KONUMSAL DİZİNLEME YÖNTEMLERİNİN KARŞILAŞTIRILMASI

COMPARISON OF SPATIAL INDEXING METHODS

MURAT SEÇKİN AYHAN

Başkent Üniversitesi
Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin
BİLGİSAYAR Mühendisliği Anabilim Dalı İçin Öngördüğü
YÜKSEK LİSANS TEZİ
olarak hazırlanmıştır.

2007

Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Bu çalışma, jürimiz tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan :….....
Prof. Dr. Hayri Sever

Üye :….....
Prof. Dr. Mehmet Ümit Karakaş

Üye :….....
Doç. Dr. Haşmet Gürçay

ONAY

Bu tez ...../...../..... tarihinde Enstitü Yönetim Kurulunca belirlenen yukarıdaki jüri üyeleri tarafından kabul edilmiştir.

...../...../.....
Prof.Dr. Emin AKATA
FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRÜ

TEŞEKKÜR

“Bu tez çalışmasında destek olup sonuca ulaşmamı sağlayan herkese teşekkür ederim” diyerek yetinmeyecek olduğum;

Prof. Dr. Hayri Sever’e, her zaman her konuda yardımcı olduğu, görülmeyeni gördüğü, gösterdiği ve hocam olduğu için,

Doç. Dr. Haşmet Gürçay’a, yol göstericiliği ve yardımları için,

Güven Köse’ye, veri denince akla gelen ilk isim olması, güvenli veri kavramına getirdiği yeni bakış açısı, yardımseverliği ve anlayışı için,

Murat Hacıömeroğlu, Sarper Gözütok ve Koray Açıcı arkadaşlarıma harcadıkları yoğun emekler için,

Ve tabi ki her zaman yanımda olan aileme yürekten teşekkür ederim.

ÖZ

KONUMSAL DİZİNLEME YÖNTEMLERİNİN KARŞILAŞTIRILMASI

Murat Seçkin Ayhan

BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI
YÜKSEK LİSANS TEZİ
Ankara, 2007

Bilgisayar bilimlerinde yaşanan gelişmelerle ve bunların sonucu olarak üretilen teknolojilerin artmasıya birlikte coğrafi bilgi sistemleri (CBS) ve bilgisayar destekli tasarım (BDT) uygulamaları gibi temel veri tiplerinden farklı, konumsal veri tipleriyle de ilgilenen sistemler günlük hayatta daha sık yer almaya başlamıştır. Bu farklı ve karmaşık veri tiplerinden oluşan büyük veri kümelerinde sorguları verimli işleyebilmek için konumsal erişim yöntemleri gereklidir. Open Geospatial Consortium (OGC) tarafından sektörel bazda ortak standartlar belirlense de dizinleme konusunda şu ana kadar kabul gören herhangi bir standart yoktur.

TUBITAK tarafından desteklenen SOBAG-105K040 Evliya Çelebi Coğrafi Bilgi Çekirdeği Projesi kapsamında, konumsal dizinleme yöntemleri incelenmiş ve karşılaştırmalı olarak değerlendirilmiştir. İncelenen yöntemler R-ağacı, STR-ağacı ve MX-CIF 4'lü ağacıdır. İstatistiksel dağılımlar ve Bezier eğrileri yardımıyla üretilen sentetik veriler kullanılarak uygulanan test senaryoları ile başarımları kıyaslanan bu yöntemler arasında en başarılı sonuçları STR-ağacı vermiştir. Fakat durağan veri kümelerine daha uygun olan bu yöntem sık veri güncellemesi yapılan sistemlerde sıkça yeniden yapılandırma işlemi gerektirecektir. Dinamik ortamlarda R-ağacı veya MX-CIF 4'lü ağacı kullanılması daha uygun olacaktır.

Anahtar Kelimeler : Konumsal Dizinleme, Konumsal Erişim Yöntemi, Konumsal Veritabanı, Bezier Eğrisi, PostgreSQL, R-ağacı, STR-ağacı, MX-CIF 4'lü ağacı

ABSTRACT

COMPARISON OF SPATIAL INDEXING METHODS

Murat Seçkin Ayhan

DEPARTMENT OF COMPUTER ENGINEERING
MASTER THESIS
Ankara, 2007

According to achievements in computer sciences and resulting technologies, such as geographic information systems (GIS) and computer aided design (CAD) applications those have interest in spatial data types which are different from primitive data types like integers, characters etc., take more place nowadays. To query these different, complex and also huge volumes of data in an efficient way, some spatial indexing methods are urgent. Common standards on data types and representations have been defined by the Open Geospatial Consortium (OGC), but nothing about spatial indexing has occurred yet precisely.

In the concept of the research project named Evliya Çelebi Geographic Information Core, funded by TUBITAK under the code SOBAG-105K040, general objective is to study spatial indexing methods and to compare their performances. Three spatial indexing methods examined are: R-tree, STR-tree and MX-CIF Quadtree. Data sets are generated synthetically by Gaussian and uniform distributions, and Bezier curves. Window queries are applied in test scenarios.

To conclude, we can say that STR-tree has the best results. Because of its internal structure and definition, STR-tree targets static data sets. If dynamic data insertions and deletions occur frequently, reorganization is urgent. R-tree and MX-CIF Quadtree are more suitable for dynamic environments.

Keywords : Spatial Indexing, Spatial Access Method, Spatial Database, Bezier Curve, PostgreSQL, R-tree, STR-tree, MX-CIF Quadtree

İçindekiler

Şekiller Dizini

Tablolar Dizini

Simgeler ve Kısaltmalar

Simge	Açıklama
^{0}	Nesnenin iç kısmı
∂	Nesnenin sınırları
Ø	Nesne kesişimleri boş küme
$\neg$Ø	Nesne kesişimleri boş küme değil

Kısaltma	Açıklama
CBS	Coğrafi Bilgi Sistemi
VTYS	Veritabanı Yönetim Sistemi
KVT	Konumsal Veri Tipi
PBN	Pencere Büyüklük Numarası
XML	Extensible Markup Language
SQL	Structured Query Language
CIF	Caltech Intermediate Form
OGC	Open Geospatial Consortium
WKT	Well Known Text
WKB	Well Known Binary

1. Giriş

Coğrafi bilgiyi sentezleme ve göstermenin temel yolu uzun yıllar boyunca kağıt haritalar olmuştur. İnsanoğlunun hayatında harita kullanımının çok eskilere kadar uzanmasına rağmen bu harita bilgilerini işleme süreçleri genellikle el yordamlı, etkileşimli olmayan bir yapıda gerçekleşmiştir. Coğrafi bilgileri toplama ve sayısallaştırma teknolojilerinin hızlı gelişimiyle birlikte bu verilerin işleme ve analiz süreçlerine sağlanan teknolojik destek artmıştır. Bu çalışmalar 20. yüzyılın ikinci yarısında ABD gibi teknolojiye liderlik eden ülkelerin hükümetleri ve akademik kuruluşlar tarafından desteklenmiş ve yürütülmüştür. Dünya coğrafyasını bilgisayar ortamında izlemeye, verilerin etkin kullanımına ve gerektiğinde verilerin sayısal ortamdan eski kağıt harita ortamına aktarılmasına imkan sağlaması hedeflenen bilgi sistemleri 1970'li yıllarda ilk örneklerini vermiştir. Bu tür uygulamalara destek veren özel amaçlı yazılımlara genel olarak Coğrafi Bilgi Sistemi (CBS) adı verilir. Bütün yeni gelişen teknolojiler gibi bu sistemler de ilk başlarda sınırlı bir kullanıcı kitlesine sahip olmuştur. Fakat sayısal teknolojinin büyük atılımlar yaptığı son 20 yılda iş istasyonları ve kişisel bilgisayarların maliyetlerinin düşmesi ile birlikte kullanımlarının artması, veritabanı yönetim sistemlerinden alınan desteğin de güçlenmesi sayesinde CBS'ler daha geniş kullanım alanına sahip olmuştur. Hatta günümüzde Internet üzerinden XML standartlarına uygun yapıları kullanarak diğer sistemlerle de etkileşime girerek hizmet verebilen bu sistemler hayatın içinde giderek daha çok yer almaktadır.

Bilginin toplanması, depolanması, gerektiğinde işlenmesi, işe yarar hale getirilmesi sistematik işlem adımlarıdır, yani bir sistem gerektirir. Bu amaçla ortaya konan sistemlere bilgi sistemleri denir. Bilgi sistemi organizasyonların yönetimsel fonksiyonlarını desteklemek amacı ile bilgiyi toplayan, depolayan, üreten veya dağıtan bir mekanizma olarak tanımlanır [26]. Bilgi sistemi, bilgiyi daha etkin, verimli kullanmayı sağlayan bir sistem olarak düşünülebilir. Bilgi sistemlerinin temel işlevi doğru karar verme potansiyelini artırmaktır.

Bir CBS, sadece harita üretme amaçlı bir haritacılık aracı değildir. CBS coğrafi verileri saklar, geri getirir ve yeni gösterimler için hazırlar, konumsal analizler için araçlar sunar. Yönetim, taşıma ağları, askeri ve çevresel uygulamalar gibi birçok alanda karar alma süreçlerinde uzmanlara simülasyon yapma imkanı sağlar. CBS'nin bu tür uygulamalardaki rolü, kullanıcılara ve karar vericilere karmaşık ve genellikle yarı yapısal konumsal problemleri çözme adına etkili araçlar sunarken tatmin edici de bir performans sağlamaktır [17]. CBS'lerin genel kullanım alanlarını şöyle örneklendirebiliriz :

- Şehir ve bölge planlaması
- Kadastro planlarının hazırlanması
- İmar planlaması
- Afet bilgi sistemleri
- Askeri projeler

Coğrafi verilerin sürekli bir artış içinde olduğu düşünülürse, CBS'nin temel görevlerinden biri de büyük miktarlardaki karmaşık verileri etkili bir şekilde yönetmektir. Klasik bir yazılım mimarisinde bu görev veritabanı yönetim sisteminindir (VTYS). Birçok VYTS ilişkiseldir. Her ne kadar ilişkisel VTYS'ler, basit verilerden oluşan büyük veritabanlarını kullanan günümüz iş uygulamalarında güçlü olsalar da, bu sistemler coğrafi veriler söz konusu olduğunda yetersiz kalmaktadırlar. Çünkü coğrafi verilerin doğasından kaynaklı konumsal bileşenleri vardır. Bu nedenle VTYS teknolojilerini geliştirmek ve konumsal verilere uygun hale getirmek için ciddi çalışmalar yapılmaktadır. CBS ve VTYS teknolojileri her ne kadar karşılıklı bağımlılık gösterse de temel olarak iki farklı görüş olduğu öne sürülebilir. Bunlardan bir tanesi, CBS'nin VTYS'yi saklama ve geri getirme aracı olarak gördüğü, diğer alt bileşenlerinden biri olarak varsaydığı görüştür. Fakat diğer taraftan, VTYS tasarımcıları da, CBS'leri, kendine has olmayan bir uygulama alanı olarak düşünmektedirler. Onlara göre CBS'ler, VTYS'lerin konumsal veriler üzerine bir uygulamasıdır. Veritabanı açısından bakıldığında önemli olan uygulamanın kendisi değil, işlevselliktir. Bir başka deyişle sırasıyla; büyük miktarlardaki konumsal verinin saklanması, hesaplama tekniklerinin bu verilere uygulanması, ve saklama birimlerinde bulunan yüksek hacimli veri kümeleri içerisinden ilgili adaylara etkin erişim (*dizinleme*) yöntemlerinin

gerçekleştirilmesidir. Bu çerçevede düşünüldüğünde konumsal nesnelerin gösterimi ve bunlar üzerine tanımlı işlemlerin yürütülebilmesi için başta hesaplamalı geometri ve bilgiye erişim olmak üzere farklı alanlardan gelen bilgi birikiminin bir arada kullanıldığı söylenebilir.

Birçok açıdan gereksinimler coğrafi veri yönetiminin ötesine de geçebilmektedir. Konumsal bir VTYS çekirdeğinde bulunan temel özelliklerden biri olan konumsal bilgiyi yönetebilme yeteneği, bilgisayar destekli tasarım, yüksek ölçekli birleştirme, robotik ve imge işleme uygulamalarında büyük konumsal veri kümeleri söz konusu olduğunda temel ihtiyaçtır. Belirtildiği gibi birçok uygulama alanı olsa da konumsal VTYS'lerin temel uygulama alanı CBS'lerdir. İlk olarak coğrafi uygulamalar için konumsal VTYS'lerden bahsedilmeye başlandığı için de tasarımlarını büyük ölçüde etkileyen yine coğrafi uygulamaların gereksinimleri olmuştur.

Coğrafi bilgi haritalarla gösterilir. Dolayısıyla bir harita, toprak parselleri, ırmaklar ve yollar gibi aynı coğrafi bölgeye ait coğrafi nesneleri içerir. Her coğrafi nesne en genel haliyle şu iki temel bileşene sahiptir.

- Geometrik özellik veya konumsal kaplam olarak öne sürülebilecek bu bileşen, yer, şekil, büyüklük, yönelim gibi 2 veya 3 boyutlu uzayda bulunan nesnenin konumsal özelliklerini belirtir.

- Konumsal olmayan olarak düşünülebilecek bileşen ise nesnenin tanımlayıcı, tematik özelliklerini belirtir. Örneğin nüfus, büyüme oranı, tarım kapasitesi, tip, isim, vb.

Farklılaşarak daha da ileri giden karakteristikler sadece coğrafi verilere has değildir. Örneğin yüksek ölçekli birleştirim uygulamalarında kullanılan yerleşim tanımları da çok boyutlu uzaya yayılmış kaplamlardaki nesnelerle ilgilenir. Bu sebeptendir ki konumsal veritabanları, bilgi sistemlerinin, konumsal bileşeni olan nesnelerle uğraşan daha özel bir hali olarak görülür. Konumsal veritabanları 2 ve 3 boyutlu sahnelerden oluşan veri kümelerini yöneten imge veri tabanlarından ayrılmalıdır. Tıbbi veritabanları, elektronik belgeler ve kültürel koleksiyonlar

potansiyel imge kaynaklarıdır. İmge veritabanları belirli nesneleri ve numuneleri içeren sahnelerin aranmasına, sorgulanmasına destek verse de yer, yön, büyüklük gibi konumsal kavramlar daha az önemlidir ki bu da konumsal veritabanlarıyla aralarındaki temel farkı göstermektedir [26].

Günümüzde CBS yardımıyla yürütülen işlerin bir kısmı CBS teknolojileri ortaya çıkmadan önce de mevcuttu. Bunlara örnek olarak nüfus sayımları, atlaslar, yönetim planlama, yapılaşma planları, mühendislik ve tabi ki haritacılık verilebilir. Daha farklı alanları da kapsamak üzere daha büyük veya küçük ölçeklerde de olsa CBS teknolojilerinin, kullanıldıkları alanlarda bir adım öteye gitmeyi sağladığı her ne kadar açıkca görülse de bu farklılıklar çoğu zaman tartışmalara, uyumsuzluklara ve belirsizliklere yol açabilmektedir.

Bir CBS'nin ne gibi özelliklere sahip olması gerektiğini belirleyecek ortak bir standart oluşturma çabaları uzun yıllar sonucunda olgunlaşıp Open Geospatial Consortium (OGC) adında uluslarası bir kurumun ortaya çıkmasıyla sonuç vermiştir. Konumsal verilerin işlenmesi, kullanıcıya sunulması ve diğer teknolojik sistemlerle olan bağlantıları sağlayacak olan ara yüzler konusunda standartlar belirleyen bir kuruluş olan OGC, uluslarası ortamlarda söz sahibi olmak isteyen CBS geliştiricileri için ortak buluşma noktalarını işaretleyerek bu alanda belirleyici bir rol oynamaktadır.

Bir CBS, hem konumsal hem de alfasayısal verileri saklamak zorundadır. Bu temel gereksinim dosya sistemini kullanarak doğrudan uygulamanın kendisi tarafından veya bir VTYS aracılığıyla yerine getirilebilir.

İlk CBS örnekleri doğrudan dosya sisteminin üzerine yerleşerek hizmet vermekteydi. Hatta bazı az bilinir CBS'ler hala bu yoldan yürümektedirler. Konumsal ve alfasayısal veriler dosyalarda saklanır, uygulama tarafından kontrol edilir ve bu veriler üzerine tanımlı fonksiyonlar vardır. Bu yöntem, veri bağımsızlığı prensibine aykırı düşmesinin yanında, kendini ispatlamış VTYS teknolojilerinin sağladığı veri güvenliği, eş zamanlı erişim gibi avantajlardan da yoksundur [26].

Konumsal ve konumsal olmayan verilerin daha net bir şekilde anlaşılabilmesi için örnek bir konumsal sorguyu incelemek faydalı olacaktır.

“Van Gölü’ne 10 km. mesafedeki köyleri bul.”

Bu sorgu incelendiğinde Van Gölü’nü temsil eden merkezi bir nokta, gölün konumsal bilgilerine, sorgu alanına giren köylerin koordinatları da köylerin konumsal bilgilerine örnektir. Köylerin isimleri, nüfus bilgileri gibi bilgiler ise konumsal olmayan bilgilerdir. Merkezi Van Gölü, çapı 10 km. olan alan ise konumsal veriler üzerine tutulan sorgu penceresidir. Bu pencereye giren tüm nesneler sonuç olmaya adaydır, fakat örnek sorguda sadece köyler istendiği için adaylar üzerinde bir ayıklama (filtreleme) yapılması gerekecektir. Eğer pencereye giren başka il ve ilçeler gibi köyden farklı yerleşim yerleri varsa bunlar elenecek ve sonuç kümesine alınmayacaktır.

Örneğimizin de ortaya koyduğu gibi konumsal ve konumsal olmayan veriler birbirini tamamlayan, ayrı olamayacak veri türleridir. Sorgumuzdan sadece konumsal veriler dönseydi, elimizdeki sonuç listesi sadece koordinatlardan oluşan fakat çok da anlam ifade etmeyen bir yapıda olacaktı.

Konumsal VTYS’lerin mimari açıdan temel bir sorunu da konumsal ve konumsal olmayan verilerin nasıl bir organizasyonla yönetileceğidir. Konumsal ve konumsal olmayan veriler beraber tutulacaksa, geleneksel VTYS’ler yetersiz kalacaktır, konumsal verilere yönelik eklentileri olan VTYS’lerin geliştirilmesi gerekecektir. Konumsal ve konumsal olmayan veriler ayrı tutulacakca bu sefer de verilerin işlenmesi ve sorguların değerlendirilmesi sırasında bu verilerin tekrar birleştirilmesi gibi gereksinimler ortaya çıkacaktır.

Bahsi geçen bu temel ikilem, mimari açıdan iki temel yaklaşımın ortaya çıkmasa sebep olmuştur. *Gömülü sistemler*, konumsal ve konumsal olmayan verilerin birlikte tutulduğu mimarilerdir. *Ayrık katmanlar sistemi* ise konumsal ve konumsal olmayan verilerin ayrı tutulduğu, gerekli işlem adımlarında birleştirildiği mimari tipidir.

1.1 Gömülü Sistemler

Bu mimariye sahip Oracle Spatial, IBM DB2 Spatial gibi sistemler günümüzde başarıyla hizmet vermektedirler. Gömülü sistemlerde konumsal ve konumsal olmayan veriler aynı tablolarda tutulur. Konumsal verileri destekleyebilmek için konumsal olmayan verilerden farklı veri tiplerine ihtiyaç vardır ve VTYS bu farklı veri tiplerini işleyebilir yetenektedir. Örnek sorgumuzu ele alacak olursak;

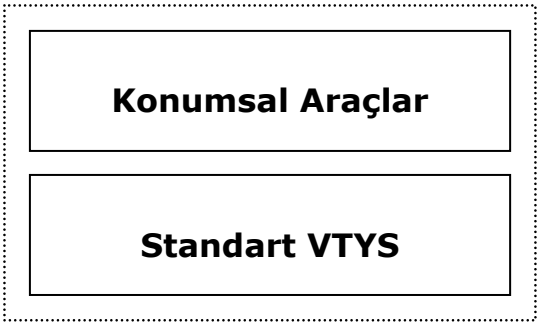
"Van Gölü'ne 10 km. mesafedeki köyleri bul."

Sorgu sonucunda bulunan köylerin konumsal bilgileri (yer, alan) ve konumsal olmayan bilgilleri (isim, nüfus, vb.) aynı tablolarda tutulmaktadır. VTYS, sorguyu değerlendirirken uzaklık hesaplayıp konum bilgilerini getirmenin yanında diğer konumsal olmayan verileri de getirmektedir.

Gömülü sistemler, konumsal sorgulama dilini destekleyen, veri erişim mekanizmaları ve dizinleme yapıları konumsal veriler için yapılandırılmış, grafik ve geometri işleme çekirdekleri olan özel amaçlı veritabanı yönetim sistemleridir.

1.2 Ayrık Katmanlı Sistemler

Ayrık katmanlar mimarisinde konumsal işlevsellik VTYS'nin üzerine yerleşir.

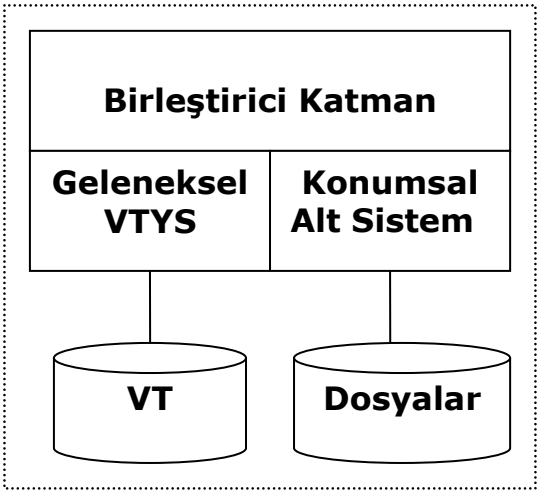


Şekil 1.1 : Ayrık Katmanlı Sistem Mimarisi (genel)

Bu mimaride konumsal veri tiplerini gösterebilmek için iki olası genel yol vardır. İlk olarak denenen yöntem poligon gibi çok parçalı geometrik nesneleri nokta veya doğru parçaları gibi alt nesnelere ayırıp çoklular halinde veritabanı tablolarında tutmaktır [4,6]. Bu yöntemin dezavantajı, konumsal verilerin yeniden yapılandırılmasının pahalıya mal olmasıdır. İkinci bir yol olarak konumsal verileri parçalamak yerine bir bütün olarak uzun sekizli(byte) dizileri halinde saklamayı deneyenler de olmuştur [70,3]. Herhangi bir değerlendirme veya işlem sırasında bu bayt dizilerinin yeniden yorumlanması yine üst katmanda yapılacağından bu ikinci yaklaşım da tam bir çözüm olamamıştır.

1.2.1 Çift Parçalı Katman

Bir öncekinden farklı olarak bu mimaride üstteki birleştirici katmanın altında iki bağımsız yapı bulunmaktadır.



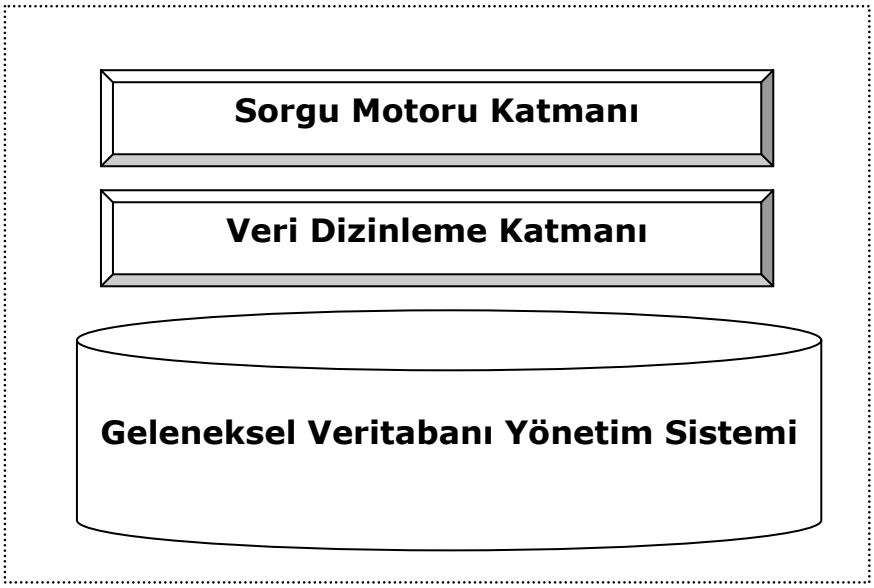
Şekil 1.2 : Çift Parçalı Katman Yapısı

Geleneksel VTYS konumsal olmayan verileri barındırırken, konumsal alt sistem doğrudan dosya sistemi üzerine yerleşip uygun veri yapılarını da kullanarak konumsal verilerle konumsal olmayan verilerin eş güdümünü sağlar. Bu mimari ARC/INFO (ESRI) ve SICAD gibi ticari CBS'ler tarafından uygulanmıştır [47,61].

Bu mimaride konumsal veri tiplerinin gösteriminde rahatlık sağlandığı gibi konumsal alt sistem dahilinde veri dizinleme ve sorgu işleme adımları için kullanılacak olan veri tipleri ve algoritmalar da özgürce seçilebilmektedir. Buradaki temel problem, bir sorgu işlenirken sorgunun, konumsal ve konumsal olmayan bölümlere ayrıştırılması gerektiğidir ki bu da sorgu işleme adımlarına ek yük bindirmektedir.

1.2.2 Temel Bileşenler

Kullandığı mimari her ne olursa olsun bir CBS'de konumsal verileri dizinleyecek ve işleyecek yapıların bulunması gerekmektedir. Bunu ayrık katmanlar mimarisinde örnekleyecek olursak Şekil 1.3 yardımcı olacaktır.



Şekil 1.3 : Ayrık Katmanlar Mimarisinde Sorgu Motoru ve Dizinleme

Geleneksel VTYS'lerde konumsal verileri işleyebilme yeteneği yoktur. Bir CBS geliştirme çalışmasında da esas amaç olan konumsal verilerle çalışabilme yeteneği yanında yeni bir VTYS geliştirme zahmetli ve maliyetli bir yoldur. Ayrıca geleneksel VTYS'lerin uzun yıllar boyunca güvenilirlik, güvenlik, kararlılık ve performans gibi konularda kendini kanıtlamış olması yeniden bir VTYS geliştirme çabasına girişmenin çok da akılcı olmadığına işarettir. Bu yüzden mevcut bir

geleneksel VTYS üzerine konumsal özellikler eklemek, geliştirme süreci açısından çok daha hızlı ve maliyet açısından da daha karşılanabilir olacaktır.

1.2.2.1 Sorgu Motoru Katmanı

Sorgu motoru katmanının bu yapı içerisindeki görevi konumsal verilerin, konumsal olmayan verilerle beraber sorgulanabilir halde dış dünyaya, kullanıcılara gösterilebilmesidir. Fakat bunu yapmak adına geleneksel sorgulama dilleri (örneğin SQL) yetersiz kalmaktadır. Geleneksel sorgu dilleri konumsal verileri işleyebilecek özelliklere sahip değildir [11]. Bu gereksinimden dolayı konumsal sorgulama dili ortaya çıkmıştır [18].

Geleneksel sorgu dilini yapısal olarak çok fazla değiştirmeden, mevcut özelliklerini koruyarak konumsal sorgulara cevap verebilme becerisini sunabilmek için günümüzde birçok CBS uygulamasında bu amaca yönelik *sorgu motoru katmanı* eklemek daha uygun bulunmakta ve bu yol seçilmektedir.

Bu alanda yapılan ve standartlaşmayı hedefleyen çalışmalar sonucunda OGC, sorgu motoru katmanına yönelik birçok öngörüde bulunmaktadır ve sektörel ilerlemenin yol haritasını tayin etmektedir. Geleneksel bir sorgu dilinin konumsal özellikler kazanmasına giden yolda uyulması gereken kurallar [50]'te belirlenmiştir.

Daha önce verilen konumsal sorgu örneğini,

"Van Gölü'ne 10 km. mesafedeki köyleri bul."

ele alacak olursak, bu sorguyu konumsal sorgu dili ile yazdığımızda

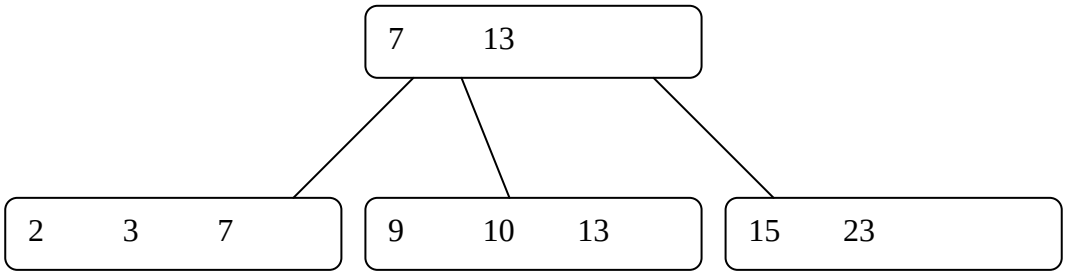
```
SELECT *
FROM koyler, goller
WHERE goller.isim ='Van Golu' AND
distance(goller.geodata, koyler.geodata) < 10000 ;
```

Sorgu katmanında yapılan işlem esasında bir ayıklamadır. Değerlendirilecek olan sorgu ilk olarak konumsal ve konumsal olmayan bölümleri açısından çözümlenir. Sorgu yapısı değerlendirildikten sonra gerekli geometrik ve ilişkisel karşılaştırmalar yapılmak üzere VTYS'ye yönlendirilir.

Tez çalışması kapsamında incelenek olan konu veri dizinleme katmanının sınırlarına girdiğinden sorgu katmanıyla ilgili verilen bu kısa bilgi konuya ilişkin fikir verme anlamında yeterlidir.

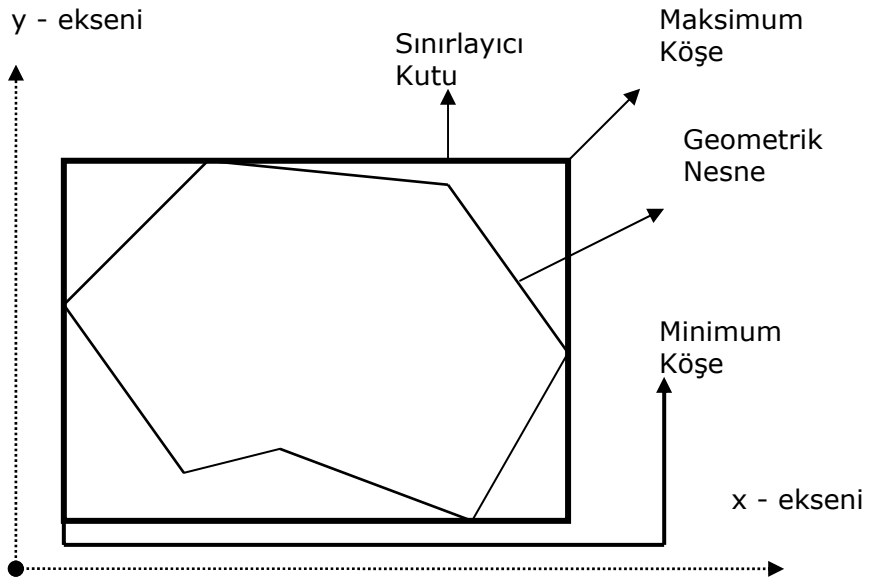
1.2.2.2 Veri Dizinleme Katmanı

Konumsal verilerin dizinlenmesindeki temel amaç, konumsal nitelikli nesnelerden oluşan büyük kümeler üzerine yapılan sorgularda, sorgu koşuluyla herhangi bir ilişkisi bulunan adayları gruplamaktır. Konumsal dizinleme yöntemleri uzayı ve nesneleri, uzayın bir bölümü veya nesnelerin bir alt kümesi dikkate alınabilecek şekilde kendi içinde organize eder. Konumsal dizinlemenin temelde ne olduğunu anlamak için B+-ağaçlarının basit, tek boyutlu veriler için yaptığı düzenlemenin, kurduğu hiyerarşik yapının (Şekil 1.4), çeşitli veri yapıları kullanılarak konumsal verilere uyarlandığını düşünebiliriz. Konumsal dizinleme yöntemleri konumsal seçim, konumsal birleştirme, alan (pencere) sorgulama, kesişim sorguları gibi temel konumsal sorgu tiplerine imkan sağlar.



Şekil 1.4 : B+ ağacı örneği

Konumsal dizinleme yöntemlerinin temelinde yatan ana fikir konumsal nesnelere yaklaşımlarda bulunmaktır. Bu sayede geometrik özellikleri bulunan konumsal nesneler geometrik karmaşıklarından biraz olsun soyutlanarak daha basit şekillerde görülebilmektedir. Bu yaklaşımların en genel ve geçerli örneği *sınırlayıcı kutu* mantığıdır (Şekil 1.5). Sınırlayıcı kutu, konumsal bir nesneyi sınırlayan, kenarları koordinat eksenlerine paralel olarak çizilebilecek en küçük dikdörtgen olarak tanımlanır.



Şekil 1.5 : Sınırlayıcı Kutu

OGC'nin, sektörel bazda bir standart oluşturmayı hedeflerken verilerin gösterimi, kodlaması, kullanılacak arayüzler gibi konularda bir çok konuyu adreslemesine rağmen konumsal verilerin dizinlenmesi anlamında ortaya koyulan açık ve net bir oluşum henüz yoktur.

Konumsal bilgi sistemlerinin temel araştırma alanlarından biri olan dizinleme hakkında yıllardır ortaya konulmuş birçok veri yapısı ve yöntem vardır. Genel yapıları açısından düşünüldüğünde bu yöntemler iki ana başlıkta toplanabilmektedirler.

- *Uzay güdümlü yöntemler :* Veri dağılımından bağımsız olarak veri uzayının dikdörtgensel hücrelere ayrılmasına dayanır. Nesneler geometrik özelliklerine göre bu hücrelerle ilişkilendirilirler.

- *Veri güdümlü yöntemler :* Uzay güdümlü yöntemlerin aksine veri kümelerinin dağılımı bölmeleri etkiler. Bölme yaparken amaç verilerin uzaydaki dağılımına uyum sağlamaktır.

Bu iki temel ayrıma sahip yöntemlere örnek vermek gerekirse, sırasıyla *dörtlü ağaçlar* ve *R-ağaçları* en belirgin örnekler olacaktır. Dörtlü ağaçlarda dikdörtgenler, yani sınırlayıcı kutular, uzayın özyinelemeli olarak dörderli hücrelere bölünmesini temel alan bir yöntemle dizinlenir. Her düğümün 4 çocuğu vardır, yapraklar ise disk sayfalarına işaret eder ve ilgili dikdörtgenlere göre etiketlenirler. R-ağaçları ise veri güdümlü yöntemler sınıfına girerler. Yapıları kendini veri dağılımına göre ayarlar. B-ağaçları gibi dengeli ve hiyerarşik yapılıdırlar. Dizinlenmiş dikdörtgenler kümesi içerisinden istenen bir tanesine ulaşmak için ağacın kökünden aşağı doğru dikdörtgenlerin kesişimlerine bakılarak gidilmesi gerekir. Bir başka deyişle genelden özele doğru giderek hedeflenen alana ulaşılmaya çalışılır.

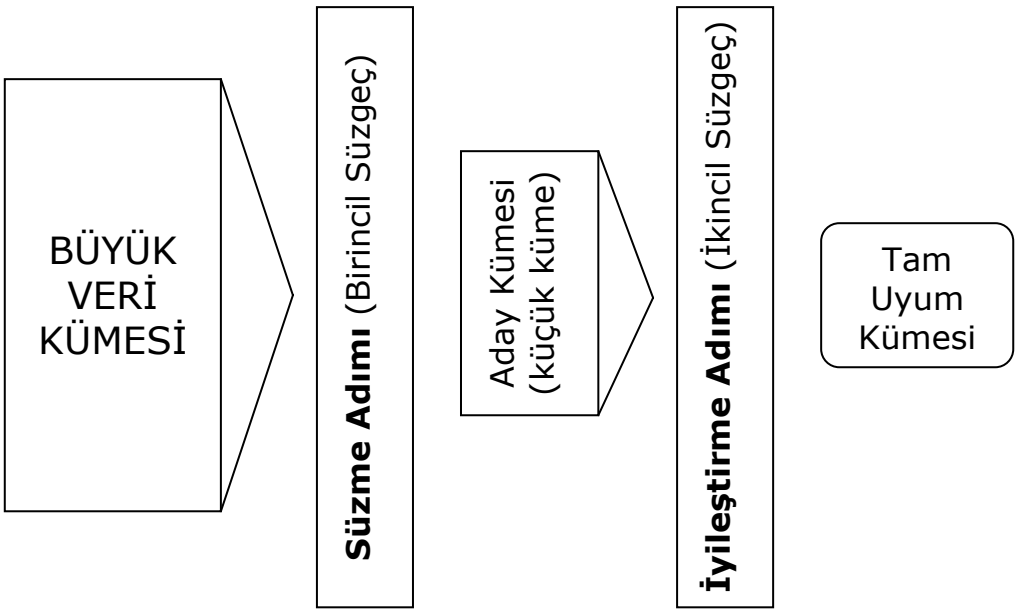
1.3 Problem Tanımı

Konumsal veriler, yoğun ve büyük miktarlardadır. Bununla birlikte yapısal ve ilişkisel olarak da oldukça karmaşıktır. Örneğin bir şehri ele alalım. Şehre kabaca baktığınızda mimari planlarlara göre yapılanmış, dizilmiş bir çok bina, park, köprü, elektrik ve boru hatları görülmektedir. Bu bakış açısını daha detaylı bir şekilde derinleştirdiğimizde nesne yoğunluğunun ve veri miktarının oldukça fazla olacağı görülecektir. Bu karmaşıklığa bir de nesnelerin birbirleriyle olan ilişkilerini eklediğimizde temel bir problemle karşılaşmaktayız; hedeflenen veriye hızlı ve etkin erişim.

Problemin karmaşıklığının yanında dikkati çeken bir diğer önemli nokta ise konumsal veriler üzerine tanımlı işlemleri gerçekleştirmek adına tanımlanan

konumsal işleçlerin geleneksel işleçlere göre daha maliyetli ve zaman alıcı olmasıdır. Konumsal veriler üzerine gönderilen bir sorgu değerlendirilirken işletilecek olan karmaşık geometrik algoritmaların uygulanacağı küme ne kadar daraltılırsa performans anlamında o ölçüde kazanç sağlanacaktır.

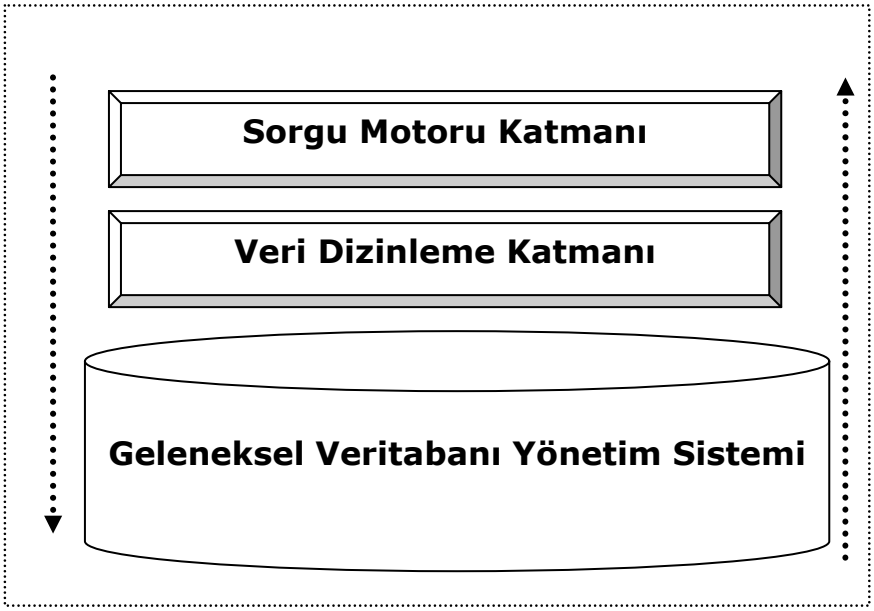
Konumsal sorgu, sorgu motoru katmanında çözümlendikten sonra, VTYS'ye yöneltilirken, yani konumsal sorgu, sonuçları elde etmek için işletilirken izlenen yöntem 2 aşamalıdır (Şekil 1.6) :



Şekil 1.6 : İki aşamalı sorgu modeli

i. **Süzme Adımı :** Nesneleri daha basit şekillerde ele alabilmek için gerçek geometrileri yerine yaklaşımlar yapıldığını ve bu sayede geometrik algoritmaların karmaşıklığından uzaklaşılmaya çalışıldığını belirtmiştik. Bu yaklaşımlardan faydalanarak oluşturulan dizin yapıları sayesinde ilgisiz nesnelere erişimler azaltılmakta ve konumsal sorgunun sonucu açısından potansiyeli yüksek olan *aday* nesneler belirlenmektedir.

ii. **İyileştirme Adımı :** Süzme adımında aday olarak belirlenen nesnelerin gerçek geometrileri üzerinden detaylı karşılaştırmalar yapılır. Adaylar kümesi içerisinde, gerçekten aranan nesneler, karmaşık geometrik hesaplamalar sonucunda bulunduktan sonra tam uyum sağlayan nesnelerin kümesi elde edilmiş olur. Bir başka ifadeyle önceki adımda bulunan sonuç iyileştirilmiş olmaktadır.



Şekil 1.7 : Ayrık Katmanlarda Sorgu Aktarımı

Şekil 1.7 yardımıyla açıklamak gerekirse, yukarıdan aşağıya doğru inen konumsal bir sorguda hedeflenen alana giren adaylar, dizinleme katmanından geçerken belirlenir (süzme adımı). Tüm nesneler yerine sadece sonuç potansiyeli olan adaylara ulaşılırsa, karmaşık ve zaman alıcı geometrik algoritmaların uygulanacağı eleman sayısı azalacaktır (iyileştirme adımı). Böylece ilgi alanı dışında kalan nesnelerle gereksiz yere ilgilenilmeyecektir. Bir başka deyişle süzme adımı dizinleme katmanı aracılığıyla uygulanır.

Genel sistem performansına doğrudan ve önemli etkisi bulunan dizinleme katmanında kullanılacak yöntemler ve veri yapıları ile ilgili uzun yıllar boyunca yapılan çalışmalar mevcuttur. Fakat konumsal sistemlerle ilgili çalışmalara öncülük

eden ve standartlaşmayı hedefleyen OGC tarafından dizinleme ile ilgili kesin bir tanımlama veya standart henüz ortaya konulmamıştır. Uygulamaya bağlı olarak geliştirme ekipleri tarafından sistemlere özel gerçekleştirimler yapılmaktadır. Mevcut sistemlerde uygulanmış olan birçok yöntem arasından da hangisinin hangisine göre ne derecede üstün olduğunu kesin bir şekilde ortaya koymak genellikle yaşanan bu gerçekleştirim farklarından dolayı mümkün olamamaktadır.

1.4 Tez Kapsamı ve Çalışma Planı

TUBITAK tarafından desteklenen SOBAG-105K040 Evliya Çelebi Coğrafi Bilgi Çekirdeği Projesi kapsamında, dizinleme yöntemlerinin karşılaştırılmasına yönelik olarak yapılan bu tez çalışmasında, dizinleme katmanı gerçekleştirilirken ortaya çıkan gereksinimlere en iyi cevap veren, genel ve verimli bir dizinleme yönteminin mevcut, kabul görmüş yöntemler incelenerek bulunması amaçlanmaktadır. Bu çerçevede düşünüldüğünde çalışmaların odağını oluşturan yer Şekil 1.6'dan örneklenebileceği üzere *dizinleme katmanıdır.*

Problem tanımı ve tez kapsamına dayanarak yapılması planlanan çalışmalar genel hatlarıyla şöyle sıralanabilir :

- Veri kümelerinin elde edilmesi
- Deney düzeneği planlaması
  - Mevcut sistemlerin incelenmesi
  - VTYS ve geliştirme ortamı seçimi
  - Dizinleme yöntemleri seçimi
- Deney düzeneği kurulumu
- Test Senaryoları ve uygulaması
- Sonuçlar ve yorumlar

1.5 Tez Düzeni

Tez çalışmasını anlatan bu dökümanın yapısal düzenini vermek gerekirse ilk bölüm konuya ilişkin genel bilgi vermeyi ve okuyucuyu problem hakkında düşündürmeyi amaçlamaktadır.

İkinci bölümde konumsal VTYS'ler ve konumsal dizinleme ile ilgili temel oluşturacak, konuya hakim olmayı sağlayacak bir altyapı kurma amacı güdülmektedir.

Üçüncü bölümde temel dizinleme yöntemlerinin başında gelen 4'lü ağaçlar anlatılmaktadır. Güncel uygulamalarda da kullanılan bu yöntem konumsal sistemler düşünüldüğünde önemli bir yere sahiptir. (Örneğin Oracle Spatial tarafından sunulan doğrusal 4'lü ağaç desteği) Tez çalışması kapsamında ise yine bir 4'lü ağaç olan MX-CIF 4'lü ağacı ele alınmaktadır.

Dördüncü bölümde ise yine temel bir konumsal dizinleme yöntemi olan ve güncel uygulamalarda karşılaşılan R-ağaçlarına yer verilmiştir. Dizinleme konusunda akla ilk gelen ve birçok yeniliğe temel oluşturan B-ağaçlarını temel alan R-ağaçları dengeli ve devingen yapılarıyla etkin sorgu başarımları elde etmektedir. R-ağaçlarında yoğunlaştırma (veri sıkıştırma) mantığının bir uygulamasına dayanan STR-ağacı da bu bölümde konu edilmiştir. Doluluk oranlarının artırılmasına yönelik bir fikre dayanan bu yöntem sık güncellenmeyen, durağan sistemlerde oldukça iyi sonuçlar verebilmektedir.

Beşinci bölümde problem tanımı ve tez kapsamı çerçevesinde yapılan çalışmalar, veri üretimi, deney ortamı, test senaryoları ve uygulamaları anlatılmaktadır.

Altıncı bölümde ise çalışma sonucu elde edilen sonuçlar verilmekte ve bir sonraki bölümde tartışılmaktadır. Son olarak ise çalışmada atıfta bulunulan kaynakların gösterildiği kaynakça ve ekler bulunmakadır.

2. Konumsal Veritabaları ve Dizinleme

2.1 Konumsal Veritabanı Sistemi Nedir ?

Geometrik, coğrafi veya konumsal veriyi, yani uzaysal bir yanı bulunan veriyi yönetmeye olan ihtiyaç birçok alanda göze çarpmaktadır. Bu uzaysallık 2-boyutta düşünülürse dünya yüzeyinin tamamının veya bir bölümünün soyutlanması olarak öne sürülebilir ki bunun en çarpıcı örneği coğrafi alanlardır. Diğer örnekleri ise beşeri uzaylar olarak verilebilir. Yüksek ölçekli birleştirmeye, insan beynini modellemeye yarayan deneysel hacimler ve hatta protein molekül zincirlerinin dizisini gösteren 3-boyutlu ortamlar beşeri uzayların önemli örneklerindendir. İlişkisel veritabanlarının ortaya çıkışından itibaren bu türde verileri hedefleyen veritabanı çalışmaları hep varolmuştur. Bu gereksinimleri karşılamayı hedefleyen teknolojinin temel karakteristiği, nispeten basit geometrik nesnelerden oluşan büyük veri kümelerini (örneğin 100.000 poligondan oluşan bir küme) yönetebilmeyi temel amaç edinmesidir. Konuların birbirine yakınlığı kesin olsa da geometrik nesnelerin hiyerarşik bir şekilde daha karmaşık yapılara dönüştürüldüğü bilgisayar destekli tasarım uygulamalarında kullanılan veritabanlarından biraz daha farklıdır.

Bu türlü destek veren VTYS'leri tanımlamak için resimsel, imgesel, geometrik, coğrafi ve konumsal gibi birçok terim kullanılmıştır. Resimsel ve imgesel veritabanı sistemi terimleri, verilerin sayısal görüntü tarama yöntemleriyle yakalanmasından dolayı kullanılmıştır. Bu tür verilerin kaynağı olarak havadan algılama uyduları ve tıbbi uygulamalarda kullanılan bilgisayarlı görüntüleme sistemleri gösterilebilir. Konumsal veritabanı sistemi terimi son on yılda daha çok benimsenmiş ve 1989 yılından itibaren uluslarası ve akademik ortamlarda bu şekliyle kullanılmaya başlamıştır [5,28,2]. Terimin böyle şekillenmesinde veritabanı açısından yapılan bir yaklaşım etkili olmuştur. Bu yaklaşıma temel oluşturan etken ise veritabanında bulunanların uzayın durağan bir resmi veya görüntüsünden

ziyade uzayda bulunan nesnelerin bizzat kendilerinin olmasıdır. Uzayda yer tutan, tanımlayıcı kimliği, alanı, yeri ve diğer nesnelerle ilişkileri bulunan nesneleri yönetmek için gerekli teknikler ve gereksinimler taranmış görüntülerden oldukça farklıdır. Bu sebeptendir ki konumsal VTYS'ler ve imgesel VTYS'ler iki farklı sınıf olarak ele alınmakta ve açık bir şekilde ayrılmaktadır [27,20]. İmge veritabanları uzayın görüntülerinden nesne çıkarım ve analiz teknikleri içermelerinin yanında bir takım konumsal veritabanı işlevselliği de sunabilmektedirler. Her ne kadar saklama, işletme, geri getirme gibi yetenekleri olsa da taranmış görüntüleri farklı varlıklar olarak ele almaktadırlar. Çalışmamız konumsal veritabanlarını ilgilendirdiği için diğer benzeri kavramları bir kenara bırakıp şu soruya yoğunlaşmak yerinde olacaktır. Konumsal VTYS nedir?

Genel bir bakış açısıyla temel olarak şunları düşünebiliriz.

1. Konumsal VTYS temelde yine bir VTYS'dir.
2. Veri modellerinde ve sorgu dilinde konumsal veri tipleri (KVT) sunar.
3. Kendi gerçekleştirimiyle birlikte barındırdığı destek ile KVT'lerini kullanmaya, konumsal dizinlemeye ve verimli konumsal birleştirme algoritmalarına imkan sağlar.

