I

T.C.

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK BOYUTLU VERİLERİN İNDEKSLENMESİ İÇİN KULLANILAN YÖNTEMLER VE BU YÖNTEMLERİN KARŞILAŞTIRILMASI

Onat GÖZET

Yüksek Lisans Tezi

Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Yrd. Doç. Dr. Cavit TEZCAN

Edirne 2008

T.C.

TRAKYA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK BOYUTLU VERİLERİN İNDEKSLENMESİ İÇİN KULLANILAN YÖNTEMLER VE BU YÖNTEMLERİN KARŞILAŞTIRILMASI

Onat GÖZET

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI

Bu tez 02/07/2008 tarihinde aşağıdaki jüri tarafından kabul edilmiştir.

Yrd. Doç. Dr. Cavit TEZCAN
Danışman

Prof. Dr. Mesut RAZBONYALI
Üye

Yrd. Doç. Dr. Aydın CARUS
Üye

ÖZET

Yüksek Lisans Tezi, Çok Boyutlu Verilerin İndekslenmesi İçin Kullanılan Yöntemler Ve Bu Yöntemlerin Karşılaştırılması, T.C. Trakya Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Yakın geçmişte gelişen bilgisayar teknolojisiyle birlikte çok boyutlu verileri işleyen uygulamalar artmış ve çok boyutlu verilere hızlı ve etkin ulaşım için çeşitli yöntemlere ihtiyaç duyulmuştur. Bunun üzerine tek boyutlu veri indeksleme yöntemleri yerine çok boyutlu veri indeksleme yöntemleri araştırılmaya başlanmıştır. Bu çalışmanın amacı da bu yöntemlerin önemli olan birkaç tanesini bulunuş zamanlarına göre incelemek, etkinliklerini araştırmak ve bu yöntemler arasında performans bakımından karşılaştırma yapabilmektir.

Bu tez 2008 yılında yapılmıştır ve 89 sayfadan oluşmaktadır.

ANAHTAR KELİMELER: R Ağaç, X Ağaç, TV Ağaç, R* Ağaç, Piramit Tekniği, Çok boyutlu veri indeksleme

ABSTRACT

Graduate Thesis, Methods For Indexing Multidimensional Data And Comparisons Between These Methods, T.C. Trakya University, Graduate School Of Natural And Applied Sciences, Department Of Computer Engineering.

Recently with improvement on computer technology, applications that process multidimensional data increased and methods to reach multidimensional data fast and effective became needed. Thereupon instead of one-dimensional data indexing methods, multidimensional data indexing methods were started to research. Purpose of this work is investigate some of these methods by order of proposed date, analyse their efficiency and compare their performances.

This work is done in 2008 and consists of 89 pages.

ANAHTAR KELİMELER: R Tree, X Tree, TV Tree, R* Tree, Pyramid Technique, Multidimensional data indexing

İÇİNDEKİLER

1. GİRİŞ 1

1.1. Çok Boyutluluk Kavramı 1

1.2. Bilgisayar Dünyasında Çok Boyutluluk 2

2. ÇOK BOYUTLU VERİ İNDEKSLEME YÖTEMLERİ 4

2.1. R Ağaç 4

2.1.1. R Ağaç İndeks Yapısı 5

2.1.1.1. Kayıtlar 5

2.1.1.2. Özellikler 6

2.1.1.3. Taşma ve Aşağı Taşma 7

2.1.1.4. M ve m Nasıl Ayarlanır 8

2.1.2. Algoritmalar 8

2.1.2.1. Arama 10

2.1.2.2. Ekleme 11

2.1.2.3. Silme 12

2.1.2.4. Bir Düğümü Bölme 15

2.1.2.4.1. Ayrıntılı Bölme Algoritması 15

2.1.2.4.2. İkinci Dereceden Yük Algoritması 16

2.1.2.4.3. Doğrusal Yük Algoritması 16

2.1.3. R Ağacın Türevleri 17

2.2. R* Ağaç 17

2.2.1. R* Ağaç Algoritmaları 18

2.2.1.1. Alt Ağaç Seçme Algoritması 18

2.2.1.2. R* Ağacın Bölünmesi 19

2.2.1.3. Zorunlu Yeniden Ekleme 20

2.3. TV Agaç 22

2.3.1. TV Ağaç Yönteminin Arkasındaki Sezgi 24

2.3.1.1. Teleskopik Fonksiyon 24

2.3.1.2. Çevreleyen Alanın Şekli 26

2.3.2. TV Ağaç 26

2.3.2.1. Düğüm Yapısı 26

2.3.2.2. Ağaç Yapısı 27

2.3.2.3. Algoritmalar 29

2.4. X Ağaç 33

2.4.1. X Ağaç Yapısı 34

2.4.2. Algoritmalar 35

2.4.3. Minimal Üst Üste Binme Bölünmesi 37

2.5. Piramit Tekniği 38

2.5.1. Veri Uzayı Parçalama 39

2.5.2. İndeks Yaratma 42

2.5.3. Sorgu İşleme 42

2.6. Yeni Piramit Tekniği Yöntemi 47

2.6.1. Yöntemin Amacı 48

2.6.1.1. Yüksek Boyutluluğun Sezilemezliği 48

2.6.1.2. Yüksek Boyutlu Veri Kümesinin Dağılımı 49

2.6.1.3. Hiperküpte Köşe,Kenar ve HiperDüzlemler 50

2.6.2. Yüzey Tabanlı Uzaysal indeks Yapısı 52

2.6.2.1. Piramit Sırası 52

2.6.2.2. İndeks Anahtarının Oluşumu 54

2.6.2.3. Aralık Arama 55

2.6.3. Piramitlere Sınır Atanması 57

3. İNDEKSLEME YÖNTEMLERİNİN KARŞILAŞTIRILMASI 60

3.1. Yöntemlerin Karşılaştırılması ve Elde Edilen Sonuçlar 60

3.1.1. R Ağaç-R* Ağaç Karşılaştırması 60

3.1.2. R* Ağaç-TV Ağaç Karşılaştırması 66

3.1.2.1. Sonuçlar 66

3.1.3. X Ağaç-TV Ağaç-R* Ağaç Karılaştırması 70

3.1.4. Piramit Tekniği-X Ağaç Karşılaştırması 73

3.1.4.1. Sentetik Veri Kullanarak Yapılan Karş. 74

3.1.4.2. Gerçek Veri Kullanarak Yapılan Karş. 76

3.1.5. Piramit Tekniği-Yeni Piramit Tekniği Karşılaştırması 77

3.1.5.1. Aday Küme Büyüklüğü CPU Zamanı İlişkisi 78

3.1.5.2. Sayfa Erişim Sayısı ve CPU Zamanı 79

3.2. Piramit Tekniği Uygulamasıyla Yapılan Karşılaştırmalar 81

3.2.1. Piramit Tekniği Uygulaması 81

3.2.2. Karşılaştırma ve Sonuçlar 82

4. SONUÇ **85**

Kaynaklar **87**

Teşekkür **88**

Özgeçmiş **89**

1. GİRİŞ

Bilgisayar dünyasındaki gelişmeler sonrasında video, ses gibi veri dosyalarının etkin bir biçimde yönetilebilmesi için bu verilere erişim için uygun yöntemler gerekli oldu. Hızlı bir erişim için günümüze kadar birçok yöntem geliştirildi. Ancak bunlara geçmeden önce çok boyutluluk kavramı ve bilgisayar dünyasında çok boyutluluğa göz atalım.

1.1. Çok Boyutluluk Kavramı

Boyut günümüzde sıklıkla kullanılan bir kavramdır. Genelde boyut kavramını tek olarak ele almaz ve önündeki kelimelerle anlamlandırırız. Mesela iki boyutlu, üç boyutlu, çok boyutlu... Böyle bir kelime öbeği duyduğumuzda beynimiz içinde bahsedilen cisimden önce onun bulunduğu ortam çizilir. Daha sonra bu ortama cismi yerleştiririz. Buradan boyut kelimesinin ilk olarak bir ortamı, bir uzayı tarif ettiğini çıkarabiliriz. Boyut kelimesinin sözlük anlamlarına bakarsak "Bir cismin herhangi bir yöndeki uzantısı" tanımıyla karşılaşırız. Cisimler bir uzayda belli yönlerdeki uzantılarının varlığıyla boyutlandırılırlar. Bir cismin uzayda en azından bir yönde bir uzantısı olmalıdır ki o cisim var olsun. Böyle tek yönde uzantısı olan cisimlere tek boyutlu cisimler diyoruz. Aynı şekilde iki yönde uzantısı olan cisimlere iki boyutlu, üç yönde olanlara üç boyutlu cisimler diyoruz ve bu isimlendirme sayılar gibi sonsuza kadar gider.

Çok boyutluluk kavramı da boyutsallığı birden fazla olan cisimler için geçerlidir. Burada birden fazla boyutta uzantısı olan sanal ya da gerçek cisimlerden bahsetmek mümkündür.

Fiziksel dünyada maksimum üç tane boyuttan bahsedilebilir. Bunlar genişlik, uzunluk ve derinliktir. Etrafımızdaki cisimlerin hepsi üç boyutlu bir uzayda yer alırlar.

Fakat teorik uzaylarda boyut sayısı sonsuza kadar çıkabilir. Beynimiz boyutsallığı üçten fazla olan uzayları zihnimizde canlandırmakta başarısızdır. Bu yüzden bu tür teorik uzay ortamları üzerinde yapılacak işlemlerde bize matematik yardım eder. Matematiksel açıdan boyut ise cisimleri ölçmeye yarayan bir kavramdır. Yani aslında bir cismi ölçerken matematikte cismin bütün boyutlarındaki uzantı miktarlarından yararlanılır.

Aslında boyutu genel anlamda ölçülebilir bir özellik olarak da ele alabiliriz. Bu bakımdan bakarsak etrafımızda çok boyutlu olarak nitelendirebileceğimiz birçok örnek bulabiliriz. Örneğin yarışmakta olan bir koşucunun ağırlık, boy, yaş ve hız gibi nitelikleri birer boyut olarak ele alınıp dört boyutlu bir uzayda bu sporcu temsil edilebilir. Her ne kadar bu dört boyutlu uzayı kafamızda canlandıramasak da üç boyutlu uzaydaki nesnelerin yerleşimlerinden bu koşucunun dört boyutlu bir uzayda bir noktayla temsil edilebileceğini çıkarabiliriz. Aynı şekilde ağırlık, boy, yaş ve hız gibi özellikleri biraz daha farklı bir sporcunun aynı uzayda farklı bir yerdeki noktayla temsil edileceğini söyleyebiliriz.

Günümüzde çok boyutlu uzay kullanılarak iki verinin özelliklerinin karşılaştırması çok fazla yapılmaktadır. Bu tip işlemlere birçok alanda sık sık başvurulmaktadır. Bu verilerin düzgün işlenmesi ve aralarındaki ilişkilerin saptanması karar vermede önemli bir rol oynar. Bu yüzden çok boyutluluğun önemi çok fazladır.

1.2. Bilgisayar Dünyasında Çok Boyutluluk

Çok boyutlu veriler üzerine işlemler zor, zahmetli ve zaman alıcıdır. Bu verilerin öncelikle uzayda karışıklığa neden olmayacak şekilde temsil edilmesi ve boyutlara göre iyi organize edilmesi gerekir. Daha sonra bu boyutsal özelliklerden yararlanarak iki veri arasında çeşitli çıkarımlara ulaşılabilir. Uzayda çok iyi temsil edilen veriler üzerinde hesaplar yine de çok vakit kaybettirici olabilir. Ayrıca hatalara da açıktır. Bu yüzden verilerin daha düzgün saklanması, bu verilere erişim ve bu verilerle ilgili işlemlerinde bilgisayarlar kullanılmaktadır.

Bilgisayarlarda bu veriler bir veritabanında saklanır ve istenildiği zaman bilgilere bu veritabanından ulaşılır. Bu tip veritabanlarına örnek verecek olursak resim veritabanları, tıbbi veritabanları, zaman serileri veritabanları, multimedya veritabanları, DNA veritabanları gibi çeşitli veritabanlarını sayabiliriz. Bu veritabanlarının ortak özelliği çok boyutlu verileri indekslemesidir.

Çok boyutlu verilerin bulunduğu veritabanlarında istenilen verilere hızlı ulaşım önemlidir. Verilere ulaşımın yavaş olduğu sistemlerde yapılan işlemler çok uzun süreceğinden mümkün olduğu kadar hızlı ve doğru bilgiye ulaşmak çok hedeflenen ve üzerinde çok araştırma yapılan bir konu olmuştur. Bu konuda birçok araştırma yapılmış ve çeşitli yöntemler ortaya koyulmuştur. Bu yöntemler verilerin saklanmasını ve bunlara ulaşmayı içeren yöntemlerdir. Çeşitli yapılar geliştirilmiş, hızlandırıcı ve yavaşlatıcı etmenler bulunmuştur. Bu konuda uzman birçok kişi makale yayınlamış, sunulan yöntemler sürekli gelişme göstermiştir.

Bu tezin konusu da bu yöntemlerden önemli olan bazı yöntemlerin tanıtımı ve bu yöntemler arasında karşılaştırma yapmaktır. Bunun için ilerdeki bölümler şu şekilde düzenlenmiştir: İkinci bölümde yöntemler tanıtılacaktır. Bu yöntemlerin çıkış fikirleri, yapıları, etkinliği anlatılacaktır. Üçüncü bölümde ise bu yöntemlerin karşılaştırması yapılacaktır. Öncelikle diğer bilimsel kaynaklardan bu performans testlerinin sonuçları aktarılacak ardından da bu yöntemlerin en son ikisi arasında karşılaştırma yapılacaktır. Dördüncü bölümde yapılan çalışmalardan elde edilen sonuçlar özetlenmekte ve gelecekte yapılabilecek çalışmalar tartışılmaktadır.

2. ÇOK BOYUTLU VERİ İNDEKSLEME YÖNTEMLERİ

Veriler bilgisayar ortamında saklanmaya başlandığından beri bu verilere ulaşım önemli bir konu haline gelmiştir. Bunun için çeşitli yöntemler geliştirilmiştir. İlk yöntemler sadece tek boyutu veriyi indekslemek amacıyla geliştirildi. Bunlar arasında ilk öne çıkan yöntem Rudolf Bayer ve Ed McCreight tarafından geliştirilen bir yöntem olan B Ağaçtı. Bu yöntem tek boyutlu verileri etkili bir biçimde indeksleyebilen bir yöntemdi. Arama ve ekleme işlemleri oldukça hızlıydı. Ardından B Ağaç biraz daha geliştirilerek B *, B + Ağaç gibi türevleri sunuldu. Bu yöntemlerle birlikte tek boyutlu veri indekslemede çok sorun kalmamıştı ancak zamanla iki ve daha fazla boyutlu verilerin indekslenmesine ihtiyaç duyuldu. Geometrik veri uygulamaları, bilgisayar destekli tasarım gibi konularda çok boyutlu indeksleme yöntemlerine ihtiyaç duyuldu. Bunun üzerine çalışmalar sonrasında R Ağaç, R* Ağaç, TV Ağaç, Piramit Tekniği gibi yöntemler ortaya çıktı. Şimdi bu yöntemlere bir göz atalım:

2.1. R Ağaç

B-Ağaç tek indeksli verileri yönetmede etkili bir yöntemdi. Fakat zaman geçtikçe ses, resim ve video tabanlı çalışmalar yoğunlaştığından ve tek boyutlu veri indeksleme yöntemleri bu verileri indekslemek için yeterli olmadığından yeni yöntemler aranmaya başlandı. İlk geliştirilen yöntemlerden birisi 1984 yılında Antonin Guttman tarafından sunulan, arama, güncelleme ve silme gibi algoritmalarıyla etkili bir yöntem olan “R-Ağaç”tı [Guttman, 1984].

2.1.1. R Ağaç indeks yapısı

Bir R-Ağacı yaprak düğümlerindeki veri objelerine işaret eden indeks kayıtlarıyla B-Ağaca benzer. Böyle bir yapı veri uzayı aramalarında sadece çok az miktarda düğümün ziyaret edilmesi için dizaynlanmıştır.

Tamamen dinamik bir indeks var olduğundan, eklemeler ve silmeler aramayla birlikte yapılmalıdır. Böylece periyodik olmayan tekrar düzenlemelere gerek kalmaz.

2.1.1.1. Kayıtlar

n-boyutlu bir veritabanı her biri erişilebilmeleri için birer eşsiz belirtece sahip bir miktar tupledan oluşur. Her yaprak düğüm şöyle bir tuple bulundurur:

(*I, yaprak-belirteci)*.

Burada *yaprak –belirteci* veritabanında nesnenin nerede saklandığını belirtir. I ise n-boyutlu bir dikdörtgendir:

$$I=(I_0,I_1,\ldots,I_{n-1})$$

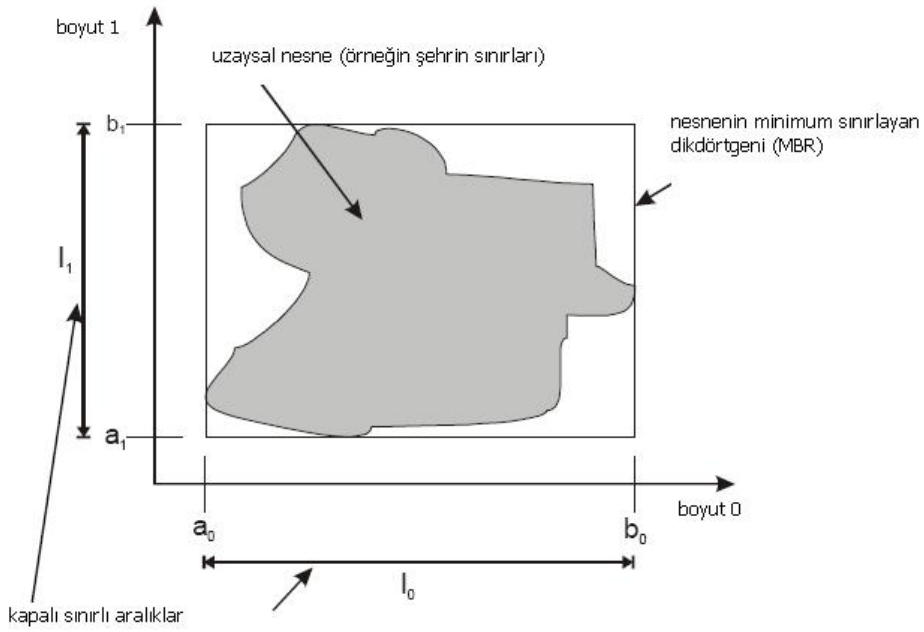
Her I_k kapalı ve sınırlı bir aralık $[a_k, b_k]$'yı temsil eder. (Şekil 2.1) Bu aralık k. boyuttaki başlama ve bitiş noktalarını gösterir. Eğer bir ya da iki nokta sonsuza eşitse bu tanımlanan nesnenin k. boyutta süresiz devam ettiğini gösterir.

Böylece her bir uzaysal nesne *sınırlı kutu I* ile temsil edilebilir. (Ayrıca buna MBR, minimum bounding rectangle: minimum sınırlayan dikdörtgen de denir) (Şekil 2.1). I nesneyi içeren en küçük dikdörtgendir.

Yaprak olmayan düğümler şu şekilde kayıtlar bulundururlar:

(I, çocuk-işaretçi)

çocuk-işaretçi R-Ağaçtaki çocuk düğümü işaret eder ve I çocuk düğümün tüm dikdörtgenlerini çevreler.



Şekil 2.1. Kapalı sınırlı aralık ve minimum sınırlayan dikdörtgen

2.1.1.2. Özellikler

Eğer M (makinenin disk sayfası ve n boyut sayısına bağlı olarak) bir düğüme sığabilecek maksimum kayıt sayısıysa, $m \leq M/2$ bir düğümdeki minimum kayıt sayısıdır. Bir R-Ağaç aşağıdaki özellikleri taşımalıdır:

- Kök olmayan her yaprak düğümü m ve M arasında indeks kaydı tutabilir.
- Yapraktaki her bir indeks kaydı (I, yaprak-belirteci) için, I tuple tarafından temsil edilen n-boyutlu veri nesnesini içeren en küçük dikdörtgendir.
- Kökte olmayan, her yaprak olmayan düğümün m ve M arasında çocuğu olabilir.
- Yaprak olmayan düğümdeki her bir kayıt (I, çocuk-işaretçi) için, I çocuk düğümü içeren en küçük dikdörtgendir.
- Kök düğüm yaprak düğüm olmadıkça en az iki çocuğa sahip olmalıdır.

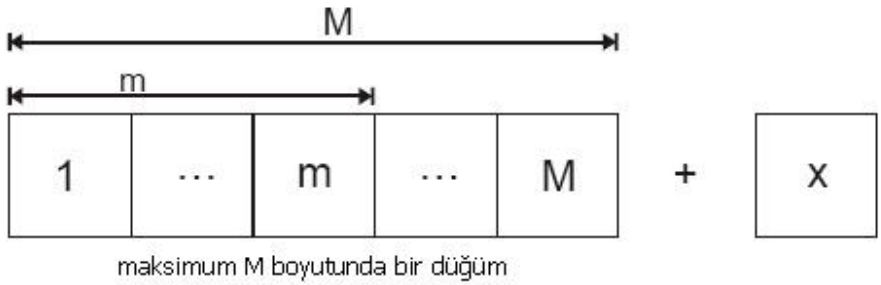
- Bütün yapraklar aynı düzeyde görünmeldir.

N indeks kaydıyla R-Ağacın yüksekliği en iyi durumda [$\log_m$ N]-1'dir, çünkü her düğüm en azından m çocuk düğüm içerir.

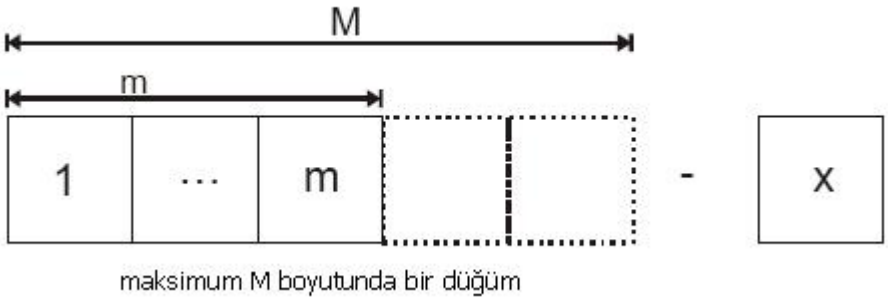
2.1.1.3. Taşma ve aşağı taşma

Eğer m çok yüksek ayarlanırsa (M'e yakın) düğüm çok yoğun dolar. Eğer bir ya da daha fazla kayıt bu düğüme yazılırsa maksimum kayıt sayısı M geçilir ve düğüm taşar. (Şekil 2.2)

Benzer şekilde m ve M arası çok geniş ayarlanırsa bir ya da daha fazla kaydın silinmesi durumunda kayıt sayısı m'in altına düşebilir. (Şekil 2.3)



Şekil 2.2. Düğüm taşması



Şekil 2.3: Düğüm aşağı taşması

2.1.1.4. M ve m nasıl ayarlanır

M ve m'in ayarlanması veritabanının verimi için çok önemlidir. M verinin saklanacağı harddiskin özelliklerinden, örneğin disk sayfa boyutu ve kapasite, çıkarılan parametrelere göre hesaplanır.

m ise veritabanı performansı için temel öğedir. Eğer veritabanı sadece arama sorguları için gerekliyse ve az güncelleme olacaksa, yüksek bir m değeri tavsiye edilir. Böylece R-Ağacın yüksekliği düşük tutulur ve arama performansı artar. Fakat bu taşma ve aşağı taşma riskini arttırır.

Diğer durumda m düşük bir değer olarak ayarlanır. Böylece veritabanı sık güncelleme ve değişikliklere karşı iyi durumda olur.

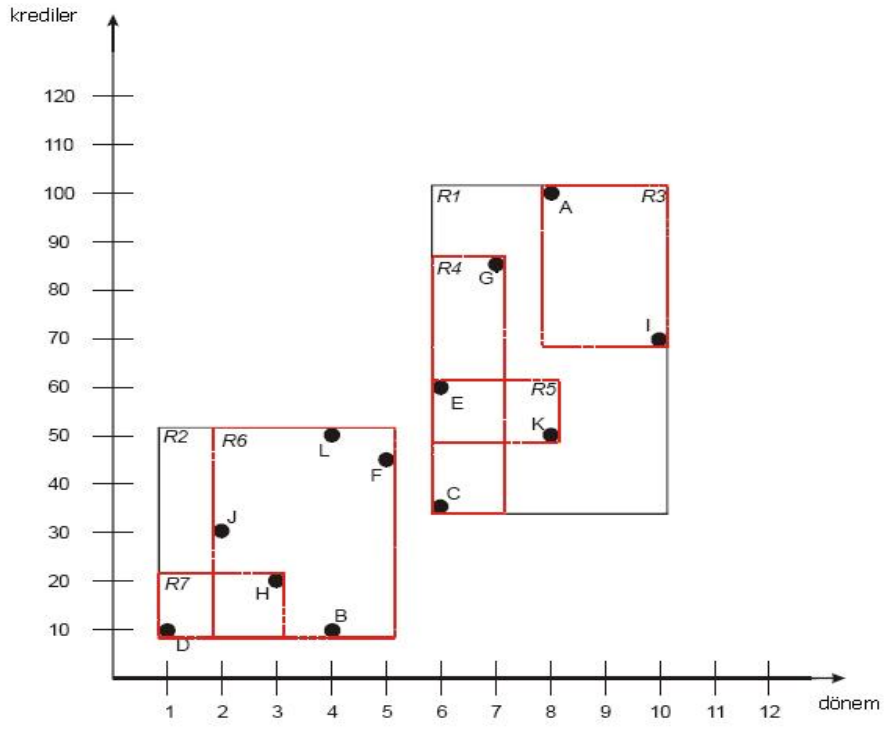
2.1.1. Algoritmalar

Guttman R-Ağaç için temel algoritmaları geliştirmiştir [Guttman,1984]. R-Ağacın bu metotları B-Ağaçtakilerin benzerleridir; sadece taşma ve aşağı taşma yönetimi verinin uzaysal yeri yüzünden farklıdır.

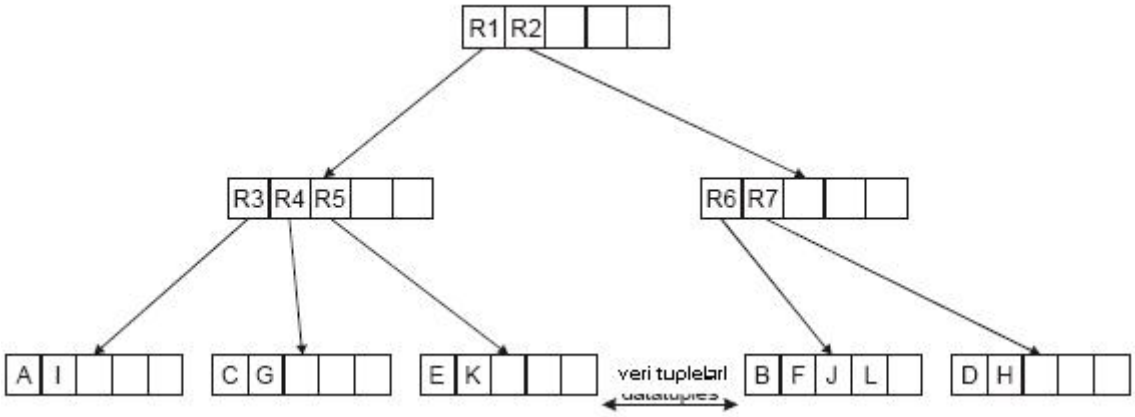
Devam eden kısımda farklı algoritmaları açıklamak amacıyla 2 boyutlu örnek bir veritabanı kullanılacaktır. Bu örnekte m=2 ve M=5 olarak seçilmiştir.

ad	dönem	kredi
A	8	100
B	4	10
C	6	35
D	1	10
E	6	40
F	5	45
G	7	85
H	3	20
I	10	70
J	2	30
K	8	50
L	4	50

Şekil 2.4. Örnek veritabanı



Şekil 2.5. Örnek veritabanının grafiksel gösterimi



Şekil 2.6. R-Ağaç yapısında örnek veritabanı

2.1.2.1. Arama

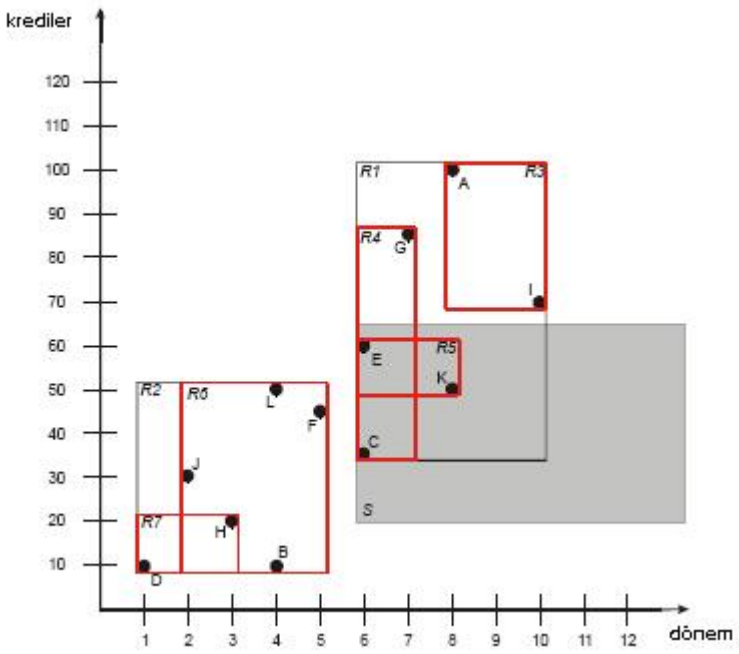
R-Ağaçta arama B-Ağaçtaki arama gibi çalışır, ağaç kökten itibaren aşağı doğru gezilir. B-Ağaçtan farklı olarak aranması gereken birkaç dikdörtgen üst üste binmiş olabilir. (Şekil 2.7) Bütün bu alt dikdörtgenler ziyaret edilmek zorunda olduğundan hiçbir iyi en kötü durum performansı garanti edilemez.

