T.C.
MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YOĞUNLUK TABANLI KÜMELEME METODLARI KULLANILARAK PARALEL VERİ MADENCİLİĞİ GERÇEKLEŞTİRİLMESİ

SÜLEYMAN ZAFER SEVER

Yüksek Lisans Tezi

Tez Danışmanı
Yrd. Doç. Dr. Turgay Tugay Bilgin

İSTANBUL – 2010

T.C.
MALTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

YOĞUNLUK TABANLI KÜMELEME METODLARI KULLANILARAK PARALEL VERİ MADENCİLİĞİ GERÇEKLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

SÜLEYMAN ZAFER SEVER

Tez Danışmanı
Yrd. Doç. Dr. Turgay Tugay Bilgin

İSTANBUL – 2010

ÖZET

Yüksek Lisans Tezi, Yoğunluk Tabanlı Kümeleme Metodları Kullanılarak Paralel Veri Madenciliği Gerçekleştirilmesi, Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Bu tez çalışmasında DBSCAN ve OPTICS algoritmaları, yazarlarının makalelerinde ortaya koyduğu sözde kodlar temel alınarak kodlanmıştır. Tek bilgisayar üzerinde gerçekleştirilen çalışmalarda komşuluk sorgularının çok zaman aldığı görülmüş ve bu sürenin kısaltılması için R*-Tree veri yapısı kullanılmıştır. DBSCAN algoritmasının paralelleştirilmesi için LAM/MPI kütüphanesi kullanılmıştır. DBSCAN uygulamasının en çok zaman harcayan kısmı olan komşuluk sorguları, LAM/MPI yardımı ile tüm bilgisayarlara eşit şekilde paylaştırılarak yapılmıştır. 3 farklı veri seti ile gerçekleştirilen testlerde DBSCAN algoritmasının paralelleştirmeye elverişli olduğu ve paralel çalışan DBSCAN'in Amdahl Kanunu'na uygun olarak çalışma süresinin kısaldığı, bununla birlikte küme oluşturma performansının ve kalitesinin etkilenmediği görülmüştür.

Toplam 6 bölümden oluşan tezin birinci bölümünde genel kavramlardan bahsedilmiştir. İkinci bölümde veri madenciliğinin genel tanımı, uygulama alanları, veri madenciliği süreci ve veri madenciliği tekniklerinden bahsedilmiştir. Üçüncü bölümde kümeleme analizinin türlerinden, yoğunluk tabanlı kümeleme yöntemleri ağırlıklı olmak üzere bahsedilmiştir. Dördüncü bölümde paralel hesaplamanın amacı, paralel bilgisayar bellek mimarileri, paralel programlama modelleri ve paralel program tasarımından bahsedilmiştir. Beşinci bölümde paralel DBSCAN uygulamasının geliştirilme amacı, geliştirme ortamı, kullanılan araçlar, kullanılan veri setleri ve uygulamanın geliştirme adımlarından bahsedilmiştir. Altıncı ve son bölümde deneysel sonuçlar tablolar ve grafiklerle verilmiş ve elde edilen sonuçlar irdelenmiştir. Ayrıca bu konuda çalışma yapacak araştırmacılar için öneriler sunulmuştur.

Bu tez 2010 yılında tamamlanmıştır ve 84 sayfadan oluşmaktadır.

Anahtar Kelimeler: Veri Madenciliği, Kümeleme, Paralelleştirme, DBSCAN, OPTICS, LAM/MPI

ABSTRACT

Master Thesis, Parallel Data Mining By Using Density Based Clustering Methods, T.C. Maltepe University, Institute of Natural Sciences, Department of Computer Engineering.

In this master thesis, DBSCAN algorithm and OPTICS algorithm have been coded by taking the pseudo-codes; that the writers set forth in their articles, as the basis. It has been noticed that neighborhood queries take too long time on the works carried out on a single computer and R*-Tree data structure is used in order to shorten this period. LAM/MPI library has been used to parallelize DBSCAN algorithm. Neighborhood queries are the part that spends most of the runtime of the DBSCAN application, and this has been performed by equally distributing to all the computers by the help of LAM/MPI. It has been evaluated in the tests; which had been implemented by 3 different data sets, that DBSCAN algorithm is suitable for parallelization and the runtime period of DBSCAN that works parallelly is shortened in accordance with Amdahl Principle.

The general concept has been mentioned in the first section of the thesis that consists of 6 chapters. In the second chapter, general definition of data mining, its application areas, data mining process and techniques of data mining have been explained. Third chapter mainly encloses density based clustering methods that is one of the types of clustering analysis. Fourth chapter includes the objective of parallel programming, parallel computer memory architecture, parallel programming models and parallel program design. Fifth chapter consist of the development objective of parallel DBSCAN application, development environment, the tools used, the data sets and the development steps of the application. In the sixth and the last chapter, experimental results have been given with tables and graphics and attained results have been examined. Besides, suggestions have been presented for those who want to make research on this subject.

This thesis has been completed in 2010 and consists of 84 pages.

Keywords: Data Mining, Clustering, Parallelization, DBSCAN, OPTICS, LAM/MPI

TEŞEKKÜR

Tez konusunu seçmemde beni yönlendiren, tez süreci boyunca destek ve yardımlarını esirgemeyen, değerli bilgilerinden istifade ettiğim danışman hocam Yrd. Doç. Dr. Turgay Tugay BİLGİN'e, tez sürecinde bana gösterdiği olağanüstü anlayış ve yardım için nişanlım Öznur ERDUR'a, maddi ve manevi desteğini benden hiçbir zaman esirgemeyen çok değerli aileme ve çalışmalarım sırasında emeği geçen herkese teşekkürlerimi sunarım.

İÇİNDEKİLER

KISALTMALAR

Kısaltma	İngilizcesi	Türkçesi
CART	Classification And Regression Tree	Sınıflandırma ve Regresyon Ağacı
CC-NUMA	Cache Coherent Non-Uniform Access	Önbellek Uyumlu Düzenli Olmayan Erişim
CLARA	Clustering LARge Applications	Büyük Veritabanlarında Kümeleme
CLARANS	Clustering Large Applications based on RANdomized Search	Rastgele Arama Tabanlı Büyük Kümeleme Uygulamaları
CLIQUE	Clustering in QUEst	Araştırmalı Kümeleme
CPU	Central Processing Unit	Merkezi İşlem Birimi
DBSCAN	Density-Based Spatial Clustering of Applications with Noise	Gürültülü Veritabanlarında Yoğunluk Tabanlı Kümeleme
DENCLUE	Density Based Clustering	Yoğunluk Tabanlı Kümeleme
GPL	General Public License	Genel Kamu Lisansı
ID3	Iterative Dichotomiser 3	Kendi Kendini Yineleyen Sınıflandırma
KDD	Knowledge Discovery in Databases	Veritabanlarında Bilgi Keşfi
LAM/MPI	Local Area Multicomputer / Message Passing Interface	Yerel Alan Çoklu Bilgisayar / Mesaj Geçirme Arayüzü
MIMD	Multiple Instruction Multiple Data	Çoklu Komut, Çoklu Veri
MISD	Multiple Instruction Single Data	Çoklu Komut, Tek Veri
NASA	National Aeronautics and Space Administration	Ulusal Havacılık ve Uzay Dairesi
NUMA	Non-Uniform Access	Düzenli Olmayan Erişim

OPTICS	Ordering Points To Identify the Clustering Structure	Küme Yapısını Tanımlamak İçin Noktaları Sıralama
PAM	Partitioning Around Medoids	Temsilciler Etrafında Bölümleme
PCI	Peripheral Component Interconnect	Çevresel Bileşen Bağlantısı
SCSI	Small Computer System Interface	Küçük Bilgisayar Sistem Arayüzü
SETM	SET-oriented Mining of Association Rules	Birliktelik Kurallarının Kümeye Dayalı Araştırılması
SIMD	Single Instruction Multiple Data	Tek Komut, Çoklu Veri
SISD	Single Instruction Single Data	Tek Komut, Tek Veri
SLIQ	Supervised Learning In Quest	Araştırmada Denetimli Öğrenme
SMP	Symmetric Multiprocessor	Simetrik Çoklu İşlemci
SPRINT	A Scalable Parallel Classifier for Data Mining	Veri Madenciliği için Ölçeklenebilir Paralel Sınıflandırma
STING	STatistical INformation Grid	İstatistiksel Bilgi Kılavuzu
STL	Standart Template Library	Standart Şablon kütüphanesi
UMA	Uniform Memory Access	Düzenli Bellek Erişimi
WaveCluster	Clustering Using Wavelet Transformation	Küçük Dalga Dönüşümünü Kullanarak Kümeleme

ŞEKİLLER

TABLOLAR

1. GİRİŞ

Bu Yüksek Lisans tezinde, yoğunluk tabanlı kümeleme algoritmalarını kullanarak paralel veri madenciliği gerçekleştirilmiştir. Veri madenciliğinde paralel hesaplamaya ihtiyaç duyulmasının nedeni veri madenciliği uygulamalarının tek bir bilgisayar tarafından çalıştırıldığında hesaplama sürecinin çok uzun zaman almasıdır. Paralel hesaplama ile iş yükü birden çok bilgisayara paylaştırılıp eş zamanlı yürütülerek uygulamanın daha kısa zamanda tamamlanması sağlanmaktadır. Böylece yoğunluk tabanlı veri madenciliği uygulamalarından daha kısa zamanda sonuç almak mümkün olmaktadır.

Kümeleme analizi, biyoloji, tıp, antropoloji, pazarlama, ekonomi ve telekomünikasyon gibi birçok alanda, ticari veya akademik amaçlarla kullanılmaktadır. Yoğunluk tabanlı kümeleme analizi, nesneleri benzerlik veya uzaklıklarına göre kümelere ayırmayı amaçlar. Bu tezde yoğunluk tabanlı kümeleme algoritmalarının en yaygınlarından olan DBSCAN ve OPTICS algoritmaları incelenmiş ve seri olarak tek bilgisayarda çalışacak şekilde kodlanmıştır. Sonraki aşamada DBSCAN algoritmasının birden çok bilgisayarda çalışacak şekilde paralel türevi kodlanmıştır. Birden çok bilgisayar kullanarak paralel hesaplama gerçekleştirebilmek için LAM/MPI kütüphanesi kullanılmıştır. Yoğunluk tabanlı kümeleme analizinin en çok zaman alan işlemi olan komşuluk sorgularının daha hızlı yapılabilmesi için uzaysal erişim sağlayan R*-Tree veri yapısı kullanılmıştır.

Paralel DBSCAN uygulaması Maltepe Üniversitesi'ndeki hesaplama laboratuvarında bulunan bilgisayarlar üzerinde test edilmiştir. Uygulama sonuçlarının doğrulanması purity, entropy ve mutual information değerleri hesaplanarak yapılmıştır. Paralelleştirmenin sağladığı kazancın ortaya konulması amacıyla 1'den 8'e kadar ardışık sayıda bilgisayar kullanılarak uygulama çalıştırılmış ve çalışma süreleri kaydedilmiştir.

2. VERİ MADENCİLİĞİ

Günümüzde üretilen verinin miktarı gün geçtikçe artmaktadır ve bu artışın sonu görünmemektedir. Bilgi teknolojilerinin gelişmesi ile üretilen her türlü verinin saklanması mümkün hale gelmiştir. Ucuzlayan ve genişleyen disk kapasiteleri ile bu verilerin saklama maliyeti de düşmüştür. "Araştırma sonuçlarına göre dünya üzerinde kaydedilen verinin miktarı her 20 ayda bir 2 katına çıkmaktadır." [2]

Gelişen teknoloji ile bir insanın verdiği kararlar, finansal hareketleri, tüm alışverişleri, ziyaret ettiği web siteleri, hatta görüntüleri ve konuşmaları bile çeşitli sebeplerle kaydedilmektedir. Bu veriler bilimsel, toplumsal, finansal, ticari veya akla gelen her türlü türden olabilir. Kaydedilen bu kadar çok verinin tek tek incelenmesi mümkün değildir. Bu verilerin otomatik şekilde analiz edilmesi, anlamlandırılması ve bilgiye dönüştürülmesi ihtiyacı doğmuştur.

2.1. Veri Madenciliğinin Tanımı

Veri madenciliği, en basit şekilde, çok fazla veri içinden "öz" bilgiyi çıkarmaktır.

Birkaç kitapta yer alan tanımlara bakıldığında:

"Veri madenciliği, veri içindeki örüntüleri keşfetme işlemidir. Bu işlem otomatik veya (genellikle) yarı-otomatik olmalıdır. Keşfedilen örüntüler anlamlı olmalı ve bir avantaj sağlamalıdır, bu avantaj genellikle ekonomik olmaktadır." [2]

"Veri madenciliği, daha önceden bilinmeyen, geçerli ve uygulanabilir bilgilerin geniş veritabanlarından elde edilmesi ve bu bilgilerin işletme kararları verirken kullanılmasıdır." [1]

“Veri madenciliği, genellikle gözlem ile elde edilmiş geniş veri kümelerini analiz ederek, umulmadık ilişkiler bulmak, veriyi anlaşılabilir ve kullanışlı şekilde özetlemektir. Veri madenciliği çalışması ile elde edilmiş bu ilişki ve özetler çoğunlukla model ve örüntüler olarak tanımlanır. Örnek olarak, doğrusal denklemler, kurallar, kümeler, çizge ve ağaç yapıları, zaman serilerinde tekrarlı örüntülerdir.” [3]

2.2. Veri Madenciliği Uygulama Alanları

Veri madenciliğinin sağladığı avantajların fark edilmesi ile birlikte bu tekniğin kullanıldığı alanlar da artmıştır. Ticari avantaj sağlamak isteyen firmalar, bilimsel araştırma yapan bilim adamları, devlet kurumları ve avantaj sağlayacak bilginin ihtiyaç duyulduğu her alanda veri madenciliği kullanılabilir.

Günümüzde veri madenciliğinin başlıca uygulama alanları aşağıdaki gibi özetlenebilir:

2.2.1. Satış ve pazarlama

Özellikle perakende sektöründeki satış ve pazarlama stratejileri geliştirilirken veri madenciliği yaygın olarak kullanılmaktadır. Müşterilerin satın alma örüntülerinin belirlenmesinde, Müşterilerin toplumsal ve sosyal özellikleri arasındaki bağıntıların kurulmasında, alışveriş sepeti analizlerinde, müşteri ilişkileri yönetiminde, satış tahmini ve satış noktası analizlerinde, promosyon ve kampanya analizlerinde, tedarik ve mağaza yerleşim kararlarını alırken veri madenciliği kullanılır.

2.2.2. Bankacılık, sigortacılık ve borsa

Bankacılık sektöründe veri madenciliğinin en yaygın kullanım alanı kredi kartı sektörüdür. Bankalar bu tekniği kredi kartı harcamalarına göre müşteri gruplarının belirlenmesinde, kredi kartı taleplerini değerlendirme aşamasında, kredi kartı ve internet bankacılığı sistemlerindeki dolandırıcılıkların tespitinde kullanmaktadırlar.

Sigortacılık sektörü veri madenciliğini, risk analizi ve risk yönetiminde, yeni poliçe talep edecek müşterilerin tahmin edilmesinde, riskli müşteri tipinin belirlenmesinde, sigorta dolandırıcılıklarının tespitinde kullanırlar.

Finans kurumları ise genel piyasa analizlerinde, farklı finansal göstergeler arasındaki gizli bağıntıların bulunmasında, hisse senedi fiyatlarının ve borsa eğilimlerinin tahmin edilmesinde veri madenciliğini kullanırlar.

2.2.3. Tıp

Tıp alanında veri madenciliği teşhis ve tanı konması aşamasında, kanser türleri ile ilgili araştırmalar, gen araştırmaları, tedavi sürecinde veri madenciliği kullanılır.

2.2.4. Telekomünikasyon

Telekomünikasyon operatörleri hat kapasitelerini hatların yoğunluklarına göre ayarlamak zorundadırlar. Çok abonenin olduğu bölgelerde az sayıda hat varsa abonelerin hat yoğunluğundan dolayı görüşme yapamama riski vardır. Eğer abone sayısına göre hat sayısı fazla ise fazla kapasitenin atıl kalması da bir problemdir. Firmalar hat kullanım yoğunluklarını veri madenciliği ile analiz ederek kapasitelerini planlarlar.

Arama kayıtları içinde veri madenciliği yaparak en çok arama başlatılan veya en çok aranan bölgeleri belirleyerek fiyat, kampanya ve promosyon stratejilerini belirlerler. Kaybetmek üzere oldukları aboneleri veri madenciliği ile belirleyerek onları geri kazanmaya çalışırlar. Hizmet kalitelerindeki iyileştirme ihtiyaçlarını belirlerler.

2.3. Veri Tabanlarında Bilgi Keşfi Süreci

Bilgiye ulaşma süresince yapılan işlerin ve süreçlerin tamamına veri madenciliği denmemektedir. Veri tabanlarında bilgi keşfi (Knowledge Discovery in Databases - KDD) adını verdiğimiz ana süreç temelde 5 alt süreçten oluşur, bunlardan birisi de veri madenciliğidir. Bu süreçler şöyledir. [4]

- Problemin tanımlanması
- Verilerin seçilmesi ve toplanması
- Verilerin hazırlanması
    - Temizleme
    - Bütünleştirme
    - Dönüştürme
- Veri Madenciliği
- Yorumlama ve Değerlendirme

2.3.1. Problemin tanımlanması

Bilgi keşfi sürecinde başarılı olmanın ve işe yarar bilgi elde etmenin ön şartı, hangi bilgiye ihtiyaç olduğunun önceden tanımlanmasıdır. Gereksinimler önceden tanımlanırsa ve amaca yönelik çalışmalar yapılırsa sonuçta hedeflenen bilgiye ulaşmak mümkün olacaktır. Bilgi keşfi yapacak kişi veya kişiler, problemi net olarak ortaya koymalıdırlar ve cevabını almak istedikleri soruları hazırlamalıdırlar.

2.3.2. Verilerin seçilmesi ve toplanması

İstenen amaca uygun olarak gerekli olan verilerin ve bu verilerin kaynaklarının belirlenmesi adımıdır. Verilerin elde edileceği kaynak, çalışmayı yapan kurumun kendi veri tabanı olabilir, üçüncü şahısların veri tabanları olabilir. Veri tabanlarından alınan verilerin dönüştürme aşaması daha kolay olmaktadır. Veriler elektronik sensör ağlarından toplanmış, düz dosya formatında veriler olabilir, dijital imaj veya video dosyaları olabilir. Verilerin seçilmesi aşamasında önemli olan, işlenecek verinin konuyla ilgili olması, gereksiz verinin işleme girmemesidir.

2.3.3. Verinin hazırlanması

Verilerin hazırlama aşaması, verilerin veri madenciliği için hazırlanmasını kapsamaktadır ve 3 alt aşamadan oluşmaktadır. Verilerin hazırlanması aşaması bilgi keşfi sürecinin toplam zamanının çoğunu almaktadır. [4]

2.3.3.1. Temizleme

"Verilerin temizlenmesinden anlaşılacak şey, kirli veri olarak da adlandırılan kayıp ve gürültünün ortadan kaldırılmasıdır. Ayrıca yanlış ve aşırı uçta bulunan verilerin ortadan kaldırılması da verilerin temizlenmesi konusuna girer." [4]

Kayıp veriler ile ilgili yapılabilecek düzeltmeler şöyledir; "Kayıp verinin bulunduğu kaydı veritabanından çıkarmak ya da bu gibi kayıtların iptal edilmesi. Kayıp verilerin elle teker teker doldurulması. Tüm kayıp verilere aynı bilginin girilmesi. Kayıp olan verilere tüm verilerin ortalama değerinin verilmesi. Regresyon yöntemi kullanarak, diğer değişkenlerin yardımı ile kayıp olan verilerin tahmin edilmesi. " [1]

Bu aşamada eksik verilerin tamamlanması, aykırı değerlerin belirlenerek gürültünün azaltılması ve verideki tutarsızlıkların giderilmesi isabetli bir sonuca ulaşılması açısından önemlidir.

2.3.3.2. Bütünleştirme

Kullanılacak olan veriler birden fazla kaynaktan alınmış olabilir. Bu kaynaklar, veri tabanları, veri küpleri veya düz dosyalar olabileceği için farklı kaynaklardan toplanmış bu verileri birleştirme ihtiyacı doğacaktır. "Bu aşamada dikkat edilmesi gereken hususlardan birisi şema birleştirme ve nesne eşleştirme konularıdır. Bir veri tabanında customer_id olarak belirtilen bir alanın diğer bir veri tabanında cust_number olarak belirtilmesi durumunda bunların eşleştiği nasıl anlaşılacaktır? Bu tip problemleri önlemek için meta veri(veriye ilişkin veri) kullanılmalıdır." [4]

Veri birleştirme konusundaki bir diğer husus da fazlalıkların indirgenmesidir. Bu indirgemeler iki veri alanı arasında ilişki analizi yapılarak bulunabilir.

"Veri birleştirmede önemli olan üçüncü nokta ise ölçekleme ve kodlamalardaki farklılıklardır. Örneğin, ağırlık bir sistemde kilogram ile kaydedilmişken başka bir sistemde pound ile kaydedilmiş olabilir. Verilerin bu şekilde heterojen bir yapı oluşturması veri bütünlüğü açısından büyük tehlikeler oluşturmaktadır." [4]

2.3.3.3. Dönüştürme

Kullanılacak olan veri madenciliği modeline göre elimizdeki veriyi uygun formata dönüştürmek gerekebilir. Dönüştürme işlemi aşağıdakileri içerebilir: [4]

- **Düzleştirme:** Gürültüleri veriden ayrıştırmak içindir. Kutulama, Regresyon ve Kümeleme bu tekniklerdendir.

- **Birleştirme:** Verinin özetlenmesi ve kümelenmesi işlemleri uygulanır. Mesela, günlük satış verileri, aylık ve yıllık toplamlar hesaplanmak üzere birleştirilebilir. Bu aşama tipik olarak detayları analiz edilecek bir veri küpünün oluşturulmasıdır.
- **Genelleştirme:** alt seviye ya da "primitif"(ham) veriler yerine (kavramsal hiyerarşiye göre) daha üst seviye kavramların kullanılmasıdır. Mesela, cadde gibi detaylı bir özellik, şehir ya da ülke gibi daha üst seviye bir kavram ile genelleştirilebilir. Benzer şekilde yaş gibi sayısal değerler de genç, orta yaşlı veya yaşlı gibi kavramlar ile eşleştirilebilir.
- **Normalleştirme:** verinin küçük bir aralığa (mesela -1:0 – 1:0, ya da 0:0 – 1:0) denk gelecek şekilde ölçeklenmesidir.
- **Özellik oluşturma ve ekleme:** Verinin var olan özelliklerinden, veri madenciliği işleminde yardımcı olacak yeni özelliklerin oluşturulması ve eklenmesidir.

2.3.4. Veri madenciliği

Veriler üzerinde anlamlı ve yararlı örüntüler çıkarabilmek için kullanılacak olan modelin ve algoritmanın belirlenmesi bu aşamada gerçekleşmektedir. Bilgi keşfi sürecinin en önemli aşamasıdır. Hedefler doğrultusunda en uygun veri madenciliği tekniği seçilir ve uygulanır. "Her tekniğin kendine göre avantaj ve dezavantajları vardır. Bunların en fazla kullanılanları, karar ağaçları, yapay sinir ağları, demetleme algoritmaları, k-en yakın komşu algoritmaları, genetik algoritmalar, bulanık mantık ve bağ analizi olarak sayılabilir." [4]

2.3.5. Yorumlama ve değerlendirme

Veri Madenciliği sürecinde ortaya çıkan sonuçların değerlendirilmesi bu aşamada yapılmaktadır. Ortaya çıkan sonuçları görselleştirerek (grafik, tablo vb.) daha kolay anlaşılır hale getirmek gerekebilir. Sonuçlar hedeflere uygun değilse, model değiştirerek süreç tekrarlanabilir. Sonuçların ne kadar yararlı olduğu ise, ilginçlik ölçüm yöntemleri kullanılarak tespit edilebilir.

