşılık gelen anlık-zaman aralıkları ile temsil edilir. Örneğin, şekildeki $o_2$ bölgesi; [1,4] arasında konum ve şekil değiştiren bir nesnedir. 3. levhada ise $\boldsymbol{Q}$ zaman-uzlamsal sorgusu ile $o_1$ noktası yakalanmaktadır.




Şekil 3.11 Zaman şeridine asılmış sinema levhaları (Kollios vd., 2001)

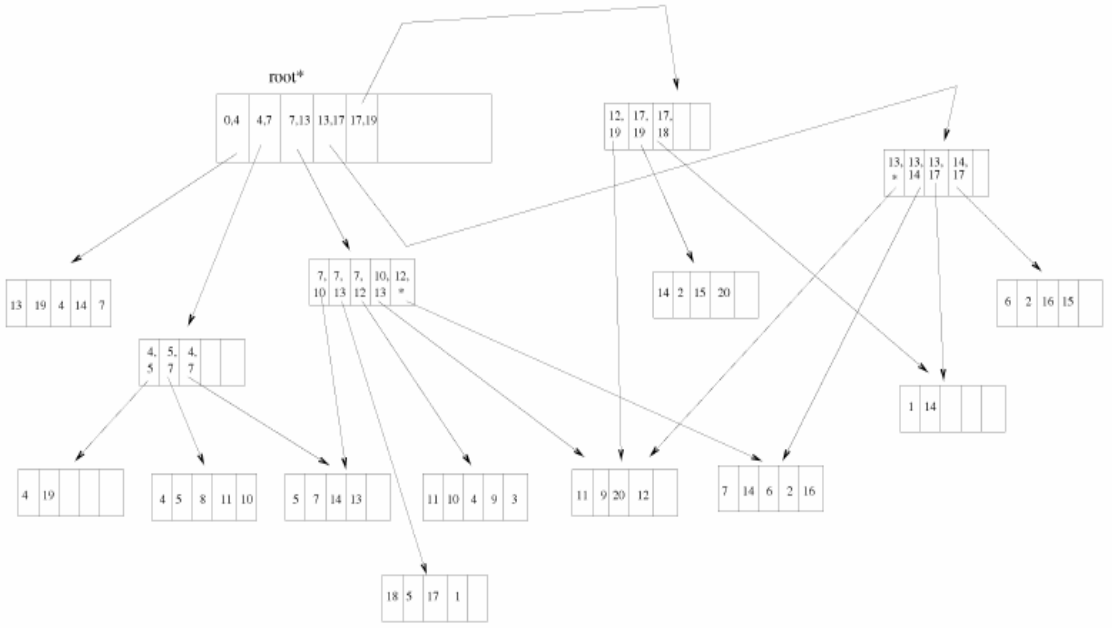
Şekil 3.12'de ise, Şekil 3.11'deki modele uygun dinamik bir ortamdaki 20 nesnenin, 20 anlık-zamandan oluşan bir senaryodaki ömürleri gösterilmiştir.




Şekil 3.12 20 adet hareketli nesnenin ömürleri (Kollios vd., 2001)

Şekil 3.13 ise Şekil 3.12'deki senaryodaki hareketlerin saklandığı MVR-tree yapısını göstermektedir Şekil 3.13'deki MVR-tree'de bazı yaprak düğümleri ebeveynlerinin iki adet olmasından anlaşılıyor ki; MVR-tree gerçekten bir graf yapısındadır. Bu özellik şekilde görülmese de, indis düğümleri için de geçerlidir. Bu şekilde bir yapılanmanın iki ana nedeni vardır. Birisi; "versiyon-bölünmesi" (*version split*); diğeri ise, "düğüm aşağı-taşması sonrası yeniden ekleme" (*reinsertions after node underflow*) işlemleridir. Bu işlemler, ikiye bölme ve birleşme işlemleri sırasında karşılaşılan alt işlemlerdir.

Diğer yandan bazı kayıtlar (indis ve yaprak düğümlerinde) tekrar edebilir. Uzun ömürlü kayıtların birden çok düğümde farklı zaman özellikleri ile tekrar etmesi kaçınılmazdır. Örneğin 14 numaralı kayıt, [3,19] zaman aralığında canlı olup aynı konumsal özellikleri 5 adet yaprak düğümde tekrarlanmıştır.




Şekil 3.13 B=5, $P_v$=0.4, e=1 özelliklerine sahip MVR-tree. (Kollios vd., 2001)

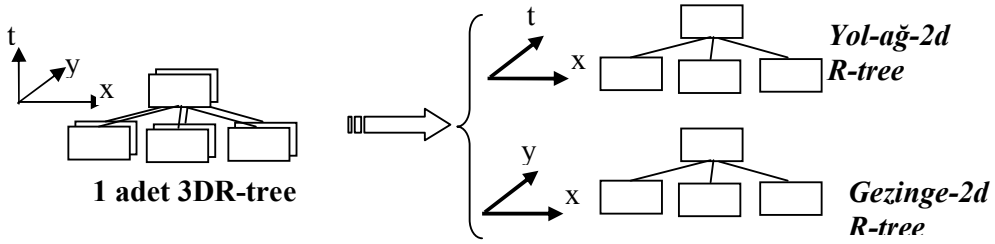
MVR-tree, özet olarak, kısa aralık ve zaman-kesit sorgulamada başarımı yüksek bir yapıdır. Bununla beraber, önemli bir dezavantajı, güncellenmede ekleme ve silme işlemlerinin karmaşıklığı ve bunun sonucu olarak, işlemci yükü ve disk erişim sayısının fazlalığıdır. Tahmin edileceği gibi bu karmaşıklığın esas nedeni, ekleme ve çıkarma işlemlerinin iki ağaçta eşzamanlı ve tutarlı olarak gerçekleşmesidir. Diğer yandan, MVR-tree'nin 3DR-tree gibi yer verimliliği yüksek yapılarla entegrasyonu ile oluşan MV3R-tree, geniş sorgu yelpazesinde başarılı örnek bir erişim yapısıdır.

### 3.2.4 Kısıtlamalı Model İçin Tasarlanan Bazı Erişim Yapıları

Ağ-kısıtlamalı hareket senaryosu modeli, Bölüm 2.3'den hatırlanacağı gibi, nesne konumlarının ağdaki yolları referans alması nedeniyle, 1,5 boyutlu uzlamsal hareketler olarak tanımlanmıştı. Bu esas üzerine ortaya atılan erişim yapıları literatürde çok olmamakla beraber son senelerde bu yönde çalışmalardaki artış dikkat çekmektedir (Vazirgiannis ve Wolfson, 2001; Frentzos, 2003; Pfoser ve Jensen, 2003; De Almeida ve Güting, 2005). Bu bölümde, performans karşılaştırması amacıyla gerçekleştirilen Gezinge-2d R-tree (Pfoser ve Jensen, 2003) ve MON-tree (De Almeida ve Güting, 2005) erişim yapılarının temel bazı özellikleri anlatılacaktır.

### *3.2.4.1 Gezinge-2d R-tree Ağacı*

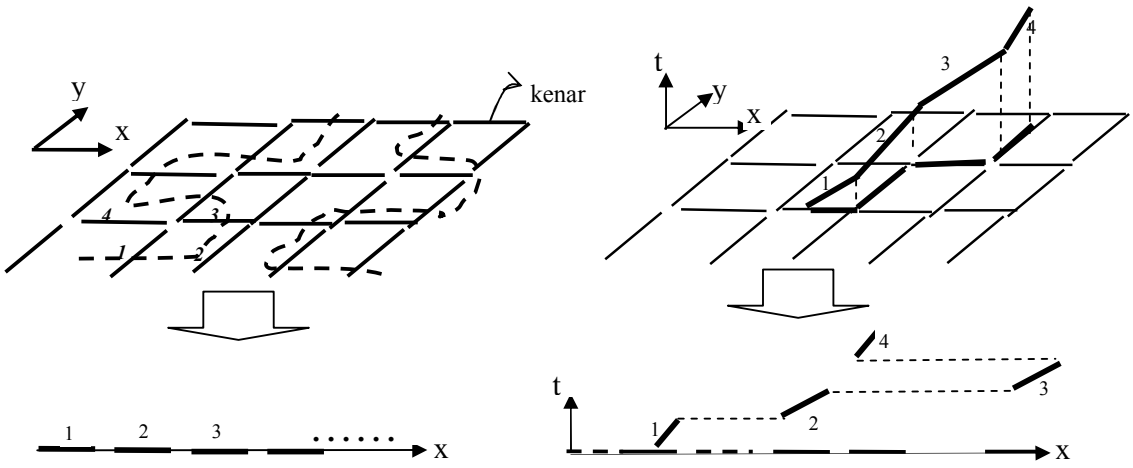
Pfoser ve Jensen (2003) tarafından geliştirilen bu erişim yapısında, Bölüm3.2.2'de bahsedilen yaklaşımlardan üçüncüsü esas alınmaktadır. Hatırlanacağı gibi bu yaklaşımda, R-tree gibi mevcut erişim yapılarının bazı dönüşümler sonrasında kullanımı esas alınmaktadır. Bu dönüşüm, 2 boyutlu yol-ağının 1 boyutta, 3-boyutlu gezingenin ve zaman-uzlamsal sorguların 2-boyutta temsil edilmesi esasına dayanır. Bu dönüşümler ile herhangi bir bilgi kaybı olmamakla beraber; esas amaç, *(x,y,t)* uzayındaki hareketli nesnelerin, mevcut 2-boyutlu indis yapılarını kullanarak indislenmesine olanak sağlamaktır. Şekil 3.14'de tasarımı verilen bu yöntemde, 3DR-tree indisi yerine kullanılan 2 adet 2d R-tree gösterilmektedir.




Şekil 3.14 Gezinge-2d R-tree'nin genel tasarım şeması

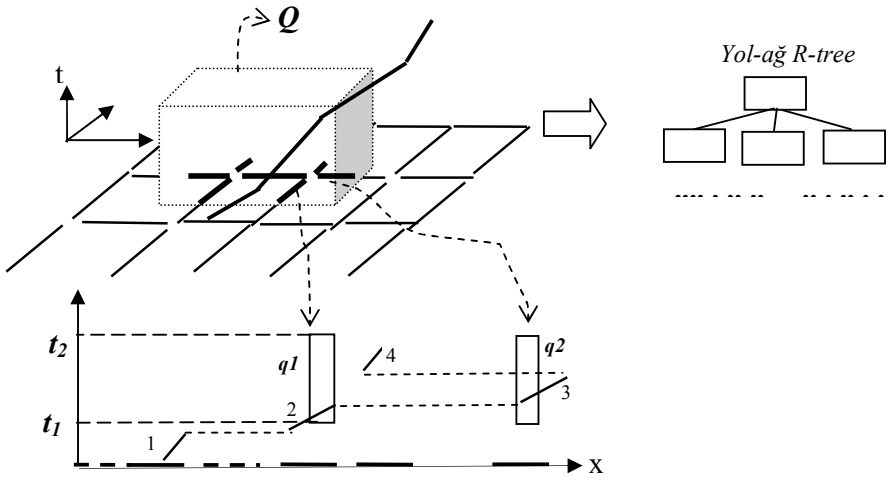
*Yol-ağı ve gezinge dönüşümü:* Yol-ağı ve gezingelerin dönüşümleri Şekil 3.15'de gösterilmiştir. İndirgenmiş yol-ağı, indirgenmiş gezinge ve sorgu uzayları için uzlamsal eksen olmaktadır. Boyut indirgenmesi, ağdaki kenarların hilbert sıralamasına göre 1-boyutta sıralanmasıyla gerçekleştirilir. Şekil 3.15a'daki yol-ağ dönüşümü, hilbert sırasını göstermedeki kolaylık nedeniyle, taramalı ağ *(raster network)* üzerinde gösterilmiştir. Ardışık gelen kenarlar arasında, çalışma uzayının koordinatlarının tamsayı olması varsayılarak, bir tamsayı boşluk bırakılmıştır. Bu nedenledir ki, 1-boyuttaki hilbert sırasında ardışık gelen kenarlar, 2-boyuttaki ağda ardışık olmak zorunda değildir.

Şekil 3.15b'de ise, taramalı ağın bulunduğu *(x,y,t)* uzayındaki 3 boyutlu bir gezingenin, 2-boyutlu *(x,t)* uzaya dönüşümü gösterilmiştir. Şekil 3.15b'deki örnekte nesne, hareketinin 1. parçasında bir kenarın ortasından başlayıp 2. ve 3. kenarları geçerek 4. parçada kenarın ortasında hareketini sonlandırmaktadır. Dönüşüm bu hareket parçalarının, her güncellenme zamanı aralığında, Şekil 3.15a'daki "indirgenmiş ağda" karşılık gelen koordinatlar üzerindeki hareket olarak, *(x,t)* uzayında temsil edilir. Böylece, yol ağında hareket eden bütün nesnelerin gezingelerini 2-boyutlu R-tree erişim yapısında tutabiliriz.




Şekil 3.15 a) yol-ağ boyut indirgemesi b) gezinge boyut indirgemesi

*Sorgu Dönüşümü:* 2-boyutlu gezinge ağacı üzerinde gerçekleştirilecek zaman-uzlamsal sorgunun da benzer şekilde indirgenmiş yol-ağı üzerinden 2-boyutlu *(x,t)* uzayına dönüşümü gerekmektedir. Şekil 3.16'da, ***Q*** zaman-uzlamsal sorgu bölgesinin, indirgenmiş uzlamsal eksen üzerinde oluşturulan birden çok zaman-uzlamsal sorgu parçacıkları ile temsil edilmesi gösterilmiştir.




Şekil 3.16 Zaman-uzlamsal sorgunun indirgenmiş uzaydaki temsili

Bu dönüşümün ilk adımı, sorgunun uzlamsal kapsamındaki yol ağı kenarlarının (veya kenarların bir kısmının) belirlenmesidir. Bu adım, yol-ağındaki kenarları saklayan 2 boyutlu yol-ağ-2d indisi ile gerçekleştirilir. İkinci adım ise, ***Q***'nun uzlamsal bölgesindeki belirlenen kenarların, indirgenmiş uzlamsal eksende karşılık gelen aralıklarının belirlenmesidir. Bu işlemin, gezinge boyut indirgenmesinde bahsedilen işlemden bir farkı yoktur. Üçüncü adım

ise, bu aralıkların sorgunun zamansal süresi boyunca yükseltilmesi ile oluşan zaman-uzlamsal sorgu parçacıklarının oluşturulmasıdır. Şekil 3.16'da, ***Q*** sorgu bölgesinin kapsamındaki 7 adet yol-ağı kenarı ve bunlardan 2 tanesinin $t_1$-$t_2$ arasında oluşturduğu, $q_1$ ve $q_2$ sorgu parçacıkları gösterilmektedir. Diğer sorgu parçacıkları gezingenin bulunduğu kenarların üzerinde olmadığı için gösterilmemiştir. Son olarak oluşan sorgu parçacıkları, *(x,t)* uzayındaki gezinge-2d R-tree ağacı üzerinde işleme girer.

*Sorgu Performans Analizi:* 3DR-tree indisi yerine, 2 boyutlu gezinge-2d indisi kullanılmasıyla, 3-boyutlu uzaydan kaynaklanan büyük örtüşmeler ve ölü alanlar azalmaktadır. Bununla beraber, 2d-ağacının yer verimliliğini arttıran diğer önemli bir özellik, her verinin (gezinge çizgisi) 4 adet reel sayı ile tutulabilmesidir. Buna karşılık, 3DR-tree'de her gezinge çizgisi 6 adet reel sayı ile tutulmaktadır.

Diğer yandan, Şekil 3.16'da örneklendiği gibi, bir sorgudan birden çok sorgu parçacığının oluşması ve oluşan sorgu parçacıklarının sayısının yol-ağının karmaşıklığına bağlı olması gezinge-2d yöntemini olumsuz etkileyen en önemli özelliklerdir. Sorgu performansını etkileyen diğer bir parametre sorguların uzlamsal kaplam genişlikleridir. Bu genişlik miktarının büyüklüğü sorgu işlemede birinci adımda oluşan sorgu parçacıklarının sayısını belirlemektedir ki; bu sayı, gezinge 2d R-tree'de sorgu performansını düşüren birinci etkendir. Bu tez çalışmasında, farklı yol haritaları üzerinde yapılan deneylerde, bu yöntemin performansının, yol- ağına ve sorgu uzlamsal özelliklerine olan bağımlılığı değerlendirilmiş ve sonuçlar Bölüm 5'de grafiklerle aktarılmıştır.

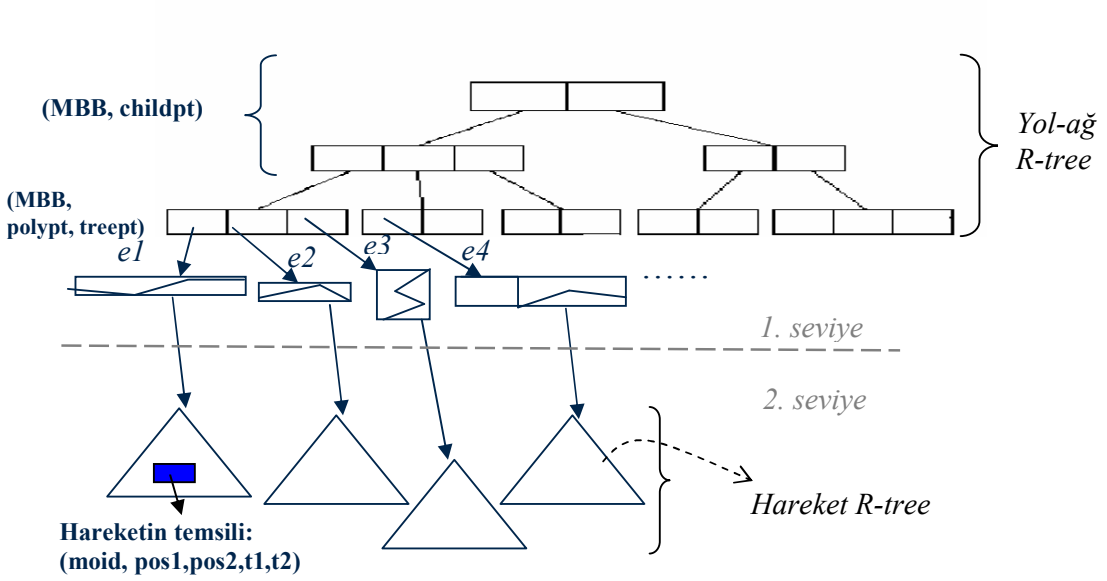
Diğer önemli bir dezavantaj, gezinge-2d R-tree ağacı ile beraber yol-ağını indisleyen bir 2d R-tree'ye olan ihtiyaçtır. Hatırlanacağı gibi, bu indis, sorgu işlemede ***Q*** sorgusunun kapsadığı uzlamsal bölgedeki yol-ağı çizgilerini belirlemede kullanılmaktadır. O zaman diyebiliriz ki; gezinge-2d R-tree tek başıyla 3DR-tree'den daha az yer kaplasa bile, yol-ağı-2d R-tree ile beraber düşünüldüğünde toplam indis kapasitesi 3DR-tree'den fazla olabilmektedir.

Sonuç olarak, Gezinge-2d R-tree boyut indirgemeye dayalı bir erişim yapısıdır. Bu bölümde ise, özellikle *(x,y,t)* uzayındaki 3 boyutlu verinin iki adet 2-boyutlu R-tree ile saklaması yöntemi anlatılmıştır. Bu erişim yapısının geliştirilmesinde, performans başarımından çok mevcut veri yapılarının kullanılması hedef alınmıştır. Bu hedefe ise yukarıda ana hatlarıyla aktarılmış olan boyut indirgeme ile ulaşılmıştır. Boyut indirgenmesinin getirdiği avantajlara karşılık bir sorgudan birden çok sorgu parçacıklarının oluşması performans yönünden bir dezavantaj olarak karşımıza çıkmaktadır.

Pfoser ve Jensen (2003) bazı koşullar altında, gezinge-2d R-tree'nin sorgu işleme performansının (disk erişim sayısı yönünden), 3DR-tree'den daha başarılı olabileceği göstermiştir. Bu koşullar daha önce de bahsedildiği gibi yol-ağı ve sorgu uzlamsal özellikleri ile ilgilidir. Bununla beraber, örneğin, düzenli yol-ağlarında (taramalı yol-ağı gibi) sorgu işleme performansının diğer erişim yapılarına yaklaştığı gözlense de; yapılan performans deneylerinde gezinge-2d R-tree'nin diğer erişim yapılarında daha başarılı olduğu görülmemiştir.

### *3.2.4.2 MON-tree Erişim Yapısı*

Sabit bir yol ağı üzerinde hareket eden nesnelerin bir indis yapısında tutulması ve bu yapı üzerinde geçmiş hareketler ile ilgili zaman-uzlamsal sorgulamaların gerçekleştirilmesi için MON-tree isimli veri yapısı tasarlanmıştır (De Almeida ve Güting, 2005).

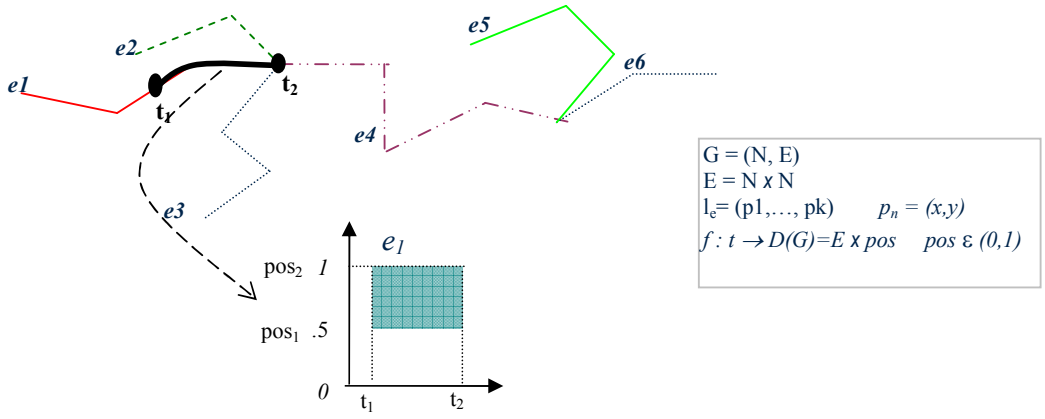



Şekil 3.17 MON-tree genel mimari yapısı (De Almeida ve Güting, 2005).

Şekil 3.17, birden çok 2 boyutlu R-tree'den oluşan MON-tree'nin iki seviyeli yapısal organizasyonunu göstermektedir. Birinci seviye olan üst seviyede, üzerinden trafik geçen kenarların MBR'larını indisleyen "yol-ağı R-tree", ikinci seviyede ise nesne hareketlerini indisleyen "hareket R-tree" yapıları vardır. Şekil 3.17'de görüldüğü gibi, her kenara karşılık gelen üçgen "hareket R-tree" yapısını temsil etmektedir. Sonuç olarak, "yol-ağı R-tree", iki kavşak arasındaki çizgilerden oluşan ve "kenar" olarak tanımlanan nesneleri tutarken, "hareket R-tree" nesnelerin kenar üzerindeki hareketlerini temsil eden dikdörtgenleri

tutmaktadır.

*Kullanılan Yol-ağı ve Hareket Modeli:* Yol-ağı **kenar yönelimli** veya **rota yönelimli** olarak modellenebilir. Şekil 3.18'de **kenar-yönelimli** modellemeye bağlı olarak oluşan *e1* kenarı üzerindeki bir nesne hareketi ve modellenmesi gösterilmiştir.




Şekil 3.18 '*e1'* kenarı üzerindeki nesne hareketinin modellenmesi

Şekil 3.18'de basit bir yol-ağı modeline bağlı olarak tanımlanan 6 adet kenar gösterilmektedir. Yol-ağı, düğümleri (kavşak) tutan *N* seti ve kenarları tutan *E* setinden oluşmaktadır. Kavşak, kendisinden en az 3 farklı yol çıkan noktalardır. Kenarlar ise, kavşakların ikili kombinasyonlarından oluşurken, bu kavşaklar,

$$l_e = (p_1,....p_k), p_n = (x, y) \tag{3.1}$$

ile modellenen çokçizgi (*polyline*) ile temsil edilir. Nesne konumu ise her *t* zamanında, nesnenin kenara göre pozisyonu; diğer bir ifade ile kenarın başına olan uzaklığı,

$$E \times pos, pos \in (0,1) \tag{3.2}$$

ile temsil edilir. Örneğin, Şekil 3.18'de $e_1$ kenarının orta noktasında ($pos_1$=0.5), konum ve hareket bilgisini yenileyen bir nesnenin, kenarın sonuna kadar ($pos_2$=1) bu hareket özellikleri ile yol aldığını düşünelim.*'e1'* kenarı üzerindeki bu nesne hareketine karşılık gelen bölge (*region*), *e1*, *(t,x)* koordinat ekseninde Şekil 3.18'de gösterilmiştir. Bu bölge, "hareket R-tree" indisinde, nesnenin $[t_1,t_2]$ zaman aralığına ait hareket bilgisini temsil etmektedir.

**Rota-yönelimli model** ise, birinci seviyedeki veri miktarını azaltmak amacıyla ardışık kenarların bir araya getirilmesiyle oluşan rotaları kullanır. Kenar yönelimden farklı olarak, hareketler rota üzerindeki konumlarıyla oluşan bölgeler ile temsil edilir. De Almeida ve

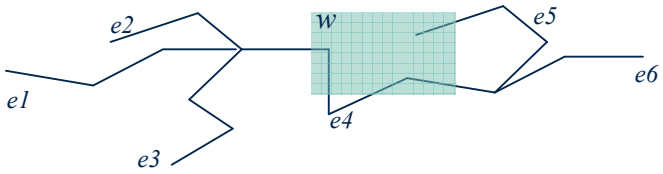
Güting'de (2005) kenar yönelimli ve rota yönelimli MON-tree arasındaki performans karşılaştırmaları üzerinde durulmuş ve rota yönelim MON-tree'nin, erişim performansının daha iyi olduğu gözlenmiştir. Bu tezde, MON-tree yapısını diğer erişim yapıları ile karşılaştırırken sadece kenar-yönelimli model kullanılmıştır.

*MON-tree Ekleme:* MON-tree indis yapısında iki farklı eklemeden bahsedilir. Bunlardan birincisi, yol-ağı R-tree'ye eklenen yol kenarlarıdır. Kenarın yol-ağı indisine eklenmesi, üzerinden ilk nesne geçmeye başladığı zaman gerçekleştirilir. Daha önce belirtildiği gibi bu oldukça akıllıca bir yaklaşımdır, çünkü farklı trafik karakteristiklerinde bazı yollar çoğu zaman boş kalmaktadır.

Diğer ekleme ise, hareket parçacıklarının (diğer bir ifade ile gezinge parçalarının) Şekil 3.18'de açıklanan modele göre karşılık gelen karesel temsillerinin, hareketin gerçekleştiği kenara ait "hareket R-tree" indisine eklenmesidir. Eklemenin yapılacağı "hareket R-tree" indisinin bulunması farklı yöntemler ile gerçekleştirilebilir. Hareketin üzerinde gerçekleştiği kenara ait "hareket R-tree" indisine, "yol-ağı R-tree" üzerinden erişilebilir. Bu, Şekil 3.17'de görüldüğü gibi yol-ağı R-tree'nin yaprak düğümlerinde tutulan "*treeptr*" bilgisi ile gerçekleşir. Fakat bu yöntemde her hareket eklemesinde yol-ağı R-tree üzerinde uzlamsal sorgu işlenmesini gerektirmektedir ki; bu MON-tree yükleme performansını oldukça düşürür. Bunun yerine, MON-tree'yi yükleme sırasında ana hafızada oluşturulan adrese dayalı bir tablo ile "hareket R-tree" indisine sonradan yapılacak erişimler bir disk erişimi ile gerçekleştirilebilir. Böyle bir tabloda *(kenar_id, hareket_R-tree)* ikilileri tutulabilir. Burada *hareket_R-tree*, *kenar_id* ile temsil edilen kenara ait "hareket_R-tree" indisinin kök düğümüne karşılık gelen disk sayfasının adresini tutmaktadır.

Yukarıdaki iki farklı eklemenin R-tree indisine yapılan ekleme algoritmasından yapısal bir farkı yoktur. Sadece "yol-ağı R-tree" indisine eklenen kenar bilgisi *(MBB, polypt, treep*t) olup, *treep*t, *polypt* ile işaret edilen kenara ait "hareket R-tree" indisine işaret etmektedir.