Algoritma: *Arama*

T, R-Ağacın kökü olsun. Dikdörtgenleri arama dikdörtgeni *S* ile üst üste çakışan bütün indeks kayıtları aranır.

- Eğer T bir yaprak değilse S ile üst üste çakışan ve kökü *çocuk-işaretçi* tarafından işaret edilen her çocuğa ***Arama*** yı uygula.
- Eğer T bir yapraksa, S ile üst üste çakışan bütün kayıtları sonuç kümesi olarak döndür.

Örnek veri tabanında *arama* için bir örnek:



Şekil 2.7. Örnek veritabanının grafiksel gösterimi üzerinde S arama dikdörtgeni

Bu örnekte 6. dönemde veya daha üst dönemde ve 20 ile 65 arası kredi kazanan bütün öğrenciler bulunmaya çalışılıyor.

R1 sorgu dikdörtgeni S ile üst üste biniyor, R2 ile S arasında üst üste binme olmuyor. Bu yüzden R1 aranmalıdır. Bir sonraki adımda R4 ve R5 S ile üst üste biniyor. Bu dikdörtgenlerin içinde sonuç kayıtlar bulunmaktadır. R4'ten C ve R5'ten E ve K. Sonuç kümesi {C, E, K}.

Nokta verisi için arama aynı şekilde yapılır, sadece sorgu dikdörtgeni sadece bir sorgu noktasıdır. (örneğin (6,35) C'yi sonuç olarak döndürür.)

2.1.2.2. Ekleme

Eğer veri tabanına yeni bir kayıt eklemek gerekirse, R-Ağaca yeni bir indeks kaydı eklenmelidir. Bu aynı zamanda R-Ağacın yüksekliğini arttıracak tek faktördür. Yani bir düğüm taşması varsa, düğüm bölünmek zorundadır. Bu durumda bölünme köke ulaşır ve yükseklik artar.

Algoritma: *Ekleme*

- E yeni bir kayıt olsun
- E'nin yerleşmesi gereken yaprak düğüm L'yi bulmak için **YaprakSeçme** algoritmasını kullan.
- Eğer L'de yeteri kadar yer varsa, E'yi ekle. Yoksa **DüğümBölme** algoritmasını L'ye uygula. Bu algoritma L ve L' olmak üzere E'yi ve L'nin önceki tüm kayıtlarını içeren iki düğüm döndürür.
- L üzerinde **AğaçDüzeltme** algoritmasını uygula. Eğer daha önce bir bölünme varsa bunu L' üzerinde de uygula.
- Eğer bölünme köke ulaşırsa ve kök bölünmek zorundaysa, çocukları kökün bölümünden çıkan iki düğüm olan yeni bir kök yarat.

Algoritma: *YaprakSeçme*

Yeni kayıt E için uygun bir yaprak düğüm seç.

N kök düğüm olsun.

- Eğer N yapraksa, onu döndür.
- N yaprak değilse, kayıt F_k'yı N içinde bul. Bu kayıt E'yi düğüme eklemek için dikdörtgeni en az genişlemeye maruz kalacak kayıttır. Birden fazla F_k bulunması durumunda, en küçük olanı seç.
- Bir yaprağa ulaşana kadar **YaprakSeçme**'yi F_k 'ya uygula.

Algoritma: *AğaçDüzeltme*

Yaprak düğüm L'den köke kadar tırman. Tırmanırken dikdörtgenleri ayarla ve gerekli düğüm bölünmelerini gerçekleştir.

N=L yap. Eğer L daha önceden bölünmüşse N'=L' yap.

- Eğer N kökse, sonlandır.
- P N'nin ebeveyni olsun. P'nin içinde N'nin kaydını düzenle. Böylece P N'nin içerdiği bütün dikdörtgenleri içerebilsin.

- Eğer bir bölünme meydana gelmişse P'ye N' gösteren yeni bir kayıt ekle. Eğer ebeveyn düğüm taşarsa **DüğümBölme** algoritmasını kullan.
  Ekleme için bir örnek:

(Q,10,65) şeklinde yeni bir öğrenci eklensin. *YaprakSeçme* R1'i ilk yeni düğüm olarak döndürür. Sonradan R3 kaydın ekleneceği dikdörtgen olarak seçilir ve burada bir taşma olmaz. Böylece Q R3'e eklenmiş olur. Bundan sonra *AğaçDüzeltme* R1 ve R3'ün dikdörtgenlerini günceller.

2.1.2.3. Silme

Eğer bir nesne veritabanından silinmek zorundaysa, kayıt E için ilgili indeksi bulup silmeniz gerekir.

Bu ayrıca R-Ağacın yüksekliğini azaltmanın tek yoludur.

Algoritma: *Silme*

- Kayıt E'yi içeren yaprak L'yi bulmak için ***YaprakBulma*** algoritmasını uygula. E bulunmazsa algoritmayı sonlandır.
- L'den E'yi çıkar.
- Dolu olmayan düğümleri yoğunlaştırmak için ***AğaçYoğunlaştır*** algoritmasını kullan.
- Eğer ayarlamalardan sonra kök sadece tek çocuğa sahipse, çocuğu yeni kök yap. (Ağacın yüksekliği azalır)

Algoritma: *YaprakBulma*

Kayıt E'yi içeren yaprak düğümü bul. T kök olsun,

- Eğer T yaprak değilse, dikdörtgenleri E ile üst üste binen bütün çocuklara ***YaprakBulma***'yı uygula. Eğer E bulunursa onu döndür.
- Eğer T bir yapraksa, her kaydı E ile karşılaştır ve eşleşirse T'yi döndür.

Algoritma: *AğaçYoğunlaştır*

Bu algoritma kaydın silindiği L yaprak düğümünü alır ve eğer yaprak m'den az kayda sahipse bu düğümü kaldırır.

Algoritma ağaç boyunca yukarıya doğru ilerler ve bütün dikdörtgenleri ayarlar (eğer gerekirse onları küçültür)

N=L olsun ve Q kaldırılan düğümlerin boş bir kümesi olsun.

- Eğer N kökse son adıma git. Diğer hallerde P'yi N'in ebeveyni olarak ele al.
- Eğer N m kayıttan daha az kayda sahipse (aşağı taşma), P'deki N'nin kayıtlarını kaldır ve onları Q'ya ekle.
- Eğer N'de bir aşağı taşma yoksa, dikdörtgenini (MBR) N'in diğer tüm kayıtlarını da içerecek şekilde ayarla.
- N=P yap ve ilk adıma dön.
- Q'daki bütün yaprakları *ekleme* kullanarak ağaca tekrar ekle. Q'da saklanan bütün yaprak olmayan düğümler ağacın daha üst düzeylerine eklenmelidir böylece ağaç yükseklik bakımından dengeli kalabilir.

Guttman şuna işaret etmiştir ki, prosedürlerin çoğu B-Ağaç'takiyle aynıdır. Fakat B-Ağaç'tan farklı olarak birleştirilen düğümler yerine R-Ağaç'ta tekrar ekleme vardır.

Silme için bir örnek:

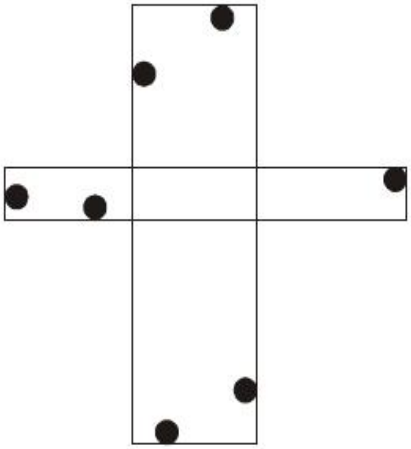
Öğrenci K veritabanından silinmek istenirse, *YaprakBulma* algoritması uygulanır ve bunun sonucunda R5 dikdörtgeni elde edilir. Kayıt K R5'ten kaldırılır ve bundan sonra R5 aşağı taşar. Ardından R5'e *AğaçYoğunlaştır* algoritması uygulanır. Bu algoritma R5'i R-Ağaç'tan kaldırır, E'yi eklemeyi kullanrak R4'e ekler ve R1'in dikdörtgenini günceller.

2.1.2.4. Bir Düğümü Bölme

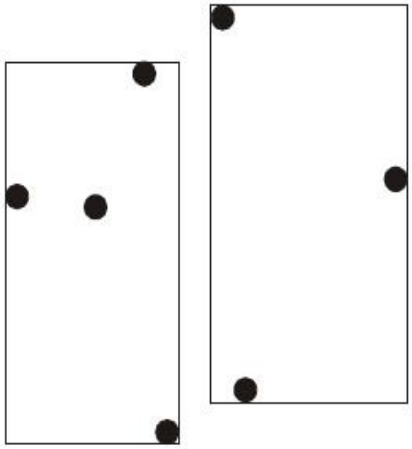
Dolu bir düğüme yeni bir kayıt eklenirken, M+1 kaydı iki düğüme bölmek şarttır.

Düğümlerin bölümünde dikkate alınacak kısım bölümden sonra oluşan iki düğümün de dikdörtgenlerinin büyüklüğünü minimize etmektir. Çünkü bu dikdörtgenler küçük olurlarsa arama durumunda sadece gerekli düğümlerin ziyaret edilme olasılığı fazla olur. Bunun nedeni daha küçük dikdörtgenlerin diğer dikdörtgenlerle üst üste binme olasılığının daha az olmasıdır.

Şekil 2.8a'da iyi bir bölünme ve Şekil 2.8b'de kötü bir bölünme örneklendirilmiştir.



Şekil 2.8a: İyi bir bölünme



Şekil 2.8b: Kötü bir bölünme

Guttman tarafından önerilen üç tane düğüm bölme algoritması vardır:

2.1.2.4.1 Ayrıntılı bölme algoritması

Bu algoritma bütün olası gruplamaları dener ve en iyisini seçer. En kalitelisidir fakat 2^{M+1} ihtimal yüzünden CPU kullanımı üssel olarak artar. Bu yüzden M'in çok yüksek olduğu (M>50) büyük veri tabanları için uygun değildir.

2.1.2.4.2 İkinci dereceden yük algoritması

Bu algoritma birbirine uzaklığı en fazla olan iki kaydı seçer ve bunları farklı düğümlere koyar. Kalan bütün elemanlar aynı şekilde düğümlere dağıtılır. Her bir düğüm için alan artışı hesaplanır. Her bir düğüm için bu düğümlerin N ve N' 'e eklenmesi için gerekli alanlar hesaplanır ve daha sonra bu iki grup arasında en büyük farkı olan düğüm başkasına eklendiğinde daha az alan gerektiren düğüme eklenir.

İkinci dereceden yük algoritması en iyi bölünmeleri sağlamaz fakat 50 kayıttan fazlasını içeren büyük veri tabanları için daha etkilidir.

Bu algoritma için kullanılan diğer iki yöntem:

KaynakSeçme: Grupların ilk iki elemanını seçer.

SonrakiniSeçme: Gruba koyulacak sonraki elemanı seçer.

2.1.2.4.3. Doğrusal yük algoritması:

Bu algoritma her bir boyut için birbirinden en uzak iki elemanı seçer ve bunları farklı düğümlere koyar. Kalan elemanlar rastgele dağıtılır. Bu yöntemin ikindi dereceden yük algoritmasından farkı sınırlandırılmış *KaynakSeçme* ve *SonrakiniSeçme* yöntemleridir.

Bu algoritma çok hızlıdır ancak arama performansı diğer yöntemlere göre daha kötüdür.

2.1.2. R Ağacın türevleri

Orijinal R-Ağacın geliştirilmesinden sonra orijinal yapı ve algoritmalara aşağıdaki bazı geliştirme ve özel yetenekler eklenmiştir.

Paketlenmiş R-Ağaç (1985): Ağacın indeks yapısı tarafından kullanılmayan alan serbest bırakılır. Genelde ekleme ve çıkarılma yapılmayacak veritabanlarında kullanılır.

R^+-Ağaç(1987): Üst üste binen alanları engellemeye çalışan yeni bir yöntemdir. Disk kullanımı daha yüksek maliyetle gerçekleşir fakat daha hızlı arama yapar.

R-Ağaç (1990):* Yapı R-Ağaçla aynıdır. R-Ağaçtan farkı bazı geliştirilmiş ekleme ve bölünme yöntemleridir.

X-Ağaç (1996): Düğüm boyutunun değişimi ile üst üste binen alanlar engellenir

2.2. R* Ağaç

R Ağaçta optimizasyon kriteri iç düğümlerdeki çevreleyen dikdörtgenlerin en az alan kaplamasıydı. Fakat R Ağaçtan sonra şu sorular soruldu: Neden bu kriter kenar uzunluğunu kısaltmak ya da üst üste binen dikdörtgenlerin alanını düşürmek değildi? Neden bellek kullanımını optimize etmek değildi? Neden bunların hepsini yapmaya çalışmıyordu? Bu sorudan yola çıkan Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider ve Bernhard Seeger adlı araştırmacılar 1990da yeni bir yöntem olarak R* Ağacı sundular [Beckmann vd., 1990]. R* Ağac diğer R Ağaç türevlerinden belirgin bir biçimde iyiydi. Ayrıca hem nokta veri hem de uzaysal veriyi tutmada başarılıydı.

2.2.1. R* Ağaç algoritmaları

R* Ağacın alt ağaç seçme, bölünme ve zorunlu yeniden ekleme gibi önemli algoritmalarına göz atalım.

2.2.1.1. Alt ağaç seçme algoritması

R* Ağaçta alt ağaç seçme algoritması R Ağaçtan farklı olarak sadece çevreleyen dikdörtgenlerin alanları toplamına göre değil aynı zamanda üst süte binen alanlara ve dikdörtgenlerin çevre uzunluklarına göre oluşturulmuştur.

Alt Ağaç Seçme Algoritması

1. N düğümünü kök olarak seç
2. N eğer bir yapraksa
   N'yi döndür

   Değilse

   N içindeki çocuk işaretçiler yaprakları işaret ediyorsa

   Dikdörtgeni en üst üste binen alan artışı gerektiren girdiyi seç. Eşitlik durumunda en az alan artışı gerektiren girdiyi seç.

   Eğer N içindeki çocuk işaretçiler yaprakları işaret etmiyorsa

   N içindeki girdilerden dikdörtgeni en az allan artışı gerektirecek olanı seç. Eşitlikleri dikdörtgeninin alanı en az olan girdiyi seçerek çöz.
3. N'yi seçilen girdinin çocuk işaretçisi tarafından işaret edilen çocuk düğüm yap ve 2. adımdan devam et.

2.2.1.2. R* Ağacın bölünmesi

R* Ağaç iyi bölünme için şu metodu kullanır: Her eksen boyunca girdiler dikdörtgenlerinin alt değerleri ve üst değerlerine göre ayrılırlar. M+1 girdinin 2 gruba her M-2m+2 dağılımı belirlenir. İlk grup (m-1)+k girdi, ikinci grup kalan girdileri içerir.

Her bir dağıtım için iyilik değerleri tespit edilir. Üç çeşit iyilik değeri vardır:

I- alan değeri :

İlk grubun çevreleyen dikdörtgeninin alanı + ikinci grubun çevreleyen dikdörtgeninin alanı

II- kenar değeri:

İlk grubun çevreleyen dikdörtgeninin kenarları toplamı + ikinci grubun çevreleyen dikdörtgeninin kenarları toplamı

III- üst üste binme değeri:

İlk grubun çevreleyen dikdörtgeninin alanı ∩ ikinci grubun çevreleyen dikdörtgeninin alanı

İyilik değeri tespiti için muhtemel yöntemler şunlardır:

- Bir eksen ya da sınıf boyunca minimum olan
- Bir eksen ya da sınıf boyunca iyilik değerleri toplamı minimum olan
- Her ikisinde de minimum olan

Elde edilen değerler bölünme eksenini bulmak veya son dağıtımı bulmak için kullanılabilir.

Bölünme Algoritması

1. Hangi bölünmenin gerçekleşeceğini bulmak için Bölünme Eksenini Seç algoritmasını çalıştır.
2. Eksen üzerinde iki gruba en iyi dağıtımı bulmak için Bölünme İndeksini Seç algoritmasını çalıştır.

3. Girdileri iki gruba dağıt.

Bölünme Eksenini Seç Algoritması

1. Her eksen için

   Girdileri dikdörtgenlerinin düşük ve daha sonra yüksek değerlerine göre sırala ve bütün dağıtımları tespit et. Farklı dağıtımlar için bütün kenarların toplamını temsil eden S'yi hesapla
2. Minimum S'e sahip ekseni bölünme ekseni olarak seç

Bölünme İndeksini Seç Algoritması

1. Seçilen bölünme ekseni boyunca minimum üst üste binme değerine sahip dağıtımı seç. Eşitlik durumunda minimum alana bak.

Bölünme algoritması m=%20, %30, %40 ve %45 M olarak seçilerek yapılan deneyler sonucunda m=%40 M alındığında en iyi performans sağlanmıştır.

2.2.1.3. Zorunlu Yeniden Ekleme

R Ağaçta girdilerin farklı sıralarda eklenmesi farklı yapıda ağaçlar oluşturur [Gutmann, 1984]. Bu yüzden R Ağaçta eski girdilerden dolayı sıkıntılar olabilir. İlk girilen veri dikdörtgenleri sonradan dizin dikdörtgenleri olabilir ancak bunlar iyi bir erişim performansı sağlayacak yapıda olmayabilirler. Bu yapıyı değiştirecek tek şey bölünmelerdir. Ancak onlar da lokal olarak gerçekleştiğinden yeterli değildir. Daha güçlü bir yeniden düzenlemeye ihtiyaç vardır.

Bu problem, minimum girdi taşıma kapasitesi altında bulunan düğümler için yapılan işlem bu girdileri ebeveyn düğümlere dağıtmak olsaydı çok daha kötü bir hal alabilirdi. Ancak bu durumda R Ağaç bu girdileri tekrar girmeyi dener ve bu da girdilerin farklı düğümlere atanmasına neden olur.

Eski veri dikdörtgenlerini silip tekrar eklemek beklenildiği gibi erişim performansını arttırabilir. Bunun için yapılan bir deneyde bir R Ağaca 2000 veri

girilmiş ardından ilk 1000 veri silinip tekrar girilmiş. Bu işlem ağacın erişim performansını %20 den %50 lere çıkarabilmiştir. Bu R Ağaçlar için basit ve etkili bir yöntemdir. Ancak bu statik bir yöntemdir ve sürekli dinamik veri girişi yapılan bir yerde uygulamak zordur.

R* Ağaç yeniden düzenleme için girdileri bir ekleme rutini içerisinde zorla tekrar eklemeye sokar. Bunun için algoritma şöyledir:

Veri Ekleme Algoritması

1. Ekleme algoritmasını yeni bir veri dikdörtgeni eklemek için kökten itibaren başlat

Ekleme Algoritması

1. Düzeyi parametre olarak kullanarak yeni girdi E için uygun bir N düğümü bulmak amacıyla Alt Ağaç Seçme algoritmasını çalıştır.
2. Eğer N, M girdiden daha azına sahipse, E'yi yerleştir.Eğer N, M girdiye sahipse N'in düzeyini parametre olarak kullanarak Taşma İşlemi algoritmasını çalıştır.
3. Eğer Taşma İşlemi Algoritması çağırılmış ve bir bölüme olmuşsa Taşma İşlemi algoritmasını üste doğru yay (gerekirse). Eğer taşma işlemi algoritması kökte bir bölünmeye neden olursa, yeni bir kök yarat.
4. Ekleme yolundaki bütün çevreleyen dikdörtgenleri çevreledikleri çocuk dikdörtgenlerini çevreleyen minimum dikdörtgenler olarak ayarla.

Taşma İşlemi Algoritması

1. Eğer düzey kök düzey değilse ve bir veri dikdörtgeni eklerken bu düzeyde Taşma İşlemi Algoritmasının ilk çağrılışıysa

   Tekrar Eklemeyi çağır

   Değilse

   Bölünmeyi çağır

Tekrar Ekleme Algoritması

1. N düğümünün her M+1 dikdörtgeni için dikdörtgenlerinin merkezi ile N düğümünün çevreleyen dikdörtgeninin merkezi arasındaki uzaklığı hesapla.
2. Girdileri 1. adımda hesaplanan uzaklığa göre azalan sırada sırala
3. N'den ilk p girdiyi çıkar ve N'nin çevreleyen dikdörtgenini tekrar ayarla.
4. 2. adımdaki sıralamada maksimum uzaklıktan ya da minimum uzaklıktan başlayarak girdileri tekrar eklemek için Ekleme Algoritmasını çağır.

Özetle:

- Zorunlu Tekrar Ekleme girdileri komşu düğümlerle değiştirir ve böylece üst üste binmeyi azaltır.
- Bir yan etki olarak bellek kullanımı gelişir.
- Daha fazla yeniden yapılanmayla daha az bölünme gerçekleşir.

Zorunlu ekleme sonrasında doğal olarak daha fazla ekleme algoritması çalıştığından cpu kullanma oranı artacaktır. Ancak daha az bölünme gerçekleştiğinden bu oran çok fazla artmayacaktır. Yapılan deneylerde bu oranın %4 arttığı tespit edilmiştir.

2.3. TV Ağaç

Zaman geçtikçe uygulamalarda çok boyutlu veriler arasında benzerlik arama ihtiyaçları arttı. Bunun için R Ağaç türevi yöntemler yetersiz kalıyordu. Çünkü bu yöntemler daha çok 2 boyutlu ya da 3 boyutlu uzay için tasarlanmıştı. Bu yöntemler daha fazla boyutlu uzaylar için de düzenlenebilirdi ancak bu sefer de çok etkili olmuyorlardı. Bunun üzerine 1995 yılında King-lp Lin, H.V. Jagadish ve Christos Faloutsos yeni bir yöntem sundular [Lin vd.,1995}. Bu yöntemin çıkış noktası değişken sayıda boyutu indeksleme için kullanmak ve bunları o anki ağaç düzeyinde indeksleme için ayarlama fikridir. Bu yüzden ağacın köke yakın kısımları indeksleme için daha az boyut bilgisinden yararlanırken daha alt kısımlar veri ayrımı yapmak için daha fazla boyut bilgisine ihtiyaç duymaktadır. Özellik vektörlerinin teleskop gibi genişleyip küçülmesinden de bu yönteme Teleskopik Vektör Ağacı yani TV Ağaç denmiştir.

Özellik çıkarımı fonksiyonları nesnelerin özellik uzayındaki bir noktayla eşleşmesini sağlar. Bu nokta da bir uzaysal erişim yöntemi ile saklanmak zorundadır.

Çeşitli sorgulama yöntemleri vardır. Bunları şu şekilde sınıflandırabiliriz:

- Tam Eşleşme Sorguları: Verilen bir sorgu nesnesine tamı tamına uyan verileri bulmak için kullanılır. Bir veritabanında istenen veri var mı yok mu gibi sorgular buna örnek gösterilebilir.
- Aralık Sorguları: Veritabanında verilen bir nesne için bu nesneye belirli uzaklık içindeki bütün verileri bulmak için kullanılır. Benzerlik sorguları da bu kategori içinde yer alır. Örnek verecek olursak: Bir evin en fazla 2 km uzağındaki bütün komşu evleri bulmak ya da aslana benzeyen bütün şekilleri bulmak gibi.
- En Yakın Komşu Sorguları: Verilen bir sorgu öğesi için veritabanındaki bu öğeye en yakın ya da en benzer öğeleri bulmak için kullanılır. Yine ev örneğinden gidersek belirli bir eve en yakın komşu evi bulmak ya da en yakın k tane komşuyu bulmak gibi.
- Bütün Çift Sorguları: Verilen bir nesneler kümesinde birbirine a uzaklığındaki bütün çiftleri bulmak ya da k tane en yakın çifti bulmak gibi sorular için kullanılır. Örneğin birbirine 1 km uzaklıktaki bütün ev çiftleri bulmak.
- Alt Model Eşleştirme: Nesnelere bir bütün olarak bakmak yerine nesnelerin içinden tanıma uyan alt modelleri bulmak için kullanılır. Örnek olarak bütün röntgen filmi resimleri içinde tümör benzeri bir alt modeli gösteren resimleri bulmak gibi

Birçok çok boyutlu indeksleme yöntemi boyut artışına bağlı olarak üssel biçimde performans kaybına uğrar. En sonunda da doğrusal aramaya dönerler. Doğrusal dörtlü ağaçlar performans sorgu alanını çevreleyen hiper yüzeyle doğru orantılıdır ve hiper yüzey de boyut artışı ile orantılı olarak üssel biçimde büyür. Grid dosyalarında da benzer problemler vardır. Boyutsallık artışı ile üssel olarak büyüyen bir dizin sorunu vardır. R Ağaç [Guttman, 1984] ve türevlerinde eğer bir özellik vektörü bir disk sayfasından daha fazla depolama alanı gerektirirse bu durumda çıkış yelpazesi 1'e düşecektir, yani yapı bağlı liste gibi olacaktır.

Benzer problemler en yakın komşu sorguları için üretilmiş yöntemlerde de vardır: Voronoi diyagramları 3 ten fazla boyut için çalışmaz, Friedman yöntemi boyut sayısı 9 üstüne çıktığında doğrusal arama olarak çalışır ve Bentley'in spiral arama yönteminin karmaşıklığı boyutsallıkla birlikte üssel olarak artar.

2.3.1. TV Ağaç yönteminin arkasındaki sezgi

Boyutsallık sorununa karşı TV Ağaç yönteminin öne sürdüğü çözüm özellik vektörlerinin dinamik biçimde kısaltılıp, uzatılması ve böylece nesneler arasında ayrım yapabilmektir. Bu aslında insanların sezgisel bir sınıflama biçimidir. Örneğin zoolojide önce birkaç özelliği kullanan ana sınıflar vardır (omurgalılar, omurgasızlar) Bu sınıflandırmada alt sınıflara gittikçe canlıları ayırt etmek için daha fazla özelliklerin kullanılır. (sıcakkanlı ve soğukkanlı gibi)

Özellik vektörünün kısaltılması ve uzatılması dışında yöntem diğer ağaç yöntemleri gibidir. Nesneler yaprak düğümlerde minimum çevreleyen dikdörtgenlerle birlikte tutulurlar. Ebeveyn düğümler de köke göre gruplanmıştır.

Diğer ağaç yöntemleriyle karşılaştırılınca TV Ağaç üt seviyelerde daha az özellik kullanarak daha büyük bir çıkış yelpazesi sağlar ve böylece gereksiz özellikler de kullanılmamış olur.

