T.C.
SÜLEYMAN DEMİREL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

STANDART DIŞI VERİLERİN PARALEL PROGRAMLAMA İLE ANALİZİ VE MODELLENMESİ

Hüseyin YAŞAR

Danışman
Yrd. Doç. Dr. Mehmet ALBAYRAK

YÜKSEK LİSANS TEZİ
ELEKTRONİK BİLGİSAYAR EĞİTİMİ ANABİLİM DALI
ISPARTA - 2017

TEZ ONAYI

Hüseyin YAŞAR tarafından hazırlanan **"Standart Dışı Verilerin Paralel Programlama ile Analizi ve Modellenmesi"** adlı tez çalışması aşağıdaki jüri üyeleri önünde Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü **Elektronik Bilgisayar Eğitimi Anabilim Dalı**'nda **YÜKSEK LİSANS TEZİ** olarak başarı ile savunulmuştur.

Danışman	**Yrd.Doç.Dr. Mehmet ALBAYRAK** Süleyman Demirel Üniversitesi
Jüri Üyesi	**Doç.Dr. Ecir Uğur KÜÇÜKSİLLE** Süleyman Demirel Üniversitesi
Jüri Üyesi	**Yrd.Doç.Dr. Ali KAVURUR** Mehmet Akif Ersoy Üniversitesi

Enstitü Müdürü **Prof.Dr.Yasin TUNCER**

TAAHHÜTNAME

Bu tezin akademik ve etik kurallara uygun olarak yazıldığını ve kullanılan tüm literatür bilgilerinin referans gösterilerek tezde yer aldığını beyan ederim.

Hüseyin YAŞAR

İÇİNDEKİLER

Sayfa

ÖZET

Yüksek Lisans Tezi

STANDART DIŞI VERİLERİN PARALEL PROGRAMLAMA İLE ANALİZİ VE MODELLENMESİ

Hüseyin YAŞAR

Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü
Elektronik Bilgisayar Eğitimi Anabilim Dalı

Danışman: Yrd. Doç. Dr. Mehmet ALBAYRAK

Bilgisayarların hayatımıza girmesiyle beraber dijital verilerin boyutları giderek artmaktadır. Dijital dünyada üretilen bu verilerin içinde benzerlerinden farklı davranış sergileyen standart dışı değerler (aykırı değerler) bulunabilmektedir. Bu değerlerin bulunması, sahtecilik tespiti, finans, tıp ve genetik bilimi gibi alanlarda büyük önem arz etmektedir.

Büyük veri setlerinde standart dışı değerlerin tespiti için daha çok veri madenciliği yöntemlerinden kümeleme teknikleri kullanılmıştır. Gürültülü ve aykırı noktalara karşı hassas olan kümeleme algoritmalarından yoğunluk tabanlı DBSCAN algoritması standart dışı değerlerin tespitinde kullanılabilmektedir.

Bu çalışmada standart dışı değerlerin tespiti için C# programlama dilinde DBSCAN algoritması kullanılarak bir uygulama geliştirilmiştir. Geliştirilen uygulamada veri sayıları farklı 4 adet veri seti kullanılmış ve analiz edilmiştir. Veri setlerinin daha kısa sürede analiz edilebilmesi için .Net 4.0 ile gelen TPL (Task Parallel Library) içindeki paralel sınıf üyeleri kullanılmıştır.

Veri setlerinde yapılan analizlerde DBSCAN algoritmasının standart dışı değerlerin tespiti açısından kullanılabilir olduğu görülmüştür. Performans açısından ise paralel programlamanın veri sayısı arttıkça daha kullanışlı olduğu sonucuna varılmıştır.

Anahtar Kelimeler: Standart dışı veri, Kümeleme, DBSCAN, Paralel programlama.

2017, 66 sayfa

ABSTRACT

M.Sc. Thesis

PARALLEL PROGRAMMING WITH NON-STANDARD DATA ANALYSIS AND MODELLING

Hüseyin YAŞAR

Süleyman Demirel University
Graduate School of Applied and Natural Sciences
Department of Electronics and Computer Education

Supervisor: Asst. Prof. Dr. Mehmet ALBAYRAK

The size of digital datas is increasing with the penetration of computers into our lives. Non-standard values (outliers) which behave differently than similar ones can be found in these datas. Determination of these values has a great importance in areas such as fraud detection, finance, medicine and genetic science.

Clustering techniques have been used more frequently than data mining methods for determination of the non-standard values in large data sets. The density-based DBSCAN algorithm, one of the clustering algorithms that are sensitive to noisy and outlier spots can be used to detect non-standard values.

In this study, for detecting nonstandard values an application was developed by using the DBSCAN algorithm in the C # programming language. In the developed application, 4 data sets with different data numbers were used and analyzed. For analyzing data sets in a shorter time, parallel class members in the TPL (Task Parallel Library) which comes with Net 4.0 were used.

It has been found that DBSCAN algorithm can be used for the determination of the non-standard values in data set analysis. From the point of performance, it was seen that usability of parallel programming increased due to the increase in the number of data.

Keywords: Non standart data, Clustering, DBSCAN, Parallel programming.

2017, 66 pages

TEŞEKKÜR

Bu araştırma için beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan değerli Danışman Hocam Yrd. Doç. Dr. Mehmet ALBAYRAK'a teşekkürlerimi sunarım.

4199-YL1-14 No`lu Proje ile tezimi maddi olarak destekleyen Süleyman Demirel Üniversitesi Bilimsel Araştırma Projeleri Birimine teşekkür ederim.

Tezimin her aşamasında beni yalnız bırakmayan sevgili eşim Gülçin YAŞAR'a ve aileme sonsuz sevgi ve saygılarımı sunarım.

Hüseyin YAŞAR
ISPARTA, 2017

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

API	Application Programming Interface
CPU	Central Process Unit
DBSCAN	Density-Based Spatial Clustering Of Applications With Noise
eps	Epsilon
GPS	Global Positioning System
GPU	Graphics Processing Unit
kNN-dist	k-Nearest Neighbor Distance
LAM	Local Area Multicomputer
LINQ	Language Integrated Query
LOF	Local Outlier Factor
MIMD	Multiple Instruction Multiple Datastream
MISD	Multiple Instruction Single Datastream
minPts	Minimum Points
MPI	Message Passing Interface
MSSQL	Microsoft Structured Query Language
PFX	Parallel Framework
PLINQ	Parallel Language Integrated Query
SIMD	Single Instruction Multiple Datastream
SISD	Single Instruction Single Datastream
TPL	Task Parallel Library

1. GİRİŞ

Dünyada her gün, 2.5 kentilyon bayt dijital veri yaratılmaktadır. Bugün dünyadaki verilerin % 90'ı son iki yılda yaratılmıştır. Bu veriler iklim bilgisi toplamak için kullanılan sensörlerden alınan bilgiler, sosyal medya sitelerindeki mesajlar, dijital resim ve videolar, satın alma işlem kayıtları ve cep telefonu GPS sinyalleridir (IBM, 2016). Bunlar gibi daha pek çok alanda bilgi üretilmektedir. Bu verilerdeki standart dışı değerleri bulmak, verilerin büyüklüğünden dolayı uzun zaman almaktadır.

Standart dışı değer (aykırı değer), bir kümedeki verilerin geneline uymayan ve verinin genel dağılımından sapmış olan anormal değerlerdir. Veri kümelerinde görülen aykırılıklar, hatalı olabileceği gibi gerçeği de yansıtabilir. Bu noktada dikkat edilmesi gereken ilk şey verinin gerçeği yansıtıp yansıtmadığına karar vermektir. Çok boyutlu verilerde bir örneğin aykırı olup olmadığını belirleyebilmek oldukça zordur. Nitekim bir örneğin bir özelliğinde görülen aykırılık diğer özelliklerin normal dağılım içerisinde olması durumunda o örneğin normalmiş gibi algılanmasına neden olabilir. Çünkü aykırı değerler ve veriler birbirlerini etkileyebilme hatta gizleyebilme özelliğine sahiptir (Güçlü, 2012).

Bilgisayar donanımları, yazılımların ihtiyaçlarına cevap vermekte zorlanmaktadırlar. Donanım parçalarındaki hafıza ya da bit derinliği arttırılabilirken işlemci hızı neredeyse fiziksel limitlere ulaşmıştır. Donanım üreticileri fiziksel limitlere dayanan işlemci hızı yerine, bilgisayarlarda kullanılan işlemci sayısını arttırmaktadırlar. Bilgisayar yazılımlarının çok işlemcili bilgisayarlardan daha verimli yararlanabilmesi için paralel olarak programlanması gerekmektedir (Ercan vd., 2013).

Bilgisayar sistemlerinin performansı, işlemci saat frekansı ve bellek kapasitesi ile doğrudan orantılıdır. Birim zamanda işlenen komut sayısını arttıracağından daha yüksek frekanslarda çalışabilen işlemciler kullanılarak performans arttırılabilir. Ancak işlemci frekanslarını arttırmak fiziksel sebeplerden dolayı sınırlıdır. Diğer

yandan performans, ilave bellek kullanılarak, bellek kapasitesinin genişletilmesiyle de arttırılabilir. Ancak bu da belirli değerlerden sonra ekonomik değildir. Görülüyor ki hem fiziksel hem de ekonomik nedenlerden dolayı tek işlemcili bilgisayar sistemlerinin performansı sınırlıdır. Sonuç olarak, performansı arttırmak için çeşitli boyutlarda paralellik işlemci içinde veya dışında ekonomik çözümler için kaçınılmaz olmuştur (Durmuş, 2003).

Paralel programlamaya olan ihtiyacı ifade edebilmek için farklı bir bakış açısı sunmak gerekirse şöyle bir örnek verilebilir. Bilgisayar ve ilk işletim sistemlerinin tarihini göz önüne alırsak, teknolojik gelişmelerin çözmüş olduğu problemler kesinlikle mevcut olabileceği gibi göz ardı edilemeyecek bir gerçek vardır. Bu en kısa ifadesi ile ister 1990 yılında olunsun ister 2012 yılında, dünyadaki mevcut teknolojik gelişmeler, bazı çalışmaları istenilen seviyede hızlandırmıyorsa ne yapılacağı gerçeğidir. Ya teknolojinin istenilen seviyeye gelmesi beklenecek ki bu bazı durumlarda çok uzun zaman alabileceği gibi imkânsız seviyede bir artışa ihtiyaç duyan problemler bile olabilir, ya da mevcut teknoloji kullanılarak, nasıl bir çözüm üretileceğini düşünmek gerekir. Mevcut kaynaklarla bir problemin çözümünü hızlandırmak için, problemi parçalara ayırarak, her parçayı farklı bilgisayarlarda çözmek en optimum çözüm olacaktır. Yani tercih edilmesi gereken yöntem kesinlikle paralel programlamadır (İnce, 2013).

Bir algoritmanın paralelleştirilmesinin temel amacı, çoklu işlemci çekirdeklerinin avantajlarından en etkili ve doğru şekilde yararlanabilmektir. Çok büyük veriler kullanan ve çok uzun hesaplama zamanı gerektiren uygulamalarda, hesaplama süresinin kısaltılması için en etkili yöntemlerin birisi, işlemleri çoklu çekirdeklere paralel olarak paylaştırmaktır. Böylece hesaplama süresinin kısaltılması ve dolayısıyla performans artışı hedeflenmektedir (Akçay ve Erdem, 2010).

Bilişim sektörünün geldiği nokta ve buna bağlı olarak ihtiyaçların her geçen gün hızla artması birçok problemi de beraberinde getirmektedir. Bilgi teknolojileri sektörü de bu konuda pastadan en fazla dilimi alarak ilk sıraya oturmaktadır.

Günümüzde gerek kurumsal, gerekse akademik çalışmalarda kullanılan veri boyutlarının artması son kullanıcılardaki performans olgusunu ilk sıraya yerleştirmiştir. Artık geliştirilen bir sistemin kararlı ve hatasız çalışmasının yanında en kısa sürede en iyi sonucu elde etmek de isteklerin başında gelmektedir. Hız konusu, üretim sektörü, mühendislik hesapları, askeri simülasyon projeleri ve hava tahmin hesaplamaları için hayati öneme sahiptir (Güneş, 2011).

Standart dışı değerlerin tespitinde istatistiksel yöntemler, veri madenliği algoritmaları ve regresyon analizi gibi yöntemler kullanılmaktadır.

Bu çalışmada, standart dışı veri analizi, veri madenciliği ve paralel programlama konuları üzerinde durulmuştur. Standart dışı verinin ne olduğu, standart dışı veri analizinin nasıl yapıldığı anlatılmıştır. Veri madenciliğinin tanımı, temel amacı, veri madenciliği süreci ve kullanılan yöntemler ve veri madenciliğinde kümeleme algoritmaları anlatılmıştır. Çalışmada bir kümeleme yöntemi olan DBSCAN algortiması hakkında bilgi verilmiştir. Paralel hesaplama ve programlama tekniklerine değinilmiştir. Meteoroloji Genel Müdürlüğü Veri Arşiv Sisteminden alınan 2015 yılına ait meteorolojik veriler, Yahoo Webscobe sitesinden alınan S5 - A Labeled Anomaly Detection Dataset, version 1.0(16M) veri seti ve Alman Yapay Zekâ ve Araştırma Merkezinden alınan veri seti DBSCAN algoritması kullanılarak geliştirilen yazılım sayesinde standart dışı değerler açısından analiz edilmiştir. Geliştirilen programın hız ve performansını artırmak için Microsoft .Net paralel programlama yöntemleri kullanılmıştır. Seri ve paralel programlamanın hız ve performans karşılaştırılması yapılmıştır.

2. KAYNAK ÖZETLERİ

Tekbir (2009), bölümleme ve birleştirme temeline dayalı R-P-DBSCAN (Recursive-Partitioned DBSCAN) algoritmasında büyük hacimli kümeleri daha küçük parçalara ayırırarak klasik DBSCAN algoritması ile kümeleme yapmaktadır. Daha sonra kümelenen her parça birleştirilmiş ve bütün bir veri kümesi kümelenmiştir. Bu yöntemle büyük veri kümelerinde DBSCAN algoritmasına göre %85'e kadar kümeleme süresinden kazanç elde etmiştir.

Atılgan (2014), kümeleme yöntemlerini ve paralel hesaplama araçlarını incelemiştir. K-means ve DBSCAN algoritmalarını C programlama dili ve OpenMP programlama arayüzü kullanarak, paralelleştirmenin etkisini göstermiştir.

Kayım (2015), büyük verilerle veri madenciliği algoritmalarının birlikte nasıl kullanalabileceklerini ele almıştır. DBSCAN ve k-means algoritmalarının paralelleştirilmesini incelemiş ve büyük veriler üzerinde performans ve hız kriterlerini olumlu etkilediğini göstermiştir. Küçük ve orta ölçekli projelerde paralelleştirme yöntemi ile veri madenciliği algoritmalarının paralleştirilip kullanılabileceğinin sonucuna varmıştır.

Çelik vd. (2011), çalışmalarında aylık sıcaklık verileri üzerinde DBSCAN algoritması kullanarak standart dışı değerlerin bulunması üzerinde çalışmışlardır. Standart dışı değerlerin bulunmasında DBSCAN algoritması istatistiksel yöntemlerle karşılaştırılmış ve DBSCAN algoritmasının bazı avantajlarının olduğuna değinmişlerdir. DBSCAN algoritmasının veri setinde aşırı değerler olmamasına rağmen anormallikleri tespit edebildiğini belirtmişlerdir.

Wu vd. (2012), telefon çağrılarındaki anormal aramaları tespit ederek, dolandırıcılık tespiti konusunda çalışma yapmışlardır. Mobil telefon kullanıcılarının artmasıyla beraber çağrı sayısındaki artış büyük veriler oluşturduğundan anormal çağrıları tespit etmek için LOF (Local Outlier Factor) algoritmasına dayalı bir küme katsayısı yaklaşımı önermişlerdir. Ayrıca

önerdikleri yöntemde performans artışı için MapReduce'a dayalı paralel hesaplamayı kullanmışlardır.

Matsumoto vd. (2015), standart dışı değerlerin tespiti için önerdikleri yöntemde mikro kümeleme ve yoğunluk tabanlı kümeleme yöntemlerini kullanmışlardır. Önerdikleri algortimayı OpenCL platformunda GPU programlama teknikleri ile geliştirmişlerdir.

Breunig vd. (2000), tarafından yapılan çalışmada uç nokta analizleri için yoğunluk tabanlı bir yaklaşımın nasıl olacağı belirtilmiştir. Bir noktanın, uç nokta olup olmadığını tespit etmek için kendisini çevreleyen komşu noktalardan ne kadar uzak olduğu incelenmiştir. Bu inceleme ile her noktaya LOF (local outlier factor, bölgesel uç faktör) adında bir derece verilmiştir. Çalışma sonucunda verilen deneysel sonuçlarda, yöntemin uygulanabilir bir yöntem olduğu belirtilmiştir.

Sever (2010), çalışmasında DBSCAN algoritmasının paralelleştirilmesi için LAM/MPI (Local Area Multicomputer/Message Passing Interface) kütüphanesini kullanmıştır. 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 Kanununa uygun olarak çalışma süresinin kısaldığı, bununla birlikte küme oluşturma performansının ve kalitesinin etkilenmediği görülmüştür.

Durrani (2013), çalışmasında iki popüler kümeleme algoritması olan, K-Means ve DBSCAN algoritmalarının paralel sürümlerini geliştirmiş ve deneysel olarak veri miktarı arttıkça başarımdaki iyileşmenin sürdüğü gözlenerek, bu paralel sürümlerin paralel ortamlar için çok uygun olduğuna değinmiştir.

Kumar vd. (2013), aykırı değer tespiti için kümeleme yaklaşımını temel alan bir çalışma yapmışlardır. Veri seti üzerinde k-medoids algoritmasının bir türevi olan PAM (Partitioning Around Medoids) algoritmasını uygulamışlardır. Küçük

kümeleri aykırı küme olarak kabul etmişlerdir. Mevcut kümedeki medoid ile her bir noktanın arasındaki mutlak mesafelerin hesaplanmasına dayanan bir yöntemle olağandışı değerleri bulan bir yaklaşım önermişlerdir.

Vatansever ve Büyüklü (2009), çalışmalarında kümeleme analizlerinde sapan değerlerin tespitinde, potansiyel küme yapılarının, uygun küme sayılarının keşfi, uygun kümeleme algoritmalarının seçimi ve küme sonuçlarının değerlendirilmesinin önemine değinmişlerdir. Çeşitli, sayısal yöntemlerle bu tür sorunların üstesinden gelinebileceğini ancak sayısal yöntemlerle bazı önemli olabilecek ayrıntılar gözden kaçırılabileceğine değinmişlerdir. Görsel veri madenciliği yöntemleri yardımıyla, insan algı sisteminin de devreye girmesiyle etkili bir şekilde, sapan değerlerin, potansiyel küme yapılarının, küme sayılarının keşfedilebileceği, uygun kümeleme algoritmalarının seçilebileceği ve küme sonuçlarının değerlendirilebileceği gösterilmiştir.

Petrovskiy (2003), çalışmasında veri madenciliğinde kullanılan aykırı değer bulma algoritmalarının avantaj ve dezavantajlarına değinmiştir. Çalışmasında yeni bir aykırı değer bulma algoritması önermiştir. Algoritmanın bulanık küme teorisi ve çekirdek işlevlerinin kullanımına dayanan ve mevcut yöntemlere kıyasla birtakım avantajlara sahip olduğunu belirtmiştir. Önerdiği sistemi saldırı tespit sistemleri olarak adlandırmıştır.

3. MATERYAL VE YÖNTEM

Çalışmanın bu bölümünde veri madenciliğinin tanımı, amaçları, analiz yöntemleri, süreci konularından bahsedilmiştir.