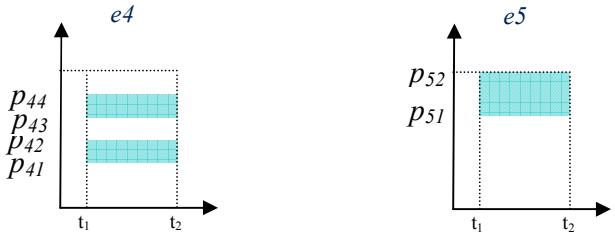
*MON-tree'de Sorgulama:* Mevcut MON-tree veri yapısı, özellikle geçmiş zaman hareket bilgi yığınları üzerinde zaman-uzlamsal aralık sorgulamalarına *(spatio-temporal range querying)* olanak tanımaktadır. Örneğin, Şekil 3.19'daki $Q(w, t_1, t_2)$ sorgusu, $t_1$-$t_2$ zaman aralığındaki herhangi bir anda, $\boldsymbol{w}$ uzlamsal sorgu bölgesi içerisinde bulunan nesnelerin bulunmasını ifade etsin.




Şekil 3.19 Ağ üzerinde zaman-uzlamsal bir sorgu

Bu sorgunun MON-tree indis yapısı kullanılarak işlenmesi 3 aşamada gerçekleşmektedir.

1. ***w* penceresi ile kesişen kenarların bulunması:** Bu işlem, birinci seviyedeki yol-ağ R-tree kullanılarak gerçekleştirilir.
2. ***w* penceresinin, bulunan kenarların hangi kesimlerini içine aldığının bulunması:** ($t_1$, $t_2$) sorgudaki zamansal aralık olmak üzere, ***w'*** = { ($p_{11}$, $p_{12}$, $t_1$, $t_2$), …, ($p_{n1}$, $p_{n2}$, $t_1$, $t_2$) } dörtgenler kümesi, birinci adımda belirlenen kenarların hangi aralıklarının ***w*** bölgesi içinde kaldığını belirten bir settir. Bu işlem ana hafızada yapılmaktadır. ($p_{i1}$, $p_{i2}$) *i* kenarının, ***w*** penceresi içerisinde kalan kısımlarını ifade eder. Örneğin, ($p_{i1}$, $p_{i2}$) = (0,1), *i* kenarının tamamıyla ***w*** içerisinde olması durumudur. ***w'*** penceresinin içerdiği dörtgenler ayrık ve sıralıdır. Şekil 3.20, Şekil 3.19'daki sorgu penceresiyle kesişen $e_4$ ve $e_5$ yollarının pencere ile kesiştikleri bölgeleri göstermektedir.




Şekil 3.20 Sorgu işlemenin ikinci aşamasının sonucu

3. **w' penceresindeki dörtgenler ile kesişen nesne hareketlerinin bulunması:** Bu adımda w' penceresindeki dörtgenler ile kesişen nesne hareketleri, 2. seviyedeki "hareket R-tree" indisleri kullanılarak bulunur.

*MON-tree'de Sorgulama Performansı:* MON-tree iki önemli özelliği sayesinde gezinge-2d R-tree'den daha iyi performans göstermesi beklenmektedir. Bunlardan birincisi, MON-tree birinci seviyedeki yol-ağ R-tree'nin, yol-ağının bütün yollarını içermesi yerine sadece üzerinden trafik geçen yollarını içermesidir. Bu MON-tree indisi yer verimliliğini iyileştirmektedir. Diğer özellik ise sorgulamanın ikinci aşaması ile ilgilidir. Her iki erişim

yapısı da, birinci aşama olan uzlamsal sorgulama aşamasında kenarların bulunmasından sonra, hareket gezingeleri indisi üzerinde sorgulama gerçekleştirir. Bu işlem, boyut indirgeme yönteminde, bütün gezingeleri saklayan "gezinge-2d R-tree" isimli tek bir yapı üzerinde gerçekleştirilirken; MON-tree'de bulunan kenarlara ait olan (sadece bu kenarlar üzerindeki trafiği indisleyen) "hareket R-tree" indisleri üzerinde gerçekleştirilir. Bunun çok önemli bir iyileştirme olacağı açıktır. Ancak, gezinge-2d R-tree bazı yol ağı ve sorgulama uzlamsal karakteristiklerinde MON-tree'ye yaklaşmaktadır. Bu analizler, Bölüm 5'de deney sonuçlarına bağlı olarak aktarılmıştır.

## 3.3 Sonuç

Bu bölümde hareketli nesne erişim yapılarının sınıflandırılması ve probleme olan yaklaşımlar üzerinde durulmuştur. Bunun için ilk olarak uzlamsal erişim yöntemlerinden R-tree ve benzeri yapılar tanımlanmış daha sonra literatürde bulunan hareketli nesne yapılarının tasarımındaki hedefler, genel mimari yapıları ve yapısal özellikler üzerinde durulmuştur. Özet olarak, kısıtlamasız senaryo modeli için 3DR-tree ve MVR-tree incelenmiş; ağ-kısıtlamalı modeli için ise gezinge-2d R-tree ve MON-tree yapıları incelenmiştir. 3DR-tree oldukça basit fakat karşılaştırma amacıyla önemli bir uygulama olmakla beraber; MVR-tree oldukça karmaşık ve geniş uygulama yelpazesinde kullanımı hedeflenen bir zaman-uzlamsal erişim yapısıdır. Gezinge 2d R-tree ve MON-tree ise ağ-kısıtlamalı ortamlar için özel tasarlanmış veri yapılarıdır.

Bundan sonraki bölümde, sistemin tasarlanması kapsamında bu yapıların gerçeklenmesinde aldığımız kararlar ve karşılaştırmaların hangi trafik, ağ ve sorgulama senaryolarında gerçekleştirileceği incelenecektir.

# 4. HAREKETLİ NESNE SİSTEMİNİN TASARIMI

Bundan önceki bölümlerde hareketli nesne veri tabanı tanımlanmış ve mevcut hareketli nesne modelleri ve bazı erişim yapıları hakkında genel bilgilendirme yapılmıştır. Bu bölüm ise, bu tez çalışmasında amaçlanan hareketli nesne sisteminin ana modüllerinin tasarlanması konularını içermektedir. Bu tasarım aşamasında, modelleme, sorgulama sistemi ve erişim yapılarının gerçeklenmesinde bazen literatürde mevcut olan yaklaşımlar kullanılmış; bazen de bu yaklaşımlarda yapılan değişiklikler veya yeni tasarlanan modeller anlatılmıştır.

## 4.1 Sistemin Modellenmesi

Bu tezde, modelleme, *nesnelerin modellenmesi, nesne hareketinin modellenmesi* ve *hareketin gerçekleştiği ortamın modellenmesi* olmak üzere üç farklı başlık altında incelenmiştir. Her bir modelde tanımladığımız kriterler, Bölüm 2.2.3'de anlatılan MOST modelinden esinlenerek geliştirilmiştir. Tahmin edileceği gibi hareketli nesnelerde, nesnenin modellenmesi ve hareket edilen ortamın modellenmesi için "zamandan bağımsız" uzlamsal nesne modelleri kullanılabilir. O zaman mevcut olan uzlamsal nesne modelleri üzerine, "zamana bağlı" hareket modeli entegrasyonu ile hedeflenen hareketli nesne modelleri geliştirilebilir. Bu düşünceden hareketle, ilk olarak, hareket eden nesneler için uygun bir *nesne modeli* ve bu nesnenin *hareket edeceği ortamın modellenmesi* incelenecektir. Daha sonra, ağ ortamındaki hareketlerin temsil edilmesi ve belirsizlik için tanımlanan model; MOST modelinden farklılıkları ile beraber açıklanmaktadır.

### 4.1.1 Nesne Modeli ve Hareket Ortamının Modellenmesi

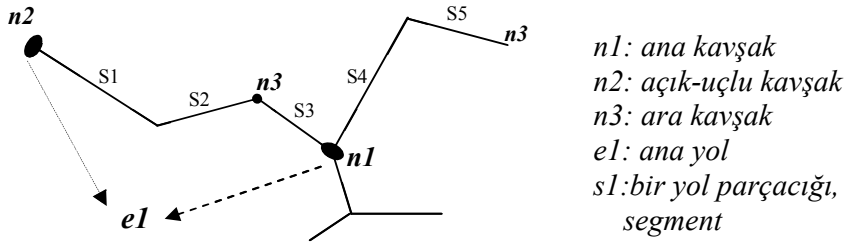
Bu tez çalışmasında aksi belirtilmedikçe, "nesne" ile kastedilen zaman-uzlamsal nesneler, aşağıdaki ana özellikleri sağlamaktadır:

- Nesneler, **nokta**, *Point* uzlamsal tipi ile modellenmiştir.
- konumsal özellikleri "sürekli zamanda" değişmektedir.
- nesnenin büyüklük ve şekli değişmemektedir.
- 2-boyutlu uzayda hareket etmektedir.
- Belirli bir yol-ağı üzerinde, ağ-kısıtlamalı hareket senaryosuna uygun hareket etmektedir.

Hareketli nesnelerde nesne tipi olarak uzlamsal nesne tiplerinden biri seçilebilir. Örneğin, bu

tezde, şehir yol haritasında hareket eden araçları saklayan veri tabanı üzerinde çalışılmaktadır. Bu yüzden nesneler, zamanla tipi değişmeyen iki boyutlu **nokta** ile modellenmiştir*. Nesne konumsal özellikleri gerçek hayatta sürekli zamanda değişiyorsa da, sistemin gerçeklenmesinde belirli güncellenme modellerine göre konumsal örneklerin erişim yapılarında saklanması esas alınmıştır. Bu modeller hareketin ve güncellemenin modellenmesi başlığı altında incelenmiştir.

Amaçlanan hareket senaryosu ağ-kısıtlamalı olmakla beraber, erişim yapılarının performansını karşılaştırırken diğer hareket senaryoları için geliştirilen tekniklerden de faydalanılacaktır. Buna göre kullanılan ağ modeli, örnek bir ağ özerinde Şekil 4.1'de gösterilmiştir.




Şekil 4.1 Yol ağının modellenmesi

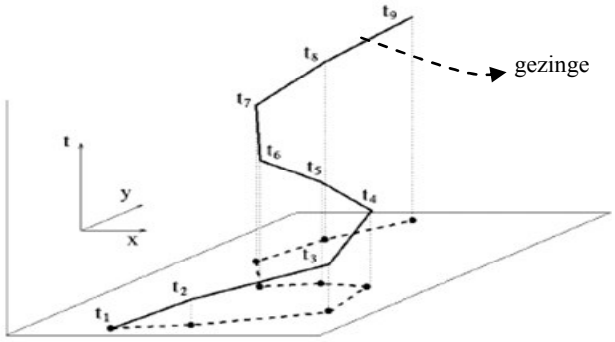
Bu ağ, ana kavşak düğümleri ve bunları birbirine bağlayan ana yollardan oluşmaktadır. Ana yollar ise; bir veya daha çok segmentin (yol parçacığının) ucuca eklenmesi ile oluşmaktadır. Böylece iki ana kavşak arasında ara kavşaklar vardır. Bir ara veya ana kavşaktan en az bir segment çıkmaktadır. Eğer sadece bir adet segment çıkıyorsa bu kavşak açık-uçlu kavşak adını almaktadır. Bu özellikler ile oluşturulan ağ kendi içinde tamamıyla bağlantılı bir özelliktedir.

Ağ organizasyonunun sistem performansında etkin rol almasının en önemli nedeni, indis yapısı gibi ikincil hafızada tutulmasının gerekliliğidir. Sabit disk üzerinde ağ organizasyonu, sistemin gerçeklenmesi kapsamında Bölüm 5.2'de açıklanmıştır.

* Kısıtlamasız model üzerine kurulan erişim yapılarında, iki boyutlu noktanın iki boyuttaki konumsal değerleri kullanılırken, ağ-kısıtlamlı yapılarda nesnenin üzerinde bulunduğu yola göre konumu esas alınmaktadır.

### 4.1.2 Hareketin Modellenmesi

Nesnelerin sürekli zamanda olan hareketinin, sonlu durum makinesi olan bilgisayar saklama biriminde saklanıp, sorgulanabilmesi için hareketin modellenmesi gerekmektedir. Bu modelden beklenen, nesnenin herhangi bir andaki konumsal özelliklerini "mümkün olan" en yüksek doğrulukta bulmaya olanak sağlamasıdır. O zaman, her andaki konum bilgisini saklamanın mümkün olmadığı bir nesnenin gezingesini, belirli güncellenme zamanlarındaki hareket vektörleri üzerine inşa edebiliriz. Tipik olarak, 2 boyutlu *(x,y)* uzayında hareket eden bir nesnenin zamana bağlı konum bilgisi, 3 boyutlu *(x,y,t)* uzayında temsil edilebilir. Basit ve yaygın olarak kullanılan bu temsilde, 3 boyutlu noktalar birbirine bir doğru ile bağlanarak nesnenin gezingesi (*trajectory*) adı verilen çokçizgili yapı oluşur (Şekil 4.2). Bu doğrusal bir ara değerlendirmedir (*enterpolasyon*) ve güncellenme periyodunun artması ile karşılaşılan bilgi kaybı için yapılan bir tahmindir. Daha yüksek doğruluk gerektiren uygulamalarda, doğrusal olmayan yüksek dereceli fonksiyonlar da kullanılabilir.




Şekil 4.2 Uzlamsal bir nesnenin hareketi ve oluşan gezingesi

Bu hareket modeli temel olarak Bölüm 2.2.3'de açıklanan Wolfson vd.'nin (1998) MOST modeline benzerdir. Bu model bu tezde kullanılmakla beraber, MOST modeli ile arasındaki farklılıklar şu şekildedir:

- MOST'da gelecek zamandaki hareket konumları esas alınırken, aynı model geçmiş zaman hareketlerine de uygulanmıştır.
- MOST modelinde nesne, kısıtlamasız (veya her nesnenin tek bir yönde gittiği) hareket ortamındayken, bu tezde kullanılan modeldeki hareket gezingesi, bulunduğu kısıtlamalı yol-ağına bağlıdır.

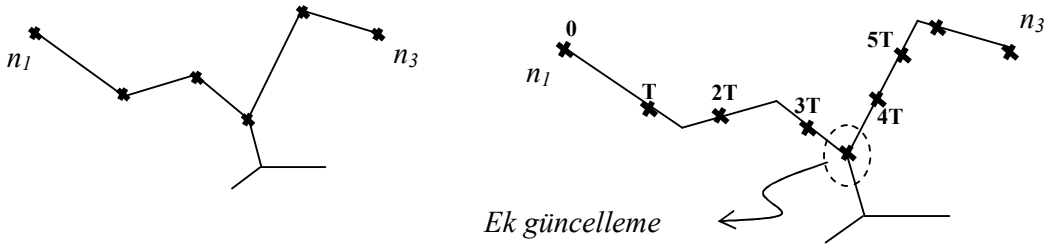
Nesnelerin hızları her güncellenmede $v_{min}$ ve $v_{max}$ alt ve üst hız sınır değerleri arasında düzgün (*uniform*), normal (*gaussian*) veya Zipf dağılımına göre modellenebilir. Düzgün dağılım basit olması, diğerleri ise gerçek hayat uygulamalarını temsil etmede başarılı olmaları nedeniyle seçilmişlerdir. Bu rastlantısal dağılım modelleri, belirsizlik bölgesini, dolayısıyla sorgu sonuçlarının doğruluğunu değiştirebilir. Bunun dışında sistemin sorgu performansına, önemli bir etkisi olmayacağı söylenebilir.

### 4.1.3 Güncellemenin Modellenmesi

MOST modeli kapsamındaki güncelleme modellerinden (Bölüm 2.2.3.2) farklı olarak konum örneklemesi, zamana bağlı ve hareket ortamına bağlı olmak üzere iki farklı yol ile gerçeklenmiştir:

- Segmente dayalı güncelleme
- T-zamansal periyoda bağlı güncelleme

Her segment başında, nesnelerin hız ve yönlerinin değiştiği fikrinden hareketle, nesnelerin yeni konum ve hız bilgilerini bu noktalarda yapması esasına dayanan *segmente dayalı güncelleme* hareket ortamına bağlı olan bir modeldir. *T-zamansal periyot ile güncellemede* ise, hareket ortamından bağımsız olarak nesnelerin her biri kendileri için belirlenen T, güncelleme periyodu sonunda konum ve hız bilgilerini günceller. Şekil 4.3, bu güncelleme modellerini örnek güncelleme noktaları ile açıklamaktadır.




Şekil 4.3 Güncelleme modelleri: *segmente-dayalı* ve *T-zamansal periyoda dayalı*

Şekil 4.3'deki güncelleme modelleri arasındaki temel fark T-zamansal periyoda dayalı güncellemenin kullanıcı tarafından kontrol edilebilmesidir. Segmente dayalı güncelleme ise tamamıyla hareket ortamına bağımlıdır. Şekil 4.3'teki T-zamansal periyoda dayalı güncellemedeki *ek güncelleme* ile Şekil 4.1'de tanımlanan ana kavşaklardaki güncellemeler kastedilmiştir. *Ek güncellemeler,* ağ-kısıtlamalı indis yapılarında nesnenin ana kavşaklardan hangi zamanda geçtiğinin bilinmesinin gerekliliği sonucunda ortaya çıkan güncellemelerdir.

Bu gereklidir çünkü Bölüm 3.2.4'de incelendiği gibi bu tip erişim yapılarında nesnelerin bulundukları yol üzerindeki konumları esas alınmaktadır. T-zamansal periyot dışındaki ekstra güncellemelerin getireceği yük yol ağındaki ana kavşak sayısıyla orantılıdır.

Örneğin, aynı trafik karakteristiklerinde nesnelerin daha küçük hacimli indiste daha yüksek belirsizlik ile saklanmasını istersek, T-zamansal periyotlu güncellemede T değerini arttırmamız gerekir. Bu ancak T-zamansal periyotlu güncelleme modeli ile sağlanabilir. Bu örnekten anlaşıyor ki; segmente dayalı güncellemede hareket ortamının indise etkisi ağır basmaktadır; T-zamansal periyotlu güncellemede ise gerek belirsizlik, gerek indis hacmi T değeri ile kontrol edilebilir.

Hareket modelinde kullanılan diğer bir parametre, en büyük güncelleme zamansal periyodu, **mui** (***m***aximum ***u***pdate ***i***nterval) değeridir. Bu değer, bütün nesneler için genel (*global*) bir değişken olup, hareketli nesne sisteminin hareketliliğinin derecesi ile ilgilidir. Bütün nesnelerin T-zamansal güncellenme periyodu [1, ***mui***] değer aralığında düzenli dağılıma göre belirlenmiştir. Buna göre yüksek ***mui*** değeri sistemin az güncellenen (daha az dinamik), belirsizliği yüksek ve hareketli nesne indisinin daha az hacimli olması anlamına gelirken; tam tersi olan düşük ***mui*** değeri ile sistem dinamikliği yüksek, belirsizliği düşük ve daha fazla hacimli hareketli nesne indis yapılarını oluşturması anlamına gelmektedir. Bölüm 5'de açıklanan sistemin gerçeklenmesi ve deneylerde, burada bahsedilen parametreler ve bunların sistemin davranışına etkisi ve diğer karşılaştırmalı deneyler yer almaktadır.

Bununla beraber, bu tezde gerçeklenen zamana veya hareket ortamına bağlı olan güncellemeler, MOST modelindeki *L.uncertainty* sınır değeri ile tetiklenen güncelleme modelinden farklıdır. MOST modelindeki güncelleme yaklaşımları daha gerçekçi olmakla beraber gerçek hayatta araçlardan GPS sayesinde alınan güncellenmelere dayanmaktadır. Bu tip güncelleme modeli, şu andaki sistemin trafik üreten modülünde bazı değişiklikleri gerektiren ileri bir çalışma olarak düşünülebilir.

### 4.1.4 Belirsizliğin Modellenmesi

Genel olarak, veri tabanlarında, belirsizlik birçok farklı nedenden kaynaklanabilir. Veri kaynağından alınan bilginin doğruluğu, veri tabanında çalıştırılan fonksiyonların sonlanma zamanı, gizlilik, bilgi eksikliğine dayalı sonuçlardaki yanlışlık bu sebeplerden bazılarıdır. Belirsizliğin, klasik veri tabanlarında, özellik seviyesindeki en basit modeli ise NULL değeri kullanmaktır. NULL ile anlatılmak istenen, özelliğin bilinmemesi (gizlilik veya başka bir nedenle) veya tanımlanmamış olması anlamına gelebilir (Cheng vd, 2003, 2004). Gelişmiş belirsizlik modellerine genel olarak baktığımızda karşılaşılan ortak problemlerden en

önemlileri, modelin gerçeklenmesinin getireceği maliyet ve belirsizliğin indis yapısına olan entegrasyonudur.

Hareketli nesnelerde ise, konum bilgisinin güncellemeler ile modellenmesinin bir sonucu olarak gerçek konum ile veri tabanında saklanan konum arasındaki farklılık kaçınılmazdır. Bununla beraber konumsal belirsizliğin kaynakları (Pfoser, Jensen, 1999)'da aşağıdaki gibi sıralanmıştır:

- konumu ölçen sistemin (GPS gibi) konum ölçmede veya örnekleme zamanlamasındaki hatası
- haberleşme kanalındaki gecikme
- veri tabanı güncelleme operasyonundaki gecikme
- nesne güncellemeleri arasında nesne hızındaki değişmeler

Bu tezde, konumsal belirsizlik modeli, listedeki sonuncu kaynak üzerine kurulmuştur. Diğer kaynaklar bu tezde gerçeklenen prototip sistemde söz konusu olmadığı gibi, 2. ve 3. sıradakilerin (Pfoser, Jensen, 1999)'da gerçek hayatta ihmal edilebileceği belirtilmiştir.

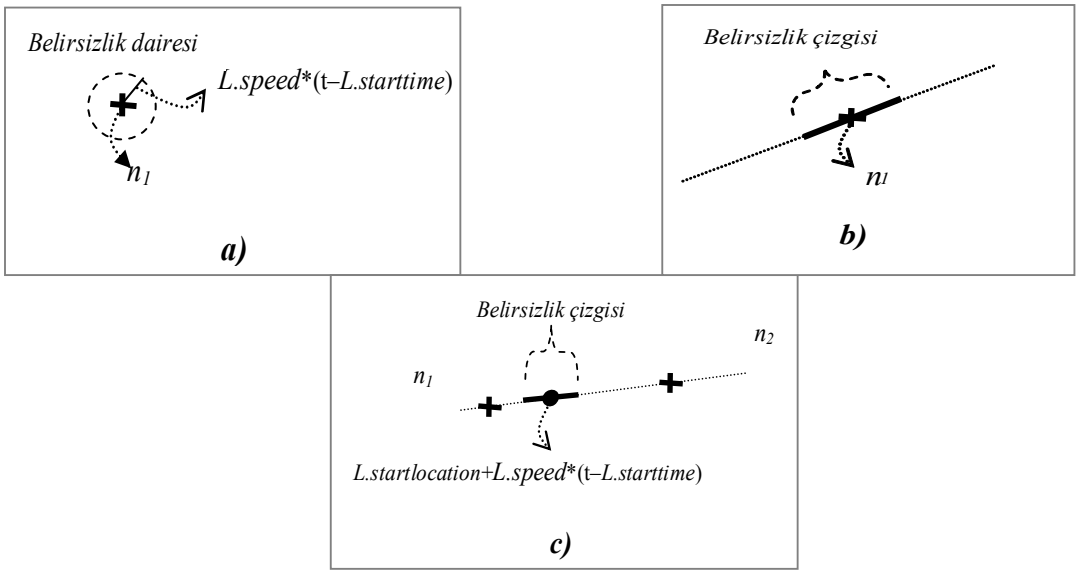
Belirsizlik modeline ihtiyaç duyulmasının nedenleri 2 grupta incelenebilir:

- Belirsizliğin olmaması, nesne hızının değişmemesi ile veya nesne hızındaki değişikliğin her anda takip edilmesiyle sağlanabilir. Her iki durum da gerçek hayat uygulamalarına uygun değildir. Çünkü bir yolda giden nesnenin hızında küçük değişimler kaçınılmazdır ve bunları takip etmek ise her an güncelleme gerektirir.
- Belirsizlik modeline olan ihtiyacın diğer bir sebebi, nesnedeki yön değişimleridir. Serbest harekette ve yol-ağ üzerinde yön değişimleri konum güncellenmesini gerektirir. Bu tür ortamlarda güncellenme sayısını azaltmak için belirsizlik modeli kullanmak gerekir.

Literatürde farklı belirsizlik modelleri üzerine kurulan hareketli nesne veri tabanı prototip çalışmaları vardır. Farklı hareket ortamı ve hareket modelleri üzerinde tanımlanan belirsizlik modellerinden, Wolfson vd. (1999), Barbara vd. (1992), Lakshmanan vd. (1997), Cheng vd. (2003) modelleri bunlardan bazılarıdır. Örneğin, Şekil 4.4a'daki serbest hareket modeli için '**t'** zamanındaki belirsizlik bölgesi, en son güncelleme noktası ($n_1$) merkezli, {*L.speed* ***(t** – *L.starttime*)} yarıçaplı bir daire olacaktır (Wolfson vd., 1999). Burada, *L.speed* sabit olup, *L.starttime* en son güncelleme zamanını ifade etmektedir.

Diğer yandan, nesnenin düz bir çizgide hareket etmesi durumunda belirsizlik bir çizgi ile ifade edilebilir. Şekil 4.4b'de nesne ($n_1$) noktasında güncellendikten sonra '**t'** zamanındaki belirsizlik bölgesi, en son güncelleme noktası ($n_1$) merkezli, {**2***$v_{max}$ ***(t** – *L.starttime*)} uzunluğunda bir çizgi olacaktır. Burada $v_{max}$, hızı [*0,*$v_{max}$] arasında değişebileceği düşünülen nesnenin en büyük hızıdır (Cheng vd., 2003).

Son olarak, belirli bir yönde *pdf(v)* özelliğindeki hız rastlantısal değeri ile hareket eden nesneye ait belirsizlik bir çizgi ile ifade edilebilir. Bu örnek modelde belirsizlik daha azdır. Buna göre, Şekil 4.4c'de çizgi üzerinde hareket eden nesne *L.startlocation* olan $n_1$'den $n_2$ tarafına doğru *L.speed=Exp(v)* ortalama hızıyla, *L.starttime* zamanında yola çıkmaktadır. '**t'** zamanında nesnenin veri tabanındaki konumu *L.startlocation* + *L.speed* ***(t** – *L.starttime*) olup, nesnenin gerçek konumu ise orta noktası { *L.startlocation* + *L.speed* ***(t** – *L.starttime*) } olan belirsizlik çizgisi üzerinde herhangi bir yerdedir (Cheng vd., 2003).



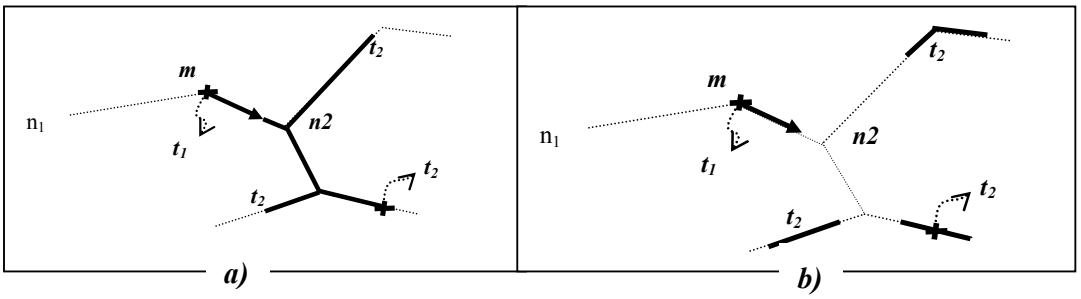

Şekil 4.4 Belirsizlik modellerine örnekler **a)** Serbest hareket, sabit hız **b)** düz çizgide [0,$v_{max}$] aralığında hızla hareket **c)** düz çizgide [$v_{min}$,$v_{max}$]aralığında hızla hareket

Yukarıdaki örnek modellerden anlaşıldığı gibi hareket ve/veya hareket ortam modelindeki karmaşıklığın artması derecesinde, konum belirsizliği azalmaktadır. Nesnenin '**t'** zamanındaki gerçek konumu ise belirsizlik bölgesi ile ifade edilen bölge (daire veya çizgi) içinde herhangi bir yerde olabilir. Belirsizlik bölgesi genişliği nesne hızının en küçük ve en büyük değerlerine bağlıyken; nesnenin bu bölgedeki yerinin dağılımı hızın rastlantısal dağılımına bağlıdır.