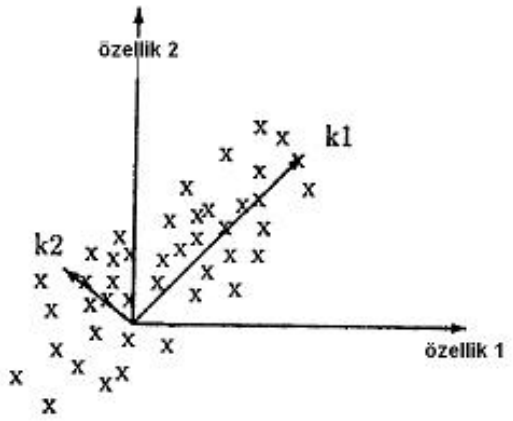
Ağaca daha fazla nesne eklendiğinde nesneleri ayırt etmek için yeni özellikler tanıtılabiliyor. Burada kilit nokta yeni özelliklerin sadece ihtiyaç olduğunda tanıtılıyor oluşudur. Bu şekilde boyutsallık sorunu biraz yumuşatılır.

2.3.1.1. Teleskopik fonksiyon

Önerilen yöntem özellikleri asimetrik olarak değerlendirir, ilk birkaç özelliği diğer özelliklere tercih eder. Benzerlik sorgularında daha iyi sıralanmış bir vektör daha

odaklanmış bir sonuç döndürür. Eşleşme sorgusunda da iyi bir sıra seçimi performansı arttırır. Yine benzer şekilde kötü sıralama birçok yanlış alarma neden olur ve performansı düşürür.

Birçok uygulamada özellik vektörü dönüştürmek iyi bir sıralama sağlar. Karhunen Lowe (KL) [Fukunaga, 1990] dönüşümü de özellikleri önemine göre sıralayan bir yöntemdir: Bu dönüşüm her birinde d özellik bulunan n tane vektör kümesinde, eskilerin lineer kombinasyonu olan ve ayrımcı güçlere göre sıralanmış d tane yeni özellik döndürür. Şekil 2.9' da k1 ve k2 KL dönüşümünün sonuçlarıdır.



Şekil 2.9. Karhunen Lowe dönüşümü

Eğer veriler önceden biliniyorsa KL dönüşümü en uygun olanıdır. Güncelleme işleminin sık sık yapılmadığı veri setleri için kullanılabilir. Aynı zamanda verilere ait çok büyük sayıda örnek varken ve diğer yeni veriler de bu istatistiğe uygun ise kullanılabilir.

Dinamik verilerin dönüşümü için Ayrık Kosinüs Dönüşümü (DCT) [Wallace, 1991], Ayrık Fourier Dönüşümü (DFT), Hadamard Dönüşümü [Hamming, 1977] ve Küçük Dalga Dönüşümü [Ruskai vd., 1992] kullanılabilir. Ancak veri eğer belirli bir istatistik modeldeyse birçok veri-bağımsız dönüşüm KL gibi performans gösterir. Özellikler birbirine çok bağlıysa DCT çok iyi bir seçimdir. Bu duruma örnek olarak 2 boyutlu resimlerde yakın piksellerdeki renklerin benzerliği verilebilir. JPEG resim

sıkıştırma standardı bu olguyu kullanır. Böylece resmin kalitesinde ihmal edilebilir bir kayıpla iyi bir sıkıştırma oranı sağlar.

Genel olarak eğer verinin istatistiksel özellikleri iyi anlaşılmışsa veri-bağımsız dönüşümler birçok durumda yakın sonuçlar, önemlerine göre sıralanmış özellikler, döndürecektir. TV Ağaca ortogonal bir dönüşüm verilere erişimi hızlandırır.

2.3.1.2. Çevreleyen alanın şekli

Nokta gruplarının minimum çevreleyen alanlarının (MÇA) şekilleri uygulamaya uygun olarak seçilebilir. Bu şekiller hiperküp, hiperdikdörtgen ya da küre olabilir. Çevreleyen alanı en kolay temsil etme şekli küredir. Çünkü temsil etmek için kürenin sadece merkezi ve yarıçapı yeterlidir. Merkezden uzaklığı Öklid uzaklığına göre r yarıçapından küçük noktalar bu çevreleyen alan içindedirler.

TV Ağaçta algoritmalar bütün şekiller için uygundur. Tek farkları her şekil için minimum çevreleyen alanın hesaplanma şeklidir. Küre dışındaki şekiller için hesaplama şeklinde küçük değişiklikler yapmak gerekir. Bu şekillerin küreden farkı küredeki gibi her boyutta tek bir yarıçap değeri yerine her boyutta farklı yarıçap değeri olabilmesidir.

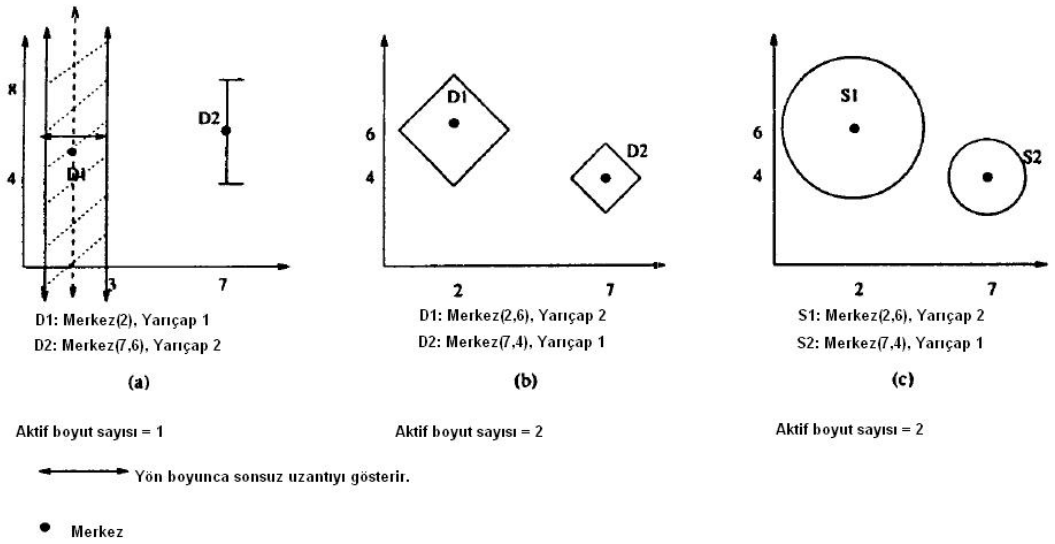
2.3.2. TV Ağaç

TV Ağacın düğüm yapısı ve ağaç yapısına bir göz atalım.

2.3.2.1. Düğüm yapısı

TV Ağaçta her düğüm çocuklarının minimum çevreleyen alanlarını(küre) temsil eder. Her alan da nesneleri temsil eden teleskopik vektör tarafında belirlenen bir vektör

olan merkez ve yarıçapla temsil edilir. Teleskopik Minimum Çevreleyen Alan için "TMÇA" denilebilir.



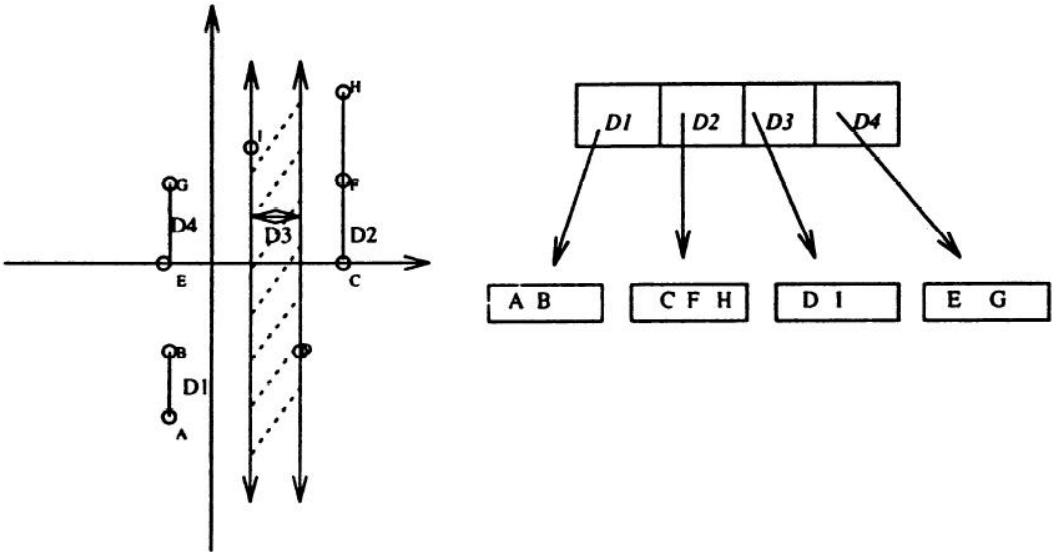
Şekil 2.10. Çeşitli şekillerde TMÇA örnekleri

Şekil 2.10a da D2 1 pasif (ik boyut) ve 1 aktif boyuta sahiptir. D1'in de bir tane aktif boyutu vardır. D1'in boyutsallığı 1 ve D2'nin 2'dir.

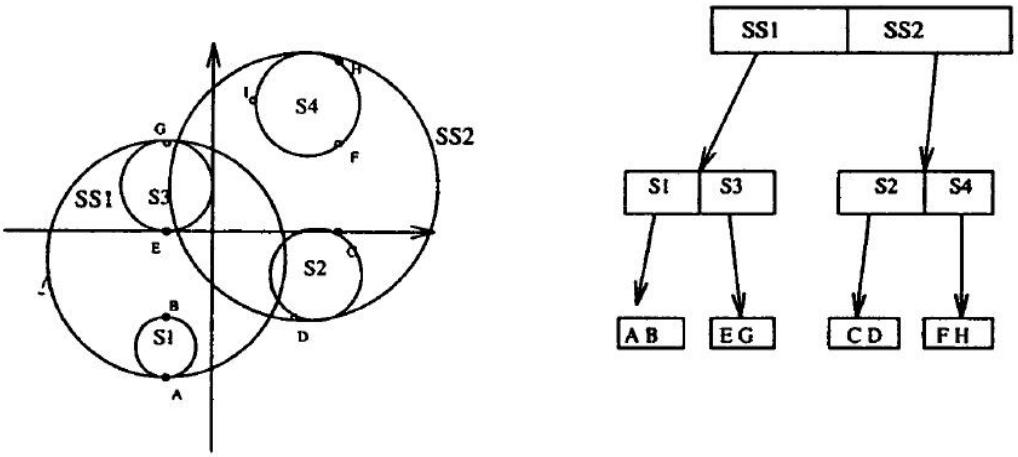
Bu bilgiye ihtiyaç vardır çünkü ağaç büyüdükçe düğümler ilk k boyutta aynı değerleri alacaktır. Böylece bu boyutlar pasif olacaktır çünkü bu boyutlar nesneler arasında ayrım yapmaya yaramaz.

2.3.2.2. Ağaç yapısı

Tv Ağacın R Ağaçla bazı benzerlikleri bulunmaktadır. Her düğümde dallar vardır ve her dal bir tane çevrelenen alanı belirten TMÇA içerir. TMÇA'lar üst üste binebilir ve her düğüm bir dik sayfası kadar yer kaplar.



Şekil 2.11. TV-1 Ağaç örneği (karolarla)



Şekil 2.12. TV-2 Ağaç örneği (kürelerle)

TV-1 Ağaç'ta aktif boyut sayısı 1'dir, böylece karolar sadece bir boyut boyunca yayılabiliyorlar. Sonuç olarak şekiller keskin hatlar, dikdörtgen biçiminde bloklardan oluşuyor. TV-2 Ağaçta TMÇA iki boyutlu çemberlere benziyor.

Her aşamada aktif boyut sayısı özellikle belirtilir. Bazen ağacın birden fazla düzeyi aynı aktif boyutları kullanabilir. Ancak genelde her yeni düzeyde yeni bir aktif boyut kullanılır.

2.3.2.3. Algoritmalar

Arama: Hem tam benzerlik hem de aralık sorgularında algoritma kökten başlar ve kesişen her dal gezilir. TV Ağaç üst üste binmeye izin verdiğinden iki daldan birden ilerleme olabilir.

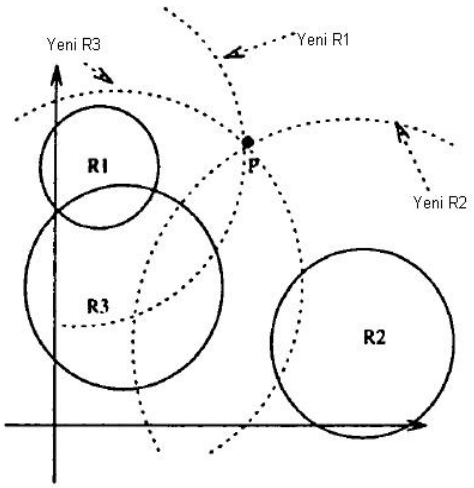
Mekansal birleştirme sorguları da uygulanabilir. Bu tür sorgular birbirine yakın nokta çiftlerini gerektirir. Bunun için gereksiz uzak dalları atan bir yinelenen algoritma kullanılabilir.

Benzer şekilde en yakın komşu sorguları da dallanma ve sıçrama algoritmasıyla yapılabilir. Algoritma şu şekilde çalışır: Verilen bir nokta için üst düzey dallar test edilir. Uzaklığa göre üst ve alt sınırlar hesaplanır. En yakın dala inilir. Çok uzaktaki dallar göz ardı edilir.

Ekleme: Yeni bir nesne eklerken ağaç gezilir ve her bölümde uygun dal bulunur. Yaprak düzeyine gelinince nesne yaprak düğüme eklenir. Taşmalar düğümü bölmekle halledilir. İkinci bir yöntem de düğümün bazı elemanlarını tekrar eklemektir. Ekleme işleminden sonra TMÇA düğümdeki bütün elemanları kapsayacak şekilde ayarlanır. Bu işlem yarıçap uzatılıp, kısaltılarak yapılır.

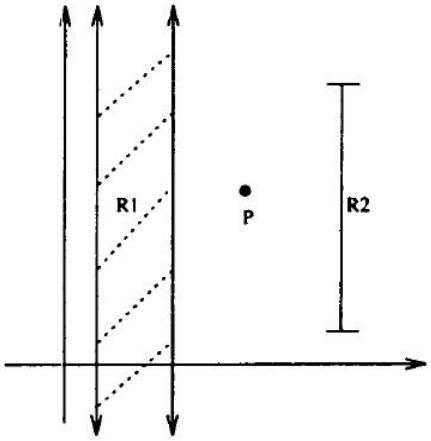
Dal seçme işlemi N düğümündeki bütün dalları sınar ve e elemanını kapsayacak en uygun dalı döndürür. Bu işlemdeki kriterler şöyledir:

1. Düğüm içindeki üst üste binmelerde minimum artış
2. Boyutsallıkta minimum azalma (örneğin mümkün olduğunca çok koordinatı nesneye uyan TMÇA'yı seçmek. Böylece yeni nesneyi TMÇA'nın merkezini mümkün olduğunca az daraltarak yerine yerleştirebilir. Şekil 2.13b'de R2'deki daralma yerine R1 seçilmiştir.)
3. Yarıçapta minimum artış (Şekil 2.13c)
4. TMÇA merkezinden noktaya olan minimum uzaklık (Şekil 2.13d'de diğer iki kriterde seçim yapılamayıp bu kritere göre seçim yapılmıştır.)



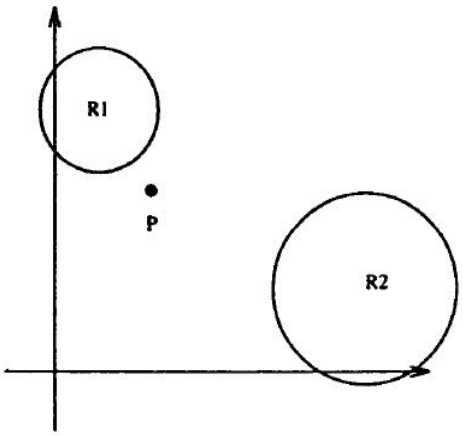
(a)

(a) R1 seçilmiştir, çünkü R2 ve R3'ü genişletmek üst üste binmeyi arttıracaktır.



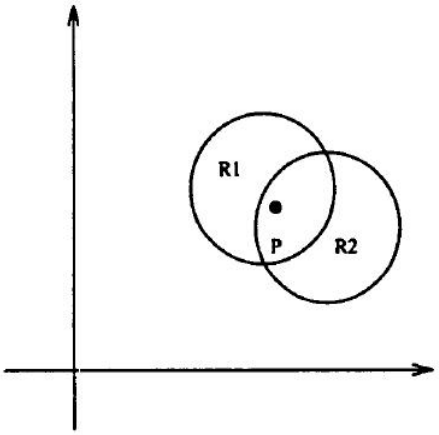
(b)

(b) R2 yerine R1 seçilmiştir çünkü R2'nin seçilmesi boyutsallığında azalmaya yol açacaktır.



(c)

(c) R1 seçilmiştir, çünkü R1'in yarıçapı daha küçük olacaktır.



(d)

(d) R2 yerine R1 seçilmiştir çünkü R1'in merkezi noktaya daha yakındır.

Şekil 2.13. Dal seçme kriterleri

TV Ağaçta taşmalar sadece dolu bir düğüme yeni bir eleman ekleneceği zaman olmaz. Aynı zamanda bir teleskopik vektör genişletilirken de taşma meydana gelebilir. Taşmalar düğümde bölünmeyle giderilebilir. Bir başka yöntem de düğümün bazı elemanlarını tekrar eklemektir. Bu daha düzenli bir ağaç oluşmasını sağlar.

Bir taşma olduğunda şu şema izlenebilir:

- Yaprak düğüm için, eğer o anki ekleme işleminde düğüm ilk kez bölünüyorsa düğüme ait elemanların sayısının daha önceden belirlenmiş bir yüzdesi (p_{ri}) kadar eleman tekrar eklenir. Diğer durumda ise düğüm ikiye bölünür. Tekrar ekleme için seçilecek elemanlar için değişik kurallar tanımlanabilir. Bunlardan birisi de merkezden en uzak elemanları tekrar eklemektir.
- İç düğümlerde ise bölünme gerçekleşir. Bölünme yukarıya doğru yayılır.

Ekleme Algoritması

(Kökü N olan ağaca e elemanını ekleme)

1. Dal Seçme algoritmasını takip edilecek en iyi dalı bulmak için kullan, yaprak düğüm L'yi bulana kadar ağaçta aşağıya inmeye devam et.
2. Elemanı L düğümüne ekle
3. Eğer L taşarsa

   Eğer ilk kez taşma oluyorsa ekleme sırasında

   L'nin merkezine en uzak ilk p_{ri} elemanı seçip en üstten tekrar ekle

   Yoksa

   Yaprak düğümü iki düğüme böl

4. TMÇA'ları güncelle

   Eğer iç düğümde bir taşma olursa onları da böl

Bölme: Bölmenin amacı daha iyi yönetim ve uzay kullanımı sağlamak için TMÇA'lari iki gruba dağıtmaktır. Bubu yapmanın birkaç yolu vardır. Bunlardan birisi birbirine benzeyen özellik vektörlü elemanları kümeleyerek bölmedir:

Kümeleyerek Bölme

(N bir ara düğüm olsun. Algoritma yaprak düğümler için de aynıdır)

1. B1 ve B2 dallarını birbirinden en farklı TMÇA'lara sahip iki tohum olarak seç. B1 ve B2'nin grupları R1 ve R2 olsun
2. Geriye kalan tüm dal B'ler için:
   B'yi Dal Seçme algoritmasını kullanarak R1 ya da R2'ye ekle

Bir başka bölme yöntemi de sıralayarak bölmektir. Bu durumda vektörler bir şekilde sıralanır ve en uygun bölme seçilir. Kriterler şöyledir:

1. Oluşacak iki TMÇA'nın yarı çapları toplamının en küçük olması
2. Oluşacak TMÇA'lerın yarı çapları toplamından ile merkezleri arasındaki farkın çıkarılmasından çıkan sayının en küçük olması

Özetle ilk önce TMÇA'nın kapladığı alan minimize edilmeye çalışılır sonra da üst üste binen alan minimize edilmeye çalışılır.

Sıralayarak Bölme

(N bir ara düğüm olsun, aynısı yaprak düğümler için de geçerlidir.)

(min_fill de bir düğümün işgal edebileceği minimum yüzde olsun(byte))

1. Dalların TMÇA'larının merkezlerini satır öncelikli olarak yükselen yönde sırala.
2. Sıralamada en iyi kırılım noktasını bul, iki tane alt küme yarat: (a) bir alt kümenin çok küçük olması durumunu göz ardı et(< min_fill bytes) (b)kalan durumlar içinde TMÇA'larının yarıçapları toplamı en küçük olan kırılım noktasını seç. Beraberlik durumunda (TMÇA yarı çap toplamı – merkezler arasındaki mesafe) en küçük olmasına bak.
3. Eğer şart (a)'dan dolayı hiçbir aday kalmıyorsa dalları byte büyüklüklerine göre sırala ve (a) şartını geçerek yukarıdaki adımı uygula.

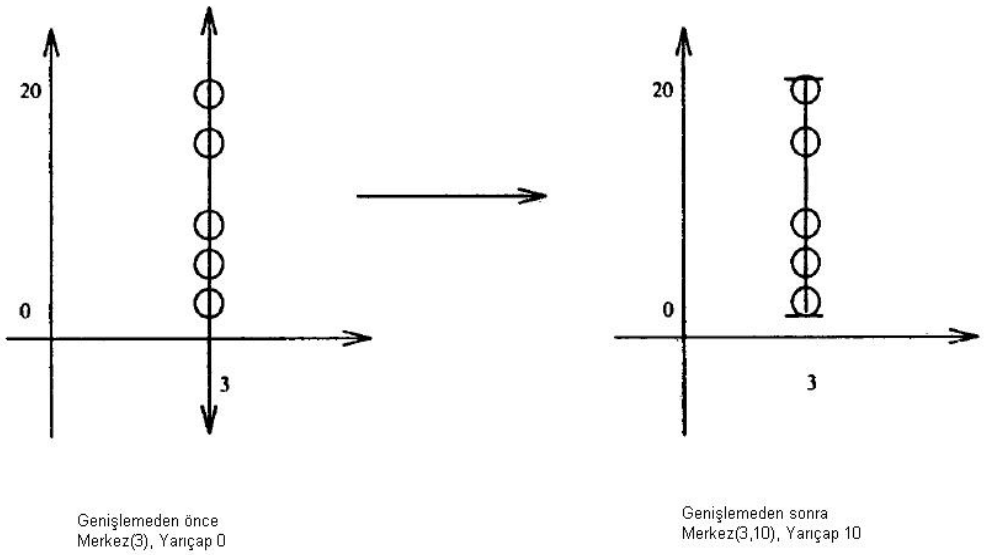
Silme

Silme bir aşağı taşma yapmıyorsa doğrudan yapılır. Aşağı taşma durumunda düğümün geri kalan dalları silinir ve tekrar eklenir. Bu yukarıya doğru yayılır.

Genişletme ve Kısaltma

Bu kısım algoritmanın önemli bir kısmıdır. Genişletme bölünme ve tekrar eklemede yapılır. Bir düğümün içindeki nesneler tekrar dağıtılacaksa geri kalan nesnelerin ilk birkaç aktif boyutta aynı değere sahip olması durumu oluşabilir. Bu yüzden yeni TMÇA hesaplanırken genişletme meydana gelir. (Genişleme örneği şekil 2.14te verilmiştir. Genişlemeden sonra karolar sadece y'de devam ediyorlar.)

Kısaltma ise ekleme sırasında yapılır. Nesne pasif boyutları kendilerinkiyle uyuşmayan bir TMÇA'ya eklendiğinde yeni TMÇA'da vektör kısaltılır. Bu da daha az boyutsallığı sağlar.



Şekil 2.14. TMÇA'yı genişletme

2.4. X Ağaç

X Ağaç [Berchtold vd., 1996], R Ağaçtan esinlenilerek oluşturulmuştur. R Ağaç tabanlı bir indeksleme yöntemi kullanmanın nedeni sadece nokta şeklindeki bir veriyi değil uzay üzerinde uzantıları olan bir nesneyi de indeksleyebilmektir. R Ağaç tabanlı

algoritmalar uzaysal veriyi indeksleyebilmek için bu veriyi nokta haline dönüştürmek ihtiyacı duymazlar ve bu yüzden daha iyi bir gruplama yapabilirler.

X Ağacın ana fikri dizinlerde üst üste binen alanlardan dizini yüksek boyutlu uzaya göre yeniden organize ederek sakınmaktır. Sonucunda yüksek bir üst üste binen alan elde edilecek bölünmelerden kaçınılır. Bunun yerine dizin düğümleri genişler. Bu düğümlere "süper düğüm" denir. Süper düğümler geniş olabilir ve bu düğümler yeterince büyük olursa düğümler içinde doğrusal arama bir problem olabilir. Fakat diğer seçenek seçiciliği azaltan ve birden fazla düğümde aramaya yol açan ve sayfa erişimini arttıran bölünmelere izin vermektir.

2.4.1. X Ağaç yapısı

X Ağacın genel yapısı Şekil 2.15'te gösterilmiştir. En alt düğümler nesneleri işaret eden MÇDlerden (minimum çevreleyen dikdörtgen), diğer düzey düğümler ise alt MÇDleri işaret eden MÇDlerden oluşur. X Ağaçta üç çeşit düğüm bulunur: veri düğümleri, normal düğümler ve süper düğümler. Diğer düğümlerden farklı olarak süper düğümler daha geniş ve değişken boyuttadırlar. Süper düğümlerin amacı etkisiz dizin yapısı meydana gelecek bölünmelerden sakınmaktır. Süper düğümlerin sayısına ve boyutuna bağlı olarak boyut sayısı artımıyla artan sayfa erişimi sayısı boyut sayısı arttıkça azalır.

Süper düğümler ekleme sırasında eğer üst üste binmeden kaçınmak için yapılabilecek başka bir şey yoksa yaratılırlar. Birçok durumda örtüşen minimal bölünme ekseni seçilerek süper düğümlerin yaratılması ve genişlemesi engellenir. Bu bölünme ekseninin hızlı tespiti için dizin düğümlerinde bulunan ek bir bilgi gereklidir. Süper düğümler eğer yeterli bellek varsa ana bellekte tutulurlar. Süper düğümlerin depolanması normal dizin düğümlerinden biraz daha fazla kaynak harcar.

X Ağaçta iki tane ilginç durum vardır: (1) Dizin düğümlerinin hiçbirinin süper düğüm olmaması (2) dizinin tek bir süper düğümden oluşması (kök) İlk durumda X Ağacın hiyerarşik yapısı aynı R Ağaç gibidir [Guttman, 1984]. Bu durum düşük boyutlu

ve üst üste binmeyen verilerde meydana gelir. İkinci durum ise ağacın bir süper düğüm olan kökten oluşmasıdır. Bu düğüm R Ağacın hepsini tutar şekildedir. Dolayısıyla burada arama doğrusal aramaya dönüşür. Bu durumda yüksek boyutlu ve üst üste binen verilerin çokluğunda oluşur. Böyle durumlarda dizin tamamen taranmalıdır. Bu iki durum da bu yapının iki uç durumudur. Tamamen hiyerarşik yapıda dizinin yüksekliği ve genişliği r Ağacınkine benzer. Kök süper düğüm durumunda ise dizinin boyutu doğrusal olarak uzayın boyutuna bağlı olarak değişir.