3.1. Veri Madenciliği Nedir?

Veri madenciliği, bilgi ve iletişim çağında en çok tercih edilen teknolojilerden biridir. Günümüzde artan veri sayısı, bilgisayar ve iletişim teknolojilerindeki gelişmeler çok daha fazla verinin hızlı bir şekilde depolanmasına, işlenmesine ve bilgiye dönüştürülmesine imkân vermektedir. Bu yüzden, büyük miktardaki verilerden yararlanılmasını ve bu verilerin anlamlı hale getirilmesini sağlayan teknikler büyük önem taşımaktadır. Bu gibi durumlarda çözüm sağlayan veri madenciliği son yıllarda yoğun olarak kullanılmaya başlanmıştır. Veri madenciliği, istatistik ve başka alanlarda farklı isimler ile birlikte birçok bilim dalında kullanılmış ve bilgisayar teknolojisindeki hızlı gelişmelerden sonra, veri arkeolojisi, veri taranması, veri balıkçılığı, özbilgi çıkarımı veya veri tabanlarında özbilgi keşfi gibi farklı isimlerle de anılmıştır (Koldere Akın, 2008).

Veri madenciliği, büyük boyutlu gözlenmiş verilerden kuralların ve modellerin ortaya çıkarılmasıdır (Shaw vd., 2001). Bir başka ifade ile veri madenciliği, veri tabanları veya veri ambarlarında yer alan veriler içindeki gizli örüntüleri ve ilişkileri bulmak için istatistiksel algoritmaları ve yapay zekâ yöntemlerini kullanan karmaşık bir veri arama yeteneği olarak tanımlanabilir (Gargano ve Raggad, 1999). Veri madenciliği, aynı zamanda bilgisayar bilimini, makine öğrenmesini, veritabanı yönetimini, matematiksel algoritmaları ve istatistiği birleştiren disiplinler arası bir alandır (Liao, 2003).

Veri madenciliğini istatistiksel bir yöntemler serisi olarak da görmek mümkün olabilir. Ancak veri madenciliği, geleneksel istatistikten birkaç yönde farklılık gösterir. Bu bağlamda, veri madenciliği insan merkezlidir ve bazen insan – bilgisayar arayüzü birleştirilir. Veri madenciliği sahası, istatistik, makine bilgisi, veritabanları ve yüksek performanslı işlem gibi temelleri de içerir.

Veri madenciliğinin amaçları genellikle sınıflandırma, kümeleme, tahmin, öngörü ve benzer gruplama olarak sıralanmaktadır. Amaçlardan biri olan kümeleme, istatistiksel veri analizi, örüntü tanıma vb. birçok alanda sık kullanılmaktadır. Veri tabanlarındaki verilerin gruplar veya kümeler altında toplanarak, benzer özelliklere sahip nesnelerin bir araya gelmesini sağlayan kümeleme algoritmaları veri madenciliği alanında büyük bir öneme sahiptir (Sarıman, 2011a).

3.2. Veri Madenciliğinin Temel Amacı

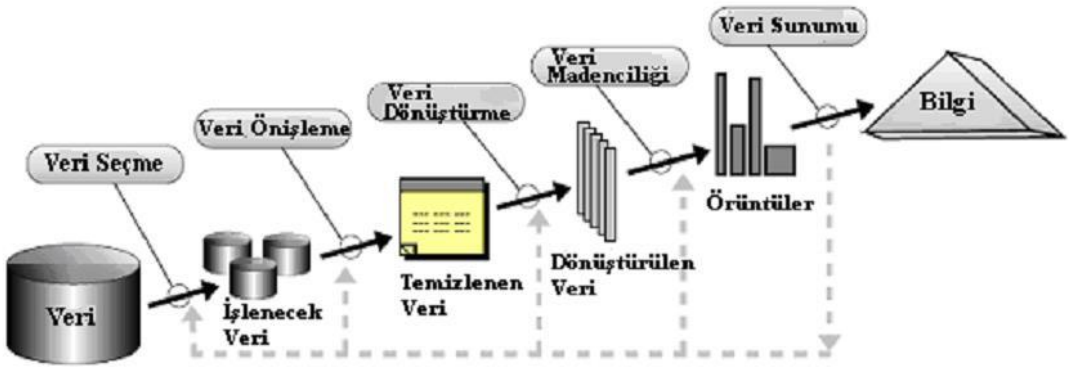
Veri madenciliği süreci genellikle uygulamadaki iki temel soruna cevap bulmayı amaçlamaktadır. Bunlardan birincisi mevcut veri tabanından tahminler yapmak (tahmin edici model), ikincisi ise veriden elde edilecek davranışları (tanımlayıcı bilgi) betimlemektir (Moshkovich vd., 2002).

Tahmin edici modeller, verideki eğilimleri, örüntüleri ve gizli ilişkileri bulmaya yardımcı olurken; tanımlayıcı bilgi ilginç örüntü ve ilişkileri bulmak için veriyi, görselleştirme ve keşfetmeye odaklamaktadır (Kiang ve Kumar, 2001).

3.3. Veri Madenciliği Süreci

Veri Madenciliği algoritmalarının, analiz edilen verilerin özelliklerinin bilinmemesi durumunda fayda sağlaması mümkün değildir. Bu nedenle tüm aşamalardan önce veriye ait özelliklerinin anlaşılması gereklidir (Turgut, 2012).

Veri madenciliği, çeşitli karar verme işlemlerinin etkileşimli ve tekrarlayan aşamalar sonucunda gerçekleştirilmesidir. Bu aşamalar, Brachman ve Anand (1996) tarafından Şekil 3.1'deki gibi gösterilmektedir.



Şekil 3.1. Veri madenciliği süreci (Akgül, 2016)

1. Veri Seçme: Analiz yapılan konu ile ilgili verilerin veritabanından seçilip alınması işlemidir.
2. Veri Önişleme: Gürültülü, tutarsız verilerin işlenmesi veya gerekiyorsa veri setinden çıkarılması, eksik verilerin uygun şekilde düzenlenmesi, farklı kaynaklardan alınan veritabanlarının düzenlenmesidir.
3. Veri Dönüştürme: Özetleme veya biriktirme işlemleri ile verinin madencilik işlemleri için uygun hale getirilmesidir.
4. Veri Madenciliği: Veritabanındaki anlamlı ve kullanışlı örüntülerin bulunması işlemidir.
5. Veri Sunumu: Bulunan örüntülerin gösterim ve sunumu için uygun tekniklerin belirlenmesidir.

3.4. Veri Madenciliğinde Kullanılan Yöntemler

Veri madenciliği teknikleri işlevlerine göre 3 temel gruba ayrılır (Akbulut, 2006):

1. Sınıflama (Classification) ve regresyon (regression)
2. Kümeleme (Clustering)
3. Birliktelik kuralları ve sıralı örüntüler (Association rules and sequential patterns)

Gerek tanımlayıcı gerekse tahmin edici modellerde yoğun olarak kullanılan belli başlı istatistiksel yöntemler; sınıflama (classification) ve regresyon (regression),

kümeleme (clustering), birliktelik kuralları (association rules) ve ardışık zamanlı örüntüler (sequential patterns), bellek tabanlı yöntemler, yapay sinir ağları ve karar ağaçları olarak gruplandırılabilir. Sınıflama ve regresyon modelleri tahmin edici, kümeleme, birliktelik kuralları ve ardışık zamanlı örüntü modelleri tanımlayıcı modellerdir (Domouchel, 1999).

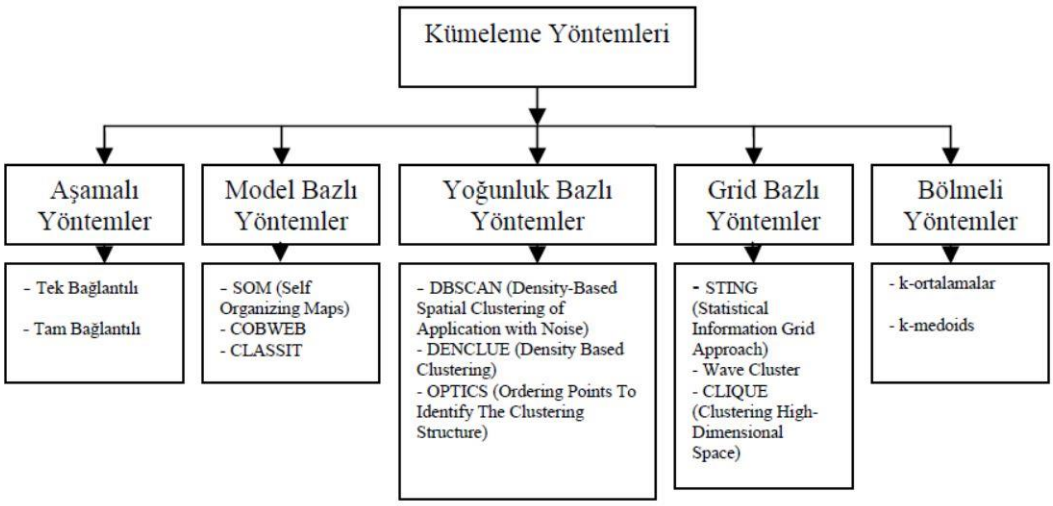
3.4.1. Kümeleme

Kümeleme, verideki benzer kayıtların gruplandırılmasını sağlayan bir tekniktir. Kümelemede, genellikle DBSCAN, k-means algoritması ya da Kohonen şebekesi gibi yöntemler kullanılmaktadır. Hangi yöntem kullanılırsa kullanılsın süreç aynı şekilde işler. Her kayıt var olan kümelerle karşılaştırılır. Bir kayıt kendisine en yakın kümeye atanır ve bu kümeyi tanımlayan değeri değiştirir. Optimum çözüm bulununcaya kadar kayıtlar yeniden atanır ve küme merkezleri ayarlanır (Hui ve Jha, 2000).

Veri Madenciliğinde kullanılan kümeleme teknikleri aşağıdaki gibi sıralanabilir (Özdamar, 2002):

1. Merkeze dayalı ayırıcı kümeleme teknikleri
2. Hiyerarşik kümleme teknikleri
3. Yoğunluğa dayalı kümeleme teknikleri
4. Grafik esaslı kümeleme teknikleri

Vatansever ve Büyüklü (2009) ise kümeleme yöntemlerini Şekil 3.2'deki gibi gruplandırmışlardır.



Şekil 3.2. Kümeleme yöntemleri (Vatansever ve Büyüklü, 2009)

3.4.1.1. Yoğunluk tabanlı kümeleme

Yoğunluk tabanlı yaklaşımda, verilen bir veritabanında (veri noktalarından oluşan set) bulunan ve bir birbirine çok yakın noktaların belirli bir yoğunluğa bağlı olarak oluşturduğu kümeler bulunmaktadır. Bu şekildeki kümeler herhangi bir biçimi alabilirler. Veri setini oluşturan elemanlar küme içindeki yoğunluklarına göre kimi zaman içi boş bir halka kimi zaman da kıvrılmış bir yay şeklinde kümelenebilir. Bir veritabanındaki herhangi bir kümeye ait olmayan veri noktaları kümesinin gürültü olduğu düşünülür. Bunun yanı sıra, kümenin içine uzanan çekirdek nokta bir tanedir ve bir sınır nokta da kümenin sınırına uzanan bir noktadır (Tekbir, 2009).

Yoğunluk tabanlı kümeleme yöntemlerinin öne çıkan özelliği, farklı ve düzensiz şekillerdeki kümeleri belirleyebilmesidir. Bir kümeyi oluşturmak için, birbirine bağlantılı yoğun noktalar belirlenir. Yani bir küme, yoğunluğun yönlendirdiği herhangi bir yönde genişleyebilir. Böylece herhangi bir şekildeki küme oluşturulabilir. Yoğunluk değerlendirmesinin doğal bir etkisi olarak, yoğunluk tabanlı kümeleme yöntemleri gürültü (sıradışı veya anormal özelliklere sahip veri) ile baş edebilirler. Ayrıca, bu yaklaşımı benimseyen yöntemler genellikle düşük hesaplama karmaşıklığına sahiptirler. En belirgin dezavantaj ise, ortaya çıkan kümelerin nasıl değerlendirileceği ve yorumlanacağıdır (Atılgan, 2014).

Yoğunluk tabanlı kümeleme yöntemlerinin en yaygın olanı DBSCAN (Density-based spatial clustering of applications with noise) algoritmasıdır. DBSCAN algoritması aykırı değerlere duyarlı bir yöntem olduğundan tezde bu algoritma kullanılmıştır.

3.4.1.1.1. DBSCAN algoritması

DBSCAN algoritması, Ester, Kriegel, Sander ve Xu tarafından KDD'96 konferansında sunulmuştur. Bu algoritma, nesnelerin komşuları ile olan mesafelerini hesaplayarak belirli bir bölgede önceden belirlenmiş eşik değerden daha fazla nesne bulunan alanları gruplandırarak kümeleme işlemini gerçekleştirir.

DBSCAN algoritması için çekirdek nesne, eps, minPts, doğrudan yoğunluk erişilebilir nokta, yoğunluk erişilebilir nokta, yoğunluk bağlı nokta terimleri temel kavramlardır. Algoritma, eps ve minPts değerlerini giriş parametresi olarak alır. Veritabanındaki herhangi bir nesneden başlayarak tüm nesneleri kontrol eder. Eğer kontrol edilen nesne daha önce bir kümeye dâhil edilmiş ise işlem yapmadan diğer nesneye geçer. Eğer nesne daha önce kümelenmemiş ise, bir bölge sorgusu yaparak nesnenin eps komşuluğundaki komşularını bulur. Komşu sayısı minPts'den fazla ise, bu nesne ve komşularını yeni bir küme olarak adlandırır. Daha sonra, önceden kümelenmemiş her bir komşu için yeni bölge sorgusu yaparak yeni komşular bulur. Bölge sorgusu yapılan noktaların komşu sayıları minPts'den fazla ise kümeye dâhil eder (Bilgin ve Çamurcu, 2005).

DBSCAN, içinde gürültünün de olduğu uzaysal veritabanlarında, noktaların yoğunluğuna bağlı olarak kümeleme yapmak için kullanılan bir yöntemdir. Yöntem dışarıdan iki giriş parametresi alıp, rastgele şekillerde kümeleme yapabilen, uzaysal büyük ölçekli veritabanlarında etkin bir şekilde kullanılabilen bir yöntemdir (Ester vd., 1996).

Algoritma kümeleri oluştururken nesnelerin yoğunluklarını hesaba katar. Kümeler, yüksek yoğunluklu veri nesneleri ile tanımlanmakta; düşük yoğunluklu

nesnelerin bulunduğu kümeler ise aykırı veya gürültülü noktaları göstermektedir. DBSCAN, özellikle büyük veritabanları ve gürültülü nesneler içeren veri setleri için oldukça kullanışlıdır. Bunun yanısıra farklı büyüklük ve şekillerdeki kümelerin belirlenmesinde de sıkça kullanılmaktadır (Moreira vd., 2005).

Aşağıda DBSCAN algoritmasının psido (sözde) kodu verilmiştir.

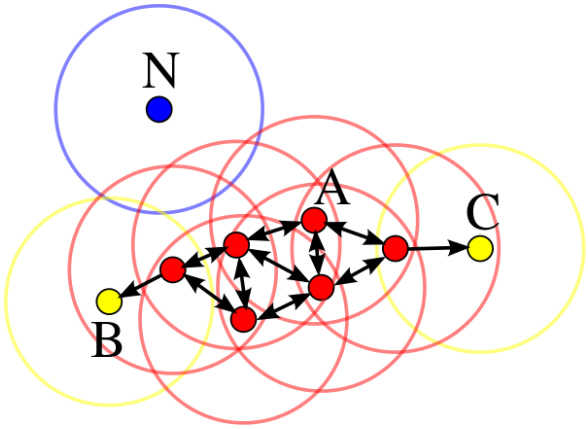
```
DBSCAN(D, eps, MinPts) {
  C = 0
  for each point P in dataset D {
    if P is visited
      continue next point
    mark P as visited
    NeighborPts = regionQuery(P, eps)
    if sizeof(NeighborPts) < MinPts
      mark P as NOISE
    else {
      C = next cluster
      expandCluster(P, NeighborPts, C, eps, MinPts)
    }
  }
}

expandCluster(P, NeighborPts, C, eps, MinPts) {
  add P to cluster C
  for each point P' in NeighborPts {
    if P' is not visited {
      mark P' as visited
      NeighborPts' = regionQuery(P', eps)
      if sizeof(NeighborPts') >= MinPts
        NeighborPts = NeighborPts joined with NeighborPts'
    }
```

```
    if P' is not yet member of any cluster
      add P' to cluster C
  }
}
regionQuery(P, eps)
  return all points within P's eps-neighborhood (including P)
```

DBSCAN algortiması eps ve minPts diye adlandırılan iki parametreye ihtiyaç duyar. İki nokta arasındaki uzaklık, eps değerinden büyük ve bu noktaların sayısı MinPts sayısından daha fazla ise bu nokta asıl nokta olarak değerlendirilir. Asıl nokta tespit edildikten sonra çevresindeki diğer noktaların da asıl olup olmadıkları test edilir. Bu şekilde bulunan asıl noktalar, belirli bir yoğunluğa ulaştığı için bir küme olarak değerlendirilir (Ester vd., 1996). DBSCAN bir noktayı çevresel komşuluklarına göre incelediği için standart dışı değerlerin bulunmasında başarı sağlamıştır.

Şekil 3.3'de minPts değeri 4 olarak alınmış ve A noktası ve diğer kırmızı noktalar temel noktalardır. Çünkü eps yarıçapındaki bu noktaları çevreleyen alan en az 4 nokta içerir (noktanın kendisi de dâhil). Hepsine birbirinden ulaşılabilir ve tek bir küme oluşturmuştur. B ve C noktaları temel noktalardan değildir, ancak A'dan (diğer çekirdek noktalar üzerinden) erişilebilir ve böylece kümeye ait olmuş olur. Nokta N ise, ne bir çekirdek nokta ne de yoğunluğa erişilebilir olmayan bir gürültü noktasıdır.



Şekil 3.3. DBSCAN algortimasının çalışma mantığı (Wikipedia, 2016)

DBSCAN algoritmasının çalışması aşağıdaki gibidir.

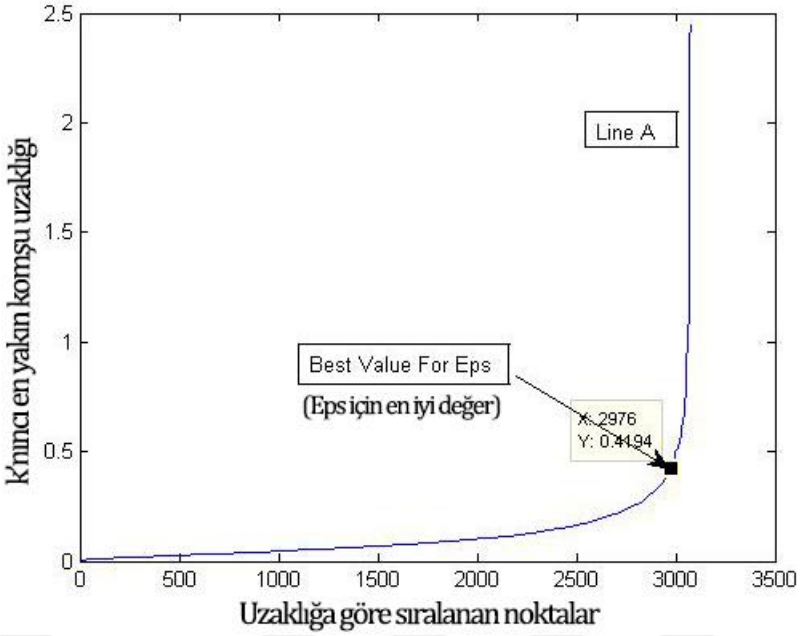
1. Rastgele bir nokta seçilir.
2. Seçilen bu rastgele noktanın eps uzaklığı içerisindeki komşuları bulunur.
3. Eğer komşu sayısı minPts sayısına eşit ya da büyük ise bir küme oluşturulur. minPts sayısından küçük ise bu nokta gürültü olarak işaretlenir. Gürültü olarak işaretlenen bu nokta ilerde başka bir kümeye dâhil edilebilir.
4. Bir nokta bir kümeye dâhil edildi ise bu durumda tüm epsilon komşuları da bu kümeye dâhil edilir. Bu işlem eklenecek nokta kalmayana kadar devam ettirilir ve küme oluşturulmuş olur.
5. Yeniden daha önce üzerinde işlem yapılmamış bir nokta rastgele seçilir ve işlem tekrarlanır.