Birinci madde basit görünse de konumsal veya geometrik bilginin konumsal olmayan, örneğin alfasayısal, veriyle her zaman ilişkili olduğunu vurgulamaktadır. Standart veri modellemelerini ve sorgu görevlerini yerine getiremeyen bir özel amaçlı sistem, amacı adına yetersiz kalacaktır. Bu yüzden konumsal veritabanı sistemleri, konumsal verilerle başedebilmek için ek özellikler barındıran tam donanımlı veritabanı sistemleri olarak tanımlanabilir. İkinci gereksinimde konumsal veri tiplerine destek vermek için temel soyutlamalar yapılmalıdır. Bu veri soyutlaması sayesinde uzayda bulunan geometrik varlıkların (nokta, doğru parçası, bölge gibi) yapısal olarak modellenmesi sağlanır. Bu modellemede nesnelerin birbiriyle olan ilişkileri (örneğin kesişim), özellikleri (örneğin alan > 1000) ve nesneler üzerine tanımlı işlemler de (kesişim(l,r) gibi) probleme dahil olmaktadır. Hangi veri tiplerinin kullanılacağı hiç kuşkusuz desteklenecek uygulamaya göre değişecektir. Yüksek ölçekli birleştirim uygulaması düşünüldüğünde dikdörtgenler, 3-boyutlu uygulamalarda hacim ve yüzeyler

gerekmektedir. Konumsal veri tipleri olmaksızın konumsal bir uygulamayı hedefleyen bir sistem modelleme açısından yeterli desteği veremeyecektir. Üçüncü ve son gereksinim ise bütün sistemin genel performansını etkileyecek kadar önemli bir noktayı işaret etmektedir.

Bu noktada konumsal birleştirmenin tanımını vermek gerekirse, Günther'in tanımı şöyledir :

R ve S, iki ilişki olmak üzere, bunların konumsal birleşiminini $R \bowtie_{i\Theta j} S$ olarak gösterirsek, RxS kümesinden, R'nin i. ve S'nin j. sütununun konumsal bir veri tipi barındırdığı ve θ'nın ikili bir konumsal ilişki öngördüğü kabulüyle R.i ve S.j arasında konumsal bir ilişki olması durumudur [49].

$$R \bowtie_{i\Theta j} S = \{R.i(\theta)S.j \, , \wedge RS \in RxS\}$$

Belirli bir uzayda bulunan nesnelerden oluşan büyük veri kümelerinden belirli bir alt uzaya giren adayları, tüm kümeyi dolaşmadan geri getirebilecek bir düzenin sağlanması sistemin genel performansı açısından son derece önemlidir. Bu yüzden *konumsal dizinleme* zorunludur. Bu dizinleme sayesinde farklı tipte nesneler, aralarındaki konumsal ilişkiler bağlamında bir araya toplanabilmelidir. Bunu yaparken de kartezten çarpımı sonrası yapılan filtreleme yönteminden daha iyi sonuç verdiğini garanti etmelidir.

2.2 Modelleme

2.2.1 Gösterim ve Gereksinimler

Konumsal veritabanı sistemleri ile ilgili araştırmaları sürükleyen temel uygulama alanı coğrafi bilgi sistemleridir. Bundan dolayı modelleme konusunda anlatılacaklar genellikle bu alanı ilgilendirmektedir fakat diğer uygulamalar için de geçerlidirler. Örnekler çoğu zaman 2-boyutlu uzaydan verilecektir, 3 veya daha fazla boyutlu durumlara genişleyebilme durumları da mevcuttur.

Modelleme söz konusu olduğunda, esas alınacak iki önemli nokta vardır :

- Uzaydaki nesneler : Herbiri kendi geometrik tanımlamasına sahip olan, uzayda bulunan ayrık varlıklar

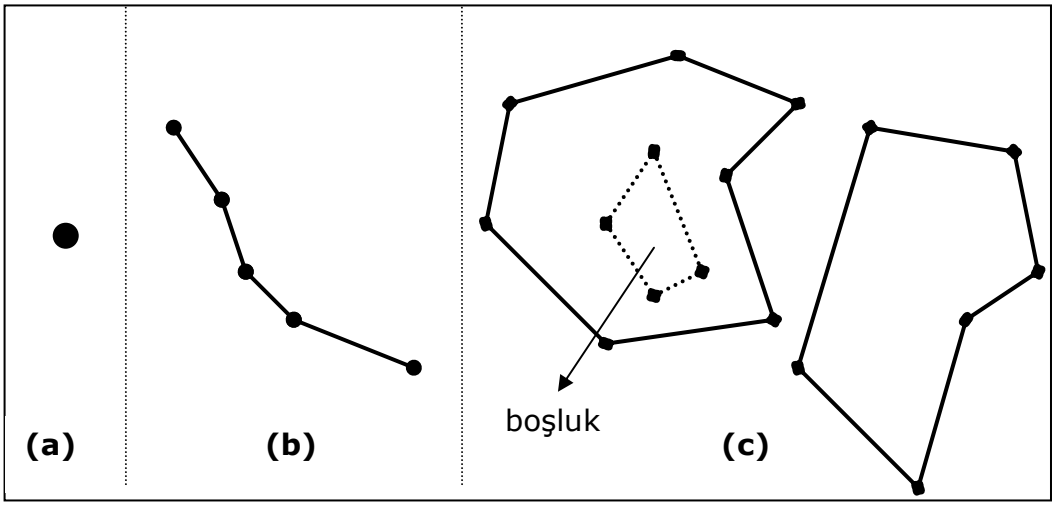
- Uzay : Uzayın kendisi, bir başka deyişle uzayı kaplayan her bir nokta

İlk madde şehirler, ormanlar, ırmaklar gibi uzayda yer tutan, konumsal nesneleri modellemekle ilgilidir. İkincisi ise tematik harita tanımı ile örtüşmektedir, örneğin toprak kullanımı veya bir ülkenin farklı bölümlerinin ifadesi. Taranmış görüntüler uzaydaki her nokta için bilgilendirici olduğundan ikinci görüşe yakındırlar. Tekil nesneler ile konumsal ilişkide bulunan nesne kümelerinin modellenmesi söz konusu olduğunda her iki görüşün de bağdaşması muhtemeldir.

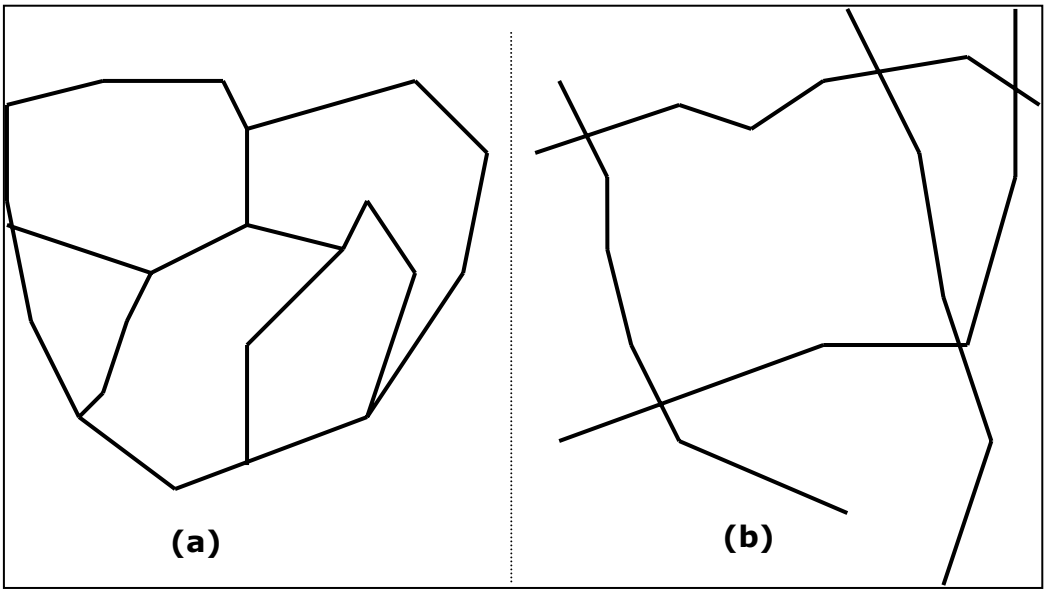
Tekil nesnelerin modellenmesi için temel soyutlamalar *nokta, doğru parçası* ve *bölgeler*dir. Geometrik ifadeyle bir *nokta*, bir nesneyi kaplam, yani uzayda kapladığı alan olarak değil, sadece yer olarak ifade eder. Fakat bu iki kavram birbiriyle alakalı olabilir. Örneğin bir şehir geniş bir coğrafi alanı ifade ederken yüksek ölçekli bir haritada bir nokta ile gösterilebilir. Bir *doğru parçası*, uzayda uzanmakta olan bir doğrunun başı ve sonu belli bir parçasıdır. Doğru parçalarının ucuca eklendikleri takdirde bir eğri oluşturduğu da düşünülebilir. Eğriler genellilkle çok parçalı, yani doğru parçası dizileri olarak gösterilir. Öyle ki uzay boyunca uzanan (yollar, nehirler, kablo ve elektrik hatları) veya uzaydaki bağlantıları gösteren konumsal nesnelerin temel soyutlama mantığını oluşturur. *Bölge* ise 2-boyutlu uzayda bir kaplama sahip olan nesneler (ülke, göl, milli park gibi) için bir modelleme aracıdır. Bir bölge boşluklar, delikler içerebilir veya ayrık parçalardan oluşabilir. Şekil 2.1, tekil nesneler için üç temel soyutlamayı örneklemektedir.

Konumsal açıdan alakalı nesne koleksiyonları örneklerinden en önemli iki tanesi bölümler ve ağlardır (Şekil 2.2). Bir *bölüm*, ayrık olması beklenen bölgelerden oluşan bir küme olarak görülebilir. Komşuluk ilişkisi yani ortak sınıra sahip bölgeler olduğunda ortaya çıkar. Bölümler tematik harita gösterimleri için

kullanılabilir(coğrafi alanlar, eyaletler). Bir *ağ* ise düzleme entegre edilmiş, düğümleri nokta nesnelerinden oluşan, geometrik kenarların doğru parçaları ile temsil edildiği çizge olarak ele alınabilir. Coğrafi olarak ağlar her yerde mevcuttur (yollar, nehirler, taşıma hatları, enerji nakil hatları).



Şekil 2.1 : 3 temel soyutlama : (a) nokta, (b) doğru parçalarından oluşan bir eğri ve (c) bölge



Şekil 2.2 : (a) Bölme ve (b) ağ

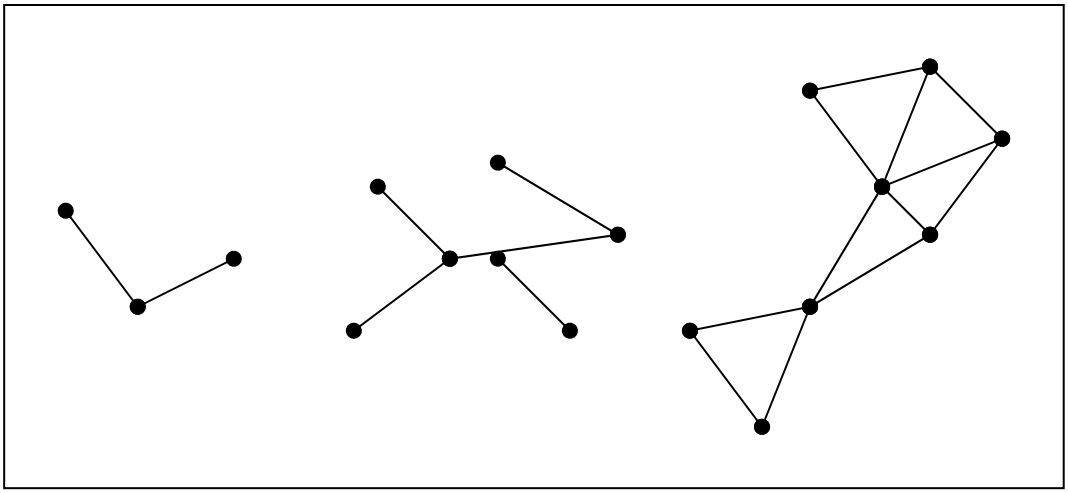
Verilen soyutlama örnekleri ve tanımlar, konumsal bir veritabanı yönetim sisteminde desteklenmesi gereken en temel soyutlamalar ve temel tanımlardır. Diğer konumsal alakalı nesne koleksiyonları *içiçe bölümler* kavramını getirmektedir (örneğin illere ayrılmış ülkeler, ilçelere ayrılmış iller). Modelleme hakkında daha detaylı bilgilere ulaşmak için [20, 63] kaynak olarak kullanılabilir.

2.2.2 Uzay Organizasyonu : Ayrık Geometrik Temeller

Geometrik modellemenin temeli olarak sıkça kullanılan veya içsel tanımlı olarak kabul edilen yapı, Öklit uzayıdır. Özünde şu anlama gelir; bir nokta bir düzlemde gerçel sayı çiftleri yardımıyla belitilir. Ama ne yazıktır ki bilgisayar ortamında gerçel sayı yoktur, sadece sonlu ve göreceli olarak sınırlı yaklaşımlar vardır. Bu da geometrik hesaplamalarda birçok probleme yol açmaktadır [25,21]. Örneğin iki doğrunun kesişim noktası, en yakın ve gösterilebilir ızgara (grid) noktasına yuvarlanabilir fakat takip eden işlemlerde bu kesişim noktasının, kesişen doğrulardan herhangi birinin üzerinde olup olmadığına bakıldığında yanlış sonuçlara varılabilir. Sonlu gösterim kullanımından kaynaklanacak potansiyel problemler modellemede göz ardı edilse bile, sorgu değerlendirme aşamalarında ortaya çıkacağı açık olan bu problemlerin çözümü konumsal veritabanı sistemini geliştirenlere bırakılmıştır. Bazı otoriteler buna istinaden modelleme açısından ayrık geometrik temelleri, gerçekleştirim adımları yanında öne sürmüşlerdir [19, 15, 30].

[19 , 15] de sunulan yaklaşım birleşimsel topolojiye dayanmaktadır. *Basitçe (simplex)* ve *basitçeler karmaşası* (*simplicial complex)* temel kavramlardır. d boyutta düşünüldüğünde d-basitçesi (d-simplex) o boyuttaki en küçük nesnedir. Örneğin 0-basitçesi (0-simplex) bir nokta, 1-basitçesi (1-simplex) doğru parçası, 2-basitçesi (2-simplex) üçgen olabilir. Herhangi bir d-basitçesi (d-simplex), d-1 boyutta d+1 bileşenden *(simplice)* oluşur. Mesela bir üçgen 3 adet doğru parçasından oluşur. Her bir doğru parçası ise 2 adet noktadan oluşur. Bir basitçeyi oluşturan bileşenler onun yüzleri olarak adlandırılır. Bir üçgen ele alındığında kenarları ve köşeleri buna örnek verilebilir. Bir basitçeler karmaşası ise herhangi

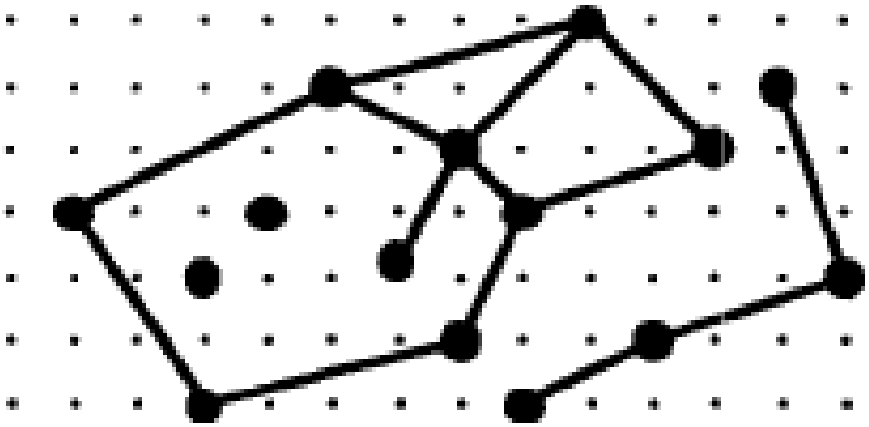
iki bileşenin kesişiminin bir yüz olduğu sonlu bileşenler kümesidir. Şekil 2.3'te 1-karmaşası (1-complex) ve 2-karmaşası (2-complex) örnekleri verilmiştir.



Şekil 2.3 : Basitçeler Karmaşası
(a) ve (b) 1-karmaşası (c) 2-karmaşası

Ayrık geometriye alternatif bir öneri *realm* kavramıdır[30]. Tanım olarak realm, uygulama uzayının tamamının altyapısını oluşturan geometrik temeldir. Esasen bir realm, ayrık bir ızgara (grid) üzerine yayılan nokta ve doğru parçalarının sonlu bir kümesidir. Öyle ki ;

i. Her bir nokta veya doğru parçasının bitim noktası ızgara üzerinde bir noktadır.
ii. Her doğru parçasının bitim noktası, aynı zamanda realm noktasıdır.
iii. Hiçbir realm noktası, doğru parçası üstünde yer alamaz. Örneğin bitim noktası olmamasına rağmen doğru parçası üzerinde olma durumu.
iv. Hiçbir realm parçası ikilisi bitim noktaları dışında kesişemez.



Şekil 2.4 : Örnek bir realm

Her iki yaklaşımda da amaç, geometrik nesnelerin, geometrik temellere dayanarak basit şekillerde modellenebilmesidir. Nokta, doğru parçası veya bölge nesnelerinin, bileşenler (simplice) cinsinden veya realm elemanı olarak ifade edilebileceği açıktır. Hatta konumsal bakımdan alakalı nesneler, örneğin bölümler ve ağlar, böyle bir geometrik temel üzerinde gösterilebilir. Paylaşılan ve paylaşımda bulunan geometrilerin tutarlılığı ve nesneler arası konumsal ilişkiler bu temel sayesinde ortaya konabilir.

Sayısal güçlülük sorunları geometrik tabanda ele alındığı takdirde üst katmanda tanımlanan konumsal veri tipleri veya konumsal cebir, kapalılık özelliklerinden sonuna kadar faydalanacak ve bu sadece teoride kalmayıp uygulamaya da yansıyacaktır [30]. Realm tabanlı konumsal veri tiplerinin sağladığı önemli avantajları örnekler yardımıyla açıklamak gerekirse şunları verebiliriz :

- Geometrik tutarlılık : Komşu 2 bölgenin ortak sınırı her iki bölge için de tamamen aynıdır.

- Konumsal cebir işlemlerinin kapalılık özelliği : Realm tabanlı iki konumsal verinin, kesişim, birleşim, fark gibi konumsal ilişkilerinin sonucu yine realm tabanlı bir konumsal veridir.

- Geometrik özerklik : Geometrik kavramlar, sayısal işlem hataları ve hassasiyet sorunlarından bağımsızdır. Ayrı bir katmanda görülürler.

2.2.3 Konumsal Veri Tipleri

Konumsal veri tipleri ve konumsal cebir sayesinde nokta, doğru parçası ve bölge gibi kavramlar için temel soyutlamalar yapılabilmekte, aralarındaki ilişkiler tanımlanmakta ve kompozisyon oluşturabilmeleri sağlanmaktadır (örneğin bölgelerin kesişimi, birleşimi).

Bölüm 2.1'de de belirtildiği üzere konumsal bir VTYS için konumsal verileri ve sorgu dillerini destekleyecek bir veri modeli sağlayabilmek zorunlu bir ihtiyaçtır. Konumsal veri tipleri ve işlemler birçok çalışmada tanımlanmıştır [6,45,31,40, 51,57,65]. Biçimsel tanımlamaya ilişkin çalışmalar [62,24,29] ile sonuç vermeye başlamıştır. Bu çalışmaların en başarılı örneklerinden bir tanesi [29]'da Güting ve Schneider tarafından sunulan, ROSE (Robust Spatial Extension) cebiri olarak bilinen konumsal cebir örneğidir.

ROSE cebirinde realm tabanlı 3 veri tipi öne sürülür : nokta, doğru parçası ve bölge. Bunlar realm elemanlarından oluşur. Bunları daha iyi anlamak için R-blok ve R-yüz terimlerine bakmak gerekebilir. Verilen bir R realmı üzerinde bir R-bloğu, R realmına ait, bağlantılı doğru parçaları kümesidir. R-yüzü ise realm parçaları üzerinde tanımlanabilen delikli poligonlardır. Noktalar, R-noktaları kümesinden değerler alır. Doğru parçaları ise ayrık R-bloklarının kümelerinden örneklenir. Bölgeler ise ortak kenarı olmayan, ortak köşelere sahip R-yüzlerinin kümesinden değerlere gider. (bkz. Şekil 2.1, Şekil 2.2)

ROSE cebiri, ikinci dereceden imzalara dayanır. Öyle ki çok şekilli işlemler daha basit türler aracılığıyla ölçeklenir. Temelde iki küme EXT = { doğru parçaları, bölgeler } ve GEO = { noktalar, doğru parçaları, bölgeler } olmak üzere 4 farklı işlem sınıfı vardır.

1) Topolojik ilişkileri ifade eden konumsal öngörüler:

$$\forall geo \in GEO. \forall ext1, ext2 \in EXT. \forall alan \in b\ddot{o}lgeler^{alan-ayrik}$$

geo X bölgeler	→ bool	**içinde**
ext1 X ext2	→ bool	**kesişir, karşılar**
alan X alan	→ bool	**komşu, kapsar**

geo değişkeni GEO kümesinde bulunan 3 farklı tipten değerler alır. Böylece *içinde* işlemi, noktaları, doğru parçalarını veya bölgeleri bir başka bölge ile karşılaştırabilir. *Kesişim* işlemi, EXT kümesindeki aynı veya farklı tipteki 2 değere uygulanabilir. bölgeler$^{alan—ayrık}$ gösterimi veri tipleri içerisinde bölümleri ifade etmeye yöneliktir. Bütün bölümler için değer kümesi belirtmeye çalışır. Bölümler, bölge nesnelerinin bir koleksiyonu olduğundan dolayı, bölgeler kümesinin herhangi bir alt kümesi şeklinde düşünülebilir. Her bir tekil bölüm (tematik harita) değerini bu alt kümelerden almaktadır. Böylece alan değişkeni bir bölümü gösterir ve komşuluk işlemi aynı bölüm içindeki herhangi 2 bölgeye uygulanabilir.

2) Atomik konumsal veri tipi döndüren konumsal işlemler:

$$\forall geo \in GEO$$

doğru parçaları X doğru parçaları	→ noktalar	**kesişim**
bölgeler X bölgeler	→ bölgeler	**kesişim**
geo X geo	→ geo	**artı, eksi**
bölgeler	→ doğru parçaları	**sınır**

artı ve eksi, birleşim ve farkı işaret etmektedir.İşleme alınan her iki değer de aynı tiptedir.

3) Sayısal değer döndüren konumsal işlemler :

$$\forall geo1 X geo2 \in GEO$$

geo1 X geo2	→ reel	**uzaklık**
bölgeler	→ reel	**çevre,alan**

4) Nesne kümeleri üzerine konumsal işlemler

OBJ, tanımlı nesnelerin herhangi bir kümesi olmak üzere;

$$\forall obj \in OBJ . \forall geo, geo1, geo2 \in GEO$$

küme(obj) X (obj→geo) → geo **toplam**
küme(obj) X (obj→geo1) X geo2 → küme(obj) **en_yakın**

toplam, konumsal bir grup fonksiyonudur. Nesne kümelerini konumsal özellikler üzerinden tipi GEO kümesinde tanımlı olan nesnelerle bir araya getirir, ve bütün özellik değerlerinin geometrik birleşimini sağlar. Örneğin ülkenin toplam alanını bulmak için, bütün illerin birleşimi.

en_yakın işleci ise sorgu nesnesine konumsal özellik bakımından en yakın mesafede bulunan nesnelerin kümesini belirler.

Verilen bilgiler ve örnekler konumsal cebir ile ne tür işlemler yapılabileceğini göstermektedir. Tipler ve işlemlerin anlamsal ve biçimsel tanımlamaları için [29]'a bakılabilir. Konumsal veri tipleri ve konumsal cebirle ilgili birkaç önemli konu şunlardır :

- Genişletilebilirlik : Tip ve işlem tanımlamalarının uygulamaya özel olduğu konusu açıktır. Bu sebeptendir ki alternatif, ek veri tipleri ve işlemler daha sonradan tanımlanabilmelidir. Bu temel ihtiyaç genişletilebilir olmayı gerektirmektedir.

- Bütünlük : Esas soru, tanımlı işlemlerden oluşan herhangi bir kümenin, bir şekilde bütünlüğü bozup bozmadığıdır. Bununla ilgili biçimsel bir kriter var mıdır? Bu konuya ilişkin sorunlar topolojik ilişkiler sayesinde çözülmeye çalışılmıştır.

- Tip çeşitliliği : Nokta, doğru parçası, bölge gibi farklı veri tiplerine gerçekten gerek var mıdır? Bazıları tek bir veri tipinin, örneğin *geometri*, bütün bu farklı tipteki veriler ve hatta karışık koleksiyonları için dahi yeterli olduğunu savunmaktadırlar[24,44]. Bu aslında alışılageldik bir sorudur. Bir sistem *tamsayı* ve *reel sayı* gibi farklı tipler yerine sadece *sayı* tipinde bir yapı sunabilir. Bu sayede işlemlerin kapalılığını sağlamak oldukça kolay olacaktır. Diğer taraftan birden fazla tip kullanımı ifade gücünü ve işlemlerin uygulama hassasiyetini artırır.

- Küme işlemleri : Konumsal cebir sadece atomik KVT değerleri (bir bölge, alanı çok geniş olsa bile atomik bir değerdir) için değil, konumsal açıdan ilgili nesne kümeleri (tematik haritalar, parseller) üzerine de tanımlı işlemler öne sürer [31,65,62,68]. Örneğin üst üste binen (overlay), çakışan iki bölüm, komşu iki bölümün birleşimi (fusion), ya da sorgu nesnesine en yakın nesneler. Bu türden işlemlerin gerektirdiği, VTYS veri modellerine ulaşmayı sağlayan arayüzler, atomik işlemlerden çok daha karmaşıktır [29].

2.2.4 Konumsal İlişkiler

Konumsal cebirde öne sürülen kavramlardan en önemlisi *konumsal ilişkiler*dir. Bu ilişkiler sayesinde sorgu nesnesiyle konumsal açıdan ilgili olan bütün diğer nesneler üzerine sorgular yöneltilebilmektedir, örneğin bir sorgu penceresine giren bütün nesneler. Bu ilişkiler [53,12,72] gibi çalışmalarda 3 ana sınıfa ayrılmıştır:

i. Topolojik ilişkiler : Komşuluk, içinde olma, ayrık olma, örtüşme gibi ilişkiler bu sınıfa girer.
ii. Yön ilişkileri : Yukarı, aşağı, kuzey, güney gibi yön içeren ilişkilerdir.
iii. Metrik ilişkiler : Uzaklık gibi sayısal değeri bulunan ilişkiler.

Bu ilişki türleri arasında en temel ve en çok çalışma yapılanı topolojik ilişkilerdir. Topolojik ilişkilerin tamamını gösterebilmek esas konudur. Geometrik

nesneler arasında ilişkileri gösterebilmek, ilişkisel işlemleri de ortaya koymak demektir. Bu amaca yönelik yapılan çalışmalarda [12,13] Egenhofer ve Herring bir yöntem ileri sürmüşlerdir. Geometrik nesneler arasındaki ilişkileri belirlemek amacıyla önerilen bu yöntemde karşılaştırılan iki geometrik nesnenin içleri, dışları ve sınırları arasındaki ikili ilişkiler test edilir. Test sonuçları kesişim matrisine göre değerlendirilir. Bu matrisin oluşumu için 4 kesişim modeli sunulmuştur[14]. Bu model daha sonra gelişen ihtiyaçlar doğrultusunda geometrik nesnelerin farklı ilişkilerini de kapsayacak şekilde genişletilerek 9 kesişim modeline geçilmiştir.

Karmaşık konumsal ilişkilerin modellenmesi hakkında temel bilgileri oldukça iyi ortaya koyabileceği düşüncesiyle 4 kesişim modeli hakkında genel bir bilgilendirme yararlı olacaktır. Bu ve genişletilmiş 9 kesişim modeli hakkında detaylı bilgi edinmek için [14,46,16] kullanılabilir.

2.2.4.1 Dört Kesişim Modeli

Dört kesişim modeli, Egenhofer[14] tarafından iki geometrik nesne arasındaki kesişimleri değerlendirip ilişkileri sınıflandırmak için geliştirilmiştir. Modelde nesnelerin iç ve sınır değerleri temel alınır. A nesnesinin içi A^0, sınırları ise ∂A ile temsil edilir. İki nesne için 4 ilişki kümesi vardır: $\partial A_1 \bigcap \partial A_2$, $\partial A_1 \bigcap A_2^0$, $A_1^0 \bigcap \partial A_2$, $A_1^0 \bigcap A_2^0$.

Bu durumda karşımıza Tablo 2.1'de gösterilen $2^4 = 16$ değişik olasılık çıkmaktadır.

$\partial A_1 \cap \partial A_2$	$\partial A_1 \cap A_2^0$	$A_1^0 \cap \partial A_2$	$A_1^0 \cap A_2^0$	İlişki
Ø	Ø	Ø	Ø	A_1 ve A_2 Ayrık
Ø	Ø	Ø	≠Ø	
Ø	Ø	≠Ø	Ø	
Ø	Ø	≠Ø	≠Ø	A_2, A_1 'in içinde
Ø	≠Ø	Ø	Ø	
Ø	≠Ø	Ø	≠Ø	A_1, A_2 'in içinde
Ø	≠Ø	≠Ø	Ø	
Ø	≠Ø	≠Ø	≠Ø	
≠Ø	Ø	Ø	Ø	A_1, A_2 'ye dokunur
≠Ø	Ø	Ø	≠Ø	A_1 ve A_2 eşittir
≠Ø	Ø	≠Ø	Ø	
≠Ø	Ø	≠Ø	≠Ø	A_1, A_2 'yi kapsar
≠Ø	≠Ø	Ø	Ø	
≠Ø	≠Ø	Ø	≠Ø	A_2, A_1 'yi kapsar
≠Ø	≠Ø	≠Ø	Ø	
≠Ø	≠Ø	≠Ø	≠Ø	A_1, A_2 'yi örter

Tablo 2.1 : 4 kesişim matrisi

2.3 Konumsal Sorgular ve Süzme Adımı

Bir önceki bölümde konumsal veri tiplerinden ve konumsal özellik barındıran nesneler arası ilişkilerden bahsetmiştik. Geometrik ve yapısal özelliklerinin dışında konumsal verilerin yönetim zorluklarının diğer sebeplerini incelemek gerekirse, ilk olarak karşımıza çıkacak olan şudur ki konumsal veriler sayısal olarak büyük miktarlarda bulunmaktadırlar ve çevreleriyle karmaşık ilişkilere sahiptirler. Kabaca bir örnek vermek gerekirse gerçek hayatta uygulamaya geçmiş bir konumsal veritabanı sisteminde veri miktarı 10.000'lerle ifade edilmekte ve bu sayı milyonlara ulaşabilmektedir. Bu nesnelerin çevresiyle

olan ilişkileri 4 kesişim modeliyle bile düşünüldüğünde ortaya çıkan hesaplama zorlukları ve iş yükü, daha fazla ilişkiyi gösterebilmemizi sağlayan 9 kesişim modeli söz konusu olduğunda çok daha karmaşık, çözümü zor hale gelmektedir. Örneğin bir şehir planı temel alınarak

"Çocuk oyun alanı içeren bütün parkları getir"

gibi bir konumsal sorgu verildiğinde konumsal ilişkilerin değerlendirilmesi gerekmektedir. Fakat konumsal işleçler, geleneksel seçim ve birleştirim işleçlerine oranla çok daha maliyetli ve zaman alıcıdır.

Bu konumsal işleçlerin verimliliği, konumsal verilerin gösterim şekillerine ve ilgili veri kümelerinin ne kadar hızlı geri getirilebildiğine bağlıdır. Yüksek miktarlarda veriye erişim söz konusu olduğunda ikincil saklama birimleri kuşkusuz ki devreye girecektir. Bu nedenle verimin artmasına yönelik olarak disk sayfalarına ve gereksiz verilere yapılan erişim sayıları azaltılmalıdır.

Konumsal ilişkiler içeren sorguların işletim verimliliği *dizinleme yöntemlerine* bağlıdır. Verilerin yoğunluğu ve çokluğu düşünüldüğünde olası bütün ilişkilerin önceden hesaplanması, ve gerektiğinde de önceden yapılmış olan bu değerlendirmeleri kullanmak gibi yaklaşımlar da öne sürülmüştür[69,7]. Fakat veritabanı açısından bakıldığında gereksiz veri miktarını artırmak ve olası güncellemeler sonrası ortaya çıkacak yeniden yapılandırma gibi ek yükler sebebiyle tercih sırasında geride kalmıştır. Böyle bir yaklaşımın aksine konumsal ilişkiler sorgu işleme esnasında dinamik olarak ele alınmalıdır. Nesnelerin konumsal özellikleri üzerine tanımlı dizin yapıları kurulmalı ve bu yapılar konumsal ilişkileri desteklemelidir.

Dizinleme yöntemlerinin çoğu *böl-fethet* ilkesine dayanır. Dizinler hiyerarşik yapılardır. Bu yapı birincil hafızaların sınırlı olduğu veritabanı sistemleri için çok uygundur. Hiyerarşik yapı sayesinde hedef alınan özel alt birimlere ulaşılır ve ilgilenilen veri sayısı azaltılır.

Hiyerarşik yapıların bir diğer avantajı ise konumsal verilerin üzerine yapılan pencere (alan) sorgularında verimli olmasıdır. Doğrudan sorgu penceresi ile belirlenen hedef alana yönelerek ilgi alanı dışında kalan veri kümesi dışarda tutulabilir. Fakat konumsal veritabanlarında durum geleneksel veritabanlarına göre daha farklıdır. Söz konusu veriler çok boyutlu nesneler olduğundan ve konumsal koordinatlarla ilişkili olduklarından dolayı, sorgulardaki arama kriterleri konumsal özellikleri barındırmaktadır.

Birçok konumsal dizinleme yöntemi, öyle veya böyle bir şekilde mevcut noktasal, tek boyutlu dizinleme yöntemlerini temel alır. Örneğin R-ağaçları B+-ağaçlarını, kd-ağaçları ise ikili arama ağaçlarını temel alır. Bu yöntemler temel aldıkları yöntemlerin yeteneklerini, konumsal verileri de kapsayacak şekilde genişletirler.

Konumsal verilerin geometrik özelliklerinden dolayı yapılarının geleneksel, tek boyutlu verilere göre karmaşık ve ilişkilerinin daha özel olduğunu birçok kez belirmiştik. Bunun sebebi şekillerinin düzensiz olmasıdır, yani her nesne kendine has bir geometriye sahip olabilir. Böyle bir durumda bunlar üzerine tanımlı işleçler maliyetli ve zaman alıcıdır. Örneğin 2 tane çok yüzlünün kesişimini bulmak istediğimizde her birinin bütün noktalarını diğerinin bütün noktalarıyla karşılaştırmak gerekecektir. Gerçek hayatta da yollar, göller, nehirler gibi coğrafi açıdan önemli olan varlıklar kesin öngörülebilir, sabit şekilli değillerdir. Bu belirsizlik, karmaşık veri tiplerine ve dolayısıyla sorgu işlem maliyetlerinin artmasına sebep olmaktadır.

Bir dizinleme yöntemi, gereksiz erişimleri engellemeli ve disk erişimlerini azaltmalıdır. Konumsal bir sorguda, tepki zamanı açısından temel amaç karmaşık konumsal verilerin karmaşık ilişkilerini değerlendirirken, üzerinde işlem yapılacak eleman sayısını azaltmaktır. Bunu etkin ve hızlı bir yolla yapmak, verilerin şekil belirsizliğinden sıyrılmayı, onları olduklarından daha basit bir şekilde ele almayı gerektirir. En genel görüş, nesneyi kapsayan, çevreleyen, bir başka deyişle nesnenin kendisini temsil edebilecek ama nesnenin geometrik yapısından daha basit bir yapıya sahip geometrik yaklaşımlar kullanmaktır. Bunun en geçerli örneği sınırlayıcı kutulardır. Sınırlayıcı kutu, konumsal bir nesneyi sınırlayan, kenarları

koordinat eksenlerine paralel olarak çizilebilecek en küçük dikdörtgen olarak tanımlanır. Şekil 1.5'te sınırlayıcı kutu örneği verilmiştir. Bu sınırlayıcı kutu üzerinden yapılan karşılaştırma olumlu sonuç verirse, nesnenin gerçek geometrisi detaylı karşılaştırma yapmaya değer niteliktedir. Aksi takdirde nesne ilgi dışıdır.

Aynı mantıkla ortaya konan fakat dikdörtgen yerine farklı geometrik şekiller tercih eden yöntemler de mevcuttur. Nesne etrafına sınırlayıcı çember, küre, dış bükey kabuk yerleştirmek gibi farklı uygulamalar yapılmıştır. Fakat temel amaç geometrik nesnelerin yapısal karmaşıklarından ve belirsizliklerinden uzaklaşıp daha basit bir yolla ele alınabilmelerini sağlamak olduğundan, en verimli seçenek sınırlayıcı kutulardır.

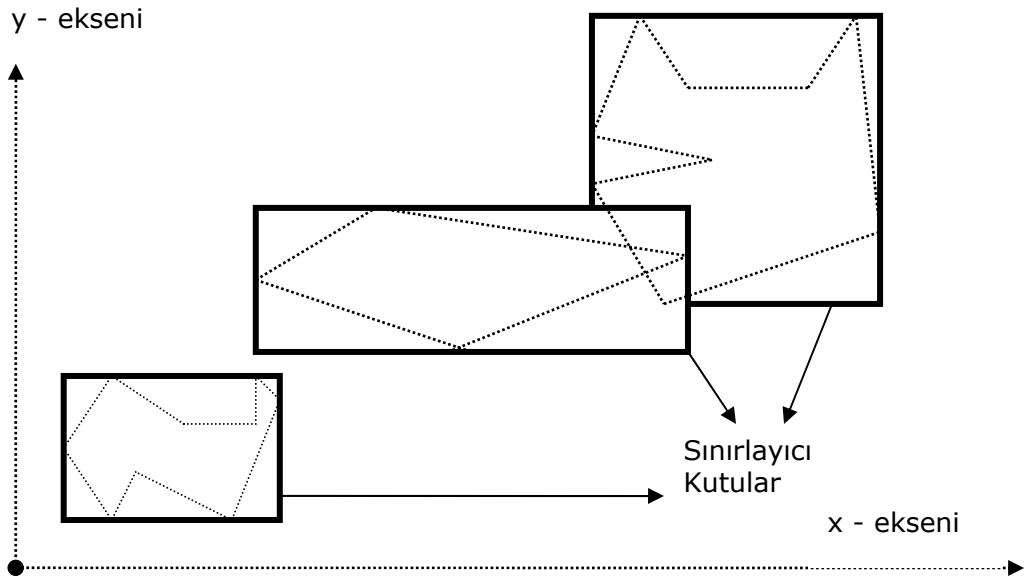
Bu sınırlayıcı kutular yardımıyla dizin yapıları kurulur, ve ilk arama bunlar üzerinden yapılır. Eğer nesnenin aranan nesne olma potansiyeli varsa, sınırlayıcı kutular konumsal açıdan benzerlik gösterecektir. Aksi durumda ise nesneler tamamen ayrıktır ve ilgi dışıdır, bu nesneler elenir. Böylece gereksiz erişimlerin önüne geçilmiş ve karmaşık konumsal işleçlerin uygulama alanı daraltılmış olur. Şekil 2.5'te, kesişen ve ayrık nesneler ile bunların sınırlayıcı kutuları örneklenmektedir.

Daha önce verdiğimiz 2 çok yüzlünün kesişimi örneğini tekrar düşünelim ve bunu işlemin sınırlayıcı kutular aracılığıyla yapıldığını varsayalım. İki nesnenin sınırlayıcı kutuları, sadece sol alt (minimum) ve sağ üst (maksimum) köşeleri dikkate alınarak karşılaştırılır. Herhangi bir kesişme, üst üste binme durumu söz konusu ise detaylı karşılaştırmaya geçilir. Eğer yoksa nesneler zaten elenmiştir.

Basit dikdörtgenlerin karşılaştırılması, karmaşık geometrik nesnelerinkinden çok daha karşılanabilirdir. Sürekli tekrarlanan arama işlemlerinde hesaplama zamanı azaltılır, hız ve verim artışı bu sayede sağlanmış olur.

Nesnelere geometrik yaklaşımlarda bulunmak, konumsal işleçlerin uygulama kümesini daraltmak söz konusu olduğunda yaygın kabul gören bir yoldur. Fakat bir dizin yapısının genel performansını veya verimliliğini bu geometrik yaklaşımlar doğrudan etkilemez. Dizin yapısının karakterini belirleyen,

kendi içinde organizasyon yapısını şekillendiren ve hiyerarşik yapının kurulmasına temel oluşturan mantıktır.



Şekil 2.5 : Ayrık ve Kesişen Sınırlayıcı Kutular

Desteklenen süzme tekniği ne olursa olsun, bir dizin yapısı ekleme, silme ve arama işlemlerine destek vermelidir. En genel ve geleneksel arama tipi olan anahtar tabanlı alan sorgusu (ilişkisel sorgu) hatırlanacak olursa, verilen aralığa düşen anahtarlara sahip tüm nesneleri geri getirdiği söylenebilir. Buna genel bir isim olarak *kesişim sorgusu* denilmektedir. Kesişim sorgusu, alan (range) sorgusu veya pencere sorgusu olarak da bilinmektedir. Bu çalışmada pencere sorgusu olarak anılacak olan bu sorgu tipi, karakter olarak, veri kümesine tutulan bir pencere içerisine giren, yani kesişen tüm elemanları işaretleyen bir sorgudur. Bir başka deyişle sorguda verilen alanla kesişim ilişkisi olan bütün nesneler sonuç kümesinde varolacaktır. Kesişim sorgusundan yola çıkılarak daha özel sorgu tipleri olan *nokta sorgusu* ve *kapsama sorgusu* gerçekleştirilebilir. Nokta sorgusunda sorgu penceresi aslında bir noktadır ve sonuç kümesine, bu noktayı içeren nesneler girer. Kapsama sorgusunda ise pencere sorgusundaki kesişim ilişkisi tamamen içerme, yani kapsama koşuluyla sınırlandırılmıştır. Pencere sorgusunda seçilen elemanlardan, verilen sorgu penceresinin tamamen içinde kalma koşulunu ihlal edenlerin elenmesi şeklinde uygulanabilir.

Arama işlemleri veritabanlarında en sık yapılan hareketlerdir. Bu yüzden dizin yapılarının konumsal seçim ve konumsal birleştirme işlemlerine elverişli olması mutlak bir gereksinimdir.

Konumsal seçim, konumsal bir koşul öne sürerek, konumsal özellikleri temel alıp koşulu sağlayan tüm nesneleri seçme işlemidir. "İç Anadolu Bölgesi'ndeki bütün illeri bul" sorgusunda İç Anadolu bölgesinin konumsal tanımı biliniyorsa bütün iller nesne bazında düşünülüp bu koşula uyan bütün iller seçilecektir.

Konumsal birleştirme ise iki farklı varlık kümesinden aynı konumsal koşula bağlı kalarak seçim yapılması ve ilgili olanların birleştirilmesidir. Ankara'nın şehir planına sahip olduğumuzu düşünelim ve bunun üzerinden bir örnek verelim. "Yanında otopark bulunan bütün alışveriş merkezlerini bul" sorgusunda bütün otoparklar ve alışveriş merkezleri 2 farklı varlık kümesinden gelen nesneler olarak ele alınacak ve komşuluk ilişkileri bakımından değerlendirileceklerdir.Bu türden konumsal sorguları gerçekleştirmek için ilk olarak dizin yapısı içerisindeki nesne yaklaşımları gözden geçirilecektir.

2.4 Dizinleme

Konumsal verilerin dizinlemesi söz konusu olduğunda, veri uzayının alt uzaylara parçalanarak daha küçük, yönetilebilir bölümler oluşturma fikri yeniden gündeme gelmektedir. Öz yinelemeli olarak sürdürülen bu bölümleme işlemi alt uzayların barındırdığı nesne kümeleri, tek disk sayfasında saklanacak düzeye inene kadar devam eder. Geri getirim zamanı ve saklama alanlarını indirgemek adına bir çok farklı bölümleme stratejileri ortaya konulmuştur. Geleneksel VTYS'lerden gelen ortak gereksinim olarak konumsal VTYS'lerde de dizin yapılarının adreslemesi gereken iki temel nokta şunlardır :

i. Saklama birimlerinin etkin kullanımı
ii. Erişim kolaylığı

Bu amaçlara yönelik birçok veri yapısı sunulmuştur. B-ağaçları, ISAM (Indexed Sequential Access Method) yapıları, kıyımlı arama (hashing), ikili ağaçlar uzun yıllar boyunca büyük veritabanlarında etkin erişim, ekleme ve silme olanakları sunarak kullanılmışlardır. Fakat bunların ortak özelliği tekil değerlere dayanmalarıdır, genellikle birincil anahtarlar üzerine kurulurlar. İkincil anahtarlar ile dizinleme yapabilmek adına dönüştürülmüş dizinler (inverted index) kullanılmış olsa da çok boyutlu konumsal veriler söz konusu olduğunda yeterli olmamıştır.

Konumsal dizinlemeye yönelik olarak ilk örnekler ızgara (grid) dosyaları, çok boyutlu B-ağaçları, kd-ağaçları ve 4'lü ağaçlardır. Bu yapılar nokta dizinleme için kullanılırlar. Temel tasarım ilkeleri konumsal nesneleri çok boyutlu uzayda bir nokta ile temsil etmektir. Fakat bu ilke pencere sorguları için yeterli değildir. Örneğin nesnelerin kesişim ilişkileri değerlendirilmek istendiğinde, nesneler sadece 2 nokta ile temsil edildiğinden dolayı gerçek bir kesişim sonunu elde edilememektedir.

2.4.1 B+-ağacı

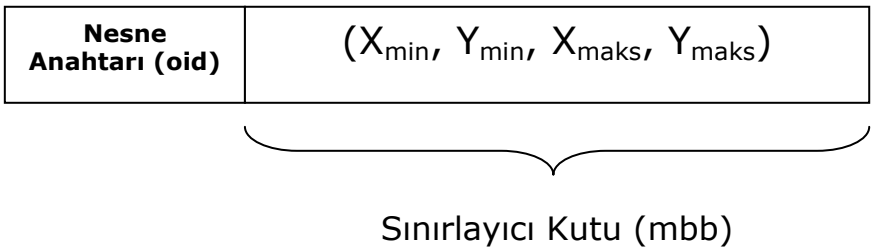
B+-ağacı tüm kayıtların ağacın yapraklarında tutulduğu ve yaprakların dizisel olarak bağlantılı olduğu bir dengeli ağaçtır, yani, özel bir B-ağacıdır. Şekil 1.4'te {2,3,7,9,10,13,15,23} *anahtar* değerlerine sahip bir B+-ağacı gösterilmektedir. Her bir düğümde girdilerin ([anahtar, ptr] şeklinde bir ikili ile *ptr* sayesinde, *anahtar* niteleyicisi ile temsil edilen bir nesneye işaret eder) olduğu bir disk sayfasını gösteren dengeli bir ağaçtır. Bu örnekte her bir sayfanın en fazla 4 adet girdi tutabileceği kabul edilmiştir. Düğümlerde tüm girdiler sıralıdır. Bir e girdisi için o.anahtar $\leq$ e.anahtar olanlar sol alt ağacı, o.anahtar > e.anahtar olanlar sağ alt ağacı oluşturur.

Bir *val* adlı anahtar değer verildiğinde, ağaçta yukarıdan aşağıya doğru bir arama başlatılır ve val $\leq$ e.key olacak şekilde en küçük girdi bulunur. Eğer *val* değeri düğümdeki girdiden büyükse en sağdaki alt ağaç seçilir. Böyle bir ağaçta

tüm yapraklar aynı derinliktedir ve okunması gereken sayfa sayısı ağacın derinliğine eşittir.

2.5 Konumsal Dizinleme Yöntemleri

Konumsal bir dizin yapısı, temel dizinleme mantığına uygun bir şekilde hizmet verirken kendine göre uyarlamaları da beraberinde getirir. Konumsal dizinlerde kayıtlar [oid, mbb] ikilileri biçiminde tutulur. Bu yapıda, *oid,* en küçük çevreleyen dikdörtgeni, yani sınırlayıcı kutusu *mbb* olan geometrik nesneyi tanımlayan bir anahtar değeridir ve diskte saklanan veriye erişim sırasında birincil anahtar olarak kullanılır.



Şekil 2.6 : Dizin Kayıt Yapısı

Konumsal dizinlerde kullanılacak olan veri yapılarının kendini ispatlamış geçerli örnekleri bulunan B+-ağaçlarını temel alması akılcı bir seçenektir. Bu sayede sorgular için gerekli I/O işlemlerinde performans logaritmik olarak değişecektir (sabit kolleksiyon boyu için). Ayrıca B+-ağacı bize bir *anahtar* tanım kümesi üzerinden, yaprak seviyesinde bir tam-sıralama sağlayacaktır.

Dizinleri oluştururken temel tasarım ilkelerinden biri de alt uzaylara bölümleme sırasında, bu alt uzayların sınırlarının kesişimlerinin mümkün olduğunca aza indirgenmesidir. Bir başka deyişle alt uzaylar olabildiğince kesin bir düzende ayrık olmalıdır. Aksi takdirde aynı nesneler, farklı uzaylarda tekrarlı görünecektir. Bu ilkeyi yeterince uygulayamayan bir dizin yapısı içerisinde dolaşırken tekrarlı yollar kullanmak performans açısından kötü etki yapacaktır. Bunun yanında alt uzayların sınırlarının dinamik olarak belirlenebilmesi, sık güncellemeler yaşanan bir sistemde ek yük bindirecektir.