2.4. Veri Madenciliği Teknikleri

Veri madenciliği çalışmalarında kullanılan teknikler 3 temel grupta toplanır. [1]

- Sınıflama
- Kümeleme
- Birliktelik Kuralları ve İlişki Analizi

2.4.1. Sınıflama

"Sınıflama, veri sınıflarını ve kavramlarını tanımlayan ve ayırt eden bir modelin (ya da fonksiyonun) bulunması işlemidir. Bu model kullanılarak sınıf etiketleri belli olmayan nesneler sınıflandırılabilir. Elde edilen model bir grup çalışma verisinin analizine dayanır (mesela, sınıf etiketleri belli olan nesneler)." [4]

Sınıflandırma işleminde Bayesyen sınıflandırma algoritması, karar ağaçlarına dayalı algoritmalar, yapay sinir ağları temelli algoritmalar ve k-en yakın komşu algoritması gibi birçok teknik ve algoritma geliştirilmiştir.

2.4.1.1. Karar ağaçları

Karar ağacı, veriyi sınıflandırma kullanılan popüler bir veri madenciliği tekniğidir. Karar ağaçları diğer yöntemlere göre yapılandırılması ve anlaşılması daha kolaydır. "Bu teknikte sınıflandırma için bir ağaç oluşturulur; daha sonra, veritabanındaki her bir kayıt bu ağaca uygulanır ve çıkan sonuca göre de bu kayıt sınıflandırılır. Temel olarak iki adımdan oluşur: Birincisi ağacın kurulması, ikincisi de verilerin teker teker ağaca uygulanarak sınıflandırmanın gerçekleştirilmesi şeklindendir." [1]

Karar ağaçlarına dayalı olarak geliştirilen algoritmalardan bazıları, ID3 [34], CART [29], SLIQ [35], SPRINT [38], C4.5 [29] ve C5 [39]'tir. Bu algoritmalar birbirlerinden kök, düğüm ve dallanma kriteri seçimlerinde izledikleri yol açısından ayrılırlar.

2.4.1.2. Yapay sinir ağları

Yapay sinir ağları, insan beyninin fizyolojisini taklit ederler. Bir sinir ağının yapısında girişi, çıkışı ve işlem bölümleri bulunan düğümler bulunur. Her bir düğümün değeri, onu besleyen diğer düğümlerin toplam ağırlıklarından yola çıkılarak hesaplanır. Bir değeri tahmin etmede kullanılmak üzere bir model oluşturabilmek için bu düğümlerin giriş değerlerin çeşitli kombinasyonları oluşturulur.

"Yapay sinir ağları, veri modellerinin incelenmesi ve ilginç örüntü veya yapıların belirlenmesine yardımcı modellerin oluşturulması için basit hesap araçlarıdır. Bu modellerin oluşturulması için kullanılan veri çalışma verisi olarak bilinir. Bir sinir ağı bir kere çalışma verisine maruz kalınca ve bu veri içerisinde bulunan örüntüleri öğrenince yeni verilere uygulanarak sonuçlar çeşitlendirilebilir.

Yapay sinir ağları;

- Geçmiş çalışma verisi gözlemlenerek gelecek olayların *tahmin edilmesinde*,
- Görülmemiş verinin çalışma verisinde gözlemlenen karakteristik özelliklere dayanarak önceden tanımlanmış gruplara göre *sınıflandırılmasında*,
- Çalışma verisinin karakteristik özelliklerindeki benzerliklere göre doğal gruplar halinde *kümelenmesinde.*

kullanılabilir." [5]

2.4.1.3. K-en yakın komşu algoritması

"Eğer kayıtlar veri uzayında noktalar olarak yorumlanırsa, komşu kayıtlar kavramı birbirine yakın yaşayan birbirine komşu olarak tanımlanabilir. Çok-boyutlu veri uzayı benzetmesi açısından tip, bu veri uzayında bir bölgeden başka bir şey değildir. Bu anlayışa dayanarak basit ama güçlü bir öğrenme algoritması geliştirilebilir. K-en yakın komşu'nun temel felsefesi şudur: "komşularımızın yaptığı gibi yapalım". Eğer bir bireyin davranışını tahmin edilmesi isteniyorsa onun komşularının davranışlarına bakılmalıdır. K harfi araştırılan komşuların sayısını temsil eder. Basit k-en yakın komşu gerçek bir öğrenme algoritması değil daha çok bir araştırma metodudur." [6]

"En yakın komşu sınıflandırması temel olarak bütün özellik değerleri (kategorik özelliklere değiştirilebilir olmasına rağmen) kesintisiz ise kullanılır. Fikir, görülmemiş bir örneğin sınıfını, örneğin veya ona en yakın örneklerin sınıflarını kullanarak kestirebilmektir."[7]

2.4.1.4. Bayesyen sınıflandırma

"Bayesyen sınıflandırıcıları istatistiksel sınıflandırıcılardır. Bir verinin belli bir sınıfa ait olma olasılılarını tahmin ederler. Bayesyen sınıflandırması Bayes teoremine dayanır. Sınıflandırma algoritmalarını karşılaştıran çalışmalar, saf Bayesyen sınıflandırması olarak bilinen basit Bayesyen sınıflandırmasının performansını karar

ağaçları ve yapay sinir ağları ile karşılaştırmışlardır.[4] Bayesyen sınıflandırıcılar, büyük veri tabanlarında uygulandıklarında hızlı ve yüksek isabetle çalışmaktadırlar.

Saf Bayesyen sınıflandırıcılar bir sınıfın özelliklerinden birisinin etkisinin diğer özelliklerinden bağımsız olduğunu varsayarlar. Bu varsayım koşullu bağımsızlık olarak adlandırılır. Bu kural hesaplamaları basitleştirmek için kullanılır ve bu sebeple saf olarak adlandırılır.

Bayesyen güven ağları, saf Bayesyen sınıflandırıcılara benzemeyen, grafiksel modellerdir. Sınıf özelliklerinin oluşturduğu alt kümeler arasındaki bağımlılıklara izin verirler. Bayesyen güven ağları sınıflandırma için de kullanılabilirler." [4]

2.4.1.5. Regresyon analizi

"Regresyon analizinin amacı tahmin için kullanılabilecek bir veri madenciliği modeli oluşturmaktır. Böyle bir model oluşturabilmek için her birinde bir çok özellik bulunan öğretici kayıtlar verilmiştir. Bu özellikler nümerik (yaş, ücret vs.) veya kategorik (meslek, cinsiyet vs.) olabilirler. Bu özelliklerden birisi diğerlerinden ayrılmıştır ve bağımlı özellik olarak adlandırılır. Diğer özellikler tahminci(predictive) özelliklerdir. Eğer bağımlı özellik kategorik ise problem bir sınıflandırma problemidir. Eğer nümerik ise problem bir regresyon problemidir. Sınıflandırma ve regresyonun amacı bir kaydın bilinmeyen değerini(bağımlı özellik) tahmin eden bir veri madenciliği modeli oluşturmaktır." [5]

2.4.2. Kümeleme

"Kümeleme analizi, sınıflandırmada olduğu gibi sahip olunan verileri gruplara ayırma işlemidir. Sınıflandırma işleminde, sınıflar önceden belirli iken kümelemede sınıflar önceden belirli değildir. Verilerin hangi kümelere ve kaç değişik gruba

ayrılacağı eldeki verilerin birbirlerine olan uzaklığa göre belirlenir. Kümeleme analizi biyoloji, tıp, antropoloji, pazarlama, ekonomi ve telekomünikasyon gibi birçok alanda kullanılmaktadır.” [1]

Tezin 3. bölümünde kümeleme analizi detaylı olarak anlatılacaktır.

2.4.3. Birliktelik kuralları ve ilişki analizi

“Birliktelik kuralı X → Y şeklinde ifade edilir. Bu ifadede X nesneler kümesi, Y ise tek bir nesnedir. Birliktelik kuralı yöntemleri genellikle çok büyük veri kümelerinde uygulanırlar. Bu yöntemin kullanıldığı en yaygın alan alışveriş sepeti analizidir. Daha verimli algoritmalar üzerinde çalışmalar devam etmektedir.[28] İlişki analizi önemli ilişkileri bulmak açısından önemlidir. İlaç sektörü ve sağlık sigortası dolandırıcılığı gibi bir çok alanda uygulanmaktadır.” [8]

Bu alandaki en popüler algoritmalar, Apriori [40], AprioriTid [40] ve SETM [41] algoritmalarıdır.

3. KÜMELEME ANALİZİ

"Kümeleme analizi, bazı birliktelik ölçülerini kullanarak örnek verilerin gruplanarak otomatik sınıflandırmasını yapan yöntemlerin genel adıdır. Bir gruptaki örnekler benzerdir, farklı gruplardakiler benzer değildir. Kümeleme analizinin girdisi örnek verilerden oluşan küme ve iki örnek arasındaki benzerlik (veya benzersizlik) ölçütüdür. Kümeleme analizinin çıktısı ise birbirine benzer örneklerden oluşan gruplardır (kümeler)." [9]

"Kümelemenin sınıflandırmadan farkı kümeleme için hedef bir değişken olmamasıdır. Kümeleme işi hedef değişkenin değerini sınıflandırmaya, tahmin etmeye veya öngörmeye çalışmaz. Kümeleme algoritmaları, tüm veri setini -göreceli olarak- homojen alt gruplara veya kümelere böler. Bu kümelerdeki kayıtların(verilerin) benzerliği en yüksek, farklı kümelerdeki kayıtların da benzerliği en düşük seviyede tutulamaya çalışılır." [10]

"Literatürde birçok kümeleme algoritması vardır. Yeni bir yöntem geliştirip bunu literatüre sunmak giderek zorlaşmaktadır çünkü her yeni algoritma eskilerden bir veya birkaç özellik içermekte ve birbirleriyle kesişmektedirler. Bununla birlikte, eski algoritmaları farklı organize edilmiş ve yorumlanmış şekilde sunmak faydalı olmaktadır. Genel olarak kümeleme yöntemleri şu kategorilere ayrılmıştır." [4] :

- Bölümlemeli Yöntemler
- Hiyerarşik Yöntemler
- Yoğunluk Tabanlı Yöntemler
- Grid Tabanlı Yöntemler
- Model Tabanlı Yöntemler
- Diğer Yöntemler

3.1. Bölümlemeli Yöntemler

"Bölümlemeli kümeleme algoritmaları, n tane nesne ve k değerini girdi olarak alır ($k \leq n$ olacak şekilde), n nesneyi k tane gruba (kümeler) ayırırlar. Kümeler, bölümleme kriterini en iyi şekilde karşılamak üzere ayarlanmıştır. Bu bölümleme kriterine genellikle benzerlik fonksiyonu adı verilir. Benzerlik fonksiyonları iki nesne arasındaki uzaklığı hesaplarlar. Bir kümedeki nesneler birbirlerine daha benzer (daha yakın), farklı kümelerdeki nesneler birbirlerinden daha farklıdır (daha uzak).

Bölümlemeli kümeleme algoritmaları, dendogram oluşturması mümkün olmayan büyük veri setleri içeren uygulamalarda avantaj sağlar. Bölümlemeli algoritmalarla birlikte gelen bir problem ise çıktı olarak istenen kümelerin sayısını belirlemektir. Bölümleme teknikleri, genellikle yerel olarak veya global olarak tanımlanan kriter fonksiyonuna göre en iyi kümeleri oluşturur. Algoritma birden fazla defa ve değişik giriş parametreleriyle çalıştırılarak en iyi sonuç alınabilir. Bilinen en iyi ve yaygın olarak kullanılan algoritmalar k-means, k-medoids ve bunların varyasyonlarıdır." [5]

- K-means [42]
- PAM [31]
- CLARA [31]
- CLARANS [30]

3.2. Hiyerarşik Yöntemler

"Hiyerarşik kümeleme yöntemi, nesnelerin bir ağaç yapısına yerleştirilmek üzere gruplanması şeklinde çalışır. Hiyerarşik kümeleme, kullanılan ayrıştırma tekniğinin aşağıdan yukarıya (birleştirme) veya yukarıdan aşağıya (ayrıştırma) oluşuna göre toplaşımlı ya da bölünür olarak sınıflandırılabilir. Saf bir hiyerarşik kümelemenin kalitesi, birleştirme veya ayrıştırma kararı bir kere uygulandıktan sonra düzeltme

yapılamaması nedeniyle düşer. Belirli bir birleştirme veya ayrıştırma kararı eğer kötü bir tercihse, bu yöntem geri alınamaz ve düzeltilemez. Güncel çalışmalar, hiyerarşik toplanmaların tekrarlanan yerleştirme yöntemleri kullanılarak entegre edilmesini vurgulamaktadır." [4]

3.2.1. Toplaşımlı (Agglomerative) yöntemler

"Toplaşımlı hiyerarşik kümeleme bir aşağıdan yukarıya kümeleme yöntemidir. Toplaşımlı algoritmaların asıl amacı, bir grup kümenin bazı mesafe hesaplamalarına dayanarak tekrarlı olarak birleştirilmesidir. Bir toplaşımlı algoritma başlangıçta her bir veri nesnesini bir küme kabul eder. Her başarılı tekrarlamada algoritma, bazı mesafe kriterlerini sağlayan en yakın küme çiftlerini bir araya toplar.

Toplaşımlı algoritmaların en önemli kısmı kümeler arasındaki mesafelerin hesaplanmasıdır." [11]

"K-means [42] kümelemesi için iki nesne arasındaki mesafeyi ölçmek için bir yol seçilmesi gerekmektedir. Bu yöntem için yaygın olarak kullanılan mesafe ölçüm tekniği Öklid uzaklığıdır. İki boyutta, Öklid uzaklığı sadece iki nokta arasındaki bir 'düz çizgi'dir.

Toplaşımlı hiyerarşik kümeleme arkasında basit bir fikir vardır. Başlangıçta her nesne bir küme olarak kabul edilir ve birbirine yakın küme çiftleri birleştirilerek devam edilir. Temel olarak algoritma aşağıdaki gibidir:" [7]

1. Her nesneyi bir kümeye ata. Küme çiftleri arasındaki mesafeyi hesapla.
2. En yakın küme çiftlerini seç ve tek bir küme olarak birleştir. (böylece küme sayısını bir azaltır)
3. Yeni kümenin her bir eski küme ile mesafesini hesapla.
4. 2 ve 3 numaralı adımları bütün nesneler tek bir küme içerisinde olana kadar tekrarla.

3.2.2. Bölünür (Divisive) yöntemler

“Bölünür hiyerarşik kümeleme daha az yaygın bir yukarıdan aşağıya kümeleme yöntemidir. Toplaşımlı kümeleme yöntemine benzer bir şekilde fakat tam ters yönde çalışır. Bölümlemeli hiyerarşik kümeleme bütün nesneleri içeren tek bir küme ile başlar. Başlangıç kümesi, bir kümedeki nesneler diğer kümedeki nesnelere göre uzakta olacak şekilde ikiye bölünür. Yöntem daha sonra, oluşan kümeleri her nesne kendi kümesinde olana kadar böler.

Tekli(monothetic) yöntemler ve çoklu(polythetic) yöntemler bölümlemeli hiyerarşik kümelemenin iki çeşididir. Tekli yöntemler, nesneleri belirlenmiş tek bir niteliğini değerlendirerek ayırır. Çoklu yöntemler ise nesnelerin birçok niteliğini değerlendirir. Tekli yöntemler genellikle çoklu yöntemlere göre daha etkilidir ancak yetersiz sonuçlar vermeye eğilimlidir.” [11]

3.3. Grid Temelli Yöntemler

“Grid hesaplama (veya hesaplanabilen gridlerin kullanılması) birden fazla bilgisayarın kaynaklarının ortak bir işi yapmak için birleştirilmesidir. Genellikle, yüklü bilgisayar işlemlerini veya çok fazla verinin işlenmesini gerektiren bilimsel, teknik ve ticari problemlerde kullanılır.

Grid temelli kümeleme yaklaşımı çok çözünürlüklü grid veri yapısını kullanır. Veri uzayını üzerinde bütün kümeleme işlemlerinin gerçekleştirileceği grid yapısını oluşturan sonlu sayıda hücreye böler. Bu yaklaşımın en büyük avantajı işlem süresinin kısa olmasıdır. Bu işlem süreci veri nesnelerinin sayısından bağımsız, sadece bölünmüş olduğu hücrelerin sayısına bağımlıdır. İstatistik bilgilerini grid hücrelerine yerleştirip inceleyen STING [36], nesneleri Wavelet [43] dönüşüm yöntemini kullanarak kümeleyen WaveCluster [37], ve grid ve yoğunluk temelli

yaklaşımlar kullanarak çok boyutlu uzay nesnelerini kümeleyen CLIQUE [32], grid-temelli yaklaşıma örnek olarak verilebilir.” [4]

3.4. Yoğunluğa Dayalı Yöntemler

“Farklı şekillerdeki kümeleri belirlemek için yoğunluğa dayalı algoritmalar geliştirilmiştir. Bazı değişik şekilli kümelerin k-means algoritması gibi sadece noktalar arasındaki Öklid uzaklığına dayanarak belirlenmesi zordur. Bu tür kümelerin tespit edilmesinde uç verilerin (outliers) ayıklanması da gerekmektedir. Çünkü aslında hiçbir kümeye ait olmayan bu uç veriler kümelerin belirlenmesi için yapılan hesaplamaları etkileyeceğinden gerçek kümeler ortaya çıkartılamayacaktır.” [1]

“Uydu fotoğraflarının ve X-ray kristalleşmelerinin analizi gibi araştırmalar yapılan birçok alanda uzaysal (spatial) veriler kullanılmaktadır. Uzaysal veri tabanlarında otomatik bilgi keşfi günden güne önem kazanmaktadır. Sınıf tanımlama işi, veri nesnelerini anlamlı şekilde gruplamakla yapılmaktadır. Kümeleme algoritmaları sınıf tanımlama işlerinde kullanılmaktadır.” [12]

En popüler yoğunluğa dayalı kümeleme algoritmaları DBSCAN [12], OPTICS [13] ve DENCLUE [33]’dur.

3.4.1. DBSCAN

DBSCAN algoritması Münih Üniversitesi’nden Prof. Dr. Hans-Peter Kriegel ve ekibindeki Martin Ester, Jörg Sander, Xiaowei Xu tarafından geliştirilmiştir. 1996 tarihli “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise” başlıklı makalede ortaya konmuştur. Değişik biçimlerdeki çok büyük veri tabanlarının kümelenmesi amacıyla geliştirilmiştir. Geliştiren grup,

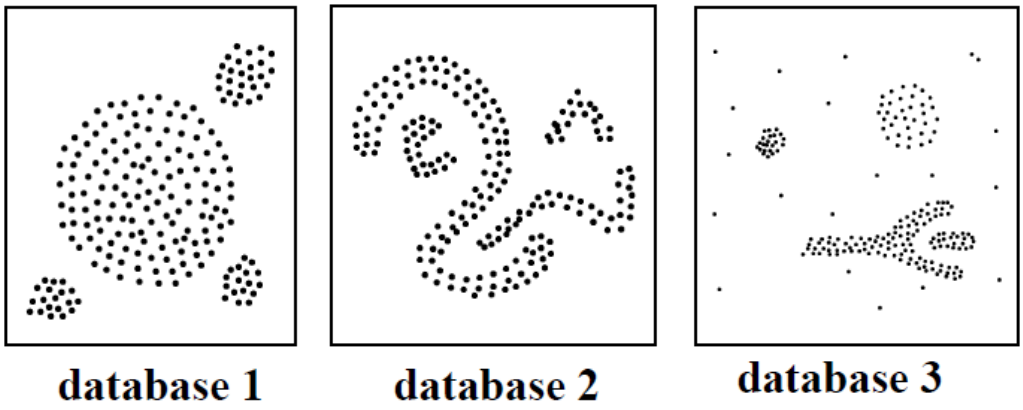
DBSCAN'in değişik şekilleri bulmada CLARANS[30] algoritmasından daha etkili olduğunu ve CLARANS'tan daha verimli olduğunu açıklamaktadır.

"Kümeleme algoritmaları büyük uzaysal veritabanlarına uygulandıklarında bazı isterler ortaya çıkmaktadır. [12]

1. Girdi parametrelerini belirlemek için veri kümesi ile ilgili bilgi sahibi olunması gerekmektedir. Çünkü büyük veritabanlarıyla uğraşırken uygun değerler önceden bilinememektedir.
2. Sıra dışı şekillerdeki kümelerin bulunması uzaysal veritabanlarındaki kümelerin şekillerinin küresel, uzatılmış, doğrusal ve ince uzun olabilmesinden dolayı zor olabilir.
3. Birkaç binden fazla nesne içeren veritabanlarıyla çalışırken verimlilik önem kazanmaktadır."

DBSCAN'in geliştiricileri DBSCAN'i yayınladıkları güne kadar yukarıdaki isterleri karşılayan bir kümeleme algoritmasının bulunmadığını, fakat DBSCAN'in bu isterleri karşıladığını belirtmişlerdir.

Şekil 3.1'deki örnek veritabanlarına bakıldığında kolaylıkla noktaların oluşturduğu kümeleri ve bu kümelerin hiçbirisine ait olmayan gürültü noktaları görülebilir.



Şekil 3.1 Örnek veritabanları [12]

“Bu kümelerin bariz bir şekilde görebilmesinin sebebi, kümelerin her birinde dışarısına oranla belli bir yoğunluk olmasıdır. Aynı zamanda gürültülerin bulunduğu bölgedeki yoğunluk da kümelerin bulunduğu bölgelerden daha azdır.” [12]

DBSCAN makalesinde algoritma sunulmadan önce bazı tanımlar yapılmıştır. Bu tanımlar şöyledir:

Tanım 1: (Bir noktanın Eps komşuluğu)

Bir p noktasının Eps komşuluğu $N_{Eps}(p) = \{ q \in D \mid dist(p,q) \leq Eps \}$ ile tanımlanır.

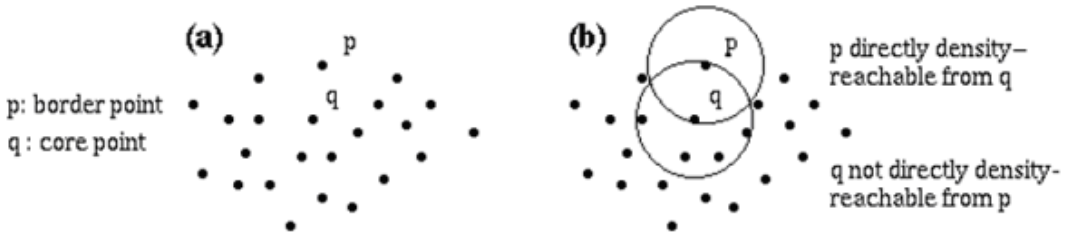
Bu tanımdaki dist(p,q) fonksiyonu p ve q noktalarının birbirine olan uzaklığını hesaplayan fonksiyonu ifade eder. Bu tanıma uyarak, bir kümedeki her noktanın Eps komşuluğunda minimum sayıda (MinPts) nokta olduğu söylenebilir. Ancak bu önerme geçersizdir çünkü kümedeki noktalar, iç noktalar (core points) ve sınır noktalar (border points) olmak üzere ikiye ayrılmaktadır. Sınır noktaların Eps komşuluğunda iç noktaların Eps komşuluğunda olduğundan daha az nokta vardır.