3.4.1.1.2. Uygun eps ve minPts değerlerinin seçimi

DBSCAN algroritması, ihtiyaç duyduğu eps ve minPts parametrelerinin dışardan girilmesini ister.

Algoritma, analizi yapılacak veri setindeki her noktanın eps komşuluğunu kontrol eder. Bir noktanın komşu sayısı minPts'den büyük veya eşitse bir küme oluşturulur. Nokta ve komşuları bu kümeye eklenir. Veri setindeki her nokta için bu işlem tekrarlanır. Bir noktanın komşu sayısı minPts değerinden küçükse nokta gürültü olarak işaretlenir.

DBSCAN algoritmasının en iyi sonucu vermesi için eps ve minPts değerlerinin seçimi çok önemlidir. Elbatta ve Ashour (2013), eps ve minPts değerlerini belirlemek için noktaların k'nıncı en yakın komşu mesafelarine bakmak gerektiğini belirtmişlerdir ve noktaların uzaklıklarını kNN-dist olarak adlandırmışlardır. kNN-dist tüm noktalar için hesaplanır ve bu k mesafeleri artan bir düzende grafikleştirilir. Grafiğimizde keskin bir değişikliğe denk gelen k değeri optimum eps değeri olarak belirlenir. Şekil 3.4'de k=3 değeri için örnek grafik gösterilmiştir.



Şekil 3.4. k=3 değeri için optimum eps değeri (Elbatta ve Ashour, 2013)

minPts değeri ise, kullanıcı tarafından belirlenen k değeri olarak kabul edilir (STHDA, 2016). Bu yöntemle bulunan eps ve minPts değeri, k parametresine bağlıdır.

3.5. Standart Dışı Veri Analizi

Bu başlık altında standart dışı veri ve analiz yöntemleri hakkında bilgi verilmiştir. Standart dışı veri işlemenin önemi, istatistiksel analiz yöntemleri, veri setinin farklı oluşu problemi özelleştirmektedir.

3.5.1. Standart dışı veri (Aykırı değer)

Veri setlerinin bilimsel olarak değerlendirilebilmesi için bazı ön işlemlerden geçmesi gereklidir. Verinin eksik veya tutarsız olması gibi sorunların yanı sıra aşırı veya uç değerler içermesi özellikle istatistiksel analizlerin yapılması için dikkate alınması gereken bir sorundur (Ovla ve Taşdelen, 2012).

Veri setindeki diğer değerlerle karşılaştırıldığında veri setine uygun olmadığı tespit edilen aşırı değerlere, aykırı değer (standart dışı veri) denir. Aykırı

değerlerin fazla olması veri setinin normal dağılımdan sapmasına ve yapılacak istatistiksel analizlerin etkilenmesine sebep olabilir (Aktürk ve Acemoğlu, 2010).

Standart dışı veri, veri setindeki diğer verilerle karşılaştırıldıklarında ya çok büyük ya da çok küçük olduğu görülen verilerdir. Veri analizi çalışmaları yapılırken çok büyük etkiye sahip olabilecekleri gibi hiçbir etkiye sahip olmayabilirler. İstatistiksel analizler için standart dışı verilerin belirlenmesi önemlidir. Standart dışı veriler hatalı olmamaları durumunda bir süreci nasıl iyileştireceğimiz konusunda ipuçları veririler. Standart dışı verileri etkileyen diğer değişkenlerin incelenmesi yaptığımız çalışmalarda maliyet düşürücü veya gelir artırıcı fırsatlar sunabilir (Gürsakal, 2001).

Üzerinde çalışılan veri seti, standart dışı değerlere karşı sürekli kontrol edilmelidir. Çünkü standart dışı değerler veri seti hakkında kullanışlı bilgiler sağlayabilmektedir. Standart dışı veriler, ölçüm veya kaydetme hatalarından dolayı ortaya çıkabilirler. Bir veri setinde bir veya iki adet rahatça fark edilebilen standart dışı değerlerin saptanabilmesi ihtimaline rağmen, kuşku uyandırmayan birçok gözlem olabilmektedir.

Standart dışı veriler genelde 3 tip değişkenlik kaynağından oluşur. Bunlar (Yaycılı, 2006);

1. Verinin doğal yapısı
2. Ölçüm hatası
3. Veri elde etme sürecinde oluşan dolaylı değişkenlik kaynağıdır.

3.5.2. Standart dışı veri analiz yöntemleri

Veri setlerindeki değerlerin aykırı değer olup olmadığını tespit etmek için bazı yöntemler geliştirilmiştir. Bunlar klasik belirleyiciler, sağlam (robust) yöntemler ve veri madenciliği yöntemleridir.

Klasik belirleyiciler, tek bir aykırı değer olması durumunda sorun yaşamazken çok sayıda aykırı değer varlığında aykırı değerlerin tespit edilmesinde zorluklarla karşılaşmaktadırlar. Birden çok aykırı değer durumunda çoklu klasik belirleyicilere gerek duyulmakta ve bu belirleyiciler de karmaşık işlemlere ihtiyaç duymaktadırlar (Vural, 2007). Bu yöntemler büyük verilerde hesaplama karmaşıklığı yüzünden tercih edilmemektedir. Büyük verilerde daha çok uzaklığa ve yoğunluğa dayalı yöntemler kullanılmaktadır.

Kriegel vd. (2010), SIAM (Society For Industrial And Applied Mathematics) uluslararası veri madenciliği konferansındaki sunumlarında standart dışı veri (aykırı değer) analiz yöntemlerini şu şekilde belirtmişlerdir;

1. İstatistiksel testler (Statistical Tests)
2. Derinlik temelli yaklaşımlar (Depth-based Approaches)
3. Sapma temelli yaklaşımlar (Deviation-based Approaches)
4. Mesafeye dayalı yaklaşımlar (Distance based Approaches)
5. Yoğunluğa dayalı yaklaşımlar (Density-based Approaches)
6. Yüksek boyutlu yaklaşımlar (High-dimensional Approaches)

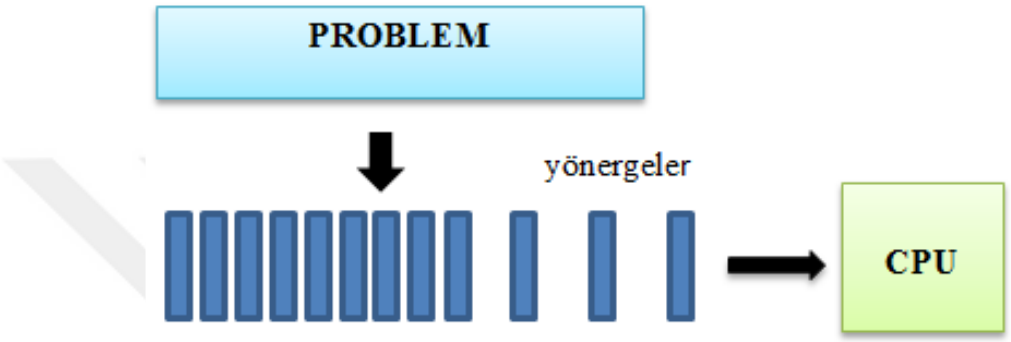
3.6. Paralel Hesaplama

Bilişim sektörünün geldiği nokta ve buna bağlı olarak ihtiyaçların her geçen gün hızla artması birçok problemi de beraberinde getirmektedir. Günümüzde gerek kurumsal, gerekse akademik çalışmalarda kullanılan veri boyutlarının artması son kullanıcılardaki performans olgusunu ilk sıraya yerleştirmiştir. Artık geliştirilen bir sistemin kararlı ve hatasız çalışmasının yanında en kısa sürede en iyi sonucu elde etmek de isteklerin başında gelmektedir. Hız konusu, üretim sektörü, mühendislik hesapları, askeri simülasyon projeleri, hava tahmin hesaplamaları ve veri analizi için hayati öneme sahiptir.

Günümüzde stardart bir hal alan çok çekirdekli işlemciler, tek çekirdekli işlemcilere göre oldukça yüksek hız ve performans getirdiler. Donanım alanındaki bu gelişmelere paralel olarak programların birden fazla işlemci

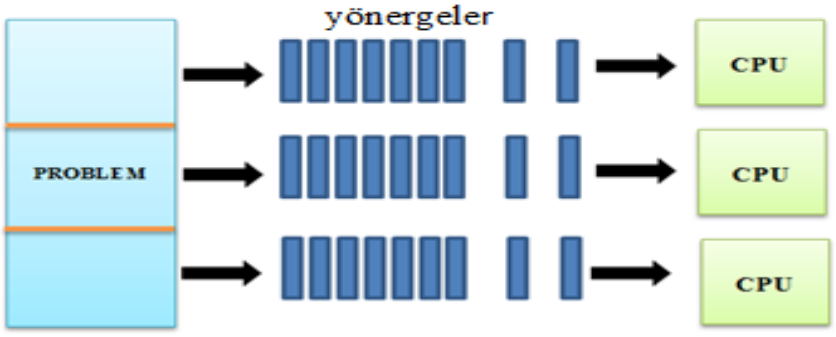
çekirdeği üzerinde çalıştırılması gerektiğinden paralel programlamaya ihtiyaç duyulmuştur (Aktaş, 2010).

Bilgisayar yazılımları daha çok seri hesaplama teknikleri kullanılarak geliştirilmişlerdir. Seri hesaplama yöntemlerinde bir problemi çözmek için seri algoritmalar oluşturulur. Seri algoritmlara göre yazılan komutlar CPU'da sırasıyla yürütülür. Şekil 3.5'de seri hesaplama süreci gösterilmiştir.



Şekil 3.5. Seri hesaplama süreci (Sarıman, 2011)

Paralel hesaplama, bir hesaplama problemini çözmek için birden fazla bilgisayar kaynaklarının aynı anda kullanılmasıdır. Bir işlem çoklu işlemci kullanarak çalıştırılır ya da çoklu çekirdek teknolojisine sahip işlemci üzerinde gerçekleştirilir. Bir problem parçalara ayrılarak aynı zamanda çözülebilir. Her parça farklı işlemci üzerinde aynı anda çalıştırılır (Altıntaş ve Yeğenoğlu, 2011). Şekil 3.6'da paralel hesaplama süreci gösterilmiştir.



Şekil 3.6. Paralel hesaplama süreci (Sarıman, 2011)

Veri analizi, dijitalleşen dünyamızda giderek daha büyük öneme sahip olmaktadır. Bilişim dünyasında verilerin hızlı artışı, veri analizlerinde hız ve performansı önemli hale getirmiştir. Paralel hesaplama büyük verilerin analizinde ve karmaşık hesaplamalarda kullanılmaktadır. Bu yüzden paralel hesaplamanın önemi giderek artmaktadır.

3.6.1. Paralel hesaplama teknikleri

Paralel programlamayı gerçekleştirmek için çeşitli donanım mimarileri ve bu mimarileri destekleyen yazılım modelleri mevcuttur (Kaçka, 2011). Michael J. Flynn 1996 yılında paralel bilgisayar hesapları için ilk sınıflandırmayı yapmıştır. Flynn'a göre bu sınıflandırma 4 ana gruptan oluşur:

- Tek Komut Tek Veri Akışı (SISD)
- Tek Komut Çok Veri Akışı (SIMD)
- Çok Komut Tek Veri Akışı (MISD)
- Çok Komut Çok Veri Akışı (MIMD)

3.6.1.1. Flynn sınıflandırması

SISD (Tek komut, tek veri)

Bu sınıfta, tek işlemci ve tek hafıza bulunmaktadır. Geleneksel tek işlemcili Von Neumann mimarisine sahip paralel bilgisayarlar bu gruba girmektedir. Bu mimariye göre tek bir kontrol ünitesine bağlı tüm işlemciler (çekirdek) aynı komutu paylaşımlı bellek üzerinde yer alan aynı veri setleri için seri olarak işletirler.

SIMD (Tek komut, çok veri)

Bu mimari merkezi bir kontrol ünitesi ve işlemci birimlerinden meydana gelmektedir. Bir saat çevriminde aynı komutlar tüm işlemciler tarafından farklı

veri setleri üzerinde işletilmektedir. Bu mimariye sahip bilgisayarlar ile veri seviyesinde paralelleşme yapılmaktadır.

MISD (Çok komut, tek veri)

MISD paralel bilgisayar mimarisinde, farklı komut setleri farklı işlemciler ile aynı veri seti için işletilmektedir. Bu mimari, karşılaşılan birçok problem için uygun değildir ve bu mimariye sahip çok az makina üretilmektedir.

MIMD (Çok komut, çok veri)

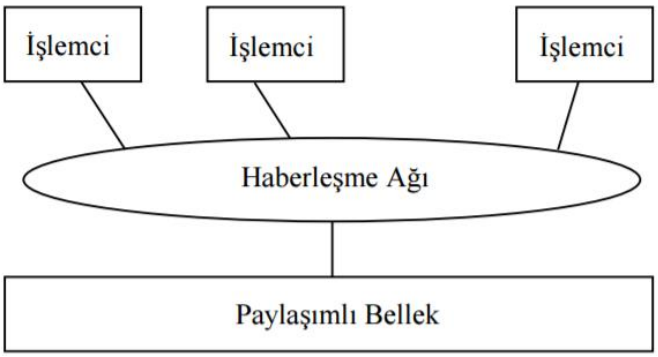
Bu mimaride ise, herhangi bir zaman diliminde farklı işlemciler farklı komutları bir veri setinin farklı parçaları üzerinde işletirler. Paralel bilgisayarlar olarak adlandırılan günümüz süper bilgisayarları bu gruba girmektedir. MIMD mimarisine sahip bilgisayarlar kendi içerisinde bellek bölgesine erişim yöntemlerine bağlı olarak paylaşımlı-bellekli MIMD veya dağıtık-bellekli MIMD olarak sınıflandırılırlar.

3.6.1.2. Paralel bilgisayar bellek mimarileri

Paralel bilgisayar yapıları; paylaşımlı bellek ve dağıtık bellek olmak üzere 2 gruba ayrılır. Problemin büyüklüğü ve hâlihazırdaki donanıma göre uygun mimariler tercih edilmektedir.

Paylaşımlı bellek mimarisi

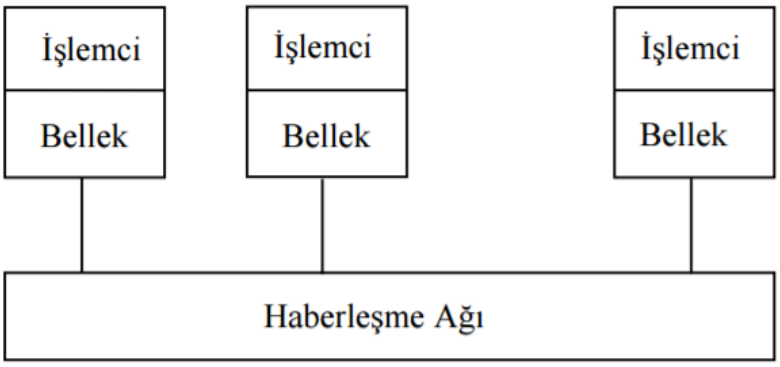
Paylaşımlı bellekli mimariler; çok sayıda işlemcinin aynı belleği paylaşmalarıdır. İşlemciler arasındaki veri iletimi aynı bellek üzerinde olduğu için veri paylaşımı hızlıdır. Şekil 3.7'de paylaşımlı bellekli bilgisayar sisteminin mimarisi görülmektedir.



Şekil 3.7. Paylaşımlı bellekli bilgisayar mimarisi (KTÜ, 2016)

Dağıtık bellek mimarisi

Dağıtık bellekli sistemlerde ise her bir işlemci kendi belleğine sahiptir ve bir işlemci başka bir işlemcinin belleğindeki veriyi mesaj geçme (message passing) ile elde eder**.** Şekil 3.8'de dağıtık bellekli bilgisayar mimarisinin yapısı görülmektedir.



Şekil 3.8. Dağıtık bellekli bilgisayar mimarisi (KTÜ, 2016)

3.6.2. Paralel hızlanma ve performans

Performans: Bir kullanıcı için performans, herhangi bir görevin başlama ve bitiş zamanı arasındaki geçen süredir. Bu süre cevap zamanı ya da çalışma zamanı olarak da bilinir. Performans artışı ise bu sürenin kısalmasına bağlıdır. Bir başka açıdan bakılacak olursa amaç gerçekleşene kadar bir zaman diliminde yapılacak iş sayısının artırılmasıdır. Örneğin bir bilgisayarın işlemcisinin daha hızlı bir işlemciyle değiştirilmesiyle sistemin cevap zamanı azalacak ve aynı zamanda

birim zaman diliminde daha fazla iş yapılacaktır, yani performans artışı sağlanmış olacaktır.

CPU çalışma zamanı: İşlemcide yürütülen bir görevin tamamlanma süresidir. Bu süre giriş/çıkış zamanını ve diğer programları beklemek için geçen süreleri içermez.

Toplam süre: Bir programın yürütülmesiyle geçen toplam süre, CPU çalışma süresi ve bekleme süresinin toplamıdır. Buradaki bekleme süresi program içerisindeki giriş/çıkış işlemleri ve diğer bekleme sürelerinin toplamıdır (Güneş, 2011).

Paralel hesaplamalarla ilgili sistemin bir parçasının hızlandırılması sonucunda, sistemin bir bütün olarak ele alındığında toplam hızlanmanın ne olacağını hesaplamak için Amdahl kanunu kullanılır.

Amdahl Kanunu

Çok işlemcili ortamlarda, paralel çalışma sonucunda elde edilebilecek azami kazancı tahmin etmek için kullanılır. Gene Amdahl tarafından geliştirilen bu kurala göre paralel çalışma sonucunda zaman kazanımı formüllenmiştir.

Amdahl kanuna göre hızlanma;

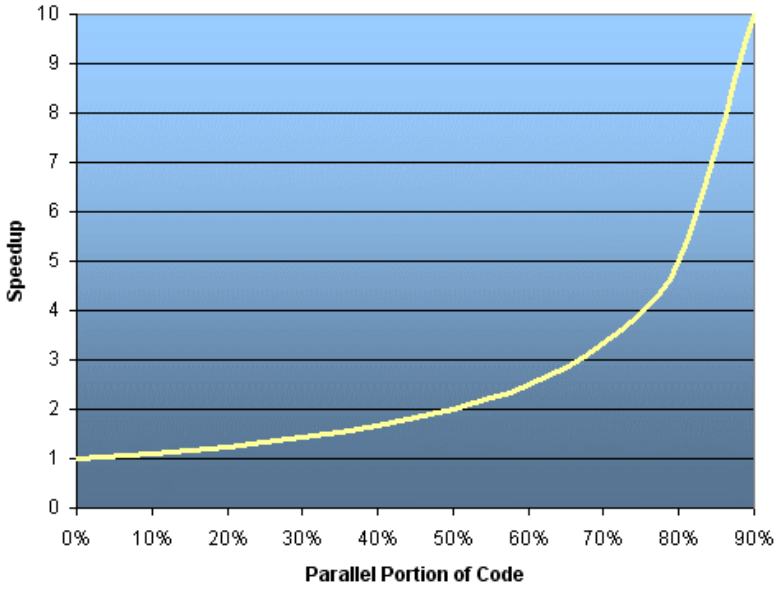
$$\text{Hızlanma} = \frac{1}{(1-P) + \left(\frac{P}{N}\right)} \tag{3.1}$$

Denklem 3.1'de; P, Kodun paralelleştirme yüzdesi ve N ise işlemci sayısıdır.