Yapılan araştırmalar ve çalışmalar sonucunda, varolan bir tekniği genişleten veya yeni bir bakış açısı getiren birçok dizinleme yapısı ve yöntemi öne sürülmüştür. Her bir yöntemin kendine göre zayıflıkları ve güçlü yanları olmasından dolayı, bu yöntemleri birbirlerinin zayıflıklarını örtecek bir kombinasyonla kullanan uygulamalar vardır. Fakat bu sefer de gözden kaçırılmaması gereken nokta, bir yöntemi temel alıp genişleterek farklı bir yaklaşım ortaya koyarken, temelden gelen kalıtımsal zayıflıkların bulunduğudur.

Konumsal dizinleme yöntemlerinin çoğu belirleyici değildir (nondeterministic). Verilerin eklenme sırası dizinleme ağaçlarının yapısında farklılıklar yaratabilmektedir. Öyle ki aynı veri kümesine sahip olan dizinler, farklı performanslar gösterebilmektedirler. İyi bir dizinleme yönteminde veri ekleme alogritması, farklı ekleme sıralarının performans farkı yaratmasını engelleyecek, dinamik bir yapıda olmalıdır. Konumsal dizinleme yönteminin tasarım çerçevesi çizilirken en aza indirgenmesi gereken kavramlar şöyle verilebilir :

- İç (ara) düğümlerde saklanan dikdörtgenlerin kapladığı toplam alan
- Dizin yapısı içerisinde oluşturulan alt uzayların kesişimleri
- Farklı alt uzaylarda tekrarlanan nesne anahtarları
- Dizin boyutu ve derinliği

Bu gereksinimleri olduğu gibi tümden karşılayan bir dizin yapısı ulaşılmak istenen ideal noktadır. Fakat bu nokta gerçek bir uygulamada her zaman yakalanamamaktadır. Bir dizin yapısı kurulurken günün şartları düşünülerek işlem gücü bazında hesaplama güçlükleri de dikkate alınmalıdır. Bunun yanında dizin performansını etkileyen dış faktörlere belli başlı örnekler vermek gerekirse ilk sıralarda gösterilecek olanlar şöyle verilebilir.

- Tampon bellek tasarımı
- Tampon değişim stratejileri
- Disk yönetimi
- Eş zamanlı erişim mekanizmaları

2.5.1 Uzay Güdümlü Yöntemler

Veri uzayını temel alarak bir yapı kurmayı amaçlayan yöntemlerden oluşan bu genel kategoride ilk olarak değinilmesi gereken nokta, *ızgara (grid)* ve *ızgara dosyası* kavramlarıdır. Izgara dosyası çeşitli niteleyicilerin değerleri üzerinden nesneleri dizinlemek amacıyla kullanılır. Bu yapıların daha özel ve gelişmiş uzantıları, ticari ilişkisel veritabanı yönetim sistemlerinin konumsal destek verecek şekilde genişletilmeleri için de kullanılmıştır. Örneğin Oracle Spatial, doğrusal bir 4'lü ağaç ile donanmıştır.

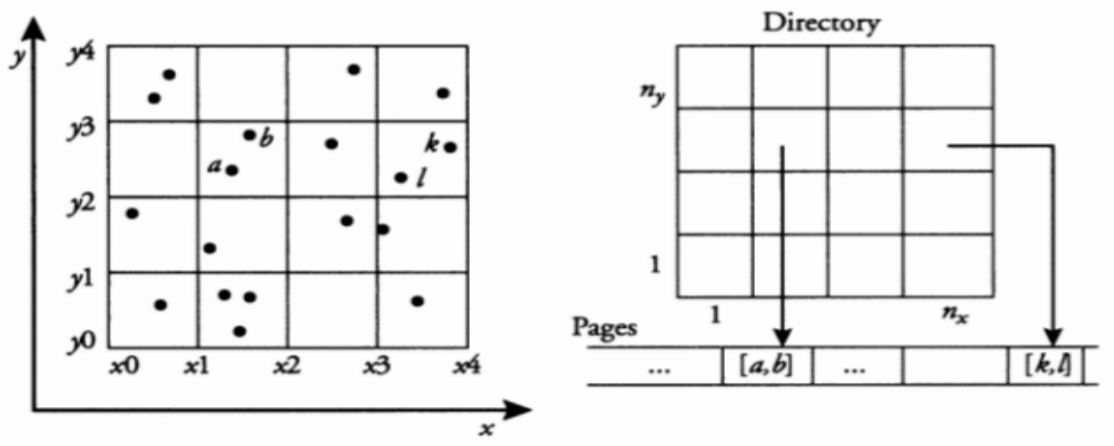
2.5.1.1 Izgara Dosyası (Grid File)

Izgara dosyası, devingen kıyım (hash) teknikleri ile düzenlenen bir veri yapısıdır. Çok boyutlu, noktasal ve noktasal olmayan verilere yönelik bir konumsal erişim yöntemi olarak önerilmiştir. [41,37,38]. Izgara temelli bir erişim yönteminin avantajı, dizin yapısın en başta genel hatlarıyla belli olmasıdır. Daha sonradan eklenen veriler, dizin yapısı içindeki yerlerini alırlar. Bu yüzden, bu tür yöntemler uzay güdümlü olarak tanımlanmanın yanında veriden bağımsız olarak da anılırlar.

Sabit Izgara

Arama uzayı dikdörtgensel hücrelere parçalanır. Sonuçta elde edilen, eşit-ebatlı hücrelerin bir n_x x n_y dizisidir. Her bir hücre bir disk sayfası ile ilişkilendirilir. Eğer bir c hücresi ile ilişkili olan c.rect dikdörtgeni bir P noktasını içeriyorsa, P noktası c hücresine atanır.

Şekil 2.7'de 18 noktadan oluşan sabit bir ızgaranın dizinlenmesi görülmektedir. Arama uzayı için (x0,y0) koordinatları orijin kabul edilmiştir. Bu dizinleme DIR[1:n_x ,1:n_y] şeklinde 2 boyutlu bir dizi oluşturmaktadır. Dizinin her bir DIR[i,j] öğesi, $c_{i,j}$ hücresi ile ilişkili olan noktaların tutulduğu bir sayfanın PageID adresini içermektedir. Eğer arama uzayının ebadı [S_x,S_y] ise, her bir hücrenin ebadı [S_x/n_x, S_y/n_y] olmaktadır.



Şekil 2.7 : Sabit Izgara dizin yapısı

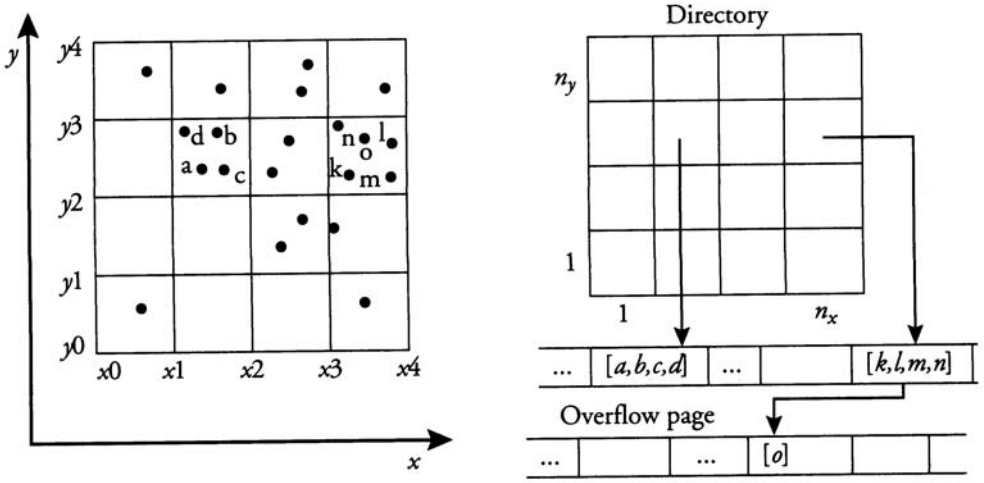
- P(a,b) noktasının ilave edilmesi: $i=(a-x_0)/(S_x/n_x)+1$ ve $j=(b-y_0)/(S_y/n_y)+1$ değerleri hesaplanır, daha sonra DIR[i,j].PageID sayfası okunur ve P noktası eklenir.

- Nokta Sorgulama: P(a,b) noktası verildiğinde, ekleme yaparmış gibi ilgili sayfaya gidilir, sayfa okunur daha sonra kayıtlar taranır ve P noktasının var olup olmadığı kontrol edilir.

- Pencere Sorgulama: W penceresi tarafından örtülen c hücrelerinin c.rect dikdörtgenlerinin S kümesi alınır, S deki herbir $c_{i,j}$ hücresi için DIR[i,j].PageID sayfası okunur ve W tarafından kapsanan noktalar döndürülür.

Izgaranın çözünürlüğü, dizinlenecek olan nokta sayısı olan N değerine bağlıdır. M nokta kapasiteli bir disk sayfası verildiğinde, en az N/M sayıda hücre içeren bir sabit ızgara oluşturulur. Her bir hücre ortalama olarak M sayıda nokta içerir. M adetten daha fazla sayıda nokta içeren bir hücre "*taşma"* (overflow) olarak işaretlenir. Bunlar taşma (overflow) disk sayfaları ile ilişkilendirilir. Taşma durumunu anlamak için, her bir disk sayfasının en fazla dört adet kayıt içerdiğini kabul edelim. Nokta dağılımı Şekil 2.8'deki gibi olsun. k, l, m, n ve n noktaları ile ilişkili olan hücre taşma durumundadır. Bu sebeple k, l, m ve n noktaları p

sayfasında tutulur. o noktası ise, p sayfasının işaret ettiği bir taşma disk sayfasında tutulur.



Şekil 2.8 : Sabit Izgarada Taşma Sayfası

2.5.1.2 Izgara Dosyası ile Nokta Dizinleme

Tıpkı sabit ızgarada olduğu gibi, ızgara dosyasında da her bir hücre bir disk sayfası ile ilişkilendirilir. Burada farklı olarak taşma olan hücreler iki ayrı hücreye bölünür. Bu yüzden hücreler eşit ebatlı değildir ve üç farklı veri yapısı gereklidir:

- Hücreler ile ilişkili olan sayfalara işaret eden DIR yapısı (dizin), 2 boyutlu bir dizidir. Yapı sabit ızgaraya benzer. Önemli olan fark iki komşu hücrenin aynı disk sayfasını gösterebilmesidir.

- S_x ve S_y ölçeklendirmeleri koordinat ekseninin aralıklara ayrılacağı koordinat değerlerini tutar.

Şekil 2.9'da bir ızgara dosyasının oluşturulmasında gerçekleşen dört adım görülmektedir. Burada sayfa kapasitesinin M=4 olduğu kabul edilmiştir.

A adımında, yapı sadece dört nokta içermektedir. Dizin tek bir hücreden oluşmaktadır ve bu bir p1 sayfası ile ilişkilendirilmiştir. B adımında ise a, b ve c ardışık olarak ilave edilmiştir. a noktası ilave edildiğinde yeni bir p2 sayfası için bellekte yer açılmıştır, daha sonra beş adet nokta p1 ve p2 sayfalarına dağıtılmıştır. Dikey bölünme x_1 değerli S_x ölçeklendirilmesi ile yapılmıştır. a ve b noktaları p1 sayfasına ve c noktası p2 sayfasına ilave edilmiştir.

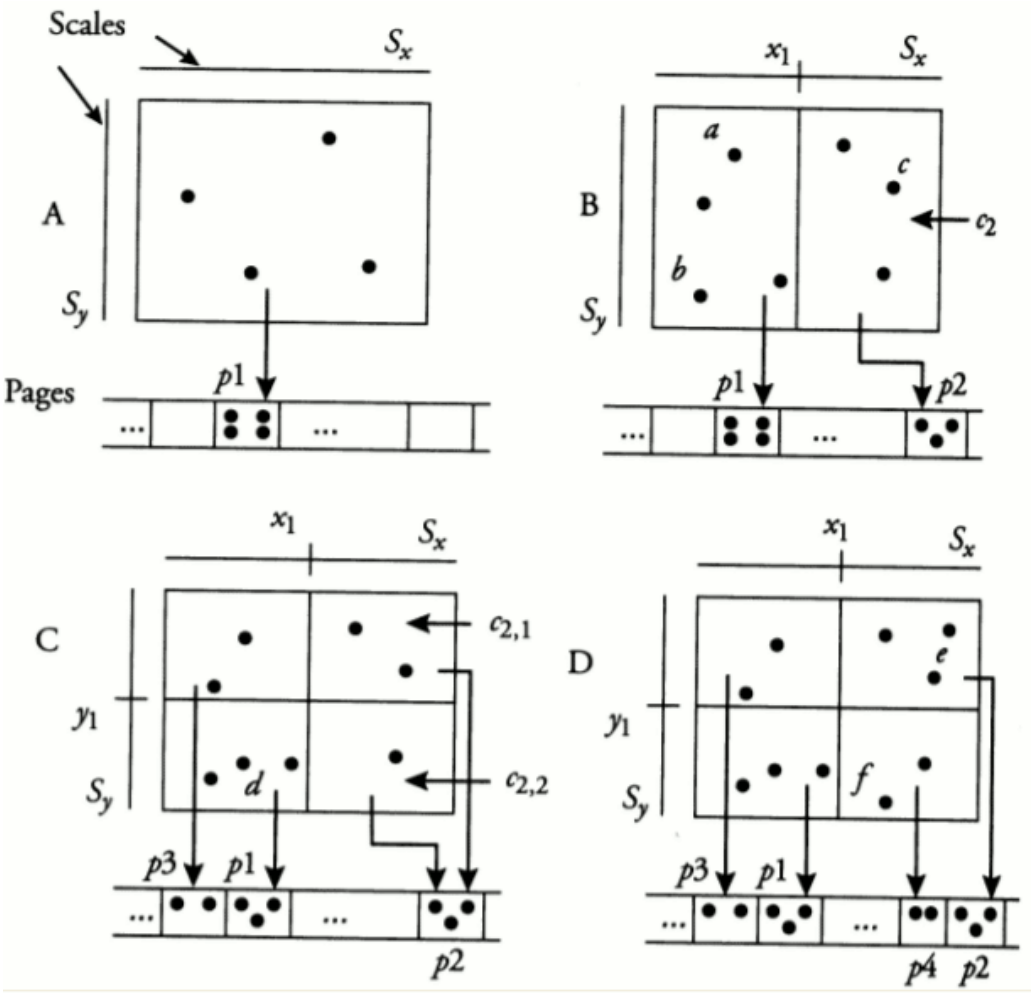
C adımında, d noktası ilave edilmiştir. p1 sayfası dolu olduğu için p1 'e ilave edilememiştir. Bu sebeple bir yatay bölünme işlemi yapılmıştır. Bu y_1 değerli bir S_y ölçeklendirilmesi ile yapılmıştır. Beş adet nokta p1 ve p3 sayfalarına dağıtılmıştır.

Açıkça görülmektedir ki bir bölünme işlemi tüm hücrelerde bölünme yaratmaktadır. Buna karşın, taşma olmayan hücreler aynı sayfaları göstermeye devam etmektedirler. c_2 hücresi yerine $c_{2,1}$ ve $c_{2,2}$ hücreleri gelmekte ama her ikisi de aynı p2 sayfası ile ilişkilendirilmektedir.

Son olarak D adımında, e ve f noktaları ilave edilmektedir. Bu halde p2 sayfası taşma durumuna düşmekte ve yeni bir p4 sayfası için bellekte yer açılmaktadır. Bu aşamada aralık bölme işlemine gerek yoktur, çünkü DIR yapısı içinde zaten tanımlı bir bölme vardır. $c_{2,1}$ noktası p2 sayfasında tutulur. $c_{2,2}$ noktası ise, p4 sayfasında tutulur. Bir P noktası ilave edildiğinde üç farklı durum göz önüne alınmaktadır.

- Hücre bölünmesi yok ise : En basit durumdur. P noktası dolu olmayan sayfalardan birinde tutulur.

- Hücre bölünmesi var fakat dizini bölme yok ise : P noktası c hücresi içine yerleştirilir. c ile ilişkili olan p sayfası dolu olduğundan yeni bir $p^{/}$ sayfası açılır ve c ile ilişkilendirilir.



Şekil 2.9 : Izgara Dosyasına veri ekleme

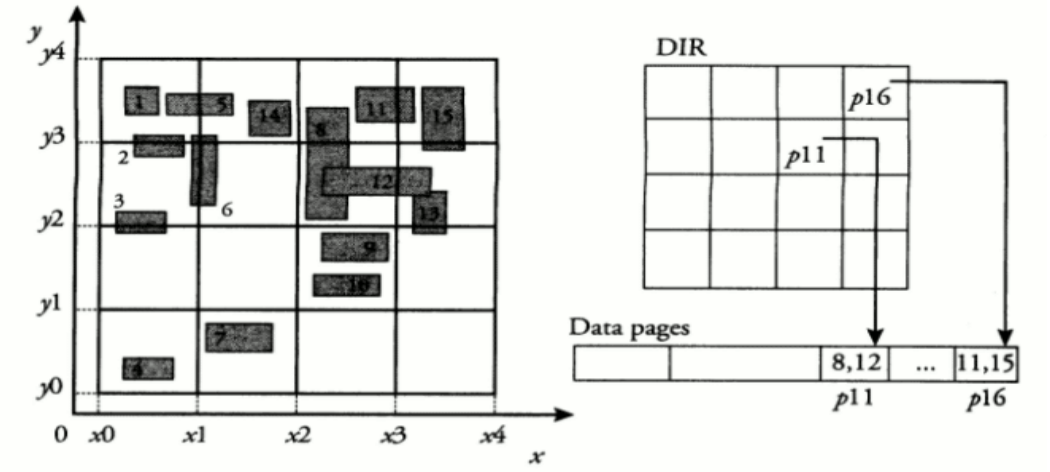
- Hücre bölünmesi ve dizin bölünmesi var ise : En karmaşık durumdur. $c_{i,j}$ hücresi ile ilişkili olan p sayfası doludur ve ilişkilendirilecek başka bir sayfa da yoktur. Bu halde $c_{i,j}$.rect dikdörtgeni x ya da y'ye göre bölünmelidir. Kabul edelim ki, bölünme işlemi x eksenine göre yapılsın. Bu durumda S_x ölçeklendirilmesine göre hücre iki aralığa bölünecektir ve i+1 mertebesi ile S_x içine bir absis eklenecektir. Eski $c_{i,j}$ hücresi $c_{i,j}$ ve $c_{i+1,j}$ olarak iki hücreye bölünecektir. Yeni bir p' sayfası oluşturulacak ve bu $c_{i+1,j}$ hücresi ile ilşkilendirilecektir. Geriye noktaların p ve p' sayfaları arasında dağıtılması kalmaktadır. DIR içinde yer alan l mertebesi ile tüm önceki sütunlar, l>i olanlar l+1 olarak anahtarlanacak ve yeni bir sütun i+1 damgası ile oluşturulacaktır. Son olarak da her $k \neq j$ için yeni $c_{i+1,k}$ hücresi $c_{i,k}$ hücresi olarak atanacaktır (C adımı).

2.5.1.3 Izgara ile Dikdörtgenlerin Dizinlenmesi

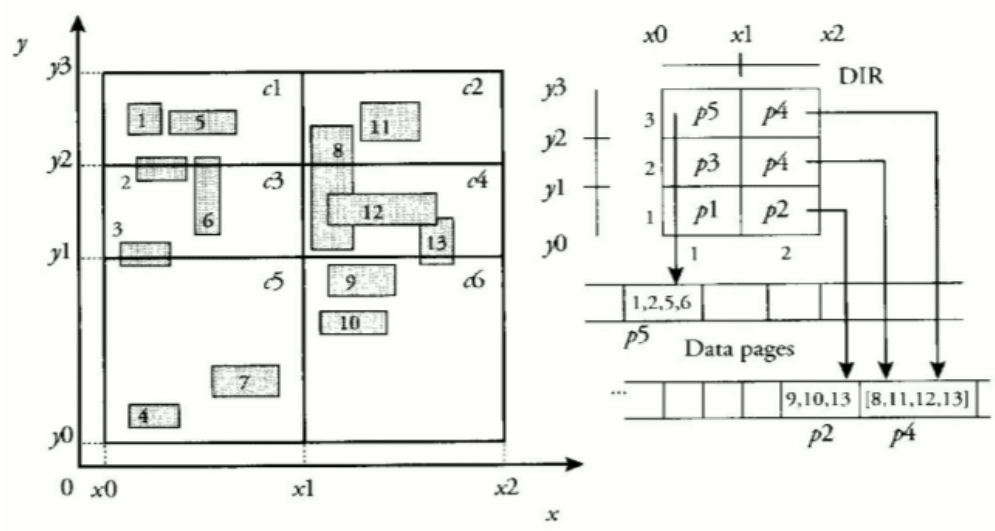
Noktasal olmayan, çok boyutlu verileri dizinlemek uğruna ızgara dosyası temel alınarak yeni yollar denenmiştir [39]. Dikdörtgenleri hücrelere dönüştürmek için en basit yol dikdörtgeni örten hücreleri kullanmaktır. Üç farklı durum vardır :

- Dikdörtgen hücreyi içerir.
- Dikdörtgen hürceyle kesişir.
- Hücre dikdörtgeni içerir.

Şekil 2.10'da 15 adet dikdörtgenden oluşan bir sabit ızgara oluşumu örneklenmiştir. Şekil 2.11'da ise, aynı dikdörtgenlere karşı gelen ızgara dosyası görülmektedir. Her iki durumda da sayfa kapasitesi 4 kabul edilmiştir.



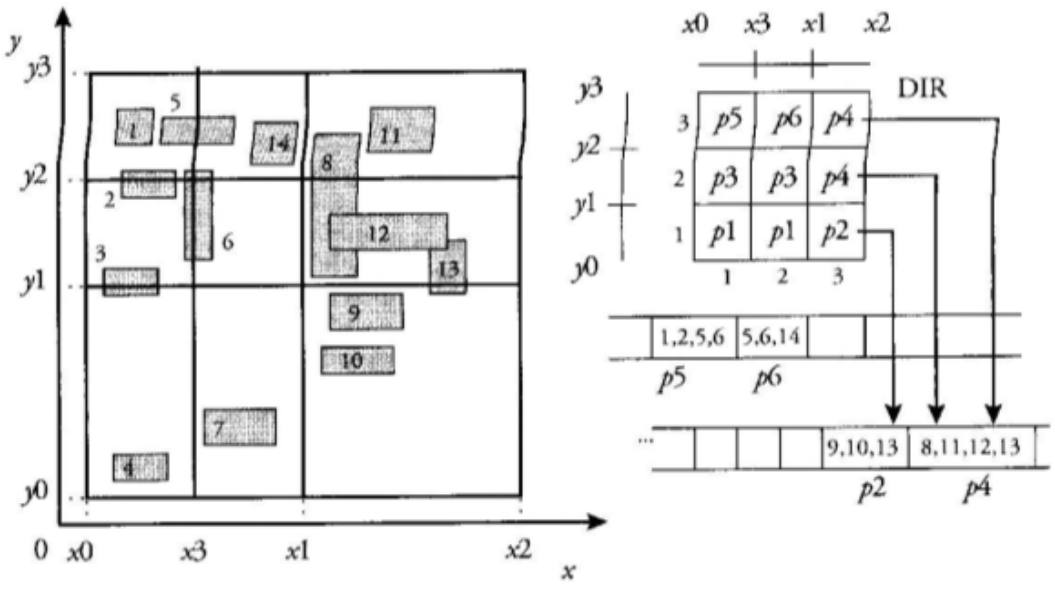
Şekil 2.10 : Sabit Izgara ile Dikdörtgen dizinleme



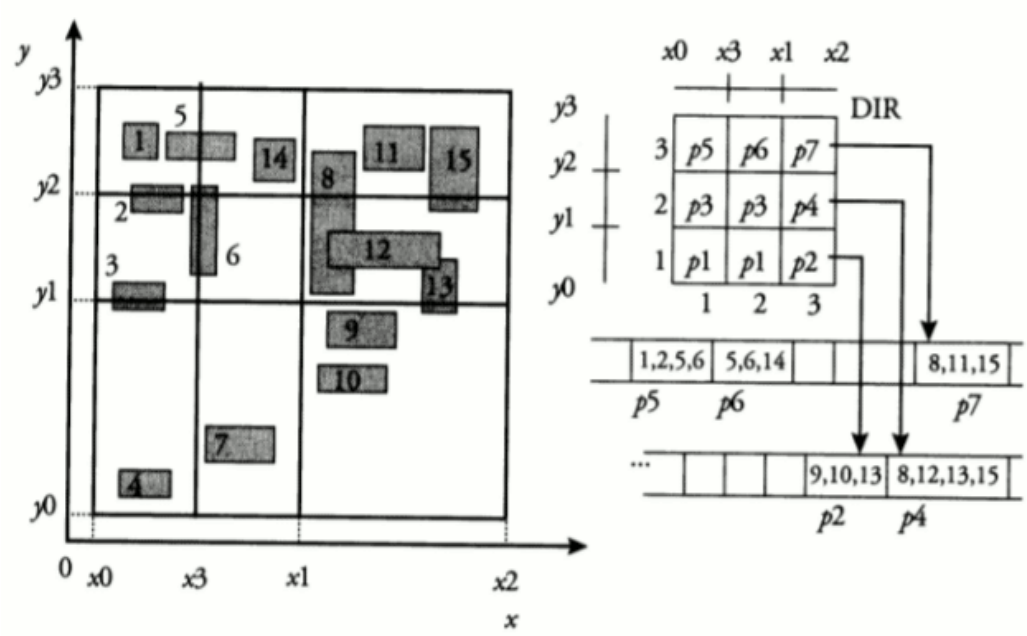
Şekil 2.11 : Izgara Dosyası ile Dikdörtgen dizinleme

Şekil 2.12'de ise yeni bir dikdörtgen ilave etme işleminin ilk aşaması gösterilmektedir. Şekil 2.12'de görüldüğü gibi 14 numaralı dikdörtgenin DIR[1,3] hücresine ilave edilmesi p5 sayfasını taşma durumuna düşürmektedir. Bu durumda x3 değerinden yeni bir sınır oluşturulur ve DIR[1,3] hücresi x eksenini keserek bölünür. x3 değeri S_x de ilave edilir ve dizin 3x(2+1)=9 hücre içerir duruma gelir. DIR[2,1] hücresi ve DIR[2,2] hücresi var olan sayfalara (p1 ve p3) ilişkilidir. p5 hücresindeki nesneler p5 ve p6 sayfalarına dağıtılır (bunlar ise DIR[1,3] ve DIR[2,3] ile ilişkilidirler).

Dizin yapısında bir bölünmeye yol açmayan bir ekleme Şekil 2.13'te gösterilmiştir. 15 numaralı nesne ilave edildiğinde p4 sayfası taşma durumuna gelmektedir. Bu durumda bir p7 sayfası oluşturulur ve DIR[3,2] ve DIR[3,3] hücreleri dizin yapısı içerisinde güncellenir.



Şekil 2.12 : Izgara Dosyası veri ekleme



Şekil 2.13 : Dizin bölünmesi yaratmayan ekleme

2.5.1.4 Izgara Dosyasında Nokta ve Pencere Sorgulama

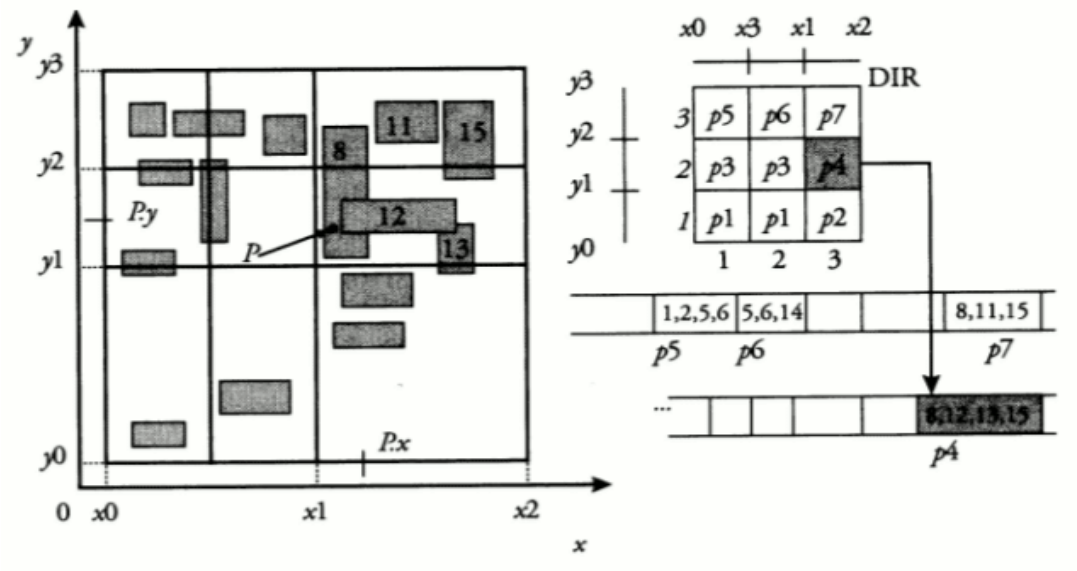
Nokta sorgulama işleminde ilk önce verilen bir P(a,b) noktası için, önce onu içeren bir hücre olup olmadığına bakılır. Sabit ızgaralarda bu, doğrusal zamanda yapılabilmektedir. İlk önce hücre ile ilişkilendirilmiş olan sayfayı okumak amacıyla bir kez diske erişmek gerekir. Daha sonra E kayıtlarının kolleksiyonu elde edilir. Geriye, E içerisindeki her bir e kaydı için e.mbb dikdörtgeninin P noktasını içerip içermediğinin test edilmesi kalmaktadır. Üçüncü adımda e.oid tanımlayıcılı nesne döndürülür.

Aşağıda görülmekte olan algoritma, sınırlayıcı kutusu P noktasını içeren nesne tanımlayıcılarının kümesini döndürür. RANK(v,S) ise v yi içeren, S dizisindeki aralığın sırası olan bir tamsayıyı döndürür. READPAGE(p) fonksiyonu ise p sayfasını belleğe getirir.

```
GF-POINTQUERY (P(a,b):point)
begin
        result = ∅
        // P noktasını içeren hücreyi bul
        i = RANK(a,Sx); j= RANK(b,Sy)
        page = READPAGE (DIR[i,j].PageID)
        for each e in page do
                if (P∈e.mbb) then result+= {e.oid}
        end for
        return result
end
```

Algoritma 2.1 : Izgara Dosyası Nokta Sorgulama

Izgara dosyası üzerinde bir P noktasının sorgulanmasına ilişkin bir örnek Şekil 2.14'te görülmektedir. Örnekte DIR[3,2] hücresi döndürülmekte ve dört adet kayıt içeren p4 sayfası belleğe yüklenmektedir. Dört adet kayıt üzerinden test yapıldığında ise 8 ve 12 kayıtları sonuç olarak gelmektedir.



Şekil 2.14 : Izgara Dosyası Nokta Sorgulama

Pencere sorgulama algoritması, nokta sorgulamasının bir adım ileriye gitmiş hali olarak düşünülebilir. İlk adımda pencereyi örten tüm hücreler hesaplanır. İkinci adımda bu hücrelerin her biri nokta sorgusunda olduğu gibi taranır. Sonuç kümesi içerisinde tekrarlı nesneler olabileceğinden ayıklama yapılması gerekir. Bu ayıklama, sıralama algoritması ile sağlanır.

```
GF-WINDOWQUERY (W(x1,y1,x2,y2): rectangle)
begin
  result = ∅
  // W ile sol-alt köşeden kesişen hücreyi bul
  i1 = RANK(x1,Sx); j1=RANK(y1,Sy)
  // W ile sağ-üst köşeden kesişen hücreyi bul
  i2 = RANK(x2,Sx); j2=RANK(y2,Sy)
  // Bulunan aralıktaki hücreleri tara
  for (i=i1; i≤i2; i++) do
          for (j=j1; j≤j2; j++) do
                  // Sayfaları oku ve her bir kaydı test et
                  page = READPAGE(DIR[i,j].PageID)
                  for each (e in page) do
                          if (W∩e.mbb≠∅) then result +={e.oid}
                  end for
          end for
  end for
```

```
  // Sonuçları sırala ve tekrarlı kayıtları yok et
  SORT(result); REMOVEDUPL(result)
  return result
end
```

Algoritma 2.2 : Izgara Dosyası Pencere Sorgulama

Altı çizilmesi gereken iki önemli husus şöyledir :

♦ Tekrarlı nesnelerin ayıklanması maliyetlidir. Zaman karmaşıklığı eleman kümesine oranla çok hızlı büyümektedir.

♦ Birden fazla disk sayfasının geri getirim hızı, ardışık disk sayfaları söz konusu olduğunda daha yüksek olacaktır.

Nokta sorgularının I/O performansının zamana göre sabit olduğu ortadadır. Fakat pencere sorgularında bu performans sorgu penceresinin ebatlarına bağımlıdır. Büyük bir pencere, sorgu zamanında artışa sebep olacaktır.

Izgara yapısının, dikdörtgen dizinleme konusunda birçok temel gereksinimi karşıladığını her ne kadar bazı problemler hala devam etse de söyleyebiliriz. Tekrarlı nesneler dizin yapısındaki kayıt sayısını, dolayısıyla da dizin ebadını artırır. Bu ters rüzgar, veri kümesinin eleman sayısı yüksek değerlere ulaştığında daha da sertleşmektedir. Bunun bir diğer yan etkisi de veri sayısı arttıkça dizin hücrelerinin ebatları, nesnelerin sınırlayıcı kutularına yaklaşmaktadır, yani küçülmektedir. Benzer şekilde veri sayısı arttığında tekrarlı kayıtların getirdiği yük de artmaktadır. Bu gibi zorlukların üstesinden gelmek için konumsal dizin yapısının birincil (ana) bellekte tutulması verimlilik açısından önemli bir yaklaşımdır. Fakat gerçek hayatta çok çok büyük veri kümeleri söz konusu olduğunda, deneyimler göstermiştir ki bu da tam bir çözüm olamamaktadır. Dizin yapısı kısmi de olsa disk üzerine aktarıldığında ise yönetimi zorlaşmakta ve tepki süresi artmaktadır.

2.5.2 Veri güdümlü Yöntemler

Uzay güdümlü yöntemlerden farklı olarak, genel hatları baştan belli olan bir yapı yerine, kendisini veri dağılımına göre düzenleyen, uyarlayan bir ilke ile ortaya çıkan veri güdümlü konumsal erişim yöntemlerinin en belirgin örneği R-ağaçlarıdır. Veriye uyum gösterebilme yetenekleri sayesinde saklama birimleriyle daha yüksek bir uyum sağlamaktadırlar. Uygulamarda bu türlü bir yöntem tercih edildiği takdirde saklama birimlerinin özelliklerine göre dizin yapısında değişiklik yapmak da genellikle mümkün olabilmektedir.

R-ağaçları, B-ağaçlarından aldığı kalıtımsal özellikler sebebiyle dengeli ve hiyerarşik yapıdadır. Ağaçtaki her bir düğüm bir disk sayfası yönlendirir. B-ağaçlarında olduğu gibi, R-ağaçlarının da yapılan çalışmalar ve gelişen teknolojiler sonucunda, yeni gereksinimleri karşılamak ve daha iyiye ulaşmak amacıyla geliştirilmiş, değiştirilmiş, yeni özellikler kazanmış tipleri vardır. İmge işleme, video işleme ve yine konumsal veri yönetimi gibi birçok alanda geçerli örnekleri olan R-ağacı tiplerinden belirgin birkaç tanesi, R*-ağacı, R+-ağacı, STR-ağacı olarak verilebilir.

3. 4'lü Ağaçlar

4'lü ağaç kavramı, veri uzayını öz yinelemeli olarak dörderli parçalara bölümleyerek, hiyerarşik bir yapı kurma ilkesine dayanan veri yapılarına verilen genel isimdir. Konumsal verileri hedefleyen, en eski yöntemlerden biridir. Ortaya çıktığı ilk günden bu yana yapılan çalışmalar sonucu birçok türü öne sürülmüştür. 4'lü ağaç denince genelde 2-boyutta düşünülse de, daha yüksek boyutlu uzaylara uygun yapılar da mevcuttur(örneğin 3-boyut için 8'li ağaç(octree)). Türleri arasında ayrım yapabilmek için genellikle şunlara bakılır [34] :

- Barındırılan veri türü
- Bölümleme yöntemi
- Çözünürlük

4'lü ağaçlar başta noktasal, dikdörtgensel, bölgesel, eğrisel, yüzeysel ve hacimsel veriler olmak üzere birçok veri tipine uygun olarak kullanılmaktadır. Bölümleme yöntemi ise veri uzayının bölümlenme şeklini tanımlar. Bölümler karesel, dikdörtgensel veya herhangi bir başka düzene göre yapılabilir. Bu bölümlerin ebatları belirlenirken belirli bir takım kısıtlar öne sürülebilir. Her bir karenin kenar uzunluklarının 2'nin herhangi bir kuvveti olması [43]'te önerilmiştir ve Q-tree olarak adlandırılmıştır. Çözünürlük ise bölümleme sürecinin ne kadar devam ettirileceği ile ilgilidir. Bir başka deyişle ne kadar çok bölümün oluşacağını, veri uzayının ne kadar hücreye bölüneceğini belirler, bir sabite bağlı olabilir veya girdiye göre değişebilir. Benzer ağaç yapılarının *4'lü ağaç (quadtree)* olarak ilk ele alınışı [23] ile görülmüştür. [55]'te bölümleme açısından benzerlik gösteren ve yine 4'lü ağaç olarak adlandırılan, ikili arama ağaçlarının 2 boyutlu ortama uyarlanması olarak düşünülebilecek bir yöntemle çok boyutlu noktasal veriler hedeflenmiştir.

Noktasal verilerden farklı olarak, dikdörtgensel veriler de ilgi alanına girmiştir. İmge işleme ve coğrafi sistemler gibi alanlarda nesneleri sınırlayan,

çevreleyen varlıklar, dikdörtgenler olarak kabul edilebilir. Ayrıca yüksek ölçekli birleştirim (VLSI) teknolojilerinde de devre elemanları yerleşim planları kontrol edileceğinde her bir elemanın yerleşim alanı bir dikdörtgen olarak görülmektedir. Çok fazla sayıda devre elemanı söz konusu olduğunda konumsal bir durum ortaya çıkmaktadır. Devre kartı üzerine yayılan devre elemanları, veri uzayına yayılan dikdörtgenler olmaktadırlar.

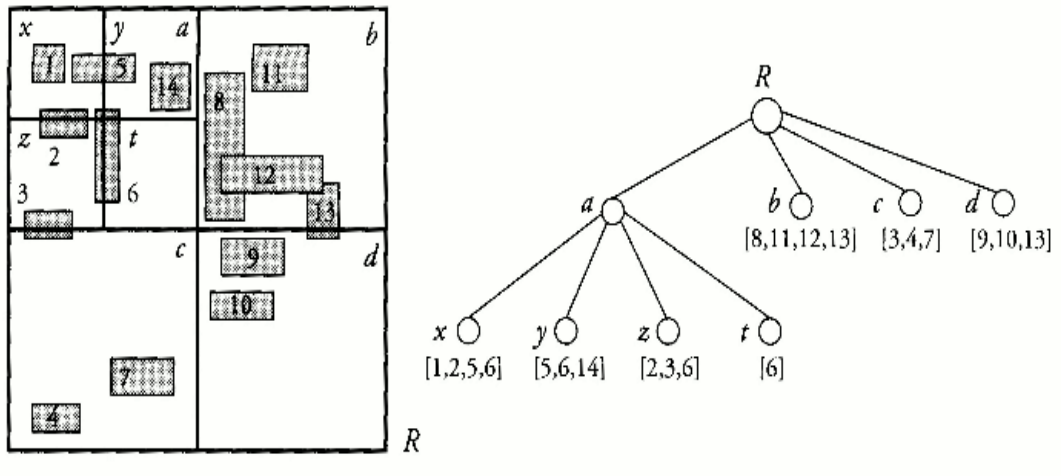
3.1 Dikdörtgenler ve 4'lü Ağaçlar

Dizinlenecek olan dikdörtgenler, uzayın öz yinelemeli olarak karelere bölümlenmesi sonucu oluşan hücrelerle ilişkilendirilirler. Bu bölümlemeye 4'lü ağaç bölümlemesi denir. Bu hücreler ve ilişkilendirilen dikdörtgenler, hücre mertebesi *anahtar* değer olarak kullanılarak bir B+-ağacı ile dizinlenirler.

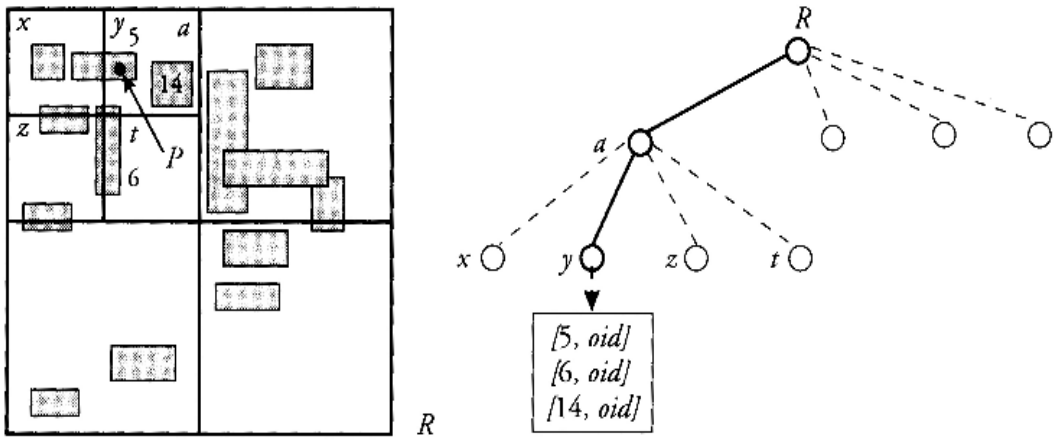
4'lü ağaçlar, diskte tutulan dikdörtgenlerin (nesnelerin sınırlayıcı kutuları) bir kümesini dizinlemek amacıyla, hücreler tarafından örtülen dikdörtgenlerin sayısı disk sayfası kapasitesinden büyük olduğu sürece, arama uzayını özyinelemeli olarak kare hücrelere bölümlemeye devam eder. Bu yüzden 4'lü ağaçların dengeli olması her zaman beklenemez. Bölgesel yoğunluklar gösteren, düzgün dağılıma sahip olmayan veri kümeleri söz konusu olduğunda ağaç dengesi bozulmakta ve arama performansı düşmektedir. Düzgün dağılmış verilerde ise bölümle süreci daha rahat ilerlemekte ve ağaç daha dengeli olmaktadır. Hücrelere Kuzey-Batı (NW), Kuzey-Doğu (NE), Güney-Batı (SW) ve Güney-Doğu (SE) adları verilmektedir. Düğümlerin her biri, dört çocuğa sahiptir. Her bir yaprak dizin girdilerini tutan bir disk sayfası ile ilişkilendirilmektedir. Şekil 3.1'de bir bölümleme ve onunla ilişkili olan ağaç görülmektedir. Yapraklar, karşı gelen dikdörtgenlerle etiketlenmiştir. Bu örnekte disk sayfası kapasitesinin 4 olduğu kabul edilmiştir.

4'lü ağaçlar ile nokta sorgulama işlemi çok basittir. Ağaçın kökünden yaprağa kadar inen tek bir yol izlenir. Her bir seviyedeki dört hücreden, aranan noktayı içeren bir tanesi seçilir. Yaprak okunur ve taranır. Şekil 3.2'de sonuç olarak 5 dikdörtgenini döndüren bir nokta sorgulama işlemi görülmektedir. Pencere

sorgulaları için ise yapraktaki tüm hücreler, sorgu penceresi ile kesişme durumuna göre kontrol edilir, kesişim var ise sonuca eklenir.



Şekil 3.1 : 4'lü ağaç



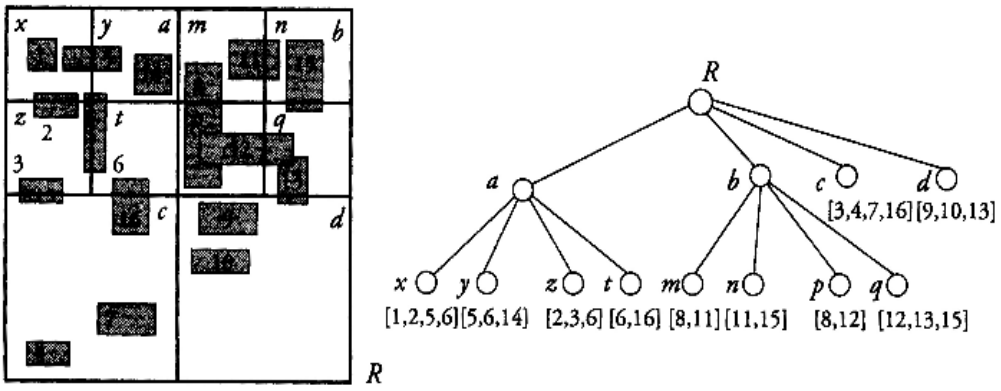
Şekil 3.2 : 4'lü ağaçta nokta sorgulama

3.1.1 Dikdörtgen Ekleme

Bir dikdörtgen kendisini kısmen örten her bir yaprak içine ilave edilir. Dikdörtgeni ilave etmek amacıyla dikdörtgeni örten yaprakların yolu takip edilir. Her bir yaprak ile ilişkilendirilen p sayfası okunur. Sonra iki durum olabilir :

- p dolu değildir (yeni bir girdi yapılabilir)
- sayfa doludur

Hücre doluysa dört yeni alt hücreye bölünür. Yeni üç adet sayfa için bellekte yer açılır. Eski sayfanın girdileri, kendisi ve diğer üç yeni (toplam dört adet) sayfa arasında dağıtılır. Bir e girdisi, hücresi e.mbr ile kesişen her sayfaya ilave edilir. Şekil 3.3'te 15 ve 16 numaralı dikdörtgenlerin ilave edilmesiyle oluşan dizin yapısı görülmektedir. 15 'in ilave edilmesi b'yi m,n,p ve q karelerine ayırmaktadır. 15 numaralı dikdörtgen n ve q yapraklarına ileve edilir. 16 nın ilave edilmesi herhangi bir bölünmeye yol açmaz. 16, c ve t yapraklarına ilave edilir.



Şekil 3.3 : 4'lü Ağaçta Dikdörtgen Ekleme

Mevcut 4'lü ağacın bu yapısı, bir konumsal erişim yönteminin tüm gereksinimlerini sağlamaz. Bunun en başta gelen sebebi düğümlerdeki çocuk sayısının 4, yani çıkış yelpazesinin (fan-out) dar olmasıdır. Bu sayı disk sayfası açısından yeterli değildir, bir disk sayfası bundan çok daha fazlasını tutabilir. Bu yapıyı disk sayfalarına yönlendirmek zordur. B-ağacı, R-ağacı gibi yüksek çıkış yelpazelerine imkan tanıyan yapılar, disk saylarıyla daha uyumludur.

4'lü ağacın sorgu performansı, ağacın derinliğine bağlıdır. Çıkış yelpazesi dar olduğundan derinlik fazla olmaktadır. Her düğümün farklı bir sayfayı gösterdiği en kötü durum düşünülürse disk erişim sayıları derinlik değerine yaklaşacaktır. Durağan verilerin söz konusu olduğu bir durumda düğümlerin paketlenmesi

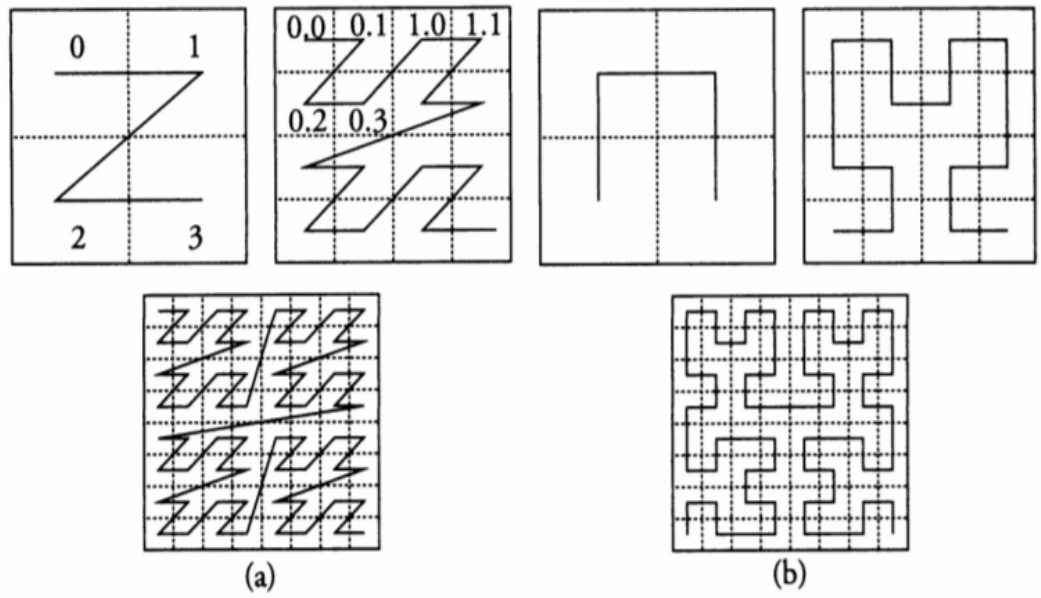
düşünülse de devingen durumlarda performans düşüklüğü olacaktır. Ayrıca ızgara dosyasında da olduğu gibi, 4'lü ağaçlar da tekrarlı nesneler sebebiyle zorluk yaşamaktadır.

3.1.2 Uzayı Dolduran Eğriler

Uzayı dolduran eğriler iki boyutlu bir ızgaranın hücreleri üzerinde bir tam-sıralama tanımlar. Bu sıralama kısmen yakınlık sağladığı için faydalıdır; yani, uzayda iki hücre yakınsa bunlar tam-sıralama içinde de birbirilerine yakındırlar. Bu durum eğri tipine göre değişebilmektedir ve bazı eğriler için sağlanmayabilir.

Eğrilerin dolduracağı uzay, ızgara yapısı ile ifade edilecek olursa, $N = 2^d$ olmak üzere N x N boyutlarında tam bir 4'lü ağaç yapısı düşünülebilir. Buna verebileceğimiz ilk örnek *z-sıra* veya *z-sıralaması* olarak bilinen yoldur.