(Kalay, Kalıpsız, 2004)'de, Şekil 4.4b ve Şekil 4.4c'deki modelin yol-ağ hareket ortamına

uygulanması üzerinde durulmuştur. Buna göre, son güncellemeden sonra nesne belirsizlik bölgesi hareket modeline bağlı olarak iki farklı şekilde olabilir:

- Yol-ağındaki yolları içeren *sürekli çizgisel bölge* (Şekil 4.5a)
- Yol-ağında bulunması olası olan çizgisel bölgelerden oluşan *ayrık çizgisel bölge.*




Şekil 4.5 Ağ-kısıtlamalı hareket modelinde belirsizliğin modellenmesi **a)** $[0,v_{max}]$ ile oluşan *sürekli çizgisel bölge* **b)** $[v_{min},v_{max}]$ ile oluşan *ayrık çizgisel bölge*

Şekil 4.5'de $\mathbf{t_1}$ zamanındaki güncellemeden sonraki ilk güncelleme $\mathbf{t_2}$ zamanındadır. $\mathbf{t_2}$ zamanından hemen önce nesnenin olabileceği bölgeler nesnenin hareket modeline bağlı olarak Şekil 4.5a ve b'de gösterilmiştir.

Nesnenin iki güncellenme arasında, $[v_{min},v_{max}]$ değer aralığında belirlenen hızla sabit gitmesi durumunda herhangi bir belirsizlik olmayacaktır. Öte yandan, iki güncellenme arasındaki simülasyon adımlarında nesnenin hızı en son güncellenmede belirlenen $[v_{min},v_{max}]$ değer aralığında değişiyorsa oluşan belirsizlik çizgisi (veya çizgi kümesi) o nesneye ait bir özellik olarak saklanmalıdır. Çizelge 4.1 bir nesneye ait konum belirsizliğinin olduğu durumlarda, örnek hız değişimi ve belirsizlik çizgisinin (veya çizgi grubundaki her bir çizginin) uzunluğunu göstermektedir.

Çizelge 4.1 Bir nesnenin iki güncellenme arasındaki hız değişimi ile değişen belirsizlik bölgesi ve sapma miktarı

| ***Sim_adım, t*** | $v_{min}$ *(km/sim_adım)* | $v_{max}$ *(km/sim_adım)* | ***Hız*** *(km/sim_adım)* | ***belirsizlik çizgisi*** *(km)* | ***Sapma miktarı*** *(km)* |
|---|---|---|---|---|---|
| 0, **güncelleme$_1$** | **0.8** | **1.1** | **1.0** | 0 km | 0 km |
| 1 | | | 1.1 | 0.3 km | 0.1 km |
| 2 | | | 1.0 | 0.6 km | 0.1 km |
| 3 | | | 0.9 | 0.9 km | 0 km |
| 4 | | | 0.8 | 1.2 km | 0.1 km |
| 5, **güncelleme$_2$** | **1.1** | **1.5** | **1.1** | 0 km | 0 km |
| 6 | | | 1.3 | 0.4 km | 0.2 km |

Sonuç olarak, belirsizlik modeli, kaçınılmaz olan konum bilgi kaybının kontrol altında tutulması esasına dayanır. Bunun bir sonucu olarak, belirsizlik modeli üzerinde alınan sorgu sonuçları olasılıksal olarak değerlendirilebilir. Olasılıksal değerlendirme ise, hatalı yakalama (*false hits*) ve ıskalamayı (*false misses*) ortadan kaldıracağı için sistemin güvenilirliği için önemli bir adımdır. Buna göre, ağ üzerinde herhangi bir *t* zamanındaki zaman-uzlamsal olasılıksal sorgulamanın; nesnelerin *t* zamanındaki belirsizlik bölgelerinde konumsal dağılımına göre hesaplanabileceği bir sonraki bölümde, sorgulama tiplerinde açıklanmıştır.

## 4.2 Sistemde Amaçlanan Sorgulama Tipleri

Bu bölümde hareketli nesnelerin sorgulanmasında, konumsal belirsizlikten kaynaklanan kesin olmayan sorgu sonuçlarının bulunması incelenmiştir. İlk olarak, hareketli nesneler üzerinde tasarlanabilecek sorgu tipleri kısaca incelenmiştir. Daha sonra, ağ-kısıtlamalı hareket senaryosu kapsamında, Bölüm 4.1.4'de tanımlanan belirsizlik modeline dayalı örnek bir olasılıksal sorgulama algoritması anlatılmıştır.

### 4.2.1 Hareketli Nesneler Üzerinde Tanımlanabilecek Sorgulama Tipleri

Porkaew vd. (2001), zaman-uzlamsal sorgu tipleri için hem uzlamsal değişkenleri hem zamansal değişkenleri içeren aralık *(range)* ve en yakın *k*-nesne anlamında olan kNN *(k nearest neighbor)* sorgularını tanımlamıştır. Bu durum, bir arada düşünülmesi anlamlı olanlara örnek verilerek Çizelge 4.2'de gösterilmiştir.

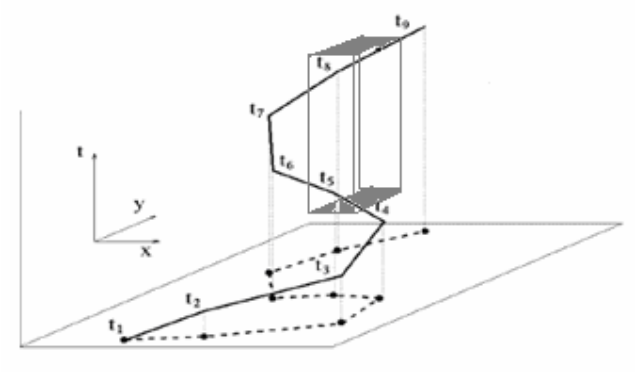
Çizelge 4.2 Hareketli nesneler üzerinde uygulanabilecek bazı sorgu tipleri

| | **Zamansal k-NN** | **Zamansal Aralık** ***(zaman damgalı veya zaman aralığı)*** |
|---|---|---|
| **Uzlamsal k-NN** | Zaman ve konum boyutundaki sıralarının aynı olmasını (*total order*) gerektirdiği için tutarlı cevap kümesi oluşamaz. | "Belirli bir zamanda (veya zaman aralığında) belirli bir noktaya en yakın olan **k** adet nesnenin uzaklıklarına göre artan sırada bulunması" |
| **Uzlamsal Aralık** ***(nokta veya bölge)*** | "Belirli bir bölgeye (belirli bir t zamanından sonra) giren **k** adet nesnenin t ile bölgeye giriş zamanı arasındaki farka göre artan sırada bulunması" | "Belirli bir bölgede, belirli bir zaman aralığında olan nesnelerin bulunması" |

Uzlamsal ve zamansal aralığın miktarına göre, kendi içinde farklı sorgu tipleri oluşur. Örneğin, uzlamsal aralık-zamansal aralık hücresine dahil olacak şekilde, sorgu bir nokta veya

bölge olabilirken; zaman aralığı ise zaman-kesit (*timestamp*), kısa aralık veya uzun aralık olabilir. Diğer yandan, k-NN tip sorgu '**k'** değişkenine göre çeşitlenir.

Şekil 4.6 bir nesnenin zamana bağlı olan iki boyutlu uzaydaki konumunu, (x,y,t) uzayında göstermektedir. Şekildeki nesnenin güncellenme zamanları ($t_1,t_2,t_3,t_{4,..}$) olup konumları ($x_i,y_i,t_i$), *i=1,2,3,4* noktaları ile gösterilmiştir. Bununla beraber 3-boyutlu dörtgen bölge, zaman-uzlamsal bir aralık sorgusuna karşılık gelmektedir. Sorgu belirli bir zaman aralığında, sorgu ile belirlenen *x* ve *y* aralıklarında bulunan nesneleri aramaktadır.




Şekil 4.6 Bir nesnenin hareket rotası üzerinde zaman-uzlamsal aralık sorgusu örneği

Tasarlanan bir sistemde Çizelge 4.2'deki sorgu tipleri bir ölçüm parametresi olabilir. Bununla beraber yukarıdaki örnekler bütün durumları kapsamıyor. İlişkisel sorgular *(relationship query)* ve rota sorguları*(trajectory-based query)* gibi farklı bakış açılarını esas alan sorgulama sınıflandırmaları yapılabilir. Örneğin, "*P poligon bölgesine girinceye kadar aralarındaki mesafe 5km'den az olan **o** ve **n** nesnelerini bulunuz*" ilişkisel bir sorgu iken; "*Bugün sabah saat 8:00-9:00 arasında Beşiktaş bölgesinden ayrılan araçların sonraki 1 saat içindeki rotalarını (yol güzergahları) bulunuz.*" sorgusu ise rota sorgulamaya bir örnektir.

Zaman-uzlamsal sorgulamada diğer önemli bir gruplama, Wolfson vd.'nin (1998) bahsettiği sorgunun işlenmesi ile ilgili olan farklılaşmadır. Buna göre, anlık (*instantaneous*)**,** sürekli (*continuous*) ve bellekli (*persistent*) sorgulama olmak üzere üç tip sorgu işleme tanımlanmıştır. Anlık sorgulamada**,** sonuçlar sorgunun veri tabanında işlendiği zamana bağlıdır. Sürekli sorgulamada ise, sorgu sadece sorgunun ilk sorulduğu zaman değil, bundan sonraki zaman dilimlerinde de tekrar çalıştırılır. Bellekli sorgulama, sorgunun ilk sorulduğu **t** zamanından sonra her zaman diliminde, **t**'deki veri tabanı durumlarına bağlı olarak çalıştırılır. Bellekli sorgulamanın, ilk durumlara bağımlılık hali ile çalıştırılması, sürekli sorgudan ayrıldığı noktadır.

### 4.2.2 Olasılıksal Sorgulama

Hareketli nesne hızının sabit olamaması nedeniyle, veri tabanında saklanan konum bilgisi ile gerçek hayat konum bilgisi arasında konumsal sapma adı verilen bir farklılık olacaktır. Bu farklılığın değeri konum yenileme ile doğrudan ilgili olup belirli bir model üzerine kurulması gerekmektedir. Konum belirsizliğinin veri modellemede, sorgulamada ve indislemede birçok etkileri vardır.

**Modellemenin** incelendiği Bölüm 2'de bahsedildiği üzere, belirsizlik nesnenin konumsal sapması için bir üst sınır olarak düşünülebilir. Veri tabanı herhangi bir zamanda bir nesnenin konum bilgisini bu belirsizlik bilgisi ile verir. Örneğin, belirli bir belirsizlik modeli kapsamında, "***m*** *nesnesinin şu andaki konumu nedir?*" sorgusunun cevabı, "***m*** *nesnesinin şu andaki konumu 1 km'lik belirsizlik ile (x,y)'dir*" olabilir. Bu cevaptaki belirsizlik, nesnenin bulunduğu hareket ortamına göre anlam ifade eder. Örneğin serbest hareket ortamı için nesnenin (x,y) merkezli 1 km yarıçaplı alanda olduğu anlaşılırken; ağ-kısıtlamalı bir ortam için nesnenin (x,y) orta noktalı, 2 km'lik bir yol üzerinde olduğu anlaşılmaktadır. Bazı belirsizlik modelleri ve yol-ağı için tanımlanan modeller Bölüm 4.1.4'de açıklanmıştı.

Belirsizliğin, **sorgulamaya** olan anlamsal etkisi ise sorgulama dilinde ve sorgu sonuçlarında kendini göstermektedir. Örneğin, "*P poligonu içerisindeki nesneleri bulunuz*" sorgusu *may* anlamsal kapsamında, belirsizlik bölgesi, P poligonu ile kesişen nesneleri bulunurken; *must* anlamsal kapsamında, belirsizlik bölgesi bütünüyle P poligonu içerisinde olan nesneleri bulacaktır. Belirsizliğin sorgu sonuçlarında yansıması ise, sonuçların olasılıksal özelliklerinin belirlenmesi şeklinde karşımıza çıkmaktadır. Diğer bir ifadeyle, *olasılıksal sorgulama*, bir olasılık modeli altında sorgunun işlenmesi ve sonuçların olasılıksal değerler ile belirlenmesi demektir. Olasılıksal sorgulamada sorgu sonuçları, her bir sonucun ($r_i$) belirli bir olasılıkla ($p_i$) doğru olmasını ifade etmesiyle, $\{(r_1, p_1),(r_2, p_2),....\}$ şeklinde listelenebilir.

Hareketli nesneler üzerinde önceki altbölümde tanımlanan sorgu yelpazesi kapsamında olasılıksal sorgulama farklı tipte sorgulara uygulanabilir. Sorguda aranan konumsal veya zamansal özellikler nesnelerin birbirine bağlı olan özelliklerini içermiyorsa bu tip sorgulara olasılıksal modeli uygulamak kolaydır. Diğer yandan; tam tersi durumda, yani istenen özellikler nesnelerin birbirine bağlı olan özelliklerini içeriyorsa bu tip sorgularda olasılıksal modeli uygulamak daha zor olur (Cheng vd., 2003). Örneğin, olasılıksal zaman-uzlamsal aralık sorgulamada nesnelerin istenen zamansal andaki/aralıktaki belirsizlik bölgesi sorgu sonucunda olasılığın belirlenmesi için yeterlidir. Diğer yandan, *kNN* tipte sorguların olasılıksal olarak işlenmesinde, bir nesnenin sorgu sonucunda yer alması olasılığı, sadece o

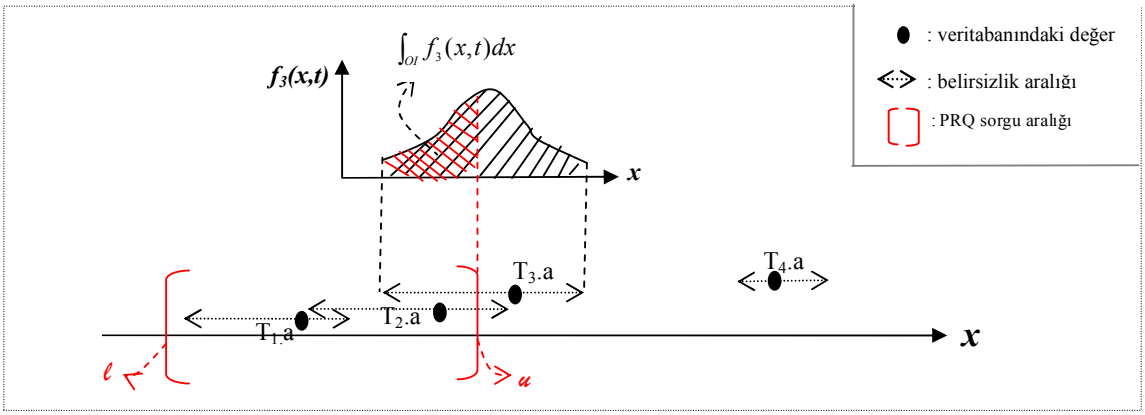
nesnenin belirsizlik bölgesi ile değil, diğer nesnelerin de belirsizlik bölgelerine bağlı olarak belirlenir.

#### *4.2.2.1 Belirsizlik Modelinden Bağımsız Olasılıksal Aralık Sorgulama*

Bölüm 4.1.4, Şekil 4.5'de yol-ağı için tanımladığımız belirsizlik modeli üzerinde olasılıksal aralık sorgusu algoritması, Cheng vd.'deki (2003) Olasılıksal Aralık Sorgulama (OAS) algoritmasından esinlenilmiştir. Belirsizlik modelinden bağımsız olarak tanımlanan bir olasılıksal sorgulama algoritması olan OAS'ın ana tasarımı şu şekildedir. Dinamik bir veri tabanında bulunan **T** setindeki nesnelerin, **a** reel-değerli dinamik özellikleri üzerinden sorgulanmasını ele alalım. *i= 1,2,....*|**T**| olmak üzere $\mathbf{T_i}$, *i*. nesneyi; $\mathbf{T_i.a}$, *i*.nesneye ait olan **a** dinamik özelliğini temsil etmektedir. $\mathbf{T_i.a}$ sürekli rastlantısal bir değişken olup **t** zamanında $\mathbf{U_i(t)}$, belirsizlik aralığında tanımlı, $f_i(x,t)$ olasılıksal yoğunluk fonksiyonu (*probability density function*, *pdf*) ile temsil edilebilir. Zamana bağlı belirsizlik aralığı $\mathbf{U_i(t)}$, $l_i(t) \leq u_i(t)$ olmak üzere, $[l_i(t),u_i(t)]$, değer aralığında olup $\mathbf{T_i.a} \in [l_i(t),u_i(t)]$'dir. $f_i(x,t)$ ise $x \notin \mathbf{U_i(t)}$ için 0 ve $\int_{l_i(t)}^{u_i(t)} f_i(x,t)dx = 1$ şartlarını sağlayan herhangi bir dağılım (düzenli, normal, zipf gibi) olabilir. Bu örnek dağılımlar arasında düzenli (*uniform*) dağılım, belirsizliğin en fazla olduğu dağılımdır.

Şekil 4.7'de 4 adet nesneye ait **a** reel-değerli dinamik özelliğin, $\mathbf{t_1}$ zamanındaki değerleri belirsizlik bölgeleri ile beraber gösterilmektedir. Bununla beraber, Şekil 4.7'de görülen $f_3(x,t)$ fonksiyonu, normal dağılıma uygun olduğu varsayılan 3.nesneye ait dinamik özelliğin, $U_3(\mathbf{t_1})$ belirsizlik bölgesinde sınırlı normal dağılım *pdf* fonksiyonudur. Son olarak, Şekil 4.7'de, $\mathbf{t_1}$ zamanında, [l,u] değer aralığındaki nesneleri bulan *OAS* olasılıksal aralık sorgusu gösterilmiştir. Bu sorgunun sonuç listesinin, $T_1$ nesnesini kesinlikle içereceğini, $T_4$ nesnesini kesinlikle içermeyeceğini ve $T_2$ ve $T_3$ nesnelerini belirli bir olasılıkla içereceğini kestirebiliriz. Örneğin, *OAS* sorgusunu sonucu ,{($T_1$,1), ($T_4$,0), ($T_2$,0.8), ($T_3$,0.4)} olmaktadır.

Yukarıdaki model genel bir belirsizlik modelini ana hatlarını tayin etmektedir. Çünkü mesela, nasıl ki $\mathbf{U_i(t)}$ içerisindeki konum dağılımı tayin edilmemiş; aynı şekilde $\mathbf{U_i(t)}$, belirsizlik bölgesinin zamana bağlı fonksiyonu da belli değildir. Bu koşullar altında, Cheng vd, (2003) herhangi bir OAS[l,u] olasılıksal aralık sorgusu algoritmasını, Şekil 4.8'deki gibi tasarlamıştır.




Şekil 4.7 Olasılıksal sorgulama için örnek bir senaryo

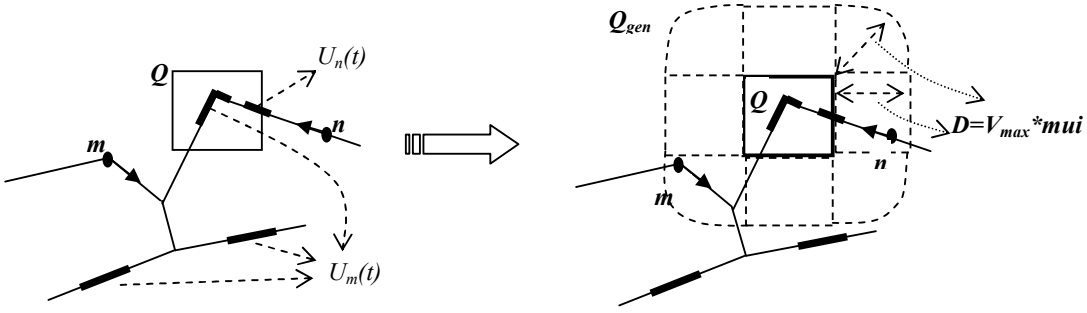
***1***. $R \leftarrow \emptyset$
***2*. for** $i \leftarrow 1$ **to** $|T|$ **do**
(a) $OI \leftarrow U_i(t) \cap [l,u]$
(b) **if** ($OI$ genişliği $\neq 0$) **then**
i. $p_i \leftarrow \int_{OI} f_i(x,t)dx$
ii. **if** $p_i \neq 0$ **then** $R \leftarrow R \bigcup (Ti,pi)$
***3*. return** $R$

Şekil 4.8 OAS algoritması

Şekil 4.8'deki algoritmada; ilk olarak, *2b*'de T setindeki nesnelerden belirsizlik bölgeleri, [l,u] sorgu aralık bölgesi ile kesişenler belirlenir. *2b'*deki koşulu geçen nesnelere ait $f_i(x,t)$ olasılık dağılım fonksiyonunun, *OI* kesişim bölgesi üzerinden entegrasyonu, $p_i$, bu nesnenin dinamik değişkeninin [l,u] sorgu aralık bölgesinde olması olasılığıdır. *2bii* ise, $p_i$ değeri 0 olmayan nesnelere ait *(Ti,pi)* bilgisinin, sonuç listesine alınmasını ifade etmektedir.

#### *4.2.2.2 Yol-Ağı Üzerinde Tanımlanan Belirsizlik Modeli Üzerinde Olasılıksal Aralık Sorgulama*

Bölüm 4.1.4'de tanımlanan ve Şekil 4.5b'de gösterilen belirsizlik modeli üzerinde olasılıksal aralık sorgulama, $OAS^N$ algoritması, Şekil 4.8'de tanımlanan genel yaklaşımdan yola çıkarak geliştirilmiştir. $OAS^N$ algoritmasının tasarımı ve sözde kodu Şekil 4.9'da gösterilmiştir.




**(A)** **(B)**

***OlasılıksalAralıkSorgulama (Q, t)***

2. $R \leftarrow \emptyset$
3. $Q_{gen} \leftarrow f_{gen}(Q, v_{max}, mui)$
4. T $\leftarrow$ HNI($Q_{gen}$, *t-mui, t*)
5. **for** $i \leftarrow 1$ **to** $|T|$ **do**
   i. $p_i = \sum_{PT}(\mathrm{Pr}_Q * p_i')$
   ii. $R \leftarrow R \bigcup (Ti,pi)$
6. **return** *R*

Şekil 4.9 Yol-ağ belirsizlik modeli üzerinde $OAS^N$ işleme algoritması

$OAS^N$ algoritması, *N* yol-ağı üzerinde hareket eden nesnelerden, **t** zamanında **Q** bölgesi içerisinde bulunması beklenenlerin, olasılıksal değerleri ile beraber listelenmesini gerçekleştirir. Bunun için, ilk olarak, Şekil 4.9(A)'da görülen **Q** bölgesi köşelerinden ve kenarlarından, D öklit uzaklığı kadar genişletilerek $Q_{gen}$ genişletilmiş sorgu bölgesi elde edilir. Bu işlem, $D=V_{max}*mui$ olmak üzere $f_{gen}(Q,D)$ fonksiyonu ile gerçekleştirilir. Bu genişletmenin amacı [t-*mui*,t] zamansal aralıkta konumu güncellenen nesnelerin, 3.adımda görülen HNI, hareketli nesne indisi tarafından yakalanmasına olanak sağlamak içindir. $V_{max}$ global değeri, hareketli nesnenin alabileceği en büyük hız değeri olması ve *mui* global değerinin Bölüm 4.1.3'de tanımlanan en büyük güncelleme aralığı olması koşulları altında; D, genişletme değerinin, $V_{max}*mui$ en büyük öklit değeri kadar olacağı açıktır.

**Özellik**: $Q_{gen}$ bölgesinde [t-*mui*, t] zamansal aralığında güncelleme alan nesnelerin **t** sorgu zamanında kendileri (eğer **t** bir güncellenme zamanı ise) veya belirsizlik bölgeleri **Q** bölgesi ile kesişir.

**İspat**: *t-mui* zamanında, $Q_{gen}$ sorgu bölgesi dışındaki herhangi bir noktadan **Q** bölgesine, *mui* zamanda yetişen bir **o** hareketli nesnesini düşünelim. Özelliğin ispatı için, böyle bir **o** nesnesinin mevcut olamayacağını ispatlamamız yeterlidir. $Q_{gen}$ bölge sınırına gelmeden güncelleme alan **o** nesnesinin düz bir rota üzerinde **Q** bölgesine doğru gittiğini düşünelim. **Q** ile $Q_{gen}$ arasındaki bölge genişliği $D=V_{max}*mui$ olduğuna göre, bu nesne *mui* zamanda

güncellense ve $V_{max}$ en büyük hızıyla bu düz rotada gitse de, **Q** bölgesine giremeyecektir. En kısa yol olan öklit uzaklığında bu mümkün değilse, diğer rotalar üzerinden giderek (ağ-uzaklığı ile) **Q** bölgesine girebilmesi mümkün değildir. Öyleyse böyle bir **o** nesnesi yoktur.

HNI indisi, sorgulama verimini arttırması amacıyla, $Q_{gen}$ sorgu bölgesini çevreleyen minimum dörtgeni, MBR($Q_{gen}$) sorgu olarak seçebilir. Böylece, 3. adımdaki zaman-uzlamsal sorgulama sonucunda oluşan T seti, **t** zamanında **Q** sorgu bölgesinde olabilecek nesneleri içermektedir. T setindeki her bir nesnenin, **t** zamanında **Q** bölgesinde bulunma olasılığı diğerlerinden bağımsızdır. Nesnenin **Q** bölgesine giren bir rota üzerinden giderek, **Q** bölgesine girmesi olayının olasılık değeri, 2 çarpandan oluşmaktadır. Bunlardan $p_i'$, Şekil 4.8'de tanımlanan genel belirsizlik modeli üzerindeki OAS algoritmasındakine benzer şekilde, $f_i(x,t)$'nin *OI* kesişim bölgesi üzerinden entegrasyonu ile elde edilir.

$$p_i' = \int_{OI} f_i(x,t)dx \tag{4.1}$$

Diğer çarpan olan $\Pr_Q$ değeri ise, nesnenin **Q** sorgu bölgesine giden rotayı seçmesi olasılığıdır.

$$\Pr_Q = \prod_{j=1}^{u} \frac{1}{k_j} \tag{4.2}$$

Denklem (4.2)'de *u*, nesnenin **Q** bölgesine ulaşan rotadaki ara veya ana kavşakların sayısıdır. Her kavşakta verilen kararın diğerlerinden bağımsız olduğu varsayımıyla $j \in \{1,2,..u\}$ olmak üzere *j*. kavşakta $k_j$ farklı yön arasından, doğru rotayı seçme olasılığı $\frac{1}{k_j}$ olur. Bununla beraber, nesnenin farklı yollar üzerinden de **Q** bölgesine erişebilmesi gerçeğini göz önüne alırsak $p_i$ ifadesini, toplamsal olarak hesaplamamız gerekir.

$$p_i = \sum_{PT} \left( \prod_{j=1}^{u} \frac{1}{k_j} * p_i' \right) \tag{4.3}$$

Denklem (4.3)'de, *PT,* **Q** bölgesine giren farklı rota sayısını ifade etmektedir. Buna göre, denklem (4.3), bir rotanın takip edilmesi olasılığının bu rota üzerinden oluşan belirsizlik bölgesine ait $p_i'$ olasılık değeri ile çarpımının, bütün rotalar için hesaplanıp toplanmasını ifade etmektedir. Bu işlem, Şekil 4.9'da *4i'de* görülmektedir. Son olarak, *(Ti,pi)* bilgi ikilisi, *4ii'*de *R* olasılıksal sorgu sonuç listesine eklenmektedir.

$ONS^N$ algoritmasında kullanılan HNI, hareketli nesne indisi, yol-ağı üzerindeki hareketleri saklamaya uygun bir indis olup, Bölüm 3'deki hareketli nesne indis yapılarından herhangi biri kullanılabilir. (Kalay, Kalıpsız, 2004)'de $OAS^N$ algoritmasına benzeyen, yol-ağ belirsizlik modeli üzerinde olasılıksal nokta sorgusu, $ONS^N$ algoritması tasarlanmıştır. $ONS^N$ algoritması, yol-ağ üzerinde herhangi bir **Q** noktasına **t** zaman içerisinde yetişebilen nesnelerin olasılıksal değerleri ile beraber bulunmasına olanak sağlar.