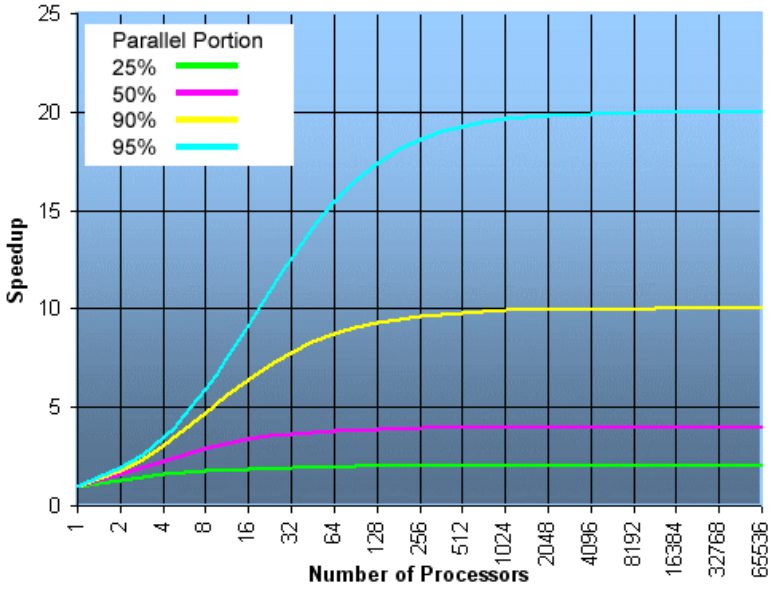
Denklem 3.1'e göre seri kodun hiçbir bölgesi paralelleşmiyorsa yani P=0 ise hızlanma 1'dir. Yani hızlanma yoktur. Eğer seri kodun tamamı paralelleşebiliyorsa yani P=1 ise hızlanma teorik olarak sonsuzdur.

Örnek bir senaryoya göre, %75'i paralelleştirlmiş uygulama 3 çekirdekli bir bilgisayarda çalıştırılırsa ve hızlanma formülü uygulanırsa; hızlanma 2 olacaktır ve uygulamamız 2 kat daha hızlı çalışacaktır.

Amdahl kanunu daha iyi anlamak için Şekil 3.9'da kodun paralellik oranı ile hızlanma arasındaki ilişki ve Şekil 3.10'da işlemci sayısı ile hızlanma arasındaki ilişki gösterilmiştir.



Şekil 3.9. Kodun paralellik oranı ile hızlanma arasındaki ilişki (Barney, 2016)

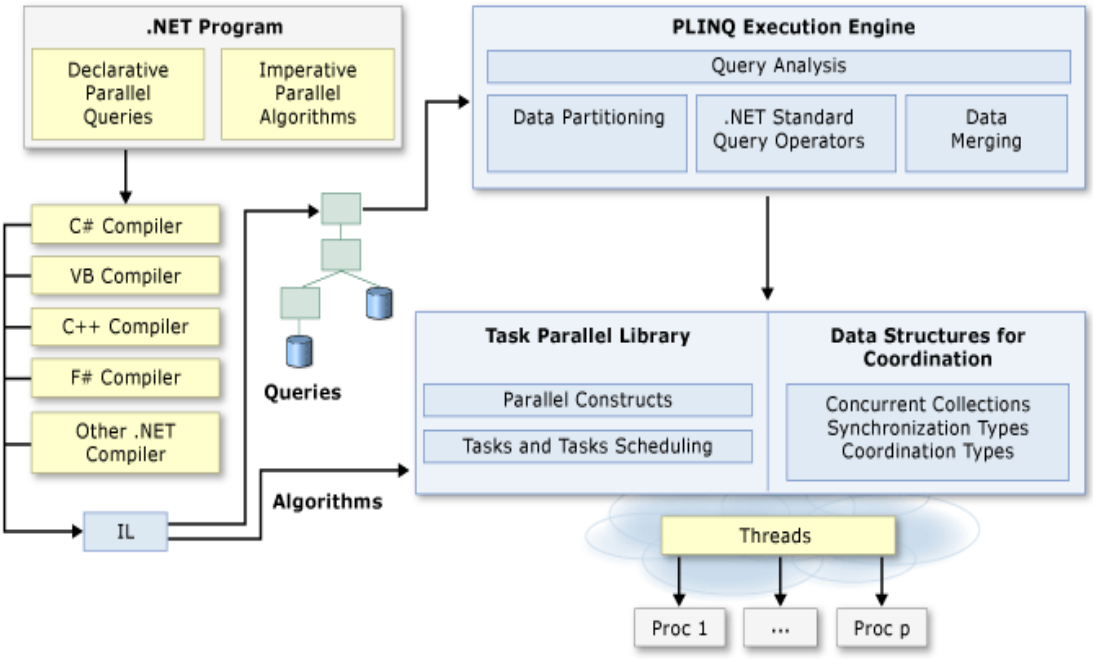


Şekil 3.10. İşlemci sayısı ile hızlanma arasındaki ilişki (Barney, 2016)

3.6.3. Microsoft .Net ile paralel programlama

Günümüzde satın alınan bilgisayarlar üzerinde birden fazla çekirdek olmasından dolayı işlemciler üzerinde aynı anda birden fazla kanal çalıştırma imkânımız olmaktadır. Günümüzde kullanılan seri programlama yapısıyla çekirdekler etkin bir şekilde kullanılamamaktadır ve geliştirilen programlar tek bir kanal üzerinde çalıştığı için daha yavaş çalışabilir. Bu sebeplerden dolayı Microsoft programlama alt yapısına yeni bir sürüm kazandırarak paralel programlamayı alt yapısına kazandırmıştır. Microsoft .Net Framework 4.0 versiyonu ile uygulamalar eş zamanlı olarak birden fazla çekirdek üzerinde çok kanallı yöntemler ile daha hızlı ve verimli bir şekilde çalıştırılabilmektedir. .Net framework ile paralel programlama kütüphanesi içerisinde çekirdek yenilikleri, task parallel library (TPL) ve parallel linq bulunmaktadır (Sarıman, 2011b).

Şekil 3.11'de .Net framework 4.0 mimarisi görülmektedir.



Şekil 3.11. .Net framework mimarisi (Microsft, 2016)

.Net framework ile gelen API'ler (Application Programming Interface) şu şekilde sıralanabilir.

- Paralel Linq (PLINQ)
- Paralel Sınıf (Parallel Class)
- Görev Paralellik Yapısı (Task Parallelism Constructs)
- Eşzamanlı Koleksiyonlar (Concurrent Collections)
- SpinLock ve SpinWait (SpinLock and SpinWait)

Bu API'ler topluca PFX (Parallel Framework) olarak bilinir. Paralel sınıf ve görev paralellik yapıları ise; birlikte TPL (Task Parallel Library) olarak adlandırılır (Albahari, 2011).

3.6.3.1. Task parallel library

TPL in en büyük amacı, eş zamanlı veya paralel olarak yürütülmek istenen işlemlerin, daha kolay ve basit bir şekilde ele alınmasını sağlamaktır. Bu anlamda günümüz işlemcilerinin çekirdek sayısı veya sistemlerdeki işlemci sayısının birden fazla olması durumunda, TPL verimli sonuçlar elde edilmesini sağlamaktadır. Bu açıdan bakıldığında TPL alt yapısına tüm sistem çekirdek gücünü verme imkânına sahip olduğu belirtilebilir. TPL kullanımı ile ilişkili olarak önemli bir nokta ise, işlemlerin Multi-Threading mantığına göre yapılıyor olmasıdır. Dolayısıyla, programın çalışma zamanı yükünü arttırıcı bir etkendir. Bir başka deyişle her işlemin, elimizde TPL var diye paralel olarak yürütülmeye çalışılması doğru değildir. Bazı süreçlerin gerçekten ve bilinçli olaraktan ardışık (sequential) yürümesi gerekebilir. TPL, .Net Framework 4.0 ile birlikte gelen ve paralel işlem yapma yeteneklerini ele alan kütüphanedir (Sarıman, 2011b).

TPL altyapısı süreçlere, görevler olarak bakar ve tıpkı insanların görevleri yürütebildiği mantıkla organize edebilmektedir.

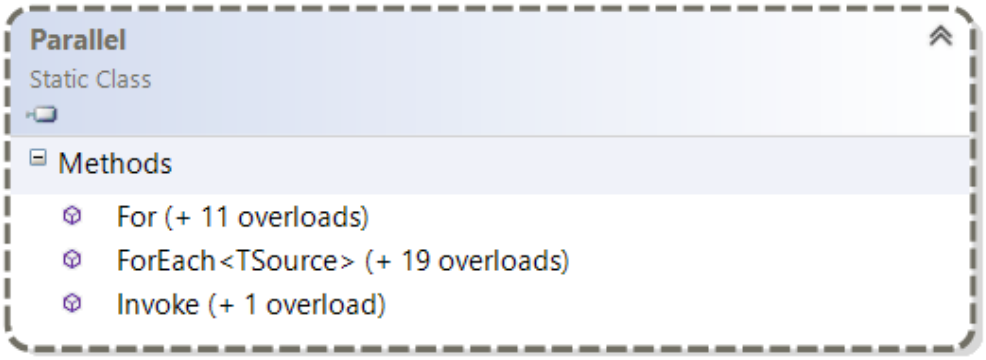
- Yeni görevler oluşturmak, bu görevleri başlatmak, duraklatmak ve sonlandırmak mümkündür.

- Bir görevin bittiği yerden başka bir görevi başlatmak mümkündür.
- Başarıyla yerine getirilen görevlerin sonucunda değerler döndürmek mümkündür.
- Bir görev kendi içinde alt görevler başlatabilir.
- Görevler aynı veya farklı thread'ler tarafından yerine getirilebilirler.

Task (görev), başarılı bir şeklide tamamlanmasını istediğimiz bir süreçtir. Süreç T1 zamanında başlar ve T2 zamanında sonlanır. Bu sürecin tamamlanma süresi, işlemcimizin özelliklerine ve kod yazım biçimlerine göre değişebilir.

Thread(ler) ise görevleri yerine getiren işçilerdir. Bu işçiler bir veya birden fazla olabilir. Her görev ayrı bir thread tarafından yürütülmek zorunda değildir. Bir thread birden fazla görevi yerine getirebilir. Ya da bir görev birden fazla thread (multithread) yardımıyla yapılabilmektedir (Üçüncü, 2012).

TPL; paralel sınıfın üyesi olan for, foreach döngülerinin paralel versiyonlarını ve Invoke metodunu içermektedir. Şekil 3.12'de paralel sınıfın üyeleri görülmektedir.

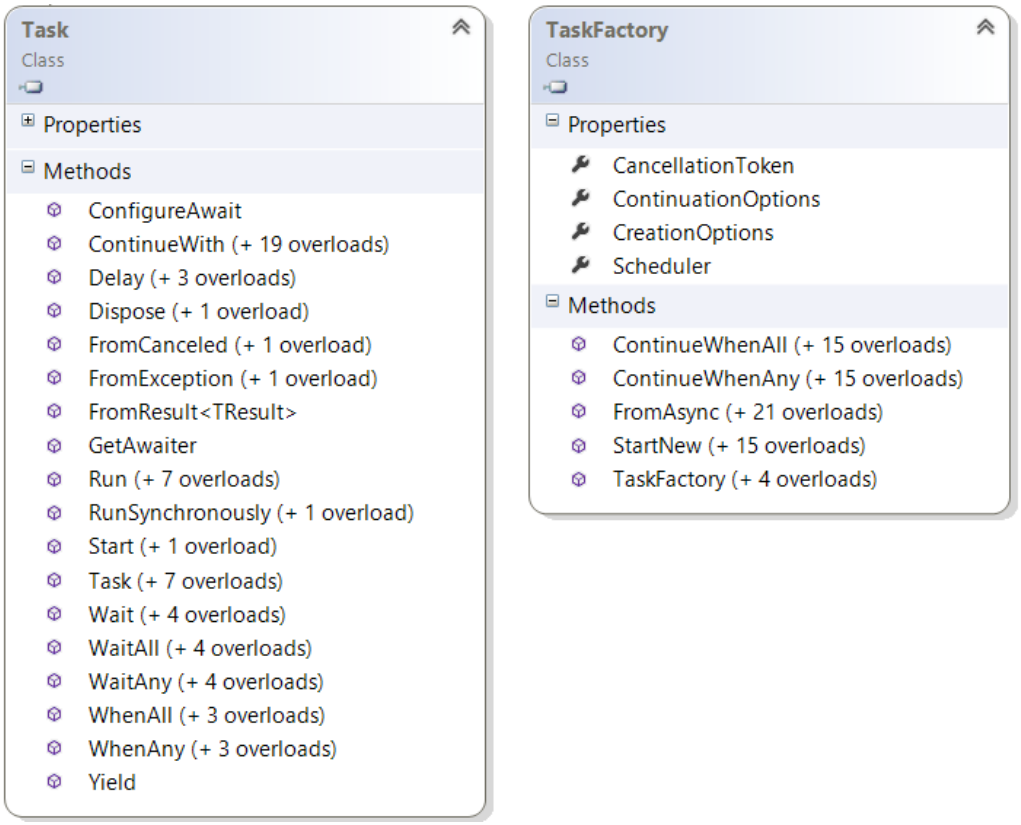


Şekil 3.12. Paralel statik sınıf üyeleri

Paralel döngüler görevleri bizim yerinize oluşturur ve yönetir. Parallel for ve foreach metodları dizi veya koleksiyon yapıları üzerinde döngüsel işlemlerin paralel olarak yürütülmesini sağlamaktadır. Invoke metodu ise sunduğu Action temsilcisi (delegate) yardımıyla, birden fazla metodun aynı anda paralel olarak çalıştırılabilmesine olanak sağlamaktadır. For veya ForEach gibi Paralel sınıfına

ait üyeleri kullandığımız hallerde de, arka planda Task sınıfı ve üyeleri gizlice devreye girerler.

TPL, paralel çalışacak olan görevlerin başlatılması, iptal edilmesi, bekletilmesi, arka arkaya eklenerek bir süreç tesis edilmesi gibi pek çok işlemi yapabilen Task ve TaskFactory sınıfını da içermektedir. Şekil 3.13'de Task ve TaskFactory sınıfının üyeleri görülmektedir.



Şekil 3.13. Task ve TaskFactory sınıfının üyeleri

Task sınıfı normal şartlarda geriye değer döndürmeyen fonksiyonelliklerin eş zamanlı olarak çalıştırılmasında ele alınmaktadır. Geriye değer döndüren metodlar söz konusu olduğunda ise, Task<T> generic tipinden yararlanılabilir. Buradaki T, paralel çalışan metodun dönüş tipi olarak düşünülebilir. Aslında Task ve Task<T> sınıflarının static Factory özelliği üzerinden gidildiğinde StartNew metodu yardımıyla görevlerin başlatılması sağlanmaktadır. Diğer yandan Task<T> sınıfının Result özelliği, geri dönüş tipini belirtmektedir (Şenyurt, 2009).

Paralel for ve foreach metotlarının kullanımı

Parallel.For metodunun kullanımı şu şekildedir;

Parallel.For (int fromInclusive, int toExclusive, Action<int> body);
fromInclusive: for döngüsünün başlangıç değeri.
toExclusive: for döngüsünün bitiş değeri.
Body: for döngüsünün içerisinde yapılacaklar.

Örnek bir paralel for döngüsü ise şu şekildedir:

```
Parallel.For(0,10, p=>
{
//yapılacak işlemler
});
```

Parallel.Foreach metodunun kullanımı ise şu şekildedir;

Parallel.Foreach (IEnumerable<TSource> source, Action<TSource> body)
source: Foreach işlemi boyunca içerisinde dolaşacağımız IEnumerable arayüzünü implemente etmiş olan koleksiyon.
body: Foreach döngüsü içerisinde ele alacağımız işlemler.

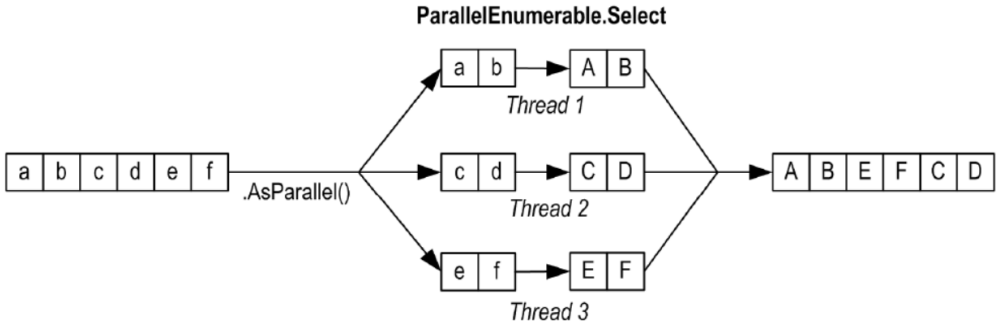
Örnek bir paralel foreach döngüsü ise şu şekildedir:

```
IEnumerable<int> numbers = Enumerable.Range(1,1000);
Parallel.ForEach(numbers, i =>
{
//numbers hangi tipi taşıyan koleksiyon ise
//i parametresi de o tipte
//işlemler
});
```

3.6.3.2. Parallel linq (PLINQ)

PLINQ, Microsoft Research ve CLR takımları tarafından ortaklaşa geliştirilen Parallel Extensions isimli genişletmelerin bir paçasıdır. Diğer parça ise daha önce bahsedilen TPL'dir. Her iki yapının kullanım amacı, yönetimli kod (managed code) tarafındaki eş zamanlı işleyişlerin kolay bir şekilde sağlanmasıdır (Sarıman, 2011b).

PLINQ, LINQ sorgularının kendi içerisinde parçalanarak farklı threadlerde çalışması ve bu parçların paralel yürütülerek sonuçların elde edilmesidir. Şekil 3.14'de PLINQ sorgusunun örnek bir gösterimi vardır.




```
"abcdef".AsParallel().Select (c => char.ToUpper(c)).ToArray()
```

Şekil 3.14. Örnek PLINQ sorgusu (Albahari, 2011)

PLINQ ifadeleri çift çekirdek ve üstü işlemcilerin ya da birden fazla işlemcinin olduğu sistemlerde daha anlamlıdır. PLINQ motoru, çalışmakta olan sorgu sürecini, makinenin sahip olduğu çekirdek sayısına göre parçalara ayırır ve yürütür. PLINQ' in avantajı gerçekten çok uzun sürebilecek sorgular söz konusu olduğunda ortaya çıkmatadır. Bu nedenle her LINQ sorgusunun PLINQ formatına dönüştürülmesinin de anlamlı olmadığını söyleyebiliriz (Şenyurt, 2009).

4. ARAŞTIRMA BULGULARI

Bu bölümde analiz için kullanılan veri setlerinin özellikleri, geliştirilen uygulamanın özellikleri tanıtılmıştır. DBSCAN algortiması için uygun eps ve minPts değerleri tespit edilmiştir. Seri ve paralel uygulamaların sonuçları karşılaştırılmıştır.

4.1. Veri Seti ve Uygulama Platformu

Bu çalışmada 4 adet veri seti kullanılmıştır. Analizi yapılan veri setlerinin özellikleri Çizelge 4.1'de gösterilmiştir.

Çizelge 4.1 Analizi yapılan veri setleri

Veri Seti Adı	Uzunluğu	Temin Edildiği Yer
maxSicaklikOcak	31	Meteoroloji Genel Müdürlüğü
maxSicaklik2015	331	Meteoroloji Genel Müdürlüğü
real_2	1439	Yahoo Webscope
dfkiartificial300	3000	German Research Center for Artificial Intelligence

maxSicaklikOcak veri seti; Meteoroloji Genel Müdürlüğü TÜMAŞ veri arşiv sisteminden alınmıştır. Isparta ilinin 2015 yılı Ocak ayına ait maksimum sıcaklık değerlerini içermektedir. Şekil 4.1'de veri setinden örnek parça gösterilmektedir.

Gun	MaxSicaklik
1	7,7
2	5,7
3	6,7
.	.
.	.
.	.
29	6,8
30	6,6
31	9,3

Şekil 4.1. maxSicaklikOcak veri seti örnek parçası

maxSicaklik2015 veri seti ise; Meteoroloji Genel Müdürlüğü TÜMAŞ veri arşiv sisteminden alınmıştır. Isparta ilinin 2015 yılına ait maksimum sıcaklık değerlerini içermektedir. Şekil 4.2'de veri setinden örnek parça gösterilmektedir.