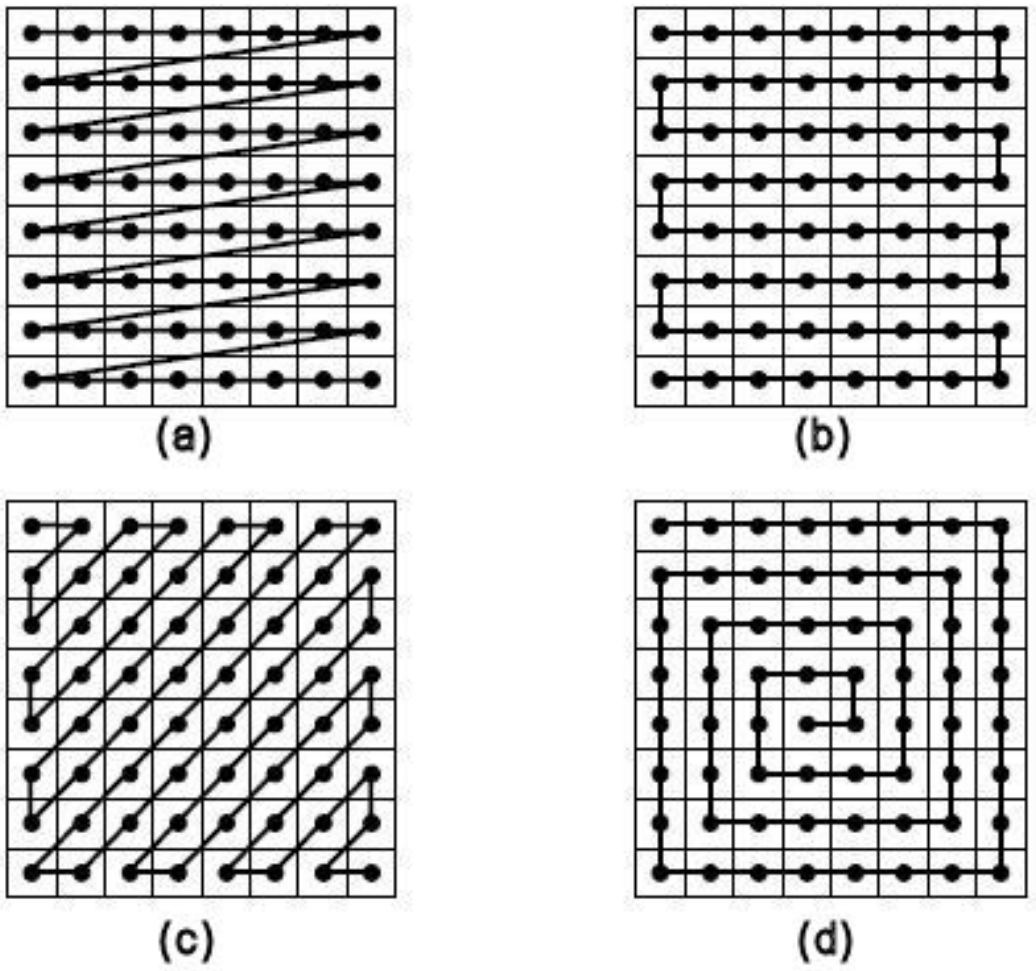
Ağacın her bir düğümü bir etiket ile ilişkilendirilir. Kökün etiketi boş bir katardır(string). Etiket değeri k olan bir iç düğümün NW, NE, SW, SE altları sırasıyla k.0, k.1, k.2 ve k.3 etiketlerine sahiptir. Buradaki "." işareti katarlar arasındaki ucuca ekleme işlemini göstermektedir. Bu haliyle hücreler boyu d olan katarlar ile etiketlenirler. Hücreler etiketlerine bağlı olarak (sözlük sıralamasına göre) sıralanabilir. Örneğin d=3 seçildiğinde "212" etiketli hücre "300" etiketli hücreden önce ve "21" etiketli hücreden sonra gelmektedir. NW, NE, SW, SE sıralaması z harfine benzediği için z-sıralaması adı verilmiştir(Şekil 3.4). Bu sıralamaya aynı zamanda *Morton* sıralaması da denir [26]. SW,NW,SE,NE şeklinde bir yol izlenirse de buna N-sıralama denir.



Şekil 3.4 : Uzayı Dolduran Eğriler (a) z-sıralama (Morton order) (b) Hilbert eğrisi (Peano order)

Hilbert eğirisi, iki boyutlu bir ızgarada her bir hücreye ulaşan bir başka eğri türüdür. Şekil olarak **Z** değil de $\prod$ şeklindedir. z-sıralamasının tersine Hilbert eğrisi sabit uzunlukta doğru parçalarından oluşur. Yani hiçbir zaman uzayda dolaşırken bir konumdan başka bir konuma ani ve aşırı bir sıçrama yapılmaz. Bu eğrilerin önemli bir dezavantajı, uzayda yakın bir konumda olan nesnelerin, eğri üzerinden yapılan takipte daha uzak görünebilmesidir. Örneğin z-sıralamasında NW ve SW hücrelerinde olan bir nesneye eğri üzerinden bir yolla ulaşılmak istendiğinde NE hücresi araya girmektedir. Benzer bir durum Hilbert eğrisi kullanıldığında da görülmektedir. SW ve SE bölümleri yakın olsa bile sıra SW,NW,NE,SE şeklinde gittiğinden dolayı araları normalden daha açıkmış gibi bir görüntü oluşmaktadır.

Literatürden belli başlı verilebilecek örneklere baktığımızda z-sıralaması ve Hilbert eğrilerine ek olarak verebileceğimiz temel örnekler şunlardır [32] :



Şekil 3.5 : (a) Satır Sıralama (b) Devreden öncelikli satır sıralama (c) Cantor sıralama (d) Sarmal sıralama

Bütün bu sıralama yöntemlerinde temel amaç uzayda dolaşırken her hücreye uğramak ve bunu yaparken de rastgele, belirsiz durumda kalmamak için bir düzen takip etmektir. Bu eğrilerde bulunması gereken temel, ortak özellikler şunlardır :

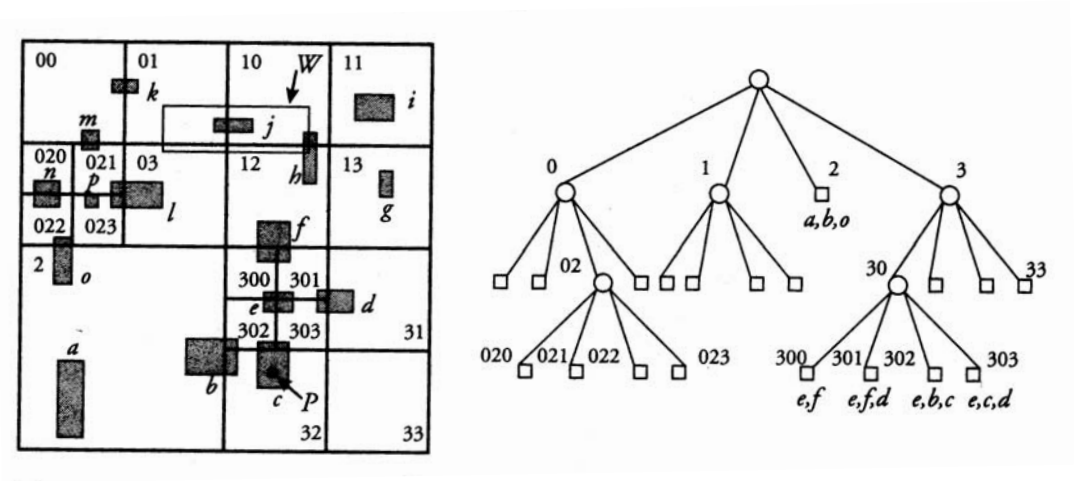
- Eğri her hücreye, sadece ve sadece 1 kez uğramalı
- Uzayda komşu olan herhangi 2 hücre, eğri üzerinde de komşu olmalı
- Herhangi bir hücre için komşulara erişim imkanı olmalı
- Uzayın genişlemesine uyum sağlamalı
- 2-boyutlu uzaydan eğriye ve bunun tersine geçişlere imkan sağlamalı

3.1.3 4'lü Ağaç Etiketleme

Derinliği d olan bir 4'lü ağaç ele alalım. Bunu kenar uzunluğu, $N=2^d$ olan bir ızgaraya dönüştürelim. Her bir yaprak hücre, uzunluğu d olan bir katar ile etiketlenebilir. Şekil 3.6'da böyle bir 4'lü ağaç gösterilmektedir. Açıkça görüldüğü gibi;

- yaprakların sırası, yaprakların soldan sağa doğru taranmasına karşı gelmektedir.
- etiketler aynı ebatta değildir.

Etiketlerin ebadı, yaprağın ağaçtaki derinliğine bağlıdır. Yaprağın etiketi kökten yaprağa giden yolun etiketi gibi düşünülebilir.

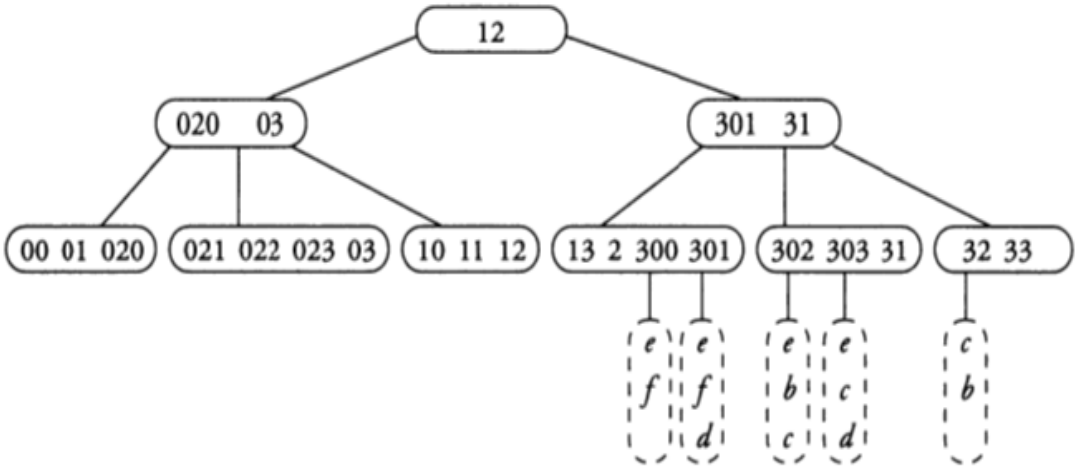


Şekil 3.6 : z-sıralama ile 4'lü ağaç etiketleme

Eğer bir L yaprağı bir C hücresini içeriyorsa, bu durumda L'nin l etiketi, C hücresinin c etiketinin önekidir ve l<c dir. Örneğin, 3<31<312 gibi.

3.1.4 Doğrusal 4'lü Ağaç

Anahtar düşünce ilk olarak [oid, mbb] girdilerinin, yani dizin kayıtlarının etiketler aracılığı ile 4'lü ağacın yapraklarına aktarılması ve p ile işaret edilen bir disk sayfasında tutulmasıdır. Bu düşünce başarılı bir şekilde uygulanırsa (l,p) ikililerinden oluşan bir küme için, l etiketleri anahtar değeri olarak kullanılarak, B+-ağacı temelli bir dizin yapısı kurulabilir (Şekil 7).



Şekil 3.7 : B+-ağacı ile dizinlenmiş 4'lü ağaç yaprakları

Böyle bir etiketleme yöntemi sayesinde 4'lü ağaç yapısı tekil anahtarlar ile B+-ağacına uyumlu bir şekilde aktarılmaktadır. Bu aktarım devingendir, yani ekleme ve silme işlemlerine izin verir. Örnekte görüldüğü üzere 13, 2, 300 ve 301 etiketlerine karşı gelen yapraklar aynı disk sayfasındadır. Bu düzen içerisinde şöyle bir problem ortaya çıkmaktadır: Birden fazla 4'lü ağaç yaprağında birden görünen veriler, yani mbb'ler sayfalar içinde tekrar edilebilmektedir. Elimizdeki örnekte (Şekil 3.7), e kaydı 2 farklı sayfada, 4 kez tekrar etmektedir.

3.1.5 Doğrusal 4'lü Ağaç ile Nokta Sorgulama

Bir P noktasını içeren nesne tanımlayıcılarını geri alma işlemi üç temel adım gerektirir.

- *Adım 1 , etiket hesaplama :* P noktasını içeren ızgara hücresinin uzunluğu d olan l etiketi hesaplanır.

- *Adım 2 , B+-ağacındaki 4'lü ağaç yaprağını geri getirme :* P noktasını içeren 4'lü ağaç yaprağının etiketi L olsun. Bu durumda yaprağın derinliği d ise L=l olur. Uzunluğu d'den az ise ve yaprak P noktasını içeren ızgara hücresini içeriyorsa L değeri, l nin bir önekidir. Daha somut olarak söylemek gerekirse etiketi L olan hücre, B+-ağacında anahtar değeri l'den küçük veya eşit olan en büyük kayıttır. Bunu B+-ağacının kendisi ile sağlayamayız. Bu amaca yönelik fonksiyon MAXINF(l) ' dir.

- *Adım 3- Yaprak erişmi ve taraması:* Önce [L,p] girdisine sahip B+-ağacı bulunur, bu halde adresi p olan sayfaya erişilmelidir. Bu sayfada [oid, mbb] şeklinde olan tüm çiftler taranır ve mbb'nin P noktasını içerip içermediğine bakılır. Sonuç, nesne tanımlayıcılarının listesidir.

Bu algoritma Şekil 3.6 da gösterilmiştir. P noktasının koordinatlarından önce P noktasını içeren ızgara hücresi bulunur. Bunun etiketi "320"dir. Sonra MAXINF yöntemi kullanılarak B+-ağacından en büyük yaprak etiketi, "320" den küçük olan bilgi alınır. Örneğimizde bu "32" etiketine sahip yapraktır. Geriye kalan, sayfaya erişmek ve sayfa içindeki her nesnenin sınırlayıcı kutusunun P noktasını içerip içermediğini kontrol etmektir. Bu kontrol ile c nesnesi sonuç kümesine eklenir.

Nokta sorgulama algoritması şöyledir :

```
LQ-POINTQUERY (P:point)
begin
        result = ∅
        //adım 1: noktanın etiketini hesapla
        l=POINTLABEL(P)
        //adım 2: B+-ağacı l etiketi ile gezilerek [L,p] kaydına ulaşılır.
        [L,p]=MAXINF(l)
        //adım 3: Sayfayı oku ve nesneleri getir
        page = READPAGE(p)
        for each e in page do
                if (e.mbb contains P) then result += {e.oid}
        end for
        return result
end
```

Algoritma 3.1 : Doğrusal 4'lü Ağaçta Nokta Sorgulama

3.1.6 Doğrusal 4'lü Ağaç ile Pencere Sorgulama

Pencere sorgusunda ana fikir, W penceresinin örttüğü tüm karelerin etiketlerinin A aralığını hesaplamaktır. Sonra A ile B+-ağacı üzerinde bir aralık (range) sorgulaması yapılabilir. Bu üç adımda yapılabilir.

- *Adım 1:* Nokta sorgulama algoritmasında olduğu gibi pencerenin NW köşesinin l etiketi hesaplanır. Sonara MAXINF(l) bulunur. Bu aralığın alt sınırı L olan [L,p] 'yi verir. Daha sonra SE penceresini köşesinin l' etiketi hesaplanır ve sonra MAXINF(l') bulunur. Bu bize L' üst sınır olmak üzere [L',p'] bilgisini verir.

- *Adım 2:* B+ağacı üzerinde [L,L'] aralığı sorgulanır. Etiketi l olan ve bu aralığa düşen tüm [l,p] girdilerini içerir.

- *Adım 3:* Girdisi e=[l,p] olan her bir B+-ağacı kaydı için QUADRANT(e.l) çağırımı ile etiketi e.l olan hücre bulunur. Eğer QUADRANT(e.l) pencereyi örtüyorsa e.p sayfasına erişilir ve W ile örtülen 4'lü ağaç kayıtları,

[oid,mbb], test edilir. Eğer W penceresi ile kesişen kayıt varsa sonuç kümesine eklenir.

```
LQ-WINDOWQUERY (W:rectangle)
begin
        result = ∅
        //adım 1: W penceresinin köşelerinden [L,L'] aralığını hesapla.
        //B+-ağacında arama için gerekli
        l=POINTLABEL(W.ne); [L,p] = MAXINF(l)
        l'=POINTLABEL(W.ne); [L',p'] = MAXINF(l')
        //adım 2: B+-ağacında, l ∈ [L,L'] olan [l,p] kayıtlarından oluşan Q
        //kümesini oluştur. Yani bu aralıktaki kayıtları al
        Q = RANGEQUERY([L,L'])
        //adım 3: Q kümesinde olan ve W ile örtüşen kayıtlar için sayfaya eriş
        for each q in Q do
                if (QUADRANT(q.l) overlaps W) then
                        page = PAGEREAD(q.p)
                        //sayfayı tara
                        for each e in page do
                                if (e.mbb overlaps W) then
                                        result += {e.oid}
                        end for
                endif
        end for
// Sonuçları sırala ve tekrarlı kayıtları yok et
        SORT(result); REMOVEDUPL(result)
        return result
end
```

Algoritma 3.2 : Doğrusal 4'lü Ağaçta Pencere Sorgulama

Şekil 3.6 incelenecek olursa, W sorgu penceresinin NW köşesinin etiketi "012", SE köşesinin etiketi ise "121" olacaktır. Bu etiketler ile B+-ağacı üzerinde arama yapılacak olursa 012'den küçük en büyük ve 121 arası ziyaret edilecektir. Bu da [01,12] aralığı demektir. Bu aralıktan alınan sayfalarda bulunan [oid,mbb] kayıtları, W penceresi ile karşılaştırıldığında 013, 102, 103 ve 121 etiketleri ile gösterilen hücrelerden "j" ve "h" dikdörtgenleri sonuç kümesine girmektedirler.

Bu yöntemle oluşan bir dizin yapısında nokta sorgulamanın 2 temel bölümü vardır. Erişim sayıları açısından önemlidir.

i) B+-ağacında yaprak erişimi
ii) 4'lü ağaçta sayfa erişimi

B+-ağaçları için derinlik d olduğunda I/O sayısı d+1 olmaktadır ve B+-ağacının yer ve zaman karmaşıklığı eleman sayısına bağlıdır. En kötü durumda bile yeterince etkin performansı nedeniyle konumsal erişim yöntemlerinin tasarımında da etkili olmaktadırlar.

Pencere sorgusunda I/O işlem sayıları çok daha fazla ve zor olmaktadır. Alt ve üst sınırları bulmak için B+-ağacı 2 kere gezilir. Daha sonra alt sınıra ait yapraktan üst sınıra kadar gerekli olan bütün yapraklar taranır. Bu aşama aralığın büyüklüğüne bağlıdır. Bu aşamadaki performansı iyileştirmek, W penceresi ile örtüşmeyen hücreleri elemek adına birçok çalışma yapılmıştır. Bu, tasarım ilkesinde ve veri yapılarında değişiklikler gerektirmektedir ve farklı yöntemlerin doğmasına ön ayak olmuştur.

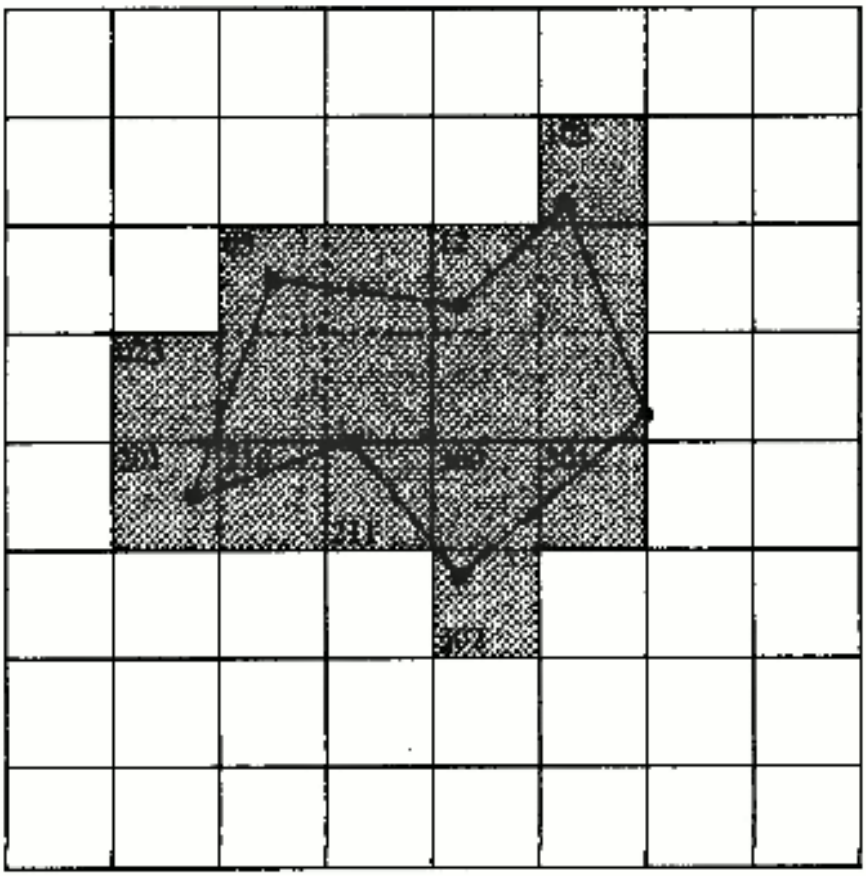
3.1.7 Z-sıralama Ağacı

Nesnelerin sınırlayıcı kutuları gibi yaklaşımlar kullanmak yerine, her nesnenin geometrik yapısını parçalara ayırarak derinliği d olan bir 4'lü ağaca aktarmak ve ağacın yarpakları aracılığı ile nesneye yaklaşımda bulunmak z-sıralama ağacının temel mantığını oluşturur. Uzunlukları d'den küçük veya eşit olan yaprak etiketleri, B+-ağacına yerleştirilir. Bu yapıya *z-sıralama ağacı* denir.

Bu yapı oluşturulurken izlenen adımlar dizisi şöyledir. Bir o geometrik nesnesi ve q hücresi verildiğinde, q hücresinden büyüyen bir 4'lü ağacın tüm yapraklarının, o nesnesi ile örtüşme durumuna bakılır. Eğer böyle bir durum var ise q, sonuç kümesine eklenir. Aksi takdirde o nesnesi, q'nun alt hücreleri ile örtüşen bölümlerine göre 4'ten az veya eşit sayıda parçaya ayrılır. Bu süreç özyinelemeli olarak, d derinliğine ulaşılıncaya kadar her parçaya uygulanarak

devam eder. Sonlanma anında her bir çeyrek hücre (quadrant), *asgari(minimal)* olarak kabul edilir.

Sonuç kümesi, 4^d adet hücreden oluşan sabit bir ızgara üzerinde yapılan bir taramada nesnenin yaklaşımı olarak da görülebilir.



Şekil 3.8 : z-sıralaması ve nesne ayrıştırması

Şekil 3.8'de nesne ayrıştırması ve yaklaşımı örneklenmiştir. Nesnenin yaklaşımı {023,03,103,12,201,210,211,300,301,302} etiketli hücreler ile sağlanmaktadır. Bu hücrelerin 03 ve 12 dışında kalan her biri ağaçta 3. seviyede bulunurlar. 03 ve 12 ise 2. seviyededirler. + işleci küme birleşimini, U ise geometrik birleşimi göstermek üzere, nesne ayrıştırma algoritması aşağıdaki gibidir.

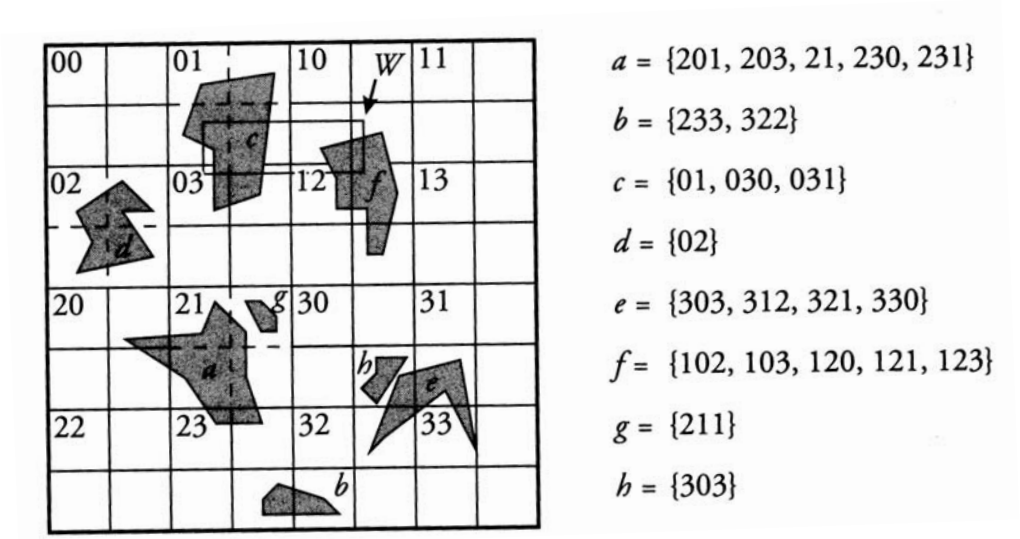
```
DECOMPOSE (o:geometry, q:quadrant):
begin
        decomp_NW, decomp_NE, decomp_SW, decomp_SE :set(quadrant)
        result = ∅
        // q ile o'nun örtüşmesini kontol et, örtüşme yok ise boş geç
        if (q is overlaps o) then
                if (q is minimal) then
                        result = {q}
                else
                // o'yu her bir alt hücre için ayrıştır
                        for each sq in {NW(q),NE(q),SW(q),SE(q)} do
                                decomp_sq = DECOMPOSE(o,sq)
                        end for
                        // eğer ayrıştırma dolu bir alt hücre ile sonuçlanırsa q'yu
                        //döndür
                        if (decomp_NW ∪ decomp_NE ∪ decomp_SW ∪ decomp_SE = q) then
                                result = {q}
                        else
                                // dört ayrı parçanın birleşimini al
                                result = decomp_NW+decomp_NE+decomp_SW+decomp_SE
                        end if
                end if
        end if
        return result
end
```

Algoritma 3.3 : Nesne Ayrıştırma

8 farklı geometrik nesneden oluşan örnek bir veri kümesinin dağılımı ve z-sıralamasına göre ayrıştırılması Şekil 3.9'da verilmiştir. 4'lü ağacın derinliği 3'tür. l hücre etiketi ve oid nesne tanımlayıcısı olmak üzere [l,oid] ikilileri ile nesne yaklaşımının örtüştüğü hücreler bulunmaya çalışılır. Örneğimizdeki a nesnesi için bu ikililer şöyledir :

{[201,a],[203,a],[21,a],[230,a],[231,a]}

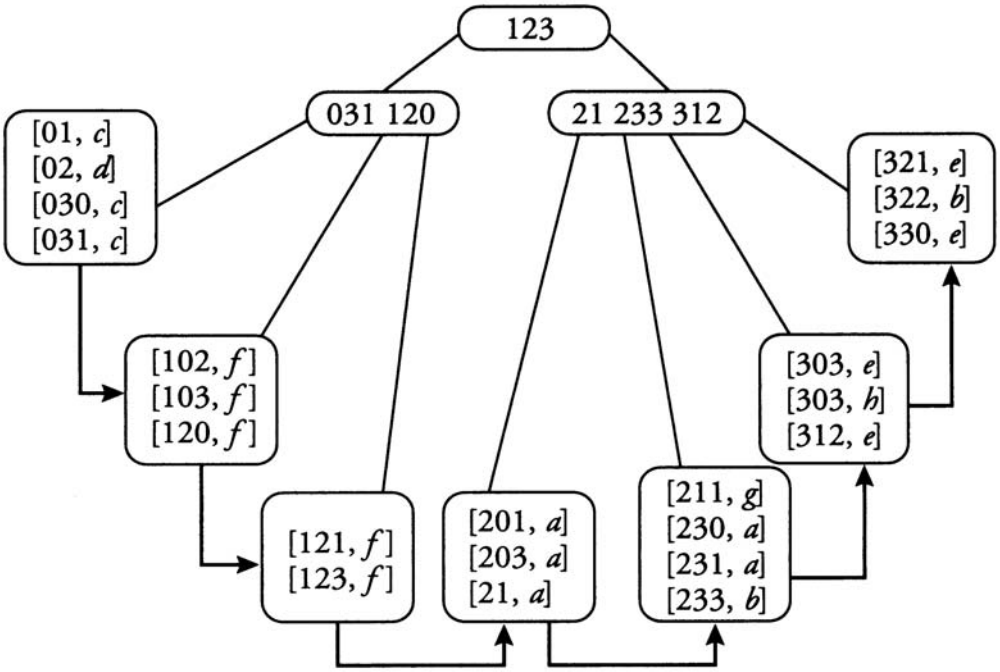
Açıkca görülmektedir ki, burada da doğrusal 4'lü ağaçta olduğu gibi nesne tekrarları söz konusudur. Bir nesnenin yaklaşımı elde edilmeye çalışılırken, aynı nesne tanımlayıcısı birçok kez tekrarlanmıştır.



Şekil 3.9 : z-sıralamasına göre parçalanmış nesnelerin kümesi

Bu durumum tam tersi olarak, aynı l etiketinin farklı oid nesne tanımlayıcıları ile görülmesi mümkündür. Aynı seviyedeki farklı nesnelerin aynı hücre denk gelmesi, yaklaşımlarının da aynı hücrelerde buluşmasına yol açabilmektedir. Bunu 303 etiketini paylaşan e ve h nesneleri ile örnekleyebiliriz. a ve g nesnelerini ele aldığımızda ise aynı hücreyle örtüşmelerine rağmen farklı etiketler ile yaklaşımlar yapıldığını görürüz.

Oluşturlan bu etiket ve tanımlayıcı ikilileri, B+-ağacında, etiketler anahtar değeri olacak şekilde dizinlenir. Şekil 3.10'da örneği görülen bu yapıda, nesne tanımlayıcıları (oid, örneğin a,b,c) doğrusal 4'lü ağaçta olduğu gibi yaprak düzeyindedir. Burada dikkat edilmesi gerek nokta, z-sıralamasından kaynaklanan ani sıçramalardır. Uzayda yakın olan 2 nokta, z-sıralamasında uzak düşebilmektedir. Örneğin b nesnesi 2 hücreye ayrıştırılmıştır fakat bunlar B+-ağacında 2 ayrı yaprakta tutulmaktadır.



Şekil 3.10 : Z-sıralama Ağacı

```
ZO-WINDOWQUERY(W:rectangle)
begin
        result = ∅
        //Adım 1: W penceresinin köşelerinden [L,L'] aralığını hesapla.
        //B+-ağacında arama için gerekli
        l = POINTLABEL(W.nw); [L,p]=MAXINF(l)
        l' = POINTLABEL(W.se); [L',p']=MAXINF(l')
        // Adım 2: l∈[L,L'] olmak üzere [l,oid] kayıtlarının kümesini, E'yi oluştur
        E=RANGEQUERY([L,L'])
        // Adım 3: E kümesinin W ile örtüşen her elemanını sonuca ekle
        for each e in E do
                if (QUADRANT(e.l) overlaps W) then result+={e.oid}
        end for
        // Sonuçları sırala ve tekrarlı kayıtları yok et
        SORT(result); REMOVEDUPL(result);
        return result
end
```

Algoritma 3.4 : Z-sıralama Ağacında Pencere Sorgulama

Nokta ve pencere sorgulama algoritmaları, doğrusal 4'lü ağaçtakine oldukça benzerdir. Z-sıralama ağacında pencere sorgulama algoritması yukarıda verilmiştir. Şekil 3.9'a dönecek olursak W penceresi ile yapılan bir sorguda pencerenin belirlediği alt sınır 012, üst sınır ise 121 etiketlerine sahip hücreler olacaktır. MAXINF fonksiyonu sırasıyla [01,c] ve [121,f] kayıtlarını verir ve B+-ağacı üzerinde bu aralıkta bir tarama yapılır. [02,d] gibi kayıtlar taranmasına rağmen sorguyla ilgili değildir.

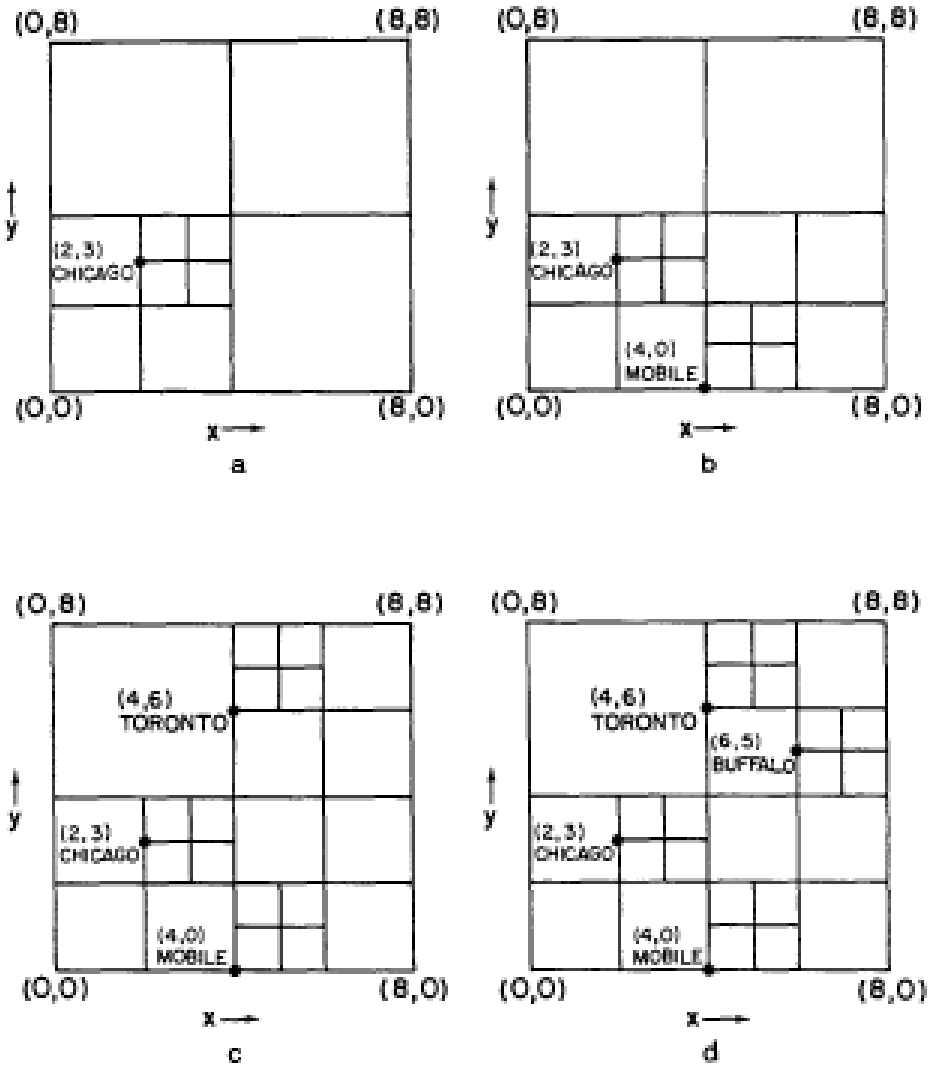
Pencere sorgusunda performansın pencere ebatlarına bağlına olduğunu daha önceki bölümde (3.1.6) belirtmiştik. Doğrusal 4'lü ağaç ile z-sıralama ağacını bu açıdan değerlendirecek olursak, z-sıralama ağacında derinliğin daha fazla olduğu görülür. Bu da I/O sayısında artış demektir. Çünkü nesneler ayrıştırılırken ne kadar çok kayıt oluşursa ağacın yükü o kadar artmaktadır. Buna çözüm olarak, kullanılan nesne yaklaşımları gözden geçirilmelidir.

Doğrusal 4'lü ağaçta, B+-ağacı, z-sıralama ağacındakinden çok daha az kayıt bulundurur. Çünkü kayıt sayısı 4'lü ağacın hücre sayısına eşittir.

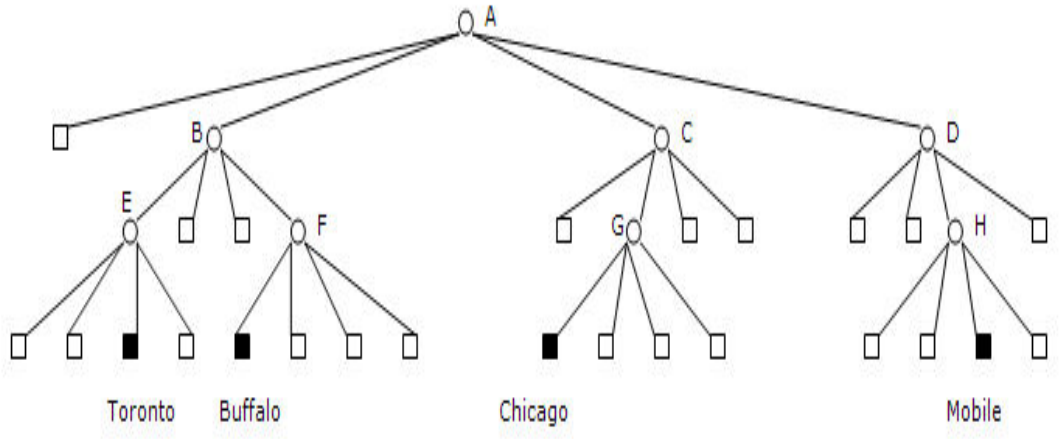
3.2 MX 4'lü Ağacı

4'lü ağaçların özyinelemeli olarak ayrıştırmayı sağlayan hiyerarşik veri yapıları olduğunu daha önce belirtmiştik. Bölme noktaları olarak verilerin bizzat kendilerinin kullanıldığı, yani noktasal verilerin dizinlenmesi gereken bir durumda da 4'lü ağaçlar kullanılabilir.

Noktasal verileri 1x1 ebatlarında dikdörtgenler olarak düşünecek olursak bu yapıya *MX (matrix) 4'lü ağacı* denir. Bu veri yapısında ilk olarak anlaşılması gereken bazı kavramlar ve kabuller vardır. Yapraklar *siyah* veya *beyaz* olarak işaretlenir. Siyah, matris üzerinde veri bulunan bir nokta, beyaz ise boş bir nokta anlamındadır. Her nokta 1x1 ebatlarında bir karedir. Şekil 3.11'de 2^3x2^3 ebatlarında bir MX 4'lü ağacının temsil ettiği yapı görülmektedir. Bu yapıya karşılık gelen ağaç ise şekil 3.12'de verilmiştir.



Şekil 3.11 : MX 4'lü Ağacının oluşumu (veri ekleme)
Chicago (b) Mobile (c) Toronto (d) Buffalo



Şekil 3.12 : MX 4'lü Ağacı

Veriler, hücrelerin sol alt köşeleri ile ilişkilendirilir. Hücrelerin sol alt köşesi kapalı kabul edilir. Yeni oluşumlar sağ üst köşeye doğru gerçekleşecektir. Matrisin sol alt köşesi (0,0) olarak kabul edilir. Ağaçtaki her düğüm 6 alan içerir. İlk dört tanesi, 4 çocuğu gösteren işaretçilerdir. P, bir düğüme işaretçi, I, 4 çocuktan biri ise bu alanlar SON(P,I) ile ifade edilir. Beşinci alan NODETYPE, düğümün tipini belirtir. BLACK (siyah) veri bulunduğunu, WHITE (beyaz) boş olduğunu, GRAY (gri) ise yaprak olmadığını belirtir. Altıncı alan NAME ise veri hakkında açıklayıcı, tanımlayıcı bilgi içerir (örneğin şehir ismi). Veri noktasının koordinatlarını tutmaya gerek yoktur. Bu bilgi ağaçta izlenen yol üzerinden zaten çıkarılabilir. Aynı noktaya birden fazla veri oturmaya başladığında ise taşma olacaktır. Bu sorun ise her düğüm için bir taşma listesi ile çözülebilir. Bu haliyle sabit ızgara benzeyen MX 4'lü ağacı, boş düğümlerin birleşimi ile kendi farkını yaratmaktadır.

Bu yapı içerisindeki en küçük alan 1x1'lik karedir. Boş bir MX 4'lü ağacı NIL olarak gösterilir. Bu verilen kavramlar çerçevesinde MX_COMPARE fonksiyonu, herhangi bir veri noktasının hangi alt bölüme ve bölünme noktasına yönlendirileceğini belirler. Belirtilen MX_COMPARE fonksiyonu, algoritmik olarak aşağıda verilmiştir.

```
MX_COMPARE(X:integer, Y:integer, W:integer)
// Merkezi (W,W)'de olan, genişliği 2*W olan MX 4'lü ağacını, (X,Y) noktasını
//içeren alt çeyreğini döndür.
begin
        return ( if ( X < W) then
                        if (Y < W) then
                                'SW'
                        else
                                'NW'
                else if (Y < W) then
                                'SE'
                else
                        'NE' );
end
```

Algoritma 3.5 : MX 4'lü Ağacı (Karşılaştırma – Yön Belirleme)

Veriler MX 4'lü ağacına sanki arama işlemi yapılıyormuş gibi eklenirler. Bu arama işlemi verinin martis üzerindeki yerine göre yapılır (örneğin veri uzayına dağılan ayrık x ve y koordinat değerleri). Bu arama işlemini MX_COMPARE fonksiyonu yönlendirir. Matris üzerinde hangi alt çeyreğe, yani 4'lü ağacın hangi alt dalına yönelilmesi gerektiğini bu MX_COMPARE fonksiyonu belirler. Eğer arama başarısız olursa yapraklarda son bulur. Ulaşılan yaprakta bir başka kayıt varsa gerekli bilgi güncellemesi yapılır. Eğer yaprak boşsa, NIL olarak görünüyorsa bu alanda yeniden özyinelemeli bölümleme yapılır. 1x1'ik ebatlara ulaşılınca bölümleme sona erer. Bu işlem yaprak olmayan, ara düğümler oluşmasına sebep olur ki bu da bölünmedir (splitting). $2^n x 2^n$ ebatlarında bir MX 4'lü ağacında böyle bir bölünme en fazla n kez gerçekleşebilir.

Verilerin eklenme sıraları, ekleme sürecinde oluşan ara ağaçlar üzerinde farklılıklar yaşanmasına sebep olabilse de sonuçta oluşacak MX 4'lü ağacının yapısını etkilemez. Şekil 3.11'de sırasıyla Chicago, Mobile, Toronto ve Buffalo şehirlerini temsil eden noktalar eklenmiştir. Ekleme işlemlerinin yapılması MX_INSERT fonksiyonu ile gerçekleşir.

```
MX_INSERT(P:node, X:integer, Y:integer, R: node, W:integer)
//(X,Y) noktasına denk gelen P düğümünü, genişliği W olan ve R'de kökü bulunan
//MX 4'lü ağacına ekler. Eğer ağaç boş ise, R yeni kök olur. Düğümler
//CREATE_PNODE ile yaratılır. GRAY (gri) yaprak olmayan düğümleri temsil eder.
begin
        T: node
        Q : quadrant
        if (W=1) then
                begin //Tek elemanı ağaç ise
                        R←P
                        return
                end
        else if null (R) then
                R←CREATE_PNODE('GRAY')//Kökü R'de olan ağaç boş ise
        T←R
        W ← W/2
        Q←MX_COMPARE(X,Y,W)
        while W > 1 do
                begin
```

```
                    if null (SON(T,Q)) then
                        SON(T,Q) ←CREATE_PNODE('GRAY')
                    T←SON(T,Q)
                    X←X mod W
                    Y←Y mod W
                    W←W/2
                    Q←MX_COMPARE(X,Y,W)
                end
        SON(T,Q) ← P
end
```

Algoritma 3.6 : MX 4'lü Ağacına Veri Ekleme

MX_INSERT fonksiyonunda ekleme işlemi sırasında yeni bir düğüm yaratma ihtiyacı doğduğunda kullanılacak olan CREAT_PNODE fonksiyonu ağaşıda verimiştir. Yeni düğüm yaratıldığında renginin "GRAY" olarak belirlenmesi o düğümün boş olduğu anlamına gelmektedir.

```
CREATE_PNODE(C:string)
//Rengi C ile belirtilen bir düğüm yarat ve işaretçisini döndür
begin

        P : node
        I : quadrant
        P←create(node)
        NODETYPE(P)←C
        for I in {'NW','NE','SW','SE'} do
                SON(P,I)←NIL
        return (P)

end
```

Algoritma 3.7 : MX 4'lü Ağacında Düğüm Yaratma

MX 4'lü ağacında pencere sorgusu önceki pencere sorgulamalarına benzer bir yolla yapılır. İlk olarak sorgulanan noktanın örtüştüğü bölüme doğru gidilir ve bu esnada alt ve üst sınırlar belirlenir. Aralık belirlendikten sonra, içerdiği tüm kayıtlar sorgu penceresi ile karşılaştırılır. En kötü durumda performans, F bulunan nokta sayısı ve n ağacın derinliği olmak üzere $O(F+2^n)$ ile üstten sınırlanır [32]. MX 4'lü ağacı, veri noktaları ayrık ve sonlu olduğu sürece yeterlidir. Bunun tersi durumlarda veriler arasında kesin bir ayrım yapılamayacağı için gereksinimler tam olarak karşılanamayacaktır.

3.3 Caltech Ara Formu ve MX-CIF 4'lü Ağacı

3.3.1 Caltech Ara Formu (Caltech Intermediate Form)

Kısaca *CIF* olarak anılan bu düzen içerisinde temel amaç tümleşik devrelerin içeriklerini grafiksel anlamda düzenlemek ve tanımlamaktır. Makine tarafından okunabilir gösterimlerle çizici, görüntü aktarım ve oynatma cihazları gibi diğer çıktı birimlerine yeniden yapılandırılabilir ve standart bir biçim sağlamayı hedefler.

80'li yıların başında üniversite çevrelerinde (California Institue of Technology) ortaya çıkan CIF sayesinde birçok araştırma araç ve gereci hakkında ortak veritabanı yapısı oluşturulur. Elektronik bir yonga üzerinde farklı katmanlarda yer alan ve 2-boyutlu yayılım gösteren devre elemanları, geometrik gösterimlerle düzenlenmeye çalışılır. CIF ile hiyerarşik bir yapı ve gösterimde tutarlılık, kesinlik sağlanır. Buna ek olarak okunabilirlik açısından da kolaylık sağlanır. CIF, yüksek ölçekli birleştirim uygulamaları için kesin ve güçlü tanımlayıcı özelliklere sahip geometrik gösterim aracı olarak da nitelenebilir [59]. Bu gösterimde en çok yer edinen temel geometrik kavram dikdörtgendir. CIF, temel devre elemanlarının dikdörtgenler ile ifade edildiği bir geometri dilidir [35]. Fakat bu dilin tam olarak nereye yerleştiğinin anlaşılması önemlidir. CIF dosyaları genellikle bilgisayar programları aracılığıyla sembolik yerleşimlerden ve etkileşimli tasarım programlarından diğer farklı gösterimlere çevrim yapılırken arada bir yerde durur.

CIF ile oluşturlan ortak zemin üzerinde birleşen tanımlamalar ile birden fazla projenin bir bütün olarak görülmesi sağlanabilir. Kodlama ve tasarım yöntemleri ne olursa olsun, ortak bir ara düzende birleşen çalışmalar, daha sonra birçok çıktı cihazı veya tasarım uygulumaları için çeşitli biçimlere dönüştürülebilir.

Bir CIF dosyası, tanımlanmış, sınırlı bir karakter kümesinden elde edilebilen karakter dizilerinden oluşan satırlar bütünüdür [54]. Bu dosya bir takım komutlar barındırır, ve komutar noktalı virgül (;) ile birbirlerinden ayrılırlar.

CIF ifadeleri genel olarak ikiye ayrılır : geometri ve kontrol. Geometrik ifadeleri ele alacak olursak :

- BOX (kutu) : Bir dikdörtgen ile sembolik olarak bir kutu tanımlar. Bir merkez noktasından uzunluk ve genişlik değerleri ile tanımlanır. Seçime bağlı olarak belirli bir eksene göre belirli bir açıda dönme de yapabilir. Uzunluğu 25, genişliği 60, merkezi (80,40) noktası olan ve (-20,20) yönünde dönmüş bulunan bir kutunun CIF ifadesi şöyledir : B 25 60 80 40 -20 20; (Şekil 3.13)

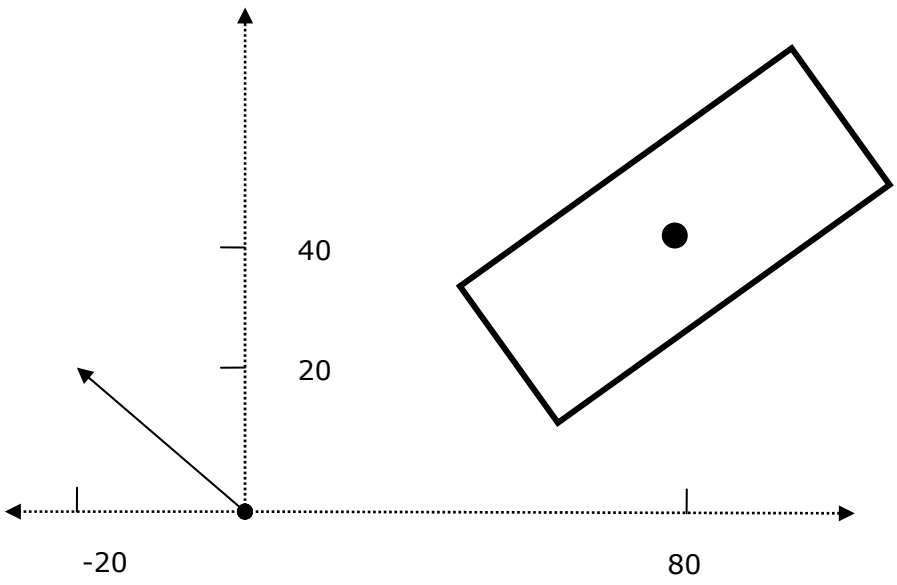
- POLYGON (poligon) : Geometrik olarak poligon, kapalı bir bölgedir. Verilen yol üzerindeki sıralı köşe noktaları ile tanımlanır. Son köşe noktası, başlangıç noktasına tanım gereği bağlanır, doğrudan gösterimde yer almaz. Örnek : P 0 0 10 20 -30 40;

- ROUNDED FLASH (çember) : Yarıçapı verilen bir çember ve merkezi nokta bilgisi ile dairesel bir nesne tanımlamak için kullanılır. Örneğin R 200 -500 800;

- WIRE (tel) : Devre elemanları arasındaki elektrik akımlarının taşısındığı yollar olarak nitelenebilecek olan yapılardır. Detaylı bir tanıma göre ise çizilen bir hat boyunca hattan sabit uzaklıkta yapılacak bir koşu tamamlandığında ortaya çıkan yapıdır. İdeal bir tel, koşu sırasında izlenen yoldan merkeze doğru, sabit uzaklığın yarısı mesafede bulunan noktalar kümesidir. (W 50 0 0 10 20 -30 40;)

- LAYER (katman) : Bir çok katmanın yer aldığı bir yongada her bir geometri elemanı ait olduğu katman ile ilişkilendirilmeli, etiketlenmelidir. Katman tanımlarken sadece katmana verilecen isim kullanılır. Örneğin LND (Layer nmos diffusion)

Bir CIF gösteriminde, bir dikdörtgenin herhangi bir dönme yapmadığı düşünülürse, kenarları koordinat eksenlerine paralel olacaktır. Bu açıdan bakıldığında, konumsal dizinleme yöntemlerinde ve dizin yapılarında kullanılan, geometrik nesnelere yapılan yaklaşımlarda nesneleri temsil eden sınırlayıcı kutularla olan benzerlik dikkat çekicidir.