2.4.2. Algoritmalar

X Ağacın en önemli algoritması ekleme algoritmasıdır. Ekleme algoritması hiyerarşik ve lineer yapı kombinasyonuyla X Ağacın yapısını oluşturur. Asıl amaç üst üste binmeyi sağlayan bölünmelerden kaçınmaktır. Ekleme algoritması ilk olarak nesnenin ekleneceği MÇD'ye karar verir. Eğer bir bölünme olmayacaksa sadece MÇD'nin boyutu güncellenir. Bölünme olması durumunda önce bölünmenin nasıl yapılacağını tespit eden algoritma çağrılır. Eğer bölünme sonucu yüksek bir üst üste binme oluyorsa bölme algoritması minimal üst üste binen bölünmeyi bulmaya çalışır. Bunu yaparken düğümlerin bölünme geçmişlerinden yararlanır. Nokta veri için her zaman üst üste binmeye yol açmayan bir bölünme şekli vardır. Minimal üst üste binmeden ortaya çıkan düğümler gerekli eleman sayısına ulaşmayabilirler. Eğer böyle olursa bölünme algoritması bir bölünme yapmadan sonlanır. Bu durumda normal düğüm genişleyerek süper düğüm oluşturur. Genişliği normal düğümün iki katı olur. Eğer bu olay bir süper düğüm için gerçekleşmişse süper düğüme ek bir blok daha eklenir. Süper düğümler hiyerarşik yapı için uygun bir bölünme bulunamadığında kullanılırlar.

```
int X_DirectoryNode::insert(DataObject obj, X_Node **new_node)
{
    SET_OF_MBR *s1, *s2;
    X_Node *follow, *new_son;
    int return_value;

    follow = choose_subtree(obj);       // choose a son node to insert obj into
    return_value = follow->insert(obj, &new_son); // insert obj into subtree
    update_mbr(follow->calc_mbr());     // update MBR of old son node

    if (return_value == SPLIT){
        add_mbr(new_son->calc_mbr());   // insert mbr of new son node into current node

        if (num_of_mbrs() > CAPACITY){  // overflow occurs
            if (split(mbrs, s1, s2) == TRUE){
                // topological or overlap-minimal split was successfull
                set_mbrs(s1);
                *new_node = new X_DirectoryNode(s2);

                return SPLIT;
            }
            else // there is no good split
            {
                *new_node = new X_SuperNode();
                (*new_node)->set_mbrs(mbrs);

                return SUPERNODE;
            }
        }
    } else if (return_value == SUPERNODE){ // node 'follow' becomes a supernode
        remove_son(follow);
        insert_son(new_son);
    }

    return NO_SPLIT;
}
```

Şekil 2.15. Dizin Düğümleri için X Ağaç Ekleme Algoritması

X ağacı sorgulama algoritmaları R* Ağacınkilere [Beckmann vd., 1990] çok benzer. Sadece süper düğümlere erişmek için küçük değişiklikler vardır. Silme ve güncelleme algoritmaları da R* Ağacınkilerle aynıdır. Tek farklı durum bir süper düğümün belirli seviyeden aşağı taşmasıdır. Eğer bu durumda süper düğüm iki bloktan oluşuyorsa düğüm sayısı bire düşer ve normal düğüm olur.

```
bool X_DirectoryNode::split(SET_OF_MBR *in, SET_OF_MBR *out1, SET_OF_MBR *out2)
{
    SET_OF_MBR t1, t2;
    MBR r1, r2;

    // first try topological split, resulting in two sets of MBRs t1 and t2
    topological_split(in, t1, t2);
    r1 = t1->calc_mbr(); r2 = t2->calc_mbr();

    // test for overlap
    if (overlap(r1, r2) > MAX_OVERLAP)
    {
        // topological split fails -> try overlap minimal split
        overlap_minimal_split(in, t1, t2);

        // test for unbalanced nodes
        if (t1->num_of_mbrs() < MIN_FANOUT || t2->num_of_mbrs() < MIN_FANOUT)
            // overlap-minimal split also fails (-> caller has to create supernode)
            return FALSE;
    }

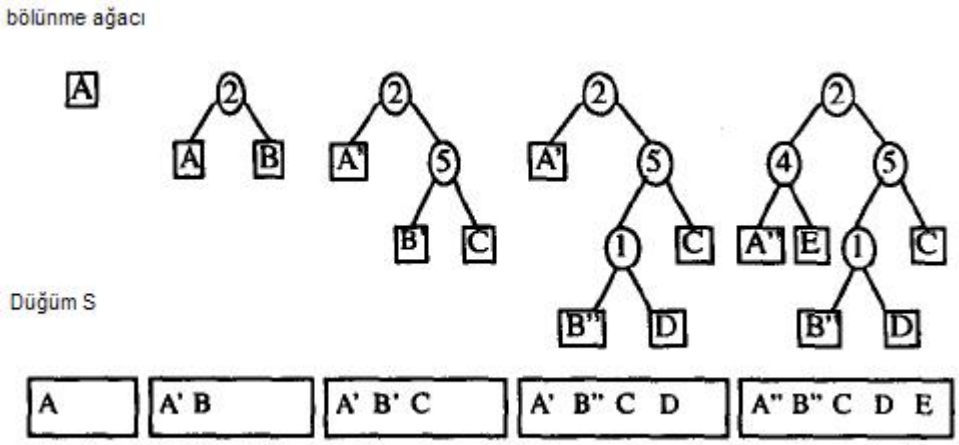
    *out1 = t1; *out2 = t2;
    return TRUE;
}
```

Şekil 2.16. Dizin Düğümleri için X Ağaç Bölünme Algoritması

2.4.3. Minimal üst üste binme bölünmesi

Minimal üst üste binen bölünmeyi bulmak için düğümü iki tane alt kümeye ayırmak ve MÇA'larının kesişim oranı en küçük olan iki alt kümeyi bulmak gerekir. Nokta veri için her zaman üst üste binmesiz bir bölünme vardır. Fakat bu iki düğümün de dengeli olacağı anlamına gelmez.

Uygun bir dizin yapısı için minimal üst üste binen bölünmelerin dengeli olmasına dikkat edilir. Ancak bölünen tüm MÇA'lara göre bölünmüş bir boyut varsa üste üste binmesiz bir bölünme vardır.



Şekil 2.17. Bölünme Geçmişi Örneği

Üst üste binmesiz bir bölünme bulmak için düğümün bütün MÇD'lerinin bölündüğü boyutu bulmak gerekir. Hangi boyutta hangi MÇD'nin bölünüp hangi yeni MÇD'yi oluşturduğuna dair gerekli bilgi bölünme geçmişi tarafından sağlanır. Bölünme geçmişi bir ikili ağaç gibi temsil edilir. Bölünme ağacındaki her yaprak düğüm S düğümünün bir MÇD'sine denk gelir. Bölünme ağacının iç düğümleri daha önce var olan ama bölündükten sonra yeni MÇD'ler oluşturup ortadan yok olan MÇD'leri temsil etmektedir. İç düğümler bölünme ekseniyle, yaprak düğümler ilgili MÇD'lerle işaretlenmiştir. İç düğümün sol alt ağacında yer alan yapraklarla ilişkili bütün MÇD'ler sağ alt ağaçtaki yaprak düğümlerle ilişkili MÇD'lerden daha düşük değerlere sahiptir.

Şekil 2.17'de S düğümünün bölünme geçmişi bir bölünme ağacıyla gösterilmektedir. İşlem tek bir MÇD A ile başlar. Düzenli dağılmış veriler için A bütün

boyutlardaki tüm aralıklara denk gelir. A'nın boyut 2'yi kullanarak bölünmesi iki yeni düğüm oluşturur: A' ve B. Bu durumda bölünme ağacının bir tane iç düğümü (2) ve iki tane yaprak düğümü (A've B) vardır. 5 boyutunu kullanarak MÇD B'yi bölmek iki yeni düğümle sonuçlanır: B' ve C. B' ve A' düğümlerini tekrar böldükten sonra Şekil 2.17'nin en sonundaki duruma ulaşırız.

Üst üste binmesiz bir bölünme bulabilmek için S'in bütün MÇD'lerinin bölündüğü bir boyutun varlığı araştırılmalıdır. Bunun için ağaç kökten sona doğru gezilmelidir. Örneğin MÇD C boyut 2 ve 5'e göre bölünmüştür. O yüzden kökten C'ye doğru olan yolda ara düğümler 2 ve 5 ile işaretlenmiştir. Şekil 2.17'de açıkça görülüyor ki bütün MÇD'ler 2 boyutuna bölünmüşlerdir. Genelde bu her MBR'nin bir kere bölündüğü boyut bölünme ağacının kök boyutudur.

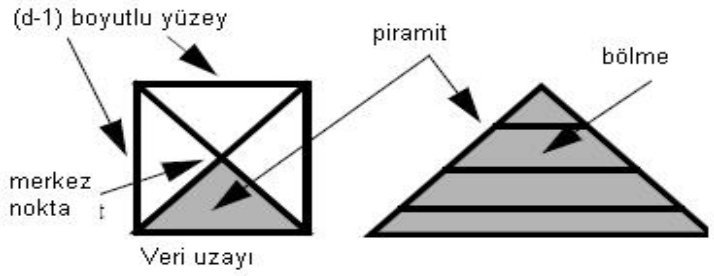
Rastgele seçilen bir bölünme ekseninin üst üste binmesiz bir bölünme için doğru eksen olma olasılığı yüksek boyutlu uzaylarda çok düşüktür. Eğer topolojik bölünme başarısız olursa algoritma üst üste binmesiz bölünmeyi bulmaya çalışır. Eğer bu bölünme sonunda da düğümler dengesizse süper düğümlere başvurulur.

2.5. Piramit Tekniği

Piramit Tekniğinin [Berchtold vd., 1998] temel düşüncesi d boyutlu veri noktalarını tek boyutlu değerlere çevirmek ve daha sonra B+ Ağaç gibi etkili indeks yapıları kullanarak bu verileri depolamak ve verilere erişmektir. Aslında herhangi bir tek boyutlu erişim metodu da kullanılabilir. Hem ekleme hem de aralık sorgusu işleminde d boyutlu veri B+ Ağaç tarafından işlenebilecek tek boyutlu bilgiye dönüştürülür. Veri tek boyutlu anahtar kullanarak indekslenmesine rağmen, B+ Ağaç yaprak düğümlerinde ilgili tek boyutlu anahtarla birlikte d boyutlu noktalar depolanır. Bu yüzden ters bir dönüşüm geliştirmek gerekmez. Dönüşüm veri uzayının d boyutlu piramitlere parçalanması üzerine kurulmuştur. Bu yüzden dönüşümü tanımlamak yerine önce Piramit Tekniğinde veri uzayını parçalama açıklanmalıdır.

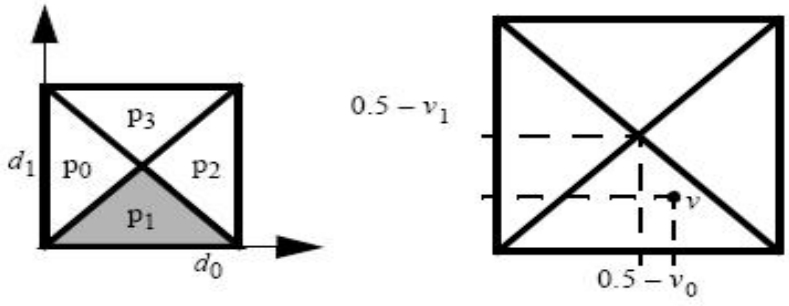
2.5.1. Veri uzayı parçalama

Piramit Tekniğinde veri uzayı parçalama iki adımda yapılır: ilk adımda veri uzayı 2d sayıda piramide ayrılır. Bu piramitler uzayın orta noktasını (0.5, 0.5, …, 0.5) tepeleri olarak kabul ederler ve tabanları (d-1) boyutlu bir yüzeydir. İkinci adımda bu piramitlerden her biri birkaç parçaya ayrılır. Bu parçalar B+ Ağaçtaki bir veri sayfasının karşılığıdır. Şekil 2.18'deki 2 boyutlu örnekte, uzay 4 üçgene bölünmüştür (üçgen d boyutlu piramitlerin 2 boyutlu karşılığıdır). Bu üçgenlerin hepsi veri uzayının orta noktasını tepe olarak almıştır ve uzayın sınırları bu üçgenlerin tabanları olmuştur (şekil 2.18 sol). İkinci adımda, bu 4 parça tabana paralel hatlarla birkaç veri sayfasına bölünmüştür (şekil 2.18 sağ). 2 boyutlu olmayan bir d boyutlu uzayda piramidin tabanı örnekteki gibi 1 boyutlu bir hat değildir ama (d-1) boyutlu bir hiper düzlemdir. Bir küp şeklindeki uzayda (d=3), yüzey olarak 6 tane (2d) 2 boyutlu (d-1) hiper düzlem ve 6 (2d) tane piramit vardır.



Şekil 2.18. Veri Uzayını Piramitlere Parçalama

Şekil 2.19a'daki 2 boyutlu örnek gibi piramitleri numaralandırarak parçalama stratejisinin temelini oluşturan şu gözlemler yapılır: Küpün i. (d-1) boyutlu yüzeyinde (piramidin tabanı) yer alan bütün noktalar genel bir özelliğe sahiptir. Bütün bu noktaların i. koordinatları 0 ya da (i-d). koordinatları 1dir. Piramidin tabanının (d-1) boyutlu hiper düzlem olduğu gözlenebilir. Çünkü bir koordinat sabit iken diğer (d-1) koordinat değişir.



a) Piramitleri numaralandırma

b) Piramitte bir nokta

Şekil 2.19. Piramitlerin Özellikleri

Diğer taraftan i<d olmak üzere i. piramit p_i de yer alan her v noktasının merkez noktaya i. koordinattaki uzaklığı diğer koordinatlardaki uzaklıklarından daha küçüktür. Daha biçimsel olarak:

$$\forall j,\ 0 \le j < d, j \ne i:\ (\left|0.5 - v_i\right| \le \left|0.5 - v_j\right|)$$
$$\forall j,\ 0 \le j < d, j \ne (i-d):\ (\left|0.5 - v_{(i-d)}\right| \ge \left|0.5 - v_j\right|)$$

Şekil 2.19b bu özelliği 2 boyutta resmediyor: düşük piramitlerdeki bütün noktaların merkez noktaya d_0 yönüne göre uzaklıkları d_1 yönüne göre uzaklıklarından daha yakındır. Bu genel özellik v noktasını içeren piramit p_i yi bulmak için çok basit bir yol sağlar: sadece merkezden maksimum sapma yapan ($|0.5 - v_i|$) boyut i belirlenmelidir. Daha biçimsel bir şekilde:

Tanım 1: (v noktasının piramidi)

p_i piramidi içinde tanımlı d boyutlu bir v noktası için,

$$i = \begin{cases} j_{max} & if(v_{j_{max}} < 0.5) \\ (j_{max} + d) & if(v_{j_{max}} \ge 0.5) \end{cases}$$

$$j_{max} = (j \,|(\forall k,\ 0 \le (j,k) < d, j \ne k:\ \left|0.5 - v_j\right| \ge \left|0.5 - v_k\right|))$$

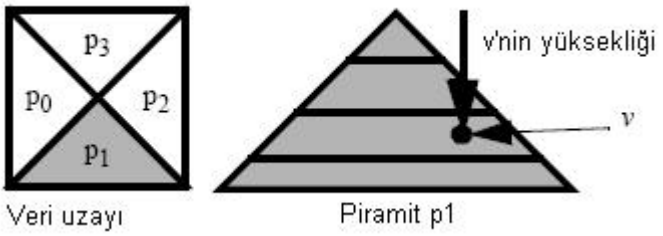
Bu tanım Piramit Tekniği için çok önemlidir. Bundan sonraki bütün düşünceler bu tanımı temel alacaktır.

Başka bir önemli özellik de v noktasının piramit içindeki yeridir. Bu yer tek bir değerle tespit edilir. Bu değer j_{max} boyutuna göre noktanın merkez noktaya uzaklığıdır. Geometrik olarak piramit içindeki noktanın yüksekliğine karşılık gelir, bu yüzden teknikte bu değere v noktasının yüksekliği denir. (Şekil 2.20)

Tanım 2: (v noktasının yüksekliği)

v d boyutlu bir nokta olsun. p_i Tanım 1e göre v noktasının içinde bulunduğu piramittir. v noktasının yüksekliği h_v şöyle tanımlanabilir:

$$h_v = |0.5 - v_{i \text{ MOD } d}|$$



Şekil 2.20. Piramit İçindeki Noktanın Yüksekliği

Tanım 1 ve Tanım 2'yi kullanarak d boyutlu nokta v'yi $(i+h_v)$ değerine dönüştürebiliriz. Daha biçimsel olarak:

Tanım 3: (v noktasının piramit değeri)

v d boyutlu bir nokta olsun. p_i v'nin Tanım1e göre bulunduğu piramit olsun ve h_v Tanım 2ye göre v'nin yüksekliği olsun. v'nin piramit değeri pv_v şöyle tanımlanır:

$$pv_v = (i + h_v)$$

i bir tam sayıdır ve h_v [0, .5] aralığında bir reel sayıdır. Bu yüzden her piramit [i, (i+0.5)] piramit değeri aralığına sahiptir ve iki piramidin de piramit değer aralıkları birbirinden ayrıdır. Dikkat edilirse v ve v' gibi iki nokta aynı piramit değerine sahip olabilirler. Fakat yukarıda bahsedildiği gibi teknik ters bir dönüşüm gerektirmiyor ve bu yüzden ikisini birbirinden ayırt etmek için bir dönüşüme ihtiyaç yoktur.

2.5.2. İndeks yaratma

q noktasının piramit değerini tespit eden dönüşümle, Piramit Tekniğine göre bir indeks inşa etme basit bir görevdir. Dinamik olarak bir nokta eklemek için, ilk olarak noktanın piramit değeri pv_v'yi bulmak gerekir ve sonra bu nokta pv_v'yi anahtar olarak kullanarak B+ Ağaca eklenir. Sonuç olarak, d boyutlu bir nokta olan v'yi ve pv_v'yi B+ Ağacın veri sayfasına uygun olarak depolanır. Güncelleme ve silme işlemleri benzer olarak yapılabilir.

Genel olarak, B+ Ağacın sonuç sayfaları aynı piramit içinde bir çok noktayı içerir. Bu noktaların genel özellikleri, piramit değerlerinin veri sayfasında verilen en büyük ve en küçük anahtar değerleri arasında bir aralıkta yer almasıdır. Böylece, B+ Ağaç veri sayfasının geometrik olarak karşılığı Şekil 2.20'de (sağ) görülebileceği gibi piramitteki bir bölmedir.

2.5.3. Sorgu işleme

Eklemenin aksine, Piramit Tekniğini kullanan silme ve güncelleme işlemleri ve sorgu işleme daha karmaşıktır. Verilen bir q noktasının veritabanında olup olmadığını sorgulayan nokta sorgulara bakılırsa, Piramit Tekniği kullanılarak bu problem öncelikle q'nun piramit değeri pv_q hesaplanarak ve B+ Ağacı pv_q ile sorgulayarak çözülür. Sonuç olarak pv_q'yu paylaşan d boyutlu noktalar kümesi elde edilir. Böylece bu indirgenmiş küme taranarak q'u içerip içermediği bulunabilir.

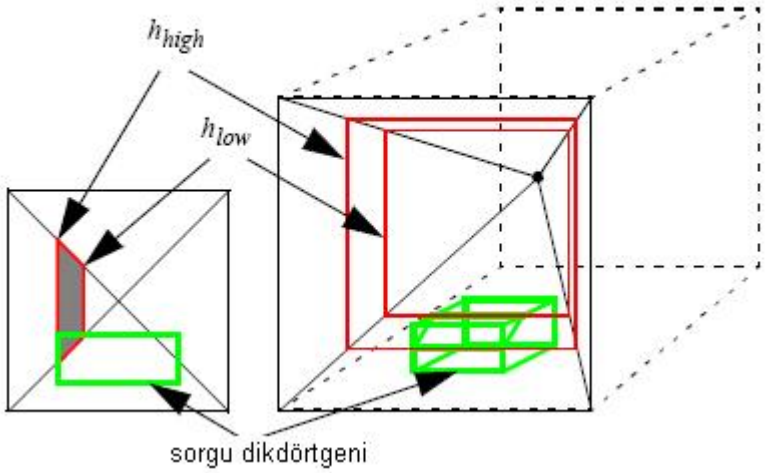
Aralık sorguları durumunda, problem aşağıdaki gibi tanımlıdır:

"Verilen bir d boyutlu aralık

$$[q_{0_{min}}, q_{0_{max}}], \ldots, [q_{d-1_{min}}, q_{d-1_{max}}],$$

da, aralığa giren veritabanındaki noktaları tespit etmek.". Çok boyutlu aralıkların geometrik karşılığı hiper dikdörtgendir. Nokta sorgularına benzer şekilde, d boyutlu

sorguyu B+ Ağaç üzerinde 1 boyutlu sorguya çevirme problemi ortaya çıkar. Bununla birlikte, Şekil 2.21 (sol) te basit 2 boyutlu örnekte gösterildiği gibi sorgu dikdörtgeni birkaç piramidi kesebilir ve kesişen alan hesaplaması kolay değildir. Ayrıca örnekten anlayabildiğimiz gibi, ilk olarak hangi piramitlerin sorgudan etkilendiğinin tespit edilmesi, ve ikinci olarak piramitler içindeki aralıkların tespit edilmesi gerekir. Bir noktanın aralık içinde yer alması testi basit bir kritere dayanır (h_v iki değer arasında mı). O yüzden bütün nesneleri tespit etmek bir tek boyutlu indeksleme problemidir. Aralık dışındaki nesneler sorgu dikdörtgenine alınmazlar. Aralık içindeki noktalar, daha sonraki bir araştırmanın adaylarıdırlar. Şekil 2.21'de bazı adayların kesiştiği, diğerlerinin ise kesişmediği görülüyor. Sonra, ileriki adımda basit bir dikdörtgende nokta testi yapılır.



Şekil 2.21. Aralık Sorgularının Dönüşümü

Basitleştirmek için, algoritmada i<d olmak koşuluyla p_i piramitlerine odaklanılabilir. Ancak algoritma i>d durumundaki p_i piramitleri için de kolayca geliştirilebilir. Algoritmanın ilk adımı olarak, sorgu dikdörtgeni q eşdeğer dikdörtgen $\hat{q}$ ya dönüştürülür. Aralıklar merkez noktaya uzaklıklarına göre tanımlanmıştır:

$$\hat{q}_{j_{min}} = q_{j_{min}} - 0.5 \text{ and } \hat{q}_{j_{max}} = q_{j_{max}} - 0.5 \ , \ \forall j, 0 \leq j < d$$

Ek olarak, şu belirtilmelidir ki bu bölümdeki bütün noktalar merkez noktaya göre tanımlıdır. Tanım1e dayanarak, bir piramidin verilen sorgudan etkilenip etkilenmediği anlaşılabilir. Görüleceği gibi, aşağıda tanımlı olan aralığın mutlak

minimum ve maksimumu belirlenmelidir: MIN(r) aralık r'nin mutlak değerlerinin minimumu olsun,

$$MIN(r) = \begin{cases} 0 & \text{if } r_{min} \le 0 \le r_{max} \\ min(|r_{min}|, |r_{max}|) & \text{diğer durumlar} \end{cases}$$

Şuna dikkat edilmelidir ki $|r_{min}|$ $|r_{max}|$ ten daha büyük olabilir. Benzer şekilde:

$$MAX(r) = max(|r_{min}|, |r_{max}|)$$

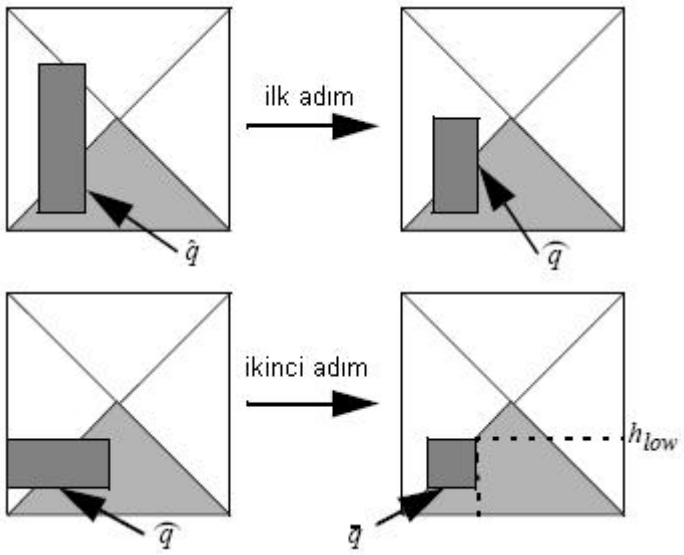
Önerme 1: (Bir Piramit ve Bir Dikdörtgenin Kesişimi)

Bir piramit p_i ile $[\hat{q}_{0_{min}}, \hat{q}_{0_{max}}], \ldots, [\hat{q}_{d-1_{min}}, \hat{q}_{d-1_{max}}]$ şeklindeki bir hiper dikdörtgen ancak aşağıdaki koşul sağlanıyorsa kesişir:

$$\forall j, 0 \le j < d, j \ne i: \hat{q}_{i_{min}} \le -MIN(\hat{q}_j)$$

İspat:

Eğer dikdörtgen içinde p_i'ye düşen bir v noktası varsa sorgu dikdörtgeni piramit p_i'yi keser. Böylece v'nin $|v_j|$ koordinatlarının hepsi $|v_i|$ den küçük olmalıdır. Bu ancak, sorgu dikdörtgeninin minimum mutlak değeri j koordinatında merkez noktaya $\hat{q}_{i_{min}}$ den daha yakınsa mümkün olabilir. Önerme 1 bu gerçeğin tüm boyutlar için gerekliği olmasından yola çıkar.



Şekil 2.22. Sorgu Dikdörtgeninin Kısıtlanması

İkinci adımda, piramit p_i'deki hangi piramit değerlerinin sorgudan etkilendiği tespit edilmelidir. Böylece, [0, 0.5] aralığında [h_{low}, h_{high}] aralığını aramış oluruz. Hem sorgu dikdörtgeni hem de p_i piramidiyle kesişen bütün piramit değerleri [$i+h_{low}$, $i+h_{high}$] aralığındadır. Şekil 2.21 bu aralığı iki boyut için göstermektedir.

h_{low} ve h_{high}'ı bulmak için ilk olarak sorgu dikdörtgeni piramit p_i'ye kısıtlanmalıdır. Merkez noktanın üstündeki bütün noktalar kaldırılır:

$$\widehat{q}_{i_{min}} = \hat{q}_{i_{min}},\ \widehat{q}_{i_{max}} = min(\hat{q}_{i_{max}}, 0),$$

$$\widehat{q}_{j_{min}} = \hat{q}_{j_{min}}, \text{ve} \quad \widehat{q}_{j_{max}} = \hat{q}_{j_{max}}, (0 \le j < d), j \ne i \quad \text{ise}$$

Bu kısıtlamalar $p_0 \ldots p_{d-1}$ için yapıldı. Bu yüzden $\hat{q}_{i_{min}}$ ve $\hat{q}_{i_{max}}$ değerleri negatiftir. Bu kısıtlamanın etkisi Şekil 2.22 deki 2 boyutlu örnekte gözükmektedir.

[h_{low}, h_{high}] aralığının tespiti eğer merkez nokta sorgu dikdörtgenine dahil ise çok kolaydır. ($\forall j, (0 \le j < d): (\hat{q}_{j_{min}} \le 0 \le \hat{q}_{j_{max}})$ durumunda). Bu durumda sorgu dikdörtgeninin uzantısı sonuç olarak kullanılır:

$$h_{low} = 0 \text{ ve } \quad h_{high} = MAX(\widehat{q}_i).$$

Eğer merkez nokta sorgu dikdörtgenine dahil değil ise önce

$$h_{high} = MAX(\widehat{q}_i)$$

olduğuna dikkat edilmelidir. Bu, sorgu dikdörtgeninin v gibi

$$v_i = MAX(\widehat{q}_i)$$

olan en azından bir nokta içermek zorunda olmasından dolayıdır. Çünkü aksi halde sorgu dikdörtgeniyle piramit p_i arasında hiçbir kesişim olmazdı.

h_{low}'ı bulmak için, sorgu dikdörtgeni ve piramit p_i içindeki nokta yüksekliklerinin en küçüğü bulunmalıdır. $\widehat{q}$ ve p_i içindeki noktalar dikkate alınırken, h_{low}'a etki etmeden

$[\widehat{q}_{j_{min}}, \widehat{q}_{j_{max}}]$ $(0 \le j < d), j \ne i$ aralıkları $[\widehat{q}_{i_{min}}, \widehat{q}_{i_{max}}]$ ile kesiştirilebilir. Daha sonra yeni dikdörtgenin bütün boyutlarının min-değerlerinin minimumu $\bar{q}$ h_{low}'a eşit olur. Şekil 2.22 (alt) bu işlemlerin örneğini gösteriyor. Açık olarak, her örneğin solundaki ve sağındaki seçili dikdörtgenler aynı h_{low} değerine sahiptir.

```
Point_Set PyrTree::range_query(range q)
{
  Point_Set res;
  for (i = 0; i < 2d; i++) {
    if (intersect(p[i], q) {
    // using Lemma 1
      determine_range(p[i], q, h_low, h_high);
      // using Lemma 2

      cs = btree_query(i+h_low, i+h_high);
      for (c = cs.first; cs.end; cs.next){
        if (inside(q, c))
          res.add(c);
      }
    }
  }
  return res;
}
```

Şekil 2.23. Aralık Sorgularını İşleme (Algoritma)

Verilen bir $\hat{q}$ sorgusu bu sorgudan etkilenmiş bir p_i olsun. Kesişim aralığı [h_{low}, h_{high}] şöyle bulunur:

Durum 1: ($\forall j, 0 \le j < d$: $(\hat{q}_{j_{min}} \le 0 \le \hat{q}_{j_{max}})$)

$$h_{low} = 0$$

$$h_{high} = MAX(\widehat{q}_i)$$

Durum 2: (diğer hallerde)

$$h_{low} = min_{(0 \le j < d, j \ne i)}(\bar{q}_{j_{min}}) \quad (*)$$

$$h_{high} = MAX(\widehat{q}_i)$$

$$\bar{q}_{j_{min}} = \begin{cases} max(MAX(\widehat{q}_i), MIN(\widehat{q}_j)) & \text{if } MAX(\widehat{q}_j) \ge MIN(\widehat{q}_i) \\ MIN(\widehat{q}_i) & \text{diğer hallerde} \end{cases}$$

Bu sorgu işleme algoritması şekil 2.23'de gösterilmektedir.