Gun	MaxSicaklik
1	7,7
2	5,7
3	6,7
.	.
.	.
.	.
18	-2,2
19	-0,8
20	3,9

Şekil 4.2. maxSicaklik2015 veri seti örnek parçası

real_2 veri seti, Yahoo Webscope kütüphanesinden temin edilmiştir. Veri seti özel olarak standart dışı değerler içermektedir. Veri setindeki değerler Yahoo servislerinden elde edilen değerlerden oluşturulmuştur. Veri seti, S5 - A Labeled Anomaly Detection Dataset, version 1.0(16M) olarak isimlendirilmiştir (Yahoo, 2016). Şekil 4.3'de veri setinden örnek parça görülmektedir.

timestamp	value	is_anomaly
1	12183	0
2	12715	0
3	12736	0
4	12716	0
5	12739	0
.	.	.
.	.	.
.	.	.
1433	117800	1
1434	153159	1
1435	108454	1

Şekil 4.3. real_2 örnek veri seti parçası

Şekil 4.3'de görüldüğü gibi veri seti zaman serisi, değer ve değerlerin standart dışı değer olup olmadığını gösteren 3 sütundan oluşmaktadır. is_anomaly sütunu 0 ise değer normal, 1 ise değer standart dışı olarak kabul edilmiştir.

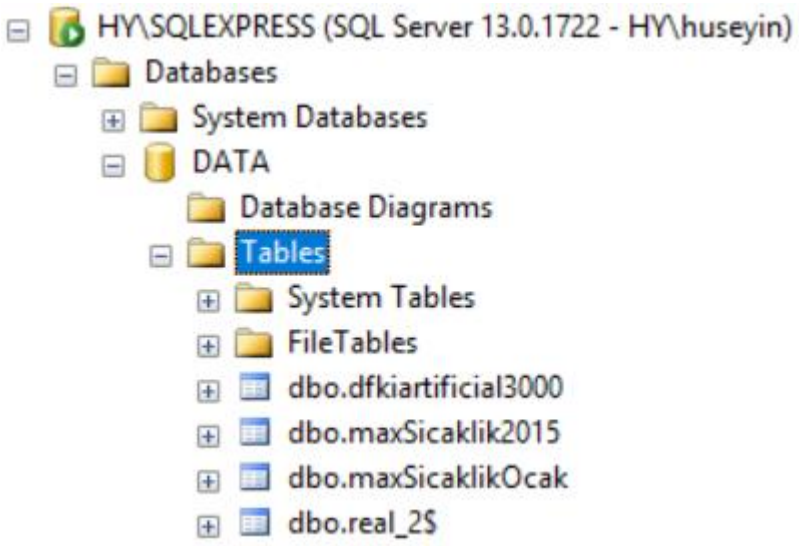
Çalışmada kullanılan diğer veri seti ise; Alman Yapay Zeka ve Araştırma Merkezinin web sitesinden temin edilmiştir. Veri seti, dfki-artificial-3000-unsupervised-ad olarak isimlendirilmiştir ve 3000 satır uzunluğundadır (German Research Center for Artificial Intelligence, 2016). Veri seti özel olarak standart dışı değerler içermektedir. Veri setindeki değerler yapay olarak üretilmiş olup, şekil 4.4.'de örnek veri seti parçası görülmektedir.

attribute_1	attribute_2	outlier
-9,798576621135520	-14,403254553422400	1
-10,605576721835800	-14,356257577774200	1
-9,958576690135520	-14,101244555626400	1
-9,515670555455520	-13,903234555555500	1
-8,798110623135520	-14,451254553422600	1
.	.	.
.	.	.
.	.	.
-3,791569025149460	4,669451951927900	0
5,945872458229040	-7,148171930690370	0
5,013359581217500	-7,924428553746300	0
3,444621469062470	-7,229945398561400	0

Şekil 4.4. dfkiartificial3000 örnek veri seti parçası

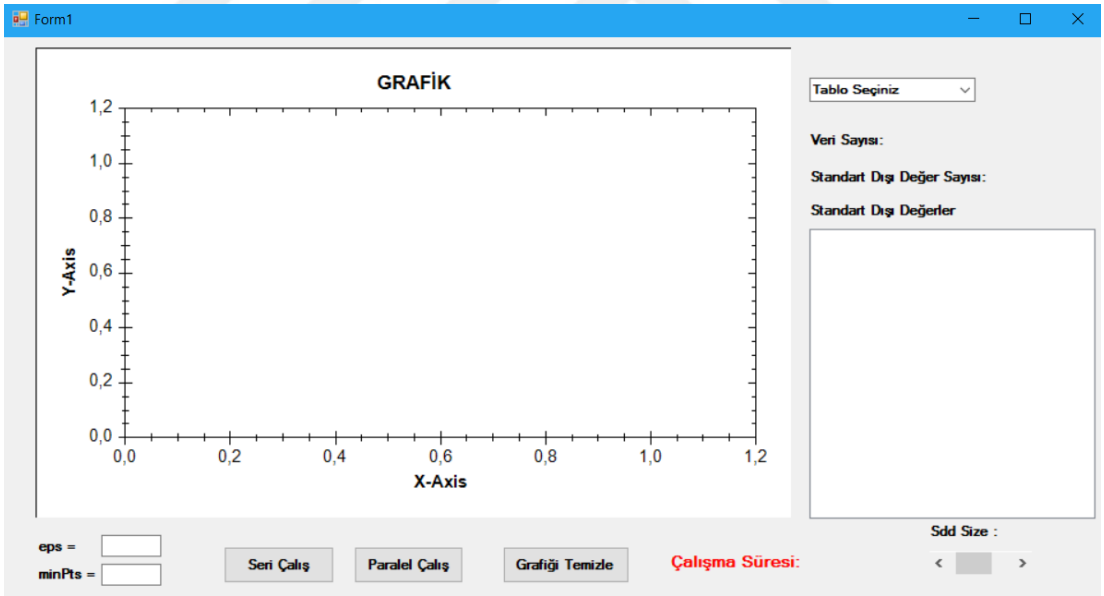
Gerçekleştirilen uygulama MS Windows 10 işletim sistemi üzerinde, Intel(R) Core (TM) i7 – 3630QM CPU 2.40 GHZ işlemci ve 16 GB bellek donanımına sahip bir bilgisayarda çalıştırılmıştır. Hazırlanan programın seri ve paralel kodları C# programlama dilinde yazılmıştır. Uygulama Visual Studio 2015 editörü kullanılarak geliştirilmiştir.

Veriler MSSQL veri tabanına aktarılmıştır. MSSQL üzerinde DATA adında bir veri tabanı oluşturularak, veri setleri eklenmiştir. Şekil 4.5'de veri tabanının görüntüsü gösterilmektedir.



Şekil 4.5. DATA veri tabanı

Uygulama C# dilinde Windows Form Application olarak geliştirilmiş ve kullanıcı arayüzü şekil 4.6'da gösterilmiştir.



Şekil 4.6. Uygulamanın kullanıcı arayüzü

Uygulamanın çalışması ve form üzerindeki nesnelerin görevleri şunlardır;

Tablo Seçiniz: Analizi yapılacak veri seti MSSQL veri tabanından çekilerek comboBox'ta gösterilmiştir.

Veri Sayısı: Seçilen veri setindeki satır sayısı gösterilmiştir.

Standart Dışı Değer Sayısı: Uygulama tarafından standart dışı veri olarak tespit edilen verilerin sayısı gösterilmiştir.

Standart Dışı Değerler: Tespit edilen standart dışı değerler listBox'a eklenmiştir.

Seri Çalış: Uygulamanın seri kodlarla çalışması sağlanmıştır.

Paralel Çalış: Uygulamanın paralel kodlarla çalışması sağlanmıştır.

eps ve minPts: Seçilen veri setine göre optimum eps ve minPts değerleri gösterilmiştir.

Grafik Alanı: Analizi yapılan veriler grafik alanında gösterilmiştir. Standart dışı değerler ise kırmızı renkte olup, diğer noklardan daha büyük ve farklı bir şekile sahip olarak işaretlenmiştir. Grafiği çizdirmek için ZedGraph eklentisi kullanılmıştır.

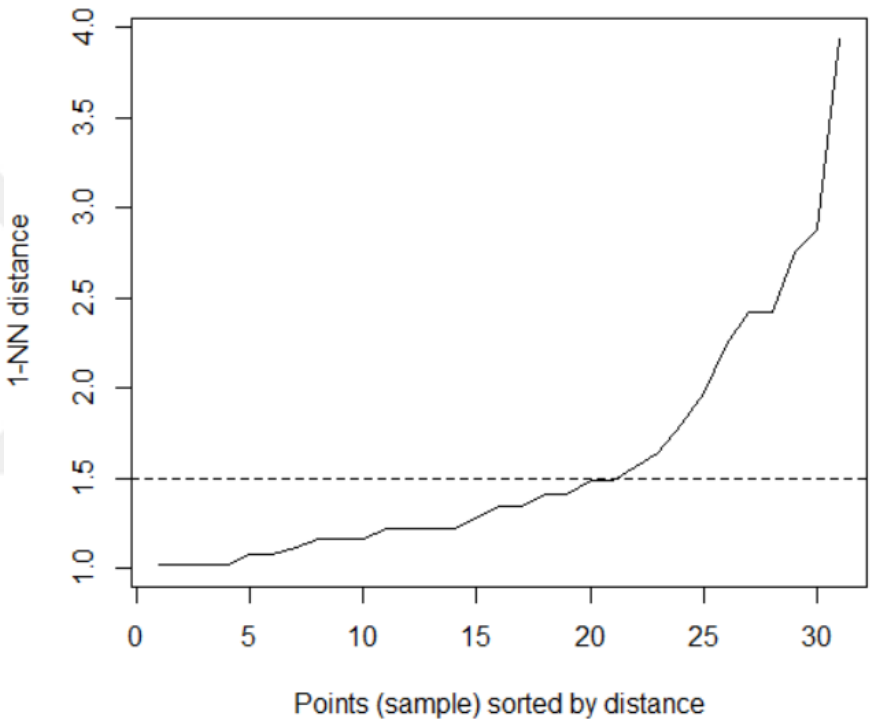
Sdd (Standart Dışı Değer) Size: Standart dışı değerlerin grafik alanındaki büyüklüğünü ayarlamak için kullanılmıştır.

Çalışma Süresi: Freeman (2010), Stopwatch sınıfının performans ölçmek için doğru bir yöntem olduğundan bahsetmiştir. Bu çalışma kapsamında yapılan uygulamada bu sınıf kullanılarak seri ve paralel metotların çalışma zamanları ölçülmüş ve milisaniye cinsinden uygulama üzerinde gösterilmiştir.

4.2. Veri Setleri İçin Optimum eps ve minPts Değerlerinin Bulunması

Her veri seti için uygun eps ve minPts değerlerini hesaplamak için daha önceki bölümlerde bahsedilen k'nıncı en yakın komşu grafiğini çizdirmek için R Project isimli istatistiksel modelleme ve geliştirme aracı kullanılmıştır.

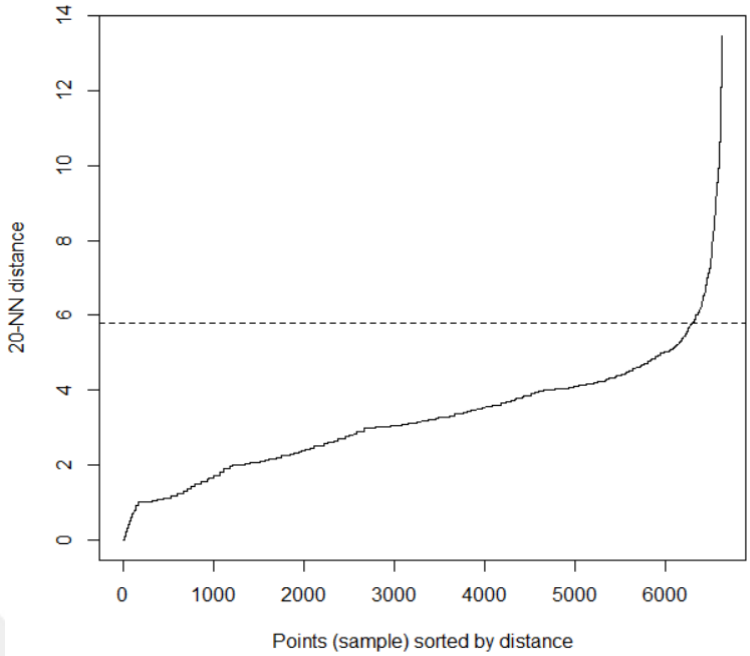
Şekil 4.7'de maxSicaklikOcak veri seti için çizdirilen knn-dist grafiği gösterilmiştir.



Şekil 4.7. maxSicaklikOcak k-dist grafiği

Grafikteki keskin değişim yaklaşık olarak 1,5 değerinde olduğundan eps değeri olarak alınmış, minPts değeri ise k'nın değerine eşit olarak alınmıştır (Şekil 4.7).

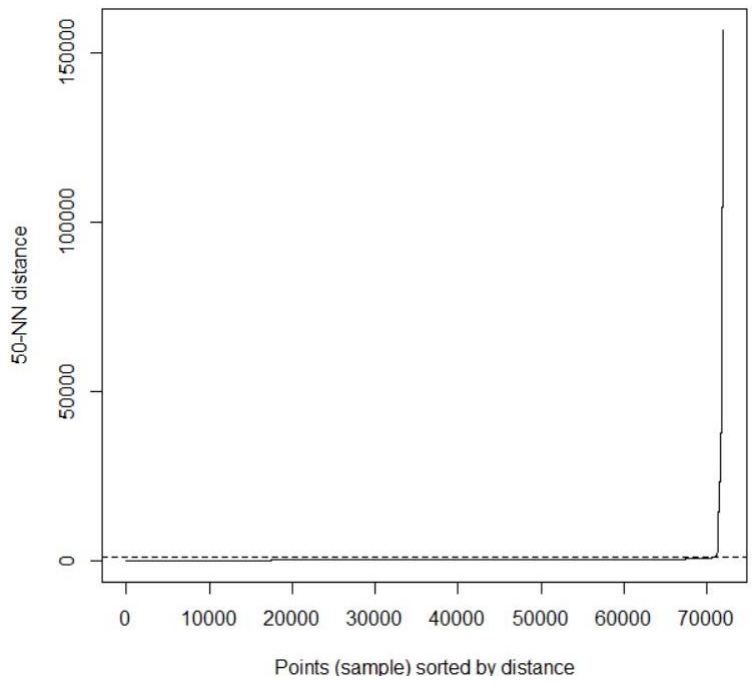
Şekil 4.8'de maxSicaklik2015 veri seti için çizdirilen knn-dist grafiği gösterilmiştir.



Şekil 4.8. maxSicaklik2015 k-dist grafiği

Grafikteki keskin değişim yaklaşık olarak 5,8 değerinde olduğundan eps değeri olarak alınmış, minPts değeri ise k'nın değerine eşit olarak alınmıştır (Şekil 4.8).

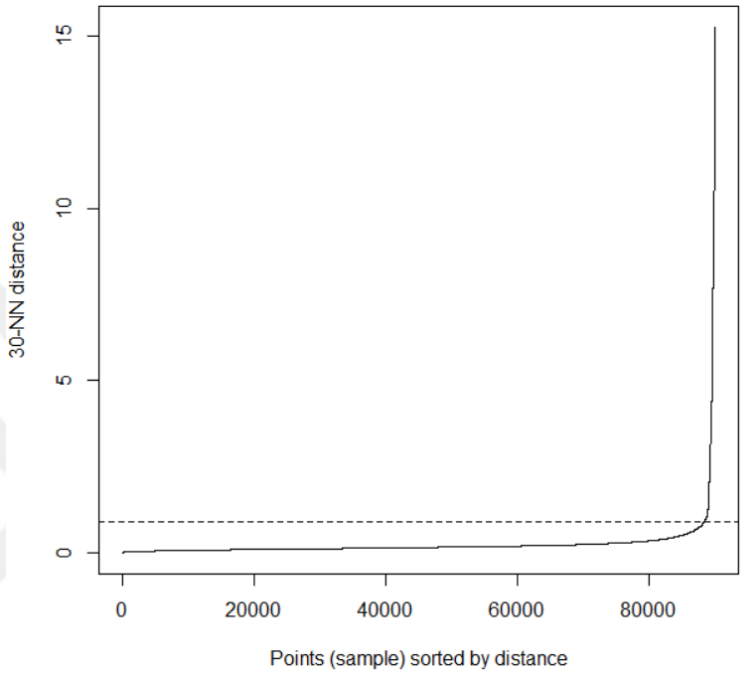
Şekil 4.9'da real_2 veri seti için çizdirilen knn-dist grafiği gösterilmiştir.



Şekil 4.9. real_2 k-dist grafiği

Grafikteki keskin değişim yaklaşık olarak 500 değerinde olduğundan eps değeri olarak alınmış, minPts değeri ise k'nın değerine eşit olarak alınmıştır (Şekil 4.9).

Şekil 4.10'da dfkiartificial3000 veri seti için çizdirilen knn-dist grafiği gösterilmiştir.



Şekil 4.10. dfkiartificial3000 veri seti için k-dist grafiği

Grafikteki keskin değişim yaklaşık olarak 0,9 değerinde olduğundan eps değeri olarak alınmış, minPts değeri ise k'nın değerine eşit olarak alınmıştır(Şekil 4.10).

Çizelge 4.2'de veri setleri için kullanılan optimum eps ve minPts değerleri gösterilmiştir.

Çizelge 4.2. eps ve minPts değerleri

Veri Seti	Eps Değeri	minPts Değeri
maxSicaklikOcak	1,5	1
maxSicaklik2015	5	20
real_2	1000	50
dfkiartificial300	0,9	30

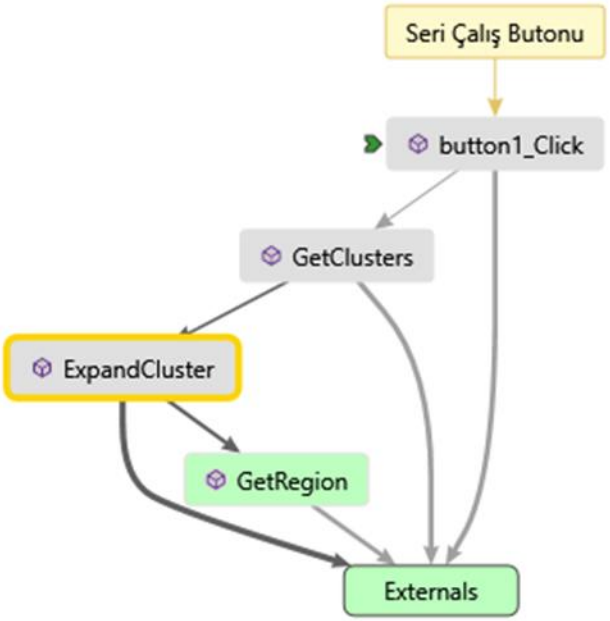
4.3. Seri DBSCAN Uygulaması

Gerçekleştirilen uygulama 3 adet metot içermektedir. Çizelge 4.3'de metot isimleri ve gerçekleştirdiği işlem gösterilmiştir.

Çizelge 4.3. Metotlar ve yaptığı işler

Metot İsmi	Yaptığı İş
GetClusters	Kümeleri içeren dizileri bulur.
ExpandCluster	Kümeleri genişletir.
GetRegion	Çekirdek noktaları bulur.

Şekil 4.11'de ise seri uygulamanın Visual Studio editörü ile oluşturulan kod haritası (code map) gösterilmiştir.