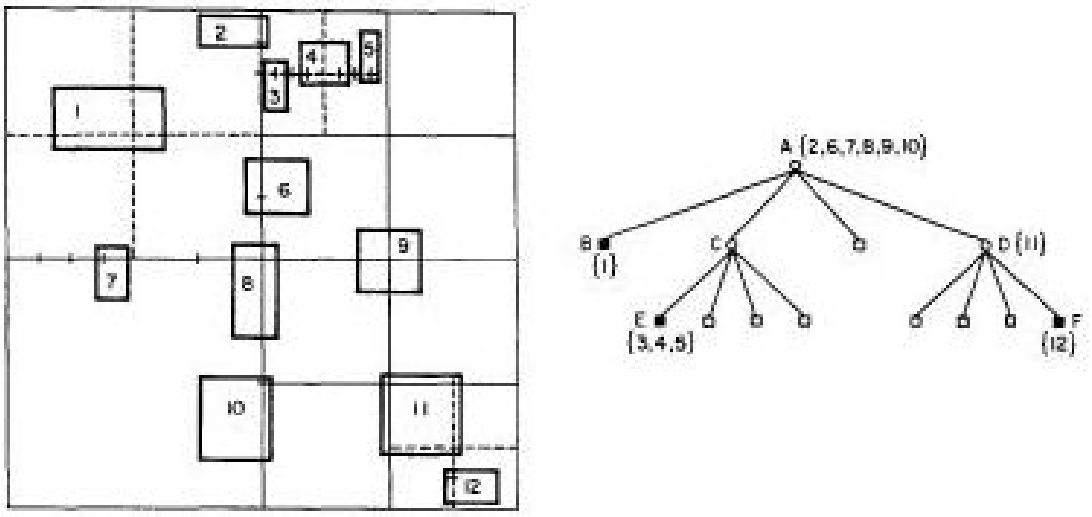
Şekil 3.13 : CIF ile kutu gösterimi, B 25 60 80 40 -20 20

3.3.2 MX-CIF 4'lü Ağacı

MX-CIF 4'lü ağacı, Gershon Kedem tarafından [22]'de öne sürülen 4'lü ağaç yapısıdır. Yüksek ölçekli birleştirim (VLSI) uygulamalarında karşılaşılan yüksek miktarda dikdörtgenlerden oluşan veri kümelerinin hiyerarşik bir yapı ile gösterimine imkan sağlar. Ana amaç verilen bir dikdörtgensel alan (pencere)

içindeki bütün nesneleri toplayabilmektir. Bu yönüyle konumsal erişim yöntemi olarak da kullanılabilir. Coğrafi nesnelerin sınırlayıcı kutuları da birer dikdörtgendir.

MX-CIF 4'lü ağacında her bir R dikdörtgeni, kendisini bütünüyle örten, kapsayan en küçük hücreyle ilişkilendirilir. Dikdörtgenler yaprak veya yaprak olmayan ara düğümlerle ilişkilendirilebilir. Özyinelemeli olarak gerçekleşen bölümleme süreci, bir düğüm hiç bir dikdörtgen kapsamayıncaya kadar devam eder. Veya bir diğer seçenek olarak, bu sonlanma koşulu bölümlenen hücre ebatlarının belirli bir eşik düzeyinin altına inmesi durumuna bağlanabilir. Bu eşik değeri genellikle beklenen bir dikdörtgen ebadına göre seçilir [22]. Şekil 3.14'te örnek bir MX-CIF 4'lü ağacı verilmiştir.



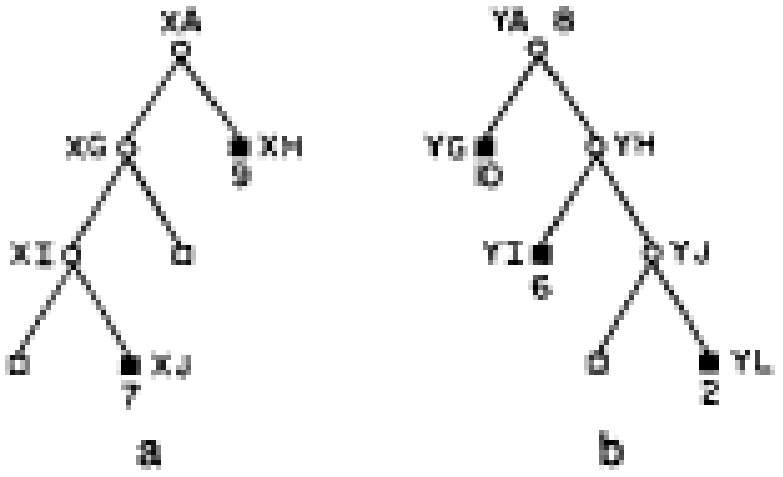
Şekil 3.14 : MX-CIF 4'lü Ağacı

Bir dikdörtgen, bir P düğümüyle ilişkilendirildiğinde P'nin çocuklarının da üyesi olacağı anlamına gelmez. Örneğin Şekil 3.14'te 11 numaralı dikdörtgen hem D hem de F düğümlerinin sınırlarıyla örtüşmektedir. Fakat ilişkilendirildiği düğümün kendisini tamamen örtmesi, kapsaması gerektiğinden dolayı sadece D düğümüyle ilişkilidir.

Bir düğüm ile birden fazla dikdörtgenin ilişkilendirilebileceği açıktır. Bu durumda dikdörtgenlerin düzenlenmesi için çeşitli yollar öne sürülebilir. Abel ve

Smith [8] herhangi bir sıra gözetmeksizin tutulan bir bağlı liste ile kendi yollarını belirlemiştir. Kedem'in benimsediği yol ise bu kadar basit değildir [22].

Kedem'e göre; P ağaçta bir düğüm ve S bu düğümle ilişkilendirilmiş dikdörtgenler kümesi olmak üzere, bu S kümesinin elemanları P hücresinin merkezinden geçen doğrularla kesişimlerine göre 2 alt kümeye ayrılırlar. Bu doğrulara *eksen* veya *eksen çizgileri* denir. Örneğin ebatları $2*L_x$ X $2*L_y$ olan bir P düğümü düşünelim. P düğümünün merkezi de (C_x,C_y) olsun. S kümesinin $x=C_x$ doğrusu ile kesişen elemanları bir kümeyi, $y=C_y$ doğrusu ile kesişen elemanları diğer kümeyi oluştururlar. Eğer bir dikdörtgen her iki eksenle de kesişiyorsa P düğümünü oluşturan hücre bloğunun merkezini içeriyor demektir. Bu durumda y ekseniyle kesişenlerin kümesine dahil edileceği kabul edilir.



Şekil 3.15 : Şekil 3.14'teki A düğümü için ikili ağaçlar
(a) x ekseni (b) y ekseni

Bu alt kümeler ikili ağaçlar şeklinde yapılandırılır. Şekil 3.15, Şekil 3.14'teki A düğümü için oluşan 2 alt kümeyi göstermektedir. Ara düğümlerde bölünme noktalarının eksenleri, yapraklarda ise ilgili bölünme noktası ile kesişen dikdörtgenlerin tanımlayıcıları vardır.

Bu noktada, MX 4'lü ağacı ve MX-CIF 4'lü ağacı arasındaki temel benzerlik ve farklara değinmek gerekirse ilk fark hedeflenen veri tipi olarak verilebilir. MX

4’lü ağacında noktasal veriler kendilerini barındıran en küçük hücre bloğu ile illişkilendirilmekteydi. MX-CIF 4’lü ağacında ise veriler noktasal değildir. Sıfırdan farklı bir ebadı olan dikdörtgenlerdir ve kendilerini bütünüyle örten, kapsayan en küçük hücre bloğuyla ilişkilendirilirler. Benzerlik gösteren her iki durumda da özyinelemeli olarak devam eden bölümleme süreci için ön tanımlı bir bölümleme sınırı vardır. Temel bir fark olarak önemli bir diğer nokta ise ağaç yapılarının oluşumundadır. MX 4’lü ağacında nesnelerin anahtar değerleri yapraklarda bulunmaktadır. Ara düğümler ise dizin yapısına yönelik olarak hizmet vermektedir. Yani ara düğümler ile yapraklar, yapı açısından birebir yanı değildir. MX-CIF 4’lü ağacında ise ara ve yaprak düğümler aynı tiptedir. Böylece veriler hem ara hem de yaprak düğümlerle ilişkilendirilebilmektedir (Şekil 3.14). Boş düğümler ise MX 4’lü ağacındaki WHITE (beyaz) gösterimine sahiptir.

MX-CIF 4’lü ağacının önemli bir avantajı, düzenli bölümleme yapıldığı için bölünme noktalarını tutan vektör benzeri veri yapılarına gerek kalmamasıdır. Örneğin ızgara dosyasındaki S_x ve S_y vektörleri x ve y eksenlerindeki bölünme noktalarını işaretlemek amacıyla değerler tutmaktaydı. Veri ekleme sırasında görülen bölünme ve yeni hücre oluşumlarında bu vektörler güncellenmekteydi.

3.3.2.1 MX-CIF 4’lü Ağacına Veri (Dikdörtgen) Ekleme

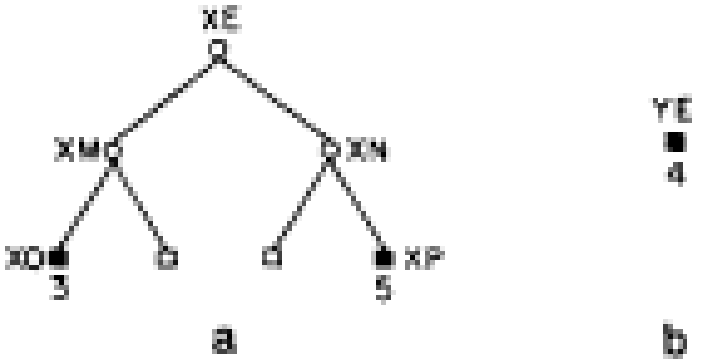
MX-CIF 4’lü ağacı yapısı üç farklı kayıt tipinden oluşur; *cdüğümü(cnode), bdüğümü(bnode)* ve *dikdörtgen(rectangle).* P, bir kaydı gösteren işaretçidir. Bir dikdörtgen kaydı 5 alan içerir:

i. Merkez koordinatları (x ve y olmak üzere 2 alan)
ii. Alt-üst ve yan sınırlara merkezden olan uzaklık (4 kenardan üst ve sağ kenara olan uzaklıklar, sırasıyla alt ve sol kenarlarınkine eşittir, 2 alan)
iii. İsim

A, x ve y eksenlerine göre *X* ve *Y* değerlerini almak üzere *C(P,A)* merkezin ilgili eksen üzerinde kestiği değeri, *L(P,A)* ise merkezden kenara olan uzaklık

değerini verir. İsim (NAME) alanı ise dikdörtgen ile ilgili tanımlayıcı, açıklayıcı bilgidir.

Bir düğümdeki dikdörtgenlerin kümesinden oluşan ikili ağacın düğümleri bnode ile gösterilir. Bir bnode 3 alana sahiptir. Bir dikdörtgen tarafından kapsanan veya ardılları dikdörtgen içeren her bir bölünme noktası için bnode tipinde bir düğüm vardır (Şekil 3.16). Bir bnode kaydı için sağ ve sol çocuklarını gösteren iki alan vardır. P, bir bnode olmak üzere SON(P,I) ile sağ ve sol çocuklara erişilir, sağ-sol ayrımı I ile sağlanır. Üçüncü alan RECT olarak gösterilir ve dikdörtgen(rectangle) tipinde, eksen üzerinde bölünme noktası içeren bir kayda işaret eder. Bu şekilde işaret etmenin gerekli şartı, böyle bir dikdörtgenin var olması ve 4'lü ağaçtaki öncülüyle ilişkilendirilmiş olarak ikili ağaçta bulunmamasıdır. Şekil 3.16 (a)'ya bakıldığında XN ve XM ile gösterilen düğümlerin sırayla sol ve sağ alt ağaçlarında herhangi bir dikdörtgen yoktur. Şekil 3.16(b)'de de görüldüğü gibi, 4 numaralı dikdörtgen E düğümünde yer almasına rağmen y-eskeni kullanılarak ilişkilendirildiği için x-ekseni ile ilgili ikili ağaçta yer almamaktadır.



Şekil 3.16 : Şekil 3.14'teki E düğümü için ikili ağaçlar
(a) x ekseni (b) y ekseni

MX-CIF 4'lü ağacındaki her düğüm *cnode* tipinde, 6 alana sahip bir kayıttır. Bunlardan dört tanesi SON(P,I) olarak gösterilir ve bir P düğümünün dört adet çocuğuna, I üzerinden işaret eder. Geriye kalan 2 alan ise P düğümündeki dikdörtgenlerle ilgili olan ikili ağaçları işaret eder. Bu alanlar ise BIN(P,I) olarak verilir, I yardımıyla eksen belirlenir.

R bir dikdörtgen, R'yi bütünüyle kapsayan hücre bloğu ise B olmak üzere, R'nin sol ve alt kenarları B'nin eksen çizgileri ile aynı doğrultuda ise R dikdörtgeni bu B bloğu ile ilişkilendirilir. Buna yönelik olarak CIF_COMPARE ve BIN_COMPARE fonksiyonları 4 çocuktan birini ve eksenler üzerindeki bölümlemeleri belirlemek için kullanılır. CIF_COMPARE ve BIN_COMPARE fonksiyonlarında kullanılacak olan algoritmalar aşağıda verilmiştir.

```
CIF_COMPARE(P:rectangle , CX:real, CY:real)
// (CX,CY)'de kökü olan MX-CIF 4'lü ağacının, P'nin ait olduğu alt çeyreğini
//döndürür
begin
        return ( if  ( C(P,'X') < CX ) then
                        if (C(P,'Y') < CY) then
                                'SW'
                        else
                                'NW'
                else if ( C(P,'Y')  < CY) then
                                'SE'
                else
                        'NE' );
end
```

Algoritma 3.8 : MX-CIF 4'lü Ağacı (Karşılaştırma – Yön Belirleme)

```
BIN_COMPARE(P:rectangle , CV:real, V:axis)
//P'nin V=CV doğrusunun sağına veya soluna mı düştüğünü, veya onu kapsadığını
//belirle
begin
        return ( if  ( (C(P,V) – L(P,V)) <= CV and CV < (C(P,V) + L(P,V)) ) then
                        'BOTH'
                else if ( CV< C(P,V) ) then
                                'RIGHT'
                else
                        'LEFT' );
end
```

Algoritma 3.9 : İkili Ağaç (Karşılaştırma – Bölünme Noktası Belirleme)

MX-CIF 4'lü ağacına bir dikdörtgen ekleneceğinde, sanki eklenecek olan dikdörtgen aranıyormuş gibi ağaç yukarıdan aşağıya taranır. Dikdörtgenin ekleneceği yeri bulmak iki aşamalı bir süreçtir. İlk olarak eksenlerinden en az birisi dikdörtgenle kesişen hücre bulunur. Bu ilk aşama CIF_COMPARE ile yönlendirilir, CIF_COMPARE fonksiyonunu da CIF_INSERT kontrol eder. İkincil olarak, hücre ve eksen (örneğin V) bulunduktan sonra, bölümleme süreci V üzerinde, dikdörtgen tarafından kapsanan ilk bölümleme noktasını bulana kadar devam eder. Bu aşama ise BIN_COMPARE ile yönlendirilir ve INSERT_AXIS tarafından kontrol edilir.

Eklenecek dikdörtgenin yerini bulma süreci boyunca, MX-CIF 4'lü ağacı ile temsil edilen uzay tekrarlı olarak bölümlenir ve bu sürec sonucunda yeni düğümler ortaya çıkar (splitting). Yeni bir ekleme yapıldığında MX 4'lü ağacında olduğu gibi, MX-CIF 4'lü ağacında da eklenen veri konumunu muhafaza edeceğinden dolayı, her ne kadar ara oluşumlar farklılık gösterebilse de oluşacak son ağaç şekli veri ekleme sırasından bağımsızdır. CIF_INSERT ve INSERT_AXIS fonksiyonları aşağıda verilmiştir.

```
CIF_INSERT (P:rectangle , R:cnode, CX:real, CY:real, LX:real, LY:real)
// P'yi, kökü R'de olan MX-CIF 4'lü ağacına ekle. Ağaç merkezi (CX,CY)'de olan ve
//2*LX x 2*LY ebatlarında bir alanı temsil eder. Eğer ağaç boş ise, R yeni kökü
//gösterir. Create, bütün işaretçileri NIL olarak belirler.
begin
        T : cnode
        Q : quadrant
        DX, DY : direction
        // Ağaç gezilirken, XF ve YF çarpanları yardımı ile alt çeyreklerin merkezleri
        //belirlenir
        preload real array XF['NW','NE','SW','SE'] with  -1.0,1.0,-1.0,1.0
        preload real array YF['NW','NE','SW','SE'] with  1.0,1.0,-1.0,-1.0

        if ( null(R) ) then // Kökü R'de olan ağaç boş ise
                R ← create(cnode)
        T ← R
        DX ← BIN_COMPARE(P,CX,'X');
        DY ← BIN_COMPARE(P,CY,'Y');
        while (DX ≠ BOTH and DY ≠ BOTH) do
                begin //P'nin ait olduğu hücrenin eksenini belirler
                        Q ← CIF_COMPARE(P,CX,CY);
```

```
                    if ( null(SON(T,Q) ) then
                            SON(T,Q) ← create (cnode);
                    T ← SON(T,Q);
                    LX ← LX / 2.0
                    CX ← CX + XF[Q]*LX
                    LY ← LY / 2.0
                    CY ← CY + YF[Q]*LY
                    DX ← BIN_COMPARE(P,CX,'X');
                    DY ← BIN_COMPARE(P,CY,'Y');
            end
        if DX = BOTH then // P, y ekseninde
            INSERT_AXIS (P,T,CY,LY,'Y');
        else // P, x ekseninde
            INSERT_AXIS (P,T,CX,LX,'X');
end
```

Algoritma 3.10 : MX-CIF 4'lü Ağacında Dikdörtgen Ekleme

```
INSERT_AXIS (P:rectangle , R:cnode, CV:real, LV:real, V:axis)
// P'yi kökü CIF düğümü tipinde olan R'de olan V ikili ağacına ekle. Merkezi CV'de
//olan 2*LV uzunluğunda, V ekseninin bir parçasıdır. V ekseni boş ise R yeni kökü
//gösterir. Create, bütün işaretçileri NIL olarak belirler.
begin
        T : bnode
        D : direction
        // Ağaç gezilirken, VF çarpanı yardımı ile alt parçaların merkezleri belirlenir
        preload real array VF['LEFT','RIGHT'] with  -1.0,1.0

        T ← BIN(R,V);

        if ( null(T) ) then // T'deki V ekseni boş ise
                T ← BIN(R,V) ← create(bnode)

        D ← BIN_COMPARE(P,CV,V);

        while (D ≠ BOTH) do
                begin // P'yi kapsayan eksen bölünme noktasını belirle
                        if ( null(SON(T,D) ) then
                                SON(T,D) ← create (bnode);
                        T ← SON(T,D);
                        LV ← LV / 2.0
                        CV ← CV + VF[D]*LV
                        D ← BIN_COMPARE(P,CV,V);
                end
```

RECT(T) ← P
end

Algoritma 3.11 : MX-CIF 4'lü Ağacında Eksen Ekleme

---

n, MX-CIF 4'lü ağacı ve ikili ağaçların en büyük derinliklerinin toplamı olmak üzere, N adet dikdörtgen için kurulan bir MX-CIF 4'lü ağacının en kötü çalışma zamanı O(n*N) ile sınırlanır. Bu durum her dikdörtgen derinliği n olan yerde bulunduğu zaman görülür. Bununla birlikte saklama birimlerindeki en kötü erişim performansı da O(n*N) ile belirlenir. Fakat beklenen gerçek davranış bundan daha iyi olabilir.

Silme işlemi ise çok daha karmaşıktır ve düğümlerin bölünmesinden (split) ziyade tam tersi olarak yeniden derlenip toplanması (collapse) gerekebilir.

3.3.2.2 MX-CIF 4'lü Ağacında Arama

MX-CIF 4'lü ağaçları veri olarak dikdörtgenleri barındırmaktadır ve bu çerçevede düşünüldüğü zaman en genel aramalar, örtüşen, bir başka deyişle kesişen dikdörtgenler üzerine yapılır. En temel sorgu biçimi olan kesişim sorguları CIF_SEARCH ve CROSS_AXIS fonksiyonları ile sağlanabilir. Bu fonksiyonlar aşağıda verilmiştir. RECT_INTERSECT fonksiyonu ise verilen iki dikdörtgenin kesişip kesişmediğini mantıksal bir değer olarak döndürür. Mantıksal değer dışında sayısal değer de döndürülebilir, örneğin kesişen bölgenin alanı. Kesişim yok ise 0 (sıfır), var ise 0'dan farklı bir değer dönecektir.

```
CIF_SEARCH (P:rectangle , R:cnode, CX:real, CY:real, LX:real, LY:real)
//P dikdörtgeninin R'de kökü olan MX-CIF 4'lü ağacının elemanlarından herhangi
//biriyle kesişip kesişmediğine bakılır. Ağaç merkezi (CX,CY)'de olan ve ebatları
//2*LX x 2*LY olan bir alanı temsil eder.
begin
        Q : quadrant
        // Ağaç gezilirken, XF ve YF çarpanları yardımı ile alt çeyreklerin merkezleri
        //belirlenir
        preload real array XF['NW','NE','SW','SE'] with  -1.0,1.0,-1.0,1.0
        preload real array YF['NW','NE','SW','SE'] with  1.0,1.0,-1.0,-1.0

        if ( null(R) ) then // Kökü R'de olan ağaç boş ise
                return (false)
        else if ( not( RECT_INTERSECT( P,CX,CY,LX,LY ) ) ) then
                return (false)
        else if (CROSS_AXIS(P,BIN(R,'Y'),CY,LY,'Y') or
                CROSS_AXIS(P,BIN(R,'X'),CX,LX,'X') ) then
                return (true)
        else
                begin
                        LX ← LX / 2.0
                        LY ← LY / 2.0
                        for Q  in { 'NW', 'NE', 'SW', 'SE' } do
                                if ( CIF_SEARCH( P, SON(R,Q), CX+XF[Q]*LX,
                                        CY+YF[Q]*LY, LX, LY ) ) then
                                        return (true)
                        return (false)
                end
end
```

Algoritma 3.12 : MX-CIF 4'lü Ağacında Arama

```
CROSS_AXIS (P:rectangle , R:bnode, CV:real, LV:real, V:axis)
//P dikdörtgeninin R'de kökü olan ikili ağaçtaki dikdörtgenlerle kesişimine bakılır.
//Ağaç V eksenin 2*LV uzunluğunda bir bölümüne denk gelir, merkezi CV'dirr.
begin
        D : direction
        // Ağaç gezilirken, VF çarpanı yardımı ile alt parçaların merkezleri belirlenir
        preload real array VF['LEFT','RIGHT'] with  -1.0,1.0

        if ( null(R) ) then
                return (false)
        else if ( not(null( RECT(R))) and RECT_INTERSECT(P,C(RECT(R),'X'),
                        C(RECT(R),'Y'), L(RECT(R),'X'), L(RECT(R),'Y') ) then
```

```
        return (true)
else
        begin
                D ← BIN_COMPARE(P,CV,V)
                LV ← LV / 2.0
                return ( if (D=BOTH) then
                        CROSS_AXIS(P,SON(R,'LEFT'), CV-LV, LV, V) or
                        CROSS_AXIS(P,SON(R,'RIGHT'), CV+LV, LV, V)
                        else
                        CROSS_AXIS(P,SON(R,D), CV+VF[D]*LV, LV, V) )
        End
end
```

Algoritma 3.13 : MX-CIF 4'lü Ağacında Eksen Kesişim Belirleme

Aralık (alan) sorguları da verilen algoritmalar kullanılarak gerçekleştirilebilir. Örneğin verilen bir sorgu penceresi ile örtüşen tüm dikdörtgenlerin bulunması söz konusu olduğunda ilk olarak pencere ile bu aralıkta yer alan kayıtların kesişimleri kontrol edilir. Kesişim ilişkisi gösteren adaylar, aday listesine eklenir.

Verilen algoritmalar düzenlenerek birleşim, kesişim gibi farklı küme işlemleri için yeniden uyarlanabilir. Aralık sorgularında, sorgu dikdörtgeni ile MX-CIF 4'lü ağacının kesişen alanlarına bakılır. Bunun yanında açıkca görülmektedir ki mantıksal sorgular da kolayca gerçekleşmektedir.

4. R-Ağaçları

Veri güdümlü konumsal erişim yöntemlerinin başında gelen R-ağaçları Antonin Guttman tarafından [1]'de öne sürülmüş, B-ağaçlarını temel alan dengeli ağaçlardır. Yapıları dikdörtgenlerin düzlemdeki dağılımına göre şekillenir. R-ağacı hiyerarşik bir yapıdadır, düğümleri aracılığıyla disk sayfalarına yönlendirir. Temel alınan B-ağaçları tekil değerli anahtarlar ve bunlar üzerine tanımlanan bir tam sıralamaya dayanır. R-ağaçları ise dikdörtgenlerin içerme ilişkilerine dayanarak bir düzenleme yapar. Bu düzen içerisinde her bir düğüm, bütün çocuklarını içeren en küçük sınırlayıcı kutu ile ilişkilendirilir.

Hedeflenen bir dikdörtgene erişmek için R-ağacı kökten yapraklara doğru izlenen bir yolla taranır ve her düğümde, o düğümün sınırlayıcı kutusuna bakılarak aranan kayıtla örtüşme durumu sınanır. B-ağaçlarından gelen mirasla, derinliği d olan bir R-ağacının performansı dizinlenen nesne sayısına göre logaritmiktir. B-ağaçlarının dizinleme konusunda ortaya koyduğu, kabul edilmiş etkinliğini yakalamayı hedefleyen R-ağaçları zaman ve yer açısından başarılı olmaktadır. R-ağacıyla kurulan bir dizin yapısı tamamen dinamiktir. Ekleme ve silme işlemleri sürekli olarak yinelenebilir, sorgular arasında gerçekleşebilir ve herhangi periyodik bir düzenleme gerektirmez [1].

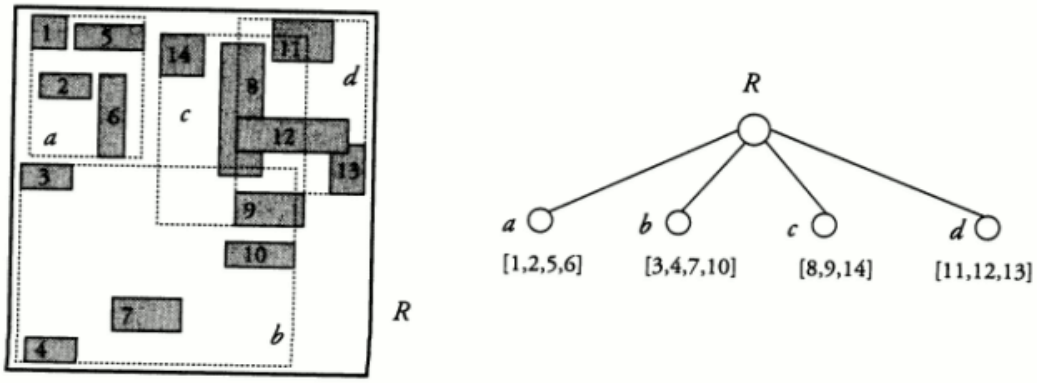
R-ağacının gelişim süreci, B-ağaçlarına benzetilebilir. İlk R-ağacının ortaya çıkmasından sonra bir takım gereksinimler ve düzenlemeler sonucu yeni R-ağacı tipleri ortaya çıkmıştır. Bunlardan en belirgin ikisi R*-ağacı ve R+-ağacıdır.

4.1 Orjinal R-ağacı

Bir R-ağacı her bir düğümü bir dikdörtgene karşı gelen dengeli bir ağaçtır. *Yaprak düğüm,* yaprak kayıtlarının bir dizisini içerir. Bir *yaprak kaydı* (mbb,oid) şeklinde bir ikilidir. d tarama uzayının boyutu olmak üzere *mbb* bilgisi

($x_1,y_1,...,x_d,y_d$) düzenindedir. Yaprak olmayan, ara düğümler ise düğüm kayıtlarının bir dizisini içerir. Bu yapı aşağıdaki özellikleri sağlar:

- Ağaçtaki her bir düğüm için (kök hariç) kayıt sayısı $m \in [0,M/2]$ olmak üzere m ile M arasındadır. Yani bir düğümde tutulabilecek en fazla kayıt sayısı M'dir. Doluluk oranı olarak da bilinen bu kavram B-ağaçlarında da mevcuttur.
- Bir N ara düğümünde her bir (dr, nodeid) kaydı için dr, N düğümünün dizin dikdörtgenidir, yani çocuklarının tamamanı içeren en küçük dikdörtgendir. nodeid ise bu dikdörtgenin sayfa adresidir.
- Her bir yaprak düğümün (mbb,oid) kaydı için, mbb bilgisi, oid adresinde tutulan nesnenin konumsal bileşenlerinin en küçük sınırlayıcı dikdörtgenidir.
- Kök, yaprak olmamak şartıyla en az iki kayda sahiptir.
- Tüm yapraklar aynı seviyedir.

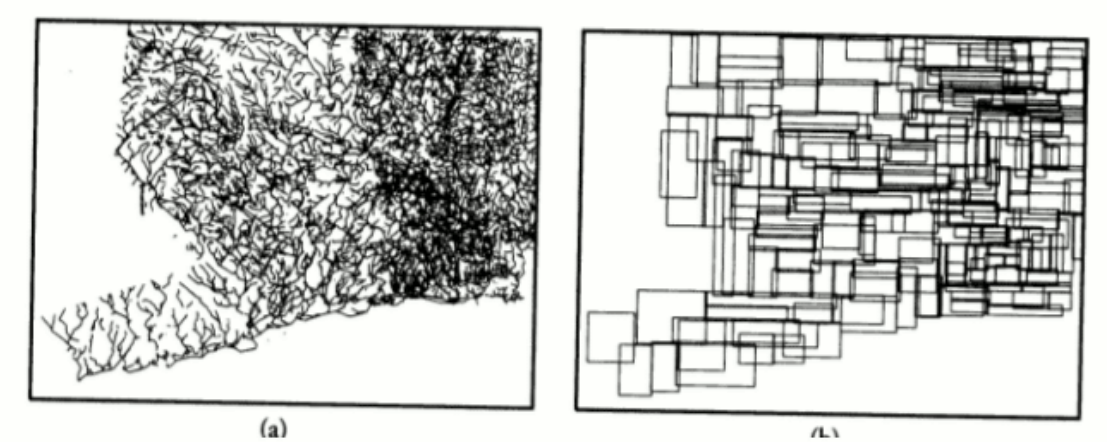


Şekil 4.1 : R-tree örneği

Şekil 4.1'de m=2 ve M=4 olan bir R-ağacı görülmektedir. Dizinlenen C kümesinde 14 adet nesne bulunmaktadır. a, b, c ve d yapraklarının dizin dikdörtgenleri noktalı hatlarla gösterilmiştir.

Yukarıdaki listede verilen özellikler devingen ekleme ve silme işlemleri sonrasında da korunacaktır. Dikkat edilmesi gereken nokta bu özelliklerin

korunması ile ağacın dengede kalması ve veri kümesinin dağılımına uyum gösterebilmesidir. Arama uzayında çok sayıda nesne içeren bir alan, çok sayıda komşu yaprak oluşmasına sebep olacaktır. Bu, 4'lü ağaçlar gibi uzayın bölümlenmesine dayanan yöntemlerde görülmeyen bir durumdur. 4'lü ağaçlarda bazı dallar bölgesel veri yoğunluğuna bağlı olarak çok uzun olabilmektedir. Bazıları da doğal olarak kısa olmaktadır. Fakat ağacın derinliğinden bahsedilirken en uzun dal belirleyici olmaktadır.



Şekil 4.2 : Hidrografik Veri (a) ve bu veri kümesine denk gelen R-ağacı (b)

Şekil 4.2'de hidrografik bir veri kümesi ile oluşturulmuş bir R-ağacındaki dikdörtgenler görülmektedir. Açıkca görüldüğü gibi bölgesel yoğunluk bulunmaktadır ve bu da örtüşen dikdörtgen sayısı bakımından önem arz etmektedir. Bir düğümde yer alabilecek en fazla kayıt sayısı M, kayıt büyüklüğüne (Size(E)) ve disk sayfasının kapasitesine (Size(P)) bağlıdır.

$$M = \lfloor \text{Size}(P) / \text{Size}(E) \rfloor$$

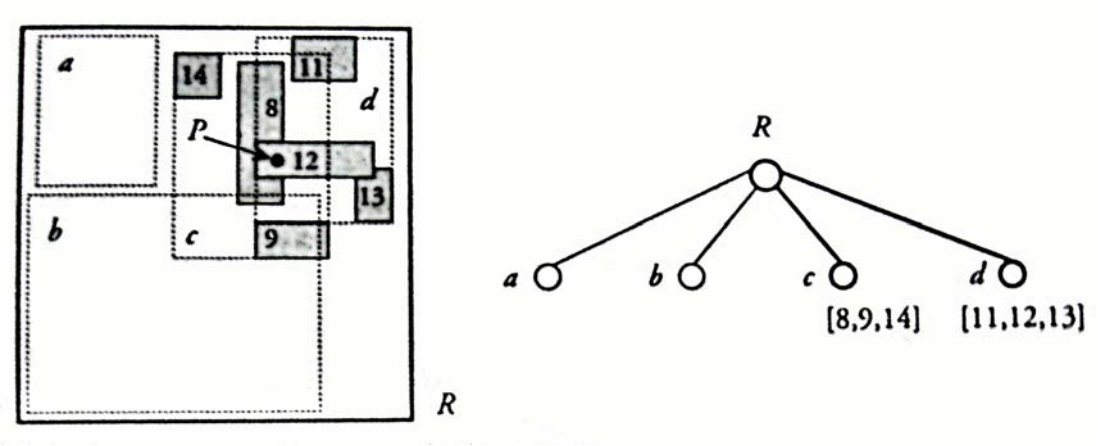
M ve m değerleri ile birlikte, derinliği d olan bir R-ağacı en az m^{d+1}, en çok da M^{d+1} kayıt içerebilir. Bir başka deyişle, N tane kayıt içeren bir R-ağacının derinliği en az $\lfloor Log_M(N) -1 \rfloor$, en çok da $\lfloor Log_m(N) -1 \rfloor$ olabilir. Tam değer sayfa kullanım oranına, etkinliğine bağlıdır.

Bunu bir örnekle açıklamak gerekirse, sayfa ebadını 4 Kbayt, kayıt ebadını da 20 bayt (16 bayt mbb için, 4 bayt oid için) olarak kabul edelim ve m, sayfa kapasitesinin %40'ı kadar olsun. M=204 ise m=81 olacaktır. Bu kabuller doğrultusunda derinliği, d=1 olan bir R-ağacı 6561 nesne, d=2 olduğunda 531441, d=3 olduğunda 8489664 nesne dizinleyebilir. Bu çerçevede düşünüldüğün 1 milyon kayıt içeren bir R-ağacı ile istenen nesneye ulaşmak için 3 erişim yeterlidir.

M ve m değerleri, VTYS verimliliği açısından oldukça belirleyicidir. M, disk sayfası boyutu gibi sabit disk ile ilgili parametrelere uygun olarak seçilmelidir. Bunun yanısıra m değeri de VTYS başarımı açısından önemlidir. Eğer veritabanı güncelleme ve veri girişinden ziyade yoğun olarak arama amaçlı kullanılacak ise m değeri dizinleme ağacının derinliğini en aza indirmek üzere büyük seçilmelidir. Derinlik az olursa dizin yapısının başarımı artacaktır. Fakat m değeri büyük olduğunda düğümlerin doluluk oranı yüksek olacağından taşma potansiyeli de yüksek olacaktır. Diğer seçenek olarak m değeri küçük belirlenirse veritabanı sık veri girişi ve güncellemelerine daha uygun bir yapı sergileyecektir. Düğümlerin doluluk oranları düşük olacağından taşma riski de azalmış olacaktır.

4.1.1 R-ağacında Arama

Arama işlemi daha önceki hiyerarşik yapılarda olduğu gibi genelden özele doğru bir yol izlenerek yapılır. R-ağacının kökünden yapraklarına doğru inen bir yolda gerekli konumsal ilişkiler kontrol edilerek istenen kayıtlar geri getirilebilir. İlk olarak nokta sorgulamayı ele alacak olursak bunu aşağıda algoritmik olarak verilen RT_POINTQUERY ve RTREE_TRAVERSAL fonksiyonları yardımıyla gösterebiliriz.



Şekil 4.3 : R-ağacında Nokta Sorgulama

```
RT_POINTQUERY (P:point)
begin
        result = ∅
        //Adım 1: Kökten başlayarak ağacı gez, dizin dikdörtgeni P'yi kapsayan
        //yaprakların listesini oluştur
        SL = RTREE_TRAVERSAL(root,P)
        //Adım 2: Yaprakları tara ve P'yi kapsayanları bul
        scan the leaves, and keep the entries that contain P
        for each e in L do
                //Yapraktaki her elemanı noktayı içerme durumuna göre test et
                //Eğer içeriyorsa sonuç listesine ekle
                if (e.mbb contains P) then
                        result += {e.oid}
        end for
        return result
end
```

Algoritma 4.1 : R-ağacında Nokta Sorgulama

```
RTREE_TRAVERSAL (nodeid:PageID, P:point)
begin
        result = ∅
        // Düğümün sayfasını oku
        N = READPAGE(nodeid)

        if (N is a leaf) return {N}
```

```
        else
                // N düğümünden kayıtları tara ve P'yi içerenlere uğra
                for each e in N do
                        if (e.dr contains P) then
                                result += RTEETRAVERSAL( e.nodeid, P)
                        end if
                end for
        end if
        return result
end
```

Algoritma 4.2 : R-ağacında Dolaşım

Nokta sorgulama işleminde ilk olarak kökün bütün çocukları gezilir ve P noktasını içerme durumlarına bakılır. Dizin dikdörtgenlerinin örtüşme durumu varsa nokta birkaç dizin dikdörtgeninin kesişiminde yer alıyor olabilir. Bütün bu kesişen dikdörtgenlere uğramak gerekecektir. Bu süreç yapraklara ulaşana kadar her seviyede devam edecektir. Ziyaret edilen her N düğümü için iki durum görülür :

- Söz konusu düğümde, hiçbir dikdörtgen noktayı içermiyorsa arama sona erer. Bu durum P noktası, düğümün dizin dikdörtgeninde yer alsa bile gerçekleşebilir. Bu durumda P noktası *ölü alana* düşmektedir.

- P noktası bir veya daha fazla kayıt için dizin dikdörtgenleri tarafından içeriliyorsa her bir alt ağaç gezilmelidir.

Kökten yapraklara doğru birden fazla yol kullanılarak gezilebilir. İlk adımda P noktasını içeren yaprakların listesi elde edilir. İkinci adımda ise bu yapraklardaki her bir kayıt P noktasını barındırma durumuna göre kontrol edilir. Şekil 4.3'te bir R-ağacında nokta sorgulama örneklenmiştir. 8 ve 12 numaralı nesnelerin sınırlarında yer alan P noktası kullanılarak bir sorgu yapılmaktadır. Sorgulama sırasında R, c ve d düğümleri olmak üzere üç düğüme erişilir.

Eğer sorgu noktası her seviyede sadece bir dikdörtgenin içinde kalıyorsa derinliği d olan bir ağaçta d tane erişim gerekir. Bu nadir görülen bir durum olarak

düşünülürse çoğunlukla kökten yapraklara doğru birkaç yol izlenebilir. Fakat ağaç yapısından dolayı erişim sayıları yine logaritmiktir. Bütün dikdörtgenlerin ortak bir kesişiminin bulunduğu ve sorgu noktasının bu alana düştüğü en kötü durumda ise tüm ağaç gezilecektir.

Örtüşen dikdörtgenler, gezilecek düğüm sayılarını yakından ilgilendirdiğinden dolayı buna yönelik olarak örtüşmeleri indirgemek amacıyla farklı R-ağacı tipleri öne sürülmüştür (R*-ağacı [48]).

R-ağacında pencere sorguları, nokta sorgularının genelleştirilmiş hali gibi düşünülebilir. P noktası ve içerme ilişki (contains P) , W penceresi ve örtüşme ilişkisi ile değiştirildiğinde alan sorguları da yapılabilir. Pencere boyu büyüdükçe gezilecek düğüm sayısı da artacaktır.

4.1.2 R-ağacına Veri Ekleme

R-ağaçlarında ekleme B-ağaçlarındakine benzer şekilde yapılır. Yeni kayıt, yaprak seviyesinde eklenir, taşan düğümler bölünür ve bu bölünmeler yukarıya, köke doğru yayılır.

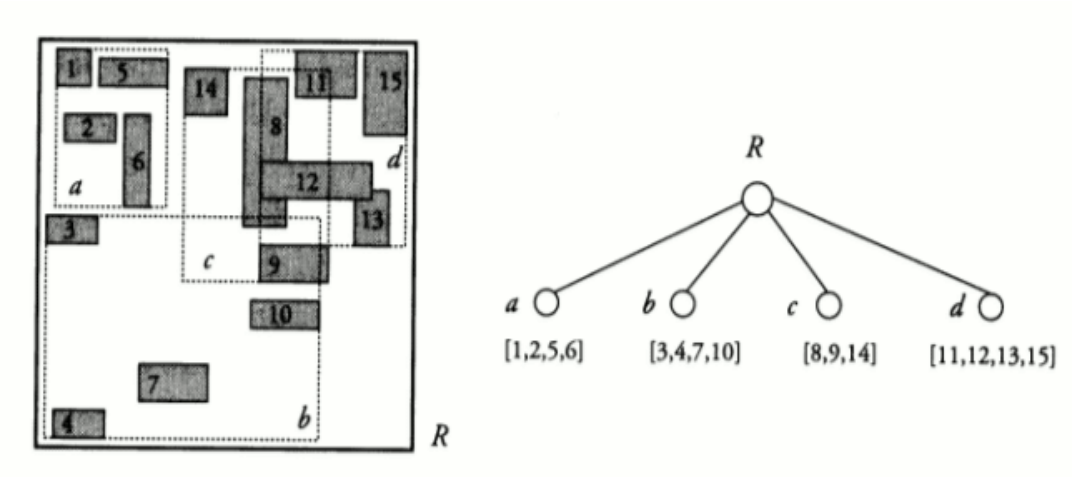
Ağaca bir veriyi eklemek için, önce ağaç yukarıdan-aşağıya doğru taranır. Her bir seviyede ya nesnenin mbb'sini kapsayan bir dizin dikdörtgeni içeren bir düğüm bulunur ve bu durumda o düğümün alt ağacına yönelinir ya da böyle bir düğüm yoktur. Daha sonra nesnenin dizin dikdörtgeninin genişletilmesi en düşük olacak şekilde bir düğüm seçilir. Bu işlem bir yaprak düğüme gelinene dek devam eder.

Eğer yaprak dolu değilse *[mbb,oid]* olarak yeni kayıt o yaprak ile ilişkili olan sayfaya kaydedilir. Eğer yaprak dizin dikdörtgeni genişeltilecekse, üst düğümde ona karşı gelen kayıt yeni dizin dikdörtgeni değeri ile güncellenmelidir. Bu güncelleme yukarıya doğru yayılabilir. En kötü durumda köke kadar ulaşır.

Eğer l yaprağı dolu ise, bu durumda bir *bölünme* (split) işlemi gerçekleşir. Yeni bir l' yaprağı oluşturulur ve M+1 adet kayıt l ve l' arasında dağıtılır.

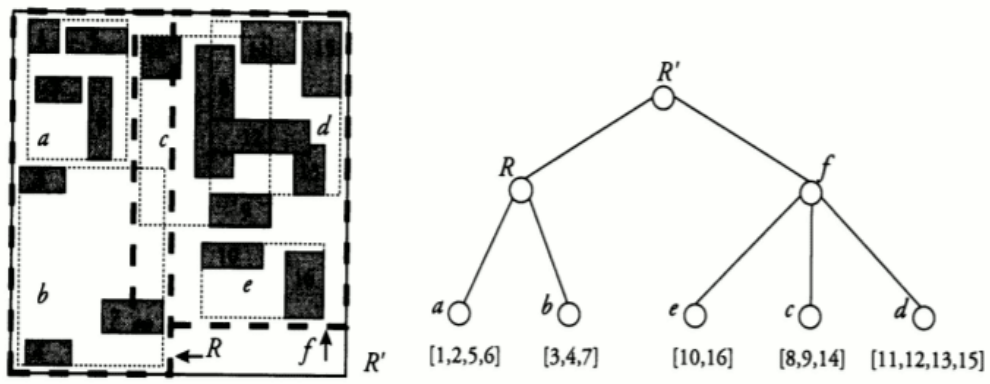
Yaprakların bölünmesi ve yukarılara doğru yayılma, algoritmanın püf noktasıdır denebilir.

Bölünme işleminden sonra eski l yaprağının atası olan f düğümünün kayıtlarının güncellenmesi gerekir. Ayrıca yeni l' yaprağının kayıdı da f'ye ilave edilmelidir. Eğer f dolu ise, benzer bölünme işlemi yeniden uygulanır. Bölünme köke doğru yayılabilir. Eğer kök de bölünürse ağacın derinliği 1 arttırılır.



Şekil 4.4 : R-ağacında veri ekleme (15 numaralı dikdörtgenin eklenmesi)

Şekil 4.4'te örnek bir R-ağacına 15 numaralı dikdörtgen eklenmiştir. Gösterimde M=4 kabul edilmiştir. 15 numaralı nesne ilk olarak d yaprağına eklenmiştir, daha sonra d yaprağının dizin dikdörtgeni yeni nesneyi kapsayacak şekilde genişletilmiştir. Bu genişlemenin sonucu olarak da kökteki d kaydı güncellenmiştir.



Şekil 4.5 : R-ağacında veri ekleme (16 numaralı dikdörtgenin eklenmesi)

Şekil 4.5'te ise 16 numaralı dikdörtgen eklenmektedir. İlk başta b yaprağına giren nesne, yaprak dolu olduğundan taşmaya sebep olmaktadır. Çünkü b yaprağı {3,4,7,10} kayıtlarını barındırmaktadır. Bu yüzden b bölünecektir, yeni bir e yaprağı oluşacaktır. {3,4,7,10,16} kümesi b ve e arasında dağıtılacaktır. Bu liste ortadan ikiye bölündüğünde {3,4,7} b yaprağına, {10,16} ise e yaprağına kaydedilecektir.

Bu bölünmeden sonra b'nin atasında da değişiklik olacaktır. Yeni kayıt, e, R'de bölünmeye yol açacaktır. Yapraktakine benzer bir şekilde yeni bir f düğümü yaratılır, a ve b R'nin içinde yer alırken, e,c,d, f'ye kaydedilir. En son ise R' yeni kök olmak üzere yaratılır R ile f, bu yeni kökün çocukları olur. Ekleme algoritması aşağıdaki gibidir.

```
INSERT (e:LEAFENTRY)
begin
        //Aramaya kökten başla
        node = root
        // Yapraklara doğru bir yol belirle
        while (node is not a leaf) do
                node = CHOOSESUBTREE (node, e)
        end while
        // Yaprak düzeyinde ekle
        INSERTINLEAF (node, e)
        // Yaprak taşma durumuna gelirse, böl ve yukarı doğru yayılmaları yukarı
        //doğru giderek düzenle, ağacı dengele
        if (node overflows) then
                SPLITANDADJUST (node)
        else
                ADJUSTPATH (node)
        end if
end
```

Algoritma 4.3 : R-ağacında Ekleme

CHOOSESUBTREE fonksiyonu, dizin dikdörtgeni e.mbb'yi içeren veya içermesi için en küçük genişletmenin yapılacağı düğümü seçer. Eğer birden fazla uygun aday varsa en küçük alana sahip olan seçilir.

ADJUSTPATH fonksiyonu ise dizin dikdörtgenlerinde ekleme sonucu oluşan genişlemelerin üst seviyelere yayılımını düzenler. Bu süreç, üst seviyedeki bir dizin dikdörtgeninin genişlemeye ihtiyacı olmadığında veya köke ulaşıldığında sona erer.

```
ADJUSTPATH (node:NODE)
begin
        if ( node is root) return
        else
                //Düğümün atasını bul
                parent = GETPARENT(node)
                //Atasındaki düğümün kaydını düzenle
                if ( ADJUSTENTRY(parent, [ node.mbb,node.id ] ) ) then
                        // Kayıt düzenlenirse, atanın yolunu düzenle
                        ADJUSTPATH(parent)
                end if
        end if
end
```

Algoritma 4.4 : R-ağacında Dizin Dikdörtgeni Genişletme

ADJUSTENTRY(node, child_entry) fonksiyonu mantıksal değer döndürür. node içinde child_entry.id'ye karşılık gelen bir e kaydı için e.mbb ile child_entry.mbb bilgilerini karşılaştırır. Eğer e.mbb, çocuğun dizin dikdörtgeni genişlediği için güncellemeye ihtiyaç duyarsa güncellenir, ve fonksiyon "true" döndürür. Aksi durumda ise dönen değer "false" olacaktır.