## 4.3 Sistemde Desteklenen İndis Yapıları

Bu bölümde, gerçeklenen hareketli nesne sisteminde kullanılabilecek erişim yapılarının neler olduğu ve bu kapsamda yapılan katkılardan bahsedilecektir. Buna göre, Bölüm 3'de zayıf ve kuvvetli yönleri ile beraber genel olarak incelenen indis yapılarından 3DR-tree, MVR-tree, gezinge-2d R-tree ve MON-tree bu tezde gerçeklenen indis yapılarıdır. Erişim yapısı tasarımında ele alınan problemler, sınırları ve çözüm önerileri şöyle özetlenebilir:

- Bu tezde ele alınan hareketli nesne tipi, 2-boyutlu uzlamsal uzayda "nokta"dır. Ancak, indiste saklanan bilgi noktalar arası gezinge (çizgi) olabileceği gibi, bu noktaların 2-boyutlu uzayda gerçek konumları da olabilir. Buna göre, gezinge parçalarını indisleyen 3DR-tree, genel nesne tipi için geliştirilen MVR-tree ve nokta tipinde nesneleri indisleyen MON-tree ve gezinge-2d R-tree performansı değerlendirilen erişim yapıları olarak belirlenmiştir.
- Bu tezde hedeflenen hareket ortamları (hareket senaryoları) ağ-kısıtlamalı modele uygundur. Bu hareket senaryosu kapsamında boyut indirgeme yöntemine dayanan gezinge-2d R-tree ve MON-tree yapıları incelenecektir. Bununla beraber, kısıtlamasız ortamlar için birçok indis yapısı (STR-tree, TB-tree, MVR-tree gibi) geliştirildiği için, bunlardan MVR-tree ve 3DR-tree'nin ağ-kısıtlamalı senaryoda üretilen trafik için performansı da değerlendirilecektir.
- Nesnelerin geçmiş zaman (çevrimdışı) ve o andaki ve yakın gelecekteki (çevrimiçi) hareketleri saklanıp sorgulanmaktadır. Nesnelerin geçmiş zaman bilgileri sorgulanabilmektedir ve bütün yapıların bu sorgulamalardaki performansları karşılaştırılmaktadır; ancak belirsizlik modeline dayalı olarak geliştirilen OAS ve $OAS^N$ gibi algoritmalar şu an için sadece modelleme kapsamında tasarlanmıştır.
- İndisteki güncellenme zamanları, diğer bir ifade ile nesnenin canlı olduğu zaman aralığı, geçerli zaman olup, işlem zamanı ile aynı olduğu kabul edilmiştir. 3DR-tree ve

MON-tree'de desteklenen zaman kavramı geçerli zaman (*valid time*) iken; MVR-tree yapısında zaman işlem zamanıdır *(transaction time)*.

- Sorgulama sistemi zaman-uzlamsal aralık sorgulama sistemidir. Tezde incelenen bütün indisler zaman-uzlamsal aralık ve nokta sorgularını desteklemektedir. Bahsi geçen bazı indis yapıları (TPR-tree gibi), kNN sorgulama için uygun yapısal özelliklere sahipse de, bu tezde bu konu üzerinde durulmamıştır.

Bunlara ek olarak, nesnelerin hareket halinde bulundukları ortamın homojen olduğu, yol-ağının statik olduğu ve bu özelliklerin zamanla da değişmediği varsayılmıştır. (Yol ağının gerçeklenmesi Bölüm 5'da açıklanmıştır)

## 4.4 Sistemde Gerçeklenen Ara Bellek Organizasyonu

Gerçek zamanlı çalışması beklenen bir veri tabanında en önemli performans kriteri sorgu sonuçlarının elde edilmesi için geçen süredeki başarımdır. Sorgu çalışma zamanı olarak isimlendirebileceğimiz bu sürenin başarımını belirleyen en önemli bileşenler kullanılan indis yapılarının kalitesi, indis üzerinde ara bellek organizasyonu, birbiriyle ilgili kayıtların sabit diskte aynı veya yakın sayfalarda tutulması ve sorgu işlemede yapılabilecek optimizasyonlar gibi geniş bir yelpazeyi kapsamaktadır. Günümüzde kullanımı yaygınlaşan çok boyutlu veri ortamları kendilerine özgü indis yapıları, ara bellek organizasyonları ve diğer modüllerde yeni algoritmaların geliştirilmesini gerektirmektedir. Bu bölümde, hareketli nesne indis yapısının performansının iyileştirilmesi kapsamında geliştirilen, **monpar** adını verdiğimiz ara bellek organizasyonu incelenecektir.

### 4.4.1 Ara Bellek

Ara bellek, disk erişim sayısını azaltmak için ana hafızada ayrılan tampon bölgedir. Bir sorgunun işlenmesi için gerekli olan disk sayfalarının ara belleğe yerleştirilmesi, bu sayfaların aynı sorgu içinde veya aynı işlem (*transaction*) kapsamında kullanılması ihtimalinin yüksek olması nedeniyle disk erişim sayısının azalmasına yardımcı bir iyileştirmedir (Kahveci vd., 2000). Ara bellek kapasitesinin dolmasıyla, ara bellekten en uygun disk sayfasının seçilmesi ve diske geri yazılması gerekmektedir. Sayfa yerleştirme (*page replacement*) algoritmaları denilen seçim kriterleri ile ara bellek organizasyonu sağlanmış olur.

Örneğin, LRU (***L***east ***R***ecently ***U***sed) organizasyonunda, ara bellekteki disk sayfaları arasından "kurban" olarak en uzun zaman kullanılmayan sayfa seçilir (O'Neil vd., 1993). Bu kriter, farklı erişim karakteristiklerine sahip disk sayfalarının olduğu bir uygulamada doğru

"kurban" sayfasını seçmek için yeterli olmayacaktır. Çünkü en son kullanım zamanı yanında kullanım sıklığı (frekansı) da seçim sırasında önemli bir kriterdir. Örneğin, ara belleğe yerleşen kullanım frekansı çok düşük olan bir sayfanın, ara bellekten çıkarılması oldukça zaman alacaktır; çünkü ara bellekteki en son erişim zamanı en yüksek olan bu sayfadır. Bu da yüksek frekanslı diğer sayfaların ara bellekten atılmasına sebep olacaktır. Diğer taraftan LFU (***L***east ***F***requently ***U***sed) organizasyonunda ise, "kurban" olarak en az kullanılan sayfa seçilir. Bu organizasyondaki temel problem ise, sayfa kullanım frekansındaki değişimlerin takip edilememesidir. LRU ve LFU yaklaşımlarının tek başına çözemediği problemler LRU-*K* algoritması ile belirli bir düzeyde çözülmüştür. O'Neil vd.'deki (1993) LRU-*K* isimli sayfa seçim algoritması ile her disk sayfasına ait en son *K* erişim zamanları tutulmaktadır. Kurban seçilen sayfa, *K* önceki erişim zamanı en uzak olan sayfadır. O'Neil vd.'de (1993), LRU-*2*'nin LRU-*1*(normal LRU)'e göre daha başarılı sonuçlar verdiği gözlenmiştir.

Önceki paragrafta açıklanan bazı klasik ara bellek organizasyonları, işletim sistemlerinde sanal hafızanın kullanım verimini yükseltmek amacıyla disk sayfası kulanım istatistikleri esas alınarak geliştirilmiştir. Bunun yanında, hiyerarşik mimarideki bir disk erişim yapısındaki sayfaları tutan ara belleğin organizasyonu, klasik sayfa yerleştirme algoritmalarından farklı olarak bu erişim yapısının mimarisiyle ilgisi olması beklenir (Leutenegger vd., 1998). Örneğin, hiyerarşik bir indis yapısı olan ve uzaysal veri tabanlarında kullanılan R-tree veri yapısı için tasarlanacak bir ara bellekte, R-tree ağacının kök-düğüm ve yüksek seviyelerdeki diğer ağaç düğümlerinin tutulması performansı olumlu etkileyecektir. Bununla beraber, indislenen veri geometrik cisimler olduğundan, ara bellekten herhangi bir sayfanın çıkarılması için yapılacak bir seçimde küçük hacimli (alanlı) kayıtlar içeren disk sayfasının tercih edilmesi daha iyi olacaktır. Çünkü disk sayfalarına düzenli bir istek dağılımı olduğu varsayımı altında küçük hacimli nesne tutan disk sayfasının istenmesi olasılığı daha düşüktür. Bu fikirden hareketle, Brinkhoff (2002a), tasarladığı "*Spatial*PageReplacement" yerleştirme algoritmasında seçim kriterini, R-tree disk sayfalarının içerdikleri bölgelere ait uzlamsal özellikler olarak belirlemiştir. Brinkhoff (2002a) farklı sorgu dağılımları altında yaptığı deneylerde, *Spatial*PageReplacement organizasyonun, LRU ve türevi algoritmalarına göre daha başarılı olduğunu göstermiştir.

Uzlamsal veri tabanları için yukarıda anlatılan örnekler, çok-boyutlu diğer veri tabanlarındaki indis yapıları için yeni ara bellek tasarımlarının gerekliliğine ışık tutmaktadır. Bu tezde, Bölüm 3'te anlatılan MON-tree hareketli nesne indisi için, geliştirilen **monpar** isimli ara bellek organizasyonu tasarlanmıştır (Kalay ve Kalıpsız, 2006) .

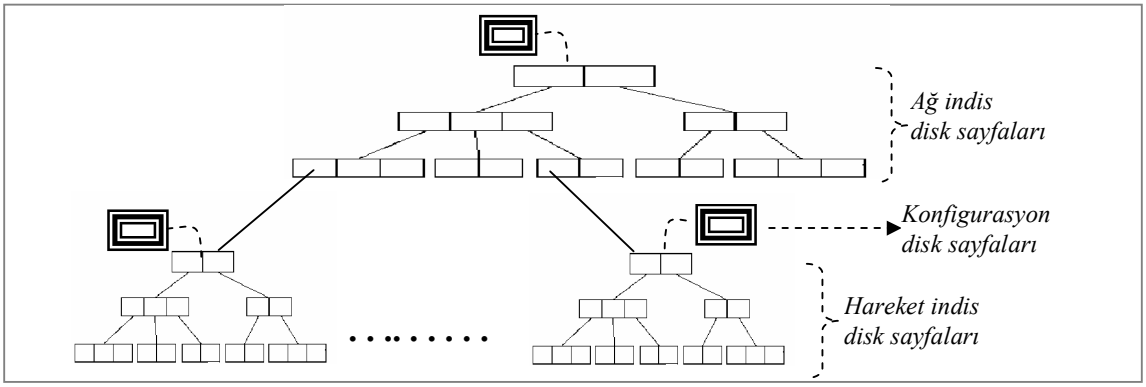
### 4.4.2 MONPAR (MON-tree page-replacement)

Bir grup R-tree'nin, 2 seviyeli kendi içinde hiyerarşik organizasyonu olan MON-tree için hedeflenen ara bellek organizasyonu için, Brinkhoff'un (2002a) bir adet R-tree için tasarladığı *Spatial*PageReplacement ara bellek organizasyonundaki yaklaşım esas alınmıştır. *Spatial*PageReplacement sayfa yerleştirme algoritması denklem (4.4)'deki seçim kriterine dayandırılmıştır.

$$\{p | p \in buffer \wedge (q \in buffer \Rightarrow SC(p) \leq SC(q))\} \tag{4.4}$$

Buna göre, ara bellekten seçilecek olan disk sayfasının, diğer bir ifadeyle indis düğüm sayfasının, *SC* fonksiyonu ile elde edilen uzlamsal kriter değeri minimum olan sayfa olması gerekmektedir. *SC* fonksiyonunda kullanılacak uzlamsal seçim kriteri, Beckmann'da (1990) tanımlanan kriterlerden biri olabilir. Örneğin, *SC* fonksiyonu, disk sayfasına ait MBR *(minimum bounding rectangle)* alanı veya marjı *(margin)* bulabilir. Brinkhoff'un (2002a) deney sonuçlarında sayfa MBR alanının, ara bellek organizasyonunda daha iyi bir seçim kriteri olduğu gösterilmiştir.

Sayfa içeriklerinin kurban seçiminde kriter olması için, MON-tree'deki farklı içerikli sayfaların ara bellek organizasyonunda kendi içinde gruplanması gerekir. Çünkü MON-tree'de en az 2 farklı içerikli disk sayfası vardır: *Ağ-indisi disk sayfaları* ve *hareket indisi disk sayfaları*. Bunlara ek olarak, MON-tree gerçeklenmesine bağlı olarak her R-tree'ye ait konfigürasyon bilgilerinin tutulduğu disk sayfalarının mevcut olması durumunda, toplam indisdeki R-tree ağacı kadar *konfigürasyon disk sayfaları* mevcut olacaktır. Buna göre MON-tree'de olabilecek disk sayfaları Şekil 4.10'deki gibi gruplanabilir.




Şekil 4.10 MON-tree indis yapısında içerik bilgisine göre disk sayfalarının gruplanması

**monpar** algoritması, *SC* uzlamsal içerik seçim kriterinin ağ-indis ve hareket indis

sayfalarında ayrı uygulamasıyla gerçekleştirilmiştir (Kalay ve Kalıpsız, 2006). Konfigürasyon sayfalarının ara bellekten atılmaları ise, FIFO veya LRU gibi klasik seçim kriterleri ile gerçekleştirilebilir. Şekil 4.11 **monpar** algoritmasının sözde kodunu ana hatları ile göstermektedir.

```
r: istek sayfa
N: ağ indis sayfaları seti
M: hareket indis sayfaları seti
H:  konfigurasyon indis sayfaları seti
C := ara bellek kapasitesi
SC(p): area(MBR(p)) //bir 'p' disk sayfasının MBR alanının hesaplanması
monpar( ){
   H=N=M= ø
   if ('r' ara bellekte)    // devam
   else
      tip := 'r' nin tipini belirle
      if (|H|+|N|+|M| < C){
        'r' sayfasını ara bellekte tipine karşılık gelen sete ekle
        C++;
      }
      else if(|H|+|N|+|M| == C) {
          victim=null;
          if (tip == ağ-indis sayfası)
                victim := {v | v ∈ N ∧ (q ∈ N ⇒ SC(v) ≤ SC(q))}
          else if (tip == hareket sayfası)
                victim := {v | v ∈ M ∧ (q ∈ M ⇒ SC(v) ≤ SC(q))}
          else
                victim := {v | v ∈ H ∧ v = LRU (H)}
          victim sayfasını diske yaz.
          'r' sayfasını ara bellekte tipine karşılık gelen sete ekle
      }
}
```

Şekil 4.11 **monpar** sayfa yerleştirme algoritması

Şekil 4.11'deki sözde programda görüldüğü gibi ara bellek kapasitesi dolduğu anda, N, M, H setlerinin ara bellekteki dağılımı belirlenmektedir. Diğer bir ifade ile ara bellek dolduğu andaki |N|, |M| ve |H| değerleri üzerinden sayfa seçim algoritması çalıştırılmaktadır. Bu dağılımın gelen sorgu karakteristiklerine göre değişebilmesini hedef alan uyarlamalı yaklaşımlar daha detaylı istatistiksel analize dayandırılabilir. Bu yönde yeni tasarımlar ileriye yönelik planlar arasındadır.

**monpar** algoritmasının performans analizi için, farklı sorgu karakteristiklerine sahip sorgu setleri, **monpar** ve diğer klasik organizasyonlar (LRU, Öncelikli gibi) üzerinde çalıştırılmıştır. Performans başarımı, ara bellekte yakalama ve ıskalama sayılarına dayandırılmıştır. Bölüm 5'de sunulan deney sonuçlarında, **monpar'**ın LRU, "Rasgele" ve

"Öncelikli" ara bellek organizasyonlarından genel olarak daha başarılı olduğu gösterilmiştir.

## 4.5 Sonuç

Bu bölümde, tezde gerçekleştirilen hareketli nesne sisteminin tasarımı anlatılmıştır. Hareketin, güncellemelerin ve belirsizliğin modellenmesi için kullanılan yöntemlerin açıklanmasından sonra sorgulama sisteminin genel karakteristiklerinden bahsedilmiş ve olasılıksal sorgulama kapsamında $OAS^N$ adı verilen olasılıksal sorgulama algoritmasının modeli anlatılmıştır. Daha sonra, sistemde gerçeklenen erişim yapılarının belirlenmesinde dikkat edilen kriterlerden bahsedilmiştir. Son olarak, sayfa içeriğine bağlı ara bellek organizasyonu üzerinde durulmuştur. Ara bellek organizasyonlarının temelini teşkil eden sayfa yerleştirme algoritması, klasik yaklaşımda sayfa kullanım sıklığı ve/veya en son kullanılan sayfa gibi istatistikleri esas alırken; erişim yapıları için tasarlanan ara bellek organizasyonları indis yapısının mimarisine, hatta indis düğümlerinin içeriğine bağlı olabilmektedir. Bu yaklaşım esas tutularak, gelişmekte olan gerek uzlamsal, gerek zaman-uzlamsal indis yapıları için ara bellek organizasyonları tasarlanabilir. Bu bölümde, *Spatial*PageReplacement algoritmasındaki seçim kriteri esas alınarak geliştirilen, **monpar** isimli ara bellek organizasyonunun tasarımı açıklanmıştır.

# 5. HAREKETLİ NESNE SİSTEMİNİN GERÇEKLENMESİ VE DENEYLER

Bu bölümde önceki bölümde açıklanan sistem tasarımına bağlı olarak geliştirilen hareketli sistemin gerçeklenmesi ve ana modüllerinin açıklanması üzerinde durulmuştur. Deneyler kısmında ise, gerçeklenen hareketli nesne indis yapılarının birbirleriyle karşılaştırılmaları ve ara bellek ile ilgili diğer simülasyon sonuçları grafiklerle aktarılmıştır.

Bu bölümde anlatılan erişim yapılarının sabit disk üzerinde gerçeklenmesi için, SaIL (***S***p***a***tial ***I***ndex ***L***ibrary) adı verilen, Hadjieleftheriou tarafından geliştirilen açık kaynak kodlu Java kütüphanesi kullanılmıştır. Gerçeklenen indis yapıları 3DR-tree, MVR-tree, Gezinge-2d R-tree ve MON-tree yapılarıdır.

SaIL, yapısındaki sabit saklama birimi ara yüzü, erişim yapıları ara yüzü, uzlamsal nesne tanımlama ve diğer yardımcı ara yüzler ile mevcut erişim yapılarının karşılaştırılması yanında yeni yapıların gerçeklenip test edilmesine olanak sağlamaktadır. Kullanıcının kolayca tanımlayabildiği sorgulama ortamları ve performans analizi için disk erişim sayısının belirlenmesi, kütüphanenin sağladığı olanaklar arasındadır (Hadjieleftheriou vd., 2005).

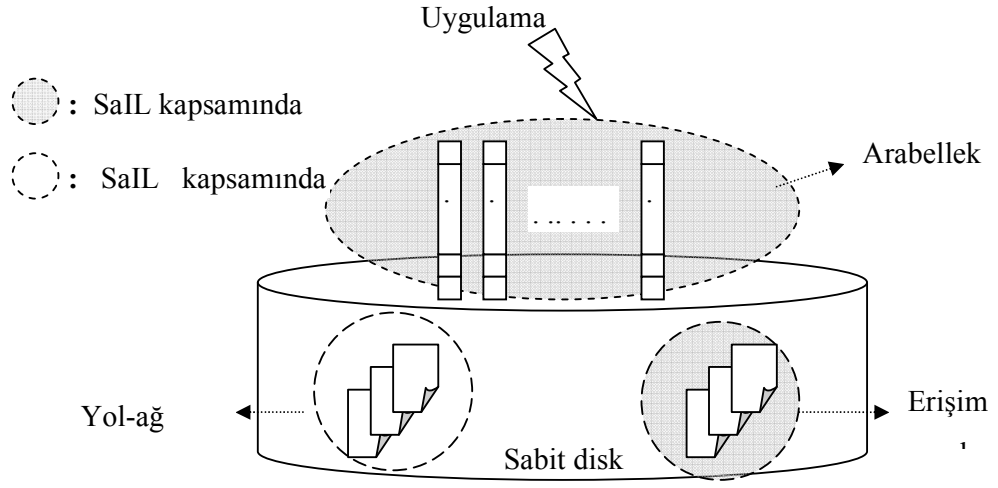
Bununla beraber, ağ-kısıtlamalı ortamlarda ağın sabit diskteki organizasyonu, ayrı bir çalışmanın konusudur. Gerek doğrudan, gerekse indis üzerinden sabit diskteki yol-ağına erişimin, indisin yüklenmesi esnasında ve sorgulamada performansa etkisi kaçınılmazdır. Çünkü birbirine yakın olan veya birbirine bağlantısı olan düğümlerin aynı disk sayfasında tutulabilmesi ortalama disk erişimini azaltmak için önemli bir kriterdir.

Buna göre, hedeflenen sistem mimarisi iki ana modülden oluşmaktadır:

- SaIL
- Yol-ağ organizasyonu

Şekil 5.1 bu tezdeki gerçeklenen sistemin ana modüllerini göstermektedir. **Uygulama programı**, istenen simülasyon ortamlarının oluşmasına olanak sağlayan programdır. Örneğin, seçilen yol-ağı üzerinde istenen trafik karakteristiklerinde trafiğin oluşturulup, erişim yapısına girilmesi ve gerekli performans analizi, uygulama programı seviyesindeki test programları ile gerçekleştirilir. **Ara bellek organizasyonları** ise, SaIL saklama birimi yöneticisi modülü kapsamında gerçekleştirilen ara bellek organizasyonlarıdır. Bu organizasyonlar *LRU, Öncelikli* gibi klasik tampon yapılar olabileceği gibi; Bölüm 4'de tasarımı açıklanan **monpar** gibi zaman-uzlamsal ortamlar için özelleştirilmiş tampon uygulamalar da olabilir. **Erişim yapıları** modülü ise SaIL'in sağladığı R-tree gibi klasik uzlamsal yapıları ve bu tezde gerçeklenen 3DR-tree, MVR-tree, gezinge-2d ve MON-tree erişim yapılarını içermektedir.

Son olarak, **yol-ağ organizasyonu** ise ağ-kısıtlamalı hareket senaryolarının temsil edilmesine olanak sağlayan disk organizasyonudur.




Şekil 5.1 Sabit diskte zaman-uzlamsal sistemin ana mimari yapısı

Gerek erişim yapıları gerekse ara bellek organizasyonları, SaIL kapsamında tanımlanmış birçok ara yüzü kullanmaktadır. Bu ara yüzlerden önemlileri sistemin ana iskeletini oluşturan SaIL Kütüphanesi kapsamında incelenecektir. Daha sonra, yol-ağ organizasyonun gerçeklenmesi ve SaIL ile olan entegrasyonu anlatılmaktadır.

## 5.1 SaIL (*Spa*tial *I*ndex *L*ibrary)

Hadjieleftheriou vd.'de (2005) temel prensipleri açıklanan SaIL, uzlamsal ve zaman-uzlamsal erişim yapılarının tasarlanması için ortak bir uygulama geliştirme ara yüzüdür. Farklı erişim yapılarının ortak bir platform üzerinde entegrasyonu ve karşılaştırılabilmesine olanak tanıması yönünden ayrı bir önemi vardır. Çünkü erişim yapılarının her biri farklı karakteristiklere sahiptir. Örneğin, uzay parçalamalı veya nesne parçalamalı olması veya ağaç yapısının dengeli olması/olmaması veya indiste tutulan nesnelerin nokta, çizgi veya bölge olması veya hedeflenen sorgulama tipinin farklı olması gibi erişim yapısını gerçekleştirmeyi doğrudan etkileyecek birçok farklı özellik vardır. Bunların ortak bir uygulama iskeleti altında toplamak SaIL'in esas hedefidir. SaIL, yeniden kullanılabilirlik (*reusability*) ve iyileştirilmiş kod kalitesini, iskelet yapısında kullanılan Gamma'daki (1995) tasarım kalıplarına borçludur. *Memento*, *proxy*, *composite*, *visitor*, *facade* isimli tasarım kalıpları SaIL'de kullanılan tasarım kalıplarından bazılarıdır. Bu kalıpların ne gibi olanaklar sağladığı Çizelge 5.1'de

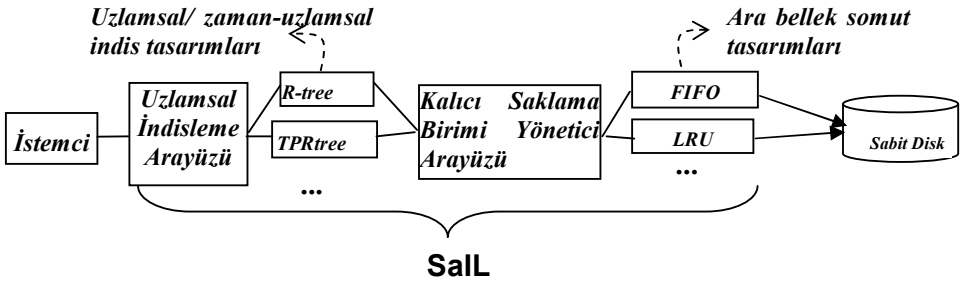
gösterilmektedir. Bununla beraber, bu kalıpların yazılım mühendisliğinde kullanımı, bu tezin kapsamının dışındadır; ancak bundan sonraki açıklamalarda, bu kalıpların SaIL'de kullanıldığı yerlerde sadece isimleri geçecektir.

Çizelge 5.1 SaIL'de kullanılan tasarım kalıplarından örnekler

| **Tasarım Kalıbı** | **Açıklama** |
|---|---|
| *Memento* | Kapsülleme(*encapsulation*) özelliğini kaybetmeden nesnenin içeriğinin dış dünyaya sunulmasına olanak sağlar. |
| *Proxy* | Yabancı bir nesneye erişmeye kontrol eden vekil nesne olarak kullanılır. |
| *Composite* | Gerek nesnelerin her birine gerekse nesnelerden oluşan kompozisyonlara ayni şekilde erişmeye olanak sağlar. |
| *Visitor* | Üzerinde işlem yapılacak olan nesnenin içsel işleyişini değiştirmeksizin yeni fonksiyonların tanınmasına olanak sağlar. |
| *Facade* | Ara yüzler üzerinde daha soyut bir erişime olanak sağlar. Bir ara yüz kompozisyonuna aynı ara yüzden erişime olanak sağlar. |

SaIL kütüphanesinin, gerek araştırma ("University of California at Riverside" üniversitesinde), gerekse ticari amaçlı kullanımları (ESRI şirketinde) mevcut olmakla beraber yakın zamanda GIS (***G**eographic **I**nformation **S**ystem*) araştırma grupları tarafından kullanılması beklenmektedir. Açık kaynak kodlu SaIL'in, C++ ve JAVA programlama dillerinde olmak üzere iki farklı gerçeklenmesi vardır. Bu tezdeki gerçekleme ve deneylerde, JAVA ortamındaki SAIL kütüphanesi kullanılmıştır.

SaIL, Şekil 5.2'de görülen erişim yapısı ara yüzü ve kalıcı saklama birimi ara yüzü ve bu ara yüzler üzerine somut gerçeklemelerden oluşmaktadır (Hadjieleftheriou vd., 2005).