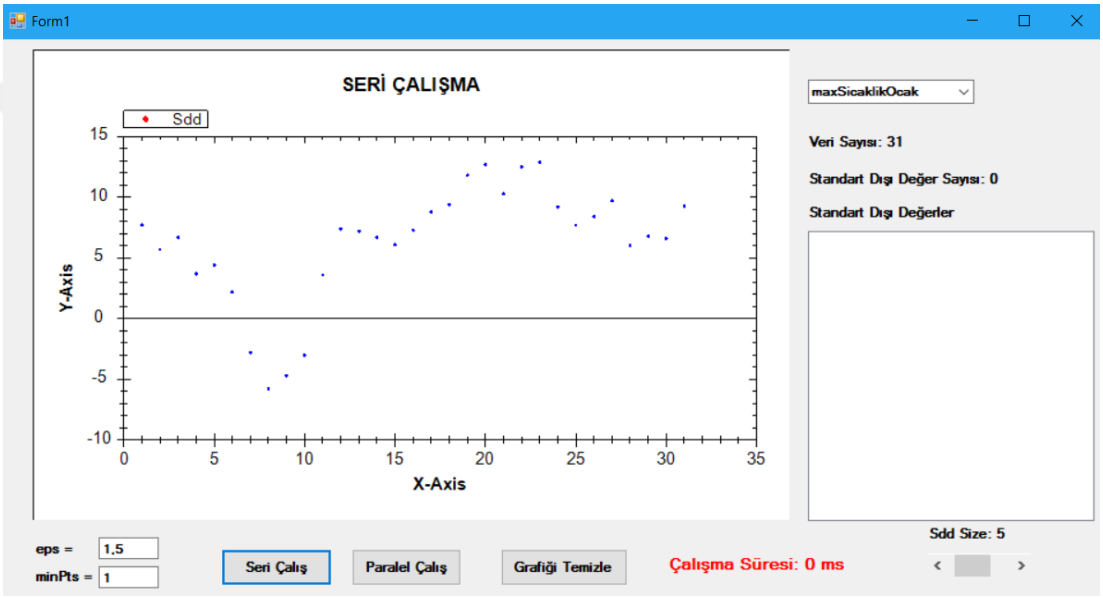
Şekil 4.11. Seri uygulamanın kod haritası

Çizelge 4.4'de gösterilen 4 adet veri seti analiz edilmiş ve analiz sonuçları gösterilmiştir. Uygulamaya dışardan daha önce tespit edilen eps ve minPts parametreleri girilmiştir. Bu değerler kullanıcı tarafından değiştirilebilmektedir.

Çizelge 4.4. Seri uygulamanın sonuçları

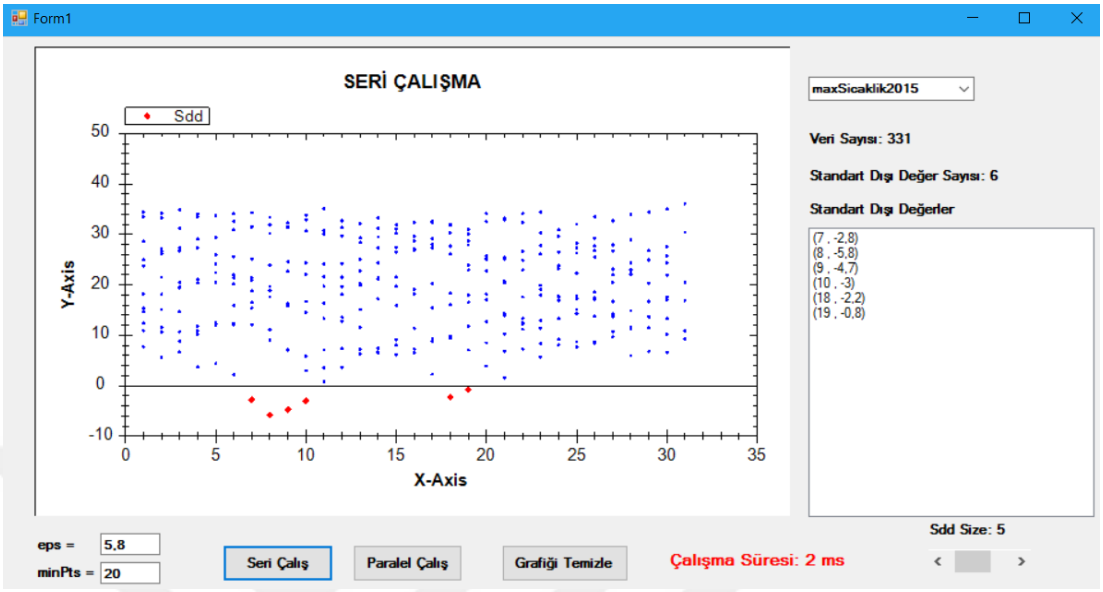
Veri Seti	Tablo Uzunluğu	Standart Dışı Değer sayısı	Çalışma Süresi (ms)
maxSicaklikOcak	31	0	0
maxSicaklik2015	331	6	2
real_2	1439	15	54
dfkiartificial300	3000	37	232

Şekil 4.12'de veri uzunluğu 31 olan maxSicaklikOcak veri setinin analiz sonuçları gösterilmiştir.



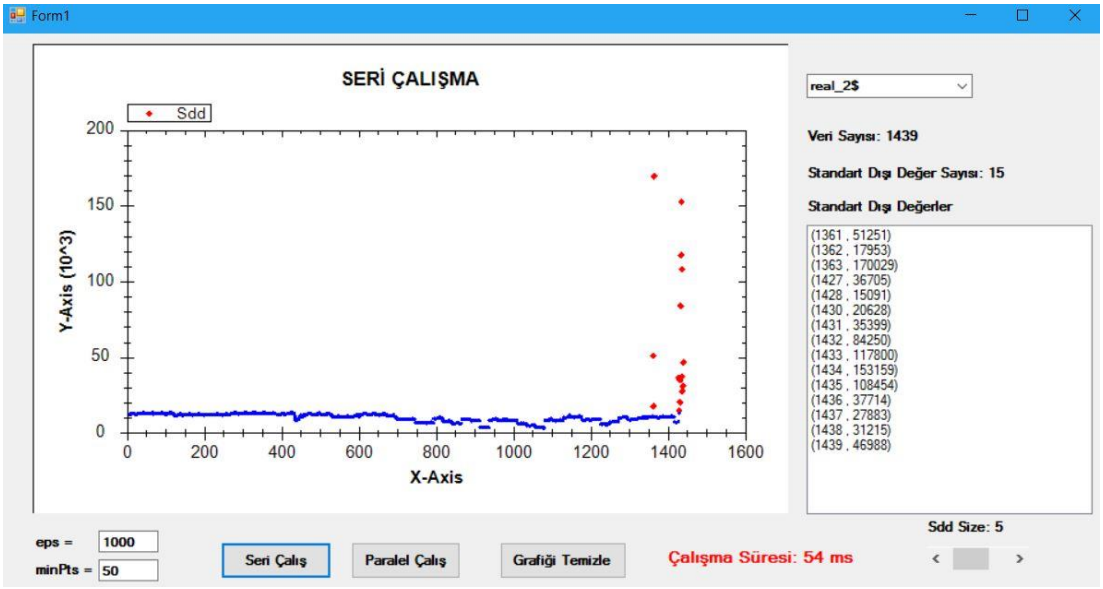
Şekil 4.12. maxSicaklikOcak veri seti analizi

Şekil 4.13'de veri uzunluğu 331 olan maxSicaklik2015 veri setinin analiz sonuçları gösterilmiştir.



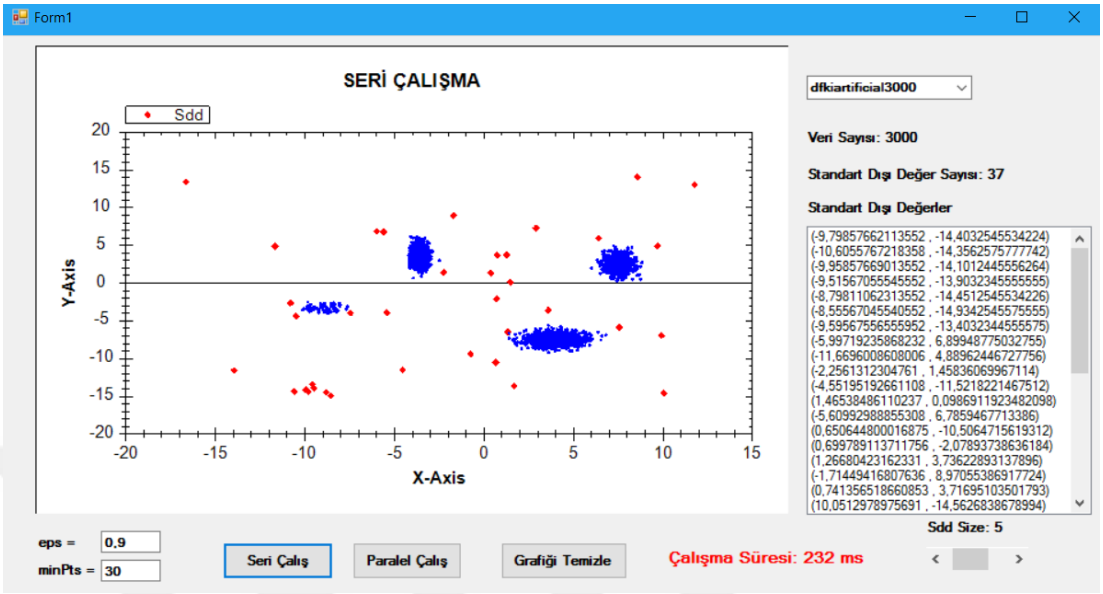
Şekil 4.13. maxSicaklik2015 veri seti analizi

Şekil 4.14'de veri uzunluğu 1439 olan real_2 veri setinin analiz sonuçları gösterilmiştir.



Şekil 4.14. real_2 veri seti analizi

Şekil 4.15'de veri uzunluğu 3000 olan dfkiartificial300 veri setinin analiz sonuçları gösterilmiştir.



Şekil 4.15. dfkiartificial300 veri seti analizi

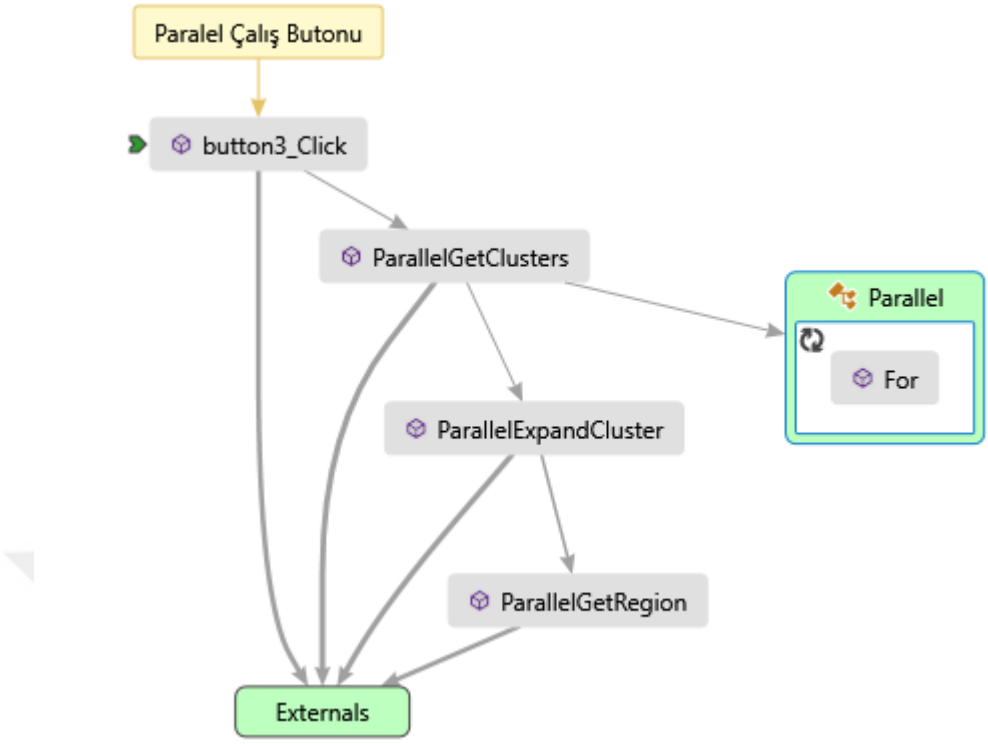
4.4. Paralel DBSCAN Uygulaması ve Sonuçları

Seri uygulamada kullanılan for döngüleri yerine Parallel For döngüsü kullanılarak performans artışı sağlanmaya çalışılmıştır. Çizelge 4.5'de paralel uygulamanın sonuçları gösterilmektedir.

Çizelge 4.5. Paralel uygulama sonuçları

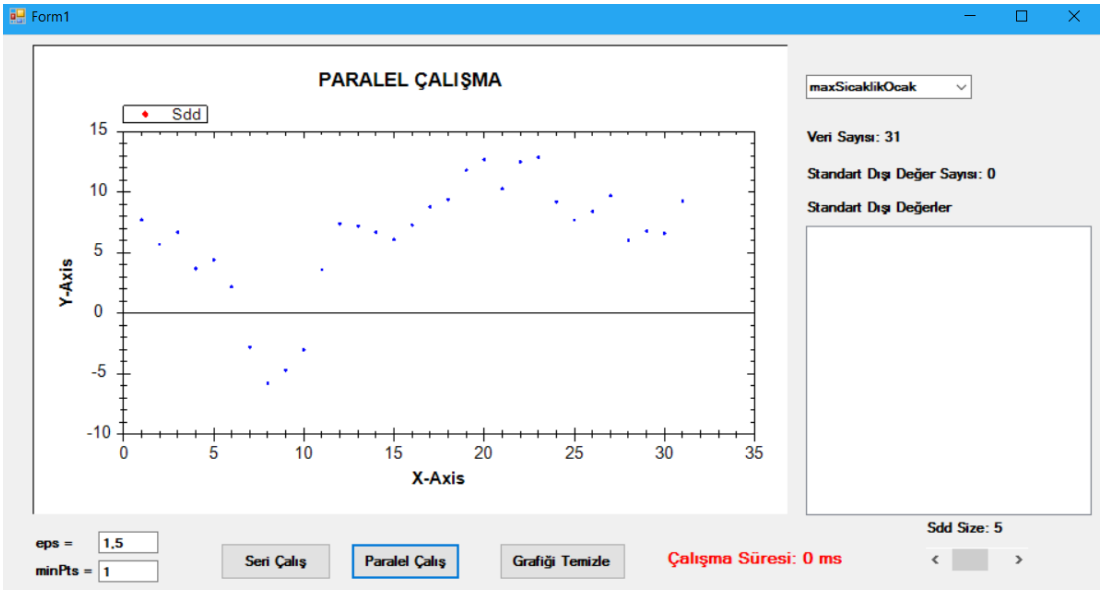
Tablo Adı	Tablo Uzunluğu	Standart Dışı Değer sayısı	Çalışma Süresi (ms)
maxSicaklikOcak	31	0	0
maxSicaklik2015	331	6	2
real_2	1439	15	39
dfkiartificial300	3000	37	108

Şekil 4.16'da ise paralel uygulamanın Visual Studio editörü ile oluşturulan kod haritası (code map) gösterilmiştir ve Parallel For döngüsü ParallelGetCluster metodunun içerisinde kullanılmıştır. Parallel For döngüsü, otomatik oluşturulan threadler sayesinde eş zamanlı olarak veri setindeki noktaları işleme almıştır.



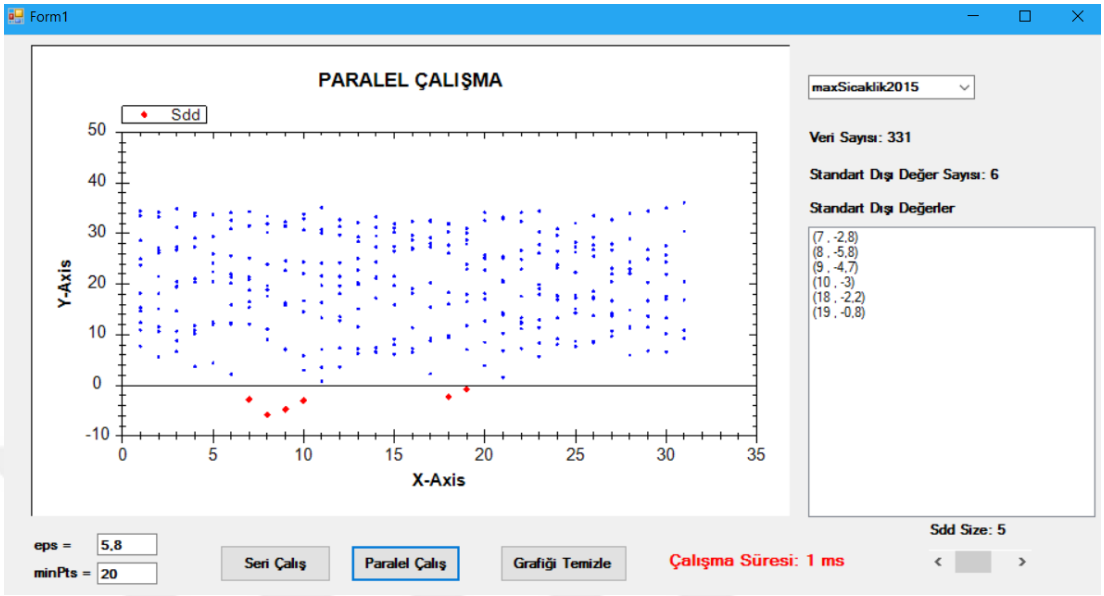
Şekil 4.16. Paralel uygulamanın kod haritası

Şekil 4.17'de veri uzunluğu 31 olan maxSicaklikOcak veri setinin paralel analiz sonuçları gösterilmiştir.



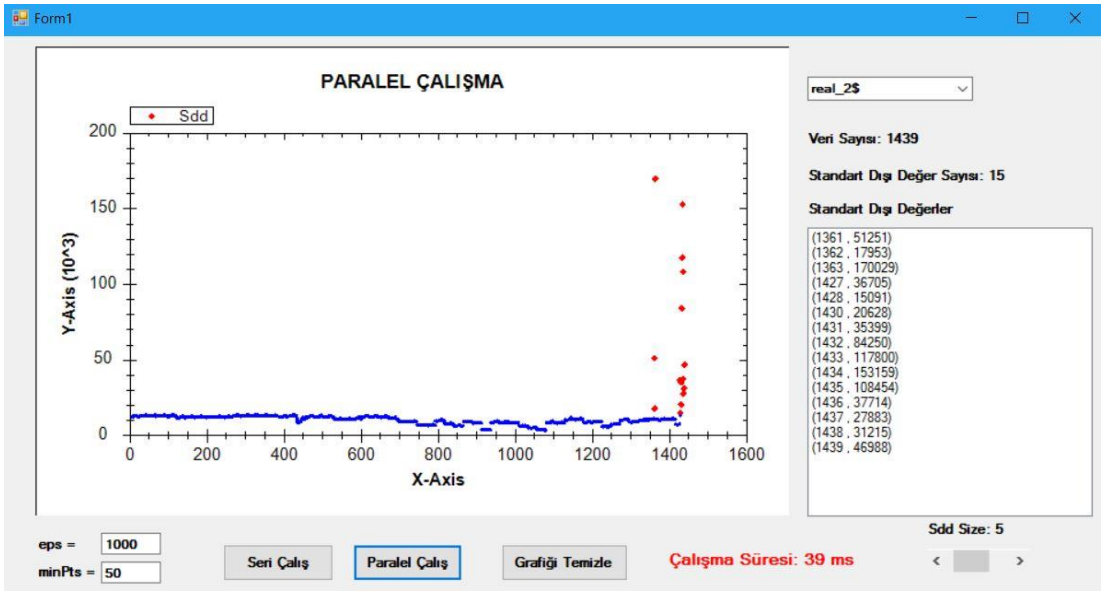
Şekil 4.17. maxSicaklikOcak veri seti paralel analizi

Şekil 4.18'de veri uzunluğu 331 olan maxSicaklik2015 veri setinin paralel analiz sonuçları gösterilmiştir.