2.6. Yüksek Boyutlu Veriler İçin Piramit Tekniğinden Esinlenilmiş Yeni Bir İndeksleme Yöntemi

Eğer bir indeksleme yöntemi verinin tüm koordinatlarını indekste tutmayı gerektirirse, indeks yapısının büyüklüğü boyutlulukla orantılı olur. İndeksin büyüklüğünü azaltmak için Berchtold ve arkadaşları Piramit-Ağacı [Berchtold vd., 1998] diye adlandırılan, yüksek boyutlu veriyi B^+-ağacı gibi bazı klasik indeks yapılarının kullanılabileceği tek boyutlu uzaya yönlendiren bir yöntem ileri sürmüşlerdir. Bu yöntem X-Ağaçlarından [Berchtold, 1996] ve R-Ağacından [Guttman, 1984] esinlenmiş diğer çok boyutlu indekslerden daha iyi bir performans göstermiştir. Ancak, bir piramitte birçok veri piramidin tabanında yoğunlaşır. Her piramitteki dilimler indeksi çok ayırımcı yapamaz ve bir piramitteki veri noktaları daima aynı indeks değerine sahip olurlar.

Bir Piramit ağacında bir veri noktası bir piramitle ilişkilendirilir ve bir veri noktası bir piramit tabanıyla indekslenir. Eğer boyut d ise, piramidin tabanı $(d - 1)$ boyutlu bir hiper düzlemdir. Daha iyi bir arama performansı sağlamak için indeksi daha

ayrımcı yapma uğrunda indeks değerlerinin sayısı artacaktır. Bu yöntem yüksek boyutlu veriyi tabanlara dayalı indeksleme çözümü sunar.

2.6.1. Yöntemin amacı

Görme yeteneğimizin sınırları yüzünden, hiç kimse 3 boyutlu uzaylardan fazlasını göremez. Onları sadece 2 ve ya 3 boyutlu uzaydan yola çıkarak hayal edebiliriz. Ancak, yüksek boyutlu bir uzayın 2 veya 3 boyutlu uzaydan hayal edilebilir olmadığı ileride örneklerle gösterilecektir. Bu yöntemin anlatımında veri uzayı (0,1]'e normalize edilmiştir. Böylece veri uzayı bir hiperküp olur. Bu hiperküpün kenarlarının uzunluğu 1'dir.

2.6.1.1. Yüksek boyutluluğun sezilemezliği

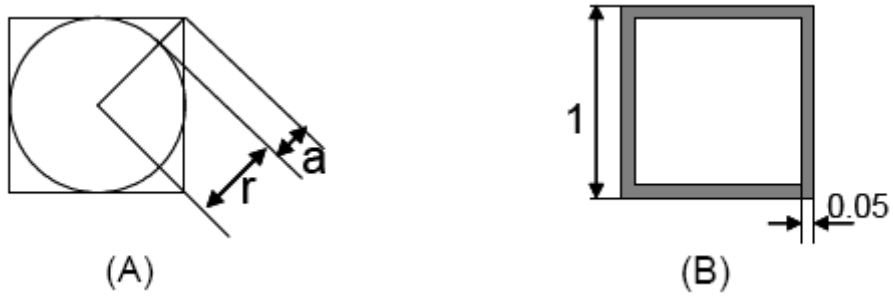
Yüksek boyutlu veri her yerde bulunabilir. Zaman serileri geleneksel yüksek boyutlu veridir. Ayrıca yüksek boyutlu veri bir nesneyi tanımlamak için kullanılabilir. Mesela bir insan uzunluk, ağırlık, yaş ve benzeri özellikleri kullanılarak tanımlanabilir. Eğer iki nesne benzerse, göremesek bile bu iki nesnenin yüksek boyutlu uzayda birbirine iki yakın noktaya denk geldiğini düşünebiliriz. Boyutsal uzayın eksenleri özellik vektörleri içerir. Ancak, yüksek boyutlu veri 2 veya 3 boyutlu uzaydaki veriden farklı karakteristik özelliklere sahiptir. Şekil 2.24(A) r yarıçaplı bir daire ve onu çevreleyen bir dörtgeni gösteriyor. Dörtgenin köşesinden daireye uzaklık *a* olarak belirtilmektedir. *a* için 2 ve 3 boyutlu uzayda şu eşitsizlik geçerlidir:

$$r \geq a$$

Ancak boyut arttıkça, bu eşitsizlik uygun olmamaya başlar. Örneğin 16 boyutlu bir uzayda *a r*'nin 3 katı uzunluktatdır. ($a=3r$) Şu eşitlikle kolayca hesaplanabilir:

$$(r+a)^2 = \underbrace{r^2 + \ldots + r^2}_{16} \qquad (1)$$

Bizim görsel alanımız 3 boyutlu uzaya alışıktır. Bu çeşit fenomenaları anlamak zordur. Birçok araştırmacı orijinal olarak 2 boyutlu uzaylarda kullanılan uzaysal indeks yöntemi R-ağacı geliştirerek sonuca ulaşmaya çalışmıştır. Bu yöntemlerin kullanılmaz olduğu kanıtlanınca da "boyutluluk belası" deyimi kullanılmıştır.



Şekil 2.24. (A) Köşegenle yarıçap arasındaki ilişki (B) Yüzey ve hacim

2.6.1.2. Yüksek boyutlu veri kümesinin dağılımı

Bir indeks ağacı düğümü oluşturmak için veri uzayı parçalara ayrılır. Boyut arttığı zaman, hacimdeki üssel değişiklikler dikkate alınmalıdır. Örneğin, *d* boyutlu ve bir kenarının uzunluğu *e* olan bir hiperküpün hacmi $vol=e^d$ dir. Burada $0 < e < 1$'dir ve hacim *d* ye göre üssel olarak azalır. Diğer taraftan $e > 1$ olduğunda hacim üssel olarak artar. Şimdi bir hiperküp içindeki veri noktalarını ele alalım. Hacmin ne kadarı yüzey tarafından işgal edilir? Şekil 2.24(B) kenarları 1 uzunluğundaki bir kareyi gösterir.

Boyut sayısı	Yüzey hacmi
2	$1 - 0.9^2 = 0.190$
3	$1 - 0.9^3 = 0.271$
. .	
50	$1 - 0.9^{50} = 0.994846$

Tablo 2.1. Yüzeyin hacmi

Kenar tabakasının hacmi 0.05 kalınlığında. Tablo 2.1 çeşitli boyutların hacmini gösteriyor. Tablodan 50 boyutlu hiperküpte hacmin %99,5'inin yüzeyde olduğu görülebilir. Bu tip yüksek boyutlu uzaylarda eğer veriler düzenli bir şekilde dağılmışsa, şöyle bir yargıya varılabilir: "Birçok veri hiperküpün içi yerine yüzeyde toplanmıştır." Bu oluşturulan yöntemin çıkış noktasıdır. Bununla başa çıkmak için yüksek boyutlu uzaylar için yüzey tabanlı indeks yapısı öne sürülmüştür.

Eğer boyu sayısı *d* ise, hiperküpün yüzeyi *d*-1, *d*-2, …, 1, 0 boyutlu hiper düzlemden oluşur. Örneğin, 3 boyutlu bir küp 6 kareden (2 boyut), 12 kenardan (1 boyut) ve 8 köşeden (0 boyut) oluşur. Bu düzlemler ileride yüksek boyutlu veriyi indekslemek için kullanılacaktır.

2.6.1.3. Hiperkübteki köşeler, kenarlar ve hiper düzlemler

Bir kare (2 boyut) 4 kenar ve 4 köşeden oluşur; bir küp (3 boyut) 6 kareden, 12 kenardan ve 8 köşeden oluşur. Genel olarak, *d* boyutlu bir hiperküpte *d*-1 boyutlu hiper düzlemlerin sayısı 2*d* dir. Hiperküp 2*d* tane hiper düzlem tarafından çevrelenmiştir. Bir hiper düzlem de (*d*-2) boyutlu hiper düzlemler tarafından oluşur. Bu yüzden, bütün hiper düzlemler arasındaki ilişki şudur: İki (*d*-1) boyutlu hiper düzlem bir (*d*-2) boyutlu

hiper düzlemde kesişir. Üç (*d*-1) boyutlu hiper düzlem bir (*d*-3) hiper düzlemde kesişir. Son olarak, *d*-1 tane (*d*-1) boyutlu hiper düzlem bir çizgide kesişir. Bir hiperküpü çevreleyen hiper düzlemlerin sayısı yardımcı teoremde verilmiştir.

YARDIMCI TEOREM 1. *d* boyutlu bir hiperküp 0, 1, …, (*d*-1) boyutlu hiper düzlemler tarafından çevrilir, i boyutlu hiper düzlemlerin sayısı $2^i \times C^i_d$.

Hiper düzlemlerin boyu sayısı	Hiper düzlemlerin şekli	Hiper düzlemlerin sayısı
19	Hiper düzlem	40
18	Hiper düzlem	760
17	Hiper düzlem	9,120
16	Hiper düzlem	77,520
15	Hiper düzlem	496,128
14	Hiper düzlem	2,480,640
13	Hiper düzlem	9,922,560
12	Hiper düzlem	32,248,640
11	Hiper düzlem	85,995,520
10	Hiper düzlem	189,190,144
9	Hiper düzlem	343,982,080
8	Hiper düzlem	515,973,120
7	Hiper düzlem	635,043,840
6	Hiper düzlem	635,043,840
5	Hiper düzlem	508,035,072
4	Hiper düzlem	317,521,920
3	Hiper düzlem	149,422,080
2	Düzlem	49,807,360
1	Çizgi	10,485,760
0	Köşe	1,048,576

Tablo 2.2. 20 boyutlu hiperküpün hiper düzlemlerinin sayısı

Boyut arttıkça, hiper düzlem sayısı hızlı bir şekilde artar. Bu sayı genellikle bir çok veri kümesinin boyutunun üzerindedir. Bu yüzden bir veri noktasının hiperküpün bir hiper düzlemine tekabül etmesi olasıdır. Hiper düzlem veri noktasının indeks anahtarı olur. Birbirine yakın veri noktalarını aramak birbirine yakın hiper düzlemleri arama olur. Veri noktasını bir hiper düzleme yönlendirmek için piramit ağaç tekniği uygulanabilir; veri uzayı tepeleri hiperküpün merkezi ve tabanları hiper düzlemler olan piramitlere parçalanabilir. Birçok veri noktası veri uzayının yüzey kısımlarında toplandığından, piramitteki veri noktaları piramidin tabanı ile temsil edilebilir. Eğer hiper düzlemler bir sıra içindelerse, piramitteki veri noktaları B^+ ağaç indeks yapısı ile indekslenebilir. Bu yüksek boyutlu verinin doğrusal veriye dönüşümüdür. Bir aralık sorgusu verildiğinde, yakın veri noktalarını arama sorgu aralığı ile üst üste binen piramitleri arama olur.

2.6.2. Yüzey tabanlı uzaysal indeks yapısı

Veri uzayı kenarı 1 olan bir hiperküpe normalize olmuş varsayılır. İki karşılıklı piramidin tabanları eksen x_i ye diktir. $x_i = 0$ ve $x_i = 1$ şeklinde ifade edilebilirler. Tabanı $x_i = 0$ olan bir piramitte her veri noktası $(x_0, x_1, \ldots, x_{d-1})$ aşağıdaki kuralı sağlar:

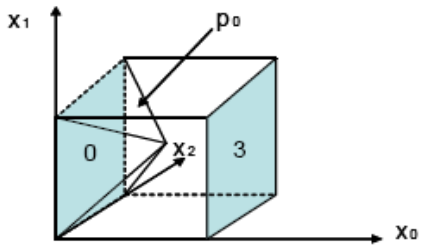
$$x_i \leq \min(x_j, 1 - x_j) \qquad (j = 0, 1, \ldots, i-1, i+1, \ldots, d-1) \text{ iken} \qquad (2)$$

(2) eşitliğine dayanarak, verilen bir veri noktası için, veri noktasının ait olduğu piramit belirlenebilir. Bu yöntemde piramit numarası indeks anahtarı olarak kullanılır. Daha sonra doğrusal B^+ ağaç indeks yapısı benzer verileri aramak için kullanılır.

2.6.2.1. Piramit sırası

Bir d boyutlu hiperküp $(d\text{-}1)$ boyutlu hiper düzlemlerle çevrilidir. Eksenlerin $x_0, x_1, \ldots, x_{d-1}$ olduğunu varsayalım. Piramitlerin sıra numaraları Tablo 2.3 te atanmıştır. Piramidin sıra numarası tabanı ile belirlenir. Şekil 2.25 3 boyutlu bir küpü

göstermektedir. Tabanı $x_0 = 0$ olan piramidin sıra numarası 0 ve gösterimi p_0 dır. Tam karşısındaki piramit p_3 şeklinde gösterilir.



Şekil 2.25. Piramit sırası. 3 boyutlu bir küp. Eksenler $x_0 = 0$ ve $x_0 = 1$ sırasıyla sıra numarası 0 ve 3 olarak atanıyor. İlgili piramitler p_0 ve p_3 olarak gösterilir.

Hiperdüzlem	sıra	Pirmaitlerin sırası
$x_0 = 0$	0	p_0
$x_0 = 1$	d	p_d
$x_1 = 0$	1	p_1
$x_1 = 1$	d+1	p_{d+1}
$x_{d-1} = 0$	d-1	p_{d-1}
$x_{d-1} = 1$	2d-1	p_{2d-1}

Tablo 2.3 *d* boyutlu hiperküpte piramitlerin sırası

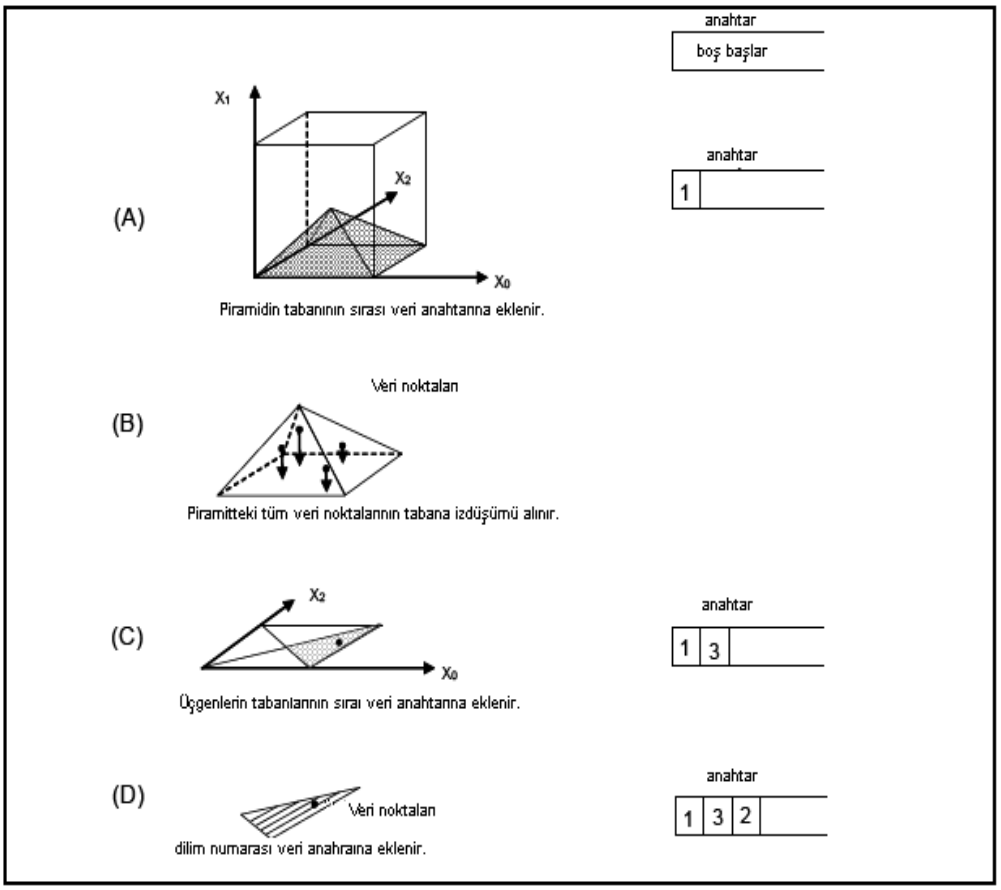
2.6.2.2. İndeks anahtarının oluşumu

Bütün verilerin indeks anahtarları boş olarak başlatılır. Bir veri noktası bir piramide ait olmalıdır. (Eğer veri noktası iki veya daha fazla piramidin sınırındaysa, nereye ait olacağı 4. bölümde tartışılacaktır). Bir hiperküp parçalandığı zaman, bir piramidin sıra numarası indeks anahtarına eklenir. Bu işlem yinelemeli olarak yapılır. Şekil 2.26 3 boyutlu küpte anahtarın nasıl oluşturulduğunu göstermektedir. Veri noktasının piramit $x_1 = 0$ da olduğunu varsayalım. İlk olarak indeks anahtarı boştur. daha sonra veri noktasının ait olduğu, tabanı $x_1 = 0$ olan piramidin sıra numarası "1" (Tablo 2.3'e bakınız) (A) kısmında gösterildiği gibi indeks anahtarına eklenir. Veri uzayını yinelemeli olarak parçalamak için, şekil (B)de gösterildiği gibi piramidin tabanına veri noktasının izdüşümü alınır. Taban 4 üçgene parçalanmıştır. Veri noktası 3 nolu üçgende yer aldığından (Tablo 2.3'te tarif edildiği gibi) indeks anahtarı (C)'deki gibi "3" ün de eklenmesiyle büyür. Son olarak üçgenin 8 dilime bölündüğünü varsayılır. Dilim numarası "2" veri anahtarına eklenir. Veri noktasının indeks anahtarı (D) de gösterildiği gibi "1", "3" ve "2" yi içerir. Bunları bir integer değer olarak birleştirmek kolaydır, şu şekilde: (1*4+3)*8+2=58. Bu formülde "4" ve "8" katsayıları sırasıyla (C) ve (D) deki çizgilerin ve dilimlerin toplam sayısıdır. İndeks anahtarının çözülümü ve veri noktasının hangi piramide ait olduğunu bulmak kolaydır. Genel olarak, parçalama aşağıdaki 2 adımla tanımlanır.

1. d boyutlu bir hiperküp $2d$ tane piramide parçalanır, bu piramitlerin tepeleri hiperküpün merkezi ve tabanları (d-1) boyutlu hiper yüzeylerdir.

2. Bir piramit içindeki bütün verilerin piramidin tabanına izdüşümü alınır. Adım 1 piramidin tabanı içinde tekrarlanır. Taban da bir hiperküptür. Boyutu (d-1) dir. Taban adım 1 de gösterildiği gibi 2(d-1) tane piramide bölünebilir.

Daha fazla parçalama yapıldıkça, piramidin sıra numarasını ekleyerek indeks anahtarı daha da uzar. Son parçalamada, verinin ait olduğu dilim indeks anahtarına eklenir. Algoritma piramit-ağaç tekniğinden esinlenmiştir. Ancak piramit-ağaç hiperküpü sadece bir defa parçalar.



Şekil 2.26. Anahtarın oluşturulması. Veri uzayı yinelemeli olarak parçalanır, verinin anahtarı verinin içinde bulunduğu piramitlerin tabanlarının sırasından oluşur.

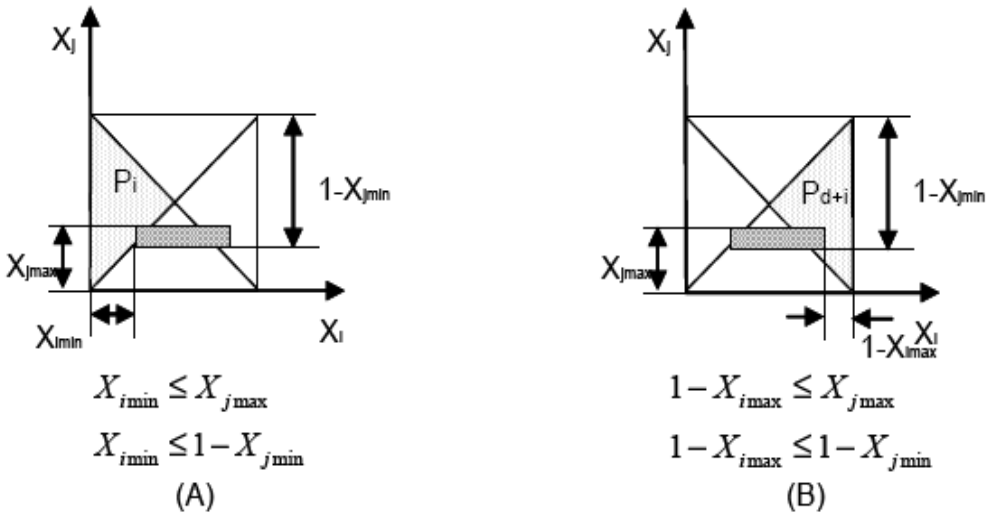
2.6.2.3. Aralık arama

İndeks anahtarının oluşturulması yöntemi kullanılarak, indeks değerleri bir sırada toplanır. Aramayı yapmak için klasik indeks yapısı B^+ ağaç kullanılabilir. Verilen bir aralık sorgusu için, aralık sorgusuyla üst üste binen piramitler hesaplanabilir. Eğer d boyutlu aralık $[X_{0min}, X_{0max}]$, $[X_{1min}, X_{1max}]$, …, gösterilirse, üst üste binen piramitleri hesaplama metodu Şekil 2.27'daki gibi tarif edilebilir:

Şekil 2.27 (A) aralık sorgusuyla kesişen piramit $p_i(0 \leq i \leq d)$ için koşulları gösterir. Aşağıdaki formüller koşulları belirtiyor:

$$X_{i\,min} \leq X_{j\,max} \qquad (3)$$

$$X_{i\,min} \leq 1 - X_{j\,min} \qquad (4)$$



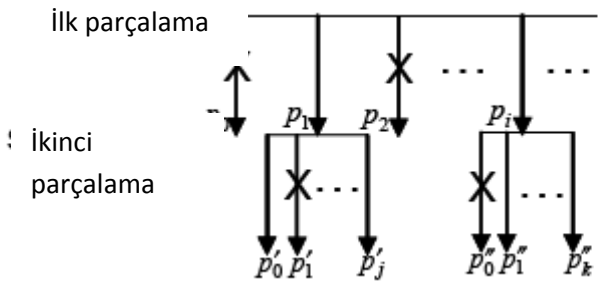
Şekil 2.27. Üstüste binen piramitleri bulmak. Aranan aralıkla üst üste binen piramitler aşağıda görünen formülle hesaplanır.

Alt şekil (A) da görüldüğü gibi p_i piramidi sorgu aralığı ile üst üste biner. Eşitlik 3 ve 4 sağlanır. Piramit $p_{d+i}(0 \leq i \leq d)$ için Şekil 2.27(B) geçerldir. Aralık sorgusuyla üst üste binme şartları şu şekilde ifade edilir:

$$1 - X_{i\max} \leq X_{j\max} \qquad (5)$$

$$1 - X_{i\max} \leq 1 - X_{j\min} \qquad (6)$$

Yukarıdaki hesaplama işlemi indeks anahtarının hesaplanması gibi yinelemeli olarak yapılmalıdır. Aralık sorgusuyla üst üste binen bütün piramitler B^+ ağaç indeks yapısı ile aranacaktır. Şekil 2.28 verilen bir aralık sorgusundan çevrilmiş piramitleri (veya indeks anahtarlarını) gösterir. p_0 ve p_2 piramitleri aralık sorgusuyla örtüşmez. Bu piramitlerin alt piramitleri kontrol edilmeyebilir. 2 parçalamadan sonra aralıkla örtüşen $p'_0 \ldots p'_j, p''_1 \ldots p''_k$ yı buluruz. İndeks anahtarları için arama yapılmalıdır. Bir kez daha parçalama arama için daha fazla indeks anahtarı ihtiyacını ortaya çıkarır. İndeks anahtarlarının doğruluğu arttığı halde hiperküpü daha fazla parçalamamanın nedeni budur.



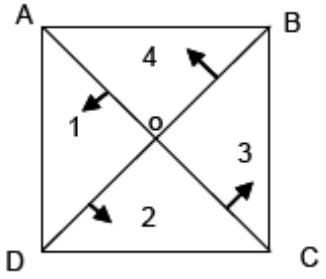
Şekil 2.28. Aralık sorgusundan indeks anahtarlarına çevirimi.

Şekil 2.28'de X işareti piramidin aralık sorgusuyla örtüşmediğini gösteriyor. İkinci parçalamada, alt piramitleri aralık sorgusuyla örtüşüyor mu diye sadece üst üste binen piramitler kontrol edilir.

2.6.3. Piramitlere sınır atanması

Piramitlerin sınırları iki piramitten daha fazlasına aittir. Etkili bir indeks yapısı için, bir sınırı bir piramitle paylaştırmak gereklidir. Ayrıca sınırlar iyi paylaştırılmalıdır. 2 boyutlu karede Şekil 2.29'da görüldüğü gibi 4 üçgene ayıran 4 tane sınır çizgisi vardır. Örneğin *OA* Δ*OAD* üçgenine atanır. *OB, OC, OD* sırasıyla Δ*OAB,* Δ*OBC,* Δ*ODC* üçgenlerine atanır

.

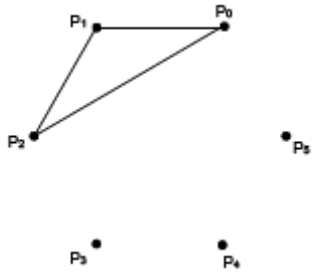


Şekil 2.29. Bir kare için kenar çizgileri paylaşımı

3 boyutlu durumlarda 6 piramit olur. Bunlar Şekil 2.25'de gösterildiği gibi 12 üçgen düzlemde kesişirler. Bu 12 sınır yüzeyi 6 piramide ikişer olarak bölünebilirler. Ancak problem çizgi sınırların nasıl atanacağıdır. 3 boyutlu uzayda 8 tane çizgi sınır

oluşur ki bu da 6'ya tam bölünmez. Kabul edilebilir bir atama 8 çizgiyi 2, 2, 1, 1, 1, 1 şeklinde atamaktır.

Bir veri noktası piramitlerin sınırında olduğunda, piramit ve veri noktasıyla ilgili bir politika belirlenmelidir. Benzer olarak bir aralık sorgusu piramitlerin sınırını kapsadığında sadece bir piramit dikkate alınmalıdır. Bu sorunu çözmek için bu raporda *radiant değeri* (*radiant value*) diye bir kavram öne sürülüyor. Bu kavramın ölçütü bir sınırın nasıl piramitlere bölüneceğidir.