Tanım 2: (Doğrudan yoğunluğa erişilebilirlik)

Bir p noktası, bir q noktasından aşağıdaki koşulları sağlarsa doğrudan yoğunluğa erişilebilirdir (directly density-reachable).

1. $p \in N_{Eps}(q)$
2. $| N_{Eps}(q) | \geq MinPts$ (iç nokta olma durumu)

Doğrudan yoğunluğa erişilebilirlik kuralı iç noktalar için simetriktir fakat bir sınır nokta ve iç nokta için aynı şey geçerli değildir. Şekil 3.2 bu durumu göstermektedir.



Şekil 3.2 İç nokta ve sınır noktalar [12]

Tanım 3: (Yoğunluğa erişilebilirlik)

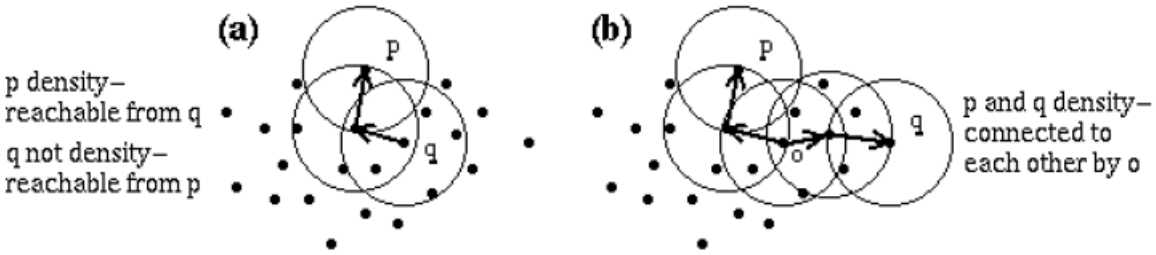
Bir p noktası q noktasından yoğunluk erişilebilir (density reachable) denilebilmesi için $p_1 \ldots p_n$ noktalar zincirinde, $p_1 = q$, $p_n = p$ ve p_{i+1} p_i'den direk yoğunluk erişilebilir olmalıdır.

Yoğunluk erişilebilirliği, doğrudan erişilebilirliğin uzantısıdır. Bu ilişki geçişlidir fakat simetrik değildir. Şekil 3.3 asimetrik durumları göstermektedir.

Tanım 4: (Yoğunluk bağlantısallığı)

Eğer bir o noktası hem p noktasına hem de q noktasına yoğunluk erişilebilir durumdaysa, p ile q noktaları birbirlerine yoğunluk bağlantılıdır (density-connected).

Yoğunluk bağlantısallığı simetrik bir ilişkidir. Yoğunluk erişilebilir noktalar için yoğunluk bağlantısallığı da vardır.



Şekil 3.3 Yoğunluğa erişilebilirlik ve yoğunluk bağlantısallığı [12]

Tanım 5: (Küme)

D, noktalardan oluşan bir veritabanı ve C kümesi ise D'nin boş olmayan altkümesi olarak şu koşulları sağlamalıdır:

1. $\forall$ p, q: eğer p ϵ C ve q noktası p noktasından yoğunluk erişilebilir durumdaysa q ϵ C.
2. $\forall$ p, q ϵ C: p noktası q noktasına yoğunluk bağlantılıdır.

Tanım 6: (Gürültü)

C_1 ,…, C_k kümeleri D veritabanının kümeleri olsun, gürültü noktaları D veritabanında hiçbir kümeye ait olmayan noktalara denir. Gürültü = { p ϵ D | $\forall$ i: p $\notin C_i$}

DBSCAN algoritması Tanım 5'te tanımlanan kümeleri ve Tanım 6'da tanımlanan gürültü noktaları bulmak için tasarlanmıştır. Bir kümeyi bulabilmek için DBSCAN, veritabanından bir p noktası seçer ve bu p noktasından yoğunluk erişilebilir olan tüm noktaları bulur. Eğer p noktası bir iç nokta ise bu işlem bir küme bulunması ile sonlanır. Eğer p noktası bir sınır nokta ise veya p noktası hiçbir noktadan yoğunluk erişilebilir değilse DBSCAN veritabanındaki bir sonraki noktayı ziyaret eder.

DBSCAN algoritmasının sözde kodu aşağıdaki gibidir.

```
DBSCAN (SetOfPoints, Eps, MinPts)
// SetOfPoints is UNCLASSIFIED
   ClusterId := nextId(NOISE);
   FOR i FROM 1 TO SetOfPoints.size DO
      Point := SetOfPoints.get(i);
      IF Point.ClId = UNCLASSIFIED THEN
         IF ExpandCluster(SetOfPoints, Point, ClusterId,
         Eps, MinPts) THEN
            ClusterId := nextId(ClusterId)
         END IF
      END IF
   END FOR
END; // DBSCAN
```

Algoritmanın girdilerinden `SetOfPoints` bütün veritabanı veya algoritmanın bir önceki çalışmasında bulunan bir küme olabilir. `Eps` ve `MinPts` yoğunluk parametreleridir. `SetOfPoints.get(i)` veritabanındaki i'inci elemanı getirir.

DBSCAN'in en önemli fonksiyonu olan `ExpandCluster` aşağıda sunulmuştur.

```
ExpandCluster(SetOfPoints, Point, ClId, Eps, MinPts) :
Boolean;
   seeds:=SetOfPoints.regionQuery(Point,Eps);
   IF seeds.size<MinPts THEN // no core point
      SetOfPoint.changeClId(Point,NOISE);
      RETURN False;
   ELSE // all points in seeds are density-
      // reachable from Point
      SetOfPoints.changeClIds(seeds,ClId);
      seeds.delete(Point);
      WHILE seeds <> Empty DO
         currentP := seeds.first();
         result := SetOfPoints.regionQuery(currentP,
         Eps);
         IF result.size >= MinPts THEN
            FOR i FROM 1 TO result.size DO
               resultP := result.get(i);
               IF resultP.ClId IN {UNCLASSIFIED, NOISE}
               THEN
                  IF resultP.ClId = UNCLASSIFIED THEN
                     seeds.append(resultP);
```

```
                END IF;
              SetOfPoints.changeClId(resultP,ClId);
              END IF; // UNCLASSIFIED or NOISE
            END FOR;
          END IF; // result.size >= MinPts
          seeds.delete(currentP);
      END WHILE; // seeds <> Empty
      RETURN True;
   END IF
END; // ExpandCluster
```

`SetOfPoints.regionQuery(Point, Eps)` fonksiyonu `Point` parametresi ile gelen noktanın `Eps` komşuluğundaki noktaların listesini döndürür. Bu tür komşuluk sorgularında R*-Tree yapıları gibi uzaysal verilere erişim yöntemleri kullanılmaktadır. R*-Tree veri yapısı tezin 5.2.4 bölümünde incelenecektir.

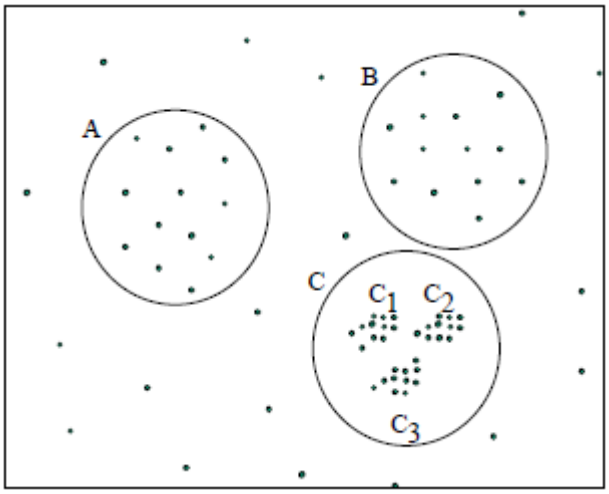
N noktalı bir veritabanında R*-Tree veri yapısı üzerinden komşuluk sorgusu yapmanın karmaşıklık derecesi O(log n) olarak belirtilmiştir. Veri tabanındaki her bir nokta için en fazla bir defa komşuluk sorgusu yapılabileceğini düşünerek DBSCAN algoritmasının karmaşıklığı O(n * log n) olduğu söylenebilir. [12]

3.4.2. OPTICS

OPTICS algoritması da DBSCAN algoritması gibi Münih Üniversitesi'nden Hans-Peter Kriegel ve ekibi tarafından geliştirilmiştir. Ekip, Mihael Ankerst, Markus M. Breunig ve Jörg Sander'den oluşmaktadır. 1999 tarihli ve "OPTICS: Ordering Points To Identify the Clustering Structure" başlıklı makale ile ortaya konmuştur.

OPTICS algoritması, DBSCAN algoritmasının geliştirilmiş halidir ve DBSCAN algoritmasının en önemli dezavantajı olan Eps ve Minpts değerlerine göre sonucun değişmesi durumuna çözüm getirmeyi amaçlar. OPTICS, diğer kümeleme algoritmalarında olduğu gibi veritabanındaki nesneleri kümelere ayırmak yerine onları belli bir kümeleme sırasına göre sıralamayı amaçlar.

“Birçok gerçek veritabanının önemli bir özelliği içlerindeki esas küme yapısının genel yoğunluk parametreleri ile belirlenememesidir. Veri uzayında birçok değişik yoğunluktaki yerel kümeleşmeler bulunabilir.” [13]



Şekil 3.4 Farklı yoğunluk parametrelerine göre kümeler [13]

“Sekil 3.4’de A, B, C1, C2 ve C3 kümelerini tek bir genel yoğunluk parametresi ile ayırmak mümkün değildir. Genel bir parametre ile A, B ve C kümeleri bulunabilir. Ya da C1, C2 ve C3 kümeleri bulunur, A ve B kümeleri ise gürültü olarak işaretlenir. Bu problemin üstesinden gelmek için özel bir sıralama ile veritabanındaki nesneleri sıralayarak tüm yoğunluk seviyelerindeki kümeleri bulan bir algoritma çalıştırılmalıdır.” [13]

OPTICS makalesinde de DBSCAN makalesinde olduğu gibi algoritma ile ilgili detay verilmeden önce bazı kavramların tanımları yapılmıştır.

Tanım 1: (Doğrudan yoğunluğa erişilebilirlik)

D kümesindeki bir p nesnesinin ε ve MinPts değerlerine bağlı olarak q nesnesi ile direk yoğunluk erişilebilir olabilmesi için

1. $p \in N\varepsilon(q)$ ($N\varepsilon(q)$ ifadesi q noktasının ε komşuluğunu ifade eder.)

2. $Card(N\varepsilon(q)) \geq MinPts$ ($Card(N)$ ifadesi N kümesinin eleman sayısını ifade eder.)

Tanım 2: (Yoğunluğa erişilebilirlik)

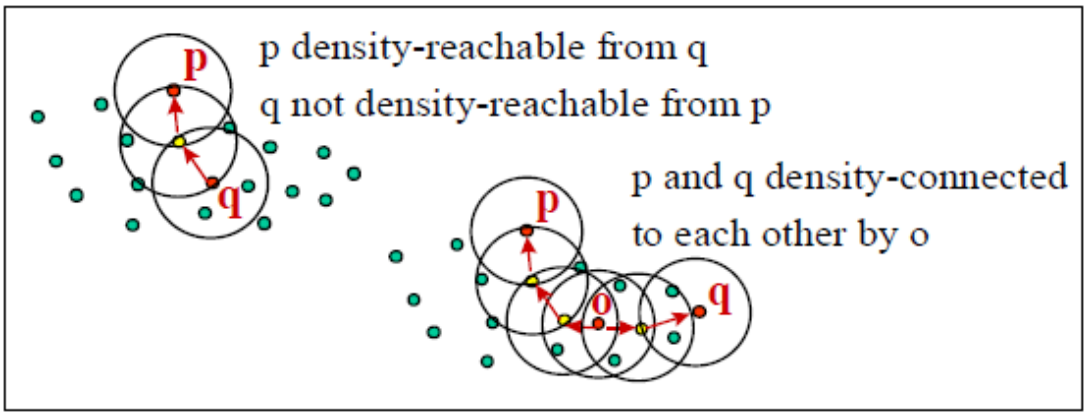
Bir p nesnesi ε ve MinPts değerlerine bağlı olarak q nesnesinden yoğunluk erişilebilir olması için D veritabanında $p_1, ..., p_n, p_1 = q, p_n = p$ şeklindeki nesneler zincirinde her p_i nesnesi p_{i+1} nesnesi ile direk yoğunluk erişilebilir olması gerekmektedir.

Yoğunluk erişilebilirliği genel olarak simetrik bir ilişki değildir, sadece iç noktalar yoğunluk erişilebilirdir.

Tanım 3: (Yoğunluk bağlantısallığı)

Bir p nesnesinin ε ve MinPts değerlerine bağlı olarak q nesnesi ile yoğunluk bağlantısallığı olabilmesi için D veritabanında bulunan bir o noktasının p ve q noktalarından yoğunluk erişilebilir olması gerekmektedir. Yoğunluk bağlantısallığı simetrik bir ilişkidir.

Şekil 3.5'te yoğunluğa erişilebilirlik ve yoğunluk bağlantısallığı gösterilmiştir.



Şekil 3.5 Yoğunluğa erişilebilirlik ve yoğunluk bağlantısallığı [13]

Tanım 4: (Küme ve gürültü)

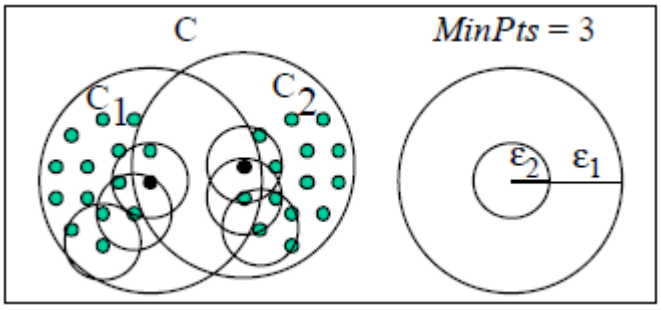
C kümesi ε ve MinPts değerlerine bağlı olarak D veritabanının boş olmayan alt kümesidir ve şu koşulları sağlar.

1. ∀ p, q: eğer p ϵ C ve q nesnesi p nesnesinden yoğunluk erişilebilir durumdaysa q ϵ C.
2. ∀ p, q ϵ C: p nesnesi q nesnesine yoğunluk bağlantılıdır.

Hiçbir kümeye ait olmayan nesneler gürültü olarak tanımlanır.

Bu tanımların ardından makalede yoğunluk tabanlı küme sıralama (density-based cluster-ordering) kavramı ortaya konmuştur. Bu kavramı ortaya koyabilmek için sabit MinPts değeri ile ε parametresinin büyük ve küçük değerlerinde ortaya çıkan durumlar karşılaştırılmıştır.

Şekil 3.6'da C_1 ve C_2 farklı kümeleri $\varepsilon_1 < \varepsilon_2$ değerlerine bağlı olarak kümelendiğinde, yüksek değer olan ε_1 ile C kümesinin tamamı bir küme olarak bulunmaktadır. ε_2 ile C_1 ve C_2 kümeleri ayrı ayrı olarak bulunmaktadır.



Şekil 3.6 İç içe yoğunluk temelli kümelerin gösterimi [13]

OPTICS algoritması, DBSCAN algoritmasını aynı anda birçok uzaklık parametresi aynı anda işlenebilecek şekilde geliştirmektedir. Bu şekilde farklı yoğunluktaki kümeler aynı anda bulunabilmektedir. Tutarlı bir sonuç elde edebilmek için nesneler işlenirken özel bir sıra oluşturulmasına gerek duyulmuştur. En küçük ε değeri ile yoğunluk erişilebilir nesneleri önce işlemek yoğunluğu fazla olan kümelerin daha önce bitirilmesi bakımından önemlidir.

Algoritma, ε adı verilen bir üretici uzaklık değerini (generating distance) kullanarak $0 < \varepsilon_i < \varepsilon$ aralığındaki sonsuz sayıdaki ε_i değerleri için DBSCAN çalıştırılmış gibi sonuç verir. Tek fark nesnelerin kümelere atanmamasıdır, bunun yerine nesnelerin işleme sırası ve daha sonra küme etiketlerinin atanabilmesi için 2 uzaklık bilgisi saklanır. Bu uzaklıklar aşağıdaki tanımlarda anlatılmıştır.

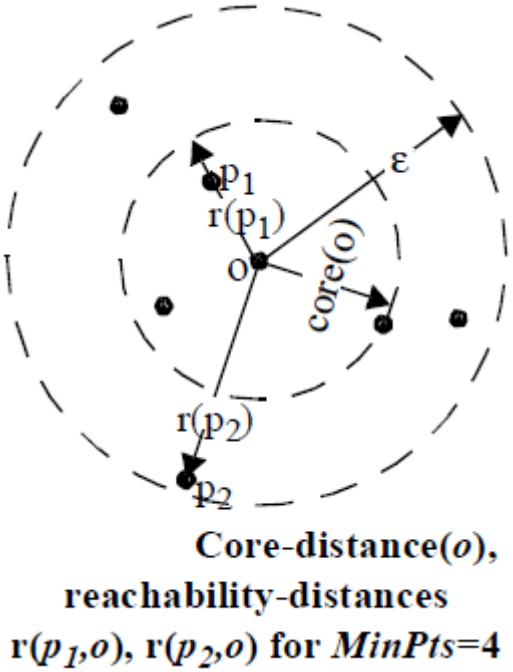
Tanım 5: (p nesnesinin iç-mesafesi)

P nesnesi D veritabanının bir elemanı ve ε mesafe değeri olsun. Bu durumda p nesnesini $N_\varepsilon(p)$ içinde bir iç nesne olarak kabul ettirecek, p nesnesi etrafında bulunan başka nesnelerle olan minimum ε mesafesi iç-mesafe olarak tanımlanır. Eğer p nesnesi bir iç-nesne değilse, p nesnesinin iç mesafesi de tanımsızdır.

Tanım 6: (p nesnesinin o nesnesine göre ulaşılabilirlik mesafesi)

Bir p nesnesinin veritabanı içindeki başka bir o nesnesine olan ulaşılabilirlik mesafesi, p nesnesini bir iç nesne olan o nesnesi üzerinden yoğunluğa eriştirecek mesafenin en küçük değeridir. Bu durumda ulaşılabilirlik mesafesi, o nesnesinin iç-mesafesinden daha küçük olamaz. Çünkü daha küçük değerler için hiçbir nesne, o nesnesi üzerinden doğrudan yoğunluğa erişilebilir olmayacaktır.

Şekil 3.7'de iç mesafesi ve ulaşılabilirlik mesafesi gösterilmektedir.



Şekil 3.7 İç nokta ve ulaşılabilirlik mesafeleri [13]

OPTICS algoritması, veritabanındaki nesneleri sıralar ve her bir nesne için iç-mesafe ve ulaşılabilirlik mesafesi değerlerini hesaplar ve kaydeder. Algoritmanın ana döngüsü aşağıda gösterilmiştir.

```
OPTICS (SetOfObjects, e, MinPts, OrderedFile)
   OrderedFile.open();
   FOR i FROM 1 TO SetOfObjects.size DO
      Object := SetOfObjects.get(i);
      IF NOT Object.Processed THEN
         ExpandClusterOrder(SetOfObjects, Object, e,
               MinPts, OrderedFile)
   OrderedFile.close();
END; // OPTICS
```

Algoritma başlangıçta OrderedFile adlı boş dosyayı yazma modunda açar, döngü bittiğinde ise kapatır. SetOfObjects adlı veritabanından seçilen nesneler eğer henüz işlenmemişlerse ExpandClusterOrder adlı fonksiyona gönderilirler. ExpandClusterOrder adlı fonksiyonun sözde kodu aşağıda sunulmuştur.

```
ExpandClusterOrder(SetOfObjects, Object, e, MinPts,
OrderedFile)
   neighbors := SetOfObjects.neighbors(Object, e);
   Object.Processed := TRUE;
   Object.reachability_distance := UNDEFINED;
   Object.setCoreDistance(neighbors, e, MinPts);
   OrderedFile.write(Object);
   IF Object.core_distance <> UNDEFINED THEN
      OrderSeeds.update(neighbors, Object);
      WHILE NOT OrderSeeds.empty() DO
         currentObject := OrderSeeds.next();
         neighbors:=SetOfObjects.neighbors(currentObject,
         e);
         currentObject.Processed := TRUE;
         currentObject.setCoreDistance(neighbors, e,
         MinPts);
         OrderedFile.write(currentObject);
         IF currentObject.core_distance<>UNDEFINED THEN
            OrderSeeds.update(neighbors, currentObject);
END; // ExpandClusterOrder
```

Bu fonksiyon Object parametresi ile gönderilen parametrenin ε komşuluğundaki noktaları bulur ve neighbors adlı listeye ekler. Object adlı nesnenin ulaşılabilirlik mesafesini tanımsız olarak tanımlar ve iç-mesafesini setCoreDistance fonksiyonu ile hesaplar. Daha sonra nesne dosyaya yazılır.

If koşulu ile nesnenin iç nesne özelliğine sahip olup olmadığı kontrol edilir, eğer değilse ana döngüye geri dönülüp bir sonraki nesne ile çalışmaya devam edilir. Eğer nesne bir iç nesne ise ε ve MinPts değerlerine göre direk yoğunluk erişilebilirliği olan noktalar tekrarlı bir şekilde toplanır. Object adlı nesneden direk yoğunluk erişilebilir olan noktalar, daha sonra genişlemek üzere OrderSeeds adı verilen listeye eklenirler.

OrderSeeds listesindeki nesneler, en yakın iç nesneye yakınlıklarını belirten ulaşılabilirlik mesafesine göre sıralı olarak listede saklanırlar. While döngüsünün her adımında OrderSeeds.next() adlı fonksiyon ile listedeki en küçük ulaşılabilirlik mesafesine sahip nesne seçilir ve currentObject adlı değişkene atanır. Bu nesnenin de ε komşuluğundaki nesneler bulunur ve iç mesafesi belirlenir. Daha sonra bu nesne iç mesafesi ve ulaşılabilirlik mesafesi ile beraber dosyaya yazılır.

Eğer currentObject bir iç nesne ise OrderSeeds listesine eklenebilecek durumda olan aday nesneler olabilir. Bu nesnelerin ulaşılabilirlik mesafesini bulan ve listeye ekleyen fonksiyon OrderSeeds.update() fonksiyonudur ve sözde kodu aşağıda sunulmuştur.

```
OrderSeeds::update(neighbors, CenterObject)
   c_dist := CenterObject.core_distance;
   FORALL Object FROM neighbors DO
      IF NOT Object.Processed THEN
         new_r_dist:=max(c_dist,CenterObject.dist(Object)
         );
         IF Object.reachability_distance=UNDEFINED THEN
            Object.reachability_distance := new_r_dist;
            insert(Object, new_r_dist);
         ELSE // Object already in OrderSeeds
            IF new_r_dist<Object.reachability_distance
            THEN
               Object.reachability_distance :=
               new_r_dist;
               decrease(Object, new_r_dist);
END; // OrderSeeds::update
```

Update fonksiyonuna gelen parametrelerden birisi olan neighbors'daki nesnelerin ulaşılabilirlik mesafeleri diğer bir parametre olan CenterObject adlı nesneye göre belirlenir. OrderSeeds listesine girmemiş olan nesneler ulaşılabilirlik mesafeleri ile beraber listeye eklenirler. Zaten listede olan nesneler ise eğer yeni ulaşılabilirlik mesafeleri eskisinden küçük ise ulaşılabilirlik mesafeleri yenisi ile değiştirilir.