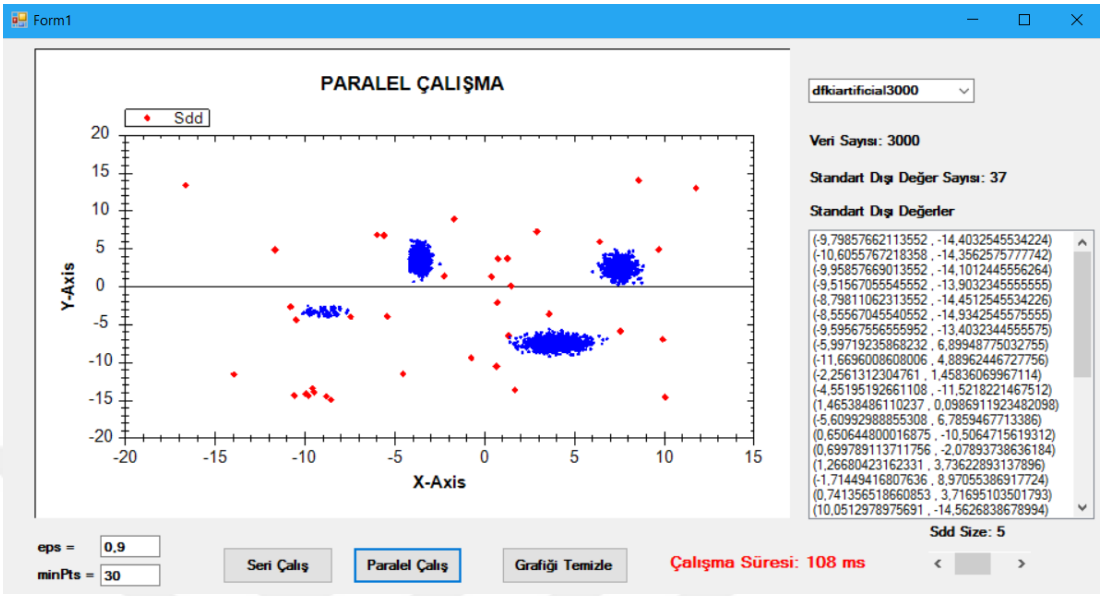
Şekil 4.18. maxSicaklik2015 veri seti paralel analizi

Şekil 4.19'de veri uzunluğu 1439 olan real_2 veri setinin paralel analiz sonuçları gösterilmiştir.



Şekil 4.19. real_2 veri seti paralel analizi

Şekil 4.20'de veri uzunluğu 3000 olan dfkiartificial300 veri setinin paralel analiz sonuçları gösterilmiştir.



Şekil 4.20. dfkiartificial300 veri seti paralel analizi

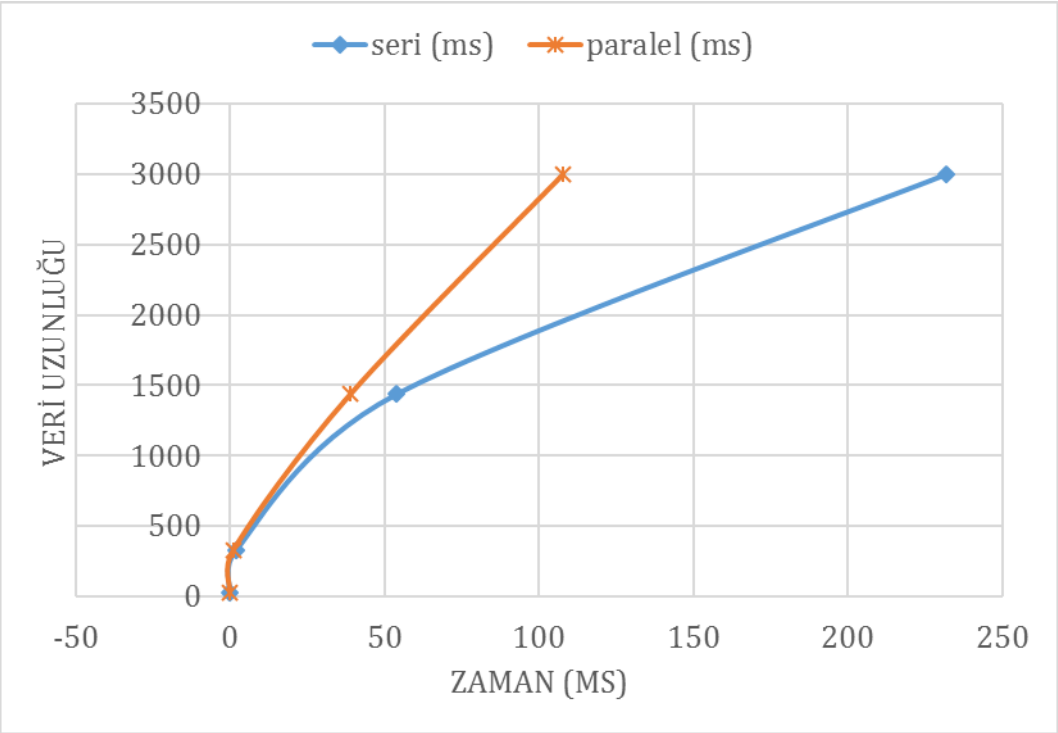
4.5. Seri ve Paralel Uygulamaların Karşılaştırılması

Çizelge 4.6'da seri ve paralel uygulamaların her veri seti için çalışma zamanı gösterilmiştir.

Çizelge 4.6. Seri ve paralel uygulamaların çalışma zamanları

Tablo Adı	Çalışma Zamanı Seri (ms)	Çalışma Zamanı Paralel (ms)
maxSicaklikOcak	0	0
maxSicaklik2015	2	1
real_2	54	39
dfkiartificial300	232	108

Şekil 4.21'de veri uzunluğuna göre çalışma zamanı grafiği gösterilmiştir.



Şekil 4.21. Veri seti uzunluğu çalışma zamanı grafiği (mS)

Şekil 4.21'de görüldüğü gibi veri uzunluğu arttıkça paralel uygulamanın çalışma zamanı azalmış ve performans artışı sağlanmıştır.

4.6. Uygulamanın Standart Dışı Değer Bulmadaki Performansı

Uygulamanın standart dışı değer bulmadaki performansı, real_2 ve dfkiartificial3000 veri setlerinde incelenmiştir. Bu veri setleri içerdiği değerlerin hangilerinin standart dışı değer olduğunu belirttiğinden tercih edilmiştir. Veri setlerindeki üçüncü sütun değerlerin anormal olup olmadığını göstermektedir. Üçüncü sütundaki değer 0 ise normal, 1 ise standart dışı değerleri göstermektedir.

Veri setlerinde belirtilen standart dışı değerler ile geliştirilen uygulamanın bulduğu standart dışı değerler karşılaştırılmıştır. Çizelge 4.7'de uygulamanın

standart dışı değerleri bulmadaki başarı oranları, önceki bölümlerde tespit edilen eps ve minPts değerlerine göre verilmiştir.

Çizelge 4.7. Standart dışı değer bulma başarı oranları

Tablo Adı	Eps ve minPts Değerleri	Veri Setindeki Standart Dışı Değer Sayısı	Uygulamanın Bulduğu Standart Dışı Değer Sayısı	Başarı Oranı (%)
real_2	Eps: 1000 minPts: 50	16	15	93,75
dfkiartificial300	Eps: 0,9 minPts: 30	37	37	94

Çizelge 4.7'de görüldüğü gibi real_2 veri setinde 16 standart dışı değer bulunmasına rağmen uygulama bunların 15 tanesini yakalamıştır. dfkiartificial300 veri setinde ise uygulamanın bulduğu standart dışı değer sayısı veri setindeki standart dışı değer sayısına eşit olmasına rağmen başarı oranı %94 olarak bulunmuştur. Bunun sebebi uygulama veri setindeki standart dışı değer olmayan değerleri, standart dışı değer olarak tespit etmesi ve bazı standart dışı değerleri bulamamış olmasıdır. R Project yazılımı kullanılarak elde edilen knn-dist grafiğinde eps ve minPts değerleri, grafik üzerinden görsel olarak (yaklaşık değer) seçilerek elde edilmiştir. Bu değerlerin seçimi yapılırken daha hassas ondalıklı ya da tam değerlerini hesaplatmak için bir metot geliştirilmesi doğruluk oranını arttırabilecektir.

4.7. Uygulamanın Farklı İşlemcilerde Test Edilmesi ve Sonuçları

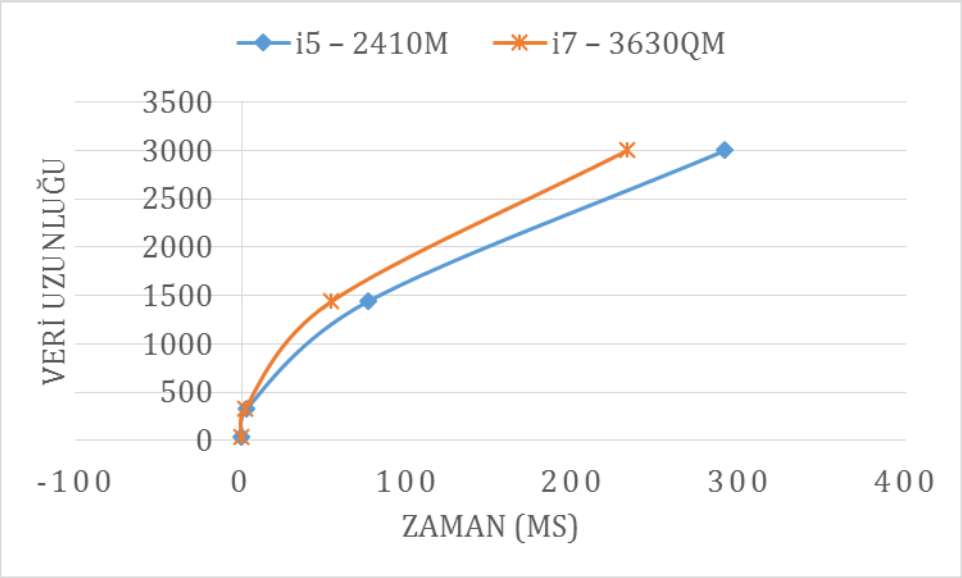
Uygulama öncelikle Intel(R) Core (TM) i7 – 3630QM CPU 2.40GHz işlemci ve 16GB belleğe sahip bilgisayarda test edilmiştir. Sonuçlar; Çizelge 4.5 ve Çizelge 4.6'da gösterilmiştir. Uygulamanın Çizelge 4.8'de özellikleri gösterilen farklı işlemcilere ve belleğe sahip bilgisayarlardaki çalışma süreleri ölçülmüştür.

Çizelge 4.8. Uygulamanın farklı bilgisayar konfigürasyonlarındaki çalışma süresi

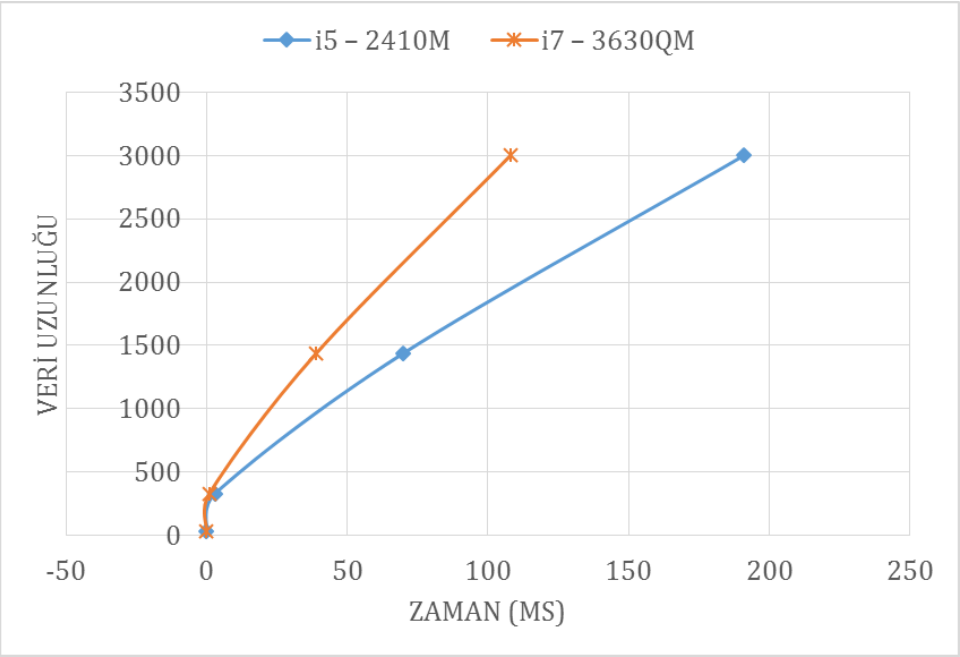
Veri Seti	Bilgisayar Konfigürasyonu			
	İşlemci: Intel(R) Core (TM) i7 – 3630QM CPU 2.40GHz (4 çekirdek- 8 iş parçacığı) Bellek: 16GB		İşlemci: Intel(R) Core (TM) i5 – 2410M CPU 2.30GHz (2 çekirdek- 4 iş parçacığı) Bellek: 4GB	
	Çalışma Süreleri (mS)			
	Seri	Paralel	Seri	Paralel
maxSicaklikOcak	0	0	0	0
maxSicaklik2015	2	2	3	3
real_2	54	39	76	70
dfkiartificial300	232	108	291	191

Bu çalışma farklı performanstaki bilgisayarların hesaplama süresini ölçmek adına Çizelge 4.8'de görüldüğü üzere iki ayrı sistem üzerinde denenmiştir. Denemeler sonucunda işlemci hızı, çekirdek sayısı ve bellek miktarının yüksek olması, toplam işlem süresini kısaltarak, sonucu olumlu yönde etkilemiştir.

Şekil 4.22'de seri uygulamanın farklı işlemcilerdeki veri uzunluğu-zaman grafiği ve Şekil 4.23'de ise paralel uygulamanın veri uzunluğu-zaman grafiği gösterilmiştir.



Şekil 4.22. İşlemcilere göre seri kodun veri uzunluğu – zaman grafiği (mS)



Şekil 4.23. İşlemcilere göre paralel kodun veri uzunluğu – zaman grafiği (mS)

5. TARTIŞMA VE SONUÇLAR

Günümüzde dijital verilerin boyutları giderek artmaktadır. Yaklaşık olarak günde 2,5 kentilyon bayt dijital veri üretilmektedir. Bu veriler sosyal medya, meteoroloji, tıp, finans gibi alanlarda üretilmektedir. Bu verilerin içerisindeki standart dışı verilerin bulunması sahtecilik, dolandırıcılık, hastalık teşhisi ve çeşitli anomalilerin tespitinin hızlı yapılabilmesi hayati önem arz etmektedir.

Bu çalışmada standart dışı verilerin tespiti için bir veri madenciliği yöntemi olan yoğunluk tabanlı kümeleme algoritmalarından DBSCAN algoritması kullanılmıştır. DBSCAN algortiması veri setlerindeki noktaları kümeleyerek küme dışında kalan noktaları standart dışı veri olarak işaretlemektedir.

Uygulama Microsoft .Net programlama dili olan C# ile Visual Studio ortamında geliştirilmiştir. Geliştirilen uygulamada 4 faklı veri seti üzerinde çalışılmış ve veri setleri analiz edilmiştir. Veri setlerindeki noktalar grafik üzerinde gösterilmiş standart dışı değerler görsel olarak izlenmiştir.

DBSCAN algoritması kümeleme ve standart dışı değerleri tespit etme bakımından başarılı olsa da eps ve minPts parametrelerine ihtiyaç duyması algoritmanın dezavantajı olarak görülmüştür. Optimum eps ve minPts değerlerini belirlemek için kNN-dist grafiği R Project yazılımı sayesinde çizdirilmiş ve uygun değerler grafikten belirlenmiştir.

.Net Framework 4.0 ile gelen TPL içindeki paralel sınıflar yardımıyla uygulama paraleleştirilmeye çalışılmıştır. Uygulamanın seri ve paralel çalışması karşılaştırılmıştır. Paralel çalışma veri sayısı arttıkça daha iyi performans göstermiş ve uygulama daha kısa sürede sonuca ulaşmıştır. Küçük verilerde paralel programlama performansa bir katkı sağlamamıştır. Uygulama aynı zamanda Intel i7 ve i5 işlemcilerde de test edilmiş performans grafikleri gösterilmiştir.

Bu çalışma, DBSCAN algoritmasının standart dışı değerlere duyarlı olduğu ve bu değerleri tespit etmede kullanılabileceğini göstermektedir. Uygulama üzerinde yapılan analizlerin yüksek doğruluk oranında standart dışı değerleri bulması veri setlerindeki anomalilerin bulunması açısından önemli olup geliştirilmeye açıktır.

Gelecekte, DBSCAN algoritması için uygun eps ve minPts parametrelerinin otomatik seçimini yapan bir uygulama geliştirilmesi, analizi yapılan veri setlerindeki standart dışı verilerin tespiti için faydalı olacaktır.

Standart dışı değerlerin bulunmasında farklı kümeleme algoritmaları kullanılarak uygulamalar geliştirilip test edilmesinin faydalı olacağı düşünülmektedir. Performans ve hız açısından C# dışında diğer programlama dilleri (C, C++, Java, Python vd.) ile de uygulamalar geliştirilip test edilebilir. Bir diğer yaklaşım ise GPU programlamanın alternatif olarak kullanılabileceğidir.

KAYNAKLAR

Akbulut, S., 2006. Veri Madenciliği Teknikleri İle Bir Kozmetik Markanın Ayrılan Müşteri Analizi ve Müşteri Segmentasyonu, Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 103s, Ankara.

Akçay, M., Erdem, H. A., 2010. Intel Parallel Studio ile Paralel Hesaplama. Erişim Tarihi: 20.11.2016. http://inet-tr.org.tr/inetconf18/bildiri/30.pdf

Akgül, M., 2015. Veri Madenciliği. Erişim Tarihi: 01.02.2017. https://akgulmustafa.wordpress.com/2015/03/01/nedir-bu-veri-madenciligi/

Aktaş, V., 2010. Visual Studio İle Her Yönüyle C# 4.0. Kodlab, 771s, İstanbul.

Aktürk Z., Acemoğlu H., 2010. Sağlık Çalışanları için Araştırma ve Pratik İstatistik, Anadolu Matbaası, 325s, İstanbul.

Albahari, J., 2011. Threading in C#. Erişim Tarihi: 20.11.2016. http://www.albahari.info/threading/threading.pdf

Altıntaş, V., Yegenoğlu, E. D., 2011. Görüntü İşlemede Seri ve Paralel Programlamanın Performansı. 6th International Advanced Technologies Symposium (IATS'11), 16-18 May, Elazığ, 132-134.

Atılgan, C., 2014. Kümeleme Algoritmaları ve Paralelleştirilmeleri. Ege Üniversitesi, Fen bilimleri Enstitüsü, Yüksek Lisans Tezi, 97s, İzmir.

Barney, B., 2016. Lawrence Livermore National Laboratory. Erişim Tarihi: 16.11.2016. https://computing.llnl.gov/tutorials/parallel_comp/

Bilgin, T. T., Çamurcu, Y., 2005. DBSCAN, OPTICS ve K-Means Kümeleme Algoritmalarının Uygulamalı Karşılaştırılması. Politeknik Dergisi, 8 (2), 139-145.

Brachman, R., Anand, T., 1996. The Process of Knowledge Discovery in Databases: A Human Centered Approach. Erişim Tarihi: 30.01.2016. https://www.aaai.org/Papers/KDD/1996/KDD96-014.pdf

Breunig, M., Kriegel, H. P., Raymond, T. N., Sander, J., 2000. LOF: Identifying Density-Based Local Outliers. ACM International Conference on Management of Data and Symposium on Principles of Database Systems, 15-18 May 2000, Dallas, USA, 93-104.

Çelik, M., Dadaşer Ç. F., Dokuz, A.,Ş., 2011. Anomaly Detection in Temperature Data Using DBSCAN Algorithm. 2011 International Symposium on Innovations in Intelligent Systems and Applications, 15-18 June 2011, İstanbul, Turkey, 91-95.

Durmuş, B., 2013. Sanal Paralel Makine. Dumlupınar Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 67s, Kütahya.

Durrani, H., 2013. Parallelization of K-Means and Dbscan Clustering Algorithms on A Hpc Cluster. Middle East Technical University, Graduate School of Natural and Applied Science, Yüksek Lisans Tezi, 27s, Ankara.