Şekil 2.30. Sınır grafiği. 3 boyutlu bir küpte bir sınır çizgisi.

Bir hiper düzlem (veya çizgi, köşe) sınırı 2 piramitten daha fazlasının kesişimidir. Eğer bir veri noktası sınır üzerindeyse, Eşitlik 2'den hangi piramitlerin sınırı oluşturduğunu hesaplamak kolaydır. Şekil 2.30'daki gibi sınır için bir grafik oluşturulabilir. Bu grafik küpün 6 piramidini gösterir. Her piramit grafikte nokta olarak gösterilmiştir. Şekil 2.30'da bir çizgiyle bağlanmış iki piramit kesişen piramitlerdir. Örneğin, p_1 ve p_2 kesişir. Noktaların saat yönünün aksine konduğu varsayılır. Şekil 30 üç piramit (p_1, p_2, p_3) tarafından oluşturulan bir sınırı gösterir. Prensip olarak piramitlere dengeli olarak dağıtılmalıdır. Başka bir deyişle bütün piramitler ortalama sayıda sınır içermelidirler. Tanım 1'de tanımlanan radiant değeri kavramı bir sınırın hangi piramide ait olduğunu bulmak için kullanılır.

Tanım 1: Bir köşenin radiant değeri *p* bir binary sayıdır. Boş olarak başlar. Bütün diğer noktalar saat yönünün tersinde izlenir ve eğer bir çizgi *p* ile bağlıysa, 1 eklenir. Aksi halde, 0 eklenir

Bu yüzden, Şekil 2.30'daki sınır için nokta p_0 "000111" radiant değerine sahiptir. Aşağıdaki adımlar bir piramit için sınırın nasıl belirleneceğini gösterir.

- Sınır grafiği çizilir ve bütün noktaların radiant değerleri hesaplanır.
- Veri noktası en düşük radiant değerine sahip piramide atanır. Eğer 2 veya daha fazla piramit aynı düşük radiant değerine sahipse, en düşük indekse sahip nokta seçilir. Örneğin, p_i, p_j aynı radiant değerine sahipse, $i \leq j$ ise, p_i seçilir.

Şekil 2.30'da p_0 en düşük radiant değerine sahiptir, bu yüzden üç piramidin kesişmesiyle oluşan sınır p_0 'a atanır.

3. ÇOK BOYUTLU VERİ İNDEKSLEME YÖNTEMLERİNİN KARŞILAŞTIRILMASI

Bu bölümün ilk kısmında çok boyutlu indeksleme yöntemleri üzerinde yapılan performans testleri yardımıyla bu yöntemler arasında karşılaştırma yapılacaktır. Bu ilk bölümdeki bilgiler bu yöntemlerin geliştiricilerinin yayınladıkları makalelerden elde edilmiştir. İkinci kısımda ise bahsedilen yöntemlerden son iki yöntem Piramit Tekniği ve onu temel almış yöntem geliştirilen program aracılığıyla karşılaştırılacak ve sonuçları belirtilecektir.

3.1. Yöntemlerin Karşılaştırılması ve Elde Edilen Sonuçlar

Bu kısımda yöntemler için yayınlanan makalelerde yer alan karşılaştırma sonuçları yer alacaktır.

3.1.1. R-Ağaç – R*-Ağaç karşılaştırması

Norbert Beckmann, Hans-Peter Knegel, Ralf Schneider, Bernhard Seeger tarafından yayınlanan R* Ağaç yayınında R* Ağaç ile R Ağaç karşılaştırması yer almıştır [Beckmann vd., 1990] . Karşılaştırmanın detayları ve sonuçları şöyledir:

Performans karşılaştırması için R Ağaç, Greene'in R Ağacı ve R* Ağaç yöntemleri seçilmiştir. Buna ek olarak en popüler R Ağaç uygulaması olan doğrusal bölünen algoritma da test edilmiştir. Doğrusal R Ağaç için m=%20 seçilmiştir. Bu şekilde en iyi performansın sağlayacağı hesaplanmıştır.

Bu 4 yapının karşılaştırması için 100,000 2 boyutlu dikdörtgen içeren 6 tane veri dosyası kullanılmıştır. Her dikdörtgenin $[0,1)^2$ birim küp olduğu kabul edilir. Her veri dosyası dikdörtgenlerinin merkezlerinin dağılımına ve tripele (n, μ_{area}, nv_{area}) göre tanımlanmıştır. Burada n dikdörtgen sayısını gösterir, μ_{area} bir dikdörtgenin alanının ortalama değerini gösterir ve $nv_{area} = \sigma_{area} / \mu_{area}$ normalize edilmiş varyanstır. σ_{area} dikdörtgenlerin alanlarının varyansıdır. nv_{area} parametresi dağılımdan bağımsız olarak büyür. Dikdörtgenlerin alanlarının ortalama değerden farkı ve ortalama üst üste binme $n^* \mu_{area}$ ile elde edilir.

(F1) "Tekdüze"

Dikdörtgenlerin merkezleri iki boyutlu bağımsız tekdüze dağılımı sağlar.

(n=100,000, μ_{area} =0001, nv_{area} =9505)

(F2) "Küme"

Merkezler 640 kümededir ve her küme 1600 nesneden oluşur.

(n=99,968, μ_{area} =00002, nv_{area} =1538)

(F3) "Parsel"

Öncelikle birim kare 100,000 birim kareye ayrıştırılır. Sonra her dikdörtgenin alanı 2,5 kat arttırılır.

(n=100,000, μ_{area} =00002504, nv_{area} =303458)

(F4) "Gerçek veri"

Bu dikdörtgenler gerçek kartografi verisinden yükseltme çizgilerinin mininmum çevreleyen dikdörtgenleridir.

(n=120,576, μ_{area} =0000926, nv_{area} =1504)

(F5) "Gauss"

Merkezler 2 boyutlu bağımsız Gauss dağılımındadır.

(n=100,000, μ_{area} = 00008, nv_{area} =89875)

(F6) "Karışık-Tekdüze"

Dikdörtgenlerin merkezi 2 boyutlu bağımsız tekdüze dağılım durumundadır.

İlk olarak μ_{area} =0000101 olan 99,000 küçük dikdörtgen alınır.

Sonra μ_{area} = 001 olan1,000 geniş dikdörtgen eklenir. Sonunda iki veri dosyası birleştirilir.

(n=100,000, μ_{area} = 00002, nv_{area} =6778)

Her dosya için ((F1)-(F6)) çeşitli sorgular hazırlanmıştır. Bu sorguların tipleri şöyledir:

- *dikdörtgen kesişim sorgusu* Verilen bir S dikdörtgeni için, R∩S≠Ø olan bütün R dikdörtgenlerini bul.
- *nokta sorgusu* Verilen bir P noktası için P ∈ R olan bütün R dikdörtgenlerini bul.
- *dikdörtgen kapsama sorgusu* Verilen bir S dikdörtgeni için $R \supseteq S$ olan bütün R dikdörtgenlerini bul.

Bu dosyaların hepsi için ((F1) - (F6)) 400 dikdörtgen kesişme sorgusu uygulanmıştır. Bu sorgularda x uzantısının y uzantısına oranı 0.25'ten 2.25'e kadar değişir ve sorgu dikdörtgenlerinin kendileri birim küpte tek düze dağılmıştır. Her dorgu dosyasındaki ((Q1) - (Q4)) sorgu dikdörtgenlerinin alanı veri uzayının alanına göre %1, %0.1, %0.01, %0.001 arasında değişir.

Her bir sorgu dosyası için sorgu başına ortalama disk erişim sayısı ölçülmüştür. Performans karşılaştırması sırasında R* Ağaç ölçüm çubuğu olarak kullanılmıştır. R* Ağaç sorguları için sayfa erişim sayıları %100'e ayarlanmıştır.

R Ağaç türevlerinin performans testine hazırlanması için insert ve stor parametreleri ölçülmüştür. Burada insert her eklemedeki ortalama disk erişim sayısını ve stor ise dosyalar oluşturulduktan sonraki bellek kullanımını gösterir. Aşağıdaki tablolarda değişik dağılımlardaki sonuçlar sunulmuştur. R* Ağaç için "# accesses" sorgu başına ortalama disk erişim sayısını gösterir.

Tek Düze

	point	Kesişim 0.001	0.01	0.1	1.0	Kapsama 0.001	0.01	stor	insert
lin Gut	225.8	212 6	207 7	183 0	144.5	224.7	248 1	64.1	7 43
qua. Gut	124.8	121.9	124.4	124.1	114.2	116 7	121.9	69 5	4.27
Greene	140 0	136 1	135 4	130 1	115 1	132.8	153.8	70.3	4.67
R*-tree	100 0	100 0	100 0	100 0	100 0	100 0	100 0	75.8	4.42
# accesses	5.26	6 04	7 63	13.29	53 42	4.85	3 66		

Küme

	point	Kesişim 0.001	0.01	0.1	1.0	Kapsama 0.001	0.01	stor	insert
lin Gut	250.9	231 0	219 7	176 6	136.9	247.8	249 4	61 7	6 13
qua Gut	166 1	152.7	160 7	139 1	120.4	155.4	182.9	66.9	4.97
Greene	159 9	151.8	152.2	144.3	116.9	151 6	153.2	69.2	4.32
R*-tree	100 0	100 0	100 0	100 0	100 0	100 0	100 0	72.2	3 77
# accesses	2 00	2.26	2 95	7 13	36 0	1.86	1 58		

Parsel

	point	Kesişim 0 001	0 01	0 1	1 0	Kapsama 0 001	0 01	stor	insert
lin Gut	264 1	265 0	258 6	214.3	177.9	269 4	281.0	60.2	23 07
qua Gut	129.5	132.3	129.9	126 1	122 1	131 0	125 6	67 0	13.30
Greene	199.8	196.2	206 9	184 1	156.5	195.8	207.5	68.9	16.02
R*-tree	100 0	100.0	100 0	100 0	100 0	100 0	100 0	72.5	10 73
# accesses	5 67	6.26	7.36	13.29	36 76	5 42	4.96		

Gerçek Veri

	point	Kesişim 0.001	0.01	0.1	1.0	Kapsama 0.001	0.01	stor	insert
lin Gut	245 6	246 7	220.8	181 6	143.8	268 1	284.1	62.9	7.30
qua Gut	147.3	153 1	143.3	132.5	116 4	158.8	160 1	68 1	5 08
Greene	147.8	144 0	146.5	130.2	115.9	155 1	169.8	69 6	5 05
R*-tree	100 0	100 0	100.0	100 0	100.0	100 0	100 0	70.5	4.22
# accesses	4 78	5.29	7.35	14 65	60.84	4.08	3 08		

Gauss

	point	Kesişim 0.001	0.01	0.1	1.0	Kapsama 0.001	0.01	stor	insert
lin Gut	171 1	165 6	168 1	150 1	143.8	171 1	180.2	63.8	19 12
qua Gut	116.2	108 0	116 0	117 6	119.2	106 4	106.8	68.8	14.0
Greene	123.2	118 7	131.2	122.9	114.2	120 7	130 6	69.9	11 41
R*-tree	100.0	100.0	100 0	100.0	100 0	100.0	100 0	73.8	9 15
# accesses	4.83	5.87	7 69	10.88	46 19	4.39	3.24		

Karışık Tekdüze

	point	Kesişim 0.001	0.01	0.1	1.0	Kapsama 0.001	0.01	stor	insert
lin Gut	354.1	332.5	311 7	233 1	165.9	358 1	401 6	63 4	12 70
qua Gut	127 6	126.3	122.7	119 0	113 0	119 6	124.7	68 2	4 94
Greene	121 4	116 7	116 0	114.5	109.3	114 0	116.3	70 1	4.58
R*-tree	100.0	100 0	100 0	100 0	100 0	100 0	100 0	73 1	4 46
# accesses	4.87	5.51	7.27	13 76	52 06	4.44	3 69		

Tablo 3.1. 6 dağıtıma göre tartılmamış ortalama

Uzaysal birleştirme sorguları için ise alınan sonuçlar şu şekildedir:

	(SJ 1)	(SJ 2)	(SJ 3)
lin.Gut	296 6	229.2	257 8
qua.Gut	142 4	154 7	144 8
Greene	187.1	166.3	160 4
R*-tree	100.0	100 0	100.0

Tablo 3.2. Uzaysal Birleştirme

Tablo 3.1'de stor ve insert parametreleri için 6 dağıtıma göre tartılmamış ortalama hesaplanmıştır. Spatial join parametresi üç uzaysal birleştirme işleminin ((SJ1) – (SJ3)) ortalamasını göstermektedir. Ortalama bir sorgu performansı için query average parametresi tanımlanmıştır. Bu her dağılım için yedi sorgu dosyası üzerine ortalama ve bütün altı dağılım için ortalamadan oluşur.

	query average	spatial join	stor	insert
lin Gut	227 5	261 2	62 7	12 63
qua.Gut	130 0	147 3	68 1	7 76
Greene	142 3	171 3	69 7	7 67
R*-tree	100 0	100 0	73 0	6 13

Tablo 3.3. Bütün dağılımlar üzerinde tartılmamış ortalama

Query avarage parametresindeki bilgi kaybı parametrenin her bir veri dosyası (F1) – (F6) için tüm yedi sorgu dosyası ortalaması olarak ayrı gösterildiği Tablo 3.4'de ve sorgu ortalaması parametresi her sorgu için (Q1) – (Q7) altı veri dosyası üzerinde ortalama olarak ayrı olarak gösterildiği Tablo 3.5'de daha azdır.

	gaussian	cluster	mix uni	parcel	real data	uniform
lin Gut	164 3	216 0	308 1	247.2	227.2	206 6
qua Gut	112 9	153 9	121 8	128 1	144.5	121 1
Greene	123 1	147 1	115 5	192 4	144.2	134 8
R*-tree	100 0	100 0	100 0	100 0	100 0	100 0

Tablo 3.4. Dağılıma bağlı yedi tip sorgu üzerine tartılmamış ortalama

	point	intersection				enclosure		stor	insert
		0.001	0.01	0.1	1.0	0.001	0.01		
lin. Gut	251.9	242.2	231 1	189.8	152 1	256.5	274 1	62.7	12.63
qua. Gut	135.3	132.4	132.8	126.4	117 6	131.3	137.0	68.1	7.76
Greene	148 7	143.9	148 0	137.7	121.3	145.0	155.2	69 7	7 67
R*-tree	100 0	100 0	100.0	100 0	100 0	100 0	100.0	73.0	6.13

Tablo 3.5. Sorgu tipine bağlı tüm altı dağılım üzerine tartılmamış ortalama

Sonuçlarda görebildiğimiz kadarıyla ilk olarak R* Ağacın her denemede R Ağaçtan daha üstün bir performans gösterdiğini söyleyebiliriz. Aşağıdakiler R* Ağacın R Ağaç üzerindeki üstünlüğünü ortaya çıkarmaktadır.

- R* Ağaç her sorgu dosyası ve her veri dosyası için diğer türevlere göre daha az disk erişme gerektirdiğinden en sağlam yöntemdir. Başka türlü de şöyle denebilir deneylerde R* Ağacın en yüksek dereceyi almadığı bir durum rastlanmamıştır.
- R* Ağaçta küçük sorgu dikdörtgenlerinde elde edilen etkinlik kazancı geniş sorgu dikdörtgenlerindekinden daha fazladır. Çünkü daha büyük sorgular için bellek kullanımı daha önemli hale gelir.
- R * Ağaç kullanıldığında maksimum performans kazanımı R Ağacın %400'ü kadardır. Bu oran Greene'in R Ağacının %200'üne, karesel R Ağacın %180'ine denk gelir.
- Beklenildiği gibi R * Ağaç en iyi bellek kullanımına sahiptir.

3.1.2. R*-Ağaç – TV-Ağaç karşılaştırması

King-lp Lin, H.V. Jagadish ve Christos Faloutsos yayınladıkları TV Ağaçla ilgili makalede o ana kadar en hızlı indeksleme yöntemi olan R* Ağaçla kendi geliştirdikleri TV Ağacın karşılaştırması vardır [Lin vd., 1995]. Karşılaştırmanın detayları ve sonuçları şöyledir:

Testler için kelimeler kullanılmıştır. Doğru ve hatalı yazımlar oluşturulup en yakın ya da birebir eşleşen kelimeler bulunmaya çalışılmıştır. Bunun için tam eşleşme ve aralık sorguları kullanılmıştır. Her kelime 27 boyutlu bir v vektörüyle temsil edilmiştir. Nesneler için çevreleyen şekiller olarak L_1 küreleri kullanılmıştır.

Karşılaştırma sırasında düğümlerin minimum doluluk oranı %45, tekrar eklenecek eleman sayısı oranı da %30 olarak seçilmiştir. Bu değerler R* Ağaçla karşılaştırmanın daha düzgün olması için hesaplanarak tespit edilmiştir.

Testlerde 2,000'le 16,000 arasında sözlükten rastgele seçilmiş kelimeler kullanılmıştır. Çeşitli parametreler de değiştirilerek kullanılmıştır. α (aktif boyutlar) 1 ile 4 arasında, aralık sorgusunun toleransı Є 0'la (tam eşleşme) 2 arasında değiştirilerek kullanılmıştır.

3.1.2.1. Sonuçlar

İndeks Yaratma İndeks yaratma için diske erişme sayıları ölçülmüştür. Sonucunda TV Ağaçta indeks yaratmanın R* Ağaca göre daha az maliyetli olduğu tespit edilmiştir. Bunun nedeni TV Ağacın R* Ağaca göre daha yüzeysel olmasıdır. Bu yüzden daha düğüm diskten çekilir ve diske daha az potansiyel güncelleme yapılır. Tablo 3.6 indeks yaratma sırasında diske erişim sayılarını vermektedir.

Dictionary size	Disk access per insertion	
	R^*-tree	TV-2 tree
4,000	5.25	4.75
8,000	5.51	5.21
12,000	6.19	5.28
16,000	6.50	5.35

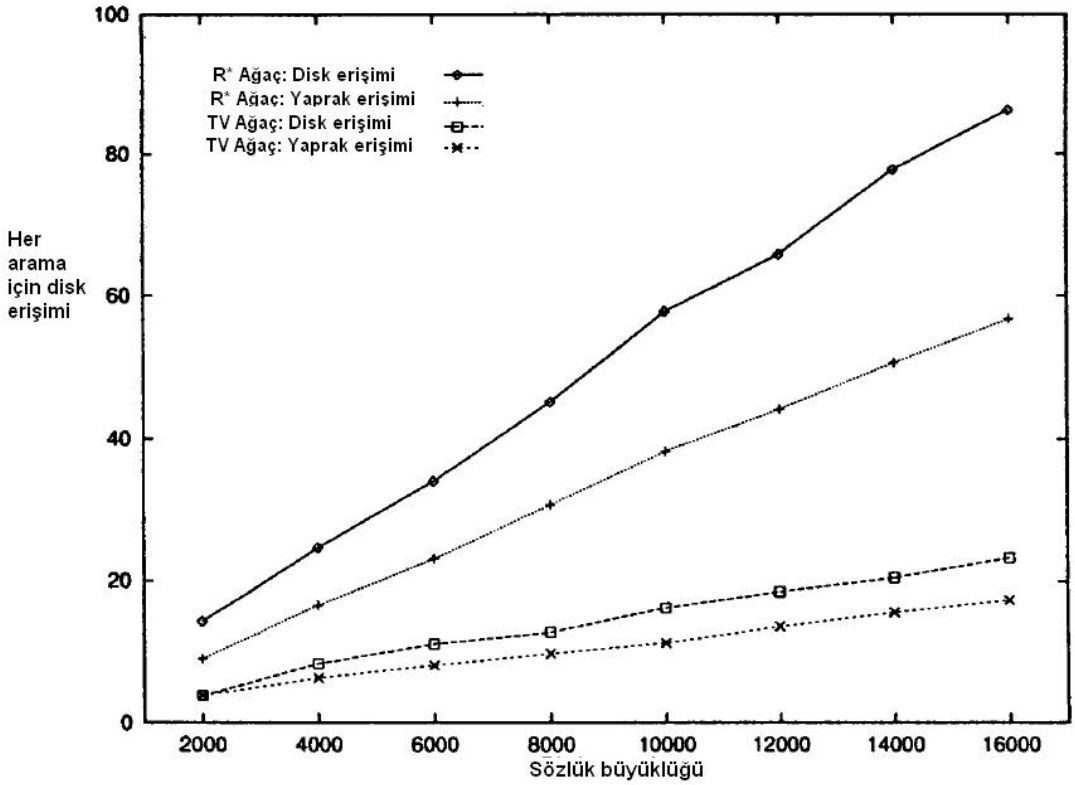
Tablo 3.6. Her eklemedeki disk erişimi – nesne boyutu 100 byte.

4,000 ile 8,000 arasındaki TV Ağaçtaki büyük atlama TV Ağacın yeni bir düzeye geçmesindedir. Yine de TV Ağaç R* Ağaçtan hep bir düzey aşağıdadır.

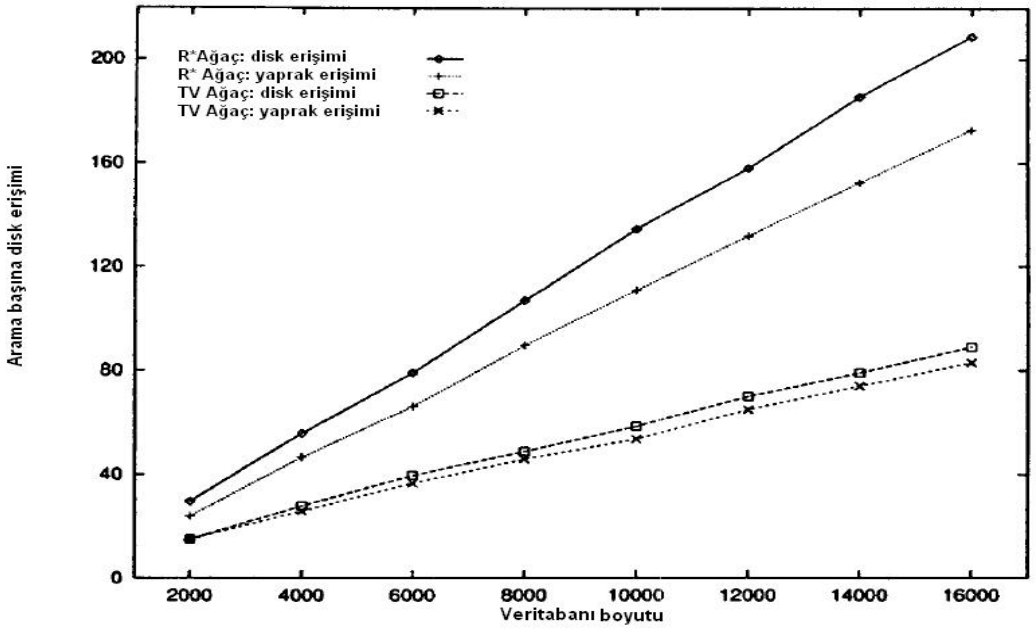
Arama Şekil 3.1'den 3.3'e kadar disk/sayfa erişim sayısı veritabanının bir fonksiyonu gibi gösterilmiştir. Yaprak erişimlerinin sayısı her zaman alçak eğridedir. 4,000 sayfa boyutu kullanılmıştır. Sonuçlar 100 bytelık nesneler üzerinedir.

Şekillerden görülebildiği gii TV Ağaç R* Ağacı performans bakımından açıkça geçmektedir. Toplam disk erişiminde %67-73 arası bir kazanım benzer ölçüde bir kazanım yaprak erişimlerinde vardır. Aralık sorgularında bu kazanım daha fazladır.

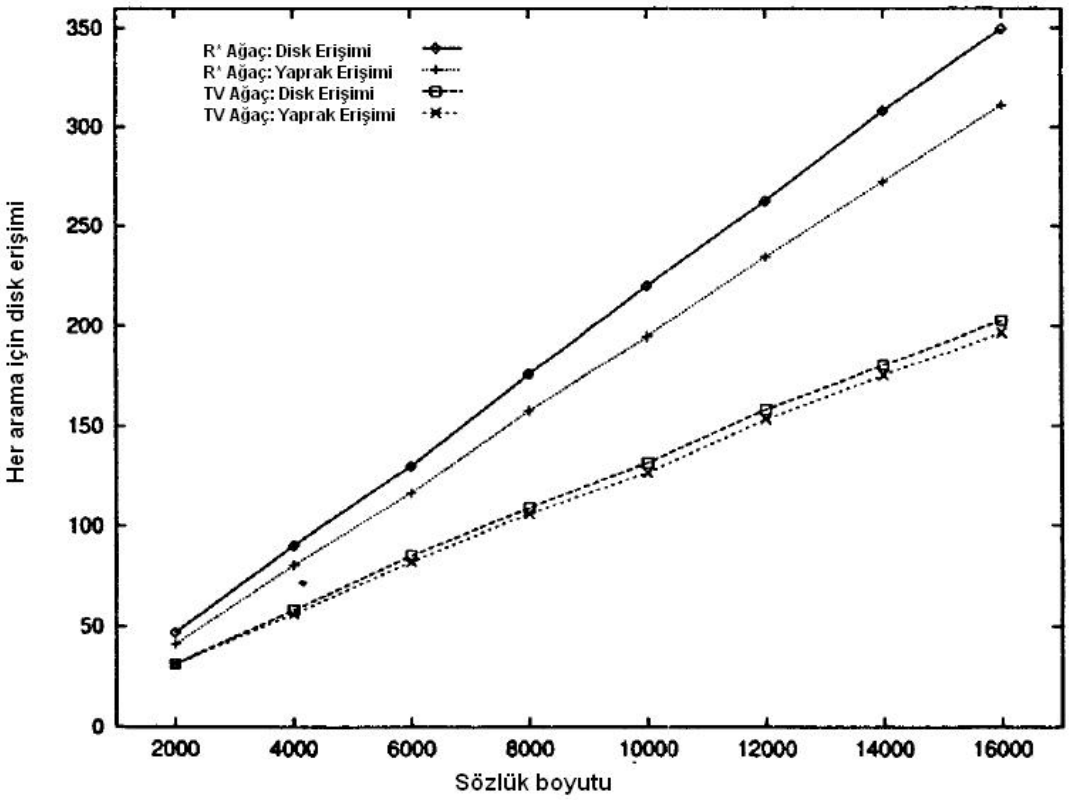
Buna ek olarak kazanımlar veritabanı büyüklüğüyle artar. Bu da Tv Ağacın gittikçe daha etkili olmaya başladığını gösterir. Veritabanı 2,000den 16,000 elemana genişletilirken yaprak erişimi kazanımları düzenli olarak artar: birebir eşleşme sorguları için %67'den %73'e, Є=1 olan aralık sorguları için %50'den %58'e, Є=2 olan aralık sorguları için %33'ten %42'ye



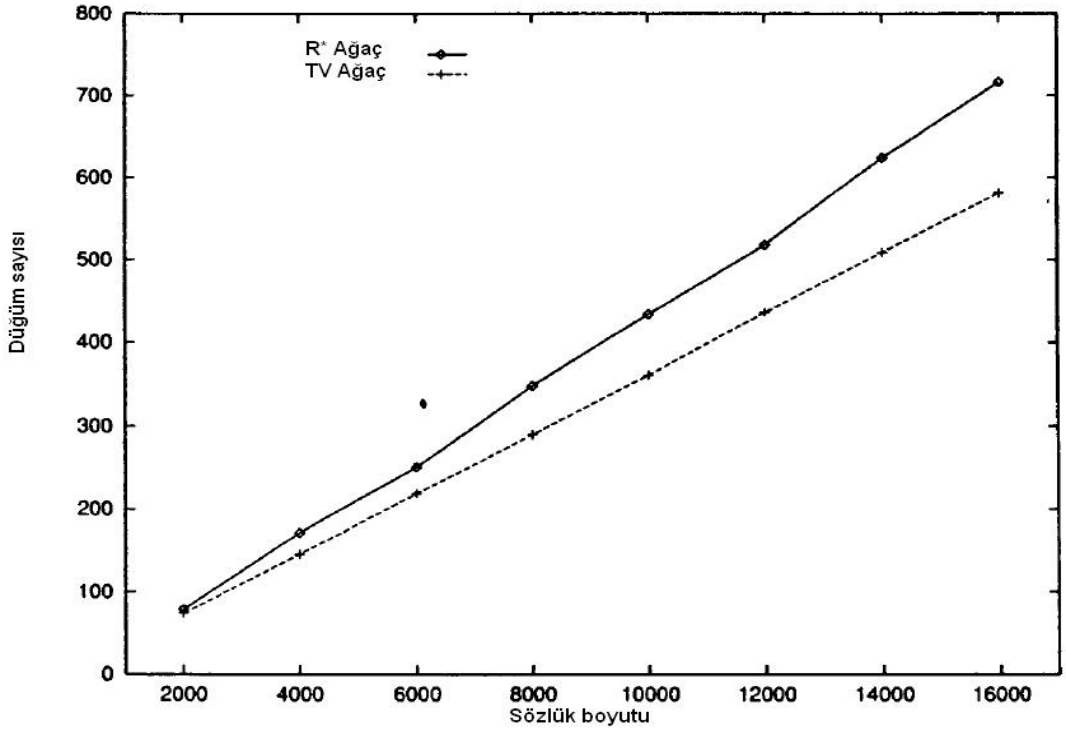
Şekil 3.1. Disk/yaprak erişimi ve veritabanı büyüklüğü – birebir eşleşme sorguları



Şekil 3.2. Disk/yaprak erişimi ve veritabanı büyüklüğü – aralık sorguları (tolerans=1)



Şekil 3.3. Disk/yaprak erişimi ve veritabanı büyüklüğü – aralık sorguları (tolerans=2)



Şekil 3.4. Boşluk gereksinimlerinin karşılaştırması

Yaprakların diskte depolandığını varsaysak bile TV Ağaç R* Ağaçtan performans bakımından üstündür.