Düğümlerin taşma durumunda düzenlenmesi yani taşan bir düğümün yeni düğümler yaratılarak kayıtlarının dağıtılması ve bu bölünmeler üst seviyelere yayılırken uygulanacak strateji başarımı etkileyen önemli bir noktadır. Aşağıda

verilen SPLITANDADJUST fonksiyonu düğümlerde taşma yaşandığında bölünme işlemini ve sonraki düzenlemeleri gerçekleştirir. Bu görevi yerine getirirken de ADJUSTPATH ve ADJUSTENTRY fonksiyonlarını kullanır, üst seviyelere yayılan taşmalar olduğunda öz yinelemeli olarak çağrılabilir.

```
SPLITANDADJUST (node:NODE)
begin
        //Yeni düğüm yarat ve kayıtları dağıt
        new_node = SPLIT(node)
        if (node is root) then
        CREATENEWROOT(node, new_node)
        else
        //Düğümün atasını bul
                parent = GETPARENT(node)
                //Atasındaki düğümün kaydını düzenle
                ADJUSTENTRY(parent, [ node.mbb,node.id ] )
                //Yeni düğümü ataya ekle
                INSERTINNODE(parent, [ new_node.mbb, new_node.id ] )
        if (parent overflows) then
                        SPLITANDADJUST (parent)
                else
                        ADJUSTPATH (parent)
                end if
        end if
end
```

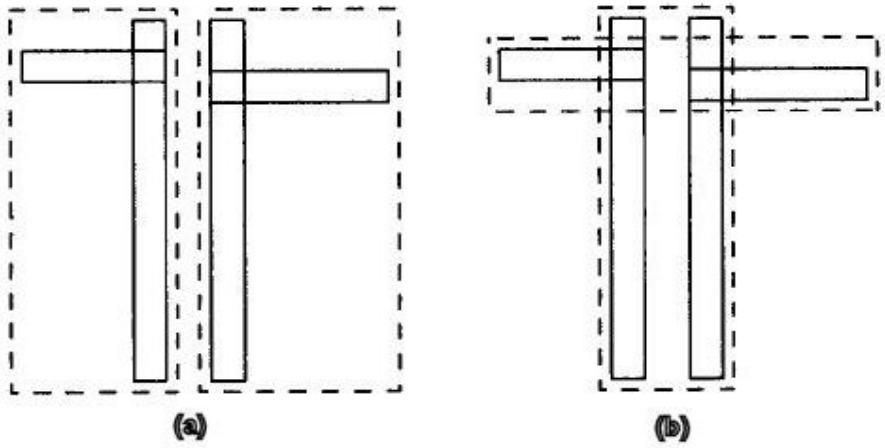
Algoritma 4.5 : R-ağacında Taşma Durumunda Bölünme Düzenleme

CREATENEWROOT(node, new_node) fonksiyonu, kök düğümde bir taşma olduğunda yeni bir seviyede yeni kökün oluşmasını sağlar. Burada dikkat edilmesi gereken husus kökün, diğer düğümlerdeki doluluk oranını sağlamak zorunda olmadığıdır.

4.1.2.1 Bölünme (Split)

Bölünme işleminin genel başarımda etkili olduğunu daha önce belirtmiştik. M tane kayıt içeren bir düğüme yeni bir kayıt ekleneceğinde M+1 nesneyi iki düğüm arasında dağıtmak gerekecektir. Bu bölünme işlemi öyle yapılmalıdır ki arama işlemlerinde her iki düğüm birden taranmak zorunda kalınmamalıdır veya

bu durum olabildiğince aza indirgenmelidir. Arama yapılırken gezilecek düğümler, dizin dikdörtgenlerinin sorgu penceresi ile kesişimlerine bakılarak belirlendiğinden, bölünme sonucu oluşan iki düğümün toplam alanlarının olabildiğince küçük olması hedeflenmelidir [1]. Şekil 4.6'da iyi ve kötü bölme örnekleri verilmiştir.



Şekil 4.6 : Kötü (a) ve iyi (b) bölünme

Taşan bir düğümü iyi bir şekilde bölümlemek için akla ilk gelen yol kabaca saldırarak olası bütün grupları üretmek ve bunlar içerisinden alanları toplamı en az olan iki tanesini seçmektir. M+1 elemanlı bir dikdörtgen kümesinde yaklaşık olarak 2^{M-1} kadar olasılık vardır [1]. İki boyutta bir dikdörtgen 4 baytlık 4 adet sayı, bir işaretçi ise 4 bayt ile gösteriliyorsa bu durumda her bir dizin kaydı 20 bayt yer kaplayacaktır. Bir disk sayfasının 1024 bayt olduğu kabul edilirse M değeri yaklaşık olarak 50 olacaktır. Böyle bir durumda da bütün olası bölünmelerin içinden en iyiyi seçmek zor ve maliyetli olacaktır.

Çalışma zamanı ve sonuç kalitesi açısından bakıldığında akılcı bir çözüm bulunmalıdır. Bu amaca yönelik olarak Guttman[1], *QuadraticSplit* algoritmasını öne sürmüştür. Bu algoritma kesin olarak garanti etmemekle birlikte en küçük alanlı bölme işlemini yapmayı hedefler ve M değerine göre karesel bir davranış sergiler[1].

İlk olarak M+1 elemandan e ve e' olmak üzere iki tanesi birbirinden en uzak olacak şekilde seçilir. Bir başka ifade ile öyle iki eleman seçilmelidir ki bu iki eleman aynı gruba konduğu takdirde boşa harcanan alan en büyük olmalıdır.

Örneğin her iki elemanı da kapsayan dikdörtgenin alanından iki nesnenin alanı çıkarıldığında kalan alan. Bu alan ölü alandır. Seçilen iki nesne bölünme sonucu oluşacak iki farklı grubun birinci elemanı olurlar. Geriye kalanlar içerisinden de her adımda birer tanesi gruplara dağıtılır.

Bir grubun, bir eleman eklendikten sonra ölü alanından geriye kalan bölüm *genişleme alanı* olarak adlandırılır. Her adımda, iki grup arasında genişleme alanları bakımından en büyük farkı yaratacak olan kayıt aranır. Bu kayıt kendine en yakın gruba, yani genişleme alanı en küçük olana eklenir. Eşitlik durumunda ikincil koşul olarak en küçük alana sahip olan grup ve en az elemana sahip grup özelliklerini sağlayan seçilebilir. Bunlar da eşitse herhangi birine eklenebilir [1].

```
QUADRATICSPLIT (E:set of entries)
begin
        J : MBB
        worst = 0, d1, d2, expansion = 0 : integer
        //Her grubun ilk elemanını seç
        //E' geriye kalanların kümesi olsun
        for each e in E do
        for each e' in E do
                J = mbb(e.mbb, e'.mbb)
                if ( (area(J) – area(e.mbb) – area(e'.mbb) ) > worst ) then
                        worst = (area(J) – area(e.mbb) – area(e'.mbb) )
                        G1 = {e}
                        G2 = {e'}
                end if
        end for
        end for

        //Her iki grup da başlangıç değerleri ile belirlendi. Şimdi kalanları dağıt
        E' = E – (G1 ∪ G2)
        while ( E' ≠ ∅ ) do
        //E' içinden her elemanın ekleme maliyetini hesapla
                for each e in E' do
                d1 = area(mbb(e.mbb, G1.mbb) – e.mbb)
                d2 = area(mbb(e.mbb, G2.mbb) – e.mbb)

                if ( (d2 – d1) > expansion ) then
                        best-group = G1
                        best-entry = e
                        expansion = d2 – d1
                end if
```

```
            if ( (d1 – d2) > expansion ) then
                best-group = G2
                best-entry = e
                expansion = d1 – d2
            end if
        end for

        //En iyi adayı en iyi gruba ekle
        best-group = best-group ∪ best-entry
            E' = E' – {best-entry}

            //Her grup en az m tane kayıt tutmalı
            //Eğer kalan kayıtların tümünün eklendiği grubun eleman sayısı m'e
            //eşit olacaksa, kalanları o gruba ekle
            if ( |G1| = m - |E'| ) then
            G1 = G1 ∪ E'
            E' = ∅
            end if

            if ( |G2| = m - |E'| ) then
            G2 = G2 ∪ E'
            E' = ∅
            end if

        end while

end
```

Algoritma 4.6 : QUADRATIC SPLIT

Algoritmanın, grupların ilk atamaları ve sonraki elemanların belirlenmesi olarak ortaya koyabileceğimiz iki bölümü de M değerine göre kareseldir.

Bunun yanısıra bir de doğrusal davranış sergileyen çözüm vardır. *Linear Split* olarak bilinen bu yöntem, Quadratic Split'e benzemekle beraber, eleman seçimleri bakımından kendi farkını ortaya koymaktadır. İlk eleman atamasını yaparken eksenler üzerinde birbirine en uzak elemanları seçer, ikinci aşamada ise her bir elemanı genişlemesi en az olacak gruba atar. Daha basit ve daha hızlı olmasına rağmen örtüşen gruplar yaratmaktadır ki bu da arama işlemlerinde az rastlanması istenen bir durumdur. Linear Split algoritması kendinden daha maliyetli yöntemler kadar iyi olduğunu kanıtlamıştır, hızlıdır ve göreceli olarak

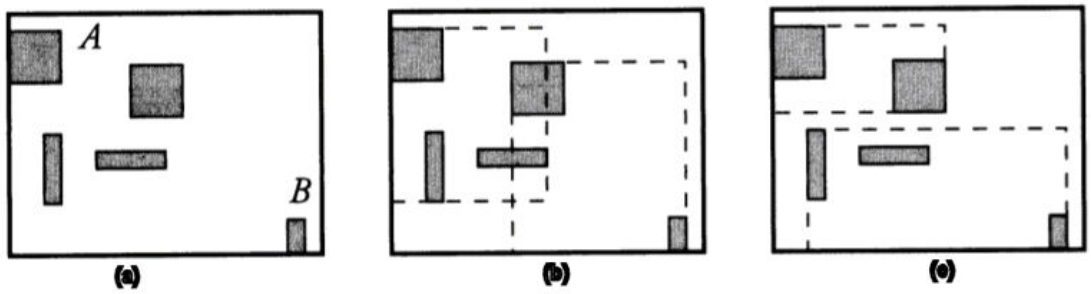
yarattığı kötü bölme sonuçları arama performansını göz ardı edilebilecek boyutlarda etkilemektedir [1].

4.2 R-ağacı*

1990 yılında [48]'de öne sürülen R*-ağacı, kendisinden önceki R-ağacına bir takım iyileştirmeler ile katkıda bulunmaktadır. Bu iyileştirmeleri sağlamak adına yapılması gerekenlerden kısaca bahsetmek gerekirse ;

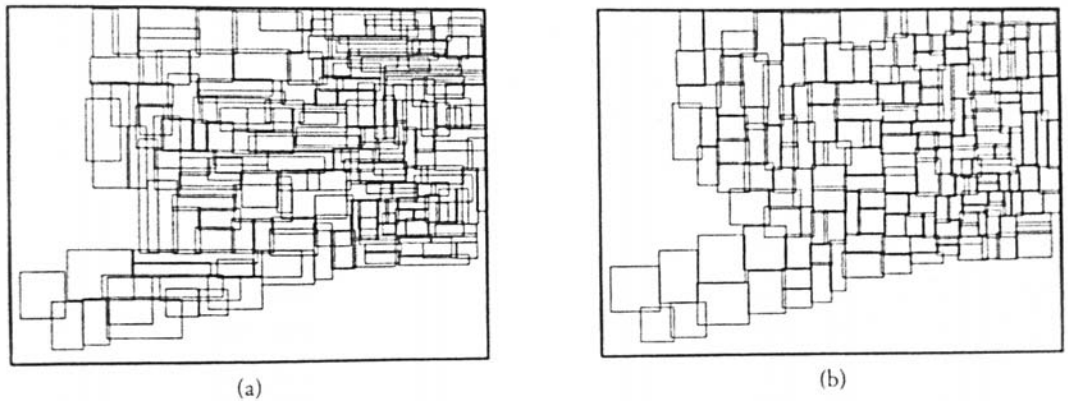
- Bir dizin dikdörtgeninin kapladığı alan en aza indirgenmelidir. Düğümlerdeki ölü alanlar azaltılmalıdır. Bu sayede arama yapılırken kökten aşağı inen yolda yapılan karşılaştırmaların başarımı yükselecektir.
- Dizin dikdörtgenlerinin örtüşme oranları düşürülmelidir. Böylece tekrarlı alanlar üzerine yapılan işlemlerden tasarruf edilecektir.
- Dizin dikdörtgenlerinin çapı küçültülmelidir.

Bu maddelerin hepsini birden en iyi düzeyde sağlayabilmek her zaman mümkün olmamaktadır. R*-ağacının temel ilkesi, R-ağacının taşma durumunda bölme çözümünü iyileştirmektir. Temel olarak R*-ağacındaki yaklaşım bir eksen belirleyip o eksen üzerinde bir hat çekip, hattın farklı taraflarında kalan gruplar oluşturmaktır. Şekil 4.7'de R-ağacı ve R*-ağacının aynı taşma durumunda gösterdiği sonuçlar örneklenmiştir.



Şekil 4.7 : (a) Taşma Durumunda Bir Düğüm (b) R-ağacı bölmesi (c) R*-ağacı bölmesi

Şekil 4.2'de verilen hidrografik kümesi kullanılarak R-ağacı ve R*-ağacı ile elde edilen sonuç dizin yapıları Şekil4.8'de sunulmuştur.



Şekil 4.8 : (a) R-ağacı ile (b) R*-ağacı Sonuçları

4.3 Veri Yoğunlaştırma ve R-ağacı

R-ağacı ve türevleri dinamik ortamlarda ekleme ve silme işlemlerini gerçekleştirebilecek bir yapıya sahiptirler. Doluluk oranı (m) belirlenirken kurulan dizin yapısının kullanılacağı ortam ve hedefler önemlidir. Dinamik bir ortamda R-ağacı %50 civarında bir doluluk oranı belirler, deneysel çalışmalar ise ortalama olarak bu oranın %70 olduğunu göstermektedir [42]. Durağan (statik) bir ortamda bu oran daha da iyileştirilebilir. Doluluk oranının artması sayesinde ağaçtaki boşluklar azalacak ve bu boşluklardan dolayı derinliği artan ağaç küçülecek, sığlaşacaktır. Bu sayede de arama performansı tepki süresi açısından iyileşecektir.

Boş alanların azaltılması ve ağacın yoğunlaştırılması için veriler eklenmeden önce bir takım *ön işlemlere* tabi tutulur. Bu mantıkla kurulan bir dizin yapısında verilerin her zaman durağan olması gerekmez. Güncelleme yapıldığında veya belirli aralıklarla veriler yeniden düzenlenerek dizin yapısı yeniden kurulabilir. Bu sayede R-ağacının doluluk oranları %100'e yakın değerlere taşınabilir. Verilerin

sıklaştırılmasına yönelik olan bu türlü algoritmalara *yoğunlaştırma (packing) algoritmaları* denir.

Genel süreç B-ağaçlarının yapılandırılmasına benzer. İlk önce yapraklar ve daha sonra üst seviyeler ve kök oluşur. Her bir düğümün tutabileceği dikdörtgen sayısı M, veri kümesindeki toplam dikdörtgen sayısı ise N olmak üzere genel bir yoğunlaştırma algoritması şöyle düşünülebilir :

- Veri kümesi (dosyası) N adet dikdörtgen, ⌈N/M⌉ tane ardışık gruba sıralı bir şekilde dağıtılmak üzere ön işlemden geçirilir. Her bir grup aynı seviyedeki yapraklara yerleşecektir. Sona kalan grup, b'den daha az sayıda eleman içerebilir.
- ⌈N/M⌉ tane grup sayfalara yerleştirilir ve (mbb, oid) düzenindeki dizin kayıtları her bir yaprak için geçici bir dosyaya aktarılır.
- Geçici dosyada tutulan dikdörtgenler (mbb) özyinelemeli olarak ağaç yapısında köke varana kadar yoğunlaştırarak yerleştirilir.

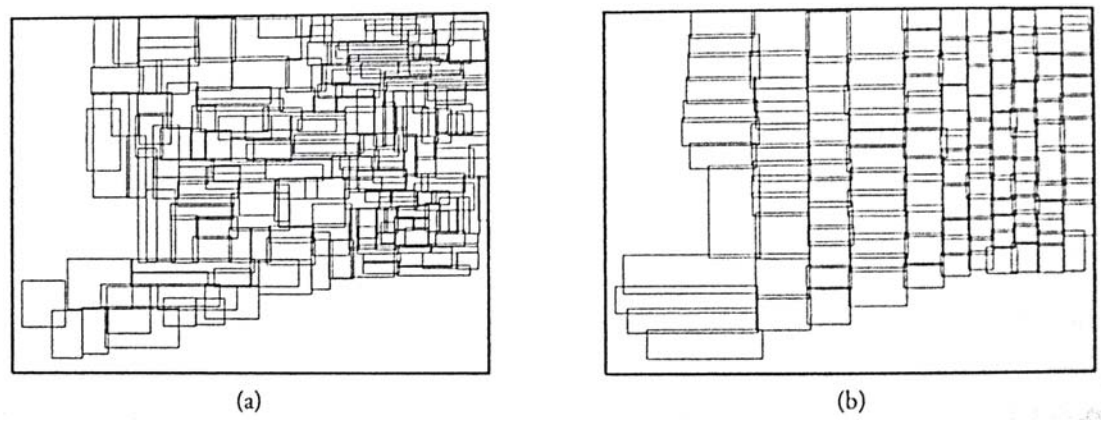
4.3.1 STR (Sort-Tile-Recursive) Yoğunlaştırma Algoritması

Genel çerçevesi maddeler halinde verilen veri yoğunlaştırma algoritmaları verileri yoğunlaştırırken (sıkıştırırken) izledikleri yola göre farklılaşırlar. [58]'de öne sürülen STR (Sort-Tile-Recursive) algoritması basit ve etkili bir çözümle kendisinden önceki yoğunlaştırma algoritmalarına oranla %50'ye varan üstünlük sergilemektedir.

STR çözümünde genel çözüm maddelerinde olan kabuller temel alınır. N tane dikdörtgen, kapasitesi M olan ⌈N/M⌉ tane yaprağa yerleştirilecektir. Yaprak sayısına P dersek, yatayda ve dikeyde sayıları eşit ve $\sqrt{P}$ olan dilimlere ayrılan düzlemde, N tane dikdörtgen sıralanıp bu dilimlere yerleştirilecektir.

İlk olarak dikdörtgenler merkezi noktalarının x koordinatına göre sıralanır ve S tane dikey dilim oluşturulur. Son dilim hariç her bir dilim, sıralı listeden SxM tane dikdörtgen içerir. Son dilimin eleman sayısı SxM'den az olabilir. Dikey dilimleri

belirledikten sonra y koordinatına göre sıralama yapılır ve M tane dikdörtgen barındıran gruplara ayrılır. Bu aşamada yoğunlaştırılmış (sıkıştırılmış) R-ağacının yaprakları açığa çıkar. Dikdörtgenler ön işlemden geçirilip yaprakların içerikleri düzenlendikten sonra öz yinelemeli olarak yukarı seviyeler oluşturulur. Şekil 4.9'da daha önceki şekillerde de örneklenen hidrografik veri kümesi kullanılmıştır ve geleneksel R-ağacı ve yoğunlaştırma tekniğinin karşılaştırmalı sonuçları verilmiştir.



Şekil 4.9 : R-ağacı (a) ve STR ile yoğunlaştırlmış R-ağacı örnekleri

Doluluk oranı olabildiğince yüksek tutularak %100'e yaklaştırılan bu yapıda yer kullanımında verim en üst düzeye çekilmek istenmektedir. Fakat dezavantaj olarak öne sürülecek en önemli nokta dinamik ekleme ve silme işlemlerine doğrudan desteğinin zayıflığıdır. Yeni bir ekleme sonucunda doluluk oranları yüksek olacak şekilde yapılandırılmış bir ağaçta taşma olma olasılığı oldukça yüksektir. Bu gibi durumlara alternatif bir çözüm olarak doluluk oranları %100'den düşük seçilebilir veya yeni duruma göre yeniden yapılandırma tercih edilebilir. Bu yöntemde tamamen ortadan kaldırılamayan bir diğer durum ise dikdörtgenlerin örtüşmesidir. Mevcut yapısı ile durağan (statik) veri kümelerini hedefleyen uygulamalarda tercih edilebilecek bir çözüm sunan STR algoritması, ağaç derinliğini azalttığından ve ön işlemler sayesinde konumsal olarak birbiriyle alakalı nesneleri aynı yapraklarda daha iyi gruplamasından dolayı konumsal sorgularda kayda değer performans sağlamaktadır.

4.4 R+-ağacı

R+-ağacı, dizin dikdörtgenlerinin ara düğümlerde de tutulabildiği ve örtüşmelerine izin verilmediği bir R-ağacı yapısıdır. Örtüşmelerin az olduğu bir yapıda sorguların tepkileri daha hızlı olacaktır ve maliyetler düşecektir. Örneğin bir nokta sorgusunda kökten yapraklara doğru tek bir yol izlenerek sadece noktayı içeren nesnelerin olduğu yaprağa erişilir. Bu nesnelerin dizin dikdörtgeni bir başkasıyla tanım gereği örtüşmeyeceğinden tek bir yol izlenerek adaylar belirnebilir.

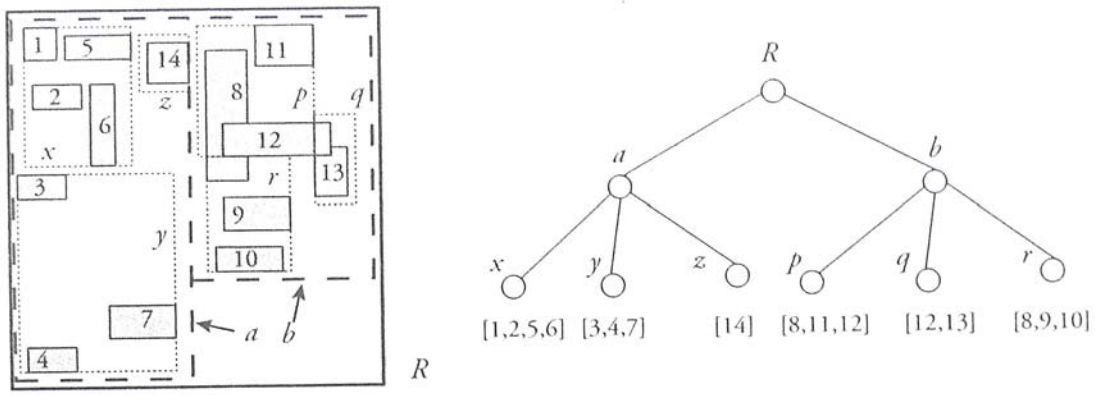
R+-ağacının sağlaması gereken özellikler [66]'da verilmiştir :

- Ara düğümlerdeki her bir (I,cp) kaydı için, R dikdörtgeni I tarafından tamamen örtülmek şartıyla, cp ile gösterilen bir düğümün alt ağacında yer almalıdır. Buna aykırı durum R'nin bir nesnenin sınırlayıcı kutusu (mbb) olmasıdır ki yaprakta yer alan bir kayıt anlamındadır. Bu durumda R, I ile kısmi olarak örtüşebilir.

- Herhangi iki ara düğümün dizin dikdörtgeninin örtüşmesi (kesişime sahip olması) söz konusu değildir.

- Düğümlerdeki kayıt sayıları için en az veya en çok olacak şekilde bir kısıt yoktur. Bu konudaki tek özel durum kökün kendisinin yaprak olmadığı durumda en az iki çocuğunun olması gerektiğidir. Bütün yapraklar aynı seviyede yer alır ve ağaç dengelidir. Bu denge ekleme ve silme algoritmaları tarafından sağlanmalıdır.

Şekil 4.10'da örnek bir R+-ağacı verilmiştir. 8 ve 12 numaralı nesnelerin tekrarlı kayıtlar olduğu görülmektedir. 8 numaralı nesne p ve r yapraklarıyla, 12 numaralı olan ise p ve q yapraklarıyla örtüşmektedir. Her iki nesne de aynı yaprak seviyesindedir fakat dizin dikdörtgenleri kesinlikle üst üste binmemektedir, kesişimleri yoktur, örtüşmemektedir.



Şekil 4.10 : R+-ağacı

R+-ağacı yapısal olarak R-ağacı ile aynı görünüme sahip olmasına rağmen doluluk oranı R-ağaçlarındaki gibi ön görülemediğinden ve nesne tekrarları yüzünden R+-ağacı aynı veri kümesini kullanan R-ağacından daha büyük olabilmektedir. R+-ağacının yapılandırması ve yönetimi diğer R-ağacı tiplerinden daha karmaşık ve zordur [33].

R+-ağacı yapılandırılırken veri ekleme sıradan bir R-ağacı ekleme süreci değildir. Dinamik eklemelere izin verilirken bir yandan da kendine özgü yoğunlaştırma (sıkıştırma) algoritması ile tüm düğümlerin olabildiğince doldurulmasını sağlar. Bu durum durağan veya sık değişmeyen veri kümeleri için elverişlidir. Ölü alanlar azaltılarak sorgulardaki yanlış adayların potansiyeli düşürülebilir. Alan kullanımı etkinliği düzenlenerek taşan düğümlerin bölünme sayıları indirgenebilir. R+-ağaçları tam kapasiteye kadar yoğunlaştırılırsa derinliği R-ağacından çok daha az olacaktır. Bu da arama performansını olumlu etkiler.

Genel bir değerlendirme yapmak gerekirse nokta sorgularında örtüşmeyen düğümler sayesinde daha az yol ile daha az yaprağa erişim yapılarak R-ağacından daha etkili sonuçlar alınabilir. Pencere sorguları için kazanç bu kadar kesin olarak tayin edilemeyebilir. Nesne tekrarlarının ağacı büyütmesi ve aday listesinin bu tekrarlaran kayıtlardan arındırılmasına yönelik ardıl işlemler buna sebep olmaktadır.

5. Yapılan Çalışma

SOBAG-105K040 Evliya Çelebi Coğrafi Bilgi Çekirdeği Projesi kapsamında, dizinleme yöntemlerinin karşılaştırılmasına yönelik olarak yapılan bu tez çalışmasında, dizinleme katmanında gereksinimlere en iyi cevap veren, genel ve verimli bir dizinleme yönteminin mevcut, kabul görmüş yöntemler incelenerek bulunmasına yönelik çalışmalar yapılmıştır.

Problem tanımı ve tez kapsamına dayanarak yapılan çalışmaların başında, çalışma planında da verildiği üzere test verilerinin elde edilmesi gelmektedir.

5.1 Veri Üretimi

Farklı veri dağılımlarının dizinleme yöntemlerinin performansına etkisini değerlendirebilmek adına sentetik veri üretimi yapılmaktadır. Üretilen veriler ilk etapta sınırlayıcı kutulardır.

Veri üretiminde temel olarak iki farklı yaklaşım benimsenmiştir. İlk olarak sabit bir istatistiksel dağılıma bağlı kalınarak veri uzayına dağıtılan sınırlayıcı kutular bulunmaktadır. Sabit ve standart dağılmlar sayesinde dağılımlara göre veri artışının dizinleme yöntemlerine etkisinin değelendirilmesi amaçlanmaktadır. Veri üretiminde kullanılan temel iki dağılım fonksiyonu *düzgün dağılım* ve *Gauss dağılımı*dır. Bu dağılımlara göre kutu üretimi yapılırken kullanılan parametreler anlamlarıyla birlikte Tablo 5.1'de verilmiştir.

Parametre	*Anlamı*
d	Kutu ebatlarının çarpanı (büyütme, küçültme değişkeni)
N	Üretilen sınırlayıcı kutu sayısı
Min_x	Veri düzlemindeki en küçük x koordinatı değeri
Min_y	Veri düzlemindeki en küçük y koordinatı değeri
Max_x	Veri düzlemindeki en büyük x koordinatı değeri
Max_y	Veri düzlemindeki en büyük y koordinatı değeri
VDF	Veri dağılım fonksiyonu

Tablo 5.1 : Sınırlayıcı Kutu Üretim Parametreleri

5.1.1 Düzgün Dağılım

En basit olasılık dağılımıdır. Bir değerin görülme olasılı, ait olduğu kümedeki diğer elemanlarınkiyle aynı ve sabittir. D, veri kümesi olmak üzere her bir elemanın olasılığı şöyledir :

$$P_i(x_i) = 1/\mid D_{i\mid} \mid$$

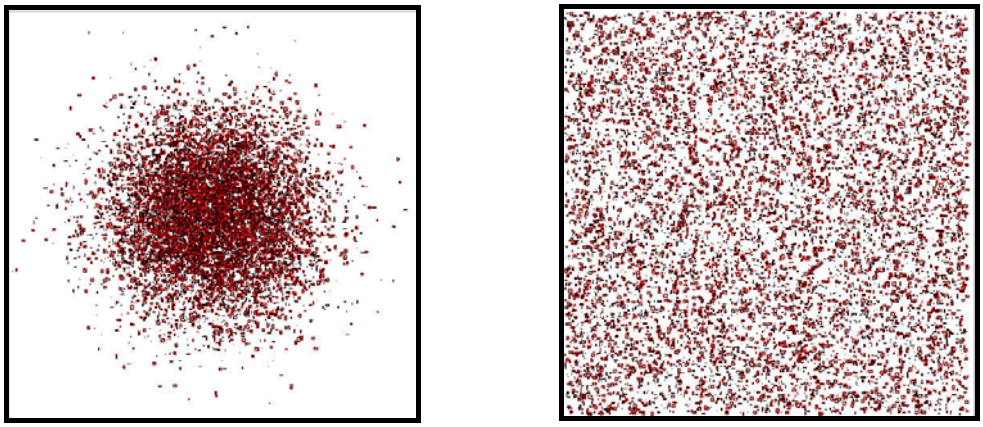
Düzgün dağılım genellikle sentetik olarak bilgisayar yardımıyla üretilen verilerin dağılımıdır. Günlük hayatta düzgün dağılıma sahip veriler az görülür. Teorik açıdan önemi bulunan bu dağılım tipi ispat ve analiz uygulamalarında basit ve standart yapısıyla yer edinir.

5.1.2 Gauss Dağılımı

Gerçek uygulamalarda oldukça sık görülen bir dağılımdır. Normal dağılım olarak da bilinen bu dağılımın, dağılım fonksiyonu aşağıdaki gibidir :

$$P_i(x_i) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

Gauss dağılımında veriler ortalama olarak μ çevresinde, σ standart sapmasıyla toplanır. Yani veriler μ merkezinde kümelenirler. Şekil 5.1'de düzgün dağılım ve Gauss dağılımı örnekleri verilmiştir.



(a) **(b)**

Şekil 5.1 : (a) Gauss dağılımı (b) Düzgün dağılım

5.1.3 İstatistiksel Dağılımlarla Üretilen Dikdörtgenler

Belirtilen istatistiksel dağılımlara göre dikdörtgenler üretilirken, Tablo 5.1'de verilen üretim parametrelerinin aldığı değerler oluşan veri kümelerine göre Tablo 5.2'de verilmiştir.

VDF	Gauss Dağılımı		Düzgün Dağılım	
N	20000	100000	20000	100000
Min_x , Min_y	0 , 0	0 , 0	0 , 0	0 , 0
Max_x , Max_y	500 , 500	500 , 500	500 , 500	500 , 500
d	0.01	0.01	0.01	0.01

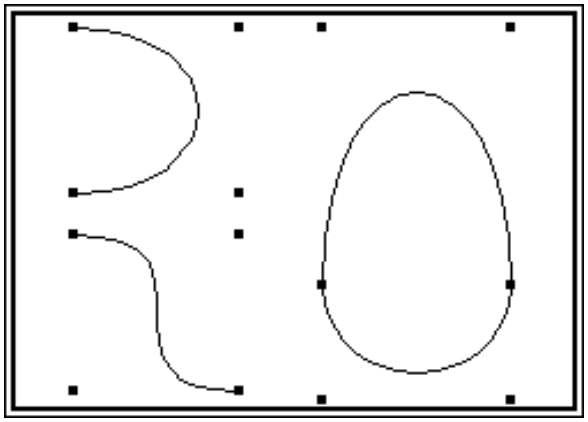
Tablo 5.2 : Kutu üretiminde kullanılan değerler

5.1.4 Bezier Eğrisi

Fransız mühendis Pierre Bezier tarafından araba gövdeleri tasarlamak amacıyla 1962'de kullanılmaya başlanan *Bezier eğrisi*, ilk olarak 1959'da Paul de Casteljau tarafından bulunmuştur fakat Bezier ismi ile özdeşleşmiştir [71].

Mühendislik ve matematik içeren birçok çalışmada faydalanılan Bezier eğrilerinin başlıca kullanım alanları olarak bilgisayar grafiği, bilgisayar destekli tasarım uygulamaları ve araba gövdesi tasarımları verilebilir. Bezier eğrilerinin daha yüksek boyutlardaki genellemeleri ile Bezier yüzeyleri oluşturulabilmekte ve 3-boyutlu sahneler yaratılabilmektedir.

Tanım olarak oldukça basit olan Bezier eğrisi, 4 noktanın bir fonksiyonudur, hesaplaması kolaydır. Bu gibi özelliklerinden dolayı birçok yazılım tarafından da desteklenir. Bezier eğrilerinin ilgi çekici bir diğer özelliği ise kendini kesebilmesidir [10]. Düzgün, pürüzsüz şekiller oluşturabilmek için ucuca eklenebilirler. Şekil 5.2'de iki tane Bezier eğrisi ve ucuca eklenen eğriler ile oluşturulmuş geometrik bir şekil örneklenmiştir.



Şekil 5.2 : Bezier eğrileri ve eğriler ile oluşturulmuş bir şekil

Bezier eğrisinin matematiksel tanımı kafa karıştırıcı gibi görünse de kısaca noktaların bir fonksiyonu olarak anılabilir. Bu noktalardan 2 tanesi bitim noktasıdır, diğer 2 tanesi ise kontrol noktasıdır. Eğri, bitim noktalarını birleştirir fakat kontrol

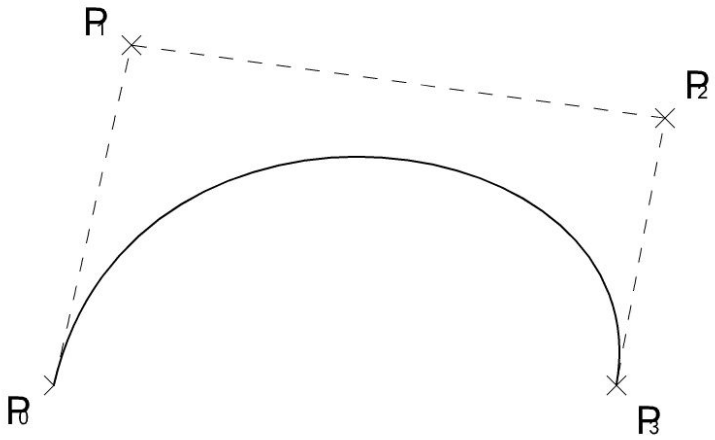
noktalarına temas etmek zorunda değildir. Bezier eğrisinin genel biçimi, eğri üzerindeki her bir noktayı gösteren şu fonksiyonla açıklanabilir :

$$B(t) = w_0(t)P_0 + w_1(t)P_1 + w_2(t)P_2 + w_3(t)P_3$$

P_0 ve P_3 bitim, P_1 ve P_2 kontrol noktaları, w_n ise ağırlık fonksiyonu olmak üzere tanımlı 4 nokta yardımıyla Bezier eğrisi oluşturulur. Daha açık bir ifadeyle şöyle verilebilir :

$$B(t) = P_0(1-t)^3 + 3P_1t(1-t)^2 + 2P_2t^2(1-t) + t^3P_3, t \in [0,1]$$

Örnek bir Bezier eğri Şekil 5.3'te verilmiştir. P_1 ve P_2 noktaları ile kontrol edilen eğri, P_0 noktasından P_3'e kadar uzanmaktadır.

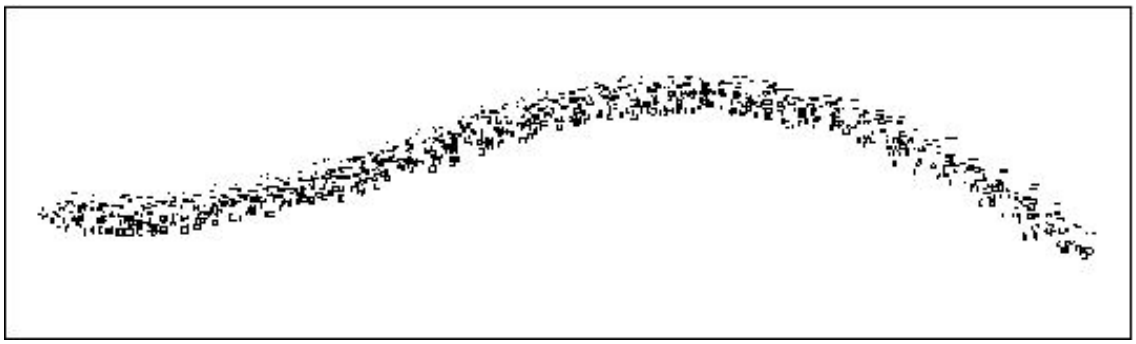


Şekil 5.3 : Bezier Eğrisi, bitim (P_0 ve P_3) ve kontrol (P_1 ve P_3) noktaları

5.1.5 Bezier Eğrileri Yardımı ile Veri Dağılımı

Standart veri dağılımlarının yanında rastsallığın da konumsal erişim yöntemlerine etkisini görebilmek ve istenen tarzda bir şehri veya coğrafi veri dağılımına sahip herhangi bir bölgeyi modelleyebilmek için Bezier eğrilerinden faydalanılmıştır. Bu sayede Bezier eğrileri ile veri uzayında istenen bölgelere ana hatlar çizilmektedir. İkinci aşama olarak ise bu hatlar üzerinde düzgün dağılıma

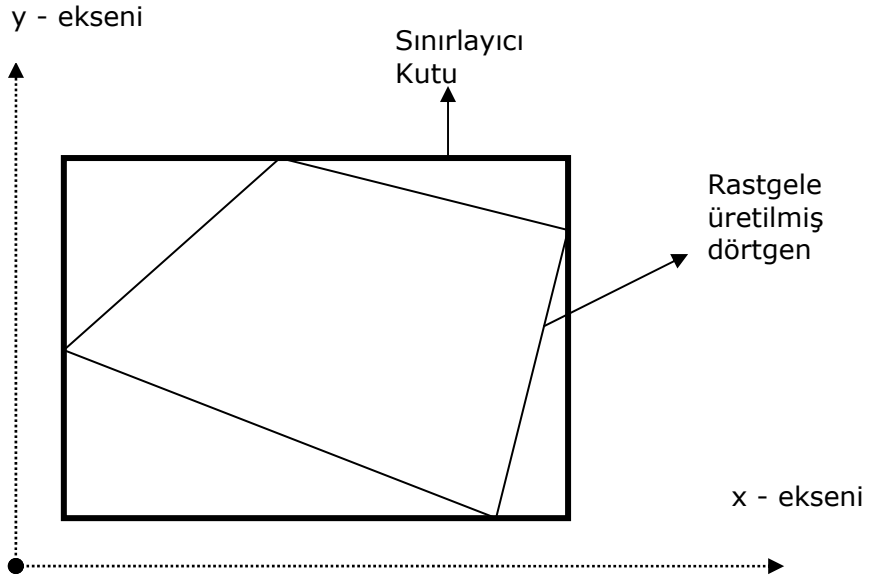
sahip dikdörtgenler oluşturulmaktadır. Şekil 5.4'te bir Bezier eğrisi ana hat olarak kullanılmıştır ve dikdörtgen dağılımında klavuz olarak görev yapmaktadır.



Şekil 5.4 : Bezier eğrisi boyunca dağılmış dikdörtgenler

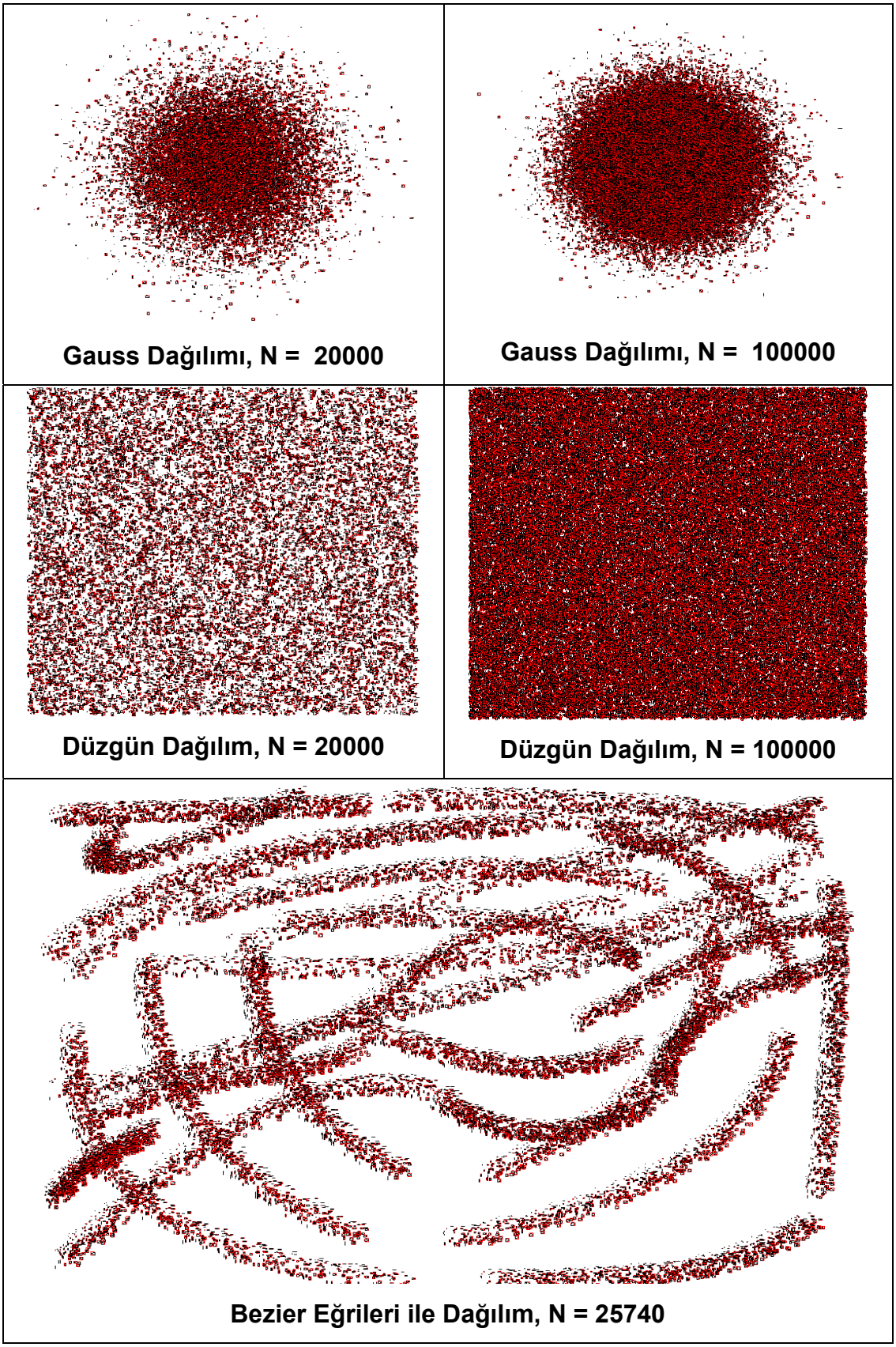
5.1.6 Sınırlayıcı Kutular ve Sınırlanan Geometrik Nesneler

Sınırlayıcı kutular üretildikten sonra, kutular içerisine her bir köşesi, sınırlayıcı kutunun farklı kenarına gelecek şekilde rastgele dörtgenler yerleştirilmektedir. Bu sayede istenen dağılıma sahip rastgele geometrik nesneler elde edilmektedir (Şekil 5.5).



Şekil 5.5 : Sınırlayıcı Kutu İçerisine Rastgele Dörtgen Yerleştirme

Bütün veri kümelerinin 2-boyutlu uzaydaki görünümleri Şekil 5.6'da verilmiştir.



Şekil 5.6 : Sentetik Veri Kümelerinin Kuşbakışı Görünümü

5.2 Deney Düzeneği

Deney düzeneğini oluşturmaya başlamadan önce daha önce yapılmış çalışmalar ve piyasada ticari veya geliştirme amaçlarıya bulunan uygulamaları kapsayan bir araştırma yapılmıştır. Yapılan genel taramalar neticesinde, mevcut uygulamalarda ve deneysel prototiplerde genel olarak R-ağaçları ve 4'lü ağaçların çeşitli türevlerinin kullanıldığı görülmüştür. Çok farklı ve karmaşık yöntemler olmasına karşın, diğerlerine göre gerçekleştirim kolaylığı ve makul performansları vasıtasıyla kabul gören R-ağaçları ve 4'lü ağaçlar bu alanda oldukça yaygındır.

Ticari uygulamalarda söz sahibi olan bir VTYS üretici olan ORACLE firmasının konumsal destek sunan çözümlerinde biri R-ağacı, diğeri 4'lü ağaç olmak üzere iki farklı konumsal dizin yapısı bulunmaktadır [56]. IBM DB2 Spatial Extender ise ızgara tabanlı yöntemleri kullanmaktadır [9]. Microsoft'un SQL Server 2005 ürününde kullandığı yol ise VTYS'de bulunan ilişkiler üzerine süzme adımını gerçekleştiren fonksiyonlar tanımlamaktan geçmektedir [36]. Aslında gerçek bir dizin olmadan dışardan bakıldığında dizin yapısı varmış gibi gösteren bu fonksiyonlar gerçek bir dizini canlandırmaktadırlar.

Açık kaynak kodlu bir VTYS olan MYSQL ise henüz "alpha" deneme sürümleri olarak bulunan konumsal desteğinde R-ağacını kullanmaktadır [64]. Yine açık kaynak kodlu bir VTYS olan PostgreSQL ise konumsal verilere sağladığı destek, R-ağacı tabanlı dizin yapısıyla ve deneme aşamalarını tamamlamış olmanın gücüyle önemli bir seçenek oluşturmaktadır.

5.2.1 VTYS : PostgreSQL

Temelleri 1977'de Berkeley üniversitesinde atılan PostgreSQL, ilişkisel modellere dayanan ve standart SQL sorgu dilini destekleyen bir VTYS'dir. Güncel sürümlerinde yeni nesil özellikler sunan PostgreSQL, nesne yönelimli tasarım ilkelerini de kapsayan yapısıyla genişleyebilme yeteneğine sahiptir. Bununla birlikte basit veri tiplerine ek olarak sonradan tanımlanabilen veri tiplerini kullanabilmeyi de sağlar. Bu sayede yeni nesil VTYS örneklerinin çoğunda görülen

nesnesel ve ilişkisel olma özelliklerini harmanlar. Ücretsiz ve açık kaynak kodludur. Özgürce kullanılabilir, değiştirilip geliştirilebilir. Ticari, akademik ve özel alanlarda güçlü özellikleri sayesinde rahatça kullanılabilir. UNIX türevi olan (Linux, FreeBSD) birçok işletim sistemi veya Windows ortamında kullanılabilir. Platformdan bağımsız olma özelliğini Solaris, SunOS, BeOS, NetBSD, MacOS gibi az bilinir ortamlarda da sürdürebilir. Teknik açıdan bakıldığında ODBC, JDBC, C, C++, PHP, Perl, TCL, Python bağlantıları için doğal arayüzler sunar.

5.2.1.1 PostGIS ile Konumsal Destek

Veri bütünleştirme ve özel yazılım geliştirme konularında özelleşmiş Refractions Araştırma Şirketi tarafından geliştirilen PostGIS, konumsal veritabanı teknolojileri ile ilgili açık kaynak kodlu bir projedir.

PostgreSQL VTYS ile beraber kullanılmak üzere geliştirilen PostGIS eklentisi mevcut VTYS'ye coğrafi nesneler için konumsal destek vermektedir. Bir başka deyişle PostGIS sayesinde, PostgreSQL, coğrafi bilgi sistemleri için güçlü bir VTYS adayı olmaktadır. PostGIS OGC tanımlarına ve kısıtlamalarına uygundur [52].

5.2.1.2 Konumsal Nesneler

OGC standartlarına göre konumsal nesnelerin gösterimi için 2 yol vardır :

- WKT (Well-Known Text) : *Düzgün biçimli metin* olarak çevrilebilecek bu kavram konumsal nesnelerin metinsel bir biçimde gösterimidir.

- WKB (Well-Known Binary) : *Düzgün biçimli ikili kod* olarak çevrilebilecek bu kavram konumsal nesnelerin metinsel gösterimlerinin ikili halde kodlanmış karşılığıdır.

Aşağıda WKT örnekleri ile birlikte konumsal nesneleri 2 boyutta modellemek için kullanılan temel geometrik kavramlar verilmiştir.

- Nokta :
  - POINT (1 1)
- Doğru parçası dizisi :
  - LINESTRING (0 0,1 1,2 2)
- Poligon :
  - POLYGON ((0 0,4 0,4 4,0 4,0 0),(1 1,2 1,2 2,1 2,1 1))
- Nokta çoklusu :
  - MULTIPOINT(0 0,1 2)
- Doğru parçası dizisi çoklusu :
  - MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
- Poligon çoklusu :
  - MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
- Geometrik nesne topluluğu :
  - GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))

OGC tarafından tanımlanan, WKT'ler ile ilgili Backus-Naur dil yapısı [50]'de verilmiştir. WKT ile WKB arasındaki ilişkiyi bir örnekle ele alalım. Örneğin bir düzlemde (1,1) noktasının WKB ile gösterimi şöyledir :

01010000000000000000F03F000000000000F03F

Sekizli Derecesi	:	01
WKB Tipi	:	01000000
X	:	000000000000F03F
Y	:	000000000000F03F

Sekizli derecesi 0 veya 1 olabilir. NDR (Network Data Representation) veya EDR (External Data Representation) olmak üzere verinin gösterim amacına yönelik bilgi verir. İkili kodlar halinde veri akışı (streaming) söz konusu olduğunda önem kazanmaktadır. WKB tipi ise geometrik tipin kodudur. 1'den 7'ye kadar

değerler alır ve sırasıyla Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, GeometryCollection anlamına gelir. X ve Y değerleri ise örneğimizde kullanılan noktanın hassas kayan noktalı koordinat değerleridir.