OPTICS algoritmasının yapısal olarak DBSCAN algoritmasına benzemesi nedeniyle çalışma zamanları da yaklaşık olarak aynıdır. Her iki algoritmanın da çalışma süresi yapılan komşuluk sorgusu sayısına bağlıdır. N elemanlı bir veritabanında komşuluk sorgusu sıralı arama ile çalıştırılırsa algoritmanın karmaşıklığı $O(n^2)$ olmaktadır. Eğer R*-Tree veya X-Tree gibi uzaysal verilerin indexli bir şekilde tutulduğu bir veri yapısı kullanılırsa karmaşıklık O(n log n) olmaktadır. [13]

OPTICS algoritması çıktı olarak nesneler küme numarası vermemektedir. Eğer veritabanındaki nesnelere küme numarası verilmesi gerekiyorsa ayrıca çalıştırılması gereken ExtractDBSCAN-Clustering adlı bir algoritma aşağıda sunulmuştur.

```
ExtractDBSCAN-Clustering (ClusterOrderedObjs,e', MinPts)
// Precondition: e' £ generating dist e for
ClusterOrderedObjs
   ClusterId := NOISE;
   FOR i FROM 1 TO ClusterOrderedObjs.size DO
      Object := ClusterOrderedObjs.get(i);
      IF Object.reachability_distance > e' THEN
         // UNDEFINED > e
         IF Object.core_distance £ e' THEN
            ClusterId := nextId(ClusterId);
            Object.clusterId := ClusterId;
         ELSE
            Object.clusterId := NOISE;
      ELSE // Object.reachability_distance £ e'
         Object.clusterId := ClusterId;
END; // ExtractDBSCAN-Clustering
```

Algoritma, ClusterOrderedObjects adlı OPTICS çıktısı olan liste üzerinde döngüyle dönerek tüm nesneler için bir koşul aramaktadır. Bu koşul, Object adlı kullanılan nesnenin ulaşılabilirlik mesafesinin kümeleme parametresi olan ε' değerinden büyük olup olmadığıdır. Eğer büyükse bu defa iç-mesafesinin ε' değerinden büyük olup olmadığı kontrol edilmektedir. Eğer o da büyükse nesne gürültü olarak işaretlenmektedir. Eğer iç-mesafe küçükse nesne bir sonraki küme numarası ile işaretlendirilir. Eğer ilk koşulda, ulaşılabilirlik mesafesi ε' değerinden küçük veya eşitse nesne mevcut küme numarası ile işaretlenir. Bu şekilde veritabanındaki nesneler kümelere atanmış veya gürültü olarak işaretlenmiş olur.

Tanım 7: (OPTICS algoritmasının sonucu)

DB, n adet nokta içeren bir veritabanı olmak üzere, OPTICS algoritması, o:{1..n} →DB şeklinde ifade edilen bir sıralama ve bu nesneler için r:{1..n}→ R≥0 olarak ifade edilen bir ulaşılabilirlik mesafesi listesi oluşturur.

3.4.3. Uzaklık fonksiyonları

Yoğunluk tabanlı kümeleme algoritmaları nesneleri birbirlerine uzaklıklarına veya yakınlıklarına göre kümelemeyi amaçlarlar. Birbirine benzer ve yakın nesneler aynı kümede yer alırken, birbirinden farklı ve uzak nesneler ayrı kümelerde yer alırlar. Nesnelerin birbirine ne kadar uzaklıkta olduğunu hesaplayan birçok yöntem vardır. Bu yöntemlerden bazıları aşağıda listelenmiştir.

Öklid Uzaklığı: Öklid bağıntısına dayanır, bir çift nesnenin koordinatları arasındaki farkların karelerinin toplamının karekökünü eşittir. [14]

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

Manhattan(City Block) Uzaklığı: Mutlak değer uzaklığı da denir. Şehir içinde ilerlemeye benzer. [14]

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

Chebyshev Uzaklığı: En büyük değer uzaklığı da denir. Bir çift nesnenin koordinatları arasındaki farkların en büyüğünü uzaklık kabul eder. [14]

$$D_{\mathrm{Chebyshev}}(p, q) := \max_i(|p_i - q_i|).$$

Minkowski Uzaklığı: Genel metrik uzaklıktır. Derece belirtilmelidir. [14]

p = 1 olduğunda Manhattan uzaklığı,
p = 2 olduğunda Öklid uzaklığı,
p = ∞ olduğunda limit alınırsa Chebyshev uzaklığı olur.

$$P = (x_1, x_2, \ldots, x_n) \text{ and } Q = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$$

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}.$$

Canberra Distance: Bir çift nesnenin koordinatları arasındaki farkın, koordinatların toplamına bölünmesinden oluşan seridir. [14]

$$d_{ij} = \sum_{k=1}^{n} \frac{\left| x_{ik} - x_{jk} \right|}{\left| x_{ik} \right| + \left| x_{jk} \right|}$$

Bray Curtis Uzaklığı: Ekoloji ve botanik bilimlerinde kullanılan bir normalizasyon yöntemidir. [14]

$$d_{ij} = \frac{\sum_{k=1}^{n} \left| x_{ik} - x_{jk} \right|}{\sum_{k=1}^{n} \left(x_{ik} + x_{jk} \right)}$$

Angular Separation: Açısal uzaklık, matematikte, astronomide, jeofizikte kullanılır. İki noktayı ya da nesneyi bu iki nesneden farklı bir yerde bulunan gözlemciye birleştiren doğrular arasındaki açı ya da açının boyu olarak tanımlanır. İki vektör arasındaki Kosinüs açısını gösterir. Değeri kosinüs değerinde olduğu gibi [1,-1] arasında değişir. Yüksek değeri nesnelerin birbirine benzer olduğunu belirtir. [14]

$$s_{ij} = \frac{\sum_{k=1}^{n} x_{ik} \cdot x_{jk}}{\left(\sum_{k=1}^{n} x_{ik}^2 \cdot \sum_{r=1}^{n} x_{jr}^2 \right)^{\frac{1}{2}}}$$

Correlation Coefficient: Açısal uzaklığın, koordinatları ortalama değerlerine merkezleyerek standart hale getirilmesidir. [14]

$$s_{ij} = \frac{\sum_{k=1}^{n} \left(x_{ik} - \overline{x}_i \right) \cdot \left(x_{jk} - \overline{x}_j \right)}{\left(\sum_{k=1}^{n} \left(x_{ik} - \overline{x}_i \right)^2 \cdot \sum_{r=1}^{n} \left(x_{jk} - \overline{x}_j \right)^2 \right)^{\frac{1}{2}}}$$

İkili değişkenler için uzaklık ölçütleri: Evet/Hayır, Doğru/Yanlış, 1/0, Var/Yok gibi değerler alan değişkenler arasında uzaklıklar olumlu ve olumsuz değerlerin görülme sıklığı ile ölçülür. [14]

		B Nesnesi	
		Evet	Hayır
A Nesnesi	Evet	p	q
	Hayır	r	s

p = Her ikisi de olumlu
q = A için olumlu, B için olumsuz
r = A için olumsuz, B için olumlu
s = Her ikisi de olumsuz
t = p + q + r + s = toplam boyut sayısı

Simple Matching Uzaklığı: $s_{ij} = \frac{p+s}{t}$

Jaccard's Uzaklığı:

$$s_{ij} = \frac{p}{p+q+r}$$

Hamming Uzaklığı: $d_{ij} = q + r$

4. PARALEL HESAPLAMA

4.1. Paralel Hesaplamanın Amacı

Geleneksel olarak yazılımlar tek bir işlemci üzerinde çalışmak üzere tasarlanırlar. Problem, birbiri ardına işlenen sıralı komutlar ile çözülür. Aynı anda tek bir komut işlenebilir. Paralel hesaplama, bir problemin birden fazla işlemci üzerinde eş zamanlı olarak çalıştırılan komutlar ile çözülmesidir. Problemin çözümü aynı anda çalışabilen parçalara bölünür.

Paralel hesaplamada kullanılan hesaplama kaynakları birden fazla işlemcisi bulunan bilgisayarlar, bir ağ ile birbirine bağlı bilgisayarlar veya bunların kombinasyonları olabilir. Bir problemin paralel hesaplama ile çözülebilmesi için aynı anda çözülebilen parçalara bölünebilmesi gerekir. Ayrıca birden fazla hesaplama kaynağı ile tek hesaplama kaynağına göre daha kısa sürede çözülebilir olmalıdır.

Paralel hesaplamanın kullanıldığı bazı alanlar şöyledir: [18]

- Fizik ve parçacık fiziği araştırmalarında
- Biyoloji ve genetik araştırmalarında
- Kimya ve moleküler bilimlerde
- Jeoloji, sismoloji
- Makine mühendisliği, aerodinamik tasarım
- Mikroelektronik, devre tasarımı
- Bilgisayar bilimleri, matematik
- Veritabanları, veri madenciliği
- Petrol arama
- Internet arama motorları,
- Tıbbi teşhis koyma
- İlaç tasarımı, üretimi

- Finansal ve ekonomik modelleme
- İleri grafik, sanal gerçeklik, eğlence sektörü
- Video ve multi-media teknolojileri

Bazı problemler o kadar büyük ve karmaşıktır ki, tek bir işlemci ile çözülmesi imkansızdır, bu tür problemlerin çözümünde paralel hesaplama tek yoldur. Paralel hesaplama ile problemleri daha hızlı çözerek para ve zaman kazancı sağlanır. Paralel hesaplama için kullanılacak olan kaynaklar yerel ağ ile birbirine bağlı olabileceği gibi, internet üzerinden yüzbinlerce bilgisayarı paralel hesaplama için kullanarak bilimsel problemleri çözme projeleri de gerçekleştirilmektedir.

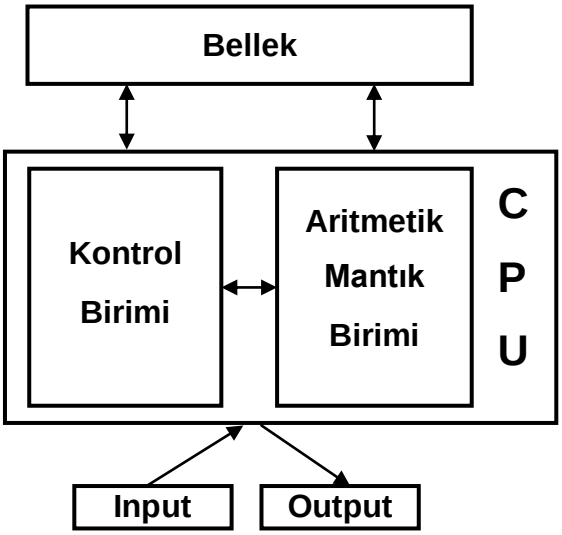
4.2. Paralel Bilgisayar Bellek Mimarileri

Paralel bilgisayar bellek mimarileri aşağıda anlatılmıştır.

4.2.1. Von Neumann mimarisi

Macar matematikçi Jon von Neumann tarafından 1945 yılındaki makalelerinde elektronik bilgisayarın genel yapısı tanımlanmıştır. Bu tarihten sonra bilgisayarlar bu temel tasarımı izlemişlerdir. Bu tasarım 4 ana bileşenden oluşur; Bellek, Kontrol birimi, Aritmetik Mantık Birimi, Girdi/Çıktı Birimi.

Rastgele erişim belleği, program komutlarını ve veriyi saklamak için kullanılır. Kontrol birimi, komut ve verileri bellekten getirir. Komutları çözümler ve komutları sıralı olarak işler. Aritmetik birim, temel aritmetik işlemleri yerine getirir. Girdi/Çıktı birimi kullanıcı ile arayüz oluşturur. Şekil 4.1'de mimari gösterilmiştir.



Şekil 4.1 Von Neumann Mimarisi [18]

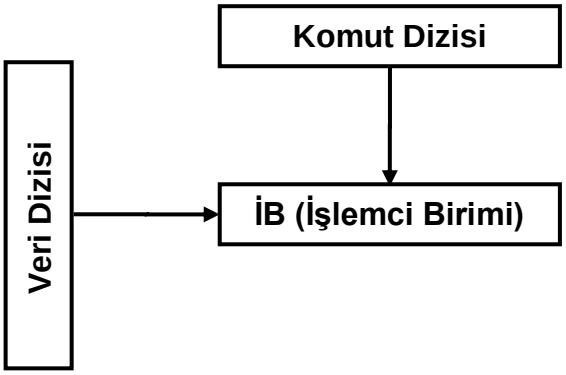
4.2.2. Flynn sınıflandırması

Paralel bilgisayarların ilk sınıflandırması 1966 yılında Michael J. Flynn tarafından yapılmıştır ve Flynn Sınıflandırması olarak bilinir. Flynn sınıflandırması çok işlemcili bilgisayarları iki farklı boyutta sınıflandırır; Komut ve Veri, Tek veya Çoklu.

Tablo 4.1 Flynn Sınıflandırması [18]

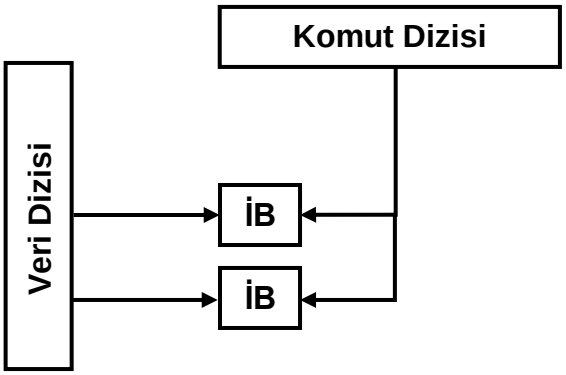
SISD Tek Komut, Tek Veri	SIMD Tek Komut, Çoklu Veri
MISD Çoklu Komut, Tek Veri	MIMD Çoklu Komut, Çoklu Veri

SISD (Single Instruction Single Data): Paralel olmayan bir bilgisayardır, bir anda sadece bir komut dizisi ve sadece bir veri dizisi işlenebilir. En eski ve günümüzde de kullanılan bilgisayar tipidir. [18]



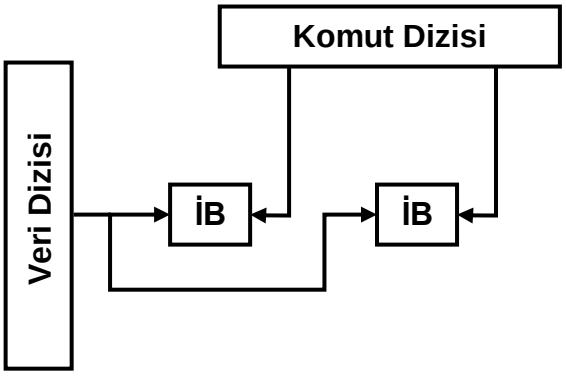
Şekil 4.2 SISD (Single Instruction Single Data) [44]

SIMD (Single Instruction Multiple Data): Paralel bir bilgisayar tipidir, tüm işleme birimleri tek bir komut dizisini işler, fakat bu işleme birimleri farklı komut dizileri ile çalışır. Görüntü işleme gibi yüksek seviyede düzen gerektiren problemlerin çözümlerinde kullanılır. Senkronize ve kesin çalışırlar. [18]



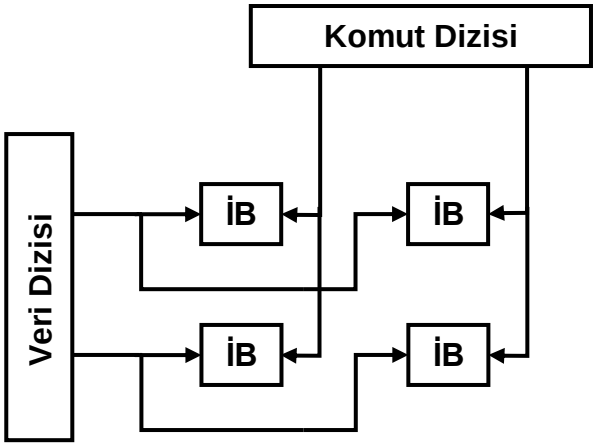
Şekil 4.3 SIMD (Single Instruction Multiple Data) [45]

MISD (Multiple Instruction Single Data): Tek bir veri dizisi, farklı işleme birimlerinde çalışan farklı komut dizilerine girdi olarak verilir. Örnek olarak uzay mekiği kontrol bilgisayarı ve şifre kırma bilgisayarları verilebilir. [18]



Şekil 4.4 MISD (Multiple Instruction Single Data) [46]

MIMD (Multiple Instruction Multiple Data): Günümüzde en yaygın paralel bilgisayar türüdür, birbirinden farklı veri dizileri ile çalışan farklı komutlardan oluşur. Eş zamanlı veya eş zamanlı olmadan çalışabilir. [18]



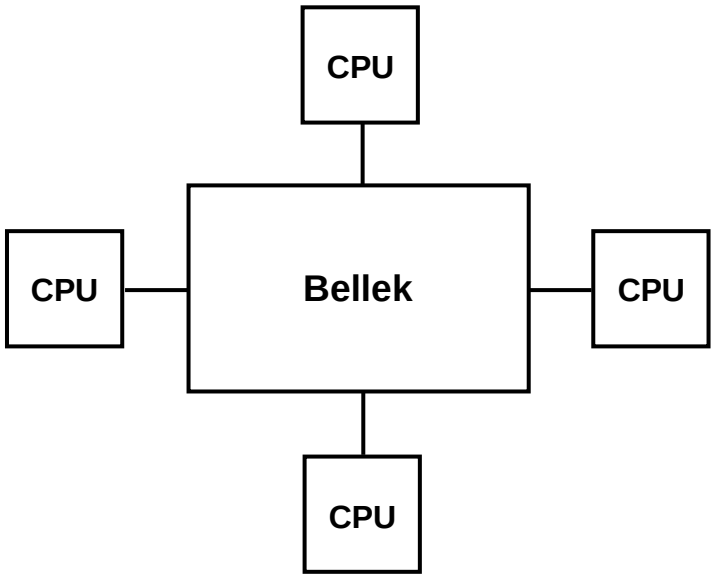
Şekil 4.5 MIMD (Multiple Instruction Multiple Data) [47]

4.2.3. Paylaşımlı bellek

Paylaşımlı bellek mimarisindeki paralel bilgisayarlarda tüm işlemciler tüm bellek kaynaklarına erişebilirler. Bu işlemciler bağımsız olarak çalışırlar fakat aynı bellek

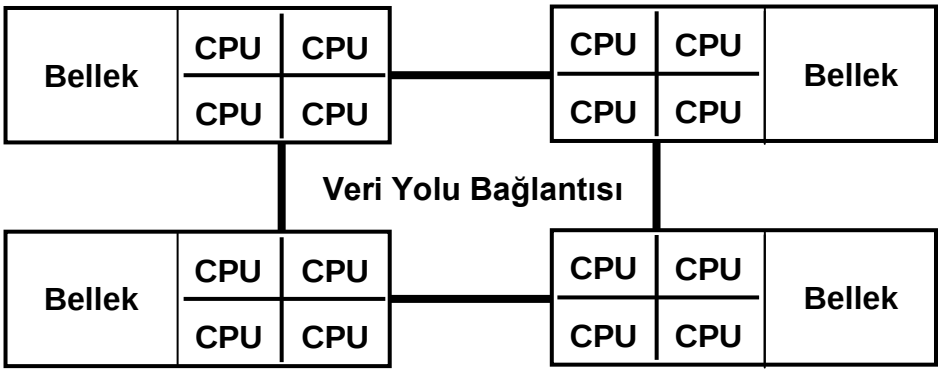
kaynaklarına erişirler. Bir işlemcinin değiştirdiği bir bellek alanı, tüm alanı ortak kullandıkları için diğer işlemcilere de görünür.

UMA(Uniform Memory Access): Bu mimarinin günümüzde en yaygın örneği SMP(Symmetric Multiprocessor)'dir. Bu makineler özdeş işlemcilerden oluşur. Belleğe eşit sıklıkta ve eşit zamanda erişebilirler. Bu tür mimariye bazen CC-UMA(Cache Coherent UMA) da denir. Önbellek uyumluluğu, bir işlemci paylaşımlı bellekte bir alanı güncellerse diğer işlemcilerin bu güncellemeden haberdar olmaları demektir. Önbellek uyumluluğu donanım seviyesinde gerçekleştirilir. [18]



Şekil 4.6 UMA (Uniform Memory Access) [18]

NUMA(Non-Uniform Access): Genellikle iki veya daha fazla SMP'yi fiziksel olarak bağlayarak oluşturulur. Bir SMP diğer bir SMP'nin belleğine direk olarak erişebilir. Tüm işlemciler tüm belleklere eşit erişim zamanına sahip değildirler. Link üzerinden belleğe erişimi yerel erişime göre daha yavaştır. Eğer önbellek uyumluluğu sağlanırsa buna CC-NUMA(Cache Coherent NUMA) denir. [18]



Şekil 4.7 NUMA (Non-Uniform Memory Access) [18]

NUMA'nın avantajı, genel adres uzayının belleğe kullanıcı dostu programlama bakış açısı getirmesidir. Diğer bir avantajı, görevler arası veri paylaşımının işlemciler ve belleğin yakınlığına göre hem hızlı hem de düzenli olmasıdır.

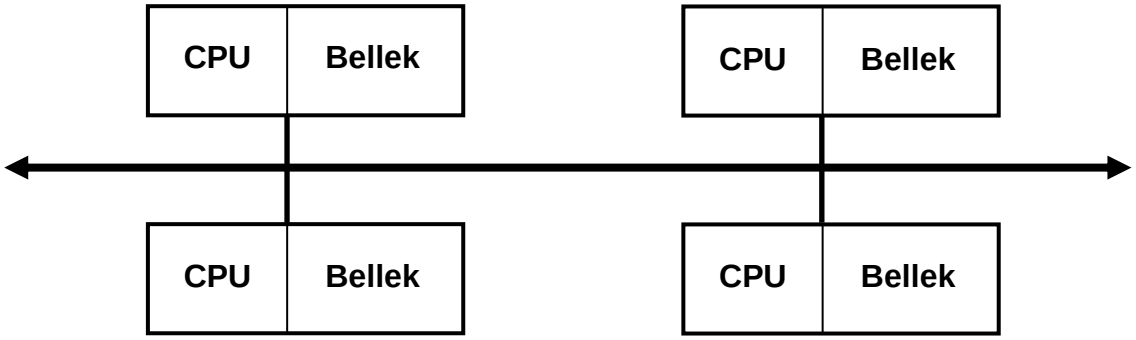
NUMA'nın en önemli dezavantajı, bellek ve işlemciler arası ölçeklenebilirlik eksikliğidir. Daha fazla işlemci eklemek, paylaşımlı bellek ve işlemciler arasındaki trafiği geometrik olarak artırır. Önbellek uyumluluğu olan sistemlerde de önbellek ve bellek yönetimi ile ilgili trafiği geometrik olarak artırır.

Senkronizasyonda programcının sorumluluğu genel adres uzayına doğru erişimi garanti etmektir. Çok fazla sayıda işlemcili paylaşımlı bellek makinesi tasarlamak ve üretmek, zor ve pahalı olmaktadır.