Boşluk Gereksinimlerinin Karşılaştırması Şekil 3.4 ağaçlardaki düğüm sayısını gösterir. TV Ağacın daha az sayıda düğüme ihtiyacı vardır. Kazanım%15-20 arasındadır.

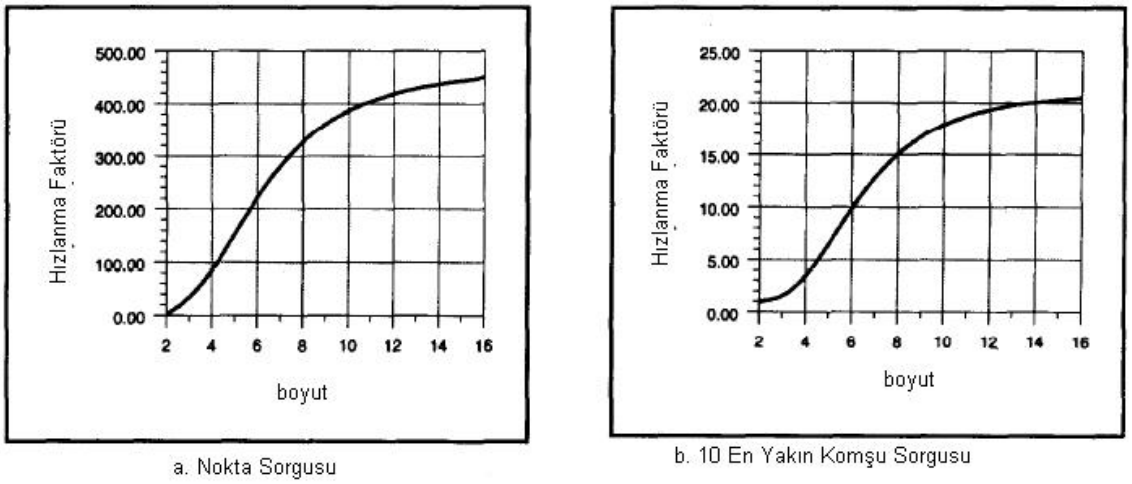
Nesne boyutları birbirine eşit olduğunda her iki yapının da yaprak düğümleri sayısı birbirine yakındır. Bu da şunu gösterir ki, TV Ağaç'ta çoğu kazanım iç düğümlerdendir. Yani yaprak olmayan düzeylerde daha düşük tampon kullanılır.

3.1.3. X-Ağaç – TV-Ağaç – R* Ağaç karşılaştırması

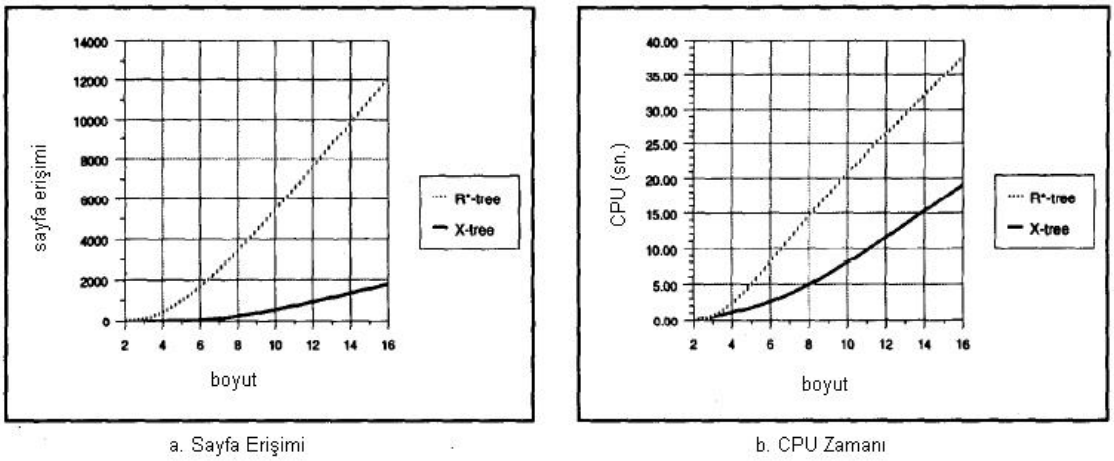
Stefan Berchtold, Daniel A. Keim ve Hans-Peter Kriegel'ın ortaya koyduğu yöntem olan X Ağaç yöntemi sunumunda bir X Ağacın TV Ağaca ve R* Ağaca göre performansı incelenmiştir [Berhtold vd., 1996]. Performans testleri ve sonuçları şöyledir:

Test verilerinde Fourier noktalarından elde edilmiş boyutları 2,4,8,16 arasında değişen gerçek nokta veri, gerçek CAD nesnelerini temsil eden boyutları 2,4,8,16 arasında değişen uzaysal veri ve tek düze dağılmış noktalardan oluşan boyutları 2,3,4,6,8,10,12,14,16 arasında değişen sentetik veriler kullanılmıştır. Blok büyüklüğü her bir yöntem için aynıdır ve bu da 4 Kbyte dır. Bütün testlerde X Ağaca eklemelerin TV Ağaca ve R* Ağaca eklemelerden daha hızlı olduğu gözlemlenmiştir.

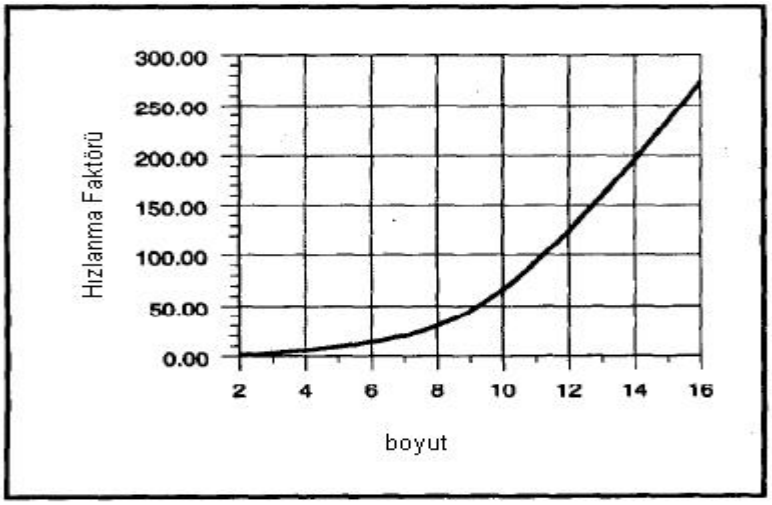
Öncelikle değişken boyutlardaki veri tabanlarında X Ağaç test edilmiştir. Şekil 3.6'da gösterilen testte, 100 MByte'lık tekdüze dağılmış nokta veriler kullanılmıştır. Veri parçaları 8.3 milyon ve D=2' den (D, boyut), 1.5 milyon D=16'ya değişmiştir. Şekil 3.6'da X Ağacın R* Ağaca göre nokta sorguları için arama zamanı hızlanması gösterilmektedir. Boyutluluk arttıkça arama hızları arasındaki fark artar. Daha düşük boyutsallıkta bu fark daha azdır. Yüksek boyutsallıkta R* Ağaç'ta üst üste fazlalaşır ve böylede R* Ağaç birçok dizin sayfasına ulaşmak durumunda kalır. Bu da G/Ç süresini arttırır.



Şekil 3.5. Gerçek Nokta Veride R* Ağaç Üzerine X Ağaç Hızlanması

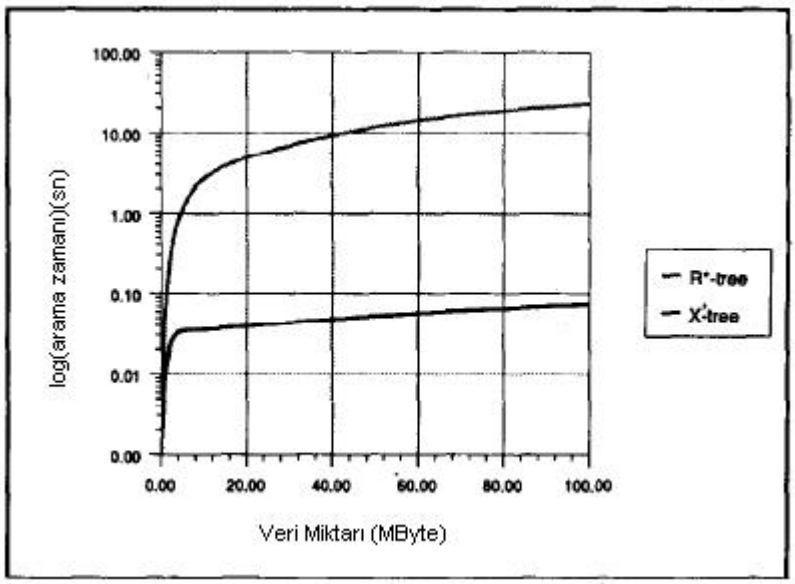


Şekil 3.6. Gerçek Veriyle Sayfa Erişim Sayısı ve CPU Zamanı (70 MByte)



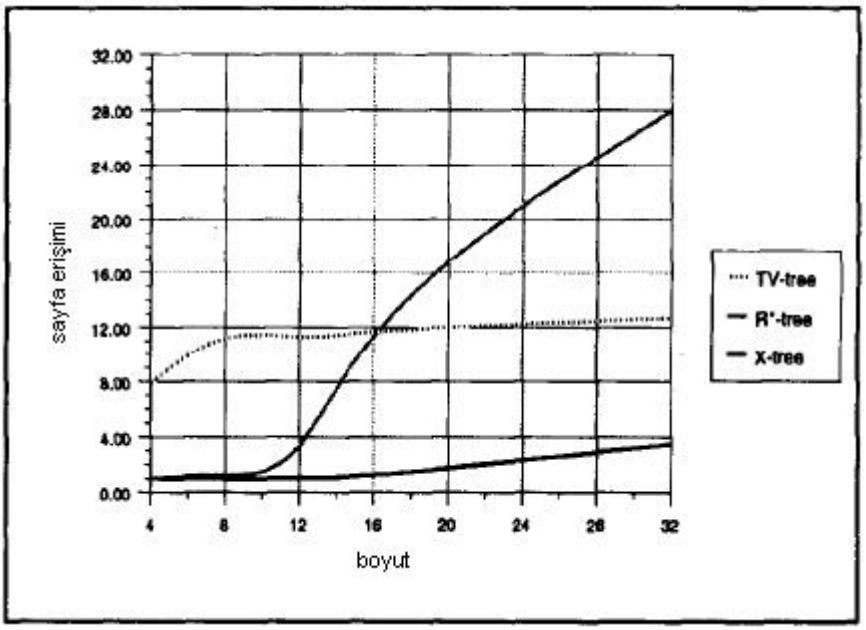
Şekil 3.7. Nokta Sorgularında R* Ağaç Üzerine X Ağacın Hızlanması (100 MByte Sentetik Nokta Veri)

Gerçek veri olarak 70 MBytelık poligon şekillerini temsil eden Fourier verileri kullanılmıştır. Sonuçlar Şekil 3.5'de görüldüğü gibi toplam arama zamanı hızlanması nokta sorgularda tekdüze dağıtılmış verilerden daha fazladır. Bu sonuç üst üste binme eğrileri üzerinde bulunan gerçek verilerin yüksek üst üste binmeye maruz kaldığını göstermektedir. Aynı zamanda en yakın 10 komşu sorgusuna göre performans ölçülmüştür. Bu algoritma TV Ağaçta olmadığından burada R*Ağaçla X Ağaç karşılaştırılmıştır. Testlerden elde edilen sonuçlar gösterir ki en yakın komşu sorgularında hızlanma D=2 için 10 ve D=16 için 20 arasındadır. En yakın komşu sorguları sıralama gerektirdiğinden CPU zamanı daha fazladır. Şekil 3.6'da X Ağaç ve R* Ağacın en yakın komşu sorgularında sayfa erişim sayıları ve CPU zamanı verilmiştir. Şekil X Ağacın R* Ağaçtan daha iyi bir performans sergilediğini göstermektedir.



Şekil 3.8. Nokta Sorguları İçin Değişken Veritabanı Boyutlarında Toplam Arama Zamanı

Şekil 3.8 nokta sorguları için geçen toplam zamanın veritabanının boyutuna bağlı olduğunu gösterir. (D=16) Bu şekil X Ağacın arama zamanının R* Ağacınkinden 2 düzey yüksek olduğunu göstermektedir. Hızlanma 20 MBytelık veritabanında 100 civarından 100 MBytelık veritabanında 270 civarına kadar artmıştır.



Şekil 3.9. Sentetik Veri ile X Ağaç, R* Ağaç ve TV Ağacın Karşılaştırılması

3 yöntemin de dahil olduğu karşılaştırmalar da yapılmıştır. Bu karşılaştırmaların sonuçları Şekil 3.9'da görülmektedir. X Ağacın TV Ağaç üzerinde hızlanması küçük veritabanlarında bile 4 ile 12 arasında değişir. İlginç bir nokta da boyutsallık 16'dan küçük olduğunda R* Ağacın performansının TV Ağaçtan daha yüksek oluşudur.

3.1.4. Piramit Tekniği – X Ağaç karşılaştırması

Stefan Berchtold, Chritian Böhm, Hans-Peter Kriegel tarafından 1998 yılından yayınlanan ve Piramit Tekniğinin sunulduğu makalede Piramit Tekniği başta X Ağaç olmak üzere Hilbert R Ağaç ve sıralı arama yöntemleriyle karşılaştırılmıştır [Berchtold vd., 1998]. Bu karşılaştırmaların yapılış şekli ve alınan sonuçlar şöyledir:

Karşılaştırma yöntemlerinden Hilbert R Ağacının seçilmesinin nedeni Hilbert eğrileri ve uzayı dolduran diğer eğriler B Ağaç ile birleşme için kullanılabileceği düşüncesidir. Piramit Tekniği'de bu tip bir yöntem olduğundan Hilbert R Ağaç Piramit Tekniği'ne iyi bir rakip olacağı düşüncesiyle seçilmiştir.

Son zamanlarda artan indeks tabanlı sorgu işleme yöntemlerinin yüksek boyutlarda pek işe yaramadığı ancak sıralı aramanın daha iyi olduğu yönünde fikirler üzerine testler için sıralı arama da seçilmiştir. Testler sonucunda sıralı aramanın yüksek boyutlarda gerçekten Hilbert R Ağaç ve X Ağaç'tan daha iyi ancak Piramit Tekniği'nin sıralı armadan daha iyi bir performans sergilediği görülecektir.

Testler hem gerçek hem de sentetik veriler için yapılmıştır. Sorgular için belirli bir seçicilik oranında aralık sorguları kullanılmıştır. Çünkü aralık sorguları diğer sorgular için temel görevi yaparlar. Sorgu dikdörtgenleri uzaydan rastgele seçilmiş ve bu dikdörtgenlerin dağılımı veri kümesinin dağılımı gibidir.

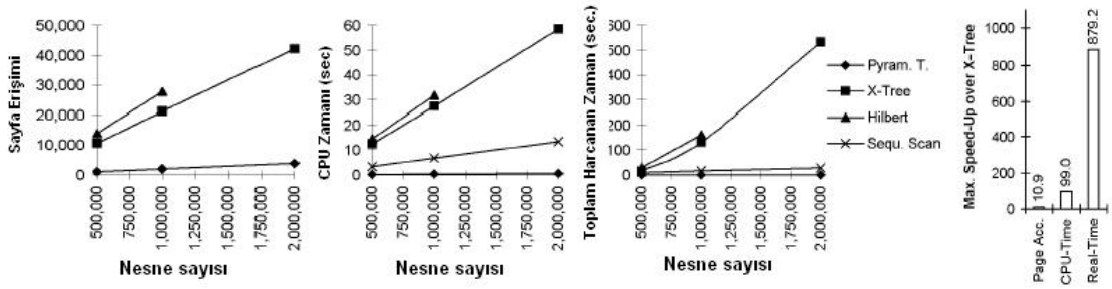
3.1.4.1. Sentetik veri kullanılarak yapılan değerlendirmeler

Suni veri kümesi 100 boyutlu veri uzayında 2.000.000 tane uniform olarak dağılmış noktadan oluşur. Ham veri dosyası disk üzerinde 800 Mbyte yer tutmuştur. Uniform olarak dağılmış nokta kümelerinin ana avantajı, sorgunun mantığını etkilemeden bazı boyutları atarak boyutluluğu azaltabilmektir. Ham veri dosyalarını kullanarak indirgeme, seçim ve inşa etme ile çeşitli boyutlarda ve çeşitli sayılarda nesne içeren dosyalar yaratılmıştır.

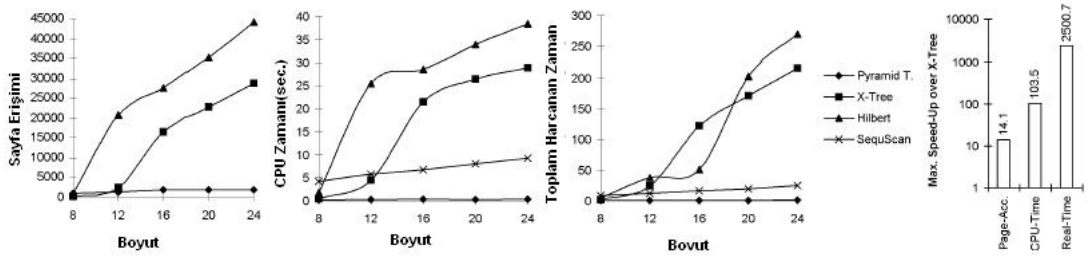
İlk deneylerde (Şekil 3.10), çeşitli sayılarda nesnelerde performans değişiklikleri ölçülmüştür. %0.1 seçicilik, 16 boyutlu veri uzayında ve 500.000den 2.000.000'a kadar değişen sayıda nesneyle aralık sorguları yapılmıştır. Hilbert R Ağaç limitli hafıza yüzünden maksimum 1.000.000 nesne için inşa edilmiştir.

Bu deneyde bütün indekslerin dosya boyutu 1.1 Gigabyte olmuştur. Sayfa boyutu 4096 Byte ve etkili sayfa kapasitesi her sayfa için 41.4 nesneye ulaşmıştır. Şekil 3.10 sayfa erişim sayısı, CPU zamanı ve toplam harcanan zaman (CPU zamanı ve diskte i/o işlemleri zamanı) bakımından sorguların sonuçlarını gösteriyor. Sayfa erişimlerine göre hızlanma hemen hemen sabit ve 9.78 ve 10.81 aralığındadır. CPU zamanında hızlanma sayfa erişiminde hızlanmadan daha fazladır, fakat veritabanı boyutuyla birlikte çok az artar. Bunun nedeni B+ Ağacın ikiye bölme veya aralık arama algoritmalarını

uygulayarak nesneleri karşılaştırmak için sayfa aramalarını kolaylaştırmasıdır. Ancak, en önemli hızlanma toplam geçen zamandadır. Faktör 53'ten başlayıp çabuk gelişir ve en geniş veritabanında en yüksek değere ulaşır: Piramit Tekniği 2 milyon nesneyle yapılan aralık sorgularında X Ağaçtan 879 kat hızlıdır. B+ Ağaç üzerinde aralık sorgusu işleme X Ağaç üzerindekinden çok daha etkilidir, çünkü ağacın büyük kısımları veri sayfalarındaki kenar bağlantılarını takip ederek etkili bir biçimde kat edilir. Bundan başka, pahalı disk kafası hareketlerine neden olan uzun uzaklık arama işlemleri, daha iyi disk kümeleme olanaklarından dolayı daha düşük olasılığa sahiptir. Şekil 3.10'ın sağ tarafındaki çubuk grafiği bu deneyde en yüksek hızlanma faktörlerini özetler.



Şekil 3.10. Veritabanı Boyutu Üstünde Performans Değişikliği



Şekil 3.11. Veri Uzayı Boyutu Üzerinde Performans Değişikliği

Şekil 3.11'de gösterilen deneyde, veri uzayı boyutunun performans üzerinde etkisi belirlenmiştir. Bu amaçla boyutları 8,12,16,20 ve 24 olan verilerden oluşan 5 veri dosyası yaratılmıştır. 1.6 Gigabyte disk alanı harcanmış ve her sayfa için boyut 4096 Byte yapılmıştır. Etkili veri sayfası kapasitesi boyuta bağlı olarak değişmiş ve 23 ile 83 arasında değerler almıştır. Seçicilik %0.01 seçilmiştir.

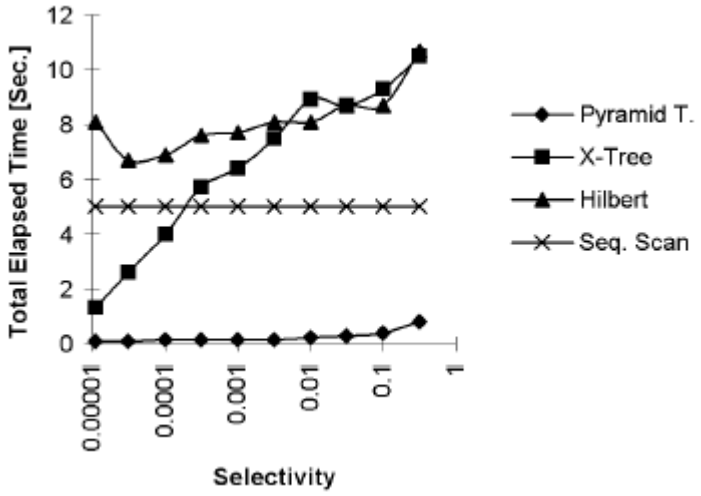
X Ağacı kullanarak sorgu işlemede verimin boyut arttıkça düştüğü gözlemlenir. 16 boyutlu veri uzayından sonra sayfa erişimleri indeks boyutuyla doğrusal olarak artar.

Daha kötü performansı Hilbert R Ağacı gösterir. Artan boyutta performanstaki bozulma Piramit Teknikte gözlemlenebilir değildir. Erişilen sayfalar tüm sayfalara oranla büyüyen boyutla birlikte azalır.(8 boyutluda %7.7 den 24 boyutluda %5.1 e düşer). Deneyler sonucunda X-Ağaç için hızlanma faktörü sayfa erişim sayısı için 14,1, Cpu zamanı için 103,5 olmuştur. Piramit Tekniği X Ağaçtan toplam harcanan zamanda 2500,7 kat daha hızlıdır.

3.1.4.2. Gerçek Veri Kullanarak Yapılan Değerlendirmeler

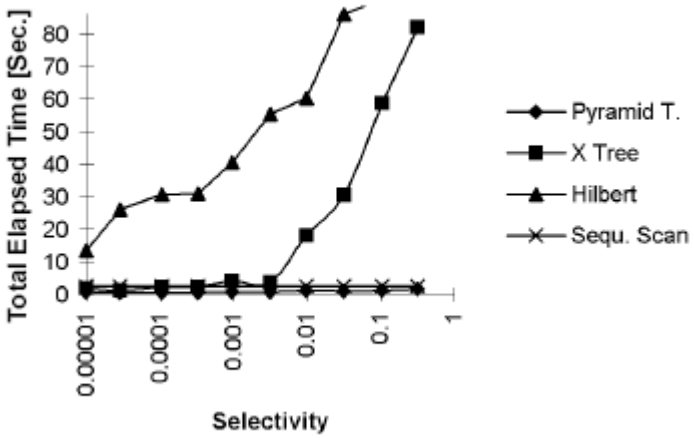
Bu deney serilerinde iki değişik uygulama bölgesinden (bilgi geri çekme, depolama) gerçek veri kümeleri kullanılmıştır.

İlk veri kümesi WWW sayfalarından alınan geniş metin veritabanlarının alt dizgilerini tanımlayan metin açıklayıcıları içerir. Bu metin açıklayıcıları 16 boyutlu uzayda 300,000 noktaya çevrilir ve normalize edilir. Aralık sorgularının seçiciliği 10-5 ten %31e kadar değiştirilir ve sorgu işleme süresi ölçülür. Piramit Teknik Şekil 3.10'da gösterildiği gibi açıkça diğer tekniklerden performans olarak üstün çıkmıştır. En yüksek hızlanma faktörü 51 olarak gözlemlenebilir. Bu deney gösterdi ki, Piramit Teknik diğer tekniklerden herhangi bir seçicilikte daha üstündür.



Şekil 3.12. Metin Verisi Üzerinde Sorgu İşleme

Deneylerin son serisinde veri depolamadan alınan veri kümeleri kullanılır. Burada da Piramit Teknik diğer indeks yapılarından daha iyi performans gösterir. İşte sonuçlar:



Şekil 3.13. Veri Depolama Üzerinde Sorgu İşleme

Deneyleri özetleyecek olursak, şu gözlemler dile getirlebilir:

1) Hemen hemen hiperküp şekilli sorgularda Piramit Teknik diğer tekniklerden daha üstün performans gösteröiştir.

2) Kötü seçiciliğe sahip sorgularda da Piramit Teknik diğerlerinden daha iyi sonuç vermiştir.

3.1.5. Piramit Tekniği Tabanlı Bir Yöntem - Piramit Tekniği karşılaştırması

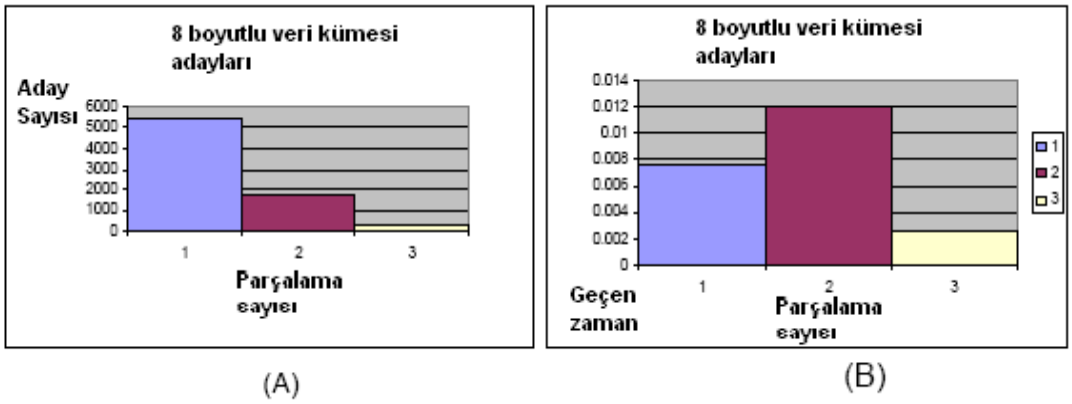
2005 yılında Jiyuan An, Yi-Ping Phoebe Chen, Qinying Xu ve Xiaofang Zhou tarafından "Piramit Tekniği" temel alınarak geliştirilen öçok boyutlu veri indeksleme yönteminde Piramit Tekniği ile bu tekniğin karşılaştırması yapılmış ve sonuçlara yer verilmiştir [An vd., 2005]. Bu karşılaştırma detayları şöyledir:

3.1.5.1. Aday kümelerin büyüklüğüyle CPU zamanı arasındaki ilişki

Piramit ağacı da içeren yüzey tabanlı indeks tekniği benzerlik sorgusu için ilişkili olmayan veri noktalarını süzer. Daha küçük aday kümesi istenir. Çünkü kesin cevapları elde etmek için her adayı arıtılır ve bu da G/Ç ve CPU maliyetine neden olur. Eğer veri uzayı (ya da hiperküp) birden fazla kere bölünmüşse, indeks anahtarı daha doğru ve aday kümesi daha küçük olur. Ancak daha önce de bahsedildiği gibi daha fazla sorgu yapılmalı. Bu anahtarların doğruluğu için sorgu zamanını uzatmak anlamına gelir.