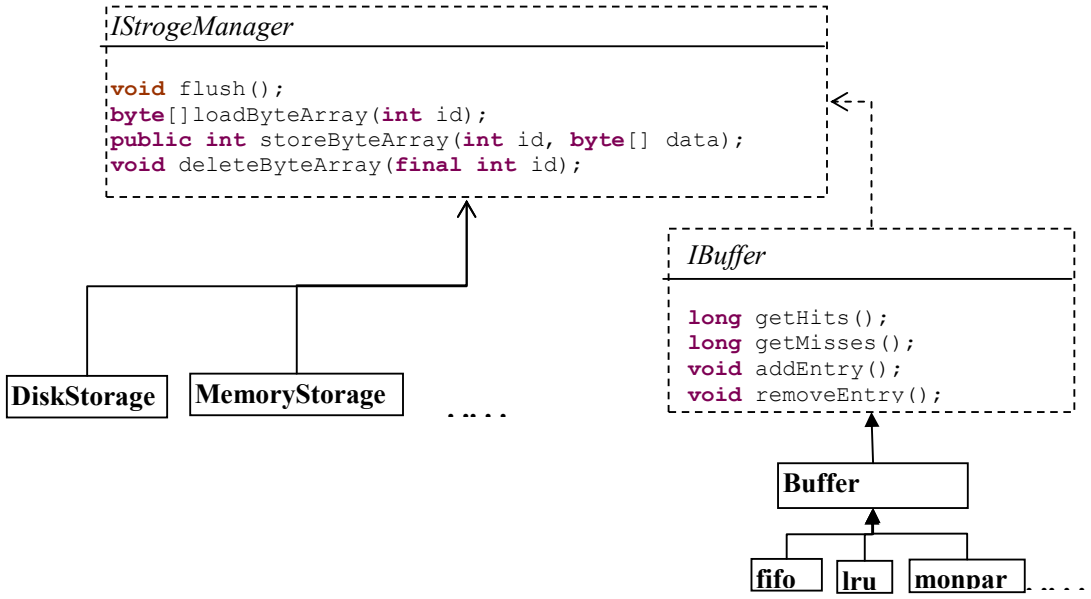



Şekil 5.2 SaIL kütüphanesinin ana mimari yapısı

### 5.1.1 Kalıcı Saklama Birimi Yönetici Ara yüzü

Bu modül erişim yapısı gerçeklemelerinin gerçek saklama biriminden ayrılması için

kapsülleme (*encapsulation*) vazifesi görmektedir. Kapsülleme ile esas amaç, saklama birimine çeşitli varlıkları kayıt etmek isteyen istemcinin, saklama biriminin gerçeklenmesi mekanizmalarından habersiz olmasını sağlamaktır. Diğer bir ifade ile kalıcı ortam bir sabit disk olabileceği gibi, bir haberleşme ağı veya bir veri tabanına ait ilişkisel tablo veya ana hafıza olabilir. İstemcinin bu ortamların her birine aynı şekilde erişmesi istemci seviyesindeki programlamada önemli bir esnekliktir. Şekil 5.3'de kalıcı saklama birimine ait ara yüzün UML *(**u**nified **m**odelling **l**anguage)* diyagramı gösterilmiştir.




Şekil 5.3 Kalıcı Saklama Birimi ara yüzü UML diyagramı

*IStorageManager* ara yüzünün şu an için iki somut gerçeklemesi vardır. Bu tezdeki kalıcı ortam saklama birimi, sabit disk (*DiskStorage*) olarak belirlenmiştir. *IStorageManager* ara yüzündeki gerçeklenmesi gereken fonksiyonlardan **loadByteArray(int id),** *id* ile belirlenen disk sayfasının ana hafızaya alınmasına; **storeByteArray(int id, byte[] data)** ise ana hafızada *byte* dizisi olarak paketlenen bilginin *id* ile belirlenen disk sayfasına yazılması işlemini yerine getirir. Gerek indis oluşturmada gerekse disk üzerinde diğer organizasyonlarda (yol ağı gibi) bu iki fonksiyon, *byte* dizisi gibi başarılı bir soyutlama ile her türlü bilginin sabit diske yazılıp, okunmasına olanak sağlar.

*IBuffer* ara yüzü, Şekil 5.3'deki diyagramda görüldüğü gibi bir ara bellek için gerekli olan **addEntry()** ve **removeEntry()** gibi karakteristik fonksiyonları içerir. Bunların yanında, **getHits()** ve **getMisses()** gibi yardımcı fonksiyonlar; ara bellekte yakalanan disk sayfa sayısı (*hits*) ve ara bellekten olmayan ve diskten erişilen disk sayfa sayısı (*misses*) gibi istatistikleri hesaplayarak

ara bellek performansının belirlenmesine yardımcı olur.

Ara bellek somut gerçeklenmeleri de erişim performansı yönünden önemlidir; çünkü disk ortamındaki bir indise erişim, kullanıcı seviyesinde özellikleri belirlenebilen bir ara bellek üzerinden sağlanır. SaIL kütüphanesinin kalıcı saklama birimi modülü kapsamında, LRU (***L***east ***R***ecently ***U***sed) ve "Rasgele" (*random*) gibi bazı genel ara bellek organizasyonları vardır. Fakat bu organizasyonlar R-tree ağaç yapısındaki indisler için uygun değildir ve daha özel organizasyonlara ihtiyaç duyulmaktadır. Bu noktadan hareketle, Bölüm 4'de açıklanan hareketli nesne indis yapısı için tasarlanan ara bellek organizasyonu, **monpar** gerçeklenmiştir.

Ek 2, sabit disk üzerinde R-tree indisinin oluşturulması ve daha sonra *fifo* ara bellek üzerinden indise erişim sağlayan bir Java program kodu ve ilgili açıklamaları içermektedir.

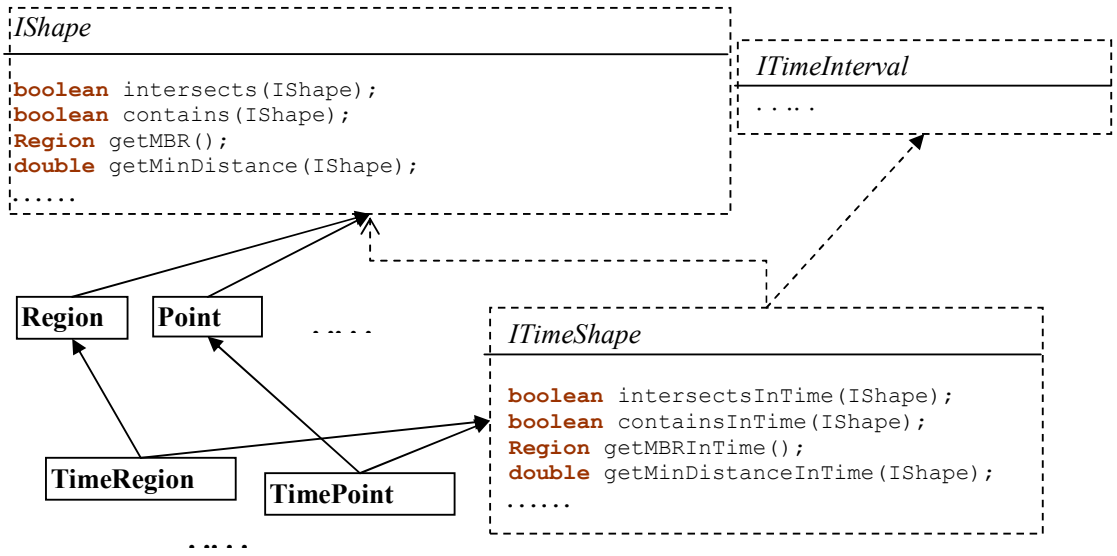
### 5.1.2 Uzlamsal İndisleme Ara yüzü

Bu arayüz, zaman-uzlamsal şekillerin, erişim yapısı temel birimlerinin ve erişim yapısında kullanıcı isteğine uygun aramaların gerçeklenmesinde gerekli olan fonksiyonları tanımlamaktadır.

#### *5.1.2.1 Uzlamsal ve Zaman-uzlamsal Şekillerin Gerçeklenmesi: IShape, ITimeShape*

Uzlamsal erişim yapısında birçok farklı, karmaşık nesne indislenebilir. Bu nesneleri kullanan ara yüzlerin mümkün olduğunca genel olması için, genel şekil özelliklerini ve fonksiyonları içeren soyutlamalar kullanılması gerekmektedir. Bu amaca hizmet eden *IShape* ara yüzü, bir uzlamsal nesneden beklenen genel fonksiyonları içermektedir. Şekil 5.4'de görüldüğü gibi, *IShape* ara yüzünü gerçeklemek zorunda olan, *Region*, *Point*, *Segment* gibi basit uzlamsal nesneler veya karmaşık uzlamsal şekiller, somut gerçeklenmeye örnektir.

Zaman-uzlamsal nesneler de, uzlamsal nesnelerin soyutlanmasına benzer şekilde, genel uzlamsal fonksiyonların zamana bağlı versiyonlarını içeren *ITimeShape* ara yüzünü gerçeklemek durumundadırlar. Bununla beraber uzlamsal özellik ve fonksiyonları, karşılık gelen somut uzlamsal nesneden kalıtımsal olarak alabilir. Örneğin, Şekil 5.4'de görüldüğü gibi *TimeRegion* somut gerçeklemesi, Region sınıfından türetilmiş ve *ITimeShape* ara yüzünü gerçeklemektedir.

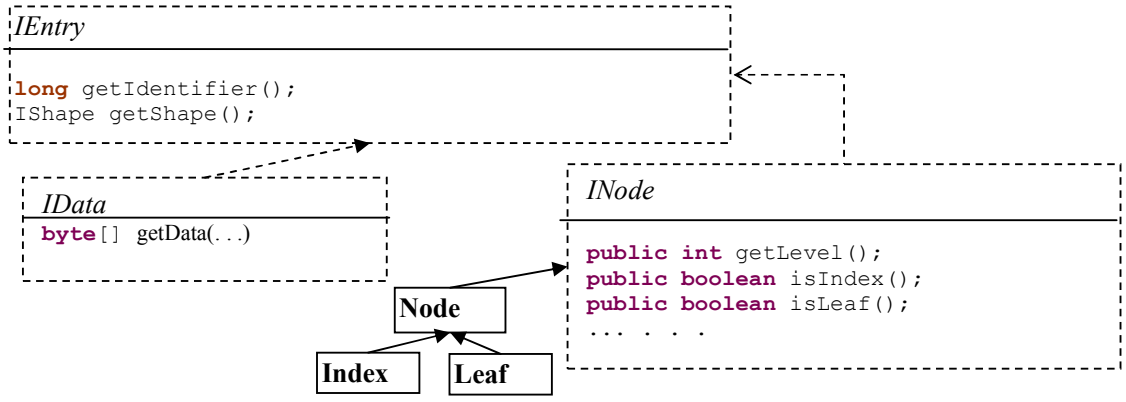



Şekil 5.4 Şekiller için kullanılan ara yüzler için UML diyagramı

*IShape* yazılım mühendisliğinde *Composite* tasarım kalıbına uygunluk göstermektedir.

### *5.1.2.2 Erişim Yapısının Temel Öğeleri: INode, IEntry*

Genel olarak, hiyerarşik bir erişim yapısı, düğümler (*Node*) ve düğümlerde tutulan veri girişlerinden (*data entries*) oluşur. Şekil 5.5'de görülen *IEntry*, veri girdisi için en temel arayüz olarak tanımlanmıştır. Bu arayüz nesnenin *id'si* ve şekli ile ilgilidir. *IEntry*'den türeyen *INode* ara yüzü, genel bir ağaç düğümü için gerekli olan ara yüzdür. Bu ara yüzde, düğümde tutulan nesne sayısı, düğümün ağaçtaki seviyesi gibi yapısal özellikler tanımlanır. Şekil 5.5'de, *INode*, ara yüzünü gerçekleyen somut sınıflara örnek olarak, R-tree indisine ait **Node** ve ondan türeyen **Index** ve **Leaf** düğümlerine ait somut sınıflar gösterilmektedir.

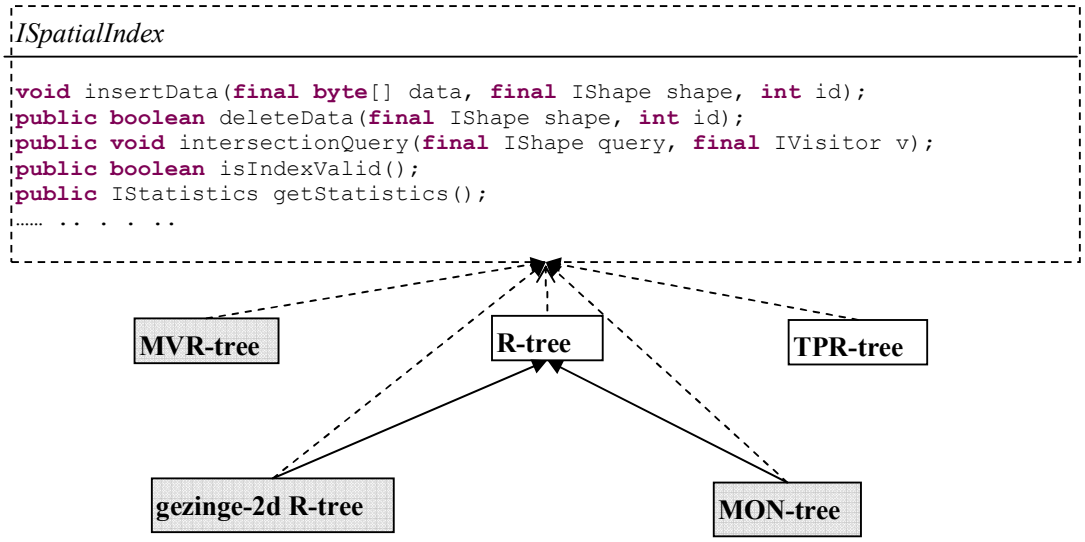



Şekil 5.5 Erişim yapısındaki veri üniteleri için UML diyagramı

*IData* ise, düğümde tutulan bilgi ile ilgili öte-bilgi (*meta-data*) veya bilginin saklayıcı ortamdaki adresini tutan işaretçiyi *byte* dizisi olarak tutan veri ünitesidir.

#### *5.1.2.3 Erişim Yapısı Ara yüzü: IspatialIndex*

SaIL kütüphanesi kapsamında tanımlanan yeni bir erişim yapısı *IspatialIndex* ara yüzünü gerçeklemek zorundadır. Yazılım mühendisliğinde *Facade* tasarım kalıbına uygunluk gösteren *IspatialIndex,* bir erişim yapısı için gerekli olan temel fonksiyonları içerir. Bütün fonksiyonlar *IShape* objesini argüman olarak alırlar. Örneğin, Şekil 5.6'daki şemada görülen, **insertData(...)** ve **deleteData(...)** fonksiyonları indise verinin girilmesinde ve silinmesinde kullanılan fonksiyonlardır.

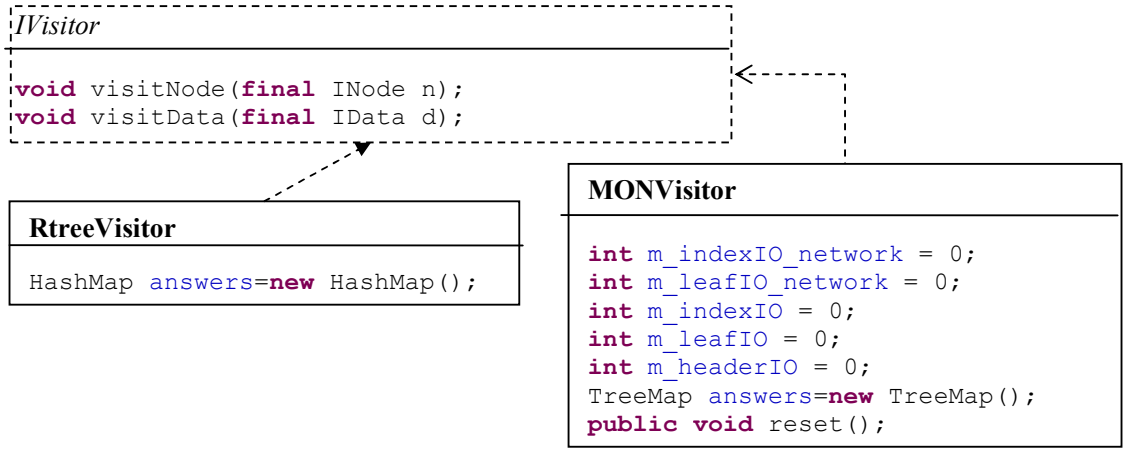



Şekil 5.6 Erişim yapısı birimi UML diyagramı

Şekil 5.6'de gri renkle ifade edilen somut sınıflar bu tez kapsamında gerçeklenen indis yapılarıdır. Gezinge-2d R-tree ve MON-tree, mimari olarak R-tree ağaçlarının bir organizasyonu olduğu için R-tree'den türetilmiştir. Diğer yandan MVR-tree, kendine özgü *ikiye bölme* ve *birleşme* algoritmaları yanı sıra çok-versiyonlu R-tree yapılandırmasıyla oluşmaktadır. Şekil 5.6'de görülmeyen 3DR-tree ise R-tree'ye 3 boyutlu veri girilmesiyle elde edileceği için R-tree'den farklı bir gerçekleme değildir. Son olarak Şekil 5.6'de görülen TPR-tree yakın gelecek tahmini hareketler üzerine geliştirilen bir indis yapısı olup bu tez konusunun dışındadır.

#### *5.1.2.4 Gelişmiş Düzeyde İsteğe Göre Uyarlama Yeteneği*

*Visitor* tasarım kalıbına uygun olan *IVisitor* ara yüzü sayesinde, indis düğümlerine erişildiği zaman, kullanıcı tarafından tanımlanan fonksiyonların çalıştırılmasına olanak sağlanır. Bu yapıya verilebilecek en kullanışlı örnek, sorgu işlenmesi sırasında disk erişim sayısının hesaplanmasıdır. Buna göre, Şekil 5.7'deki *IVisitor* ara yüzü, ziyaret ettiği düğüm veya indis düğümleri sayan fonksiyonları içeren bir ara yüzdür. Şekil 5.7'deki örnek somut gerçeklemeler gibi, bu arayüz üzerinde isteğe göre uyarlama ile herhangi bir erişim yapısının mimarisine göre disk erişim performansı hesaplanabilir.




Şekil 5.7 Erişim yapısına göre uyarlamalı *IVisitor* ara yüzü

Şekil 5.6'deki erişim yapısı ara yüzündeki **intersectionQuery(final IShape query, final IVisitor v)** fonksiyonu, üzerinde aralık sorgusu çalıştırılacak olan indisin yapısına uygun bir *Visitor* somut sınıfını argüman olarak almaktadır. Ek 3, R-tree indisi üzerinde disk erişim performansının hesaplanması ile ilgili örnek kod parçacığı ve gerekli açıklamaları içermektedir.

#### *5.1.2.5 Uzlamsal İndislerin Somut Gerçeklenmesi: R-tree, 3DR-tree, MVR-tree, gezinge-2d R-tree, MON-tree*

R-tree, düğümlerinde uzlamsal nesnelere ait **id** ve farklı şekilde nesnelerin MBR'larını saklamasıyla, *Region* tipinde nesneler üzerine bina edilen bir yapı sergiler. Böylece nesnelere; indis ikincil ise**, *id'***leri vasıtasıyla, birincil ise, **id** yerine kendisine doğrudan erişilebilir. Bu tezde kullanılan R-tree indisi ve diğer bütün yapılar, ikincil indis olarak yapılandırılmıştır.

3DR-tree, R-tree'ye, (2 uzlamsal+1 zamansal) olmak üzere 3 boyutlu nesne MBB'lerinin

eklenmesi ile oluşturulmuştur. Düğüm *ikiye bölme* ve *birleşme* algoritmaları R-tree indisi ile aynıdır.

Konum güncellemelerini çok-versiyon tekniği ile R-tree ağaç dizisine kaydeden MVR-tree ise, SaIL-C++ versiyonundan, SaIL-Java versiyonuna taşınmıştır (Heper, 2005).

Yol-ağı ve gezingeler için ayrı olmak üzere, iki adet 2-boyutlu R-tree yapılarından oluşan, Gezinge-2d R-tree, Bölüm 3'de anlatılan boyut dönüşümlerinden sonra elde edilen 2-boyutlu *(x,t)* veriyi eklemeyi esas almaktadır. Gerçeklenmesi, MON-tree'ye göre daha kolay olması daha az (fakat daha hacimli) ağaçtan oluşmasındandır.

Son olarak, MON-tree gerçeklenmesinde, De Almeida vd.'deki (2004) ekleme, arama algoritmaları esas alınmıştır. MON-tree gerçeklemesinde her iki seviyede düğüm parçalanma (*split*) ve birleşme (*merge*) algoritmaları, R-tree ile aynıdır. Ancak, MON-tree, bir grup R-tree'nin Bölüm 3'de anlatılan organizasyonu olup, yol ekleme ve hareket ekleme algoritmaları için yardımcı *hash* tablaları kullanılmıştır.

Yukarıdaki erişim yapılarının somut gerçeklenmeleri için, Şekil 5.6'de açıklanan arayüz kullanılmıştır.

## 5.2 Yol-ağ Organizasyonu

Tu tez çalışmasında kullanılan nesneler, "ağ kısıtlamalı" hareket senaryolarına göre hareket ettiklerine göre, yol ağının temsil edilmesi ve ağdaki komşuluk ilişkilerini kullanan algoritmaların (yön belirleme, en kısa yol bulma gibi) tasarlanması, sistemin önemli bir altyapısı olarak düşünülebilir. Çünkü geliştirdiğimiz sistemde, belirli bir dağılıma göre nesnelerin oluşturulmasında, trafiğin üretilmesinde, nesnelerin hareketlerinin modellenmesinde ve daha önemlisi dinamik özelliklerin indislenmesinde yol-ağ organizasyonunun etkisi oldukça fazladır.

Bu tez kapsamında, yol haritası bir ağ ile temsil edilmiştir. Bu ağ modeli Bölüm 4.1.1'de açıklanmıştır (Şekil 4.1). Ağ organizasyonunun sistem performansında etkin rol almasının en önemli nedeni, indis yapısı gibi ikincil hafızada tutulmasının gerekliliğidir. Ağ organizasyonunu, geleneksel bir ağ yapısı ile birincil hafızada tutmanın önemli bazı sakıncaları vardır. Bunlardan en önemlisi, çok sayıda yol ve kavşak içeren ağların birincil hafızada ayrılan bölgeye sığmamasıdır. Bu problem ile karşılaşılmadığı durumlarda dahi, bütün ağın her zaman birincil hafızada tutulması sistem kaynaklarının kullanım verimliliği için bir dezavantaj oluşturacaktır. Ağın birincil hafızada olmasının getirdiği diğer bir

dezavantaj, sistemin yeniden başlatılması durumunda ağın tekrar kurulması için harcanan işlemci zamanıdır. Bu da sistem ilk değer durumunu alıp kararlı bir hale gelmesi için harcanan zamanı uzatmaktadır. Son olarak; bütün yukarıdaki dezavantajların ötesinde gerçeklemede yaşanan bir problem vardır. Bu tez kapsamında sistem Java programlama ortamında gerçekleştirilmiş olup, bu ortamın sağladığı hesaba dayalı adresleme (*hashing*) tabloları ağdaki ana kavşak noktaları ve ana yolları birincil hafızada saklamak için verimli araçlardır. Fakat bu tablolar, *deterministik* olmayan çalışma yapılarının bir gereği olarak, aynı ağı farklı zamanlarda farklı temsil ettikleri için sistemin doğru çalışmasında önemli bir engel teşkil etmektedirler. Örneğin, tasarlanan bir hareketli nesne indis yapısında, nesnenin belirli bir zamandaki dinamik özellikleri ve *id'*si ve üzerinde hareket ettiği yolun *id'*sini saklaması gerekmektedir. Buradan anlaşılıyor ki; ağ'daki her bir yol ve kavşak için belirlenen yol-id ve kavşak-id gibi birincil özelliklerin, indiste yer aldığı değerlerinin korunması, sistemin yeniden başlatılmasında farklı değerler almaması gerekmektedir. İşte bunu sağlamanın en güvenli yolu da, ağ organizasyonunu ikincil hafıza birimi olan diskte saklamaktır.

Ağ organizasyonun ikincil hafızaya taşınmasına kanaat getirdikten sonra karşımıza çıkan problem, verimli bir disk tabanlı organizasyonun tasarlanması ve gerçeklenmesidir. Bu başarımın ana kriteri bir kavşak ve/veya yola ve bunların komşuları olan kavşaklara ve/veya yollara ait bilgiye erişimin, en az sayıda disk erişimi ile en kısa zamanda olmasıdır[*]. Bunun için izlenecek en önemli strateji birbirine yakın olan kavşakların aynı disk sayfasında bir araya getirilmesidir ki; bu yaklaşımla uzun süreçte ortalama disk erişim sayısı azaltılabilir. Çünkü bir kavşağa yapılan erişim, kısa bir zaman sonra bu kavşak etrafındaki yollara ve/veya kavşaklara erişimin olacağının habercisidir. Bu strateji, literatürdeki **kümeleme** (*clustering*) probleminin bir uygulamasıdır. Bu aşamada, kümelemenin uygulanmasında, iki farklı yaklaşım söz konusudur. Bunlardan birincisi, ağ-bağlantılı kümelenme modeli olup, kriter olarak nesnelerin birbirine olan yol bağlantıları esas alınmıştır. Diğer bir ifade ile aynı disk sayfasına yerleşecek olan kavşakların mümkün olduğunca bir yol ile birbirine bağlantılı olması esas tutulmuştur. Ağ-bağlantılı kümelenme modeline, Shekhar ve Liu (2003) tarafından geliştirilen CCAM *(Connectivity-Clustered Access Method)* modeli örnek olarak verilebilir.

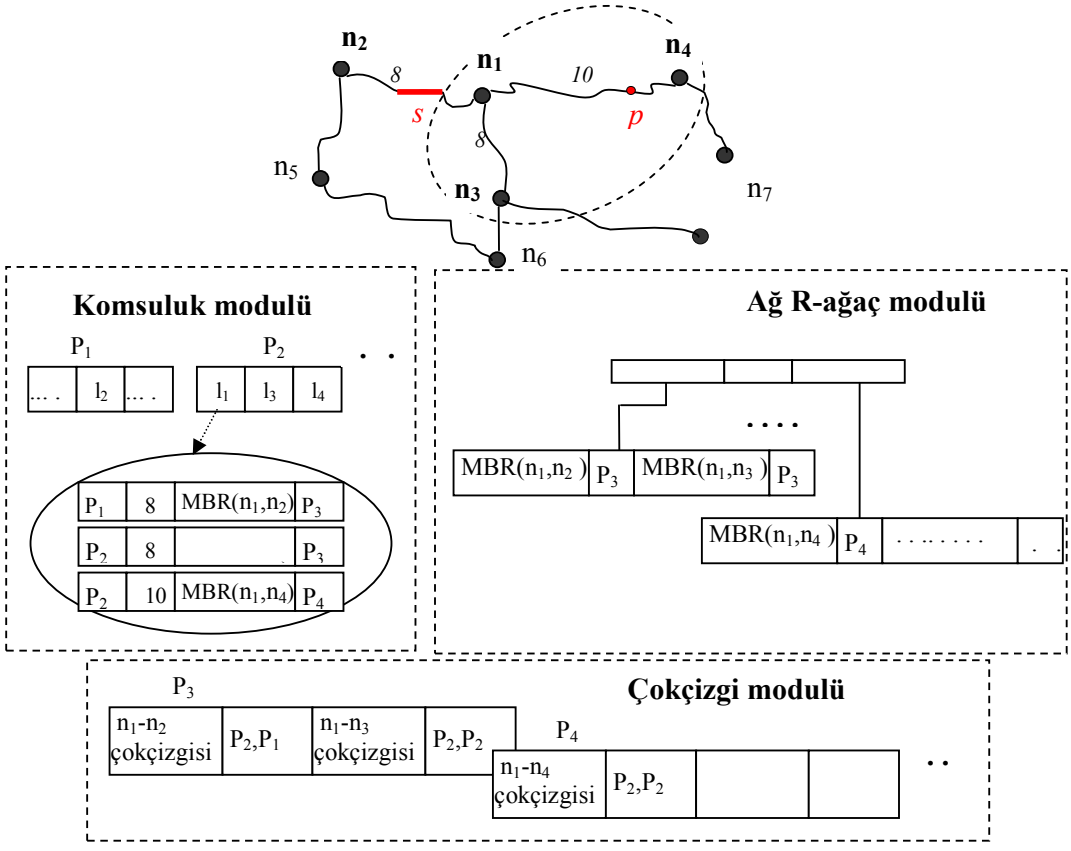
---

[*] Erişim (arama) performansı yanında, ekleme ve çıkarma performansı çok da önemli değildir; çünkü yol haritasının zamanla çok sık değişen bir yapı olmadığı varsayılmaktadır.