4.2.4. Dağıtık bellek

Dağıtık bellek mimarisi, işlemci ve bellekler arasında iletişim ağına ihtiyaç duyar. İşlemciler kendi yerel belleğine sahiptirler. Bir işlemcideki bellek adresleri diğer bir işlemcideki bellek adresine denk gelmez. Dolayısı ile tüm işlemciler arası genel adres uzayı yapısı yoktur. Her işlemcinin kendi yerel belleği olduğu için bağımsız olarak çalışırlar. Yerel bellekte yapılan değişiklikler diğer işlemcilerin belleklerine etki etmez. Bu sebeple önbellek uyumluluğu söz konusu değildir.

Bir işlemci diğer bir işlemcinin belleğine erişme ihtiyacı duyarsa bunun nasıl ve ne zaman olacağına karar vermek programcının işidir. Görevler arası senkronizasyondan da yine programcı sorumludur. [18]



Şekil 4.8 Dağıtık Bellek [18]

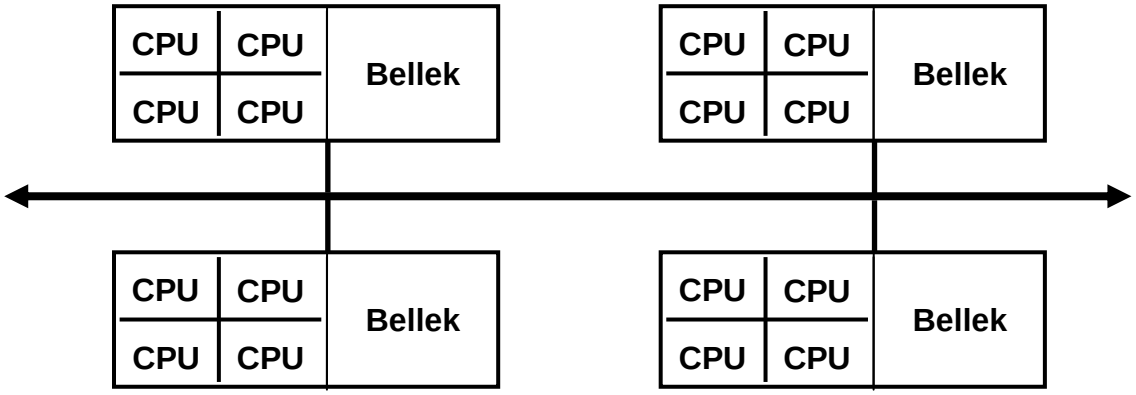
Dağıtıl belleğin avantajları, belleğin işlemci sayısı ile ölçeklenebilmesi ve işlemci sayısını artırınca bellek miktarın da orantılı olarak artmasıdır. Her işlemci kendi belleğine kesinti olmadan ulaşabilir, önbellek uyumluluğunun getirdiği yük de ortadan kalkar. Maliyet düşüktür, hazır ticari ürünler bir ağ ile birleştirilerek bu yapı oluşturulabilir.

Dağıtık belleğin dezavantajı ise, programcının işlemciler arası veri transferi ile ilgili tüm detaylardan sorumlu olmasıdır. Genel adres uzayına göre yazılmış uygulamaları bu mimariye uyarlamak zor olabilmektedir. [18]

4.2.5. Hibrid dağıtık-paylaşımlı bellek

Günümüzde en büyük ve hızlı bilgisayarlar paylaşımlı ve dağıtık bellek mimarilerini birlikte kullanan bilgisayarlardır. Bu mimarinin paylaşımlı bellek bileşenleri genellikle önbellek uyumlu SMP makineleridir. SMP üzerindeki işlemciler makinenin belleğini genel olarak adresleyebilirler. Mimarinin dağıtık bileşeni ise birden fazla SMP makinesinin ağ ile birbirine bağlanmasından oluşur. SMP

makineler sadece kendi belleklerini adresleyebilirler, diğer bir SMP üzerindeki belleği adresleyemezler. Bu sebeple SMP'ler arasında veri transferi için ağ bağlantısına ihtiyaç vardır. [18]



Şekil 4.9 Hibrid Dağıtık-Paylaşımlı Bellek [18]

4.3. Paralel Programlama Modelleri

Belli başlı paralel programlama modelleri aşağıda sunulmuştur.

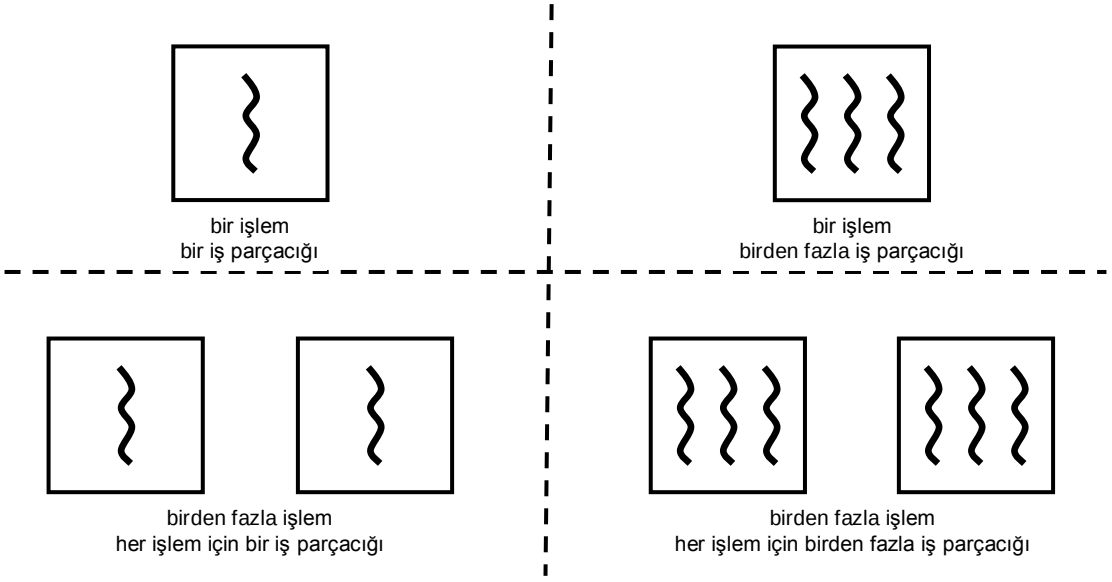
4.3.1. Paylaşımlı bellek modeli

Paylaşımlı bellek programlama modelinde görevler eş zamanlı olmadan okuma yazma yapmak için ortak bir adres uzayını paylaşırlar. Paylaşımlı belleğe erişimi düzenlemek ve kontrol etmek için kilit ve semafor gibi mekanizmalar kullanılır. Programcı bakış açısından bu modelin en önemli avantajı, verinin sahiplik kavramı ortadan kalkar böylece görevler arası özel bir iletişim yöntemi belirlemeye ihtiyaç kalmaz. Program geliştirme basitleşir. [18]

Verinin yerelliğini sağlamanın ve yönetmenin zorluğu bu modelin en önemli dezavantajıdır.

4.3.2. İş parçacığı (thread) modeli

İş parçacığı modelinde, tek bir işlem birden çok eş zamanlı çalışma kanalına sahip olabilir. Bu iş parçacıkları aynı bellek alanını paylaşabilirler. Tek bir işlemci üzerinde çalışan iş parçacıkları, zaman bölüşümlü çoğullama yoluyla sıra ile değişerek işlenilir. Bu değişim kullanıcının farkedemeyeceği kadar sık ve hızlı olur ve kullanıcı işlemlerin aynı anda çalıştığını sanır. Çok işlemcili bir sistemde ise iş parçacıkları ya da görevler farklı işlemciler üzerinde aynı anda çalıştırılırlar. [18]

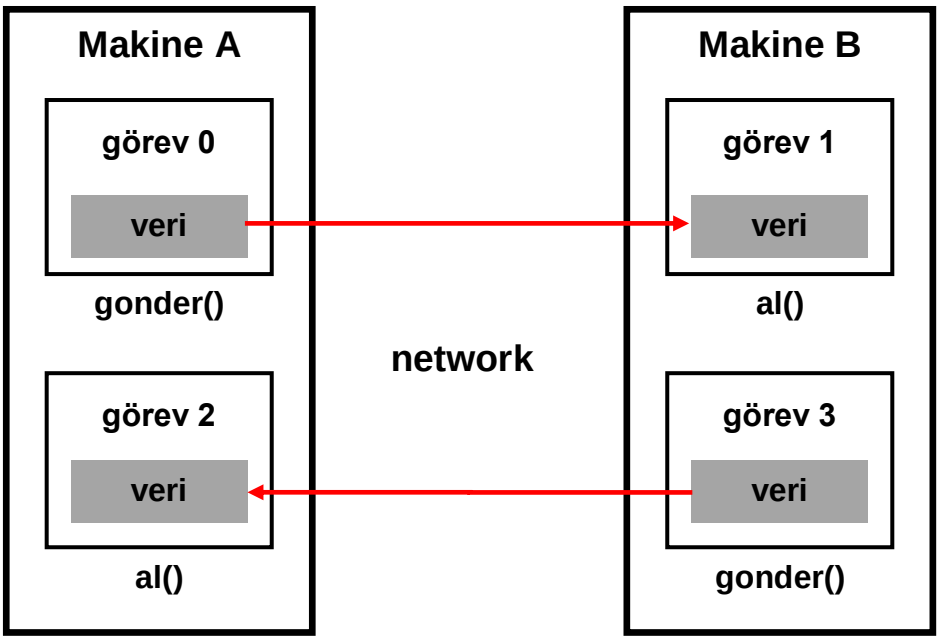


Şekil 4.10 İş Parçacığı Modeli

4.3.3. Mesaj geçirme modeli

Bu modelde görevler, hesaplama yaparken üzerinde çalıştıkları makinenin belleğini kullanırlar, aynı zamanda bu görevler ağ üzerinden birbirleri ile veri alış-verişi de yaparlar. Veri transferi, her iki görevin de ortak talebi ve ortak çalışması ile olur. Mesaj geçirme modelindeki paralel uygulamalarda program kodu içinde mesaj geçirme kütüphanelerinin sağladığı fonksiyonlar kullanılır. Programcı tüm

paralelleştirme işlerinden sorumludur. Veri gönderen ve alan tarafların senkronize olması önemlidir, senkronizasyon bazı özel sinyal mesajları ile sağlanır. [18]

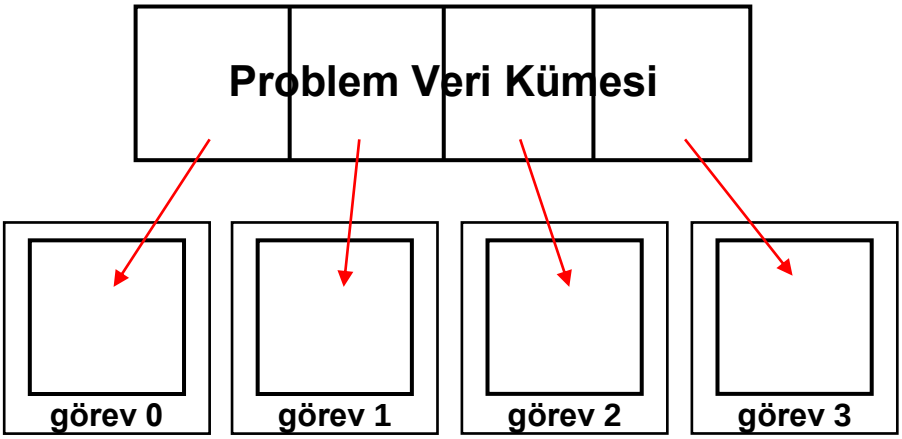


Şekil 4.11 Mesaj Geçirme Modeli [18]

4.3.4. Veri paralel modeli

Bir çok paralel işleme uygulaması tek bir veri kümesi üzerinde çalışma üzerine odaklanmıştır. Bu veri kümesi genellikle 2 veya daha fazla boyutlu diziler ile bellekte saklanır. Paralel çalıştırılacak olan görevler veri kümesinin farklı parçaları üzerinde aynı işi yapmak üzere ayarlanırlar.

Paylaşımlı bellek mimarisinde görevler, ortak bellek alanında farklı noktalara erişerek işlerini yaparlar. Dağıtık bellek mimarisinde ise her görevin çalıştığı yerel belleğe onun işleyeceği veri bölünerek kopyalanır. [18]



Şekil 4.12 Veri Paralel Modeli [18]

4.4. Paralel Program Tasarımı

Paralel algoritmalar verilen bir problemi birden fazla işlemci kullanarak çözmeye yararlar. Algoritma tasarımcısı aynı anda işlenecek komutları belirlemeli ve bunları koordine etmelidir. Algoritmanın doğru tasarlanması paralel hesaplamadan sağlanacak kazancın artması demektir. Paralel bir algoritma aşağıdaki kriterlerden hepsini veya bazılarını karşılamalıdır: [15]

- Aynı anda çalışacak iş parçalarını belirlenmesi
- İş parçacıklarının paralel çalışacak süreçlere dönüştürülmesi
- Programla ilgili girdi, çıktı ve ara ürünlerin dağıtımı
- İşlemcilerin eriştiği veriye erişimin yönetilmesi
- İşlemcilerin paralel programın farklı aşamalarında senkronize edilmesi.

4.4.1. Problemi bölümleme

Paralel program tasarımının ilk adımı problemi birden fazla göreve dağıtılabilecek parçalara bölmektir. Bu bölmenin bilinen iki basit yolu vardır, bunlar "alan bölmesi" ve "işlevsel bölme"dir.

Alan bölmesi(Domain Decomposition): Bu bölmede problemle ilgili veri bölünmelidir. Her bir paralel görev verinin bir parçası üzerinde çalışır.

İşlevsel Bölme(Functional Decomposition): Bu yaklaşımda işlenecek veri değil, hesaplama odak noktasıdır. Problem yağılacak işse göre bölünmelidir. Her bir görev tüm işin belli bir parçasını yürütür. [27]

4.4.2. İletişim

Paralel çalışan görevler arasında iletişim olup olmaması paralel algoritmanın tasarımına bağlıdır. Bazı problemler, aralarında veri paylaşımı ve iletişim olmayan görevler tarafından icra edilebilirler. Bu tip problemlere utandırıcı bir şekilde(embarrassingly) paralel denir.

Birçok paralel uygulama ise bu kadar basit değildirler ve görevler arası veri paylaşımına ve iletişime ihtiyaç duyarlar. Görevler arası iletişim daima uygulamaya ek bir yük ve maliyet getirir. Sistem kaynakları iletişim için kullanılır ve iletişim senkronizasyon gerektirir. Görevler arası iletişimin ne zaman ve nasıl olacağı programcının kontrolündedir. İletişim herhangi iki görev arasında olabileceği gibi, bir görev ile bir grup görev arasında olabilir. [18]

4.4.3. Senkronizasyon

Görevler arasında senkronizasyonu sağlamak için kullanılan mekanizmalar aşağıda anlatılmıştır:

Bariyer: Paralel görevler, bariyer adı verilen senkronizasyon noktaları ile senkronize olurlar. Hiçbir görev tüm görevler bariyere erişene kadar bariyeri geçemez. Böylece programcı görevlerin aynı anda aynı noktada olacaklarını garanti eder. Grup üyesi görevlerden birisi bariyere belli bir sayıda görev ulaşmasını bekleyebilir. [16]

Kilit / semafor: Birden fazla görev tarafından erişilme olasılığı olan ortak bir kaynağa erişimi kontrol eden değişken veya veri yapısı kullanılarak uygulanır. İkili değer veya sayaç ile kullanılır. Bir kaynağın kilitli veya kilitsiz olduğunu ikili değer ile tutulur. Sayaç ise erişilebilir durumda olan kaynak sayısını tutar. Böylece ölümcül kilitlenme(deadlock) durumu veya bir kaynağa aynı anda erişim durumları engellenmiş olur.[17]

Eş zamanlı iletişim: Paralel görevler arasında veri alışverişi sırasında alan ve gönderen tarafın gönderme ve alma bitene kadar bloke olması durumudur. Alma işlemi alan tarafı bloke etmektedir, fakat gönderen tarafın transfer bitene kadar beklemesini sağlamak mesaj geçirme kütüphanesinin görevidir. Gönderme işlemi, eşleşen bir alma işlemi tamamlanmadan bitmez. [16]

4.4.4. Yük dengeleme

Yük dengeleme, yapılacak işi görevler arasında tüm görevleri her zaman meşgul tutacak şekilde dağıtmaya çalışmaktır. Görevlerin boş kaldıkları zamanı en aza indirmek de denebilir. Yük dengeleme, paralel programlarda en yüksek performansı alabilme açısından önemlidir.

Eşit işlem gücüne sahip işlemciler üzerinde çalışan ve aynı işi yapan görevler arasında veri kümesini eşit olarak bölmek yük dengesini sağlayacaktır. Farklı işlem gücündeki işlemciler üzerinde çalışan veya aynı işi yapmayan görevler arasında veri kümesini dağıtırken işlem gücüne ve yapılacak işe göre yük dengesini ayarlamak gerekir. Bu durumda dağılım bazı analiz araçları ile ölçümler yapıldıktan sonra yapılırsa daha isabetli olur.

Bazı problemlerde veri eşit dağıtılsa da yük dengesi sağlanamaz. Bu tür durumlarda dinamik yük dengesi kurmak mantıklı olacaktır. Her bir görev belli bir iş yaptıktan sonra veri kuyruğundan bir miktar daha veri alarak işini yapar, böylece program görevlerin diğerlerine göre erken veya geç bitmesinden etkilenmez.[18]

4.4.5. Performans analizi

Paralel hesaplamada en fazla hızlanmayı sağlayacak optimum bir işlemci sayısı vardır. Belli bir sayıdan sonra işlemci sayısı paralelliği olumsuz etkileyerek çalışma süresini artırmaya başlar. Yine Amdahl adlı bilim adamının 1967 yılında bir sistemin bir parçasının iyileştirilmesinin sistemin tümünün iyileşmesi üzerine etkisini ortaya koyduğu kural, paralel hesaplamada en çok hızlanmayı sağlayacak işlemci sayısını bulmada kullanılır.

Paralel hesaplamada hızlanma, programın sıralı işlenmesi gereken parçalarının hızlı ile sınırlıdır. Örneğin tek bir işlemcide 20 saat çalışan bir uygulamanın 1 saati paralelleştirilemiyor, 19 saati paralelleştirilebiliyor olsun. Kaç tane işlemci kullanılırsa kullanılsın, bu uygulamanın en kısa çalışma zamanı 1 saatten az olamaz. Başka bir deyişle hızlanma 20'de 1'den daha fazla olamaz.

Amdahl kanununa göre,bir programın paralel hale getirilebilir kısmının oranı P ise, (1-P) paralel hale getirilemeyen kısmının oranıdır ve N paralelleştirmede kullanılan işlemci sayısı olmak üzere:

$$\frac{1}{(1-P)+\frac{P}{N}}.$$

N, sonsuza giderken limit 1/(1-P) olur. Bu değer de paralelleştirmenin sağladığı hızlanmayı gösterir. [27]

5. PARALEL DBSCAN UYGULAMASI

5.1. Uygulamanın Amacı

Veri madenciliği uygulamalarının amacı çok fazla veri içinden öz ve yararlı bilgiyi çıkarmaktır. Uygulamaya girdi olan veri kümesinin boyutu ne kadar büyürse uygulamanın çalışma süresi de o kadar artar. Bazen günler hatta haftalar alan bu çalışma sürelerini kısaltmak, bilimsel veya ticari hangi alanda olursa olsun, yapılan çalışmanın sonuçlarının en kısa zamanda alınması açısından önemlidir.

Uygulamaların çalışma zamanlarının kısaltılması için daha güçlü bilgisayarlar kullanılır ve/veya bu bilgisayarlar birlikte çalıştırılarak hızlı sonuç alınması sağlanır. Paralel DBSCAN uygulaması da DBSCAN algoritmasının paralel hale getirilmiş halidir ve çalışma zamanını azaltmak için geliştirilmiştir..

5.2. Geliştirme Ortamı Ve Kullanılan Araçlar

DBSCAN algoritmasının kodlanmasında C++ programlama dili kullanılmıştır. C++ programlama dilinin STL(Standart Template Library) kütüphanesindeki bazı veri tipleri ve “algoritm” kütüphanesi kodlama aşamasında kullanılmıştır. C++’ta yazılan kodu derlemek için “g++” derleyicisi kullanılmıştır.

Editör olarak Windows platformunda Dev-C++ adlı editör, Linux platformunda “vi” adlı editör kullanılmıştır. Dev-C++ adlı editörün içinde “g++” derleyicisi ile derleme özelliği bulunmaktadır. Bu özellik sayesinde her iki platformda da yazılan uygulama kodu herhangi bir değişiklik yapılmadan derlenip çalıştırılabilmektedir.

Geliştirme makinesi olarak kullanılan bilgisayarların işletim sistemleri Redhat 8.0 veya Windows'un farklı sürümleridir. Çalıştırma ortamı olarak kullanılan bilgisayarların işletim sistemleri ise Debian Linux'tur.

DBSCAN algoritmasını paralelleştirme aşamasında LAM/MPI kütüphanesi kullanılmıştır. Bu kütüphane algoritmanın paralel olarak çalıştığı makineler arasındaki mesajlaşma ve senkronizasyonu sağlamaktadır.

Veri setleri üzerinde komşuluk sorgularının en kısa zamanda yapılabilmesine yarayan ve uzaysal verileri saklama amacıyla kullanılan R*-Tree veri yapısı uygulamada kullanılmıştır. Bu veri yapısının kodu internet üzerinden indirilip, üzerinde bazı düzeltme ve uyarlamalar yapıldıktan sonra kütüphane olarak ana uygulamaya dahil edilmiştir.

MATLAB programı, DBSCAN ve OPTICS uygulamalarının sonuçlarını görselleştirmek, sonuçları değerlendirmek için kullanılmıştır. Görselleştirme ve değerlendirme için MATLAB'ın programlama dili ile geliştirilen uygulamalar kullanılmıştır.

5.2.1. Test sistemi

Paralel DBSCAN uygulamasının testleri Maltepe Üniversitesi hesaplama laboratuvarındaki bilgisayarlarda yapılmıştır. Aynı konfigurasyondaki 8 bilgisayar 100 Mbit ağ bağlantısı ile birbirlerine bağlıdırlar. Bu bilgisayarların donanım özellikleri şöyledir:

COMPAQ ALPHAStation XP1000 Workstation (single CPU)

- Alpha 21264A (EV67) 667-MHz CPU with 64-KB data cache,
- 64-KB instruction cache, and 4-MB Level 2 cache
- Two 64-bit PCI slots, two 32-bit PCI slots, one PCI/ISA

- 1.44-MB diskette drive
- 32-bit combination slot
- 512 MB Memory
- 400W power supply
- Onboard wide UltraSCSI adapter
- Onboard 10/100 Ethernet (Twisted pair)
- 650-MB 40X CD-ROM drive

Bilgisayarlar üzerinde Debian 3.1r8 versiyonu işletim sistemi kullanılmaktadır.

5.2.2. Linux ve C++

Linux tabanlı işletim sistemleri ticari firmalar, devlet kurumları, askeri kurumlar, üniversiteler ve kritik uygulamalar kullanan bir çok yerde yaygın olarak kullanılmaktadır. Linux işletim sisteminin bu kadar yaygın kullanılmasının başlıca sebepleri bedava olması, açık kaynak kodu olması ve güvenilirliğidir. Sadece maliyet açısından değil, verdiği performansa göre ihtiyaç duyduğu donanım açısından da çok ucuzdur.