Konumsal nesnelerin iki farklı gösterimleri arasında çevrim yapan ve bu gösterimlerden geometrik nesne oluşturan fonksiyonlar mevcuttur. Bunlar geometrik tiplere göre özelleşen çeşitler gösterse de genel olarak şöyle verilebilirler :

bytea WKB = asBinary (geometry)
text WKT = asText (geometry)
geometry = GeomFromWKB (bytea WKB, SRID)
geometry = GeomFromText (text WKT, SRID)

OGC standartları gereği konumsal veri gösterimlerinde, saklama biçimlerinde konumsal kaynak sistemi tanımlayıcısı (SRID: Spatial Referencing System Identifier) bulunmalıdır. Bu SRID bilgisi, konumsal bir nesne yaratılırken, veritabanına eklenirken gereklidir.

5.2.1.3 Konumsal Dizinleme

PostGIS eklentisi ile bütünleştikten sonra konumsal veri tipleri ve gösterimleri bakımından OGC standartlarına uygun bir şekilde hizmet veren PostgreSQL, Open GIS'in de dizinleme ile ilgili bir standart ortaya koymamasından dolayı standart bir dizinleme yöntemine sahip değildir. Her ne kadar kendi bünyesinde uygulanmış R-ağacı tabanlı bir dizin yapısı barındırsa da diğer yöntemlerin sunabileceği farklılıklar bu noktada araştırma yapma isteğini körüklemektedir.

Genelleştirilmiş Arama Ağaçları (Generalized Search Tree, GiST) ile genel bir dizinleme biçimi tanımlanmıştır. Bu yapı üzerine kurulan özel amaçlı dizinler vasıtasıyla tamsayı dizileri, çok boyutlu veriler gibi düzensiz veriler

dizinlenebilmektedir. GiST aracılığıyla sağlanan ve R-ağacının bir uygulaması olan bir dizin yapısıyla konumsal dizinleme desteği sunulmaktadır.

Konumsal veri tabloları üzerinde, konumsal veri barındıran kolonları hedef alarak tanımlanan bu dizin yapısı ile konumsal sorgularda süzme adımları gerçekleştirilir. Sorgularda sınırlayıcı kutuları kullanarak dizin yapısından faydalanabilmek için “&&” işleci kullanılır. Bu sayede iki nesnenin sınırlayıcı kutularının kesişimlerine bakılır. Örneğin :

```
SELECT id, AsText(geom) AS geom
FROM deneme_gtest
WHERE geom && 'POLYGON((0 0,0 10,10 10,10 0,0 0))' AND
Contains('POLYGON((0 0,0 10,10 10,10 0,0 0))', geom );
```

Bu örnek sorgunun açılımı şudur :

1. İlk önce sorguda parametre olarak verilen poligon nesnesi ile veritabanındaki kayıtların sınırlayıcı kutuları kontrol edilir.
2. Nesne eğer kutu testinden geçerse, bu poligonu içerip içermediğine bakılır.

5.2.2 Geliştirme Ortamı : Microsoft .Net 2005

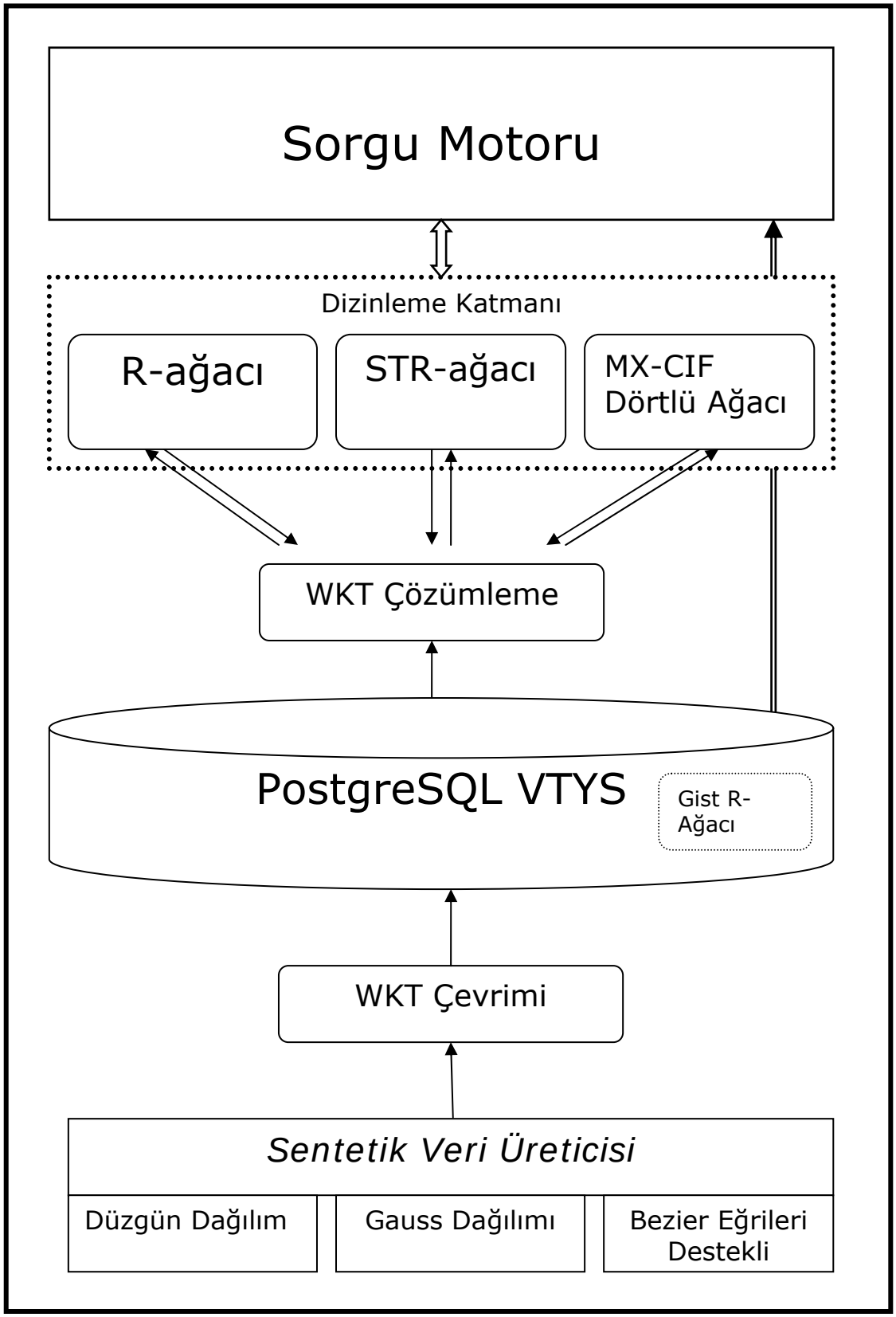
SOBAG-105K040 Evliya Çelebi Coğrafi Bilgi Çekirdeği Projesi Microsoft .Net teknolojileri üzerinde yol aldığından ve [46]’de detaylarıyla açıklanan geometri kütüphanesi Microsoft .Net 2005 geliştirme ortamında C# dili kullanılarak yazıldığından dolayı herhangi bir farklılığa sebep vermemek amacıyla dizinleme katmanında kullanılacak yöntemler de aynı geliştirme ortamında gerçekleştirilip denenmiştir.

5.2.3 Dizinleme Yöntemleri Seçimi

Problem tanımı doğrultusunda temel amaç farklı dizinleme yöntemlerinin birbirlerine göre başarımlarını karşılaştırmak ve sistem gereksinimlerini diğerlerine göre daha iyi karşılayabilecek olanın belirlenmesi olduğundan öncelikle temelden farklılaşan yöntemler seçerek bu farkların genel başarıma etkisi incelenmeliydi. Buna yönelik olarak konumsal dizinleme yöntemlerinin temel iki farkına dayanarak bir tane uzay güdümlü bir tane de veri güdümlü olmak üzere iki yöntem, yapılan taramalar sonucunda belirlendi. Bunlar MX-CIF 4'lü ağacı ve R-ağacı idi. Dinamik ortamlarda başarılı olan bu yöntemlere, durağan ortamlar söz konusu olduğunda rakip olabilecek bir yöntem olarak ise yine bir R-ağacı tipi olan fakat veri yoğunlaştırması sayesinde arama başarımı artan STR-ağacı belirlendi.

5.3 Deney Düzeneği Yapısı

Dizinleme katmanı olarak VTYS üzerine yerleşen yapıda yer alan yöntemler ile PostGIS eklentisiyle PostgreSQL VTYS'nin kendi bünyesinde yer alan Gist R-ağacının tepkilerini karşılaştırabilmek için, VTYS ile bütünleşik olan dizin yapısı da aynı testlere tabi tutulacaktır. Şekil 5.7'de verilen yapıya göre aşağıdan yukarıya doğru bir akış izlenerek deney düzeneği takip edilebilir. Bununla birlikte çalışmanın sonunda bulunan ekte Kolay Belgele s.2.0 ile verilen sistem yapısı ve belgelendirme çalışması bulunmaktadır.



Şekil 5.7 : Deney Düzeneği Yapısı

İlk olarak düzgün dağılım, Gauss dağılımı ve Bezier eğrileri yardımıyla veri üretimi gerçekleştirildikten sonra bu geometrik veriler OGC standartlarında tarif ediliği üzere düzgün biçimli metinler (WKT) halinde veritabanına girilir. Fakat içsel olarak veritabanında ikili kodlar halinde tutulur. Veritabanında tutulan her nesnenin ayırt edici bir numarası vardır. Nesnelerle ilgili kayıtlar Şekil 5.8'de gösterilen düzende tutulmaktadır :

Nesne Anahtarı	Tanımlayıcı Metin	Geometrik Veri

Şekil 5.8 : Geometrik Nesne Kayıt biçimi

Bu biçimde örnek bir kayıt şöyledir :

99132;"Object_99132";
"POLYGON((297.12894244449 285.565929805335,
298.304011110981 285.569749349621,
298.600992210464 285.566240082123,
297.251050145736 285.563092980532,
297.12894244449 285.565929805335))"

Aynı kayıt için düzgün biçimli ikili kodlu görünüm ise şu şekildedir :

99132;"Object_99132";
"0103000000010000000500000006AE2F32510927240ED5B690C0ED9714
08B34C13ADDA47240A2787EB11DD97140801102AA9DA9724071A6C25
10FD971407959284D04947240E0FFC86D02D971406AE2F32510927240
ED5B690C0ED97140"

Dizin yapıları kurulurken geometrik nesnelerin sınırlayıcı kutularına ihtiyaç duyulduğundan, dizinleme katmanıyla VTYS arasında düzgün biçimli metinleri çözümleyen bir mekanizma bulunmaktadır. Bu çözümleyici sayesinde geometrik nesnelerin sınırlayıcı kutuları hesaplanıp dizin yapıları kurulmaktadır. Dizin yapısı içerisinde verilerin nesne anahtarları ve sınırlayıcı kutuları tutulmaktadır (Şekil 5.9).

Nesne Anahtarı	Sınırlayıcı Kutu

Şekil 5.9 : Dizin Kayıt biçimi

Şekil 5.7'de verilen genel yapıyı özetlemek gerekirse; VTYS'ye aktarılan verilere sıralı erişimden daha yüksek bir başarımla erişmeyi sağlayan ve dizin yapısını kurarken verilen konumsal özelliklerini dikkate alan dizinleme yöntemleri, VTYS'nin üzerinde sorguyu karşılayarak, süzme adımının gerçekleştirilmesine olanak sağlamaktadır. Süzme adımından sonra oluşan aday kümesi, karmaşık geometrik işlemlerin yapılması için yeniden ele alınacaktır ki bu aşamada artık dizinleme katmanı görev almamaktadır. Görevini tamamlayan dizinleme katmanı, aday kümesini sorgu katmanına aktarır. Bundan sonra yapılacak karmaşık geometrik işlemler geometri ve grafik kütüphanesine ihtiyaç duyar.

5.4 Test Senaryoları

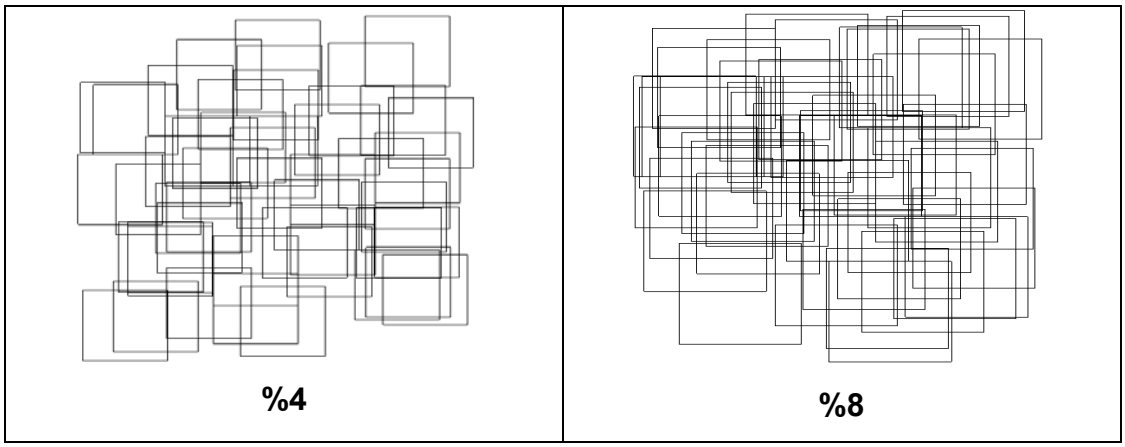
Dizinleme yöntemlerini birbirleriyle karşılaştırabilmek için ortak bir test senaryosu gerekmektedir. Üretilen sentetik veriler, dizin yapıları ve Şekil 5.6'daki yapı çerçevesinde veriler üzerinde pencere sorguları yapılacaktır. Karşılaştırmalarda esas olarak sorgu işleme sürecinde birincil süzgeçten (süzme adımı) dönen aday küme boyutları ve bunun için gereken zamanlar kıyaslanacaktır.

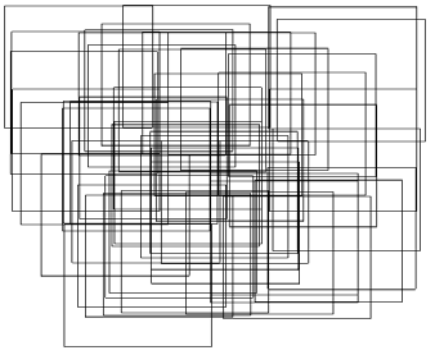
Veriler 2 boyutlu uzayda bir düzlem üzerinde dağılmaktadır. İstatistiksel dağılımlara göre üretilen veriler 500 x 500 ebatlarında bir düzlemde bulunmakatadır. Bezier eğrileri yardımıyla üretilen veriler ise 1024 x 768 ebatlarında bir düzlemdedir. Belirtilen veri uzaylarında sorgularda kullanılmak üzere 25 farklı ve her boyuttan da 50 adet farklı pencere düzgün dağılıma sahip olacak şekilde üretilmektedir. 10, 20 gibi daha düşük pencere sayılarıyla elde edilen ortalamalar, özellikle Gauss dağılımına sahip veri kümelerinden alınan sonuçlarda ciddi sapmalara yol açmıştır. Pencere sayısındaki artış, pencere dağılımını daha iyi hale getirerek ortalama değerlerdeki sapmaları gidermiştir. Bir başka deyişle rastgele dağıtılan pencerelerle yapılan sorguların ortalaması alınarak veri yoğunluğunun yer yer az veya çok olmasından kaynaklanan

sapmalar yumuşatılmaktadır. Böylece her bir dizinleme yönteminin ortalama başarımı karşılaştırılabilir noktaya gelmektedir. Tablo 5.3'te ve sonrasındaki Şekil 5.10'de bu değerlerin ne anlama geldiği açıklanmıştır.

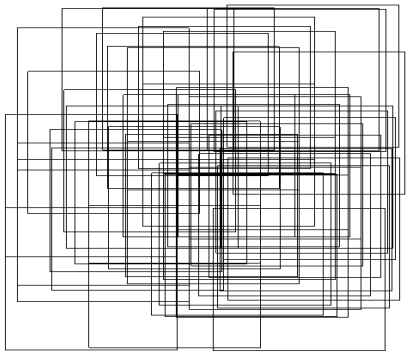
Pencere Büyüklük Numarası (PBN)	Pencere Büyüklük Yüzdesi	Pencere Sayısı
1	%4	50
2	%8	50
3	%12	50
4	%16	50
5	%20	50
6	%24	50
7	%28	50
8	%32	50
9	%36	50
10	%40	50
11	%44	50
12	%48	50
13	%52	50
14	%56	50
15	%60	50
16	%64	50
17	%68	50
18	%72	50
19	%76	50
20	%80	50
21	%84	50
22	%88	50
23	%92	50
24	%96	50
25	%100	50

Tablo 5.3 : Pencere üretimi senaryo tablosu

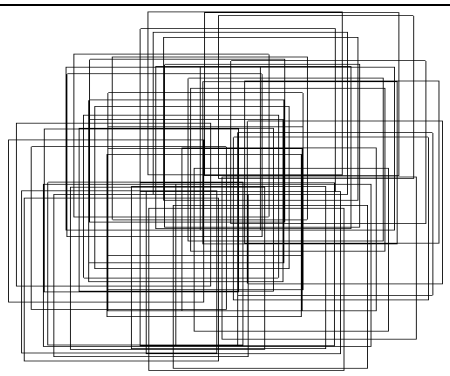




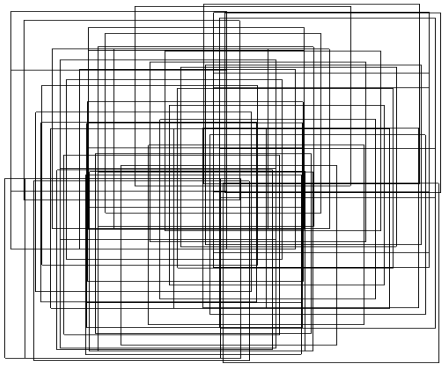
%12



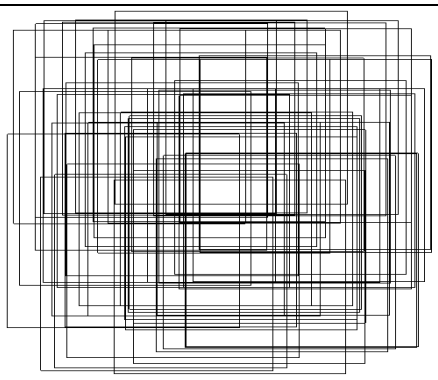
%16



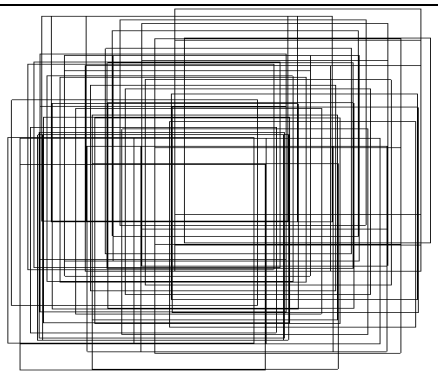
%20



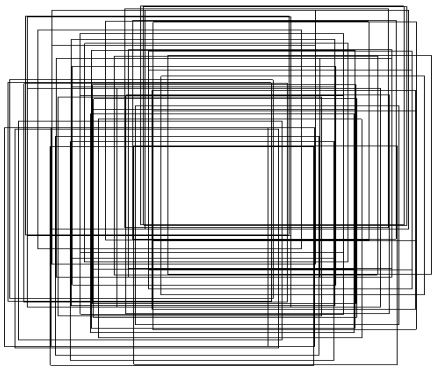
%24



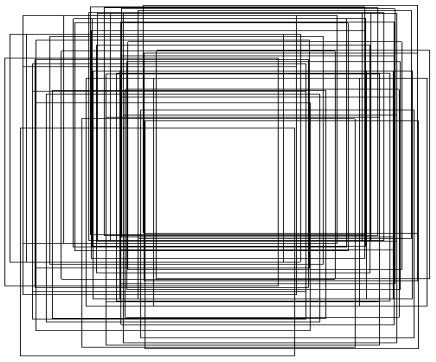
%28



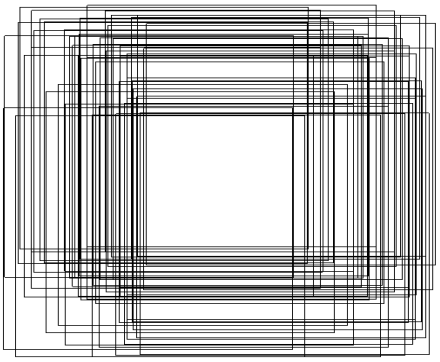
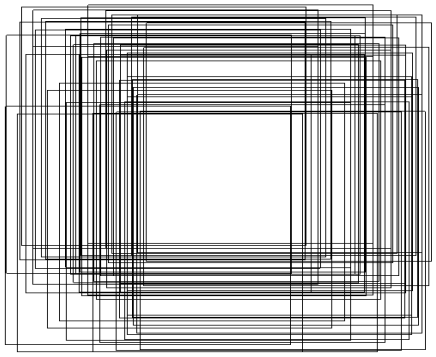
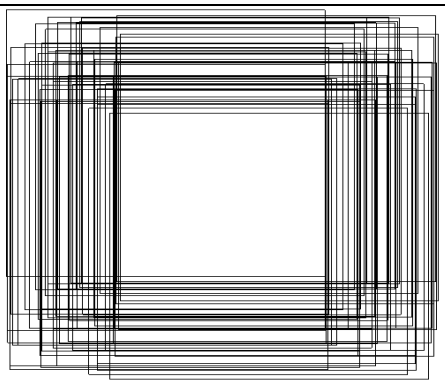
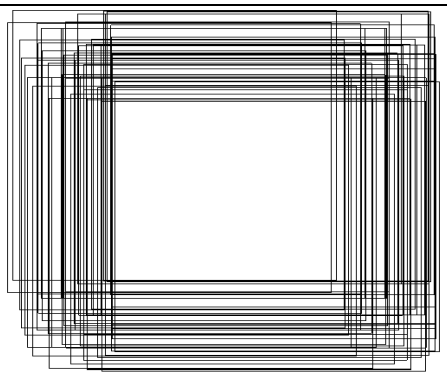
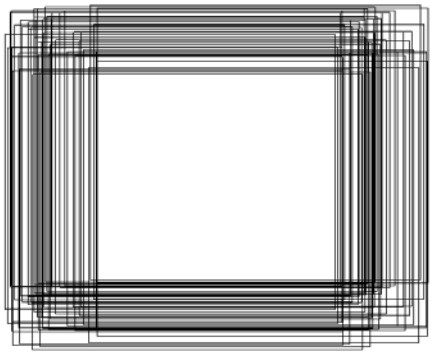
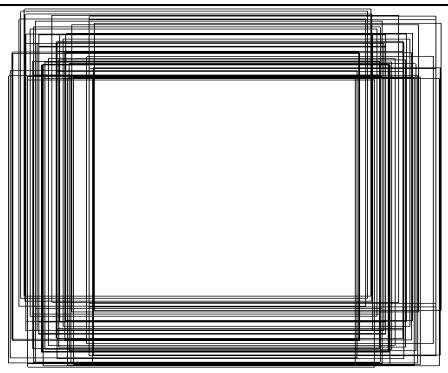
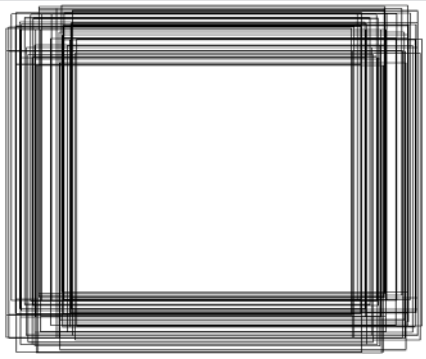
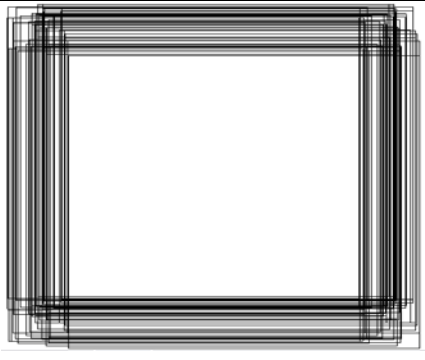
%32

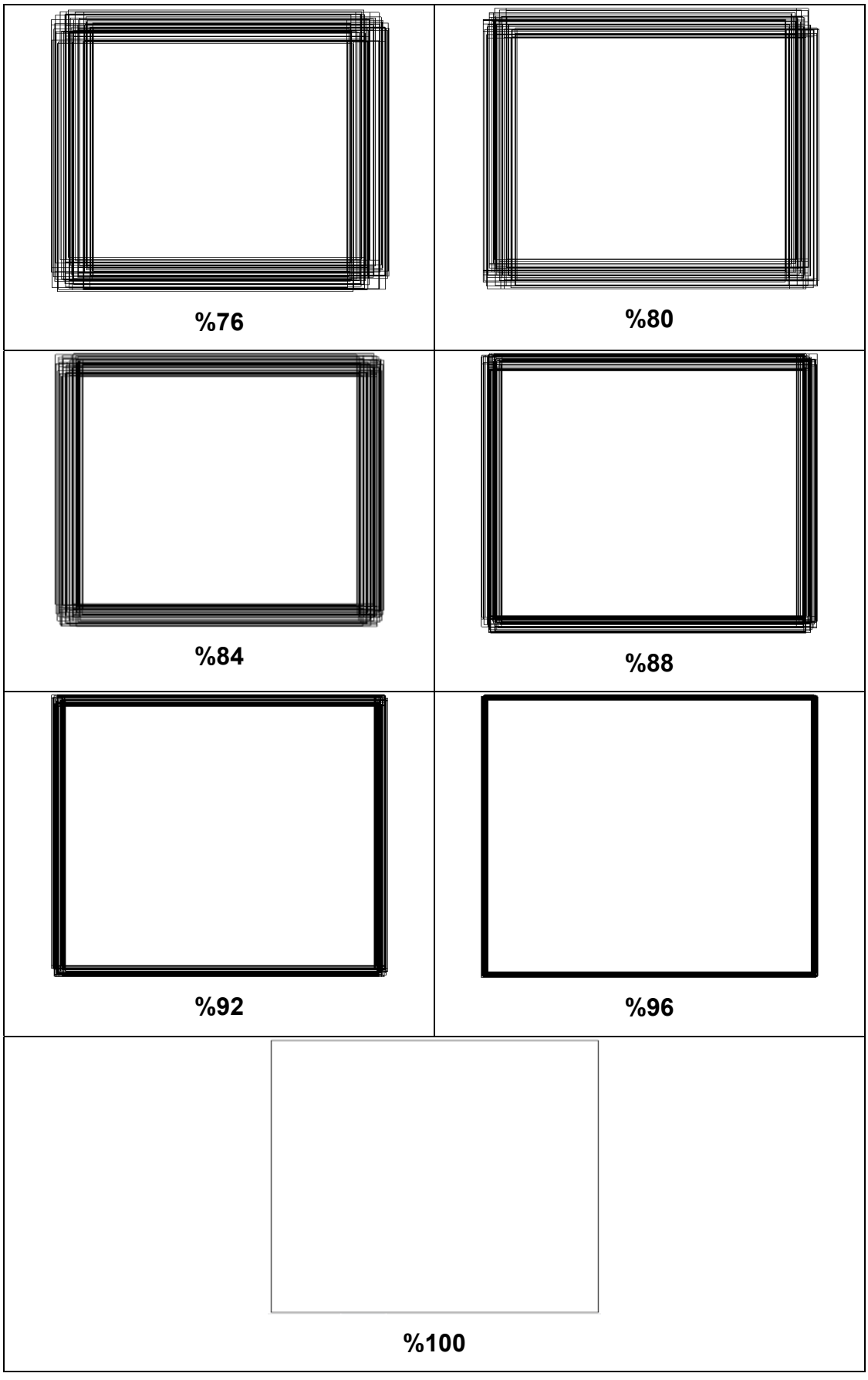


%36



%40

 %44	 %48
 %52	 %56
 %60	 %64
 %68	 %72



Şekil 5.10 : Sorgu Penceresi Büyüklükleri

Şekil 5.10'da 500 x 500 ebatlarında bir düzlemde yerleştirilen sorgu pencereleri örneklenmiştir. Bu pencereler uygulanan senaryolarda dinamik olarak yaratılır ve rastgele yerleştirilirler. Bezier eğrileri yardımıyla üretilen veriler ise 1024 x 768 ebatlarında bir alana dağılmaktadırlar. Fakat sorgu pencereleri dağıtılırken uygulamada herhangi bir değişiklik bulunmamaktadır. Tek fark pencere üretim sürecinde kullanılan veri dağılım alanının büyüklüğüdür.

6. Sonuçlar

Ortaya konan problem gereği yapılan çalışmada alınan sonuçlara geçmeden önce, kurulan deney düzeneğinin nasıl bir ortamda işletildiğini teknik açıdan da ele almak faydalı olacaktır.

Çalışmada kullanılan, VTYS için PostgreSQL ve geliştirme ortamı için de Microsoft .Net 2005 başta olmak üzere bütün araçlar, özellikleri aşağıda belirtilen sistemde çalıştırılmıştır.

İşlemci	:	Intel Pentium M 1,60 Ghz, 2 MB önbellek
Ana Hafıza	:	512 MB DDR PC2700, 333 Mhz
Sabit Disk	:	40 GB 4200 RPM, 8 MB önbellek
İşletim Sistemi	:	Windows XP Service Pack 2

Sonuçların yorumlanması açısından önemle belirtilmesi gereken bir diğer nokta ise dizin yapılarının ana hafızada tutuluyor olduğudur. Test senaryoları uygulanmadan hemen önce veritabanındaki veriler doğrultusunda dinamik olarak yüklenen dizinler VTYS üzerinde dizinleme katmanı oluşturmaktadır. Sorguları ilk karşılayan yapı olan dizinler, aday kümesinin oluşturulmasında birincil süzgeç olarak görev yaparlar.

Sonuç grafiklerinde sorgu başarımı kıyaslaması yapılırken esas alınan alınan temel iki nokta aday kümesi oluşum zamanı (*milisaniye*) ve aday kümelerinin eleman sayılarıdır. Teorik olarak bir kez daha vurgulamak gerekirse aday kümesini oluştururken yapılan süzme işleminin başarısı aday kümesini ne kadar daralttığına bağlıdır. Böylece karmaşık, maliyetli geometrik işlemler daha az uygulanacaktır.

Bütün testlerde R-ağacı ve STR-ağacının dallanma faktörü 10 (on) olarak sabit tutulmuştur. Daha sözel bir ifadeyle, her bir düğümün 10 adet çocuğu olabilmektedir.

Sonuçlara ilişkin verilen grafikler ve tablolarda, veri uzayına göre 25 farklı oranda üretilen 50 pencerenin ortalaması alınmıştır. Sorgu zamanlarının ölçümleri *milisaniye (ms)* olarak verilmiştir.

6.1 Gauss Dağılımı Sonuçları

Veri Kümesi Özellikleri :

Adı : GD20

Dağılım Tipi : Gauss Dağılımı

Veri Sayısı : 20.000

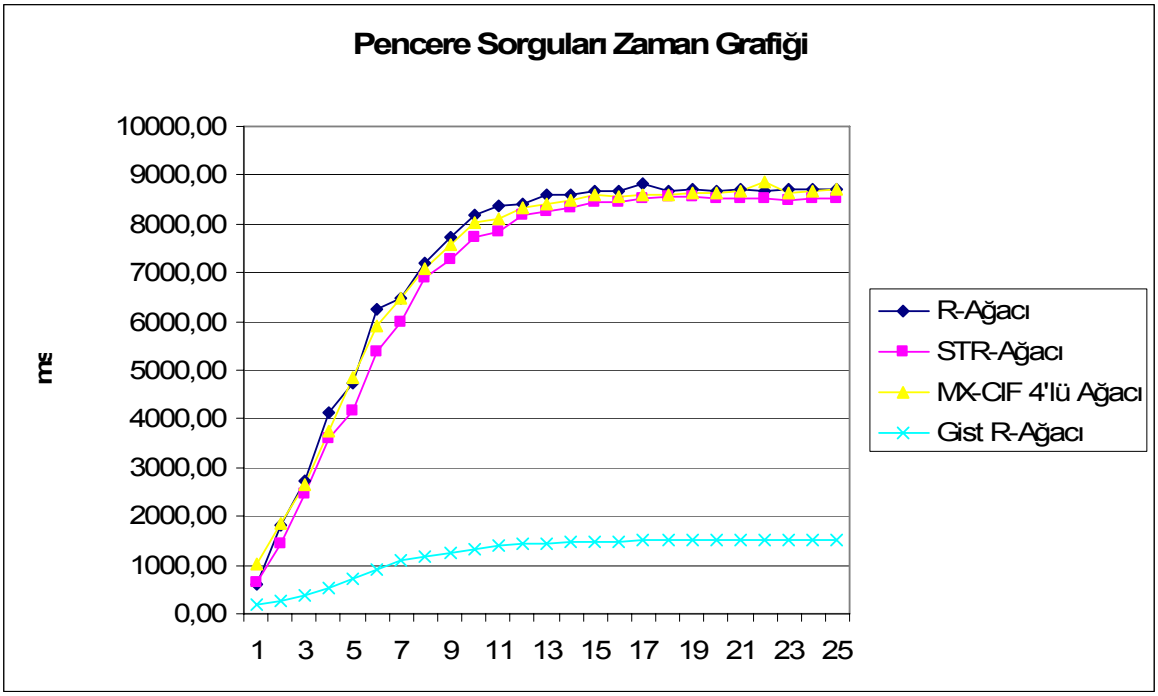
	R-Ağacı	STR-Ağacı	MXCIF 4'lü Ağacı
Derinlik	*5*	*5*	*18*
Oluşum Zamanı (ms)	*6298,13*	*6458,75*	*7887,50*

Tablo 6.1 : GD20, Dizin Oluşum Bilgileri

	Pencere Sorgusu Zamanları (Ortalama, ms)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	589,06	654,38	1004,38	175,63
PBN 2	1812,19	1452,81	1850,94	278,44
PBN 3	2722,50	2459,38	2650,00	394,69
PBN 4	4127,81	3605,00	3732,19	521,25
PBN 5	4736,25	4150,63	4860,63	720,00
PBN 6	6266,56	5360,63	5894,69	906,56
PBN 7	6488,44	5996,25	6481,88	1099,38
PBN 8	7207,81	6880,31	7087,81	1189,69
PBN 9	7714,38	7285,63	7589,38	1240,94
PBN 10	8164,38	7712,81	8013,44	1337,81
PBN 11	8387,81	7850,94	8123,75	1404,06
PBN 12	8426,25	8173,13	8318,75	1434,38

PBN 13	8597,19	8260,94	8425,31	1457,81
PBN 14	8603,75	8339,38	8497,81	1475,31
PBN 15	8683,75	8428,13	8586,56	1489,69
PBN 16	8682,81	8456,56	8545,31	1493,13
PBN 17	8827,50	8517,81	8614,69	1503,13
PBN 18	8692,50	8544,06	8615,00	1505,31
PBN 19	8723,75	8547,19	8632,81	1508,13
PBN 20	8690,94	8526,56	8636,56	1506,88
PBN 21	8705,00	8510,31	8679,06	1499,06
PBN 22	8692,19	8535,31	8877,50	1507,81
PBN 23	8704,38	8476,25	8649,06	1500,31
PBN 24	8705,94	8513,75	8670,31	1501,88
PBN 25	8711,56	8540,31	8698,75	1507,81

Tablo 6.2 : GD20, Pencere Sorguları Zaman Ortalamaları

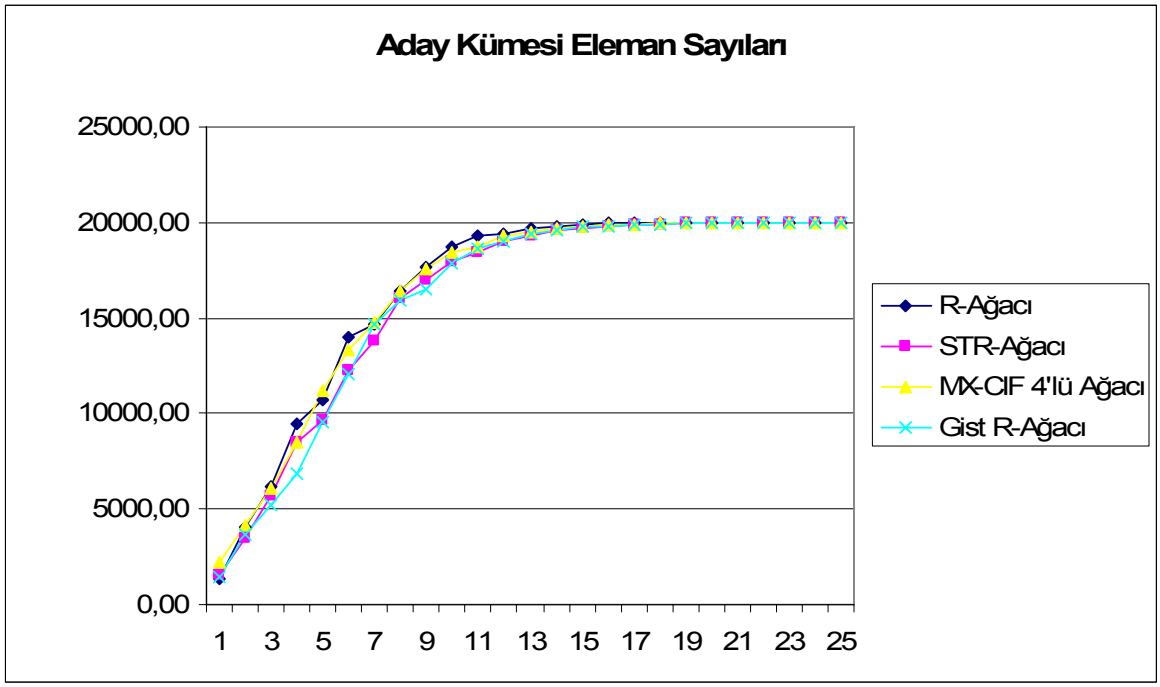


Şekil 6.1 : GD20, Pencere Sorguları Zaman Grafiği

	Aday Kümesi Eleman Sayıları (Ortalama)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	1303,14	1496,76	2264,28	1491,10
PBN 2	4018,56	3429,84	4178,50	3693,34
PBN 3	6167,68	5706,88	6107,52	5247,62
PBN 4	9493,52	8493,98	8491,60	6898,74
PBN 5	10756,38	9680,56	11225,16	9597,06

PBN 6	13962,18	12305,42	13317,78	12050,08
PBN 7	14699,88	13773,64	14786,44	14652,00
PBN 8	16437,60	16060,90	16420,82	15878,76
PBN 9	17684,46	17020,60	17526,32	16509,24
PBN 10	18683,54	17995,36	18393,92	17850,36
PBN 11	19277,06	18407,60	18764,78	18638,68
PBN 12	19361,26	19036,70	19286,00	18992,48
PBN 13	19717,76	19326,24	19483,96	19354,86
PBN 14	19825,14	19573,92	19721,50	19555,40
PBN 15	19916,54	19696,30	19799,40	19748,74
PBN 16	19963,66	19828,40	19859,10	19810,64
PBN 17	19986,38	19884,98	19922,82	19896,62
PBN 18	19994,36	19923,44	19942,76	19925,66
PBN 19	19997,40	19951,00	19965,66	19950,42
PBN 20	19999,16	19966,58	19976,92	19966,30
PBN 21	20000,00	19977,92	19984,42	19978,54
PBN 22	20000,00	19986,68	19992,08	19986,32
PBN 23	20000,00	19993,00	19996,28	19992,70
PBN 24	20000,00	19997,68	19998,58	19997,24
PBN 25	20000,00	20000,00	20000,00	20000,00

Tablo 6.3 : GD20, Aday Kümesi Eleman Sayısı Ortalamaları



Şekil 6.2 : GD20, Aday Kümesi Ortalama Eleman Sayısı Grafiği

Veri Kümesi Özellikleri :

Adı : GD100

Dağılım Tipi : Gauss Dağılımı

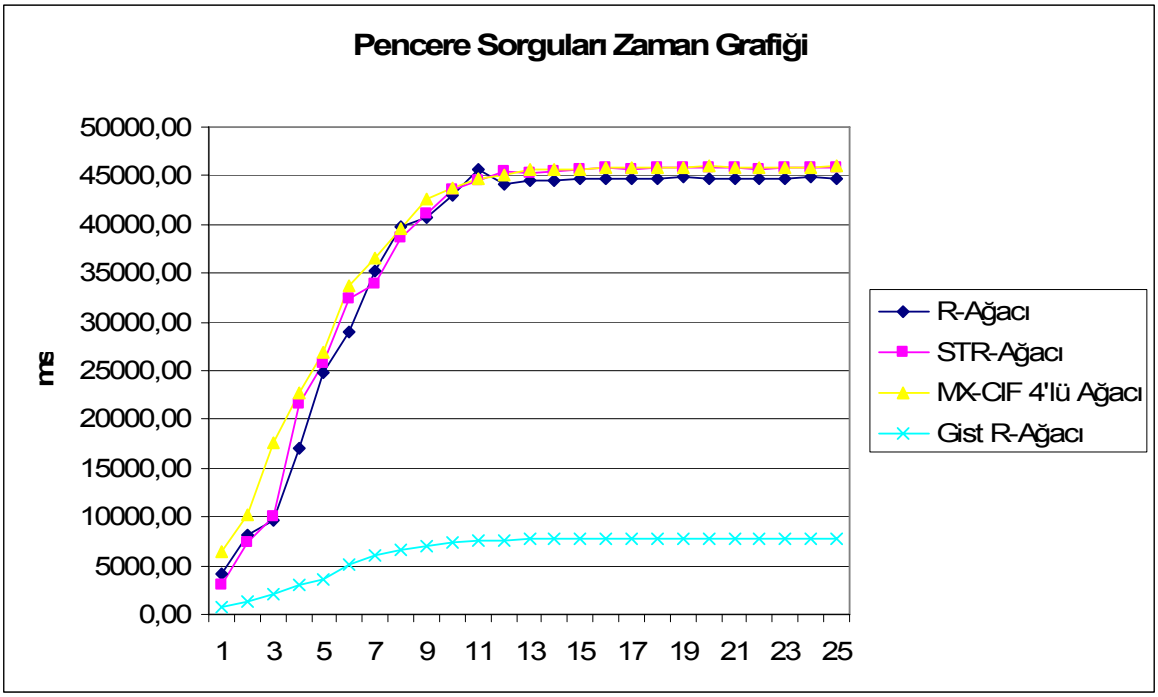
Veri Sayısı : 100.000

	R-Ağacı	STR-Ağacı	MXCIF 4'lü Ağacı
Derinlik	*6*	*6*	*23*
Oluşum Zamanı (ms)	*62991,25*	*32743,75*	*39384,38*

Tablo 6.4 : GD100, Dizin Oluşum Bilgileri

	Pencere Sorgusu Zamanları (Ortalama, ms)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	4095,94	3022,81	6505,00	691,25
PBN 2	8094,69	7336,25	10260,63	1246,25
PBN 3	9692,19	9998,44	17602,81	2122,19
PBN 4	17082,19	21655,63	22638,75	3095,94
PBN 5	24894,06	25745,94	26843,13	3524,06
PBN 6	29032,19	32400,94	33724,69	5115,31
PBN 7	35278,44	33846,88	36610,00	5967,50
PBN 8	39860,63	38710,00	39507,19	6575,94
PBN 9	40727,81	41098,75	42590,63	6965,00
PBN 10	42955,00	43538,75	43780,00	7330,00
PBN 11	45556,25	44447,50	44780,94	7483,75
PBN 12	44100,31	45395,94	45095,94	7600,63
PBN 13	44454,06	45334,38	45605,63	7702,19
PBN 14	44522,50	45395,00	45681,25	7676,88
PBN 15	44771,88	45682,50	45631,25	7734,69
PBN 16	44735,31	45741,88	45794,69	7714,69
PBN 17	44644,69	45662,19	45926,56	7709,38
PBN 18	44769,06	45866,87	45757,50	7728,13
PBN 19	44807,19	45834,69	45888,13	7782,19
PBN 20	44725,63	45808,13	45933,44	7726,25
PBN 21	44746,56	45819,06	45793,75	7764,38
PBN 22	44654,69	45681,88	45903,13	7766,56
PBN 23	44764,69	45887,50	45860,31	7773,44
PBN 24	44795,63	45848,44	45779,06	7744,38
PBN 25	44652,81	45739,69	45968,13	7739,38

Tablo 6.5 : GD100, Pencere Sorguları Zaman Ortalamaları

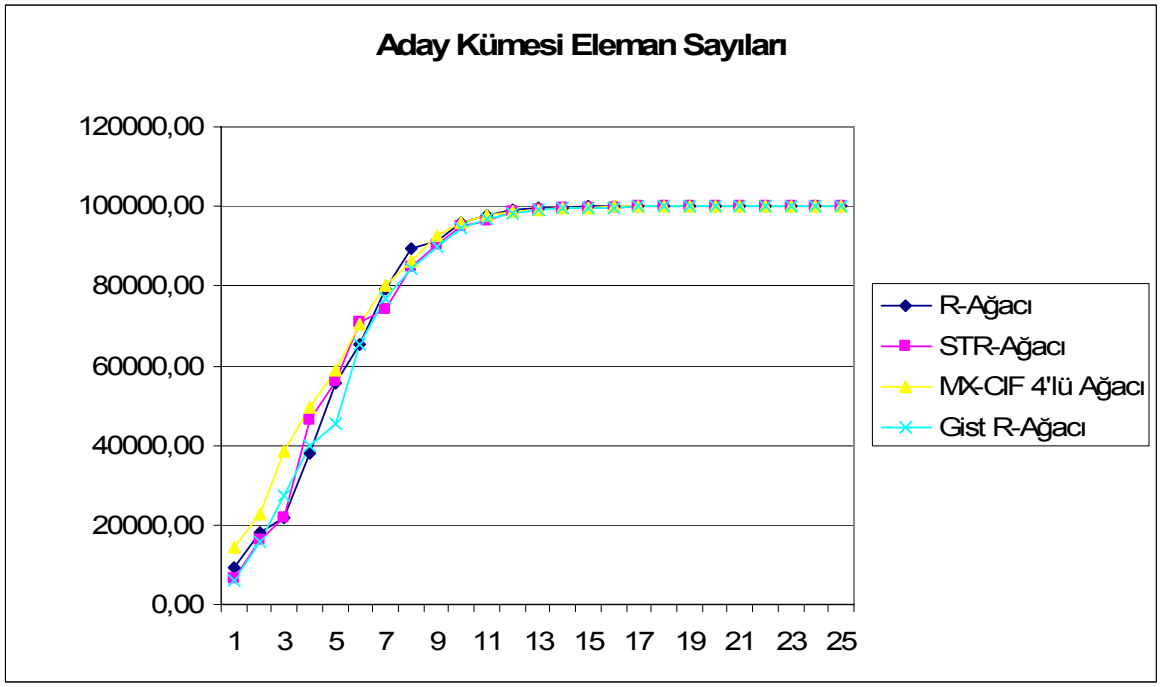


Şekil 6.3 : GD100, Pencere Sorguları Zaman Grafiği

	Aday Kümesi Eleman Sayıları (Ortalama)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	9204,06	6684,74	14173,06	6138,18
PBN 2	18279,14	16336,76	22541,46	15893,78
PBN 3	21797,02	21861,28	38335,12	27209,94
PBN 4	38115,06	46541,82	49788,98	39875,54
PBN 5	55536,62	55989,58	58776,58	45350,04
PBN 6	65273,58	70690,64	70338,82	65169,54
PBN 7	79005,88	74267,16	80223,68	76953,08
PBN 8	89386,74	84707,22	86021,54	84373,36
PBN 9	91460,72	90152,80	92774,26	89764,18
PBN 10	95762,94	95085,28	95736,06	94568,86
PBN 11	97748,14	96343,98	97599,10	96775,84
PBN 12	98926,92	98519,84	98574,34	98106,16
PBN 13	99482,42	99025,96	99331,24	99090,00
PBN 14	99768,46	99443,44	99596,88	99387,74
PBN 15	99925,66	99724,40	99774,38	99705,22
PBN 16	99971,80	99844,94	99892,58	99824,28
PBN 17	99993,30	99921,48	99946,24	99924,98
PBN 18	99997,70	99957,48	99975,00	99962,82
PBN 19	100000,00	99982,14	99988,66	99980,14
PBN 20	100000,00	99990,34	99993,66	99990,80
PBN 21	100000,00	99995,22	99997,04	99995,26
PBN 22	100000,00	99997,56	99998,86	99997,44

PBN 23	100000,00	99999,00	99999,36	99999,00
PBN 24	100000,00	99999,36	99999,74	99999,34
PBN 25	100000,00	100000,00	100000,00	100000,00

Tablo 6.6 : GD100, Aday Kümesi Eleman Sayısı Ortalamaları



Şekil 6.4 : GD100, Aday Kümesi Ortalama Eleman Sayısı Grafiği

Gauss dağılımına sahip her iki veri kümesiyle de yapılan testlerin sonucunda zaman eğrilerinin, sorgu penceresi büyüklük numarası 12 veya 13'e ulaştıktan sonra fazla değişim göstermediği görülmüştür. Test senaryolarına göre pencereler, veri uzayının tamamanı kapsayacak şekilde büyüdüğünden ve bu dağılıma sahip veriler orta bölgelerde toplandığından, belli bir pencere büyüklüğünden sonra kapsanan veri sayısında önemli artış olmamaktadır.