Kaybı göstermek için gerçek bir veri kümesi kullanılmıştır. Gerçek veri bir renkli resimlerden elde edilmiş bir renk histogramıdır. Boyut 8 ve veri kümesinin büyüklüğü 100.000'dir. Sorgu veri noktaları veri kümesinden rastgele seçilmiştir. Sorgu aralığı %1'dir. Daha fazla sorgu aralığı çok fazla cevap döndürür, çok küçük sorgu aralıkları sadece kendini döndürür. %1 sorgu aralığı 10-100 civarı cevap döndürür ve bu da benzerlik araması için yeterlidir.

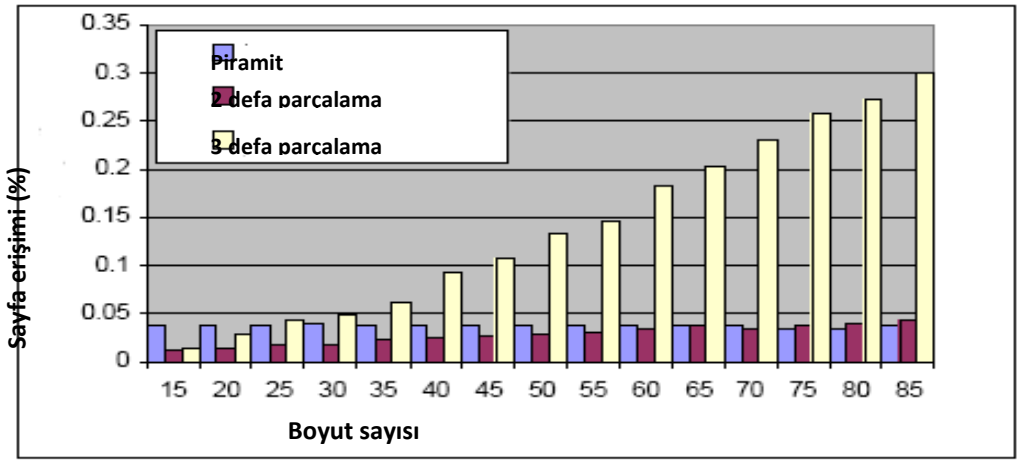
Şekil 3.14 üç farklı bölünmeye göre adayların sayısını ve CPU zamanını gösterir. Eğer ilk parçalamada durulursa, aday kümesi daha büyük olur. Gerçek cevabı almak için adayları arıtma daha fazla CPU zamanı gerektirir. Hiperküp 3 defa parçalanırsa, küçük bir aday kümesi elde edilir. Ancak aralık sorgusuyla örtüşen çok fazla piramit vardır. B^+ ağaç için daha fazla CPU zamanı harcanmak zorundadır. Sonuçta, gerçek bir veri kümesi için 2 defa parçalamanın en iyisi olduğu bulunmuştur. Bunun CPU zamanı Şekil 3.14 (B)'de görüldüğü gibi en küçüktür.



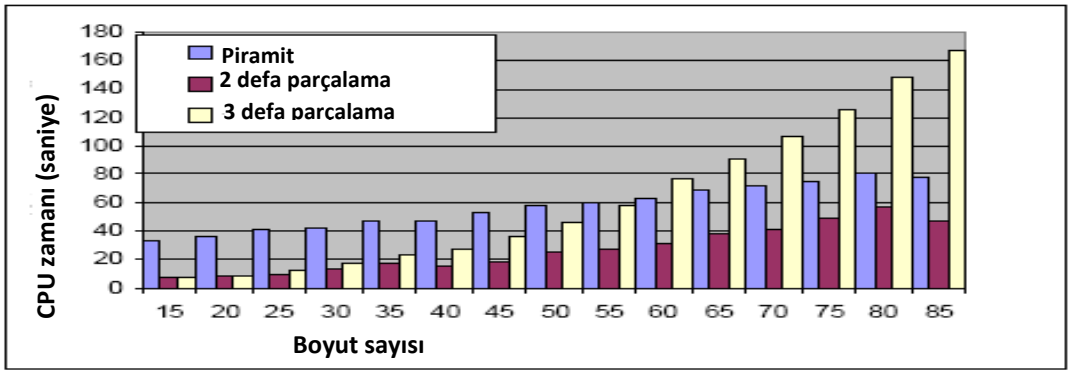
Şekil 3.14. Adayların sayısı ve gerçek veri kümesi için G/Ç maliyeti

3.1.5.2 Sayfa erişim sayısı ve CPU zamanı

Bu indeksleme yöntemini farklı boyutlu uzaylarda değerlendirmek için, 15, 20, 25, …, 85 boyutlu veri kümeleri yaratılmıştır. Veri [0,1) arasına normalize edilmiştir. 100,00 veri kullanılmıştır. Aralık sorguları veri uzayının %2'si olarak alınmıştır. Aralık sorgusu 10-50 arasında uygun cevap döndürür. B^+ ağacın düğüm büyüklüğü 8Kya ayarlanmıştır. Benzerlik aralık sorgusu için 1000 veri noktası rastgele seçilmiştir.



(A)



(B)

Şekil 3.15 (A) Sayfa erişimi sayısı (B) Aralık arama zamanı

Şekil 3.15 (A) sayfa erişim sayısı ile boyut sayısı arasındaki ilişkiyi gösterir. Dikkat edilmelidir ki hiperküp parçalaması "1 defa" olduğunda piramit-ağaç indeks özel bir durumdur, piramidin tabanı (d – 1) boyutlu hiper düzlemdir. Sayfa erişimi sayısından, "2 kere parçalama"nın diğerlerinden daha etkili olduğu bulunabilir. Şekil 3.15 (B) üç uzaysal indeks yapısı için arama zamanını göstermektedir. Açıktır ki, yine "2 kere parçalama" yöntemi eni iyi performansı verir, "1 kere parçalama" dan (örneğin Piramit ağaçları) çok daha iyidir.

3.2. Piramit Tekniği Uygulamasıyla Yapılan Karşılaştırmalar

Son zamanlarda geliştirilen yüksek boyutlu veri indeksleme yöntemlerinden Piramit Tekniği ve Piramit Tekniğini temel alan yeni yöntem üzerine bir uygulama yapıldı. Bu uygulamanın amacı hem bu iki yöntemi karşılaştırmak hem de bu yöntemlerin tek boyutlu yöntemlere göre etkinliğini ölçmektir. Öncelikle uygulamaya bir göz atalım.

3.2.1. Piramit Tekniği uygulaması

Piramit Tekniği ve Piramit Tekniğini temel alan yeni yöntem üzerine bir uygulama geliştirildi. Bu uygulamanın amacı programlamada Piramit Tekniği ve geliştirilmiş Piramit Tekniği ile günümüzdeki veritabanı yönetim sistemlerini karşılaştırmaktı. Yukarıda verilen Piramit Tekniği özetinde $i<d$ halinde kesişen p_i piramitlerinin hesaplaması yapılmıştı. Bu uygulamada bu hesaplama $i>d$ şeklindeki p_i piramitlerine genişletilmiştir. Ayrıca tek boyutlu indeksleme yapısı olarak B+ Ağaç kullanan SQL Server 2005 indeksleme yapısı kullanılmıştır.

Uygulama 7, 15 ve 25 boyutlu bir veri uzayları için tasarlanmıştır. Veri yapısı [0,1] uzayındaki veri uzayının her bir boyutuna denk gelen değerler, Piramit Tekniği anahtar değeri ve Yeni Piramit Tekniği anahtar değerinden oluşmaktadır. Program veritabanına kayıt işleme ve bu kayıtlar üzerinde hiperküp sorguları işleme yapar. Hiperküp sorguları sonrasında her bir tekniğin milisaniye cinsinden sorgu işleme süresini veri tabanına yazar. Bu bölümde her bir veri uzayı için rastgele olarak oluşturulan 10000 adet kayıttan oluşan veri tabloları kullanılmıştır.

Ekleme işleminde gelen bir kayıt piramit değeri ve yeni piramit anahtar değeri hesaplanıp kayıt bunlarla birlikte tabloya yazılır.

Hiperküp sorgusu işleminde ard arda kaç rastgele sorgu gönderileceği ve bu sorguların seçiciliğinin kaç olduğunun girilmesi gerekir. Daha sonra belirtilen sayıda ve

seçicilikte sorgular oluşturulup bu sorgular doğrusal arama, B+ Ağaç, Piramit Tekniği ve Yeni Piramit Tekniği yöntemleriyle karşılaştırılır.

3.2.2. Karşılaştırma ve sonuçlar

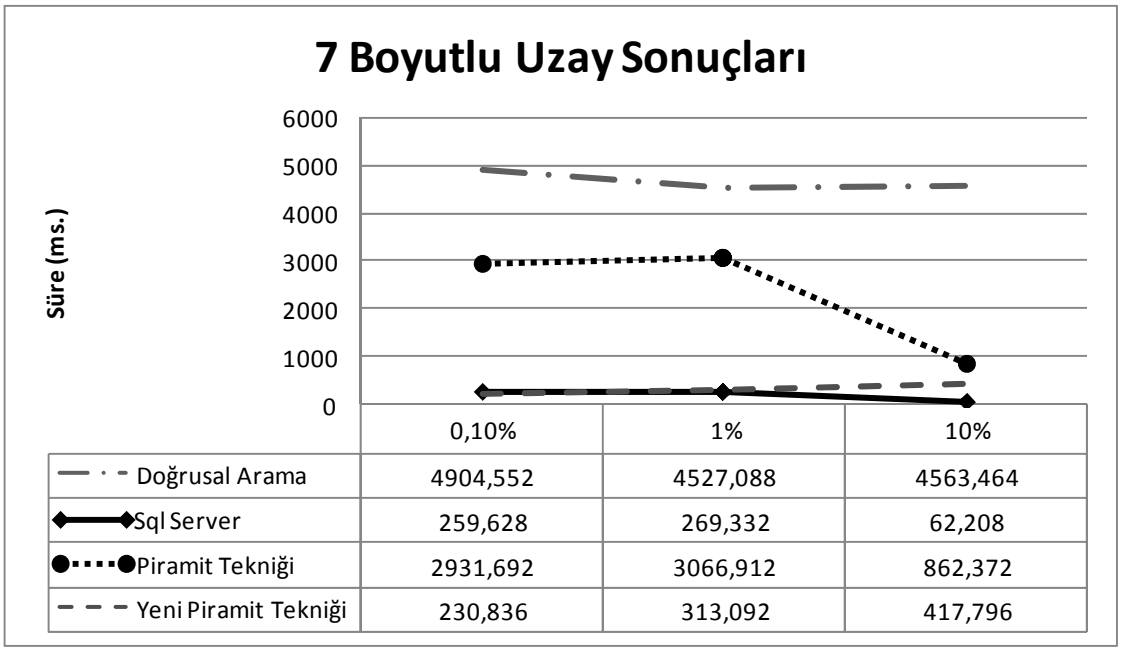
Karşılaştırma kısmında 4 teknik denenmiştir. Bunlar:

- Piramit Tekniği
- Piramit Tekniği Temelli Yeni Yöntem
- Lineer Arama
- B+ Ağaç (SQL Server 2005 ile)

Program Visual Studio C# da yazılmış ve testlerin yapıldığı bilgisayar Pentium 4, 3.00 GHz' dır.

İlk karşılaştırmalar %0,1, %1 ve %10 seçiciliğe sahip ve 7 boyutlu uzaydaki hiperküp sorguları için yapılmıştır. Her bir seçicilikteki performanslar rastgele yaratılmış 250 sorguyla denenmiştir.

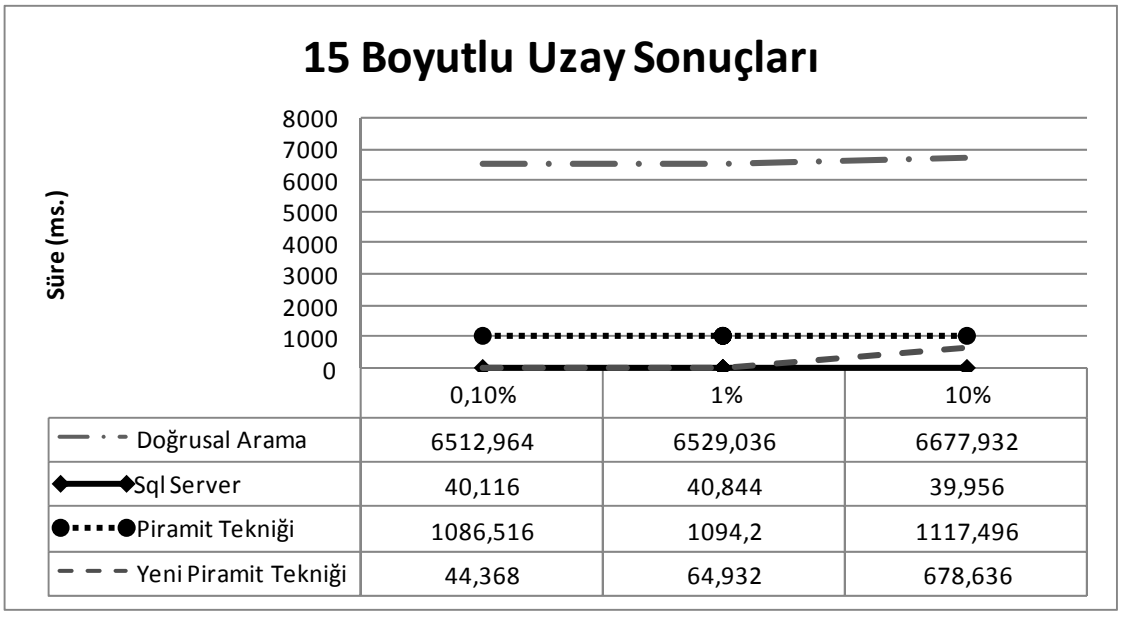
Şekil 3.16'da görüldüğü gibi her seçicilikte en kötü performansı doğrusal arama sergilemiştir. Piramit Tekniği ise %0,1 ve %1 seçiclikte kötü bir performans sergilerken %10 seçicilikte Yeni Piramit Tekniği'ne yakın bir performans sergilemiştir. Yeni Piramit Tekniği %0,1 seçilikte B+ Arama'dan daha iyi performans göstermiştir ama seçicilik arttıkça performansı daha da kötüleşmiştir. Yine de genelde B+ Ağaç'a yakın bir performans gösterdiği söylenebilir.



	0,10%	1%	10%
Doğrusal Arama	4904,552	4527,088	4563,464
Sql Server	259,628	269,332	62,208
Piramit Tekniği	2931,692	3066,912	862,372
Yeni Piramit Tekniği	230,836	313,092	417,796

Şekil 3.16. 7 Boyutlu Uzayda %0,1, %1 ve %10 Seçicilikte Sorgu İşlenme Süreleri

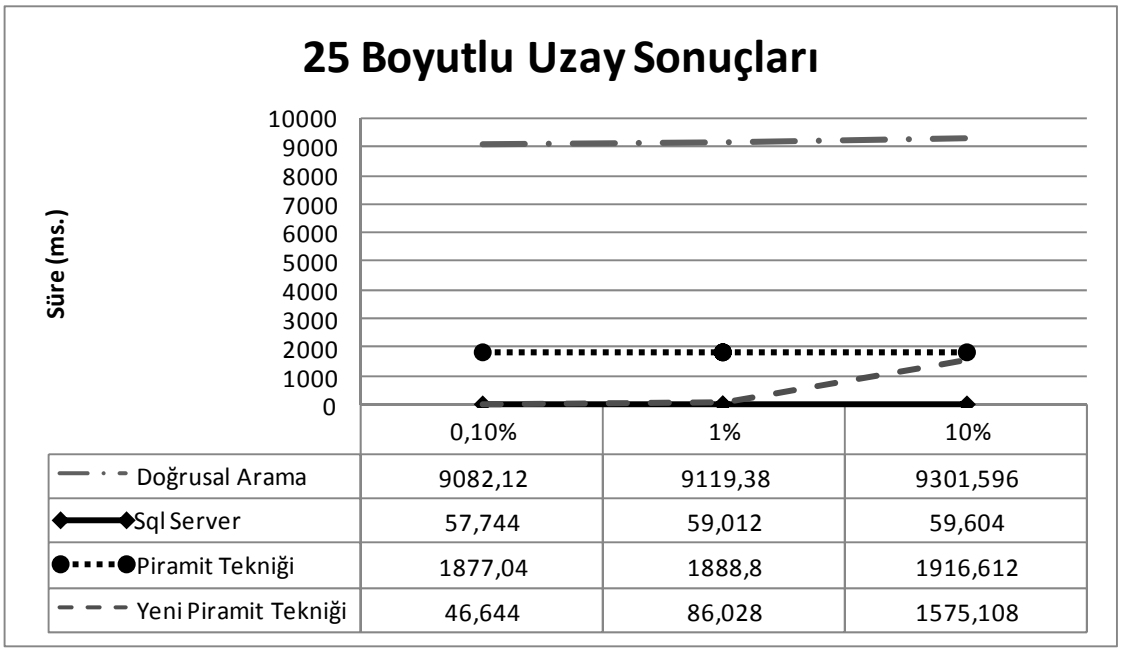
Bir sonraki karşılaştırma 15 boyutlu uzayda yapılmıştır. Şekil 3.17'den de görülebildiği gibi 7 boyutlu uzayda B+ Ağaç; Yeni Piramit Tekniği, Piramit Tekniği ve Doğrusal Arama'dan daha iyi bir sonuç vermiştir. Yeni Piramit Tekniği tıpkı 15 boyutlu uzayda olduğu gibi %0,1 ve %1 seçicilikte B+ Ağaca çok yakın sonuçlar çıkarıp %10 seçicilikte performansında bozulma olmuştur.



	0,10%	1%	10%
Doğrusal Arama	6512,964	6529,036	6677,932
Sql Server	40,116	40,844	39,956
Piramit Tekniği	1086,516	1094,2	1117,496
Yeni Piramit Tekniği	44,368	64,932	678,636

Şekil 3.17. 15 Boyutlu Uzayda %0,1, %1 ve %10 Seçicilikte Sorgu İşlenme Süreleri

25 boyutlu uzayda yapılan karşılaştırmada da B+ Ağaç diğer yöntemlerden iyi bir performans sergilemiştir. Diğer testlere benzer şekilde Yeni Piramit Tekniği %0,1 ve %1 seçicilikte iyi bir performans göstermiş ancak seçicilik %10 olunca performansı Piramit Tekniği'ne yaklaşmıştır.



	0,10%	1%	10%
Doğrusal Arama	9082,12	9119,38	9301,596
Sql Server	57,744	59,012	59,604
Piramit Tekniği	1877,04	1888,8	1916,612
Yeni Piramit Tekniği	46,644	86,028	1575,108

Şekil 3.18. 25 Boyutlu Uzayda %0,1, %1 ve %10 Seçicilikte Sorgu İşlenme Süreleri

Karşılaştırmalar sonucunda görülmüştür ki dar hiperküp sorgularında Yeni Piramit Tekniği B+ Ağaç kadar iyi sonuçlar çıkarabilir. Ancak hiperküp sorguları daha geniş bir alanı etkilediğinde Yeni Piramit Tekniği yetersiz kalmaktadır. Yine de Yeni Piramit Tekniği her testte temelini oluşturan Piramit Tekniği'nden daha iyi performans göstermiştir.

4. SONUÇ

Bu çalışmada çok boyutlu veriler için indeksleme yöntemleri incelenmiştir. İncelenen yöntemler R Ağaç, R* Ağaç, TV Ağaç, X Ağaç, Piramit Tekniği ve yine Piramit Tekniği tabanlı çok boyutlu veri indeksleme yöntemi kullanan bir yöntemdir.

İlk olarak çok boyutlu yöntemlerin atası olarak kabul edilen R Ağaç yöntemi incelenmiştir. Bu yöntem ilk çok boyutlu yöntem olduğundan genelde 2 boyutlu ve 3 boyutlu uzaylar için yeterince iyi bir indeksleme yöntemidir. Kendisinden sonra sunulacak birçok yönteme de temel oluşturmuştur. Bu bakımdan önemli bir yöntemdir.

Ardından R* Ağaç incelenmiştir. Bu yöntem ise R Ağaç üzerine bir geliştirme gibidir. R Ağaç'ta düğüm bölünmeleri alana göre yapar ve üst üste binmeler fazla olabilir. Üst üste binen bölgeye denk gelen bir sorgu üst üste binen bütün düğümleri sorgulamak zorundadır. Bu da sorgu işlemini uzatır ve zorlaştırır. Bu yüzden R* Ağaç'ta uygulanan yöntem bölünmeleri üç kritere göre yapmaktır. Bunlar üst üste binen alan miktarı, dikdörtgenlerin çevresi ve dikdörtgenlerin alanı. Bu kriterlere uygun bir bölünme uygulandığında daha az üst üste binen alan olur ve bu yüzden R* Ağaç R Ağaç'tan daha iyi performans sergiler.

R* Ağaç sonrasında ise TV Ağaç adı verilen bir yöntem ileri sürülmüştür. R Ağaç tabanlı yöntemlerin yüksek boyutlu uzaylarda başarılı olamadığı gözlemlenmiştir. Bu da bu tip yöntemlerin özellik vektörünün yüksek boyutlarda bir disk sayfasından daha fazla alana ihtiyaç duymasından kaynaklanır. Bu yöntemin bir bağlı liste gibi davranmasına yol açar. TV Ağaçta bu sıkıntıları gidermek amacıyla gerektiğinde boyut sayısı artan, gerektiğinde azalan teleskopik vektör kavramına yer verilmiştir. Bu şekilde nesneleri ayırmak için gerekli boyut sayısı genelde az olduğundan alandan tasarruf eder ve çıkış yelpazesi geniş olur. Bu da TV Ağacın diğer R Ağaç tabanlı yöntemlerden daha iyi performans göstermesinin nedenidir.

Yüksek boyutlu uzayda üst üste binen bölgeler genel bir problemdir. X Ağaç da bu sorun için yeni bir çözüm getirmek amacıyla sunulmuştur. X Ağaç'ta süper düğüm

kavramı karşımıza çıkmaktadır. Bu bir düğümün boyutunun iki veya daha fazla katı olan düğümdür. Süper düğüm ve üst üste binmeyi azaltan yeni algoritmalar sayesinde X Ağaç, TV Ağacın performansını geçebilmiştir.

X Ağaç sonrasında Piramit Tekniği adı verilen bir çok boyutlu veri indeksleme yönteminden bahsedilmiştir. Bu teknik özel bir uzay parçalama modeline dayanır. Bu parçalama sonucunda d boyutlu noktalar 1 boyutlu ve B+ Ağacın rahatlıkla indeksleyebileceği noktalara dönüştürülürler. Bu şekilde Piramit Tekniği X Ağaç'tan da daha iyi performans göstermiştir.

Piramit Tekniği sonrasında temeli Piramit Tekniği olan yeni bir yönteme değinilmiştir. Bu yöntem de Piramit Tekniği gibi uzayı piramitlere bölüp bunlardan yararlanarak çok boyutlu bir özellik vektörünü tek boyuta indirir. Daha sonra da bu verileri B+ Ağaç yardımıyla indeksler. Ancak bu yöntemde Piramit Tekniğinin bazı kısıtlamaları aşılmış ve bu sayede daha fazla piramit parçalanmış ve seçim anahtarı geliştirilmiştir. Bu şekilde de Piramit Tekniğinden daha iyi bir performans elde edilmiştir.

Her bir yöntemin kendisinden bir önceki yöntemin eksiklerine göre ortaya çıktığı gözlemlenebilir. Bu şekilde sürekli daha iyi bir yönteme doğru çalışmalar yapılmıştır. Sonunda bu çalışmada bahsedilen yöntemler arasındaki Jiyuan An, Yi-Ping Phoebe Chen, Qinying Xu ve Xiaofang Zhou tarafından sunulan yüksek boyutlu veri kümeleri için indeksleme yöntemi en iyi performansı sağlayan yöntem olarak belirlenmiştir. Ancak günümüzde çok boyutlu veri kullanan uygulamalar sürekli gelişmektedir. Bu süreçte çok boyutlu verilere daha hızlı ulaşım ihtiyacı sürekli var olacaktır. Dolayısıyla bu alanda çalışmalar sonlanmayacak ve gün geçtikçe yeni ve daha iyi yöntemler bulunacak ve bu yöntemler bilgisayar dünyasında birçok konuda insanlara yardımcı olacaklardır.

KAYNAKLAR DİZİNİ

An, J., Chen, Y.P.P., Xu, Q., Zhou, X., 2005, "A New Indexing Method for High Dimensional Dataset"

Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B., 1990, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", Atlantic City, NJ (ACM SIGMOD), 322-331

Berchtold, S., Keim, D., Kriegel, H.P., 1998, "The pyramid-Technique: Towards Breaking the Curse of Dimensional Data Spaces" Seattle (ACM SIGMOD Int. Conf. Managment of Data), 142-153

Berchtold, S., Keim, D., Kriegel, H.P., 1996, "The X-tree: An Index Structure for High-Dimensional Data" Bombay, India (Morgan Kaufmann Publishers Inc.),28-39

Fukunaga K., 1990, "Introduction to Statistical Pattern Recognition" NewYork (Academic Press)

Guttman, A, 1984, "R-trees a dynamic index structure for spatial searching", (ACM SIGMOD Int Conf on Management of Data), 47-57

Hamming, R.W., 1977, "Digital Filters" Englewood Cliffs (NJ: Prentice-Hall)

Lin, K.I., 1995, Jagadish, H.V., Faloutsos, C., "The TV-Tree: An Index Structure for High-Dimensional Data", VLDB Journal, Vol.3, 517-542

Ruskai, M.D., Beylkin, G., Coifman, R., Daubechies, I., Mallat, S., Meyer, Y., Raphael, L., 1992, "Wavelets and Their Applications" Boston (Jones and Bartlett Publishers)

Wallace, G.K., 1991, "The jpeg still picture compression standard" (CACM), 34(4), 31-44

TEŞEKKÜR

Tez konumun seçiminde ve geri kalan süreçte danışmanlığımı yapan, desteğini esirgemeyen, bana her konuda yardımcı olan Yrd. Doç. Dr. Cavit TEZCAN' a, TV Ağaçla ilgili araştırmalarından bilgi edindiğim Arş. Gör. Deniz TAŞKIN' a, R Ağaç ve türevleri ile ilgili destek veren Arş. Gör. Filiz SOYKAN' a, çok boyutlu erişim yöntemlerinin karşılaştırılması ile ilgili tavsiyeler aldığım Yrd. Doç. Dr. Aydın CARUS' a, tezin hazırlanmasında desteğini esirgemeyen Yrd. Doç. Dr. Nurşen SUÇSUZ'a ve bu süreçte beni maddi, manevi destekleyen, motive eden aileme teşekkürlerimi sunarım.

ÖZGEÇMİŞ

Kişisel

Adı Soyadı : Onat GÖZET

Doğum Tarihi : 04.12.1983

Doğum Yeri : Ankara

Medeni Hali : Bekar

T.C. Kimlik No : 13003321058

Eğitim

2005 - … TrakyaÜniversitesi, Fen Bilimleri Enstitüsü, **Bilgisayar Mühendisliği Bölümü,**

2005-2001 Trakya Üniversitesi, Mühendislik Mimarlık Fakültesi, **Bilgisayar Mühendisliği** Bölümü, Edirne

2001-1997 Edirne Lisesi, **Fen-Matematik** Bölümü, Edirne

İş Deneyimi

2007 – … E-veri Bilişim Hizmetleri, YazılımMühendisi

Dil

İyi derecede İngilizce. (Okuma, konuşma, anlama, yazma)