Linux geniş bir kitle tarafından hızla geliştirilmektedir. Çekirdek, yani kaynak kodları GPL (Genel Kamu Lisansı) çerçevesinde özgürce dağıtılabilir, değiştirilebilir ve kullanılabilir. Bu gelişimin en büyük yararı, eksikliklerin kullanıcıların talepleri ve çabaları sonucunda hızla giderilmesidir.

Linux, hemen her bilgisayar platformunda sorunsuzca kullanılabilmektedir. Çok geniş bir donanım desteğine sahip olan Linux; netbook, notebook (taşınabilir bilgisayar) , server (sunucu), workstation (iş istasyonu), smartphone (akıllı telefon), pc (masaüstü bilgisayar) gibi hemen her platformda tam bir uyum içerisinde çalışabilmektedir. Ağırlıklı olarak sunucu ve iş istasyonu platformlarında tercih

edilse de taşınabilir ve masaüstü sistemlerde de kullanım oranını günden güne arttırmaktadır.[19]

Günümüzde üniversitelerdeki akademik çalışmaların büyük bir çoğunluğu Linux üzerinde yapılmaktadır. Lisans ücreti olmaması, her türlü donanım üzerinde çalışabilmesi, hemen her dilin derleyicisi ve yorumlayıcısının bulunması, Linux üzerinde çalıştırılabilen uygulamaların bir çok programlama dili ile geliştirilebiliyor olması ve güvenilir olması akademik çevrede yaygın olarak kullanılmasını getirmiştir. Bu bağlamda tez çalışmalarında Linux platformu tercih edilmiştir.

C++ programlama dili C dilinden türeyen bir dildir. C'den farkı nesneye dayalı(Object Oriented) bir programlama dili olmasıdır. Aynı zamanda C'nin kurallarını da destekler böylece hem nesneye dayalı hem de yapısal programlama (structured programming) yapılabilir.

C++'a orta seviye dil denir çünkü hem alt seviye hem de üst seviye dillerin özelliklerini barındırır. Bu yönüyle en popüler programlama dillerinden biridir. Uygulama alanı geniştir, sistem yazılımları, uygulama yazılımları, aygıt sürücüleri, gömülü yazılımlar, yüksek performanslı sunucu istemci yazılımları ve bilgisayar oyunları gibi eğlence amaçlı yazılımlar geliştirilir.[20]

Tez çalışmalarında kullanılan LAM/MPI kütüphanesi, C,C++ ve Fortran dillerinde kullanılabilmektedir. Yine tez çalışmalarında kullanılan R*-Tree kütüphanesi C++ ile geliştirilmiştir. C++ programlama dili ve Linux işletim sistemi çok uyumlu bir ikili olmaktadırlar. Bu sebeplerden dolayı C++ programlama dili tez çalışmaları için tercih edilmiştir.

5.2.3. LAM/MPI

LAM (Local Area Multicomputer – Yerel Ağ Çoklubilgisayar), MPI(Message Passing Interface – Mesaj Geçirme Arayüzü)'ın açık kaynaklı bir uygulamasıdır.

MPI standardı paralel uygulamalar için bir endüstri standardıdır. Endüstrinin liderleri ve akademik araştırmacılar tarafından tasarlanmıştır ve 20 yıllık paralel programlama deneyimine dayanır. LAM bir kütüphanedir ve kullanıcılarına paralel uygulamaların çalıştığı bilgisayarlar arasında mesaj geçirme fonksiyonları sağlar. [21]

LAM/MPI kullanıcılarına sadece standart MPI fonksiyonları değil, aynı zamanda birçok hata ayıklama ve izleme aracı sağlar. LAM, Unix tabanlı işletim sistemi çalışan masaüstü bilgisayarından süper bilgisayarlara kadar bir çok farklı donanım üzerinde çalışabilir.[21]

LAM/MPI, bir ağ ile birleştirilmiş bilgisayarların tek bir paralel hesaplama kaynağı olarak çalışabilmesini sağlar. Bu kütüphanenin hedefi, taşınabilir ve verimli mesaj geçirme uygulamaları yazmak için programcılara standart fonksiyonlar sağlamaktır. LAM/MPI bir programlama dili değil, programlar içinden çağırılabilen fonksiyonlardan oluşan bir kütüphanedir. Çeşitli noktadan noktaya iletişim fonksiyonları, veri taşıma fonksiyonları ve senkronizasyon fonksiyonları sağlar.

Bu fonksiyonlardan paralel DBSCAN uygulamasında da kullanılanlar aşağıda listelenmiştir:

MPI::COMM_WORLD.Get_rank() : Çağırılan sürecin sıra numarasını döndürür.

MPI::COMM_WORLD.Get_size() : Paralel çalışan kaç süreç olduğunu döndürür.

MPI::COMM_WORLD.Bcast() : Ana süreçten gruptaki diğer tüm süreçlere(kendisi de dahil) gönderilecek mesajlar için kullanılır. Birinden-herkese(one-to-all) yayımlama yapılır.

MPI::COMM_WORLD.Scatter() : Veri dağıtma işlemi yapılan fonksiyondur. Başlangıç adresi verilen bir bellek alanını tüm süreçlere eşit sayıda veri gönderilecek şekilde bölerek dağıtır.

MPI::COMM_WORLD.Scatterv() : Scatter gibi tüm süreçlere dağıtma işlemi yapılır fakat süreçlere farklı sayıda veri dağıtılabilir. Hangi süreçlere ne kadar veri parçası gönderileceği liste ile fonksiyona verilmelidir.

MPI::COMM_WORLD.Gather() : Scatter fonksiyonunun tersine, tüm süreçlerden eşit boyutta veriyi ana sürece toplama işi bu fonksiyon ile yapılır.

MPI::COMM_WORLD.Gatherv() : Scatterv fonksiyonunun tersine, tüm süreçlerden farklı boyutta veriyi ana sürece toplama işi bu fonksiyon ile yapılır.

MPI::Init() : MPI'ı başlatır.

MPI::COMM_WORLD.Barrier() : Senkronizasyon sağlayan bir fonksiyondur, tüm süreçler bu fonksiyonu çalıştırana kadar süreçleri bloke eder. Tüm süreçler fonkisyonu çağırdıktan sonra hepsi aynı anda devam ederler.

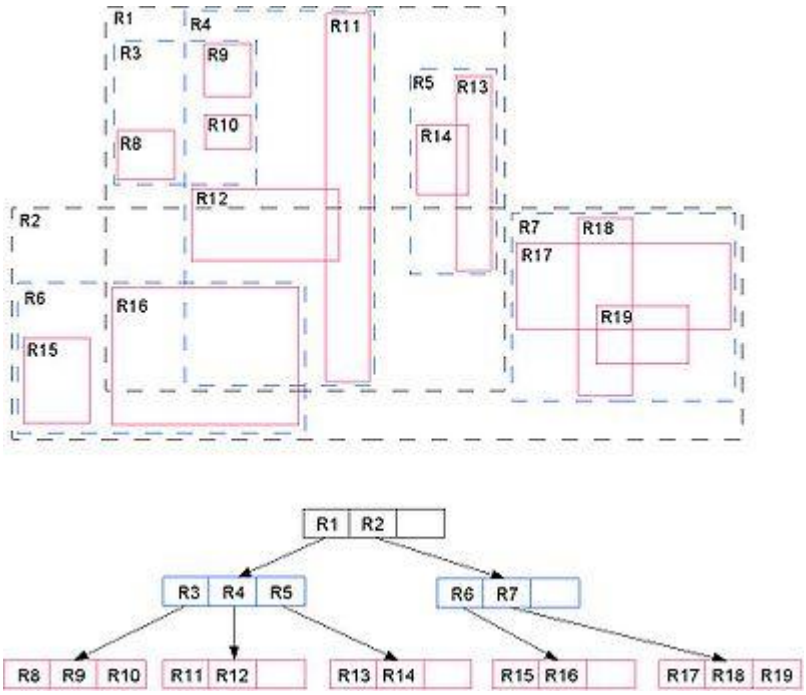
MPI::Finalize() : MPI'ı bitirir.

5.2.4. R*-Tree

R-Tree'ler, B-Tree'lere benzeyen fakat uzaysal erişim yöntemleri için kullanılan ağaç veri yapılarıdır. Çok boyutlu verileri indekslemek için kullanılırlar. Bu veri yapısı uzayı, birbiriyle kesişen hiyerarşik olarak iç içe geçmiş en az sınırlayan dikdörtgenlere böler. R-Tree'nin her bir düğümü değişken fakat sınırı belli kayıtlardan oluşur. Yaprak olmayan düğümlerdeki her bir kayıt iki bilgi taşır, birisi alt düğüme bağlantı diğeri ise bu alt düğümdeki tüm kayıtları çevreleyen bir kutudur.[22]

R*-Tree, R-Tree'nin bir farklı bir çeşididir. 1990 yılında Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider ve Bernhard Seeger tarafından ortaya çıkarılmıştır. Kapsama ve kesişimin en aza indirilmesi R-Tree'lerin performansı açısından çok

önemlidir. R*-Tree, her ikisini de azaltmaya çalışır, düğüm bölme ve yeniden ekleme yöntemleri revize edilmiştir. R*-Tree hem çok boyutlu noktaları hem de uzaysal verileri destekler. [23] Şekil 5.1'de R*-Tree veri yapısı gösterilmiştir.



Şekil 5.1 R*-Tree [22]

Tez konusu olan paralel DBSCAN uygulamasında R*-Tree veri yapısı kullanılmıştır. R*-tree veri yapısı, makalenin yazarlarından Hans-Peter Kriegel tarafından kodlanmıştır. Bu uygulamanın kaynak kodu R-Tree portal [48] adlı web sayfasından indirilip, ihtiyaca göre değiştirilerek paralel DBSCAN uygulamasında kullanılmıştır.

5.2.5. MATLAB

MATLAB programının adı Matrix Laboratory(Matris Laboratuarı)'nın kısaltılmasına dayanır. MathWorks adlı firma tarafından geliştirilen MATLAB, sayısal hesaplama ortamı ve 4. nesil bir programlama dilidir. Matris işlemleri, fonksiyon ve veri grafiği çizimi, algoritma kodlama, kullanıcı arayüzü geliştirme ve C, C++, Fortran gibi

programlama dillerinde yazılmış programlara arayüz oluşturma gibi işlevleri vardır. Mühendislik, bilim ve ekonomi alanlarından kullanıcıları vardır.

Tez çalışmaları sırasında MATLAB'dan grafik çizimleri ve hesaplama amacıyla faydalanılmıştır. OPTICS uygulaması sonuçlarını grafiğe dönüştürme, iki boyutlu veri setlerini noktalama, DBSCAN sonuçlarından purity, entropy ve mutual info değerlerinin hesaplanması sırasında MATLAB kullanılmıştır.

5.3. Kullanılan Veri Setleri

Geliştirilen kümeleme algoritmasını test etmek amacıyla girdi olarak kullanılacak veri setlerine ihtiyaç doğmuştur. Bu ihtiyaç doğduktan sonra internet üzerinde yapılan araştırmalarda doğal yollarla elde edilen veri setlerinin yanında özel geliştirilmiş programların çıktısı olan sentetik veri setleri olduğu görülmüştür.

5.3.1. Doğal veri setleri

Kısa adı UCI olan "University of California, Irvine" üniversitesinin bünyesinde bulunan "Machine Learning Repository" adlı veri tabanında[24] veri madenciliği için çoğu doğal, 189 adet veri seti bulunmaktadır. Bu veri tabanları arasında sınıflandırma, regresyon analizi ve kümeleme için ayrılmış veri setleri bulunmaktadır. Ayrıca veri türü sayısal, kategorik veya her ikisinin karışımı olarak ayrılabilir.

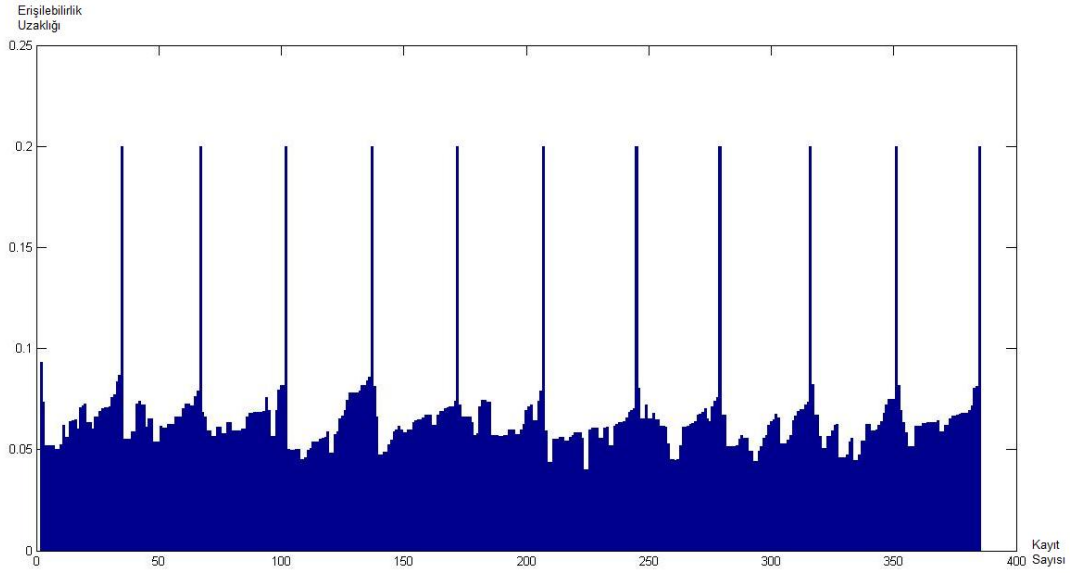
Bu sitenin sağladığı veri setleri arasında yapılan araştırma sonucunda Shuttle adlı veri setinin testler için en uygunu olduğuna karar verilmiştir. NASA tarafından sağlanan bu veri seti bir uzay mekiğinin istatistiksel kayıtlarını içermektedir. Veri seti 9 boyutlu olup 43500 kayıttan oluşmaktadır. Veri setinde her bir kaydın hangi sınıfa(kümeye) dahil olduğu bilinmektedir. Bu sınıf etiketleri kümeleme

algoritmasının ürettiği sonuçların doğruluğunu analiz ederken işe yaramaktadır. Shuttle veri seti üzerinde yapılan testlerin sonuçları 6. Bölümde gösterilmiştir.

5.3.2. Sentetik veri setleri

Paralel DBSCAN uygulamasının sağladığı faydayı ortaya koymak adına, uygulamaya verilen veri setinin çalışma süresi belli bir zamanın üstünde olmalıdır. Veri setindeki kayıtların tüm özellikleri sayısal olmalı, veri seti yeterince büyük olmalı ve kümelemeye uygun olmalıdır. Bu kriterleri sağlayan doğal veri setlerinin azlığı nedeniyle sentetik veri seti üreten programların araştırılmasına girildi.

DataGenerator adlı programın[25] ürettiği veri setlerindeki kümeler merkez-köşeler(Center-Corners) dağılımı ile üretilmektedir. Bu yöntemde çok boyutlu uzaydaki bir şeklin her bir köşesini merkez olarak alacak şekilde kümeler üretilmektedir. Bu yöntemin ürettiği kümeler birbirlerinden çok ayrık olmasından dolayı Paralel DBSCAN uygulaması %100 başarı ile kümeleme gerçekleştirmektedir. Bu tür bir veri setinin OPTICS algoritmasına girdi olarak verilmesi sonucunda ortaya çıkan sonucun grafiği Şekil 5.2'de sunulmuştur.



Şekil 5.2 Ayrık Kümeler OPTICS Grafiği

Veri setindeki gürültü noktaların azlığı veya çokluğu ve kümelerin ayrıklığı veya iç içe geçmiş olması kümeleme algoritmasının sonuçlarının başarı oranını etkilemektedir. Birbirinden çok ayrık ve gürültüsü olmayan veri setleri ile çalışmanın %100 başarılı sonuçlar vermesi Paralel DBSCAN algoritmanın başarısını tam olarak ortaya koyamayacağı için bu üreteç programın kullanılmamasına karar verilmiştir.

Daha sonraki araştırmalarda Manchester üniversitesinden 2 öğretim üyesinin geliştirdiği bir program bulunmuştur. Bu program kümelemeye uygun, tamamen sayısal veri içeren, istenilen boyut, kayıt ve küme sayısında veri seti üretmektedir [26]. Program kümeleri çokdeğişirli normal dağılım yöntemi kullanarak bulmaktadır. Bu yöntem, tek değişirli bir dağılım olan normal dağılımın (veya Gauss-tipi dağılımın) çoklu değişirli hallere genelleştirilmesidir.

Bu program ile 2 veri seti üretilmiştir. Bunlardan 1.sinde 3 boyuta sahip 4722 kayıt bulunmaktadır ve bu veri seti 10 kümeye ayrılmıştır. 2.sinde 5 boyuta sahip 10637 kayıt bulunmaktadır ve bu veri seti de 10 kümeye ayrılmıştır. Bu veri setleri üzerinde yapılan testlerin sonuçları 6. Bölümde gösterilmiştir.

5.4. Uygulamanın Geliştirme Adımları

Paralel DBSCAN uygulamasının geliştirme aşamaları ve bu aşamalardaki çalışmaların detayları aşağıda anlatılmıştır.

5.4.1. DBSCAN algoritmasının kodlanması

Paralel DBSCAN uygulamasının geliştirilme yöntemi olarak problemi parçalara bölerek geliştirme benimsenmiştir. Parçalar sırayla geliştirilip, test edilerek aşama kaydedilmiştir. Bu adımların ilki DBSCAN algoritmasının makaledeki sözde koda göre seri yapıda kodlanmasıdır.

Seri DBSCAN uygulamasının geliştirilmesinde ilk olarak text formattaki veri seti dosyasının okunarak bilgisayarın belleğine yüklenmesi sağlanmıştır. Text dosya satır satır okunarak her bir satır ayıraçlara göre boyutlarına ayrılmaktadır. Dosyada her bir kaydın boyutları arasında ayıraç olarak boşluk karakteri, sekme karakteri, virgül, noktalı virgül veya iki nokta üst üste karakteri kullanılabilir.

Bellekte kayıtları saklamak için "Point" adlı bir sınıf tanımlanmıştır.

```
class Point
{
public:
    int PointNumber;
    vector<double> Coordinats;
    bool Visited;
    bool Noise;
    int ClusterID;
};
```

PointNumber adlı özellik dosyadaki kayıt sayısını temsil eder ve tekil tanımlayıcı olarak programın bazı yerlerinde kullanılır. Coordinats adlı vektör noktanın boyutlarını saklayan bir listedir. Visited adlı boolean özellik noktanın algoritma tarafından işlenip işlenmediğini gösterir. Noise adlı özellik bu noktanın gürültü olarak işaretlenip işaretlenmediğini gösterir. ClusterID adlı integer tipindeki özellik kümeleme sonucunda noktaya atanan küme etiketini saklar.

Bellekte Point tipindeki noktaları saklamak için PointVector adlı vektör tipinde bir liste tanımlanmıştır. Dosyadan okunan değerler bellekte Point sınıfından bir nesne yaratılarak bu nesneye doldurulur, daha sonra bu nesnenin adresi PointVector listesine eklenir.

Tüm noktalar belleğe alındıktan sonra program PointVector listesinin üzerinde for döngüsü ile döner. Bu döngüde işlenen nokta için komşuluk sorgusu çalıştırılır, eğer komşu sayısı minpts değerinden büyük veya eşitse expandCluster fonksiyonu çağırılır, küçükse bu nokta gürültü olarak işaretlenir. Komşuluk sorgularında 3.4.3 maddesinde anlatılan uzaklık fonksiyonlarından birisi program çalıştırılırken komut satırından parametre ile seçilerek kullanılabilir.

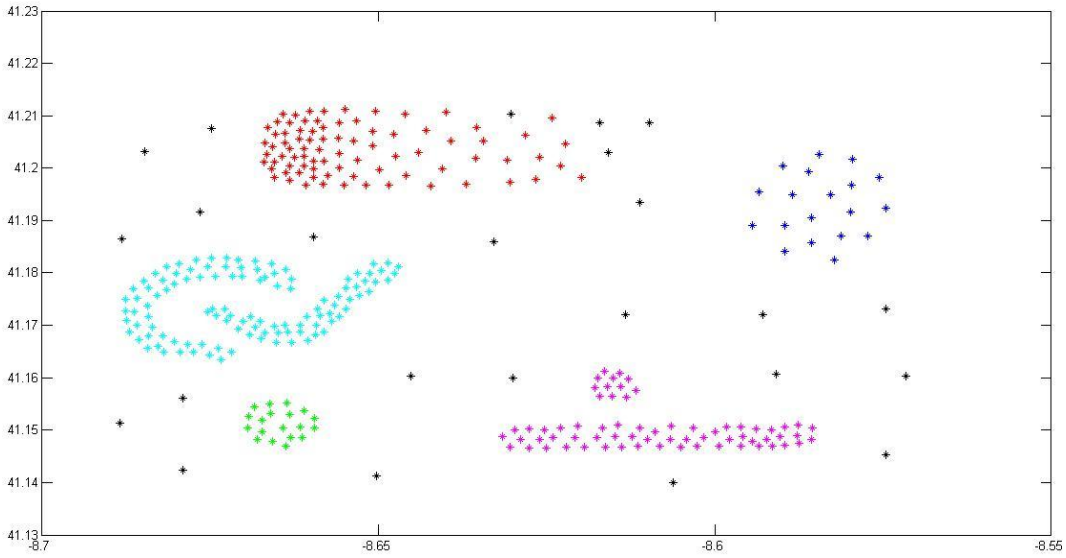
expandCluster fonksiyonunda ana fonksiyondan gönderilen noktadan başlanılarak komşuluk sorguları yapılarak küme genişletilir. Sınır noktalara ulaşılmasından sonra küme daha fazla genişletilemiyorsa program ana fonksiyona geri döner. Ana fonksiyondaki for döngüsü ziyaret edilmemiş bir nokta kalmayana kadar döndükten sonra sonlanır. Döngüden çıkıldıktan sonra tüm noktalar küme etiketleri ile beraber sonuç dosyasına yazıldıktan sonra program sonlanır.

DBSCAN uygulaması komut satırından parametre olarak 6 adet değer alabilmektedir, bu değerler isteğe bağlı olarak verilir, verilmediği durumda varsayılan değerleri kullanılır.