Elbatta, M.T.H., Ashour, W. M., 2013. A Dynamic Method for Discovering Density Varied Clusters, International Journal of Signal Processing, Image Processing and Pattern Recognition, 6 (3), 123-134.

Ercan, U., Akar, H., Koçer, A., 2013. Paralel Programlamada Kullanılan Temel Algoritmalar. Akademik Bilişim Konferansı, 23-25 Ocak, Antalya, 861-866.

Ester, M., Kriegel, H. P., Sander, J., Xu, X., 1996. A Density Based Algorithm for Discovering Clusters in Large Spatial Data Sets with Noise, 2nd International Conference on Knowledge Discovery and Data Mining, 2 – 4 August, Portland, USA, 226–231.

Freeman, A., 2010. Pro .NET 4 Parallel Programming in C#, Apress, 311p, USA.

Gargano, M., L., Raggad, B., G., 1999. Data mining–A Powerful Information Creating Tool, OCLC Systems & Services 2(15), 81–90.

German Research Center for Artificial Intelligence, 2016. Erişim Tarihi: 03.12.2016. http://madm.dfki.de/_media/downloads/dfki-artificial-3000-unsupervised-ad.zip

Güçlü, M., 2012. Yapay Bağışıklık Sistem Tabanlı Algoritma İle Aykırı Değer Tespiti. Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 93s, İstanbul.

Güneş, A., 2011. Paralel Programlama İle El Yazısı Rakamlarının Tanınması. Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 80s, Isparta.

Gürsakal, N., 2001. Bilgisayar Uygulamalı İstatistik 1, Alfa Yayınları, 527s, İstanbul.

Hui, S., Jha, G., 2000. Application data mining for customer service support, Information and Management, 38 (1), 1-13.

IBM, 2016. Erişim Tarihi: 21.12.2016. https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html

İnce, K., 2013. Çok Çekirdekli Mimarilerde Paralel Programlama ile Genetik Algoritmaların Uygulaması. İnönü Üniverisitesi, Fen bilimleri Enstitüsü, Yüksek Lisans Tezi, 80s, Malatya.

Kaçka, S., 2011. Bilimsel Hesaplama Problemlerinin Çözümünde Paralel Hesaplama Yöntemlerinin Kullanılması. Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 82s, Konya.

Kiang, M. Y., Kumar, A., 2001. An Evaluation of Self-organizing Map Network as a Robust Alternative to Factor Analysis in Data Mining Applications. Information Systems Research, 12 (4), 177-194.

Koldere A. Y., 2008. Veri Madenciliğinde Kümeleme Algoritmaları ve Kümeleme Analizi, Doktora Tezi, 189s, İstanbul.

Kriegel, H. P., Kröger, P., Zimek, A., 2010. Outlier Detection Techniques. Erişim Tarihi:01.01.2017. http://www.siam.org/meetings/sdm10/tutorial3.pdf

KTÜ, 2016. Bilgisayar Ağları ve Dağıtık Programlama. Erişim Tarihi: 15.11.2016. http://www.ktu.edu.tr/dosyalar/bilgisayar_a4c48.pdf

Kumar, V., Kumar, S., Singh, A. K., 2013. Outlier Detection: A Clustering-Based Approach. International Journal of Science and Modern Engineering (IJISME), 1 (7), 16-19.

Liao, S., 2003. Knowledge Management Technologies and Applications—Literature Review from 1995 to 2002. Expert Systems With Applications 2(25), s. 155–164.

Microsoft, 2016. Erişim Tarihi: 11.11.2016. https://msdn.microsoft.com/en-us/library/dd460693(v=vs.110).aspx

Moreira, A., Y. S. Maribel, Carneiro, S., 2005. Density-Based Clustering Algorithms-DBSCAN and SNN. University of Minho, 25 July, Portugal.

Moshkovich, H. M., Mechitov, A., I., Olson, D., L., 2002. Rule Induction in Data Mining: Effect of Ordinal Scales. Expert Systems with Applications, 22 (4), 303-311.

Ovla, H. D., Taşdelen, B., 2012. Aykırı Değer Yönetimi. Mersin Üniversitesi Sağlık Bilimleri Dergisi, 5(3), 1-8.

Özdamar, E. Ö., 2002. Veri Madenciliğinde Kullanılan Teknikler ve Bir Uygulama. Mimar Sinan Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 126s, İstanbul.

Petrovskiy, M. I., 2003. Outlier Detection Algorithms in Data Mining Systems. Programming and Computer Software, 29 (4), 228-237.

Sarıman, G., 2011. Veri Madenciliğinde Kümeleme Teknikleri Üzerine Bir Çalışma: K-Means ve K-Medoids Kümeleme Algoritmalarının Karşılaştırılması. Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü Dergisi, 15 (3), 192-202.

Sarıman, G., 2011. Paralel Programlama İle Web Madenciliğinde Log Analizi. Süleyman Demirel Üniversitesi, Fen Bilimleri Enstitüsü, 103s, Isparta.

Sever, S. Z., 2010. Yoğunluk Tabanlı Kümeleme Metodları Kullanılarak Paralel Veri Madenciliği Gerçekleştirilmesi. Maltepe Üniersitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 100s, İstanbul.

Shaw, M. J., Subramaniam, C, Tan, G.W. ve Welge, M.E., 2001. "Knowledge Management And Data Mining For Marketing", Decision Support Systems 1(31), s. 127–137.

STHDA, 2016. Erişim Tarihi: 28.12.2016. http://www.sthda.com/english/wiki/dbscan-density-based-clustering-for-discovering-clusters-in-large-datasets-with-noise-unsupervised-machine-learning

Şenyurt, B. S., 2009. TPL için Önemli Bir Kavram : Task [Beta 1]. Erişim Tarihi: 19.11.2016. http://www.buraksenyurt.com/post/TPL-icin-Onemli-Bir-Kavram-Task.aspx

Tekbir, M., 2009. Aykırı Değer Tespitinde Yoğunluk Tabanlı Kümeleme Yöntemleri. Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 70s, İstanbul.

Turgut, H., 2012. Veri Madenciliği Süreci Kullanılarak Alzheimer Hastalığı Teşhisine Yönelik Bir Uygulama. Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 101s, Isparta.

Üçüncü, B., 2012. Task Paralel Library. Erişim Tarihi: 16.11.2016. http://www.bayramucuncu.com/task-parallel-library-tlp-c/

Vatansever, M., Büyüklü, A., H., 2009. Using Visual Data Mining Techniques in Clustering Analysis and An Application. Sigma Journal of Engineering and Natural Sciences, 27(2), 83-104.

Vural, A., 2007. Aykırı Değerlerin Regresyon Modellerine Etkileri ve Sağlam Kestiriciler. Marmara Üniversitesi, Sosyal Bilimler Enstitüsü, Yüksek Lisans Tezi, 73s, İstanbul.

Wikipedia, 2016. Erişim Tarihi: 28.11.2016. https://en.wikipedia.org/wiki/DBSCAN

Wu, B., Liao, F., Zhang, D., 2012. Parallel Outlier Detection in Dial-back Fraud Calls. Proceedings of the 2012 2nd International Conference on Computer and Information Application, 29-31 Dec 2012, Changchun, China, 494-497.

Yahoo, 2016. Erişim Tarihi: 03.12.2016. http://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70

Yaycılı, A. Ö., 2006. Temel Bileşenler Analizi İçin Robust Algoritmaları. Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 65s, Ankara.

EKLER

EK A. Uygulamanın C# Kodu

EK A. Uygulamanın C# Kodu

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using ZedGraph;
using System.Data.SqlClient;
namespace WindowsFormsApplication2
{
public class Point
  {
    public const int NOISE = -1;
    public const int UNCLASSIFIED = 0;
    public double X, Y;
    public int ClusterId;
    public Point(double x, double y)
    {
      this.X = x;
      this.Y = y;
    }

  }
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    GraphPane mypane;
    SqlConnection con;
    SqlDataAdapter da;
    DataSet ds;
    string tablo;
    DataTable dt;
    double eps, minPts;
    private void button1_Click(object sender, EventArgs e)
    {
      if (comboBox1.Text=="Tablo Seçiniz")
      {
        MessageBox.Show("Tablo Seçiniz");
      }
      else
      {
        Stopwatch sw = new Stopwatch();
        sw.Reset();
```

```
            eps = Convert.ToDouble(textBox1.Text);
            minPts = Convert.ToDouble(textBox2.Text);
            zedGraphControl1.GraphPane.Title = "SERİ ÇALIŞMA";
            List<Point> points = new List<Point>();
            List<double> timestamp = new List<double>();
            List<double> value = new List<double>();
            List<Point> sddyahoo = new List<Point>();
            List <Point> sdddbscan= new List<Point>();


            for (int i = 0; i < dt.Rows.Count; i++)
            {
               timestamp.Add(Convert.ToDouble(dt.Rows[i][0]));
               value.Add(Convert.ToDouble(dt.Rows[i][1]));
               points.Add(new Point(timestamp[i], value[i]));
            }

            sw.Start();
            List<List<Point>> clusters = GetClusters(points, eps, minPts);
            sw.Stop();
            label1.Text = "Çalışma Süresi: " + sw.ElapsedMilliseconds.ToString() + "
ms";
            DrawPoints(points, clusters,hScrollBar1.Value);
         }
      }
      static List<List<Point>> GetClusters(List<Point> points, double eps, double
minPts)
      {
         if (points == null) return null;
         List<List<Point>> clusters = new List<List<Point>>();
         int clusterId = 1;
         for (int i = 0; i < points.Count; i++)
         {
            Point p = points[i];
            if (p.ClusterId == Point.UNCLASSIFIED)
            {

               //if (ExpandCluster(points, p, clusterId, eps, minPts)) clusterId++;
               ExpandCluster(points, p, clusterId, eps, minPts);
            }
         }

         //int maxClusterId = points.OrderBy(p => p.ClusterId).Last().ClusterId;
         //if (maxClusterId < 1) return clusters;
         //for (int i = 0; i < maxClusterId; i++)
         clusters.Add(new List<Point>());
         foreach (Point p in points)
         {
            if (p.ClusterId > 0) clusters[p.ClusterId - 1].Add(p);
```

```
        }
        return clusters;
    }
    static List<Point> GetRegion(List<Point> points, Point p, double eps)
    {
        List<Point> region = new List<Point>();
        for (int i = 0; i < points.Count; i++)
        {
            double diffX = points[i].X - p.X;
            double diffY = points[i].Y - p.Y;
            double distSquared = Math.Sqrt(diffX * diffX + diffY * diffY);

            if (distSquared <= eps) region.Add(points[i]);
        }
        return region;
    }
    static void ExpandCluster(List<Point> points, Point p, int clusterId, double
eps, double minPts)
    {
        List<Point> seeds = GetRegion(points, p, eps);
        if (seeds.Count < minPts)
        {
            p.ClusterId = Point.NOISE;
        }
        else
        {
            for (int i = 0; i < seeds.Count; i++) seeds[i].ClusterId = clusterId;
            seeds.Remove(p);
            while (seeds.Count > 0)
            {
                Point currentP = seeds[0];
                List<Point> result = GetRegion(points, currentP, eps);
                if (result.Count >= minPts)
                {
                    for (int i = 0; i < result.Count; i++)
                    {
                        Point resultP = result[i];
                        if (resultP.ClusterId == Point.UNCLASSIFIED || resultP.ClusterId
== Point.NOISE)
                        {
                            if (resultP.ClusterId == Point.UNCLASSIFIED)
seeds.Add(resultP);
                            resultP.ClusterId = clusterId;
                        }
                    }
                }
                seeds.Remove(currentP);
            }
        }
```

```
    }
    private void button2_Click(object sender, EventArgs e)
    {
      if (zedGraphControl1.GraphPane.Title =="GRAFİK" )
      {
        MessageBox.Show("Grafik Yok");
      }
      else
      {
        zedGraphControl1.GraphPane.Title = "GRAFİK";
        label1.Text = "Çalışma Süresi: ";
        label3.Text = "Standart Dışı Değer Sayısı: ";
        mypane.CurveList.Clear();
        zedGraphControl1.Refresh();
        listBox1.Items.Clear();
      }
    }
    private void Form1_Load(object sender, EventArgs e)
    {

      zedGraphControl1.GraphPane.Title = "GRAFİK";
      con = new SqlConnection(@"Data Source =.\SQLEXPRESS; Initial Catalog =
DATA; Integrated Security = True");
      con.Open();
      DataTable dt = con.GetSchema("Tables");
      for (int i = 0; i < dt.Rows.Count; i++)
      {
        comboBox1.Items.Add(dt.Rows[i]["TABLE_NAME"]);
      }
      con.Close();
    }
    private void button3_Click(object sender, EventArgs e)
    {

      if (comboBox1.Text == "Tablo Seçiniz")
      {
        MessageBox.Show("Tablo Seçiniz");
      }
      else
      {
        Stopwatch sw = new Stopwatch();
        sw.Reset();
        eps = Convert.ToDouble(textBox1.Text);                    minPts =
Convert.ToDouble(textBox2.Text);
        zedGraphControl1.GraphPane.Title = "PARALEL ÇALIŞMA";
        List<Point> points = new List<Point>();
        List<double> timestamp = new List<double>();
        List<double> value = new List<double>();
        for (int i = 0; i < dt.Rows.Count; i++)
```

```
            {
                timestamp.Add(Convert.ToDouble(dt.Rows[i][0]));
                value.Add(Convert.ToDouble(dt.Rows[i][1]));
                points.Add(new Point(timestamp[i], value[i]));
            }
            sw.Start();
            List<List<Point>> clusters = ParallelGetClusters(points, eps, minPts);
            sw.Stop();
            label1.Text = "Çalışma Süresi: " + sw.ElapsedMilliseconds.ToString() + "
ms";
            DrawPoints(points,clusters,hScrollBar1.Value);
        }
    }
    static void  ParallelExpandCluster(List<Point> points, Point p, int clusterId,
double eps, double minPts)
    {
        List<Point> seeds = ParallelGetRegion(points, p, eps);
        if (seeds.Count < minPts)
        {
            p.ClusterId = Point.NOISE;
        }
        else
        {
            for (int i = 0; i < seeds.Count; i++) seeds[i].ClusterId = clusterId;
            seeds.Remove(p);
            while (seeds.Count > 0)
            {
                Point currentP = seeds[0];
                List<Point> result = ParallelGetRegion(points, currentP, eps);
                if (result.Count >= minPts)
                {
                    for (int i = 0; i < result.Count; i++)
                    {
                        Point resultP = result[i];
                        if (resultP.ClusterId == Point.UNCLASSIFIED || resultP.ClusterId
== Point.NOISE)
                        {
                            if (resultP.ClusterId == Point.UNCLASSIFIED)
seeds.Add(resultP);
                            resultP.ClusterId = clusterId;
                        }
                    }
                }
                seeds.Remove(currentP);
            }
        }
    }
    static List<Point> ParallelGetRegion(List<Point> points, Point p, double eps)
    {
```

```
        List<Point> region = new List<Point>();
        for (int i = 0; i < points.Count; i++)
        {

          double diffX = points[i].X - p.X;
          double diffY = points[i].Y - p.Y;
          double distSquared = Math.Sqrt(diffX * diffX + diffY * diffY);
          if (distSquared <= eps)
          { region.Add(points[i]); }
        }

        return region;
      }
      static List<List<Point>> ParallelGetClusters(List<Point> points, double eps,
double minPts)
      {
        if (points == null) return null;
        List<List<Point>> clusters = new List<List<Point>>();
        int clusterId = 1;

        Parallel.For(0, points.Count, i =>
        {
          Point p = points[i];
          if (p.ClusterId == Point.UNCLASSIFIED)
          {
           ParallelExpandCluster(points, p, clusterId, eps, minPts);
          }
        });

        clusters.Add(new List<Point>());
        foreach (Point p in points)
        {
          if (p.ClusterId > 0) clusters[p.ClusterId - 1].Add(p);
        }
        return clusters;
      }
      public void DrawPoints(List<Point> points, List<List<Point>> clusters, float
sddsymbolsize)
      {
        label8.Text ="Sdd Size: "+ sddsymbolsize.ToString();
        double[] X1 = new double[points.Count]; double[] Y1 = new
double[points.Count];
        for (int i = 0; i < points.Count; i++)
        {
          X1[i] = points[i].X;
          Y1[i] = points[i].Y;
        }
        PointPairList pl1 = new PointPairList();
        pl1.Add(X1, Y1);
```

```
        int total = 0;
        int outlier = 0;
        for (int i = 0; i < clusters.Count; i++)
        {
          total += clusters[i].Count;
          int j = 0;
          double[] X2 = new double[clusters[i].Count]; double[] Y2 = new
double[clusters[i].Count];
          int count = clusters[i].Count;

          foreach (Point p in clusters[i])
          {
            X2[j] = p.X;
            Y2[j] = p.Y;
            j++;
          }
          mypane = zedGraphControl1.GraphPane;
          PointPairList pl2 = new PointPairList();
          pl2.Add(X2, Y2);
          LineItem mycurve = mypane.AddCurve("", pl2, Color.Blue,
SymbolType.Circle);
          mycurve.Line.IsVisible = false;
          mycurve.Symbol.Size = 2.0F;
          mycurve.Symbol.Fill = new Fill(Color.Blue);
          zedGraphControl1.AxisChange();
          zedGraphControl1.Invalidate();
          zedGraphControl1.Refresh();

        }

        zedGraphControl1.IsShowPointValues = true;
        outlier = points.Count - total;
        label3.Text = "Standart Dışı Değer Sayısı: " + outlier.ToString();
        double[] Xout = new double[outlier]; double[] Yout = new double[outlier];
        int k = 0;
        foreach (Point p in points)
        {
          if (p.ClusterId == Point.NOISE)
          {
            string std = "(" + p.X.ToString() + " , " + p.Y.ToString() + ")";
            listBox1.Items.Add(std);
            Xout[k] = p.X;
            Yout[k] = p.Y;
            k++;
          }
        }
        PointPairList plout = new PointPairList();
        plout.Add(Xout, Yout);
```

```
        LineItem mycurveOutlier = mypane.AddCurve("Sdd", plout, Color.Red,
SymbolType.Diamond);
        mycurveOutlier.Line.IsVisible = false;
        mycurveOutlier.Symbol.Size = sddsymbolsize;
        mycurveOutlier.Symbol.Fill = new Fill(Color.Red);
        zedGraphControl1.AxisChange();
        zedGraphControl1.Invalidate();
        zedGraphControl1.Refresh();
      }
      private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
      {
        label8.Text = "Sdd Size: "+hScrollBar1.Value.ToString();
        zedGraphControl1.AxisChange();
        zedGraphControl1.Invalidate();
        zedGraphControl1.Refresh();
      }
      private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
      {
        tablo = comboBox1.Text;
        string sorgu = "SELECT *FROM " + tablo;
        da = new SqlDataAdapter(sorgu, con);
        ds = new DataSet();
        con.Open();
        da.Fill(ds, tablo);
        dt=ds.Tables[tablo];
        con.Close();
        label5.Text = "Veri Sayısı: " + dt.Rows.Count.ToString();
        if (dt.Rows.Count == 31) { textBox1.Text = "1,5"; textBox2.Text = "1"; };
        if (dt.Rows.Count == 331) { textBox1.Text = "5,8"; textBox2.Text = "20"; };
        if (dt.Rows.Count == 1439) { textBox1.Text = "1000"; textBox2.Text = "50";
};
        if (dt.Rows.Count == 3000) { textBox1.Text = "0,9"; textBox2.Text = "30"; };
      }
    }
}
```

ÖZGEÇMİŞ

Adı Soyadı : Hüseyin YAŞAR

Doğum Yeri ve Yılı : Şarkikaraağaç, 1984

Medeni Hali : Evli

Yabancı Dili : İngilizce

E-posta : huseyin.yasar@outlook.com

Eğitim Durumu

Lise : Isparta Anadolu Lisesi

Lisans : SDÜ, Teknik Eğitim Fakültesi, Bilgisayar Sistemleri Öğretmenliği

Mesleki Deneyim

Isparta Valiliği (Bilgi İşlem Sorumlusu)	2007 - 2011
Milli Eğitim Bakanlığı (Bilişim Teknolojileri Öğrt)	2011 - … (halen)