Diğer yaklaşımda ise; kümelenme, belirlenmiş bir benzerlik kriterine göre sıralanmış olan nesnelerin disk sayfalarına sırayla yerleştirilmesi ile gerçekleştirilebilir. Bu genel tanım, uzlamsal nesneler içinde geçerli olmakla beraber bu tip nesnelerin sıralanması tek boyuttaki sıralama gibi olmadığı için bazı dönüşümlere ihtiyaç duyulmaktadır. Bölüm 3'de, aralık doldurma eğrileri *(space filling curve)* ile çok boyutlu nesneleri bulundukları uzaydan tek boyutlu uzaya dönüştürüp sıralanabildiği açıklanmıştı. Doldurma eğrileri ile nesneleri sıralayıp disk sayfalarına sıralı yükleme yaklaşımına Papadias vd.'deki (2003) organizasyon modeli örnek olarak verilebilir.

### 5.2.1 Bu Tezde Gerçekleştirilen Yol-Ağ Organizasyonu

Bu bölümde incelenen ağ organizasyonu, Papadias vd.'da (2003) önerilen disk organizasyonudur. Bu organizasyon, komşuluk modülü *(adjacency component),* çokçizgi modülü *(polyline component)* ve ağ R-ağacı modülü *(network R-tree component)* olmak üzere 3 alt modülden oluşmaktadır. Şekil 5.8'de, örnek bir yol ağı için kavşakların ve ana yolların modüllere yerleşimi ve modüllerin birbirine olan bağlantıları gösterilmektedir.




Şekil 5.8 Ağ disk organizasyonu modülleri

**Komşuluk modülü**, *hilbert-sırasına* göre disk sayfalarına yerleştirilen kavşakları içermektedir. Bu modülün disk sayfalarında tutulan kayıtlar, kavşakların *id'*sini ve bu kavşağa ait olan komşuluk listesini tutmaktadır. Komşuluk listesinde ise, komşu kavşağın bulunduğu disk sayfası, bu komşuya olan yolun yükü (uzaklık veya başka bir değer), komşu kavşakla beraber oluşan MBR ve bu MBR'ın bulunduğu çokçizgi modülündeki disk sayfasının adres bilgileri yer almaktadır. Bu organizasyon sayesinde, bir kavşağın etrafındaki kavşaklara veya belirli bir rota boyunca ilerleme, başka modüllerde sorgu yapmaya ihtiyaç duymadan kolayca gerçekleştirilir.

**Çokçizgi modülü**, komşuluk modülünde tutulan komşu kavşakların birbiriyle olan bağlantılarının detayını temsil eden çokçizgilerin ve bu çokçizgilerin uç noktalarındaki kavşakların, bulundukları komşuluk modülündeki disk sayfalarını saklamaktadır. Örneğin, Şekil 5.8'de görüldüğü gibi, $n_1$,$n_2$,$n_3$ kavşakları, hilbert-sıraları yakın olduğu için, $P_2$ komşuluk disk sayfasına yerleşmişlerdir. $n_1$ kavşağına ait olan $l_1$ komşuluk listesi, ($n_2$, $n_3$, $n_4$) kavşaklarını içermektedir. Komşuluk listesinde ise, örneğin, $n_2$ ile olan komşuluk bilgisinde "$n_2$'nin bulunduğu komşuluk disk sayfası $P_1$" olduğu, "$n_1$, $n_2$ ana yol ile ilgili yükün 8" olduğu, "$n_1$, $n_2$ arasındaki ana yola ait olan MBR" bilgisi ve "$n_1$-$n_2$ MBR'ının tutulduğu çokçizgi disk sayfası $P_3$" olduğu bilgisi tutulmaktadır. $P_3$ çokçizgi sayfasına baktığımız zaman, $n_1$-$n_2$ çokçizgisi ile beraber, bu çokçizginin ucundaki $n_1$ ve $n_2$ kavşaklarının bulundukları komşuluk disk sayfaları, $P_2$ ve $P_1$, bilgisi yer almaktadır.

**Ağ R-ağacı modülü**, çokçizgi modülündeki bütün ana yolları (ki bunlar ağdaki bütün ana yollardır), R-ağacı ile indisleyen modüldür. Yol haritası üzerindeki uzlamsal sorgulamaya olanak sağlayan R-tree'nin yaprak düğümlerinde, her bir çokçizginin MBR'ı ve çokçizginin yer aldığı çokçizgi disk sayfasının adresi tutulmaktadır. Şekil 5.8'deki örnekte, bir R-ağacı yaprak düğümünde, $n_1$, $n_2$ arasındaki ana yolun MBR'ı ve detaylarının $P_3$ disk sayfasında tutulduğu görülmektedir.

Açıklanan bu disk organizasyonu üzerinde, ağ ile ilgili birçok uygulama verimli olarak gerçekleştirilebilir. Örneğin,

- 'p' noktasının veya 's' yol-parçacığının üzerinde bulunduğu ana yolun özelliklerinin bulunması (*anayol-bulma (point p), anayol-bulma (line s)* )
- '$p_1$' noktasından, '$p_2$' noktasına, minimum ağ uzaklığında yolculuğun hangi kavşaklar üzerinden gerçekleşeceği (*dijikstra-yolunu-bulma(point $p_1$,point $p_2$)* )

gibi sorgular bu organizasyon ile başarılı bir şekilde gerçeklenebilecek önemli

uygulamalardır. Bu tez kapsamında gerçeklenen sistemin bugünkü halinde en çok kullanılan fonksiyon *anayol-bulma (point p)* fonksiyonudur. Bununla beraber farklı simülasyon senaryolarında *dijikstra-yolunu-bulma(point* $p_1$*,point* $p_2$*)* veya diğer ağ ile ilgili fonksiyonlar yukarıda açıklanan disk organizasyonu üzerinde gerçeklenebilir. Örneğin, yol-ağı üzerinde rasgele iki nokta arasındaki en kısa yolda hareket eden nesnelerin üretildiğini düşünelim. Nesnelerin gelecek zaman rotalarının belli olduğu böyle bir senaryoda, nesnenin geleceğe ait konumları *dijikstra-yolunu-bulma* fonksiyonu ile bulunan rota üzerinde hesaplanabilir.

## 5.3 Deneyler

Hareketli nesnelerin hareketlerinin yol-ağı üzerinde oluşturulması ve gezingelerin (veya konumların) indise yerleştirilmesi için ayrık zamanda çalışan bir simülasyon ortamı geliştirilmiştir. İndis yapılarının yüklenmesi ve bu yapıları üzerinde sorgu setlerinin çalıştırılması yanı sıra, sorgu sonuçlarının doğruluğunu kontrol etmek amacıyla nesne hareketleri grafik arayüz ile izlenebilmektedir.

Yapılan deneyler birbirinden bağımsız 3 grup altında toplanmaktadır:

- Trafik karakteristiklerinin karşılaştırılması
- Erişim yapılarının gerçeklenmelerinden sonra, bu yapıların yer verimliliği, sorgu işleme performansı gibi temel kriterler üzerinden birbiriyle karşılaştırılması,
- MON-tree hareketli nesne indisine yönelik geliştirilen **monpar** isimli özelleştirilmiş ara belleğin sorgu performansına olan etkisi

### 5.3.1 Trafik Üreteci

Yol-ağı üzerinde trafik üreteci için geliştirilen simülasyonun her adımı bir ayrık zaman dilimine karşılık gelmek üzere, Şekil 3.11'de görülen zaman şeridine asılmış sinema levhaları simülasyonuna benzetilmiştir. Ancak üretilen nesneler sadece nokta olup, iki levha arasında konumsal bağlantıları yol-ağındaki yollardır. Buna göre, herhangi **t** simülasyon adımında, ortamdaki canlı nesne sayısını sabit tutmak amacıyla **t** zaman diliminde ölen nesneler kadar yeni nesne yol-ağı üzerinde oluşturulmuştur. Her nesne için yaşama süresi [1, **sl**) arasında düzenli dağılıma göre belirlenmiştir. Çizelge 5.2, trafik üretirken kullanılan parametreleri, anlamlarını ve değer aralıklarını göstermektedir.

Çizelge 5.2 Yol-ağı üzerinde trafik üretiminde kullanılan parametreler

| ***Parametre*** | ***Anlamı*** | ***Değerler, Değer Aralığı, Özellik*** |
|---|---|---|
| **ds** *(dataset size)* | Hareketli nesne sayısı | 100, 200, 400,800,1600 |
| **sl** *(simulation length)* | Simülasyon adım sayısı | 100,200,300,400,500,750,1000 |
| **mui** *(max. update interval)* | T-zamansal periyoda bağlı güncellemede olabilecek en büyük güncelleme periyodu. | 10 |
| **T** | T-zamansal periyoda bağlı güncellemede herhangi bir nesne için tayin edilen güncelleme periyodu | [1, ***mui***] |
| **sd** *(speed distribution)* | Nesne hızı rastlantısal dağılımı | Düzenli (*uniform*) |
| **id** *(initial distribution)* | Üretilen nesnenin yol-ağındaki ilk konumu | Düzenli (*uniform*), Normal (*gaussian*), Zipf |
| $\mathbf{v_{min}}$,$\mathbf{v_{max}}$ | Herhangi bir nesneye ait en küçük ve en büyük hız değerleri | $V_{min} \leq v_{min} \leq v_{max} \leq V_{max}$ olmak üzere düzenli dağılıma uygun olarak belirlenir. |
| $\mathbf{V_{min}}$,$\mathbf{V_{max}}$ | Global en büyük ve en küçük hız değerleri | $V_{min}$= 0.1, 2.5 (km/*sim_adım*[*]) $V_{max}$= 2.5, 5.0 (km/*sim_adım*) |
| **M** *(map)* | Kullanılan yol-ağ haritası | chicago yol haritası, RN5, RN11, RN20,RN25,RN50,RN100[**] |

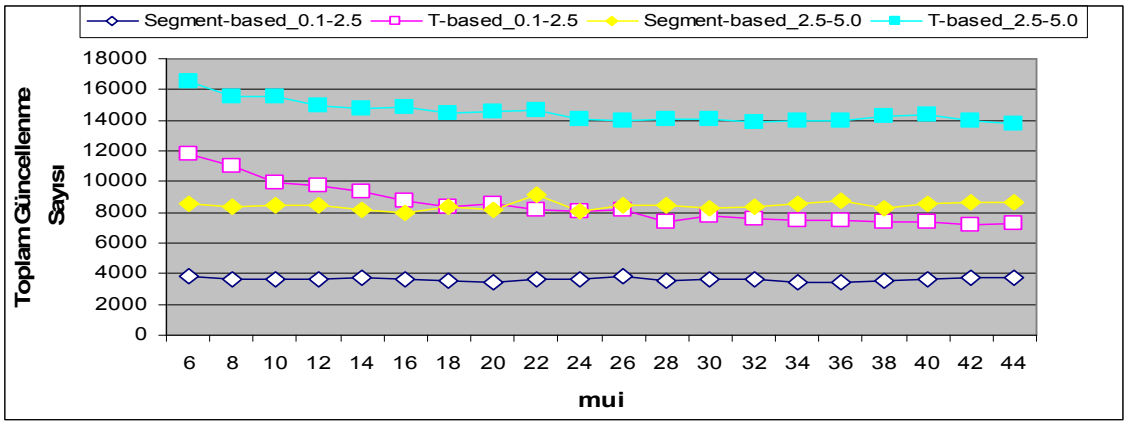
Nesne hızları, her güncellemede belirlenen [$v_{min}$,$v_{max}$] değer aralığında dağılımı seçilen **sd'**ye bağlı olarak belirlenmiştir. Nesne, iki güncellenme arasında [$v_{min}$,$v_{max}$] değer aralığında belirlenen hızla sabit gitmektedir. Bu durumda herhangi bir belirsizlik olmayacaktır. (Bu tez çalışmasında belirsizlik model olarak tanımlanmış, modelin gerçeklenmesi ve belirsizliğe dayalı ölçümler ileriye yönelik çalışmalara dahil edilmiştir.)

Bölüm 4.1.3'deki güncelleme modellerinde açıklandığı üzere 2 farklı trafik üretilmiştir: *Segmente dayalı güncelleme* ve *T-zamansal periyoda bağlı güncelleme*. Şekil 5.9, her iki trafik üretecinde oluşan güncelleme sayılarının, sistemin bir hareketlilik ölçüsü olan **mui** değerine göre ve [$V_{min}$,$V_{max}$] hız değer aralığına göre değişimini karşılaştırmaktadır. Örneğin, **segment-dayalı_0.1-2.5**, *segmente dayalı güncelleme* trafiği olup, nesnelerin [$v_{min}$,$v_{max}$] hız değerleri, global [$V_{min}$,$V_{max}$] = [0.1, 2.5] değer aralığında düzgün dağılıma uygun olarak belirlendiğini ifade etmektedir. Buna göre, Şekil 5.9'de görülen *segmente dayalı güncelleme* sadece bulunduğu hareket ortamına bağlı olduğu için, **mui'**nin değişiminden bağımsızdır. Bununla beraber, nesne hızlarının ikiye katlanmasıyla güncelleme sayılarında genel olarak ikiye katlanma gözlenmiştir. Bu beklenen bir sonuçtur çünkü nesneler, [0,**sl**] arasında düzgün

[*] *sim_adım*, 1 simülasyon adımı olup, yol uzunluklarının **km** cinsinden olduğu kabul edilmiştir.

[**] RN*K*, *(raster network)*, her uzlamsal boyutta *K* adet yoldan oluşan taramalı yapıda yol-ağıdır.

dağılımla belirlenen ömürleri süresince daha hızlı hareketle daha çok segment üzerinden geçmektedirler.




Şekil 5.9 Trafik üreteçlerinin karşılaştırılması

*T-zamansal periyoda dayalı güncellemeyle* oluşan trafiğin ise, artan **mui** değeriyle güncelleme sayısının azaldığı gözlenmiştir. Bununla beraber, bu azalma belirli bir **mui** değerinden sonra yerini sabit sayıda güncellemeye bırakmaktadır. Bunun nedeni, T güncelleme periyodundan önce yol-ağında ana yol sonlarında gerekli olan ve Bölüm 4.1.3'de bahsedilen "ek güncellemelerin" baskın olmasıdır. Hız dağılım aralığının iki katına çıkmasıyla, güncelleme sayıları, diğer trafik tipinde olduğu gibi, yaklaşık 2 katına çıkmaktadır. Bu artış, yüksek **mui** değerlerinde daha belirgin iken düşük **mui** değerlerinde, **mui**'ye olan bağımlılık daha fazla olduğu için, daha düşüktür. Örneğin, **mui**=6 olmak üzere $[V_{min},V_{max}] = [0.1, 2.5]$ için güncelleme sayısı 12.000 iken; $[V_{min},V_{max}] = [2.5, 5.0]$ için 16.500 olmaktadır.

### 5.3.2 Karşılaştırmalı deneyler

Hatırlanacağı gibi bu tezde ele alınan esas problem yol-haritası üzerindeki hareketli nesnelerin indislenmesidir. Sorgulama ortamı olarak da, uzlamsal ve zamansal aralık sorgulama esas alınmıştır. Bu amaçla indis yapılarının gerçeklenip birbiriyle karşılaştırılmasının ileriye yönelik önemli bir adım olacağı düşünülmüştür. Karşılaştırılan indis yapılarının (*3DR-tree, MVR-tree, gezinge-2dR-tree, MON-tree*) seçilmesinde şu kriterlere dikkat edilmiştir:

- Bütün indis yapıları nokta hareketleri ile oluşan gezingeleri indisleyebilmektedir.
- 3DR-tree indisi, zaman-uzlamsal indislemede bir başlangıç noktası olarak düşünülebilir. Çünkü zamansal boyutun, uzlamsal boyutlar ile doğrudan entegrasyonu

oldukça basit (belki tecrübesiz) bir çözümdür. O zaman yapılan iyileştirmeler, 3DR-tree ile karşılaştırılabilir.

- MVR-tree farklı zaman-uzlamsal modelleri destekleyebilen başarılı bir çözümdür. Bu genel çözümün, ağ-kısıtlamalı senaryolar için geliştirilen "gezinge-2d R-tree" ve "MON-tree" gibi yapılar ile yer verimliliği ve erişim performansı yönünden karşılaştırılması yerinde olacaktır.
- Gezinge-2d R-tree'nin tasarımında, yüksek performans bir yapı olması yerine, mevcut veri tabanları ile kullanılabilirliği esas alınmıştır. Bu noktadan hareketle, gezinge-2d R-tree'nin hangi koşullar atında kullanılmasının uygun olabileceğinin belirlenmesi gerekmektedir.
- MON-tree, daha önceki bölümde bahsedildiği gibi, ağ-kısıtlamalı modeller için başarılı bir indis yapısıdır. Bu başarımın hangi koşullarda sağlandığı, örneğin farklı kriterlerdeki sorgulara karşı başarımın nasıl etkilendiği, incelenmeye değer olacaktır.

Sistemin performans başarım ölçütü olarak, aşağıdaki özellikler değerlendirilmiştir.

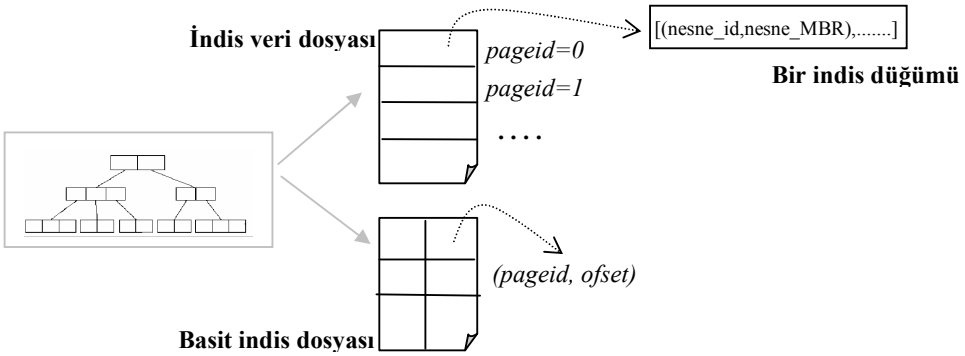
- erişim yapısı ilk oluşturulurken meydana gelen disk erişim sayısı.
- Erişim yapısının diskte kapladığı alan.
- Erişim yapısının belirli özellikleri taşıyan aralık tipindeki sorgu setlerini işlemesinin doğruluğu ve toplam disk erişim sayısı.

Seçilen erişim yapılarının gerçeklenmesi ile mevcut karakteristikleri doğrulanmış olmakla beraber, birbirleriyle karşılaştırmalarında da bazı sonuçlara varılmıştır. Karşılaştırmalar, aynı nesne hareket yığınlarının aynı disk ortamı ve aynı ara bellek sistemi üzerinde farklı erişim yapıları ile indislenmesinden sonra bu yapıların performanslarının karşılaştırılması esasına dayanmaktadır. Buna göre, indis ve ara bellek ortak karakteristiklerini belirleyen parametreler, anlamları ve değer aralıkları Çizelge 5.3'deki gibidir.

Çizelge 5.3 Erişim yapıları ve ara bellek ile ilgili parametreler

| ***Parametre*** | ***Anlamı*** | ***Değerler, Değer Aralığı, Özellik*** |
|---|---|---|
| **ps** *(page size)* | Disk sayfasının büyüklüğü | 4096 B |
| **c** *(leaf&index capacity)* | Yaprak ve indis düğüm kapasitesi | 10 |
| **f** *(fill factor)* | Yaprak ve indis düğümlerinde bölünme işleminde oluşan bir düğümde olması gereken minimum eleman sayısı, **c*f** olarak belirlenir. | 0.7 |
| **bs** *(buffer size)* | Ara bellek kapasitesi | 10 disk sayfası kapasiteli |
| **bt** *(buffer type)* | Ara bellek tipi | "Rasgele" |
| **tt** *(trafic type)* | Trafik tipi | segment_0.1-2.5, segment_2.5-5, T_0.1-2.5, T_2.5-5 |

Bir indis yapısı diskte, biri "indis veri" dosyası (örneğin *montree.dat*), diğeri bu dosya üzerinde "basit indis" *(simple index)* dosyası (örneğin *montree.idx*) olmak üzere iki adet dosyadan oluşmaktadır. Her iki dosya rasgele erişim dosyası *(random access file)* olup*;* indis veri dosyası, indisteki düğümlerin içeriklerini saklamaktadır, "basit indis" dosyası ise "indis veri" dosyasındaki disk sayfalarının *id'*leri üzerine inşa edilen ikincil indis (*secondary index*) olarak düşünülebilir. Gerçeklemelerde 1 disk sayfası, 1 indis düğümüne karşılık gelmektedir. Bir indis düğümü ise içeriğinde (**nesne_id, nesne_MBR**) nesne ikililerini saklamaktadır. Buna göre örnek bir indisin diskteki organizasyonu Şekil 5.10'deki gibidir.




Şekil 5.10 Bir indisin diskte iki dosya ile temsil edilmesi

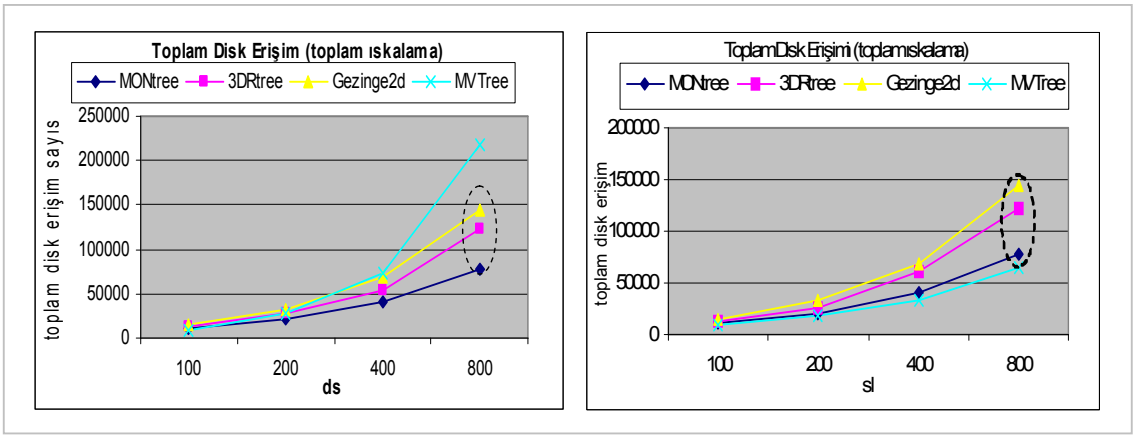
Benzer şekilde, yol-ağına ait Şekil 5.8'deki organizasyondaki her modül için, içerikleri o modülün gerektirdiği bilgileri içeren Şekil 5.10'daki disk dosyaları oluşmaktadır. Buna göre, yol-ağına ait sabit disk üzerinde 3 farklı veri dosyası *(network.dat, adjacency.dat ve polyline.dat)* ve bu dosyalara ait basit indis dosyaları *(network.idx, adjacency.idx ve polyline.idx)* vardır. Buna göre Çizelge 5.4 kullanılan yol-ağlarından bazılarına ait, diskteki "indis veri" dosyalarının içerdiği disk sayfa sayısını göstermektedir.

Çizelge 5.4 Yol-ağlarının diskteki yerleşimlerinde kapladıkları alan (disk sayfa sayısı)

| **M** *(map)* | *Komşuluk (adjacency) modülü* | *Çokçizgi (polyline) modulü* | *R-ağacı modulü* |
|---|---|---|---|
| RN5 | 1 | 1 | 5 |
| RN20 | 21 | 9 | 146 |
| RN25 | 32 | 14 | 223 |
| RN50 | 134 | 58 | 938 |
| Chicago | 40 | 52 | 283 |

### *5.3.2.1 Trafiğin indislere yüklenmesinde ölçülen değerler*

Bu bölümde, en fazla **10** disk sayfası saklayabilen "**rasgele"** tipinde ara bellek üzerinden 4 farklı indis yapısı oluşturulmuştur. Yol-ağı olarak **chicago** yol haritası kullanılmıştır. Şekil 5.11, indislerin oluşturulmasında yapılan toplam disk erişim sayısının **ds** ve **sl** parametrelerine göre değişimini göstermektedir. Grafiklerde MVR-tree dışındaki indis yapılarında, diske yapılan erişim değerleri indis düğümlerine ve yol-ağı disk sayfalarına olan erişim sayılarının toplamıdır. Çünkü sadece MVR-tree, yol-ağını kullanmayarak konum bilgilerini indisleyen genel bir indis yapısıdır.



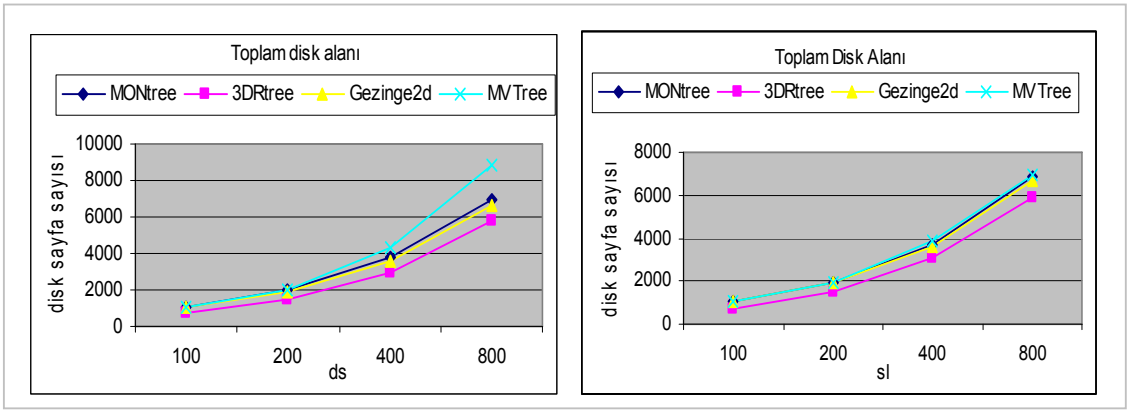

a) b)

Şekil 5.11 a) **sl**=100 için **ds** değeri artarken ve b) **ds**=100 için **sl** değeri artarken, indis ilk oluşumunda toplam disk erişim sayısının değişim grafiği (**M** = "chicago", **tt**= segment_0.1-2.5)

Şekil 5.11'deki grafiklerde ortak özellik, indis oluşumu için gerekli disk erişim sayısının **ds** ve **sl** değerlerine üstel bir fonksiyon ile bağlı olmasıdır. Bu bağımlılığın doğrusal olmamasının nedeni bütün yapıların R-tree tabanlı olup, yeni eklemeler ile ağacın seçicilik gücünü yükseltmek için yeni düzenlemelerin yapılmasının gerekliliğidir. Bu düzenlemeler eski kayıtların yerlerini değiştirerek, yeni bölünmeler/birleşmeler ile ağacın mimarisini değiştirmektedir. Bununla beraber, bu artışın **ds** ve **sl** değerlerine bağlılığının, MVR-tree dışındaki yapılarda benzer olduğunu görmekteyiz. Ancak MVR-tree'nin **ds** ile olan artışı diğer yapılar arasında en büyük iken, **sl** ile olan artışı diğer yapılar arasında en az olduğu görülmektedir. Bu netice, MVR-tree'nin diğer yapılardan farklı bir anlayışla *(çok-versiyon tekniği ile)* yapılanmasının bir sonucudur. Bunun nedeni açıktır; çünkü MVR-tree, versiyon bölünmesi tekniği ile simülasyon adımları ilerledikçe yeniden düzenlemelere fazla maruz kalmadan genişlemektedir. Diğer yandan aynı simülasyon süresi için, **ds** arttıkça, bu zaman aralığına karşılık gelen R-tree'de bölünmeler ve yeniden düzenlemelerin sayısı artmaktadır.

Deneylerde, **ds**=100, **sl**=800 için MVR-tree'de toplam düğüm bölünme sayısı 722, yeniden düzenleme sayısı 30200 iken; **ds**=800, **sl**=100 için toplam düğüm bölünme sayısı 1929, yeniden düzenleme sayısı 42265 olduğunun gözlenmesi bu sonucu doğrulamaktadır.