--dataset Programa girdi olarak verilecek veriseti dosyasının yolu
--eps komşuluk sorgularında kullanılacak Epsilon değeri

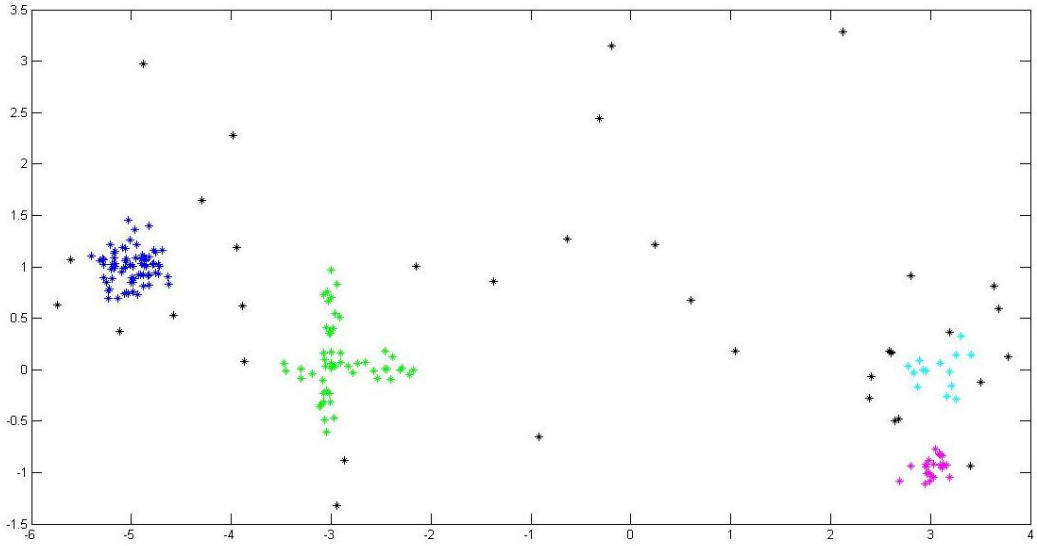
--minpts	eps komşulukta bulunması beklenen en az nokta sayısı
--distfunc	komşuluk sorgusunda kullanılacak uzaklık fonksiyonu
--result	sonuç dosyasının yolu
--help	kullanımla ilgili yardım gösterimi

2 boyutlu ve 322 kayıtlı bir veri seti DBSCAN uygulamasında kümelendikten sonra oluşan çıktı dosyası MATLAB'da grafiğe dönüştürülerek kümeleri farklı renklerle Şekil 5.3'te gösterilmiştir.



Şekil 5.3 2x322 DBSCAN Sonucu

2 boyutlu ve 200 kayıtlı bir veri setinin DBSCAN sonucu aşağıdaki Şekil 5.4'te gösterilmiştir.



Şekil 5.4 2x200 DBSCAN Sonucu

5.4.2. OPTICS algoritmasının kodlanması

DBSCAN algoritmasının kodlanmasının ardından DBSCAN'den türetilmiş bir algoritma olan OPTICS'in kodlanması kolay hale gelmiştir. Veri seti dosyasından kayıtlar aynı şekilde okunarak belleğe alınmaktadır. For döngüsü içinde komşuluk sorgusu yapılmadan expandCluster fonksiyonu çağırılmaktadır. OPTICS'te noktalara küme etiketi vermek yerine noktalar için erişilebilirlik uzaklığı ve iç uzaklık hesaplanmaktadır.

Tüm noktalar için bu değerler hesaplandıktan sonra değerler sonuç dosyasına yazılarak uygulama sonlanmaktadır. DBSCAN'den farklı olarak map container'ı kullanılmıştır. Map, noktaları erişilebilirlik uzaklığına göre sıralı olarak tutmaktadır. DBSCAN uygulaması için kullanılan uzaklık fonksiyonları ve komut satırı parametreleri OPTICS uygulaması için de geçerlidir.

5.4.3. DBSCAN ve OPTICS algoritmalarının R*-Tree veri yapısı ile kodlanması

DBSCAN ve OPTICS algoritmalarının çalışma zamanlarının %90'dan daha fazlasını komşuluk sorguları oluşturmaktadır. Komşuluk sorguları ne kadar az zaman alırsa, uygulamalar da o kadar az sürede tamamlanmaktadır. Bellekte tutulan listede sıralı arama ile yapılan komşuluk sorgusu R*-Tree gibi uzaysal erişim yöntemi sağlayan bir veri yapısına göre uzun sürmektedir. Bu nedenle, kodlanan DBSCAN ve OPTICS uygulamalarında noktaların vektör yerine R*-Tree veri yapısında saklanması sağlanmıştır.

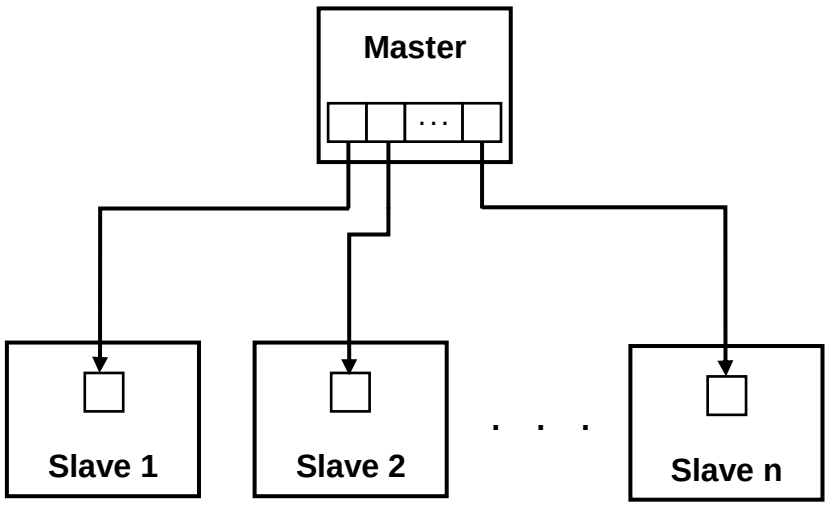
R*-Tree üzerinde yapılan komşuluk sorguları, tree üzerinde arama zamanı daha kısa olduğu ve birbirine yakın noktalar aynı düğümlerde tutulduğu için daha kısa sürmektedir.

5.4.4. R*-Tree veri yapısı ile kodlanmış DBSCAN uygulamasının paralelleştirilmesi

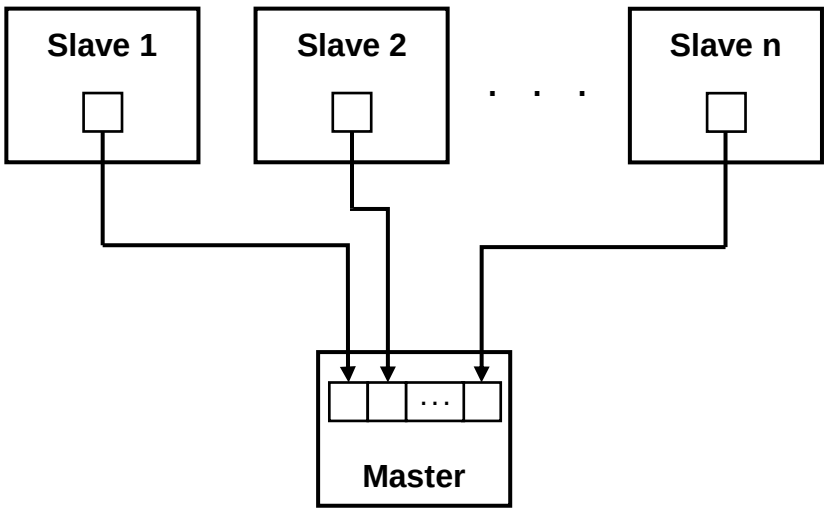
Seri halde çalışan DBSCAN uygulamasını paralel hale getirmek için 5.2.3 bölümünde detayı verilen LAM/MPI adlı mesaj geçirme kütüphanesinin kullanılmasına karar verilmiştir. Paralel uygulamanın tasarımında bir makinenin "master" rolü ile çalışması, diğer makinelerin de "slave" rolü ile çalıştırılması planlanmıştır. DBSCAN algoritmasında çalışma zamanının %90'dan fazlasını komşuluk sorgusu aldığı bilindiğine göre bu bölümdeki işlem yükünün tüm makinelere paylaştırılmasının çalışma zamanını azaltacağı öngörülmüştür.

Master makinesi, komşuluk sorgusu yapılması gereken nokta sayısını belirleyip, kendisi de dahil, havuzdaki tüm makinelere paylaştırmakla sorumludur. Şekil 5.5'te paylaştırma işlemi gösterilmiştir. Komşuluk sorgusu sırasında master makinesi de slave rolü üstlenmektedir. Slave makineler, komşuluk sorgusu yapılması gereken

noktaların listesini alır ve sırayla bu noktalar için komşuluk sorgusu çalıştırır. Sorgudan dönen noktaları her noktadan sadece bir tane bulunabilen bir listede tutar ve sorgusu yapılacak noktalar bittiği zaman bu listeyi master'a gönderir. Şekil 5.6'da toplama işlemi gösterilmiştir.



Şekil 5.5 MPI – Scatter(v)



Şekil 5.6 MPI – Gather(v)

Master, slavelerden gelen bu listeleri tekrarlanan noktaları eleyerek birleştirir ve bu işlemleri tekrarlayarak kümeyi genişletmeye çalışır. Master'ın slave'lere yaptığı gönderimler MPI kütüphanesinin Scatter ve Scatterv fonsiyonları ile

gönderilmektedir. Slave'lerden geri dönen listeler ise Gather ve Gatherv fonksiyonları ile toplanmaktadır. Master tarafından tüm slave makinelere gönderilecek kontrol mesajları Bcast fonksiyonu ile, programlar arasındaki senkronizasyon Barrier fonksiyonu ile yapılmaktadır.

Master ve slave makinelerde çalışan uygulama arasında herhangi bir fark yoktur. Master makine mpirun komutunun verildiği makine olmaktadır. Master ve slave rolleri programın içindeki kontrol komutları ile belirlenmektedir. Rank değeri 0 olan makine master rolünde, diğerleri slave rolünde çalışır.

5.4.5. Paralel DBSCAN uygulamasının test edilmesi

Geliştirilen paralel DBSCAN uygulaması 5.2.1 maddesinde anlatılan test sistemi ile test edilmiştir. Testlerde kullanılan veri setleri 5.3 bölümünde anlatılan doğal veri seti Shuttle ve sentetik veri setleri 3x4722(3 boyut, 4722 kayıt) ve 5x10637(5 boyut, 10637 kayıt)'dir. Uygulamanın ürettiği sonuçların doğrulaması bir sonraki bölümde anlatılacaktır.

Paralelleştirmenin ne kadar fayda sağladığı ile ilgili bilgi almak için tüm veri setleri 1 makineden 8 makine'ye kadar ardışık sayıda makine ile paralel çalıştırılmıştır. Çalışma süreleri kaydedilerek makine sayısı ve çalışma zamanı ile ilgili bağıntıyı gösteren grafikler hazırlanmıştır. Bu grafikler 6. Bölümde gösterilmektedir.

5.4.6. Test sonuçlarının doğrulanması

Paralel DBSCAN uygulamasının ürettiği sonuçların doğru olup olmadığını kontrol etmek amacıyla bir yöntem bulma ihtiyacı doğmuştur. Bu yöntem uygulamaya girdi olarak verilen veri setindeki bilinen küme etiketleriyle, çıktı dosyadaki küme etiketlerini karşılaştırarak sonuçların isabet oranını bulmaya yaramalıdır. Tezde

sonuçlarının doğrulanması için purity, entropy ve mutual information değerleri hesaplanmıştır. Bu değerlerin açıklamaları şöyledir:

Purity: Bir kümedeki bütün nesnelerin, o kümedeki baskın sınıfa ait oldukları varsayımına dayanarak hesaplanan sınıflandırma doğruluk oranıdır. Başka bir deyişle purity, baskın kategorideki nesnelerin sayısının toplam nesne sayısına oranı olarak tanımlanır. [50]

Entropy: Purity'e göre daha kapsamlı bir ölçümdür. Sadece baskın sınıfta olan ve olmayan nesnelerin sayısını değil, tüm dağılımı hesaba katar. Bir veri kümesindeki belirsizlik ve rastgeleliği ölçmek için kullanılır. Bir kümedeki tüm nesneler aynı kategoride olduğu sürece entropy'si 0'dır. Entropy tabanlı kalite, 1'den normalleştirilmiş(0 ile 1 arasına indirgenmiş) entropy'i çıkartılarak bulunur. [50]

Mutual Information: Kalite ölçütleri arasında teorik olarak en iyi kurulmuş değerdir. Girdi veri setindeki küme etiketleri ile çıktıdaki küme etiketleri arasındaki ortak bağımlılığı gösterir. [50]

6. SONUÇ

Bu tez çalışmasında amaç, yoğunluk tabanlı kümeleme algoritmalarının paralelleştirilebileceğini ve paralel çalışan uygulamaların seri uygulamalara göre daha çabuk sonuç vereceğini göstermektir. Bu bölümde, kümeleme sonuçlarının doğrulanması ile ilgili ölçümler tablolar ile, paralelleştirmenin sağladığı kazanç ise grafikler ile sunulmaktadır.

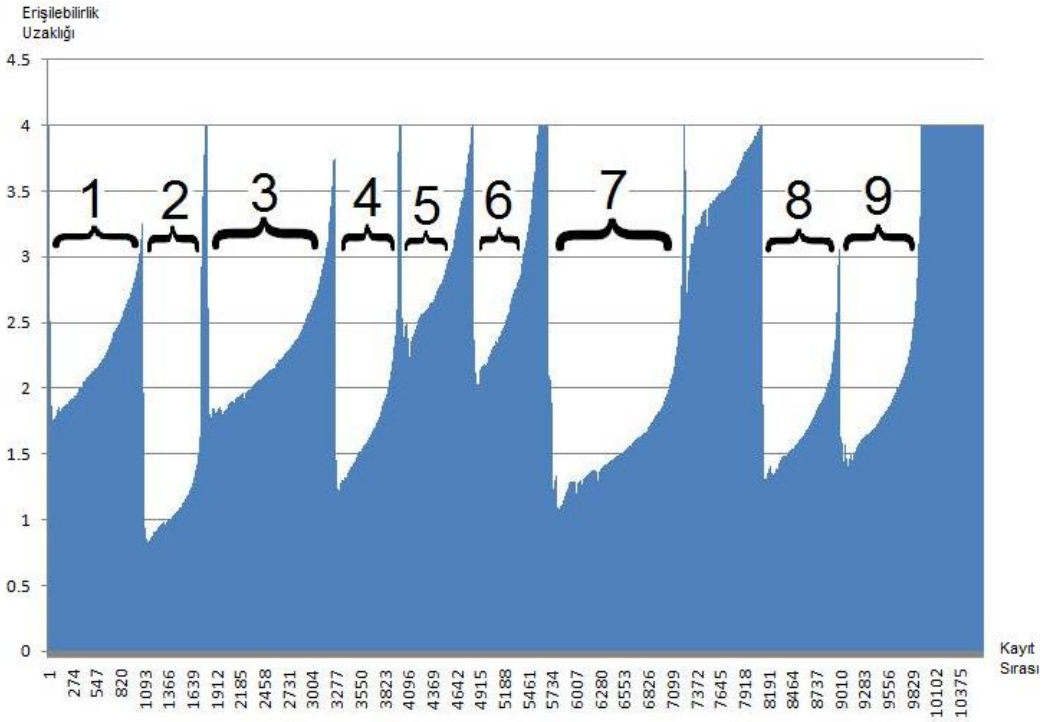
Tablo 6.1'de testlerde kullanılan veri setleri, kayıt sayıları ve boyutları gönderilmiştir.

Tablo 6.1 Test Veri Setleri

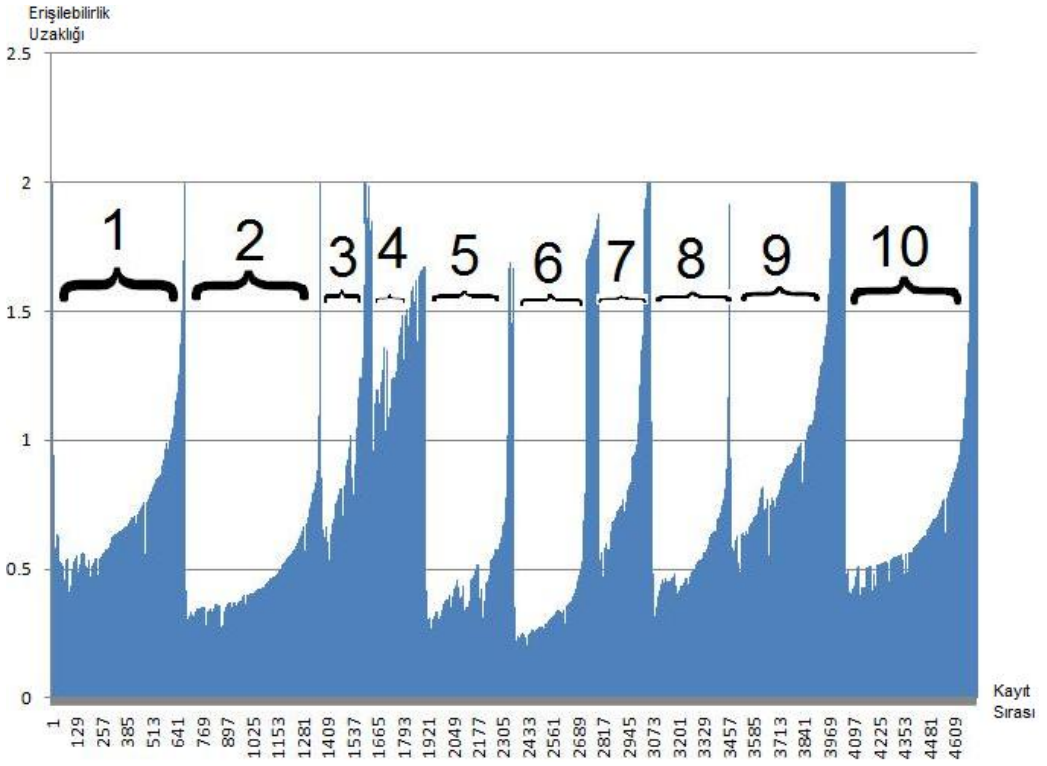
Veri Seti Adı	Kayıt Sayısı	Boyut Sayısı
Shuttle	43500	9
5x10637	10637	5
3x4722	4722	3

6.1. Test Sonuçları

Kullanılan veri setleri üzerinde, içerdikleri küme sayısını belirlemek, eps ve minpts değerlerini tahmin edebilmek amacıyla, OPTICS algoritması çalıştırılmıştır. Algoritma sonucunda oluşan çıktının grafiği ve oluşan kümeler Şekil 6.1 ve Şekil 6.2'de gösterilmiştir.



Şekil 6.1 5x10637 OPTICS Grafiği



Şekil 6.2 3x4722 OPTICS Grafiği

OPTICS grafiğinde kümeler vadiler olarak karşımıza çıkar. Veri setinde kaç tane küme olduğunu bulmak için dikey eksende istenilen bir eps değerinin karşılığından yatay bir çizgi çizilerek, bu çizginin altında kalan vadiler sayılır. Şekil 6.1'de dikey eksende 3 değerinin karşılığından yatay bir çizgi çizilerek altında kalan vadiler gösterilmiş ve 9 adet küme bulunmuştur. Aynı grafikte dikey eksende 2 değerinden yatay bir çizgi çekildiği durumda 7 küme sayılacaktır. Bu durum, istenilen eps değerine göre elde edilen küme sayısının OPTICS grafiğinden görülebildiğini ortaya koymaktadır. Şekil 6.2'de dikey eksende 1.5 değerinden bir çizgi çizilerek altında kalan vadiler gösterilmiş ve 10 adet küme bulunmuştur.

Testlerde en iyi sonuçların alınabilmesi için tahmini bir eps ve minpts ikilisi ile başlanarak sonuca göre eps ve minpts değerlerini değiştirerek denemeler yapılmıştır. Her yeni sonuç bulunduğunda sonuçların doğruluğu purity, entropy ve mutual information değerleri hesaplanarak değerlendirilmiştir. En iyi sonuca yaklaşmak için eps ve minpts değerleri revize edilerek yeni değerlerle uygulama çalıştırılmıştır.

Shuttle veri seti için en iyi sonuç alınan eps değeri 9.5, minpts değeri 15 olarak bulunmuştur. Bu değerlerle yapılan kümelemenin purity ve entropy değerleri Tablo 6.2'de gösterilmiştir.

Tablo 6.2 Shuttle Doğrulama Değerleri

	Kümeler				Mutual Information
	1	2	3	4	
Purity	0.9131	1	1	1	0.625
Entropy	0.1517	0	0	0	

5x10637 veri seti için en iyi sonuç alınan eps değeri 3.06, minpts değeri 12 olarak bulunmuştur. Bu değerlerle yapılan kümelemenin purity ve entropy değerleri Tablo 6.3'de gösterilmiştir.

Tablo 6.3 5x10637 Doğrulama Değerleri

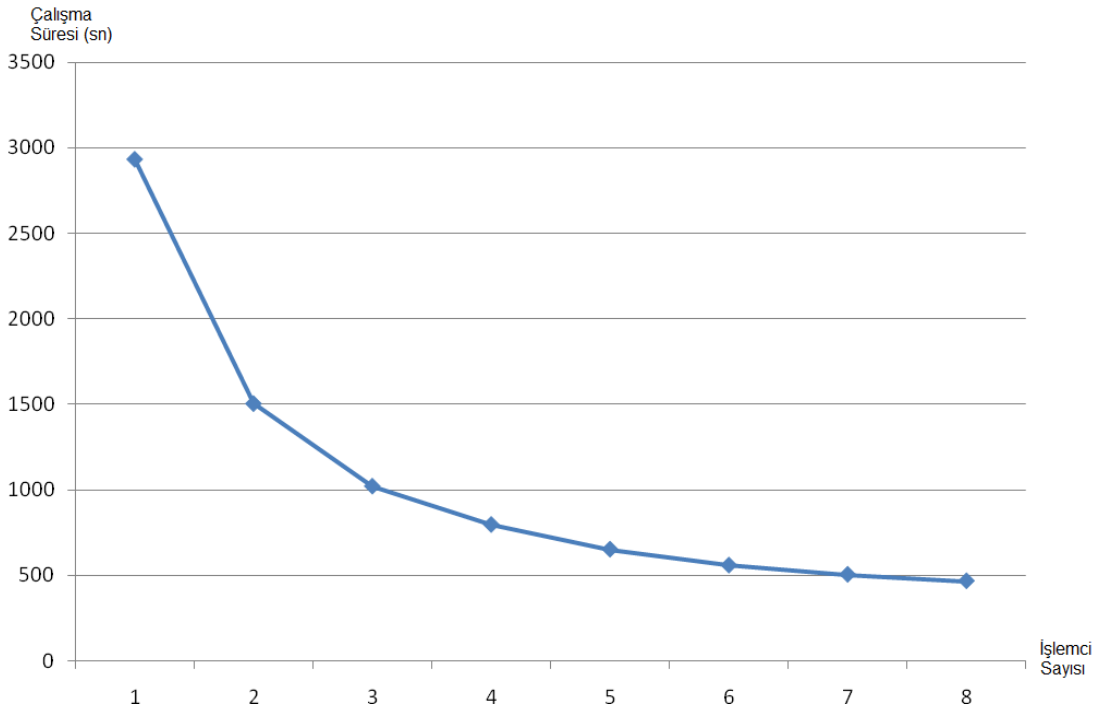
	Kümeler										Mutual Information
	1	2	3	4	5	6	7	8	9	10	
Purity	1	1	1	1	1	1	1	1	0.9989	1	0.8459
Entropy	0	0	0	0	0	0	0	0	0.004	0	

3x4722 veri seti için en iyi sonuç alınan eps değeri 1.66, minpts değeri 5 olarak bulunmuştur. Bu değerlerle yapılan kümelemenin purity ve entropy değerleri Tablo 6.4'de gösterilmiştir.