6.2 Düzgün Dağılım Sonuçları

Veri Kümesi Özellikleri :

Adı : DD20

Dağılım Tipi : Düzgün Dağılım

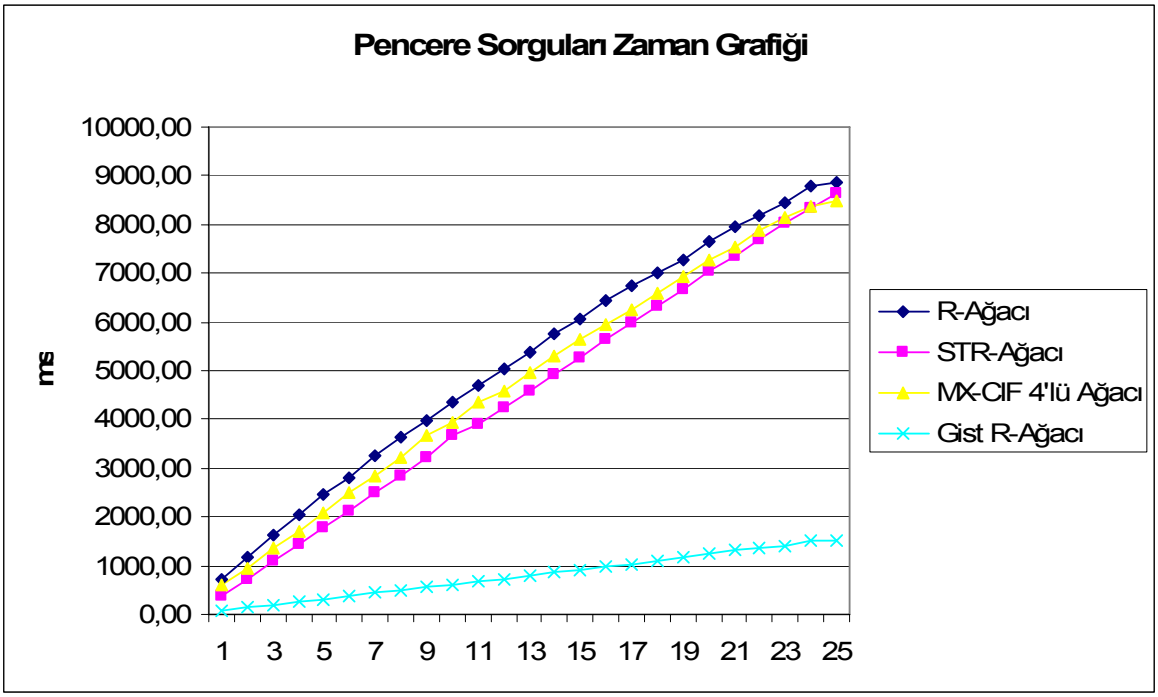
Veri Sayısı : 20.000

	R-Ağacı	STR-Ağacı	MXCIF 4'lü Ağacı
Derinlik	*5*	*5*	*22*
Oluşum Zamanı (ms)	*6131,88*	*6457,50*	*8166,25*

Tablo 6.7 : DD20, Dizin Oluşum Bilgileri

	Pencere Sorgusu Zamanları (Ortalama, ms)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	719,06	381,56	612,81	86,88
PBN 2	1169,06	712,81	954,69	135,31
PBN 3	1636,25	1098,75	1357,81	190,94
PBN 4	2051,88	1428,13	1706,88	253,75
PBN 5	2463,75	1786,25	2086,25	315,94
PBN 6	2811,25	2121,25	2505,31	376,56
PBN 7	3255,94	2494,69	2856,56	444,06
PBN 8	3649,69	2827,19	3228,44	493,75
PBN 9	3984,69	3228,44	3673,13	563,13
PBN 10	4357,50	3655,94	3951,25	615,00
PBN 11	4680,31	3898,75	4362,19	686,88
PBN 12	5028,44	4249,06	4578,44	738,13
PBN 13	5397,19	4578,75	4971,56	806,56
PBN 14	5748,75	4916,56	5311,88	857,50
PBN 15	6067,81	5271,88	5628,13	924,38
PBN 16	6430,00	5625,94	5942,19	985,94
PBN 17	6734,06	5976,56	6268,13	1037,19
PBN 18	7018,13	6329,69	6581,88	1108,44
PBN 19	7277,19	6670,63	6921,88	1175,00
PBN 20	7658,75	7033,75	7260,00	1263,44
PBN 21	7945,00	7330,31	7532,81	1314,38
PBN 22	8186,56	7691,25	7871,88	1352,19
PBN 23	8460,63	8044,38	8132,81	1419,06
PBN 24	8795,00	8345,00	8352,81	1513,44
PBN 25	8845,00	8639,69	8503,75	1528,44

Tablo 6.8 : DD20, Pencere Sorguları Zaman Ortalamaları

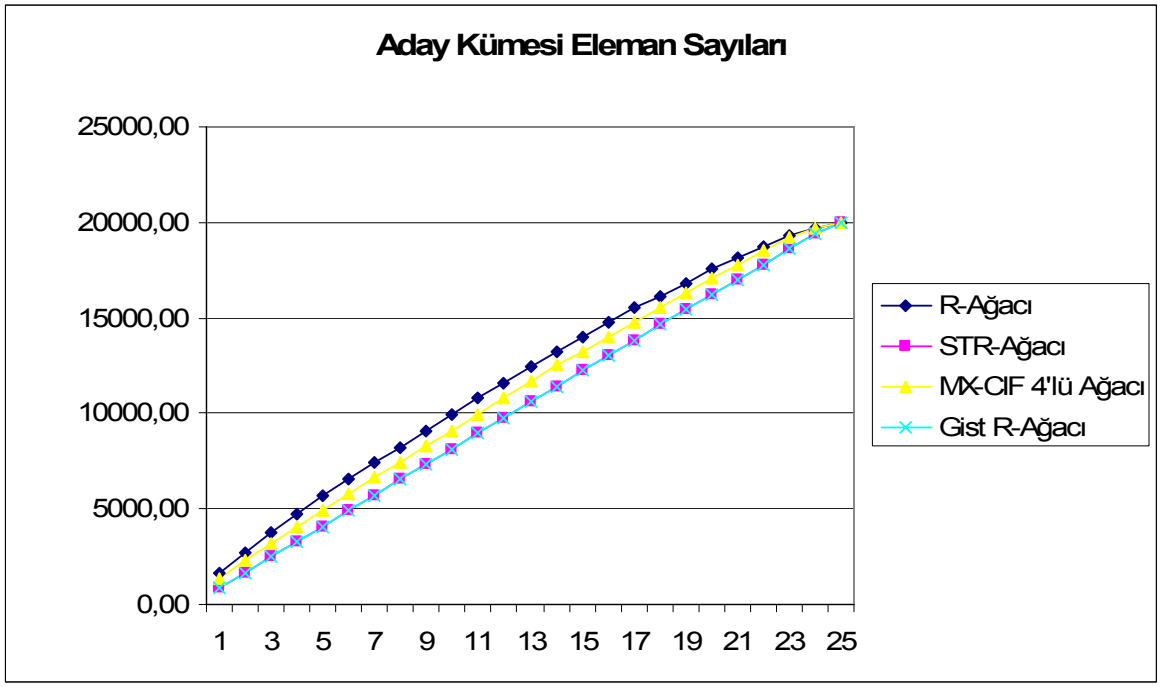


Şekil 6.5 : DD20, Pencere Sorguları Zaman Grafiği

	Aday Kümesi Eleman Sayıları (Ortalama)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	1615,18	844,44	1392,94	839,82
PBN 2	2717,36	1652,56	2273,50	1656,58
PBN 3	3720,40	2477,82	3142,82	2474,80
PBN 4	4728,18	3276,94	4013,76	3290,34
PBN 5	5656,92	4087,30	4938,38	4102,12
PBN 6	6524,16	4919,84	5816,42	4909,52
PBN 7	7413,48	5714,04	6677,30	5722,30
PBN 8	8249,04	6528,48	7472,60	6544,10
PBN 9	9090,30	7345,46	8302,34	7339,42
PBN 10	9972,18	8154,46	9115,50	8150,44
PBN 11	10807,00	8966,80	9967,30	8985,78
PBN 12	11626,98	9764,04	10783,32	9774,96
PBN 13	12469,42	10596,76	11687,50	10586,38
PBN 14	13250,08	11388,54	12502,48	11409,48
PBN 15	14003,46	12212,72	13269,32	12232,52
PBN 16	14806,32	13027,64	13987,98	13023,94
PBN 17	15536,46	13828,54	14773,48	13839,02
PBN 18	16163,90	14642,52	15504,32	14634,56
PBN 19	16836,14	15422,74	16330,22	15428,18
PBN 20	17536,26	16219,64	17101,86	16229,30
PBN 21	18190,82	17008,68	17753,56	17003,68
PBN 22	18746,14	17799,62	18493,04	17795,42

PBN 23	19276,24	18595,06	19176,66	18594,74
PBN 24	19736,78	19364,40	19693,76	19366,00
PBN 25	20000,00	19997,24	19999,44	19997,30

Tablo 6.9 : DD20, Aday Kümesi Eleman Sayısı Ortalamaları



Şekil 6.6 : DD20, Aday Kümesi Ortalama Eleman Sayısı Grafiği

Veri Kümesi Özellikleri :

Adı : DD100

Dağılım Tipi : Düzgün Dağılım

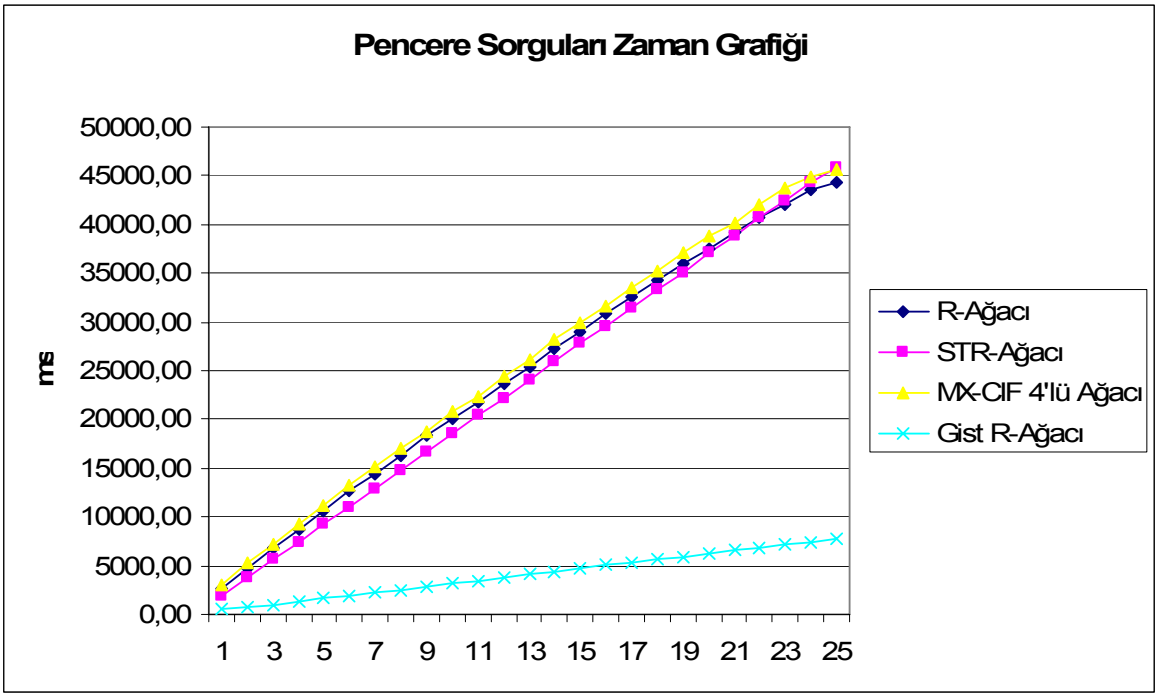
Veri Sayısı : 100.000

	R-Ağacı	STR-Ağacı	MXCIF 4'lü Ağacı
Derinlik	*6*	*6*	*26*
Oluşum Zamanı (ms)	*67136,88*	*32679,38*	*40973,13*

Tablo 6.10 : DD100, Dizin Oluşum Bilgileri

	Pencere Sorgusu Zamanları (Ortalama, ms)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	2682,50	1945,31	3108,75	520,31
PBN 2	4705,31	3767,81	5325,94	690,00
PBN 3	6835,63	5616,56	7239,38	979,06
PBN 4	8770,94	7472,19	9234,38	1295,63
PBN 5	10665,63	9259,69	11203,44	1620,94
PBN 6	12638,44	11078,13	13202,50	1923,13
PBN 7	14455,63	12928,75	15076,88	2241,88
PBN 8	16381,56	14790,00	16998,75	2539,06
PBN 9	18396,56	16610,00	18750,63	2855,63
PBN 10	20093,75	18501,88	20750,94	3151,56
PBN 11	21815,94	20375,00	22415,00	3495,31
PBN 12	23720,00	22234,06	24369,69	3791,25
PBN 13	25429,38	24080,00	26123,13	4078,13
PBN 14	27348,75	25927,81	28141,88	4375,94
PBN 15	29015,63	27819,06	29940,94	4670,94
PBN 16	30898,75	29533,75	31643,75	5019,06
PBN 17	32495,63	31415,00	33433,44	5314,06
PBN 18	34308,44	33245,00	35200,94	5588,75
PBN 19	36005,00	35074,38	37031,25	5903,44
PBN 20	37493,75	37073,13	38807,50	6218,44
PBN 21	39131,88	38894,38	40230,31	6597,81
PBN 22	40656,56	40701,25	42095,94	6852,50
PBN 23	42057,19	42477,19	43748,44	7208,75
PBN 24	43565,63	44329,69	44913,75	7460,00
PBN 25	44303,44	45864,69	45635,63	7790,31

Tablo 6.11 : DD100, Pencere Sorguları Zaman Ortalamaları

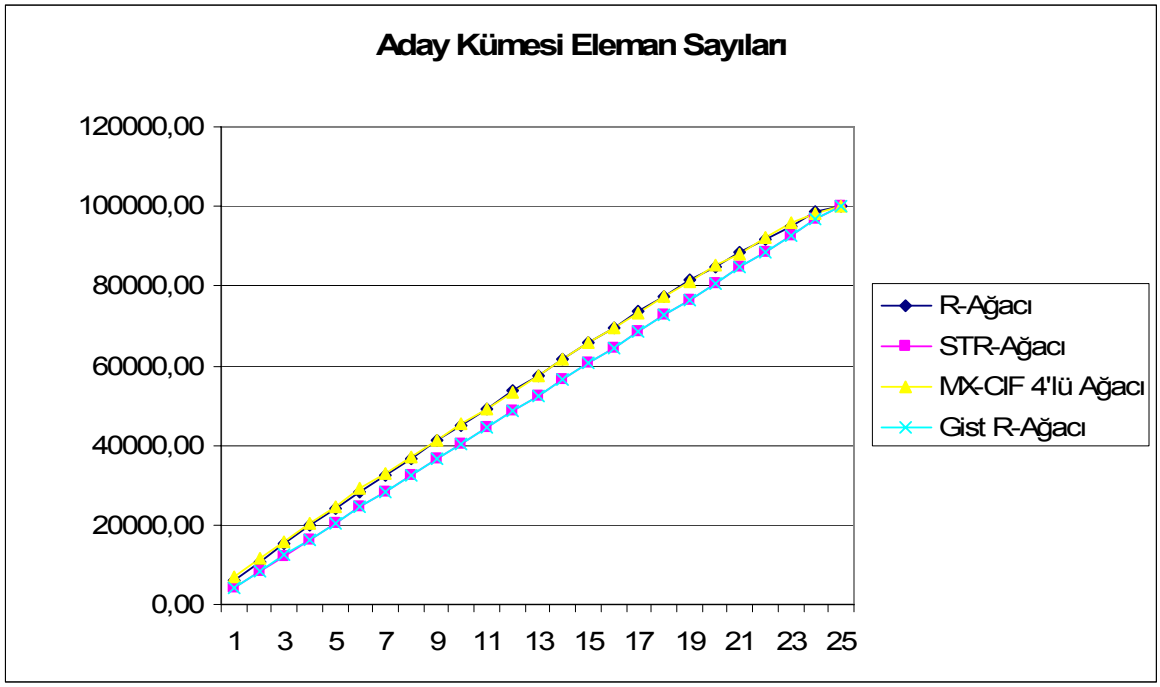


Şekil 6.7 : DD100, Pencere Sorguları Zaman Grafiği

	Aday Kümesi Eleman Sayıları (Ortalama)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	5984,52	4157,30	6790,22	4155,32
PBN 2	10567,84	8226,62	11631,24	8223,44
PBN 3	15333,28	12271,46	15907,14	12278,42
PBN 4	19755,68	16332,76	20222,50	16337,02
PBN 5	24004,72	20363,36	24590,04	20356,72
PBN 6	28432,40	24412,26	29025,92	24406,58
PBN 7	32520,16	28448,50	33071,94	28473,52
PBN 8	36822,32	32479,18	37291,76	32477,80
PBN 9	41211,40	36500,42	41162,06	36495,32
PBN 10	45085,00	40510,26	45499,38	40492,56
PBN 11	49329,74	44512,50	49205,90	44548,12
PBN 12	53650,76	48559,70	53468,20	48559,34
PBN 13	57342,92	52568,38	57340,68	52569,08
PBN 14	61850,80	56589,02	61674,80	56589,24
PBN 15	65682,78	60580,88	65601,20	60586,66
PBN 16	69708,64	64573,30	69362,76	64568,32
PBN 17	73462,64	68549,10	73347,76	68556,94
PBN 18	77433,36	72574,20	77162,48	72573,12
PBN 19	81417,44	76580,68	81097,26	76604,12
PBN 20	84860,38	80655,94	85022,02	80640,74
PBN 21	88454,68	84700,74	88238,40	84641,52
PBN 22	91826,52	88704,86	92284,84	88689,98

PBN 23	95158,22	92712,76	95711,38	92689,66
PBN 24	98456,00	96662,22	98328,56	96663,56
PBN 25	100000,00	99988,34	99997,38	99988,92

Tablo 6.12 : DD100, Aday Kümesi Eleman Sayısı Ortalamaları



Şekil 6.8 : DD100, Aday Kümesi Ortalama Eleman Sayısı Grafiği

Düzgün dağılıma sahip veri kümeleriyle elde edilen eğrilerde herhangi bir yakınsama olmamıştır. Sorgu pencereleri her adımda büyüdüğünden ve nereye yerleşirse yerleşsin veri uzayının her tarafına dağılmış olan veriler ile örtüştüğünden dolayı sürekli olarak artış gösteren karakterde sonuçlar alınmıştır.

6.3 Bezier Eğrileri ile Dağılım Sonuçları

Veri Kümesi Özellikleri :

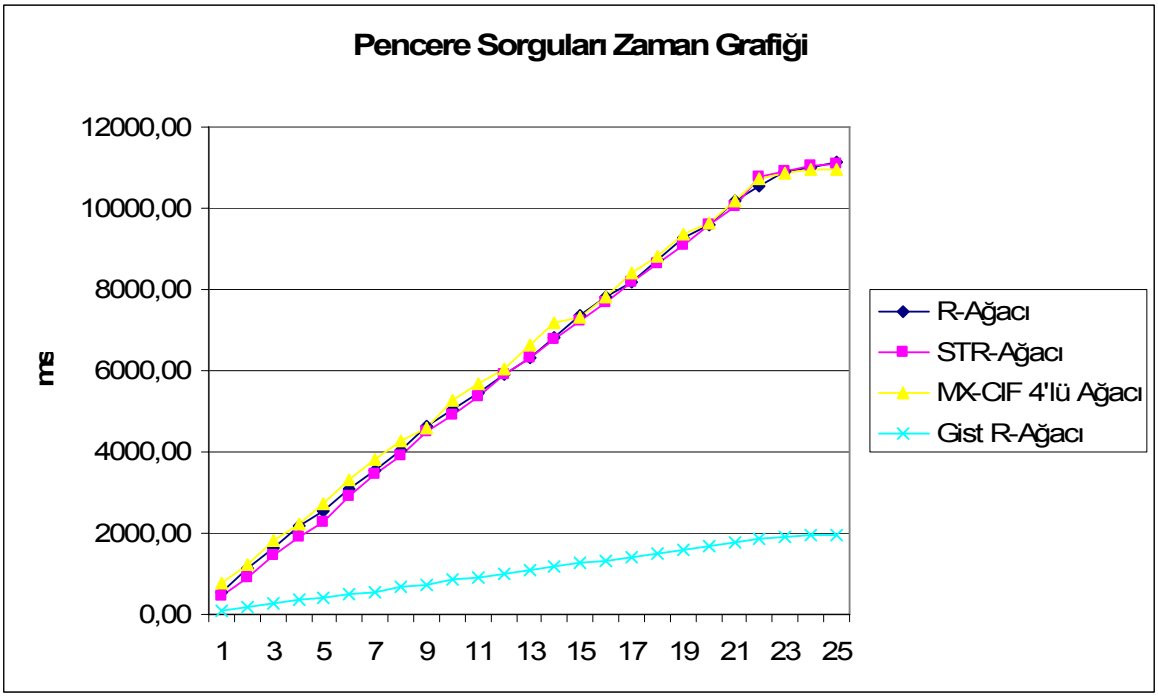
Adı	:	BED25
Dağılım Tipi	:	Bezier eğrileri kullanımı ile
Veri Sayısı	:	25.740

	R-Ağacı	STR-Ağacı	MXCIF 4'lü Ağacı
Derinlik	*6*	*5*	*16*
Oluşum Zamanı (ms)	7983,75	*8226,88*	*8742,50*

Tablo 6.13 : BED25, Dizin Oluşum Bilgileri

	Pencere Sorgusu Zamanları (Ortalama, ms)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	561,56	467,50	781,56	103,13
PBN 2	1120,94	913,75	1241,25	180,63
PBN 3	1619,38	1454,38	1823,13	275,31
PBN 4	2175,94	1922,19	2234,38	348,13
PBN 5	2560,63	2259,69	2722,81	419,69
PBN 6	3085,31	2888,44	3311,88	505,94
PBN 7	3543,13	3436,56	3806,25	559,69
PBN 8	4042,50	3886,56	4294,06	661,88
PBN 9	4640,63	4501,56	4613,13	748,44
PBN 10	5030,31	4907,81	5277,19	843,75
PBN 11	5447,81	5355,63	5660,63	925,00
PBN 12	5898,44	5894,06	6038,44	999,06
PBN 13	6326,25	6330,63	6620,94	1092,19
PBN 14	6803,75	6785,00	7168,75	1175,31
PBN 15	7342,81	7205,94	7327,50	1254,38
PBN 16	7813,75	7685,63	7809,38	1337,50
PBN 17	8175,31	8166,25	8391,25	1416,88
PBN 18	8738,75	8618,44	8817,81	1491,88
PBN 19	9273,75	9093,75	9364,69	1585,00
PBN 20	9593,75	9575,31	9642,81	1680,94
PBN 21	10177,81	10062,50	10174,69	1766,88
PBN 22	10558,75	10766,56	10745,00	1868,75
PBN 23	10893,75	10905,94	10850,63	1926,56
PBN 24	10990,63	11058,13	10940,63	1944,69
PBN 25	8845,00	8639,69	8503,75	1528,44

Tablo 6.14 : BED25, Pencere Sorguları Zaman Ortalamaları

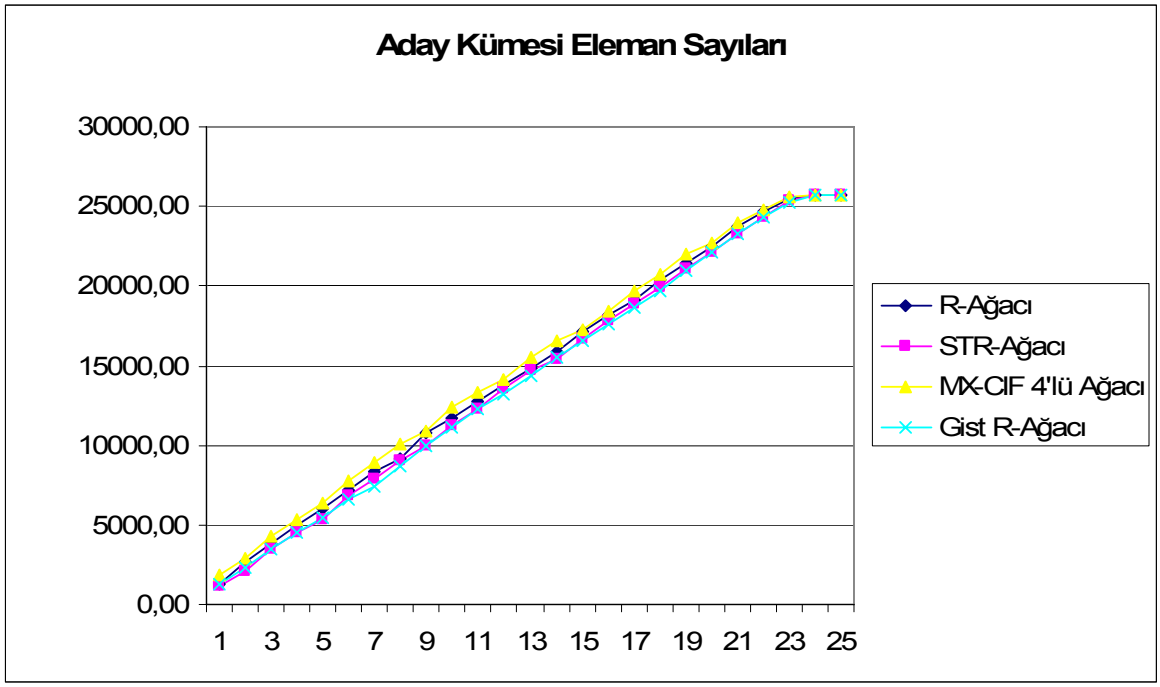


Şekil 6.9 : BED25, Pencere Sorguları Zaman Grafiği

	Aday Kümesi Eleman Sayıları (Ortalama)			
	R-Ağacı	**STR-Ağacı**	**MX-CIF 4'lü Ağacı**	**Gist R-tree PostgreSQL**
PBN 1	1271,28	1108,24	1843,02	1256,52
PBN 2	2625,46	2128,44	2936,80	2325,74
PBN 3	3771,88	3419,92	4249,44	3481,52
PBN 4	5020,92	4486,02	5306,50	4565,30
PBN 5	6027,48	5303,20	6343,32	5491,42
PBN 6	7160,04	6826,42	7755,44	6625,54
PBN 7	8314,82	7922,42	8911,00	7441,96
PBN 8	9176,86	9021,50	10103,18	8691,00
PBN 9	10727,80	9943,62	10854,94	9921,10
PBN 10	11714,28	11239,18	12409,92	11158,98
PBN 11	12794,76	12299,22	13306,90	12293,50
PBN 12	13785,74	13591,60	14181,06	13261,36
PBN 13	14791,46	14671,28	15568,22	14416,84
PBN 14	15885,34	15438,82	16556,66	15506,28
PBN 15	17143,06	16650,20	17253,48	16563,46
PBN 16	18232,18	17808,52	18398,54	17581,92
PBN 17	19112,84	18878,86	19696,98	18658,60
PBN 18	20398,18	19867,68	20769,96	19650,98
PBN 19	21379,08	21053,76	21981,14	20921,88
PBN 20	22430,88	22147,54	22701,88	22132,32
PBN 21	23780,08	23261,90	24012,20	23261,32
PBN 22	24686,18	24381,22	24804,92	24380,88

PBN 23	25434,34	25346,22	25542,30	25283,92
PBN 24	25731,50	25711,32	25736,02	25710,88
PBN 25	25740,00	25740,00	25740,00	25740,00

Tablo 6.15 : BED25, Aday Kümesi Eleman Sayısı Ortalamaları



Şekil 6.10 : BED25, Aday Kümesi Ortalama Eleman Sayısı Grafiği

7. Değerlendirme

Konumsal dizinleme yöntemlerinin başarımlarını karşılaştırmak amacıyla yapılan bu tez çalışmasında R-ağacı, STR-ağacı ve MX-CIF 4'lü ağacı kullanılmıştır. İstatistiksel dağılımlar ve Bezier eğrileri yardımıyla üretilen sentetik veri kümeleri kullanılarak uygulanan test senaryolarıyla da başarımları ölçülmüş ve karşılaştırılmıştır.

Bir konumsal dizinleme yöntemin başarısını değerlendirirken dikkat edilmesi gereken önemli noktalardan bir tanesi; veri kümesinin bütünü içerisinden seçilen, sorgu koşuluna göre sonuç olma potansiyeli bulunan adaylardan oluşan kümenin eleman sayısıdır. Dizin yapısı ne kadar verimli, ne kadar güçlüyse adayları o kadar iyi süzecektir ve aday kümesinin boyu azalacaktır. Bu temel düşünce ile zaman sonuçlarına bakıldığında aday küme eleman sayılarıyla aralarındaki ilişki anlam kazanacaktır.

Tez çalışmasının sonuçlarını iki aşamalı olarak değerlendirmek faydalı olacaktır. İlk olarak VTYS üzerinde dizinleme katmanı kurularak gerçekleştirilen dizin yapılarının (R-ağacı, STR-ağacı ve MX-CIF 4'lü ağacı) sonuçlarını ele alalım. Sonuç tabloları ve grafikleri beraber incelendiğinde bu üç yöntem arasında sorgu başarımı en yüksek olan STR-ağacıdır. Hem sorgu zamanları hem de aday küme eleman sayıları açısından diğer iki yöntemi geride bırakmayı başarmaktadır. Fakat sorgu başarımı konusunda öne çıkan bu yöntemin içsel yapısı ve tanımı gereği bir dezavantajı vardır. Sık değişim göstermeyen, durağan ortamlarda etkili sorgu başarımı gösterebilecek olan STR-ağacı, ekleme ve silme işlemlerinin bol olduğu dinamik ortamlarda yeniden yapılandırmaya ihtiyaç duyacaktır. Bu yönüyle düşünüldüğünde haritacılık gibi verilerin baştan belirli olduğu, sürekli güncellenmediği veya güncellense bile belirli aralıklarla bakım yapma, yeniden yapılandırma ihtiyacının rahatlıkla karşılanabileceği alanlarda tercih edilmesi daha uygun olacaktır.

R-ağacı ve MX-CIF 4'lü ağacı dinamik ortamalara yapısal olarak daha uygundur. Veriler dizinlenmeden önce ön işlemlerin uygulanmadığı bu yöntemler ekleme ve silme işlemleri sonrasında yapılarını dinamik olarak düzenleyebilirler. Sorgu başarımları açısından incelendiklerinde ise kendi aralarındaki fark, STR-ağacı ile aralarındaki kadar açık değildir. Veri sayısı 20.000 olduğunda daha iyi sonuç veren MX-CIF 4'lü ağacı, bu sayı artıp 100.000 olduğunda R-ağacının gerisine düşmektedir.

Sonuçları bir de ayrı dizin katmanı ve VTYS ile bütünleşen dizin açısından ele alırsak ve sorgu başarımları açısından değerlendirirsek biraz farklı bir durum ortaya çıkmaktadır. Aday kümesi eleman sayılarına bakıldığında benzerlik ortadadır. Yani süzme işlemi her yöntem ve her iki mimari yaklaşımda da başarılı tamamlanabilmektedir. Fakat sorgu zamanları açısından açık fark vardır. Ayrık katman mimarisine sahip ve bütünleşik olarak VTYS içinde tutulan dizin yapılarının ne gibi farklar yaratacağını da ortaya koyan bu sonuç açıkca göstermektedir ki ayrık katman mimarisi sorgu tepki süresi bakımından yavaş kalmaktadır.

Ayrık dizin katmanı bünyesinde karşılaştırılan yöntemlerin sonuçları ve VTYS ile bütünleşik dizin mimarisinin yarattığı fark yeni çalışma konularının ipuçlarını vermektedir. Bütünleşik bir yapı ile STR-ağacı ve MX-CIF 4'lü ağacının gerçekleştirilmesi sonucunda mevcut ticari ve deneysel sistemlere rakip bir konumsal veritabanı oluşturulabilir.

8. Kaynakça

1. A. Guttman, R-Trees: A Dynamic Index Structure for Spatial Searching, Proc. of the ACM SIGMOD Conf., 1984, 47-57.
2. Abel , D.J. and Ooi, B.C., eds. Proceedings of the Third International Symposium on Large Spatial Databases, Singapore, 1993
3. Abel, D.J. SIRO-DBMS : A database toolkit for geographical information systems. International Journal of Geographical Information Systems, 3:103-116, 1989
4. Berman, R.R. and Stonebreaker, M. GEO-QUEL : A system for the manuplation and display of geographic data, Computer Graphics, 11:186-191, 1977
5. Buchmann, A., Günther, O., Smith T.R., and Wang, Y.F., eds. Proceedings of the First International Symposium on Large Spatial Databases, Santa Barbara, 1989
6. Chang, N.S. and Fu, K.S. A relational database system for images. In: Chang S.K. and Fu, K.S., eds, Pictorial Information Systems, Berlin, Springer, 1980, pp. 288-312
7. D. Rotem: Spatial join indices. Proc. IEEE 7th Int. Conf. on Data Eng., 500-509 (1991).
8. D.J. Abel and J. L. Smith, A data structure and algorithm based on a linear key for a rectangle retrieval problem, Computer Vision, Graphics, and Image Processing 24,1(October 1983), 1-13. [rectangles] D.3.5.1 A.2.1.1
9. D.W. Adler, IBM DB2 Spatial Extender Spatial Data within the RDBMS, Proceedings of the 27th VLDB Conference, Roma, Italy, 2001
10. D.W. Smith, Bezier Curve Ahead!, The journal of Macintosh Technology, Volume 5 Issue 1, (http://www.mactech.com/articles/mactech/Vol.05/05.01/BezierCurve/index.html), Son erişim tarihi : 31.01.2007
11. Egenhofer M.. Spatial SQL: A query and presentation language. IEEE Transactions on Knowledge and Data Engineering, 6:86-95, 1994.

12. Egenhofer, M. A formal definition of binary topological relationships. Proceedings of the Third International Conference on the Foundations of Data Organization and Algorithms, Paris, 1989
13. Egenhofer, M. and Herring, J. A mathematical framework for the definition of topological relationships. Fourth International Symposium on Spatial Data Handling, Zürich, 1990
14. Egenhofer, M. J. & Franzosa, R. D. (1991), `Point-Set Topological spatial relations', International Journal for Geographical Information Systems 5(2), 161--174.
15. Egenhofer, M., Frank, A., and Jackson, J.P. A topological data model for spatial databases. Proceedings of the First International Symposium on Large Spatial Databases, Santa Barbara, 1989
16. Egenhofer, Max J.; Herring, John R. (1991): Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. Technical report, Department of Surveying Engineering, University of Maine, Orono, ME, 1992.
17. Emmanuel S., Timos S., Towars The Design of A DBMS Repository for The Application Domain of GIS, Requirements of Users and Applications, Proceedings of the 18th International Cartographic Conference, Stockholm, Sweden, pp 2030-2037
18. Frank A., “Requirements for a database management system for a GIS” Photogrammetric Eng. & Remote Sensing, vol. 54, no. 11, pp.1557-1554, Nov. 1998.
19. Frank, A. And Khun, W. Cell graphs : A provable correct method for the storage of geometry. Proceedings of the Second International Symposium on Spatial Data Handling, Seattle, 1986
20. Frank, A. Properties of Geographic Data : Requirements for Spatial Access Methods. Proceedings of the Second International Symposium on Large Spatial Databases, Zürich, 1991
21. Franklin, W.R. Cartographic errors symptomaticof underlying algebra problems. Proceedings of the First International Symposium on Spatial Data Handling, Zürich, 1984

22. G. Kedem, The Quad-CIF tree: a data structure for hierarchical on-line algorithms, Proceedings of the Nineteenth Design Automation Conference, Las Vegas, June 1982, 352-357.
23. G.M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, N J, 1978.
24. Gargano, M., Nardelli, E. and Talamo, M. Abstract data types for the logical modeling of complex data. Information Systems, 16:565-584, 1991
25. Greene, D. And Yao, F. Finite-resolution computational geometry. Proceedings of the Twenty-Seventh IEEE Symposium on Foundations of Computer Science, Toronto, 1986
26. Gültig R. H. An Introduction to Spatial Database Systems; VLDB Journal,3, 357-399 (1994).
27. Günther, O. and Buchmann, A. Research Issues In Spatial Databases, ACM SIGMOD Record, 19:61-68, 1990
28. Günther, O. And Schek, H.-J., eds. Proceedings of the Second International Symposium on Large Spatial Databases, Zürich, 1991
29. Güting, R.H. and Schneider, M. Realm-based spatial data types: The ROSE algebra. Fernuniversitat Hagen, Report 141, 1993b
30. Güting, R.H. and Schneider, M. Realms : A foundation for spatial data types in database systems. . Proceedings of the Third International Symposium on Large Spatial Databases, Singapore, 1993a.
31. Güting, R.H. Geo-relational algebra: A model and query language for geometric database systems. In: Schmidt, J.W., Ceri, S., and Missikoff, M., eds., Proceeding of the International Conference on Extending Database Technology, Venice, 1988
32. H. Samet, The Design and Analysis of Spatial Data Structures, Addison Wesley, 1990, Section 2.6
33. H. Samet, The Design and Analysis of Spatial Data Structures, Addison Wesley, 1990, Section 6.3
34. H. Samet, The Quadtree and Related Hierarchical Data Structures, Computing Surveys, Voi. 16, No. 2, June 1984, ACM
35. Introduction to VLSI Systems", Mead & Conway, A-W 1980, Section 4.5

36. J. Gray, A. Szalay, G. Fekete, Using Table Valued Functions in SQL Server 2005 To Implement a Spatial Data Library, Technical Report, MSR-TR-2005-122, August 2005
37. J. Nievergelt, H. Hinterberger and K.C. Sevcik: The Grid File: An adaptable, symmetric multikey file structure. In Trends in Information Processing Systems, iProc. 3rd ECI Conf., in A. Duijvestijn and P.Lockemann (eds.), Lecture Notes in Computer Science #123, 236-251, Springer Verlag (1981).
38. J. Nievergelt, H. Hinterberger, K. C. Sevcik: The grid file: An adaptable, symmetric multikey file structure. ACM Trans. on Database Sys. 9, 1, 38-71 (1984).
39. J. Nievergelt, K. Hinrichs: Storage and access structures for geometric data bases. Int. Conf. on Foundations of Data Organization, Kyoto, Japan, 335-345 (1985).
40. Joseph, T. And Cardenas, A. PICQUERY: A high level query language for pictorial database management. IEEE Transactions on Software Engineering, 14:630-638, 1988
41. K. Hinrichs, J. Nievergelt: The grid file: A data structure designed to support proximity queries on spatial objects. Proc. Int. Workshop on Graphtheoritic Concepts in Comp. Science. Trauner-Verlag, 100-113, (1983).
42. Kamel, I., Faloutsos, C., “On Packing R-trees”, Proc. 2nd International Conference on Information and Knowledge Management (CKIM-93), p. 490-499, Arlington, VA, November 1993.
43. Klinger and C.R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1(March 1976), 68-105.
44. Larue, T. Pastre, D. And Viemont, Y. Strong integration of spatial domains and operators in a relational database system. Proceedings of the Third International Symposium on Large Spatial Databases, Singapore, 1993
45. Lipeck, U. And Neumann, K. Modelling and manipulating objects in geoscientific databases. Proceedings of the Fifth International Conference on the Entity-Relationship Approach, Dijon, France, 1987
46. M. Hacıömeroğlu, Coğrafi Bilgi Sistemlerinde Geometri Kütüphanesi, Bilgisayar Mühendisliği A.B.D Yüksek Lisans Tezi, Başkent Üniversitesi, 2006

47. Morehouse, S. The architecture of ARC/INFO. Proceedings of the Auto-Carto 9, Baltimore, MD, 1989
48. N. Beckmann, H. P Kriegel, R. Schneider, B. Seeger, “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles”, Proc. of the ACM SIGMOD Conf., 1990, 322-331.
49. O. Günther. Efficient computation of spatial joins. In Proc. IEEE 9th Int. Conference on Data Engineering, 1993
50. Open GIS Consortium, Inc. OpenGIS Simple Features Specification For SQL Revision 1.1 May 5, 1999
51. Orenstein, J.A. and Manola, F. PROBE spatial data modeling and query processing in an image database application IEEE Transactions on Software Engineering, 14:611-629, 1988
52. PostGIS Manual, Working Paper, 30.12.2005
53. Pullar, D. and Egenhofer, M. Towards formal definitions of topological relations among spatial objects. Proceedings of the Third International Symposium on Spatial Data Handling, Sydney, 1988.
54. R. Sproull, R. Lyon, The Caltech Intermediate Form for LSI Layout Description, Silicon Structures Project, Technical Report 2686, February 11, 1980
55. R.A. Finkel and J.L. Bentley, Quad trees: a data structure for retrieval on composite keys, Acta Informatica ~, 1(1974), 1-9.
56. R.K.V. Kothuri, S. Ravada, D. Abugov, Quadtree and R-tree Indexing In Oracle Spatial: A Comparison Using GIS Data, Proceedings of the 2002 ACM SIGMOD international conference on Management of data,
57. Roussopoulos, N. Faloutos, C., and Sellis, T. An efficient pictorial database system for PSQL. IEEE Transactions on Software Engineering, 14:639-650, 1988
58. S. T. Leutenegger, M. A. Lopez, J. Edgington, STR : A Simple and Efficient Algorithm for R-tree Packing, Proceedings of 13th International Conference on Data Engineering, 497-506, 1997
59. S. T. Rubin, Computer Aids for VLSI Design, 1994 (http://www.rulabinsky.com/cavd/text/chapb.html) Son erişim tarihi : 31.01.2007
60. Samet H. The design and analysis of spatial data structures AW, 1990

61. Schilcher, M. Interactive graphic data processing in cartography. Computers & Graphics, 9:57-66, 1985
62. Scholl, M. and Voisard, A. Thematic map modeling. Proceedings of the First International Symposium on Large Spatial Databases, Santa Barbara, 1989
63. Smith, T.R., Menon, S., Star, J.L., and Estes, J.E. Rrequirements and principles for the implementation and construction of large-scale geographic information systems. International Journal o Geographical Information Systems, 1:13-31, 1987
64. Spatial Extensions in MySQL chapter of the MySQL Reference Manual (http://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html) Son erişim tarihi : 31.01.2007
65. Svensson, P. and Huang, Z. Geo-SAL: A query language for spatial data analysis. Proceedings of the Second International Symposium on Large Spatial Databases, Zürich, 1991
66. T. Sellis, N. Roussopoulos, C. Faloutsos, “R+-Tree: A Dynamic Index for Multi-Dimensional Objects”, Proc. of the 13th VLDB Conf., 1987, 507-518.
67. Tahsin Yomralıoğlu, Coğrafi Bilgi Sistemleri, Temel Kavramlar ve Uygulamalar, Bölüm 2, sayfa 36, Atlas Kitapçılık, 2006
68. Tomlin, C.D. Geographic Information Systems and Cartographic Modeling. Englewood Cliffs, NJ:Prentice Hall, 1990
69. W. Lu, J. Han: Distance-associated join indices for spatial range search. Proc. 8th IEEE. Int. Conf. on Data Engineering, 284-292 (1992).
70. Waugh, T.C. and Healey, R.G. The GEOVIEW desing : A Relational database approach to geographical data handling. International Journal of Geographical Information Systems, 1:101-118, 1987
71. WikiPedia Free Encyclopedia, (http://en.wikipedia.org/wiki/B%C3%A9zier_curve) , Son erişim tarihi : 31.01.2007
72. Worboys, M.F. A generic model for planar geographical objects. International Journal of Geographical Information Systems, 6:353-372, 1992

EK - 1

Kolay Belgele s.2.0 ile Genel Deney Düzeneği

Modül Adı : **Evliya Sentetik Veri Üretici**	Programlama Dili : **C#**
Durum : **Tamamlandı**	Tarih : **25.02.07**

No	**Alt Modül**	**Güçlük Kodu**	**Kalıtım**	**Belgelemesiz Komut Sayısı**	**Önerilen Belgeleme Yüzdesi**	**Emek Normalize Değeri**
1	UniformDeviate	**2.5**	-	**182**	%10	468
2	GaussianDeviate	**2.5**	1	**62**	%10	468+159
3	BezierCurve	**5.0**	-	**1170**	%15	5973

Modül Adı : **WKT Çözümleyici-Çevirici**	Programlama Dili : **C#**
Durum : **Tamamlandı**	Tarih : **25.02.07**

No	**Alt Modül**	**Güçlük Kodu**	**Kalıtım**	**Belgelemesiz Komut Sayısı**	**Önerilen Belgeleme Yüzdesi**	**Emek Normalize Değeri**
1	GeometricLib	**1.5**	-	**580**	%10	910
2	Parser	**1.5**	-	**210**	%10	330

Modül Adı : **Veritabanı Arayüzü**	Programlama Dili : **C#**
Durum : **Tamamlandı**	Tarih : **25.02.07**

No	**Alt Modül**	**Güçlük Kodu**	**Kalıtım**	**Belgelemesiz Komut Sayısı**	**Önerilen Belgeleme Yüzdesi**	**Emek Normalize Değeri**
1	PostgresClient	**1.3**	-	**297**	%10	407

Modül Adı : **Evliya Veri Düzenleyici**	Programlama Dili : **C#**
Durum : **Tamamlandı**	Tarih : **25.02.07**

No	**Alt Modül**	**Güçlük Kodu**	**Kalıtım**	**Belgelemesiz Komut Sayısı**	**Önerilen Belgeleme Yüzdesi**	**Emek Normalize Değeri**
1	DataEditor	**1.5**	-	**529**	%15	849
2	BezierDataView	**1.8**	-	**57**	%10	107

Modül Adı : **Sonuç Toplayıcı**	Programlama Dili : **C#**
Durum : **Tamamlandı**	Tarih : **25.02.07**

No	**Alt Modül**	**Güçlük Kodu**	**Kalıtım**	**Belgelemesiz Komut Sayısı**	**Önerilen Belgeleme Yüzdesi**	**Emek Normalize Değeri**
1	QWGenerator	**1.8**	-	**61**	%10	114
2	StatsCollector	**1.3**	-	**248**	%10	340

Modül Adı : **Dizinleme Katmanı**	Programlama Dili : **C#**
Durum : **Tamamlandı**	Tarih : **25.02.07**

No	**Alt Modül**	**Güçlük Kodu**	**Kalıtım**	**Belgelemesiz Komut Sayısı**	**Önerilen Belgeleme Yüzdesi**	**Emek Normalize Değeri**
1	R-ağacı	**8.5**	-	**3975**	%40	34900
2	STR-ağacı	**8.0**	-	**1359**	%40	11252
3	MX-CIF 4'lü Ağacı	**8.2**	-	**1305**	%40	11066

Modül Adı : **Evliya Test Programı**	Programlama Dili : **C#**
Durum : **Tamamlandı**	Tarih : **25.02.07**

No	**Alt Modül**	**Güçlük Kodu**	**Kalıtım**	**Belgelemesiz Komut Sayısı**	**Önerilen Belgeleme Yüzdesi**	**Emek Normalize Değeri**
1	MainTestForm	**1.8**	-	**478**	%15	933

Curriculum Vitae

Personal Information

Murat Seçkin Ayhan

Computer Engineer, B.Sc., M.Sc.

Date of Birth : 30.06.1981 **Place of Birth : Ankara / TURKEY**

Citizenship : Turkish Republic **Marital Status : Single**

Telephone : +90 312 215 43 31 **Cell Phone : +90 533 260 69 22**

Address : 46. Sokak 17/6 , Bahçelievler, ANKARA, TURKEY, 06500

Web : http://www.atilim.edu.tr/~msayhan e-mail : msayhan@gmail.com

EDUCATION

2004 – 2007

Başkent University, Ankara
Faculty of Engineering, Department of Computer Engineering, M.Sc.

2000 – 2004

Başkent University, Ankara
Faculty of Engineering, Department of Computer Engineering, B.Sc.

Favorite Courses

Multimedia and Wireless Data Networks, Digital Design, Microprocessors, Computer Architecture & Organization, Computer Networks, Electronics, Expert Systems, Object-Oriented Programming, Image Processing, Data Management & File Organization, Database Management Systems, Operating Systems, System Engineering, Software Engineering, Computational Geometry

Projects

Evliya Celebi : Geographic Information Core, Research Project managed by Hayri Sever, Prof. Dr. and funded by TUBITAK under the number of SOBAG-105K040.

Feature Extraction and Semantic Information Annotation From Audio, Part of Content-Based Retrieval of Audio Project, Listed in 50 Project of BT Haber Kampus Project and Presented in CeBIT 2004, Turkey (Graduation Project)

1997-1999

Çankaya Atatürk High School

1994-1997

Elazığ Anatolian High School

1992-1994

Çanakkale Anatolian High School

ACADEMIC BACKGROUND

Master Thesis

Comparison of Spatial Indexing Methods, Master Thesis, In the concept of research project Evliya Celebi : Geographic Information Core funded by TUBITAK

SKILLS and ABILITIES

Computer Oriented Languages

C, C++, C#, Java, Assembly, SQL, PERL, HTML, XML, PHP, ASP, UML, Java Script, Shell Scripts

Operating Systems

Windows, Linux

Applications and Toolkits

Microsoft Visual Studio .NET, Borland C++, Java, C# Builder, Rational Rose, Visio, Matlab, MS Office Tools, Macromedia Dreamveawer, LogicWorks, PSpices, J2ME Wireless Toolkit, TrueVision 3D SDK, OpenGL Utility Toolkit

Languages

Turkish(native), English (advanced) *[Toefl CBT: 227]*, German (beginner)

OCCUPATIONAL EXPERIENCE

Since Sep. 2006	Teaching and Research Assistant, Faculty of Engineering, Department of Computer Engineering, Atılım University, Ankara
Sep. 2004 - August 2006	Teaching and Research Assistant, Faculty of Engineering, Department of Computer Engineering, Başkent University, Ankara
June 2003	ASELSAN Electronics Industry Inc. Macunköy/ANKARA Communication Division, Software Developers Group Internship
July 2002	ASELSAN Electronics Industry Inc. Macunköy/ANKARA Communication Division, Network Group Internship

SCHOLARSHIPS and HONOURS

3rd Ranked Graduation from Department of Computer Engineering at Başkent University

Başkent University Academic Achievement Scholarship (2003-2004) (2002-2003) (2001-2002)

REFERENCES

References available upon request.