Son olarak Şekil 5.11, MVR-tree'yi göz önüne almazsak, **ds** ve **sl** değişimine en az duyarlı olan yapının MON-tree olduğunu göstermektedir. Gerek 3DR-tree, gerekse gezinge2d R-tree nesne trafiğinin hepsini tek bir ağaç yapısında tutmaktadır. Bu yaklaşım, disk yer verimliliğinde bir başarım sağlayacaktır (Şekil 5.12) fakat diğer yandan, artan **ds** veya **sl** parametreleri ile daha önce açıklanan eski kayıtların düzenlenmesi yüzünden gerekli olan bölünme/birleşmelerin sayısı artacaktır. Diğer yandan, MON-tree erişim yapısı bütün nesne trafiğini, 2 seviyeli organizasyon içinde birçok R-tree'ye düzenli olarak dağıtmasının bir sonucu olarak, bölünme ve yeniden düzenleme işlemlerine uzun süreçte daha az maruz kalmaktadır. Örneğin, **ds**=100, **sl**=800 için oluşan trafik, MON-tree'de 2.seviyedeki 1277 adet R-tree'ye dağılmıştır. Bu dağılım nesnelerin yol-ağındaki ilk dağılımlarına bağlıdır.




a) b)

Şekil 5.12 a) **sl**=100 için **ds** değeri artarken ve b) **ds**=100 için **sl** değeri artarken indisin diskte kapladığı toplam alanın değişimi (**M** = "chicago", **tt**= segment_0.1-2.5)

Şekil 5.12 ise, indisin disk üzerinde kapladığı toplam alanın **ds** ve **sl** parametrelerine göre değişimini göstermektedir. Disk alanının ölçülmesinde indislere ait toplam düğüm sayısı esas alınmıştır. Grafiklerde, gezinge 2dR-tree'ye ait değerler 2 boyutlu R-tree indis alanı ve yol-ağ R-tree ağacı indis alanının toplamı olarak alınmıştır. 3DR-tree'nin, beklenildiği gibi, yer verimliliği en iyi olan yapı olduğu görülmüştür. Bununla beraber, disk alanının **ds** ve **sl** ile beraber artması, Şekil 5.11'deki grafiklere benzer olup benzer sonuçlara varılabilir. Bununla beraber, MVR-tree'ye ait disk alanı her iki grafikte de en büyük artışı göstermiştir. Sonuç olarak, uzun simülasyonlarda MVR-tree'de çok fazla bölünme ve yeniden düzenleme olmasa

bile (Şekil 5.11b), yeni oluşan düğümler yüzünden disk alanı artışı diğer yapılara göre daha fazla olmaktadır.

#### *5.3.2.2 Sorgulama performanslarının karşılaştırılması*

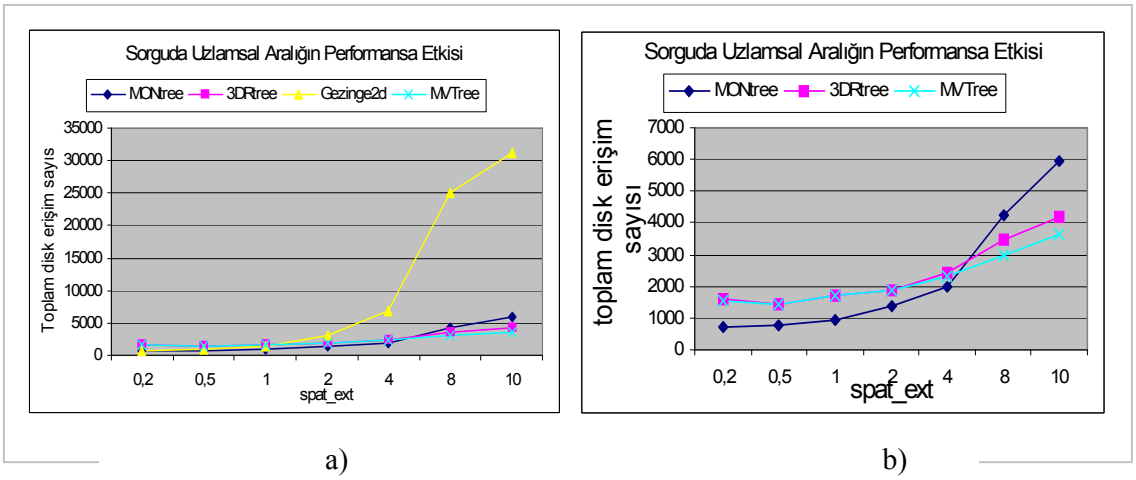
Bu bölümde, farklı sorgulama senaryoları kapsamında indis yapılarının performansları karşılaştırılacaktır. Deneylerde Çizelge 5.3'deki değerlere ek olarak sorgulama ile ilgili değerler Çizelge 5.5'deki gibidir.

Çizelge 5.5 Sorgulama ile ilgili parametreler

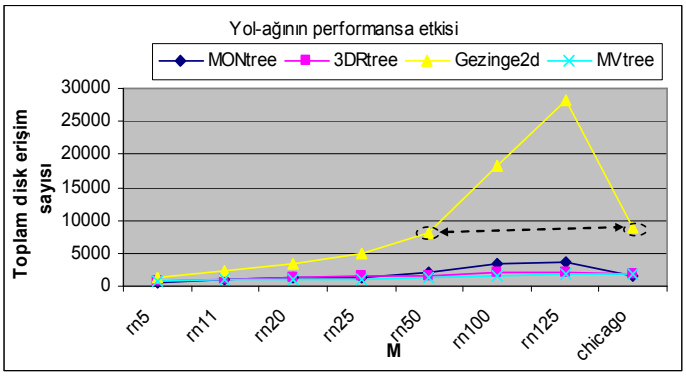
| ***Parametre*** | ***Anlamı*** | ***Değerler, Değer Aralığı, Özellik*** |
|---|---|---|
| **qs** *(queryset size)* | Sorgulama setindeki sorgu sayısı | 100 |
| **ds** *(dataset size)* | Hareketli nesne sayısı | 500 |
| **sl** *(simulation length)* | Simülasyon adım sayısı | 100 |
| **spat_ext** *(spatial extent)* | Sorguların sabitlenmiş uzlamsal genişliğinin toplam uzlamsal genişliğe olan yüzdesi | %0.2, %0.5, %1, %2, %4, %8, %10 |
| **temp_ext** *(temporal extent)* | Sorguların sabitlenmiş zamansal genişliğinin toplam simülasyon uzunluğuna olan yüzdesi | %0, %1, %2, %4, %8, %16, %32 |
| **M** *(map)* | Kullanılan yol-ağ haritası | chicago yol haritası, RN5, RN11, RN20,RN25,RN50,RN100 |
| **tt** *(trafic type)* | Trafik tipi | segment_0.1-2.5 |

Deneylerde, ilk olarak zaman-uzlamsal özellikteki 100 adet sorgudan oluşan setlerin 4 farklı indis yapısı üzerinde işlenmesi ile elde edilen sonuçların doğruluğu karşılaştırılmıştır. Bunun için her indis yapısından elde edilen sorgu sonuçlarının birbiri ile karşılaştırılması yanı sıra, grafik ara yüz ile sorgu bölgesinde kalan nesneler takip edilebilmektedir. Bununla beraber, sorgu setinin çalıştırılması sırasında oluşan disk erişim sayıları Ek 3'de açıklanan yöntem ile belirlenmektedir. Şekil 5.13a, ara bellekten ıskalanan disk sayfalarının (toplam disk erişim sayısının), zaman-uzlamsal sorguların artan uzlamsal aralık değerlerine göre değişimini göstermektedir. Artan uzlamsal aralık ile erişilen yol-ağı disk sayfa sayısı ve indis disk sayfa sayısının artması gerektiği açıktır. Çünkü artan uzlamsal aralık ile kesişen düğüm sayısı artmaktadır. Şekil 5.13'de, disk erişiminde artış miktarı en fazla olan indis yapısı gezinge 2dR-tree olduğu görülmektedir. Gezinge-2d R-tree'nin, zaman-uzlamsal sorguyu, sorgunun uzlamsal bölge kapsamına aldığı yolların sayısı kadar alt sorguya ayırıp, her bir sorguyu dönüştürülmüş uzayda ayrı çalıştıracağı daha önce açıklanmıştı. Bu yüzden, bu indis yapısının sorgulama performansı yol-ağının karmaşıklığı ile ilgilidir. Bu sonucun doğrulanması için,

yol-ağı tipinin sorgulama performansına olan etkisini gösteren Şekil 5.14'e bakılabilir. Şekil 5.14, gezinge 2dR-tree'nin, **M**, yol-ağı tipine olan bağımlılığını tam olarak gösteren bir grafiktir. Örneğin, Çizelge 5.4 tablosunun son iki satırında **RN50** ağının 134+58+938 = 1130 adet disk sayfasına dağıldığı, **chicago** ağının ise sadece 40+52+283 = 375 adet disk sayfasına yerleştiği görülmüştür. Bununla beraber her ikisi de aynı sorgu seti için, oldukça yakın performans değerleri vermektedir. Sonuç olarak, **RN50** düzenli yol-ağı olduğu için, kapladığı disk alanı olarak kendinden yaklaşık 4 kat daha küçük olan bir gerçek yol-ağının (**chicago**) performansına eşdeğer olmaktadır.




Şekil 5.13 M = "chicago" için toplam ıskalama sayısının uzlamsal sorgu genişliğine bağlı olan değişimi **a)**gezinge2dR-tree'nin sonuçlarını içeriyor **b)** gezinge 2dR-tree'nin sonuçlarını içermiyor

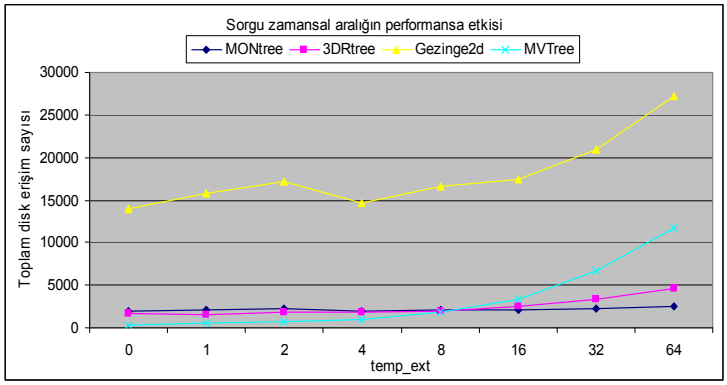



Şekil 5.14 *spat_ext*=10 olmak üzere, değişen yol-ağı tipleri için toplam ıskalama sayısı

Şekil 5.13b ise, diğer yapıların birbirine göre durumlarını daha iyi göstermek amacıyla, Şekil 5.13a'nın gezinge 2dR-tree'nin sonuçlarını içermeyen halidir. Bu grafikten anlaşılıyor ki;

MON-tree sorguda artan uzlamsal aralığa üstel olarak duyarlılık göstermektedir. Bunun nedeni MON-tree'nin yapısal mimarisidir. Diğer yapılarda, MVR-tree ve 3DR-tree, arama tek bir R-tree'de olduğu için uzlamsal genişliğe olan bağımlılıkları daha yumuşaktır. Fakat MON-tree'de artan uzlamsal aralık ile 1. seviyede yakalanan yollar artmakta, dolayısıyla bunlara ait trafiği tutan R-tree'ler üzerinde aramalar yeni disk erişimlerini gerektirmektedir. Diğer bir ifadeyle, sorgudaki uzlamsal aralığın bir miktar artması sonucunda bir yolun daha sorgu içine eklenmesi, (trafik bilgisi tutan) yeni bir R-tree'nin aranması kadar yük getirmektedir. Fakat bu yük, gezinge 2dR-tree'deki kadar olmamaktadır; çünkü gezinge 2dR-tree'den farklı olarak, artan uzlamsal aralığa eklenen yollara ait R-tree'ler, sadece o yola ait trafiği sakladıkları için oldukça küçük kapasitelidirler.

Bu alt bölümde şimdiye kadar, sorgudaki uzlamsal aralığın performansa olan etkisini gösteren grafiklere yer verilmiştir. Son olarak, Şekil 5.15 ise, indis yapılarının performansının, sorgudaki zamansal genişliğe olan duyarlılığını göstermektedir.




Şekil 5.15 M = "chicago" için toplam ıskalama sayısının sorgunun zamansal genişliğine bağlı olan değişimi

Buna göre, gezinge-2d R-tree ve MVR-tree yaklaşık 16 zamansal aralık adımından sonra sürekli bir artış göstermektedir. Bununla beraber, MVR-tree yaklaşık %50'den fazla artışla zamansal aralığa en fazla duyarlılık gösteren indis yapısıdır. Bunun nedeni, MVR-tree'nin zamana bağlı olarak ağaç düzeyinde genişlemesi nedeniyle sorgudaki zamansal aralıkla beraber, içerisinde kalan R-tree'lerin sayısının artmasıdır. Diğer yandan, MON-tree'nin zamansal aralığa bağımlılığı neredeyse yoktur. Çünkü sorgu kapsamında ikinci seviyedeki hareketleri tutan R-tree'ler tamamıyla 1. seviyede, uzlamsal aralığa göre belirlenmektedir.

#### *5.3.2.3 Ara Bellek Performansı*

Bu bölümde ara bellek performansı, Brinkhoff 'a (2002a) benzer şekilde,

$$AraBelekteToplamYakalamaSayisi \Big/ ToplamErişilenDügumSayisi \tag{5.1}$$

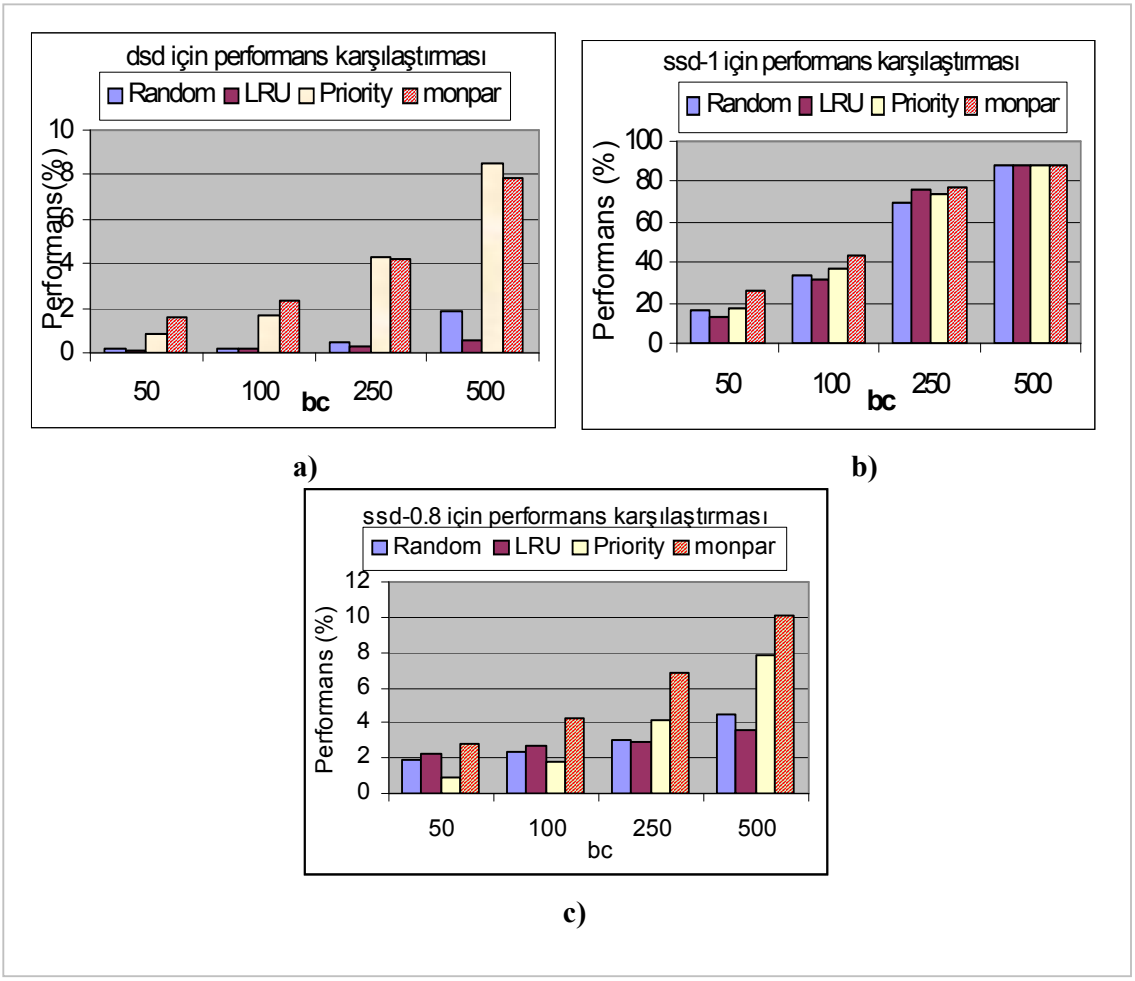
olarak tanımlanmıştır. Bu bölümün amacı Bölüm 4'de tasarımı verilen **monpar** ara bellek performansının ölçülmesi ve diğer standart ara bellekler ile karşılaştırılmasıdır. Geçen bölümlerden farklı olarak bu bölümde kullanılan parametre değerleri, Çizelge 5.6'de gösterilmiştir.

Çizelge 5.6 Ara bellek performansı deneylerinde kullanılan parametreler

| ***Parametre*** | ***Anlamı*** | ***Değerler, Değer Aralığı, Özellik*** |
|---|---|---|
| **ds** *(dataset size)* | Hareketli nesne sayısı | 250 |
| **sl** *(simulation length)* | Simülasyon adım sayısı | 400 |
| **bs** *(buffer size)* | Ara bellek kapasitesi | 50,100,250,500 |
| **bt** *(buffer type)* | Ara bellek tipi | "Rasgele", LRU, "Öncelikli", **monpar** |
| **qd** *(query distribution)* | Sorgu dağılımı | dsd, ssd-1,ssd-p |

Ara bellek kapasiteleri, **bs** ise oluşturulan MON-tree indisindeki toplam düğüm sayısının yaklaşık 0.5%, 1%, 2.5%, 5% ini içerebilecek şekilde değiştirilmiştir. Deneylerde kullanılan 3 farklı sorgu dağılımından; **dsd** (*düzenli sorgu dağılımı)*, sorgu uzlamsal ve zamansal aralıklarının rasgele olması, **ssd-1** ***(p=1** olasılıkla sıkıştırılmış sorgu dağılımı)* uzlamsal aralığın rasgele belirlenen %20'lik bölge içinde ve rasgele 10 *sim_adım*'lık zamansal aralık içinde olması ve **ssd-p** ***(p**-olasılıkla sıkıştırılmış sorgu dağılımı**)*** ise sorgunun **p** olasılık ile sıkışık sorgu, **(1-p)** olasılık ile **dsd** tipinde olması demektir. **dsd** ile ara belleğin rasgele sorgu setlerindeki başarımın ölçülmesi hedeflenirken, diğerlerinde sıkışık zaman-uzlamsal sorgulardaki başarımın ölçülmesi hedeflenmiştir.

**M**="chicago" tipindeki yol-ağ haritası üzerinde oluşturulan trafik, MON-tree indis yapısında saklanmıştır. Bu koşullar altında oluşturulan MON-tree indis yapısı 8809 disk sayfası içermektedir. Bunlardan 1330 tanesi *H-tipi* (konfigürasyon disk sayfaları), 241 tanesi *N-tipi* (1.seviyedeki yol-ağ indisine ait disk sayfaları) ve 7238 tanesi ise *M-tipi* (hareket indisine ait disk sayfaları) disk sayfalarıdır. Bu özellikteki MON-tree indis yapısı üzerinde tasarlanan **monpar** ara belleği üzerinden 100 elemanlı ve önceki paragrafta açıklanan dağılımlarda sorgu setleri çalıştırılmıştır. **monpar**'ın sorgulamalardaki performansı, gerçeklenen diğer 3 tipte ara bellek ("Rasgele", "Öncelikli", LRU) ile karşılaştırılmıştır.




Şekil 5.16 Farklı sorgu dağılımları için ara bellek performansına ait grafikler **a)** *dsd* sorgu dağılımı **b)** *ssd-1* sorgu dağılımı **c)** *ssd-0.8* sorgu dağılımı

Şekil 5.16a'da **dsd** için düşük kapasiteli ara belleklerde (toplam indisin yaklaşık %3'üne kadar) **monpar'ın** performans başarımının en iyi olduğu gözlenmiştir. Bununla beraber daha yüksek kapasiteli ara bellekler için Öncelikli (*Priority*) organizasyonundan daha iyi değilse de çok büyük bir performans kaybı gözlenmemiştir. Ara bellek kapasitesinin artmasıyla belleğin veriminin N,M,H setlerinin dağılımına olan duyarlılığı artmaktadır. Büyük kapasiteli belleklerde; ilk durumda ayrılan N,M,H kapasiteleri daha sonraki sorgular için doğru bir dağılım olmadığından böyle bir performans kaybı olduğu söylenebilir.

Şekil 5.16b'de, **ssd-1** dağılımlı sorgu setinin işlenmesi ile diske yapılan istek sayısının 500'den fazla olmadığı görülmektedir. Sıkışık bölge (*hot-spot*) karakteristiği için bu beklenen bir durumdur, çünkü hep aynı uzlamsal bölge içindeki nesnelere erişilmektedir. Dikkati çeken diğer bir özellik, LRU'nun bu tip sorgu dağılımları için başarımındaki artıştır. Büyük

kapasitelerde, **monpar'**ın rakibi olacak başarımı göstermiştir. Ara bellek kapasitesi=250 için, Öncelikli (*Priority*) organizasyonundan daha iyi sonuç vermiştir. **ssd-1** sorgu seti için **monpar'**ın başarımı bütün diğer ara bellek organizasyonlarından daha iyidir. Bunun temel nedeni "hot-spot" sorguların sabit bir karakteristik göstermesidir. Bu ilk durum karakteristikleri sorgu setinin ilerleyen diğer sorgularında da devam etmekte; böylece ara bellekte ayrılan N,M,H setlerinin ilk durum durağan kapasiteleri, simülasyon boyunca tatminkar değerler olmaktadır.

Son deneydeki **ssd-0.8** dağılımlı sorgu setinin amacı ara belleğin anlık değişimlere olan tepkisini değerlendirmektir (Şekil 5.16c). **ssd-1** dağılımındaki bütün ara belleklerin başarımı **ssd-0.8** dağılımındaki başarımdan yaklaşık %75 oranında daha düşüktür. Çünkü anlık değişimler tam uyarlamalı olmayan veya LRU-K gibi frekans değişimini sezemeyen ara belleklerin performansını düşürmektedir. Anlık değişimlerden en çok zarar gören LRU organizasyonlu ara bellektir. Özellikle, büyük kapasiteli belleklerde LRU'daki bu düşüş daha fazladır. **monpar**'ın anlık değişimlerde başarımı, diğerlerinden fazladır. Bununla beraber şu anki **monpar'**ı tam uyarlamalı hale getirmek; yani disk sayfası frekans değişimlerini sezebilen (LRU-K gibi) ve buna göre N,M,H setlerinin kapasitelerini simülasyon boyunca değiştirebilen bir ara bellek tasarımı gelecek çalışma hedefleri arasındadır.

## 5.4 Sonuç

Bu bölüm, hareketli nesne sisteminin gerçeklenmesi kapsamında önceki bölümde tasarlanan modellerin gerçeklenmesi ve gerçekleştirilen sorgulama sistemine bağlı olarak yapılan deneyler ve sonuçlarını içermektedir. Sistem, SaIL ve yol-ağ disk organizasyonu olmak üzere genel olarak iki ana modülde gerçeklenmiştir. SaIL, erişim yapıları, kalıcı saklama birimi ve ara bellek organizasyonları gibi temel birimlerin ve bu birimlere erişime olanak sağlayan ara yüzlerin gerçeklendiği JAVA kütüphanesidir. Yol-ağ organizasyonu ise, ana yollara ait indis yapısı, anayolların içerdikleri segmentleri tutan disk sayfaları ve ana kavşakların hilbert değerine göre gruplandığı disk sayfalarından oluşmaktadır.

Bu sistem üzerinde farklı karakteristiklerde trafikler üretilmiş ve oluşan hareket bilgileri sırasıyla 3DR-tree, MVR-tree, Gezinge-2d R-tree ve MON-tree indis yapılarında saklanmıştır. Bu yapıların kapladıkları disk alanı ve sorgulama performansları karşılaştırılmıştır. Bu karşılaştırmalı deneylerden elde edilen sonuçlar ve nedenlerinden bahsedilmiştir. Aşağıdaki liste, bu sonuçları özetlemektedir:

- MVR-tree'nin nesne sayısına (**ds**) olan bağımlılığı, simülasyon adım sayısına (**sl**) olan bağımlılığından daha fazladır. Diğer yapıların performansı, artan **ds** ve **sl** değerleri ile azaldığı görülmektedir. Bununla beraber **ds** ve **sl'**ye en az bağımlı olan yapının MON-tree olduğu gözlenmiştir.

- Beklendiği gibi, 3DR-tree bütün **ds** ve **sl** durumları için en az yer kaplayan indis yapısı iken, MVR-tree'nin ise bütün **ds** ve **sl** durumları için en fazla yer kaplayan indis yapısı olduğu dikkat çekmektedir.

- Sorgunun **uzlamsal genişliğine** en çok bağımlı olan indis yapısının Gezinge-2d R-tree olduğu tespit edilmiştir. Bu bağımlılık hareket ortamının karmaşıklığı ile de doğru orantılıdır. Diğer bir ifade ile yol-ağındaki ana yol ve kavşak sayısı yanında, bunların birbiriyle olan bağlantılarının bir ölçütü olan yol-ağı karmaşıklığı, Gezinge-2d R-tree'yi doğrudan etkileyen parametrelerdir. Sorgu uzlamsal genişliği ile performansı en çok etkilenen ikinci sıradaki yapı, MON-tree'dir. Artan sorgu uzlamsal genişliğinin, diğer yapıların disk erişim sayısında sebep olduğu artış oldukça azdır.

- Tahmin edileceği gibi sorguda **zamansal genişlikten** en çok etkilenen yapı MVR-tree'dir. Bununla beraber en az etkilenen yapının MON-tree olması da dikkat çekicidir.

3DR-tree ve Gezinge-2d R-tree'nin, gerçeklenebilmesi kolay ve mevcut veri tabanları ile entegre olabilen bir yapı olması diğerlerinden ayrılan yönüdür. Ancak Gezinge-2d R-tree'nin performansı, hareket ortamına bağlı olması nedeniyle oldukça düşüktür. Diğer yandan MON-tree, tam olarak özelleştirilmiş bir veri yapısı olup sorgulama performansında diğerlerinden daha iyi olduğu sonucuna varılabilir. Bunların yanında, R-tree'nin doğrudan kullanımı ile ölü alan artışındaki fazlalık, sorgulama performansında istenen verimin düşmesine neden olduğu ve bunun sonucunda, örneğin 3DR-tree'nin anlık-zamanda (veya zaman-kesit sorgulama) sorgu işleme performansının düşük olması gibi sonuçlar deneylerde gözlenmiştir.

Bu bölümde son olarak, MON-tree için tasarlanan **monpar** ara belleğinin sorgulamalardaki performansı, gerçeklenen diğer 3 tipte ara bellek ("Rasgele", Öncelikli, LRU) ile karşılaştırılmıştır. Bunların sonucunda, ilk olarak, **monpar**'ın küçük bellek kapasitelerinde başarımının diğerlerinden daha fazla olduğu gösterilmiştir. İkinci olarak, **monpar** organizasyonunun "hot-spot" sorgu setlerine ve anlık sorgu değişimlerine olan olumlu cevapları gösterilmiştir.

# 6. SONUÇ VE ÖNERİLER

Zaman-uzlamsal veri tabanlarının sürekli zamanda bir alt kümesi olarak düşünülen hareketli nesne veri tabanlarına günümüzde birçok ticari uygulamada ihtiyaç duyulmaktadır. Hareketli nesne takip sistemleri, konuma dayalı servisler, trafik kontrol, uçuş yönetimi bu alandaki uygulamalardan bazılarıdır.

Bu tez, hareketli nesne veri tabanlarında modelleme ve erişim yöntemleri konularını içermektedir. İlk olarak, bu konularda literatürdeki son gelişmelerden örnekler verilmiş, daha sonra bu gelişmelere dayandırılan katkılar anlatılmıştır. Örneğin, modelleme konusunda, "ADT ile modelleme", "kısıtlamalı lojik model" ve "MOST modeli" çalışmalarına yer verilmiştir. Erişim yapıları konusunda ise, uzlamsal erişim yapılarından R-tree ve benzer yöntemlerin temel işleyişleri üzerinde durulmuştur. Daha sonra, hareketli nesne erişim yapıları olarak R-tree tabanlı çözümler incelenmiş; özellikle 3DR-tree, MVR-tree, gezinge-2d ve MON-tree yapılarının tasarım ve algoritmalarından bahsedilmiştir.