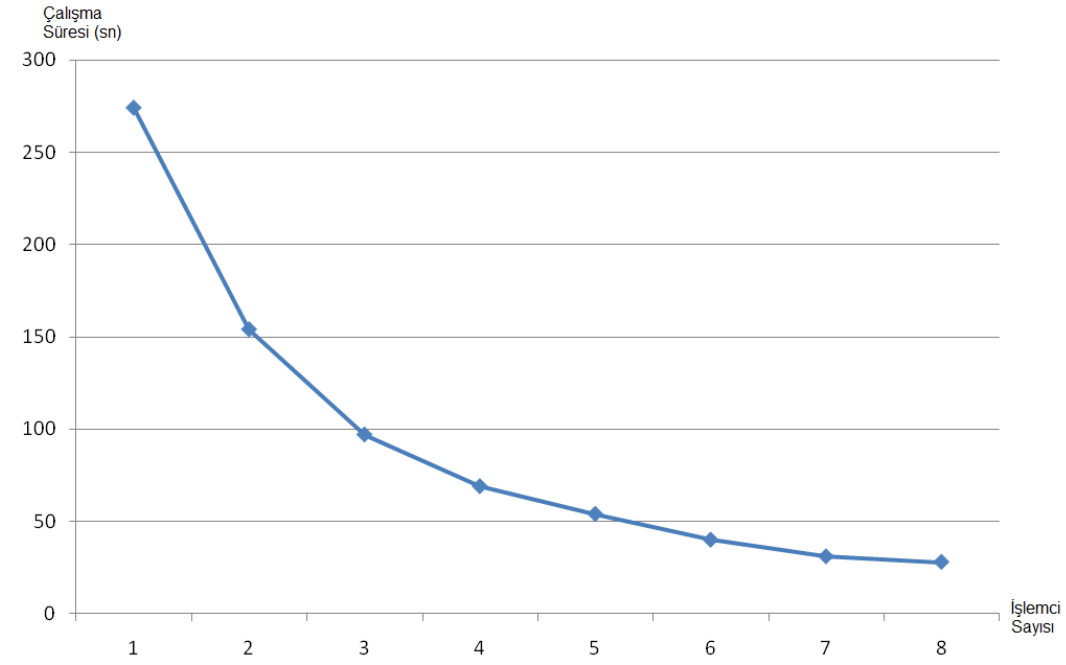
Tablo 6.4 3x4722 Doğrulama Değerleri

	Kümeler														Mutual Information
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
Purity	1	1	1	1	1	1	1	1	0.9841	0.9957	1	1	1	1	0.8971
Entropy	0	0	0	0	0	0	0	0	0.0371	0.0127	0	0	0	0	

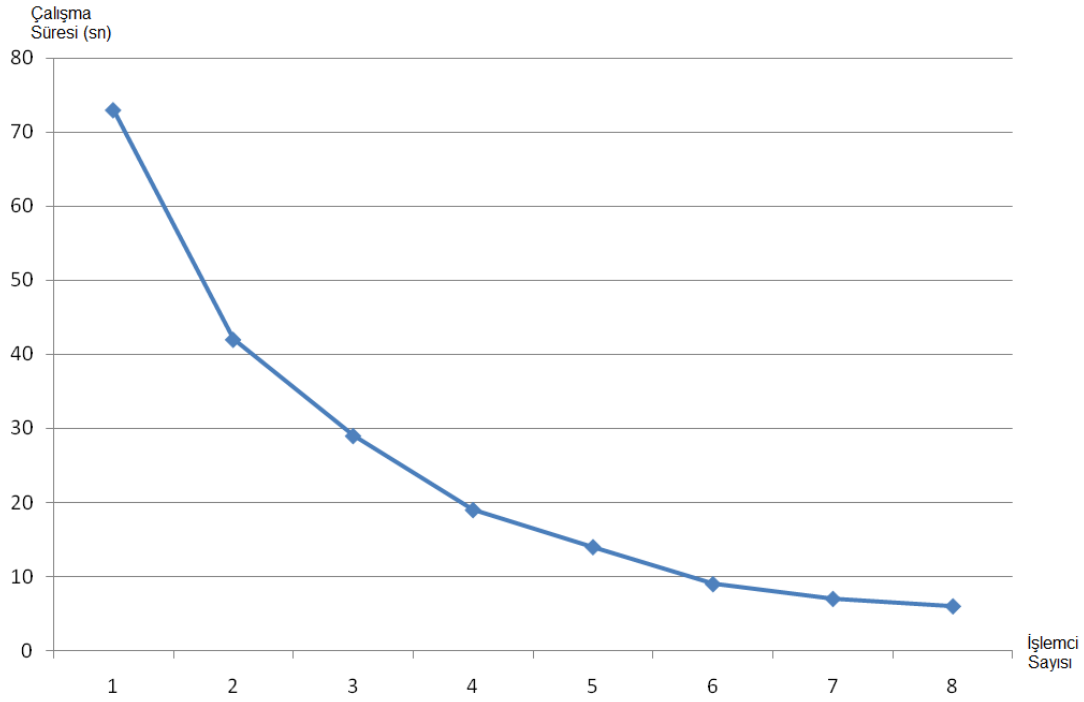
Her bir veri seti için en uygun eps ve minpts değerleri bulunduktan sonra bu değerler ile paralel çalıştırma testleri yapılmıştır. Paralel çalışmanın sağladığı kazancın görülebilmesi için 1'den 8'e kadar ardışık sayılar kadar bilgisayar kullanılarak testler yapılmıştır. Her testin çalışma süresi kaydedilmiştir. Şekil 6.3, 6.4 ve 6.5'deki grafiklerde bilgisayar sayısı ve saniye cinsinden çalışma zamanı arasındaki ilişki sergilenmiştir.



Şekil 6.3 Shuttle Performans Grafiği



Şekil 6.4 5x10637 Performans Grafiği



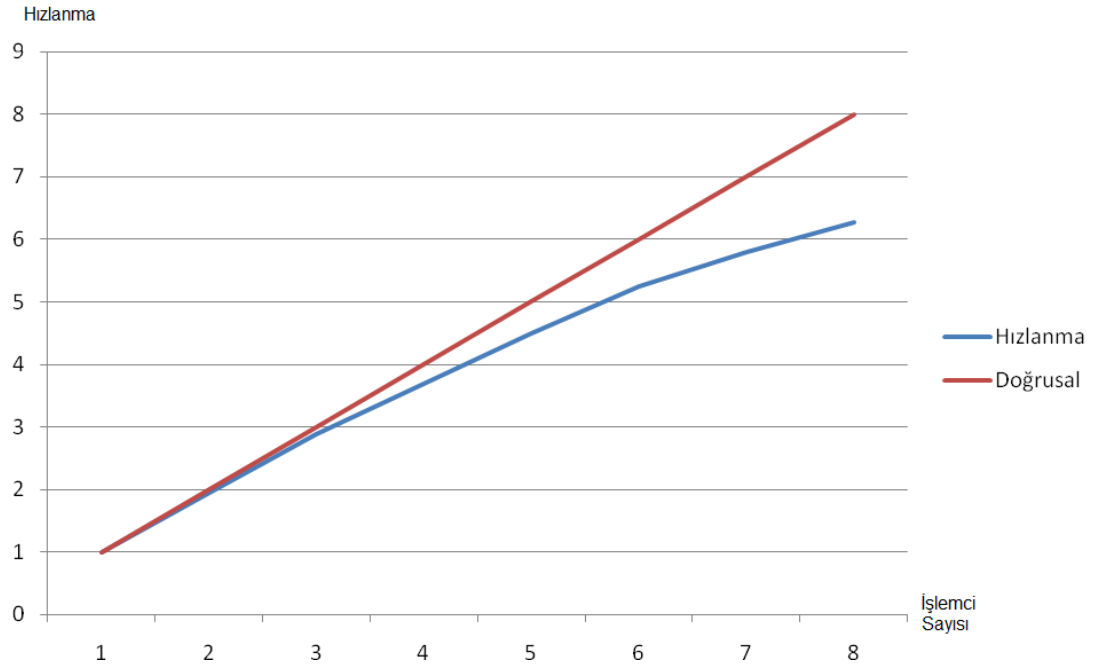
Şekil 6.5 3x4722 Performans Grafiği

Yukarıdaki grafiklerde, paralelleştirmenin çalışma zamanını kısaltma etkisi açıkça görülmektedir. Komşuluk sorgularının yoğunluk tabanlı kümeleme algoritmalarının çalışma zamanının %90'dan fazlasını aldığı, komşuluk sorgularının paralel olarak birden fazla makine üzerinde yapılmasının çalışma süresini makine sayısı ile orantılı olarak kısalttığı sonucuna varılmıştır.

Hızlanma katsayısı birden fazla işlemcili paralel sistemin, tek işlemcili sıralı sisteme oranla göreceli performans artışına denir. t_s tek işlemcili sıralı sistemin çalışma süresi, t_p ise birden fazla işlemcili sistemin algoritmayı paralel çalıştırma süresidir. Hızlanma katsayısı formülü aşağıda gösterilmiştir:

$$Hizlanma = \frac{t_s}{t_p}$$

Şekil 6.6'da işlemci sayısı ve hızlanma arasındaki ilişkiyi ortaya koyan grafikte görüldüğü üzere Paralel DBSCAN uygulaması doğrusallık karakteri taşımaktadır.



Şekil 6.6 Hızlanma Grafiği

6.2. Yorumlar

Günümüzde veri madenciliği, sağladığı avantaj ve faydalarından dolayı birçok alanda kullanılmaktadır. Ticari veya bilimsel amaçlarla veri madenciliği hayatımızda günden güne daha çok yer almaktadır. Kümeleme analizi de veri madenciliğinin önemi giderek artan bir dalıdır. Yoğunluk tabanlı kümeleme algoritmaları nesneleri birbirlerine benzerliklerine göre gruplamayı amaçlar. Bu algoritmalar büyük veri setleri üzerinde çalıştırıldığında uygulamanın tamamlanması saatler hatta günler alabilmektedir. Paralelleştirme bu gibi durumlarda algoritmaların çalışma zamanlarını kısaltmada önemli katkılar sağlamaktadır.

Bu çalışmada DBSCAN ve OPTICS algoritmaları, makalelerindeki sözde kodlara uygun olarak kodlanmıştır. Tek bilgisayar üzerinde yapılan çalışmalarda komşuluk sorgularının çok zaman aldığı görülmüş ve bu sürenin kısaltılması için uzaysal erişim sağlayan ve daha kısa sürede komşuluk sorgusu yapılabilen R*-Tree veri yapısı

kullanılmıştır. DBSCAN ve OPTICS algoritmaları R*-Tree veri yapısı ile kodlandıklarında önemli ölçüde süre kazancı sağlanmıştır.

DBSCAN algoritmasının test edilmesi sırasında kullanılan veri setlerini üreten DataGenerator programının ayrık kümeler elde ettiği ve bu durumun DBSCAN algoritmasının performansını değerlendirme aşamasında olumsuz katkı yaptığı görülmüştür. Bu sebeple başka bir sentetik veri üreten program kullanıldı. Bu program doğal veri setlerine daha yakın, birbirinin içine geçmiş kümeler üretmiştir. Sentetik veri üreten programların istenilen boyut ve kayıt sayısında ve doğal veri setlerine benzer veri üreten türleri testlerde kullanılmalıdır.

Bu aşamadan sonra R*-Tree kullanan DBSCAN algoritmasının paralelleştirilmesi için LAM/MPI kütüphanesi kullanıldı. DBSCAN uygulamasının en çok zaman harcayan kısmı olan komşuluk sorguları, LAM/MPI yardımı ile tüm bilgisayarlara eşit şekilde paylaştırılarak yapıldı. DBSCAN'in yanı sıra OPTICS algoritmasının paralelleştirilmesi çalışmaları da yürütüldü. OPTICS algoritmasının işleyiş bakımından DBSCAN'den bazı farkları olduğu ve bu farklar yüzünden paralelleştirilmesinin zor olduğu görüldü.

OPTICS algoritmasının işleyişinde önemli yeri olan ve nesnelerin sıralanmasını sağlayan erişilebilirlik uzaklığı(reachability distance) ve iç-uzaklık(core distance) değerlerinin, tüm nesneler için, tüm bilgisayarlara dağıtılması gereği vardır. Çalışma sırasında hesaplanan bu değerlerin tüm bilgisayarlara dağıtılması işi, çok iyi düşünülmesi ve tasarlanması gereken bir yapıya dayanmalıdır.

3 farklı veri seti ile gerçekleştirilen testlerde DBSCAN algoritmasının paralelleştirmeye elverişli olduğu ve paralel çalışan DBSCAN'in Amdahl Kanunu'na uygun olarak çalışma süresinin kısaldığı, bununla birlikte küme oluşturma performansının ve kalitesinin etkilenmediği görülmüştür.

6.3. Öneriler

Paralel yavaşlama(paralel slowdown), bir programın paralelleştirilmesinin çalışma zamanına olumsuz etki etmesi durumuna, başka bir deyişle geç tamamlanmasına, verilen addır. Paralel yavaşlama iletişim darboğazlarından kaynaklanabilir. Daha çok işlem birimi eklendikçe uygulama iletişime daha çok zaman harcanmasına sebep olacaktır. [51]

Paralel DBDSCAN uygulamasının testlerinde Maltepe Üniversitesi hesaplama laboratuvarındaki 8 bilgisayar kullanılmıştır. 8 bilgisayarın paralel çalıştırıldığı testlerde paralel yavaşlama ile karşılaşılmamıştır. Daha çok işlem birimi ile test yapılarak paralel yavaşlamanın başladığı bilgisayar sayısı ilerleyen çalışmalarda belirlenebilir.

DBSCAN algoritması eps ve minpts değerlerine bağımlı ve bu değerlere karşı hassas bir algoritmadır. En iyi kümeleme sonuçlarına ulaşmak amacıyla çeşitli eps ve minpts değerleri ile birçok deneme yapmak gerekmektedir. Deneme sayısını en aza indirmek ve kısa sürede en iyi sonucu alabilmek için en uygun eps ve minpts değerlerini bulacak bir yöntem kullanılabilir. DBSCAN makalesinde de yer alan 4-dist adlı yöntem denenmiş fakat bu yöntemden uygun sonuçlar alınamamıştır.

Çok boyutluluk laneti(curse of dimensionality), matematiksel bir uzaya fazladan boyutlar eklenmesi durumunda hacmin exponansiyel olarak artması durumuna verilen addır. [52] Kümeleme algoritmalarına girdi olarak verilen veri setinin boyutunun artması da veri setinin hacminin artmasına neden olmakta ve çalışma zamanını uzatmaktadır. Paralel DBSCAN uygulaması da ilerleyen çalışmalarda çok boyutlu veri setleri ile denenmeli ve boyut sayısı ve çalışma zamanı arasındaki ilişki araştırılmalıdır.

Paralel DBSCAN uygulaması, tek işlemcili ve fiziksel olarak birbirinden ayrı bilgisayarlarda test edilmiştir. Günümüzde çok çekirdekli işlemci mimarisi gelişmiş

ve daha da gelişmektedir. Paralel DBSCAN uygulaması, ilerleyen çalışmalarda çok çekirdekli işlemci mimarisi üzerinde test edilmeli ve gerekiyorsa bu mimariye uygun hale getirilmelidir.

Bu tez çalışmasında OPTICS algoritmasının paralel hale getirilmesi denenmiş fakat algoritmanın çalışması sırasında hesaplanan erişilebilirlik uzaklığı ve iç uzaklık değerlerinin tüm işlem birimlerine dağıtılması gerekliliğinden dolayı vazgeçilmiştir. Paralel OPTICS uygulaması, iyi tasarlanması gereken ve üzerinde uzun süre çalışılması gereken bir uygulamadır. İlerleyen çalışmalarda OPTICS algoritmasının paralelleştirilmesi denenebilir.

KAYNAKLAR

1. Silahtaroğlu G., "Kavram ve Algoritmalarıyla Temel Veri Madenciliği", Papatya Yayıncılık, ISBN 978-975-6797-81-5, İstanbul, 2008.

2. Chakrabarti S., Nadeau T.P., Cox E., Neapolitan R.E., Frank E., Pyle D., Güting R.H., Refaat M., Han J., Schneider M., Jiang X., Teorey T.J., Kamber M., Witten I.H., Lightstone S.S., "Data Mining Know It All", Morgan Kaufmann, ISBN 978-0-12-374629-0, Massachusetts, 2009.

3. Hand D., Mannila H.,Smyth P., "Principles of Data Mining", The MIT Press, ISBN 0-262-08290-X, Massachusetts, 2001.

4. Han J. Kamber M., "Data Mining Concepts and Techniques, Second Edition", Morgan Kaufmann, ISBN 13: 978-1-55860-901-3, San Francisco, 2006.

5. Wang J., "Encyclopedia of Data Warehousing and Mining", Idea Group Publishing, ISBN 1-59140-557-2, Hershey PA, 2006.

6. Sumathi S., Sivanandam S.N., "Introduction to Data Mining and its Applications", Springer, ISBN-10 3-540-34350-4, Berlin Heidelberg, 2006.

7. Bramer M., "Principles of Data Mining", Springer, ISBN-10: 1-84628-765-0, London, 2007.

8. Orson D.L., Delen D., "Advanced Data Mining Techniques", Springer, ISBN: 978-3-540-76916-3, Berlin Heidelberg, 2008.

9. Kantardzic M., "Data Mining: Concepts, Models, Methods, and Algorithms", Wiley-IEEE Press, ISBN-13: 978-0471228523, Hoboken NJ, 2002.

10. Larose D.T., "Discovering Knowledge In Data-An Introduction To Data Mining", JOHN WILEY & SONS, ISBN 0-471-66657-2, Hoboken NJ, 2005.

11. Berry M.W., Browne M., "Lecture notes in data mining", World Scientific Publishing, ISBN 981-256-802-6, Singapore, 2006.

12. Ester M., Kriegel H.P., Sander j., Xu X., "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96), 1996.

13. Ankerst M., Breunig M.M., Kriegel H.P., Sander J., "OPTICS: Ordering Points To Identify the Clustering Structure", Proc. ACM SIGMOD'99 Int. Conf. on Management of Data, Philadelphia PA, 1999.

14. http://people.revoledu.com/kardi/tutorial/Similarity/index.html (27.10.2009)

15. Grama A, Gupta A., Karypis G., Kumar V., "Introduction to Parallel Computing, Second Edition", Addison Wesley, ISBN 0-201-64865-2, Essex, 2003.

16. El-Rewini H., Abd-El-Baar H., "Advanced Computer Architecture and Parallel Processing", JOHN WILEY & SONS, ISBN 0-471-46740-5, Hoboken, New Jersey, 2005.

17. http://en.wikipedia.org/wiki/Semaphore_(programming) (16.11.2009)

18. https://computing.llnl.gov/tutorials/parallel_comp/ (02.12.2009)

19. http://en.wikipedia.org/wiki/Linux (13.01.2010)

20. http://en.wikipedia.org/wiki/C++ (13.01.2010)

21. http://www.lam-mpi.org/ (17.01.2010)

22. http://en.wikipedia.org/wiki/R*-tree (20.01.2010)

23. Beckmann N., Kriegel H.P., Schneider R., Seeger B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles+", Praktische Informatik, Universitaet Bremen, 1990.

24. http://archive.ics.uci.edu/ml/ (25.01.2010)

25. http://uisacad2.uis.edu/dstar/data/clusteringdata.html (28.01.2010)

26. http://dbkgroup.org/handl/generators/ (31.01.2010)

27. Karasulu B., "Paralel Hesaplama Tekniğinin Monte Carlo Simulasyonunda Uygulanması", Maltepe Üniversitesi Fen Bilimleri Enstitüsü, 2006

28. Lucchese C., Orlando S., Perego R., “Fast and memory efficient mining of frequent closed itemsets, IEEE Transactions on Knowledge & Data Engineering”, 2006, 18:1, 21–36; Souliou D., Pagourtzis A., Drosinos N., Tsanakas P., “Computing frequent itemsets in parallel using partial support trees”, Journal of Systems & Software, 2006, 79:12, 1735–1743.

29. J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

30. R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In Proc. 1994 Int. Conf. Very Large Data Bases (VLDB’94), pages 144–155, Santiago, Chile, Sept. 1994.

31. L. Kaufman and P. J. Rousseeuw. Finding Groups inData: An Introduction to Cluster Analysis. JohnWiley & Sons, 1990.

32. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. 1998

ACM-SIGMOD Int. Conf. Management of Data (SIGMOD’98), pages 94–105, Seattle,WA, June 1998.

33. A. Hinneburg and D. A. Keim. An efficient approach to clustering in large multimedia databases with noise. In Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD’98), pages 58–65, New York, NY, Aug. 1998.

34. J. R. Quinlan. Induction of decision trees. Machine Learning, 1:81–106, 1986.

35. M.Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In Proc. 1996 Int. Conf. Extending Database Technology (EDBT’96), pages 18–32, Avignon, France, Mar. 1996.

36. W.Wang, J. Yang, and R.Muntz. STING: A statistical information grid approach to spatial data mining. In Proc. 1997 Int. Conf. Very Large Data Bases (VLDB’97), pages 186–195, Athens, Greece, Aug. 1997.

37. G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In Proc. 1998 Int. Conf. Very Large Data Bases (VLDB’98), pages 428–439, New York, NY, Aug. 1998.

38. J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In Proc. 1996 Int. Conf. Very Large Data Bases (VLDB’96), pages 544–555, Bombay, India, Sept. 1996.

39. J. R. Quinlan. Bagging, boosting, and C4.5. In Proc. 1996 Nat. Conf. Artificial Intelligence (AAAI’96), volume 1, pages 725–730, Portland, OR, Aug. 1996.

40. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In Proc. 2000 European Symp. Principle of Data Mining and Knowledge Discovery (PKDD’00), pages 13–23, Lyon, France, Sept. 1998.

41. M. Houtsma and A. Swami. Set-oriented mining of association rules. Research Report RJ 9567, IBM Almaden Research Center, San Jose, California, October 1993.

42. MacQueen, J. B. (1967). "Some Methods for classification and Analysis of Multivariate Observations". Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. 1. University of California Press. pp. 281–297.

43. http://en.wikipedia.org/wiki/Wavelet (07.02.2010)

44. http://en.wikipedia.org/wiki/SISD (13.02.2010)

45. http://en.wikipedia.org/wiki/SIMD (13.02.2010)

46. http://en.wikipedia.org/wiki/MISD (13.02.2010)

47. http://en.wikipedia.org/wiki/MIMD (13.02.2010)

48. http://www.rtreeportal.org/ (15.02.2010)

50. http://www.strehl.com/diss/node68.html (21.02.2010)

51. http://en.wikipedia.org/wiki/Parallel_slowdown (28.02.2010)

52. http://en.wikipedia.org/wiki/Curse_of_dimensionality (28.02.2010)

ÖZGEÇMİŞ

Zafer SEVER, 1981 yılında Manisa / Salihli'de doğdu. Öğrenimlerini sırasıyla Turgutlu Ülkü İlkokulu, Bornova Suphi Koyuncuoğlu Ortaokulu, İzmir Atatürk Lisesi'nde tamamladı. 1999 yılında Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği bölümünü kazandı ve 2003 yılında mezun oldu. Ekim 2003 ve Mart 2004 arasında EDAKOM A.Ş.'de, Mart 2004 ve Nisan 2007 arasında MEGA TELEKOM A.Ş.'de Yazılım Mühendisi pozisyonlarında çalıştı. Nisan 2007'de başladığı askerlik görevini Milli Savunma Bakanlığı İnşaat Emlak ve NATO Enf. Daire Başkanlığı'nda OBİ subayı olarak yaptı. Nisan 2008'den bugüne kadar HAVELSAN A.Ş.'de Uzman Yazılım Mühendisi olarak çalışmaktadır. Eylül 2008'de Maltepe Üniversitesi'nde başladığı Bilgisayar Mühendisliği Yüksek Lisans'ı tez aşamasında devam etmektedir.