Modelleme ve erişim yapılarında bahsedilen ön çalışmanın ışığında hareketli nesne sisteminin tasarlanması ve gerçeklenmesi anlatılmıştır. Buna göre, bu tez çalışmasının literatüre birbirinden bağımsız üç katkısı olduğunu söyleyebiliriz:

1. **Belirsizlik modeli**: Ağ üzerindeki hareketler için geliştirilen belirsizlik modeli üzerinde, olasılıksal aralık sorgulama algoritmaları geliştirilmiştir.
2. **Erişim yöntemleri**: Gerçekleştirilen hareketli nesne indis yapıları, 3DR-tree, MVR-tree, gezinge-2d ve MON-tree olup, SaIL (*Spatial Index Library)* isimli uzlamsal indis JAVA kütüphanesi kullanılarak gerçekleştirilmiştir.
3. **Ara bellek tasarımı**: Hareketli nesne indis yapısı için, tasarlanan zaman-uzlamsal ara bellek ile zaman-uzlamsal sorgulama performansında iyileşme sağlanmıştır.

Sistem tasarımında hareketin modellenmesi, güncellenmelerin modellenmesi ve belirsizlik modellenmesi konuları incelenmiştir. Mevcut sistemde trafik üretilmesi ve güncellenmeler hareket ortamına bağlı olan *segmente-dayalı* ve *T-zamansal periyoda dayalı* olmak üzere iki farklı yöntem ile gerçekleştirilmiştir. Bu noktada, gerçek hayata daha uygun modeller geliştirilebilir. Bunun için Brinkhoff'un (2002b) trafik üreteci kullanılabileceği gibi, gerçek trafik örneklerine de Internet üzerinden erişilebilir. Modellemede yapılan katkılardan, belirsizlik modeline dayalı olasılıksal sorgulama algoritmasının, $OAS^N$ dikkat çekici olduğu düşünülmektedir. Bununla beraber, bu tez çalışmasında belirsizlik model olarak tanımlanmış, modelin gerçeklenmesi ve belirsizliğe dayalı sorgulama kalitesine ait metriklerin

Cheng'dekine (2003) benzer şekilde belirlenmesi ileriye yönelik çalışmalara dahil edilmiştir.

Mevcut erişim yöntemlerinin birbiri ile karşılaştırılması, bu yapıların literatürde geçen özelliklerini ve mevcut karşılaştırma sonuçlarını doğrulamayı amaçlamakla beraber, yeni karşılaştırmalarla bazı çıkarımlara da zemin hazırlaması yönünden önemlidir. Erişim yapılarının karşılaştırılmalarında yer verimliliği, gerçeklenebilme kolaylığı, farklı sorgulama karakteristiklerinde erişim performansı ve mevcut veritabanları ile entegrasyonundaki başarım esas alınmıştır. Sonuç olarak, tez çalışması boyunca geliştirilen hareketli nesne sorgulama sisteminin ileriye yönelik çalışmalara zemin olacağı düşünülmektedir. Örneğin, kNN sorgulamanın yol-ağ organizasyonu üzerindeki hareketli nesneler üzerinde uygulanması ileriye yönelik önemli bir araştırma konusudur. Bu yönde geliştirilecek algoritmaların analizi için mevcut sistem yeni açılımlara uygundur.

**Monpar** ara bellek organizasyonu, sayfa yerleştirme kriterini disk sayfalarının içeriklerine göre belirleyen yarı-uyarlamalı bir çözüm getirmiştir. Bununla beraber şu anki **monpar'**ı tam uyarlamalı hale getirmek; yani disk sayfası frekans değişimlerini sezebilen (LRU-K gibi) ve buna göre N,M,H setlerinin kapasitelerini simülasyon boyunca değiştirebilen bir ara bellek tasarımı gelecek çalışma hedefleri arasındadır.

İleriye yönelik diğer bir çalışma, hareketli nesne veri tabanlarında erişim yapısı olarak, hesaba dayalı adresleme ve ayarlanabilir hücre-tabanlı yapılanma gibi indisleme dışındaki yaklaşımların incelenmesidir.

**KAYNAKLAR**

Barbara, D., Garcia-Molina, H. ve Porter, D. (1992) "The management of probabilistic data", IEEE TKDE, 4(5):487-502.

Bayer, R., McCreight, E., 1972, "Organization and Maintenance. of Large Ordered Indexes", Acta Informatica, Vol. 1, Fasc. 3,. pp. 173-189.

Beckmann, N., Kriegel, H.P. ve Seeger, B., (1990), "The R*-tree: an Efficient and Robust Method for Points and Rectangles", Proceedings of the ACM International Conference on Management of Data (SIGMOD), pp.322-331, Atlantic City, NJ.

Bentley, J. L., (1990), "K-d Trees for Semidynamic Point Sets", Proc. 6th Annual Symposium on Computational Geometry (SCG '90), 187–197.

Brinkhoff, T., (2002a), "A Robust and Self-tuning Page-Replacement Strategy for Spatial Database Systems", International Conference on Extending Database Technology (EDBT) 533-552.

Brinkhoff T., (2002b), "A Framework for Generating Network-Based Moving Objects." GeoInformatica 6(2): 153-180.

Cheng, R., Kalashnikov D.V. ve Prabhakar, S., (2004), "Querying Imprecise Data in Moving Object Environments", In IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE), Vol. 16, No. 9, pp. 1112-1127.

Cheng, R., Kalashnikov D.V. ve Prabhakar, S., (2003), "Evaluating Probabilistic Queries over Imprecise Data.", SIGMOD Conference, pp. 551-562

Cheng, R., Xia, Y., Prabhakar, S., Shah R., Vitter J.S., (2004), "Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data", The 30th International Conference of Very Large Database (VLDB), Toronto, Canada.

De Almeida, V.T., Güting, R.H., (2005), "Indexing the trajectories of moving objects in networks", GeoInformatica vol.9, no.1, pp. 33-60.

Erwig, M., Güting, R.H., Schneider, M., Vazirgiannis, M., (1999), "Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases", GeoInformatica 3(3): 269-296.

Erwig, M., Güting, R.H., Schneider, M., ve Vazirgiannis, M., (1998), "Abstract and Discrete Modeling of Spatio-Temporal Data Types", Proc. 6th ACM Syp.on Geographic Information Systems, pages 131-136, Washington, D.C.

Faloutsos, C., Roseman, S., (1989), "Fractals for Secondary Key Retrieval", Proc. Of ACM Conf. on Principles of Database Systems (PODS) , 247-252

Forlizzi, L., Güting, R.H., Nardelli, E., Schneider, M., (2000), "A Data Model and Data Structures for Moving Objects Databases", SIGMOD Conference, 319-330.

Frentzos, E., (2003), "Indexing Objects Moving On Fixed Networks", 8th Int'l Conf.on Spatial and Temporal Databases (SSTD), 289-305.

Guttman, A., (1984), "R-trees: A Dynamic Index Structure For Spatial Searching", Proceedings of the ACM SIGMOD,14:47-57

Güting, R.H., (1994), "An Introduction to Spatial Database Systems", VLDB Journal, vol.3

No.4.

Güting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N., Schneider, M., ve Vazirgiannis, M., (1998), "A Foundation for Representing and Querying Moving Objects" FernUniversität Hagen, Informatik-Report 238.

Gamma, E., Helm, R., Johnson, R. ve Vlissides, J., (1995), Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, New York, NY.

Güting, R.H. ve Schneider, M., (2005), Moving Object Databases, Morgan Kaufmann Publishers.

Grumbach, S., Rigaux, P., Scholl, M., ve Segoufin, L., (1997), "DEDALE, A Spatial Constraint Database", Proc. Intl.Workshop on Database Programming Languages, 38-59.

Heper, O., (2005) Yıldız Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, Bitirme Projesi.

Hadjieleftheriou, M., (web sitesi), SaIL, Spatial Index Library, http://www.cs.ucr.edu/~marioh/spatialindex/index.html

Hadjieleftheriou, M., Hoel, E.G., Tsotras, V.J., (2005), "SaIL: A Spatial Index Library for Efficient Application Integration", GeoInformatica 9(4): 367-389.

Hadjieleftheriou, M., Kollios, G., Tsotras, V.J., Gunopulos, D., (2006), "Indexing spatiotemporal archives", VLDB Journal 15(2): 143-164.

Kahveci, T.Y., Kahveci, T., Singh, A., (2000), "Buffering of Index Structures" The International Society for Optical Engineering (SPIE)-Internet Multimedia Management Systems, Boston.

Kalay, U., Kalıpsız, O., (2005), "Probabilistic Point Queries Over Network-based Movements", 20th Int'l Conf. ISCIS, 28-30 Oct., Istanbul, published in Lecture Notes in Computer Science, Springer Verlag, 2005.

Kalay, U., Kalıpsız, O., (2006), "MONPAR: A Page Replacement Algorithm for a Spatiotemporal Database", XVI. International Enformatika Conference, Venice, Italy.

Kamel, I., Faloutsos, C., (1994), "Hilbert R-tree: An Improved R-tree Using Fractals", Proceedings of the 20th Conference on Very Large Data Bases (VLDB).

Kollios, G., Gunopulos, D., Tsotras, V.J. (1999), "On Indexing Mobile Objects", Proc. of ACM Conf. on Principles of Database Systems (PODS), pp. 261-272.

Kollios, G., Tsotras, V.J., Gunopulos, D., Delis, A., Hadjieleftheriou, M., (2001), "Indexing Animated Objects Using Spatiotemporal Access Methods", IEEE Trans. Knowl. Data Eng. 13(5): 758-777.

Koubarakis, M., Sellis, T.K., Frank, A.U., Grumbach, S., Güting, R.H., Jensen, C.S., Lorentzos, N.A., Manolopoulos, Y., Nardelli, E., Pernici, B., Schek, H.J., Scholl, M., Theodoulidis, B., Tryfona, N., (2003), "Spatio-Temporal Databases: The CHOROCHRONOS Approach", Lecture Notes in Computer Science, 2520, Springer.

Kumar, A., Tsotras, V.J., Faloutsos, C., (1998), "Designing Access Methods for Bitemporal Databases", IEEE Trans. on Knowledge and Data Engineering, vol.10, no.1, p.1-20.

Lakshmanan, L.V.S., Leone, N., Ross, R., Subrahmanian, V.S., (1997), "ProbView: A

Flexible Probabilistic Database System", ACM Trans. Database Syst. 22(3): 419-469.

Leutenegger, S. T. ve Lopez, M.A. (1998), "The Effect of Buffering on the Performance of R-Trees" ICDE'98, pp. 164-171.

Meng**,** X., Ding, Z., (2003), "DSTTMOD: A Discrete Spatio-Temporal Trajectory Based Moving Object Database System" In Proceedings of 14th International Conference on Database and Expert Systems Applications(DEXA),p: 444-453, Prague, published in Lecture Notes in Computer Science 2736, Springer.

Nascimento, M. A. ve Silva, J.R.O.,(1998), "Towards historical R-trees", In Proc. of the ACM Symp. on Applied Computing, (SAC), pages 235–240.

Nascimento, M., Silva, R., and Theodoridis, Y., (1999), "Evaluation of access structures for discretely moving points", In Proceedings of the International Workshop on Spatio-Temporal Database Management, pp.171-188.

Nievergelt, J., Hinterberger, H., Sevcik K.C., (1984), "The Grid File: An Adaptable, Symmetric Multikey File Structure", ACM Trans. on Database Systems, 9(1):38-71.

O'Neil, E.J., O'Neil, P.E., Weikum, G., (1993), "The LRU-K Page Replacement Algorithm For Database Disk Buffering", Proc. ACM SIGMOD International Conference on Management of Data, 297,306

Papadias, D., Zhang, J., Mamoulis, N., ve Tao Y., (2003), "Query processing in spatial network databases", 29th Intl. Conf. on Very Large Data Bases (VLDB), pages 802-813.

Pelanis, M, Saltenis, S, Jensen, C.S., (2006), "Indexing the past, present, and anticipated future positions of moving objects", ACM Transactions on Database Systems (TODS), Volume 31 , Issue 1 (March ), pages: 255 – 298.

Pfoser, D., Jensen C.S., (1999), "Capturing the uncertainty of moving-object representations", Proceedings of Advances in Spatial Databases, 6th Intl. Symp. (SSD), pages 111-132.

Pfoser, D., Jensen C.S., (2001) "Querying the trajectories of on-line mobile objects", 2nd ACM Intl. on Data Engineering for Wireless and Mobile Access (MobiDE):66-73.

Pfoser, D., Jensen C.S., (2003), "Indexing of network constrained moving objects", GIS 2003: 25-32

Pfoser, D., Jensen, C.S., ve Theodoridis, Y., (2000), "Novel approaches in query processing for moving object trajectories", 26th Intl. Conf. on Very Large Data Bases (VLDB), pages 395-406.

Porkaew K., Lazaridis I., Mehrotra S., (2001), "Querying Mobile Objects in Spatio-Temporal Databases", Int'l Conf.on Spatial and Temporal Databases SSTD, 59-78.

Samet, H., (1995), "Spatial Data Structures", In W. Kim, editor, Modern Database Systems: The Object Model, Interoperability and Beyond, pages 361-385, Addison Wesley/ACM.

Sacco, G.M., (1997), "Index Access with a Finite Buffer" Proc. of the Very Large Data Bases Conf. (VLDB), pp. 301-309.

Sellis T.K., (1999), "Research Issues in Spatio-temporal Database Systems", Proceedings of Advances in Spatial Databases, 6th Intl. Symp. (SSD), pages: 5-11

Sellis, T., Roussopoulos, N. ve Faloutsos, C., (1987), "The R+-Tree: A dynamic index for multi-dimensional objects". Intl. Conf. on Very Large Data Bases (VLDB).

Shekhar, S., Liu, D.R., (1997), "CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations", IEEE Trans. Knowl. Data Eng.(9): 102-119.

Shekhar, S. ve Chawla, S., (2003), Spatial Databases: Atour, Prentice Hall, New Jersey.

Sistla, A.P., Wolfson, O., Chamberlain, S., Dao S., (1997), "Modeling and Querying Moving Objects", Intl. Conf. On Data Engineering (ICDE), 422-432.

Su, J., Xu, H., ve Ibarra, O.H., (2001), "Moving objects: Logical relationships and queries", Advances in spatial and temporal databases. International symposium No7, Redondo Beach CA.

Tao, Y., Papadias, D., (2001), "The MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries" Proceedings of the 27th Very Large Data Bases Conference (VLDB), pp. 431-440, Rome, Italy.

Tao, Y., Papadias, D., (2002), "Time Parameterized Queries in Spatio-Temporal Databases", Proceedings of the ACM International Conference on Management of Data (SIGMOD).

Theodoridis, Y., Vazirgiannis, M., Sellis, T.K., (1996), "Spatio-temporal indexing for large multimedia applications", Proc. 3rd IEEE International Conference on Multimedia Computing and Systems, Hiroshima, pp. 441–448.

Vazirgiannis, M., Wolfson, O., (2001), "A Spatiotemporal Model and Language for Moving Objects on Road Networks", 7th Int'l Conf.on Spatial and Temporal Databases (SSTD), 20-35.

Wolfson, O., Sistla, P.A., Chamberlain, S., ve Yesha, Y., (1999), "Updating and Querying Databases that Track Mobile Units", Distributed and Parallel Databases Journal, vol.7, no.3 pp.257-387.

Wolfson, O., Xu, B., Chamberlain, S., ve Jiang, L., (1998), "Moving objects databases: Issues and solutions", 10th Intl. Conf. on Scientific and Statistical Database Management (SSDBM), pages 111-122.

**EKLER**

## Ek 1 MOST modelindeki konum yenileme modelleri

1-) Hız gözü-kapalı-tahmin modeli, **sdr** (***s***peed ***d***ead-***r***eckoning):

Bu modelde, hareketli nesne, hareketinin başlangıcında, bütün seyahati boyunca geçerli olan sabit bir *L.uncertainty* değerini tasarsız olarak belirler. Böylece, seyahat boyunca bütün güncellenmeler, konum sapması bu değeri geçtiği zaman gerçekleşir.

2-) Uyarlanır gözü-kapalı-tahmin modeli, **adr** (***a***daptive ***d***ead-***r***eckoning) :

Bu modele göre, nesne, her yenilemede, maliyete dayalı bir yöntemle yeni bir **th**=*L.uncertainty* sınır değerini üretir ve bu değer güncellenme aralığı boyunca sabit kalır. **th**, "bilgi maliyeti" (*information cost*) adı verilen yükün birim zaman dilimindeki değerini en küçük yapacak şekilde belirlenir. Bir güncellenme aralığında "bilgi maliyeti", sapma maliyeti, belirsizlik maliyeti ve haberleşme maliyeti değerlerinin toplamı olup, aşağıdaki gibi tanımlanmıştır. $t_1$, $t_2$ ardışık güncellenme zamanları olmak üzere;

$$COST[t_1,t_2) = COST_s[t_1,t_2) + COST_b[t_1,t_2) + C_1 \qquad \text{(Ek 1.1)}$$

Sapma ve belirsizlik, yanlış sorgu sonuçlarına sebep olduğu için bir maliyeti vardır. Bu maliyet, sapma ve belirsizliğin sayısal değeri ve süresi önemli olduğu için; sapma ve belirsizliğin zamana bağlı fonksiyonlarının, s(t) ve b(t), t üzerinden entegrasyonu ile belirlenir, şöyle ki;

$$COST_s[t_1,t_2) = \int_{t_1}^{t_2} s(t)dt \qquad \text{(Ek 1.2)}$$

$$COST_b[t_1,t_2) = \int_{t_1}^{t_2} C_2 b(t)dt \qquad \text{(Ek 1.3)}$$

Burada, birim zaman diliminde, bir birim sapma maliyeti 1 olarak seçilirken; birim zaman diliminde, bir birim belirsizlik maliyeti $C_2$ olarak belirlenmiştir. (Ek 1.1)'deki formülde, $C_1$, haberleşme kanalı ve işlemci gibi kaynak kullanımı ile ilgili maliyetleri temsil etmektedir.

Veri tabanının *L.uncertainty* olarak kaydettiği **th** değerinin belirlenmesi, nesnenin bir sonraki güncellenmeyi ne zaman yapacağının tahmin edilmesi ile diğer bir ifade ile konum sapmasının bundan sonraki davranışının tahmin edilmesi ile gerçekleştirilir. Bu tahmin değeri de, bilgi maliyetinin birim zaman dilimindeki değeri olan,

$$f(t_2) = \frac{COST[t_1, t_2)}{(t_2 - t_1)} \qquad \text{(Ek 1.4)}$$

fonksiyonunu en küçük değer yapacak şekilde belirlenir. Buna göre, yeni belirsizlik değeri, (Ek 1.4) ifadesinin, global minumum noktasındaki **th** değeri olacaktır. Katsayıların belirlenmesi ve gerekli dönüşümler ve hesaplamalar sonunda **th**=*L.uncertainty*= $\sqrt{\frac{2aC_1}{2C_2 + 1}}$ olduğu Wolfson vd.'de () gösterilmiştir.

3-) Sökme-algılamalı gözü-kapalı-tahmin modeli, **dddr** (***d***isconnection ***d***etection ***d***ead-***r***eckoning):

**sdr** ve **adr**'de nesnenin güncellenme raporu göndermemesi, "konumun belirsizlik bölgesi içinde olduğu anlamına gelmesi" her zaman geçerli değildir. Örneğin, bağlantı sökülmesi (kaza veya çeşitli nedenlerden, nesne hareketine son verebilir) durumunda, veri tabanı güncellenme raporu almayacaktır. Bağlantı sökülmesinin algılanması için belirli aralıklarla hareketli nesne ile bağlantıya geçilmesi kablosuz kanalı gereksiz kullanımına neden olacağı için, daha değişik bir yöntem önerilmiştir. Buna göre, **dddr**, son güncellenme zamanında belirlediği **th** sınır değerini, her geçen birim zaman diliminde belirli oranda azaltır. Örneğin, son güncellenme zamanında, **adr**'deki yönteme benzer şekilde bulunan **th** değeri, bir sonraki birim zamanda **th/2** ve sonraki zaman dilinde **th/3** olup bu şekilde devam edecektir. Böylece, her geçen zaman diliminde **th** değeri azalmasına rağmen hala güncellenme raporu gelmiyorsa nesnenin bağlantısının söküldüğü kanaatine varılabilecektir.

**sdr**, **adr** ve **dddr** güncellenme yönetim modellerinin deneysel ortamda karşılaştırılmaları sonucunda, **adr**'nin diğer yöntemlere göre daha başarılı olduğu gösterilmiştir.

**Ek 2 SaIL Kütüphanesi ara yüzleri ile sabit disk üzerinde bir indisin oluşturulması ve indise erişimin *fifo* ara bellek üzerinden gerçekleştirilmesini sağlayan kod parçası**

Örneğin, 4096B'lık disk sayfalarından oluşan bir sabit disk ortamında, her düğümünde (indis ve yaprak düğümlerinde) en fazla 10 en az 7 adet, 2 boyutlu uzlamsal kayıt tutabilen bir R-tree; "*rtree"* isimli dosyada saklamak isteyelim. Böyle bir organizasyon, Şekil Ek 2.1'deki "RtreeLoad.java" dosyasındaki örnek kod parçaları ile gerçeklenmiştir. `diskfile` isimli sabit disk biriminin özellikleri (*ps1*) belirlendikten sonra bu ortama erişim için *fifo* tipinde, 10 adet disk sayfası kapasiteli bir ara bellek (**fifobuffer)** tanımlanmıştır. Daha sonra, istenen yapısal özelliklerde (*ps2*), ***tree*** isimli bir R-tree indisi, **fifobuffer** üzerinden erişilecek şekilde tanımlanmıştır. Son olarak, Şekil Ek 2.1'deki "Rtree.java" dosyasında ise, *tree* indisine ait bazı özellikler *byte* dizisi olarak paketlenip **fifobuffer** üzerinden, *m_headerID id'*li disk sayfasına yazılmaktadır.

***RtreeLoad.java***

```
public RtreeLoad(){
 . . . .. .
 PropertySet ps1 = new PropertySet();
 ps1.setProperty("Overwrite",
                    new Boolean(true));
 ps1.setProperty("FileName", "rtree");
 ps1.setProperty("PageSize",
                    new Integer(4096));
 IstorageManager diskfile = new
                DiskStorageManager(ps1);
 Ibuffer fifobuffer = new
          FIFOBuffer(diskfile,10,false);

 PropertySet ps2 = new PropertySet();
 ps2.setProperty("FillFactor",
                     new Double(0.7));
 ps2.setProperty("IndexCapacity",
                    new Integer(10));
 ps2.setProperty("LeafCapacity",
                    new Integer(10));
 ps2.setProperty("Dimension",
                    new Integer(2));
 ISpatialIndex tree =
             new RTree(ps2, fifobuffer);
 .. . ..// ağaca bilgi yüklenmesi
}
```

***RTree.java***

```
public class Rtree
             implements ISpatialIndex
{
  int m_rootID;
  int m_treeVariant;
  int m_treeHeight;
  IstorageManager m_storageManager;

  public RTree(PropertySet p,
               IstorageManager s){
     m_storageManager = s;
       . . . .. .
  }

  public storeHeader()
    ByteArrayOutputStream bs = new
              ByteArrayOutputStream();
    DataOutputStream ds = new
              DataOutputStream(bs);
    ds.writeInt(m_rootID);
    ds.writeInt(m_treeVariant);
    ds.writeInt(m_treeHeight);
    ds.flush();

    m_storageManager.storeByteArray(
      m_headerID, bs.toByteArray());
}
```

Şekil Ek 2.1 Diske soyutlanmış bir erişim

Bu örneğin amacı sabit disk ortamının *byte* dizisi olarak oluşturulması ve bir ara bellek üzerinden diske erişimin sağlanmasını göstermektir. Dikkat edileceği gibi, R-tree indisindeki **storeHeader()** fonksiyonunda indisin özellikleri paketlenip diske yazılırken

**m_storageManager**.**storeByteArray**(…) fonksiyonu kullanılmıştır. Bu tasarım ile diske erişen uygulamanın diske nasıl erişildiğinden, ara bellek kullanılıp kullanılmadığından veya ara belleğin özelliklerinden habersiz olması sağlanmıştır ki; bu yönüyle başarılı bir soyutlama ve esnek kullanım örneğidir. Örnekte görüldüğü gibi kalıcı saklama birimine yazma işlemi bir satırda gerçekleştirilmektedir. Yukarıdaki örnekte, **`fifobuffer`** istemci ile gerçek saklama yönetici birimi arasında *proxy* tasarım kalıbına uygunluk göstermektedir.

**Ek 3 SaIL Kütüphanesi kapsamında R-tree indisi üzerinde gerçekleştirilen aralık sorgusundaki disk erişim sayısının belirlenmesi**

```
class MyVisitor2 implements IVisitor
{
  public int m_indexIO = 0;
  public int m_leafIO = 0;
  public HashMap answers=new HashMap();

  public void visitNode(final INode n)
  {
    if (n.isLeaf()) m_leafIO++;
   else m_indexIO++;
  }

  public void visitData(final IData d)
  {
   answers.put(new Integer(d.getIdentifier()),d.getShape());
  }
 }
```

```
void rangeQuery(int type, final IShape query, final IVisitor v){
   m_rwLock.read_lock();
   try{
    Stack st = new Stack();
    Node root = readNode(m_rootID);
    if (root.m_children > 0 && query.intersects(root.m_nodeMBR))
            st.push(root);
    if(st.empty())
            v.visitNode((INode) root);
    while (! st.empty()){
              Node n = (Node) st.pop();
              if (n.m_level == 0){
                        v.visitNode((INode) n);
                        for (int cChild = 0; cChild < n.m_children; cChild++){
                                  boolean b;
                                  if (type == SpatialIndex.ContainmentQuery)
                                            b = query.contains(n.m_pMBR[cChild]);
                                  else b = query.intersects(n.m_pMBR[cChild]);
                                  if (b){
                                            Data data = new Data(n.m_pData[cChild],
                                            n.m_pMBR[cChild],
                                            n.m_pIdentifier[cChild]);
                                            v.visitData(data);
                                  }
                        }
              }
              else{
                        v.visitNode((INode) n);
                        for (int cChild = 0; cChild < n.m_children; cChild++){
                                  if (query.intersects(n.m_pMBR[cChild])){
                                            st.push(readNode(n.m_pIdentifier[cChild]));
                                  }
                        }
              }
         }  //while
  } //try
}
```

Şekil Ek 3.1 Sorgu işlemede disk erişim sayısının belirlenmesine olanak sağlayan kod

Şekil Ek 3.1'de, R-tree indisinde aralık sorgusunun gerçeklenmesi ile ilgili kodun bir kısmı görülmektedir. R-tree indisinde aralık sorgusu işlemi, indisin kökünden başlayıp, yapraklarda sonlanır. Buna göre ilk olarak kök düğüm okunup, sorgudaki uzlamsal şekil ile kesişimi kontrol edilmiştir. Sorgunun uzlamsal şekli ne olursa olsun, sorgu *IShape* ara yüzünü gerçeklediği için **intersect (..)** fonksiyonunda doğrudan çağırılabilmektedir.

MBR'ı sorgu bölgesi ile kesişen düğümler, **st** isimli Stack'ta tutulmaktadır. Böylece **st** stack'ındaki düğümler ara yüzdeki **visitNode()** fonksiyonu ile sayılmaktadır. **visitData()** fonksiyonu ise sorgu sonucundaki nesneleri çıkış ünitesine aktarmaktadır.

## ÖZGEÇMİŞ

| | | |
|---|---|---|
| Doğum tarihi | 18.11.1975 | |
| Doğum yeri | Bursa | |
| Lise | 1989-1992 | İstanbul Atatürk Fen Lisesi |
| Lisans | 1992-1997 | İstanbul Teknik Üniversitesi Elektrik-Elektronik Fak., Elektronik-Haberleşme Mühendisliği Bölümü |
| Yüksek Lisans | 1998-2000 | Güney Kaliforniya Üniversitesi, Bilgisayar Bilimleri *(Univ. of Southern Calif., Computer Science Dept.)* |
| Doktora | 2001-2007 | Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Müh. Bölümü, Yazılım Ana bilim Dalı |

**Çalıştığı kurum(lar)**

| | |
|---|---|
| 1997-1998 | İTÜ Elektronik-Haberleşme Müh. Araştırma Görevlisi |
| 2001-Devam ediyor | YTÜ Bilgisayar Mühendisliği Bölümü Araştırma Görevlisi |