İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DEMETLEME PROBLEMİ İÇİN PARALEL KARINCA YAKLAŞIMI

YÜKSEK LİSANS TEZİ

Özlem GEMİCİ

Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Programı : BİLGİSAYAR MÜHENDİSLİĞİ

HAZİRAN 2007

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DEMETLEME PROBLEMİ İÇİN PARALEL KARINCA YAKLAŞIMI

YÜKSEK LİSANS TEZİ
Özlem GEMİCİ
(504041536)

Tezin Enstitüye Verildiği Tarih : 7 Mayıs 2007
Tezin Savunulduğu Tarih : 12 Haziran 2007

1. Tez Danışmanı : Yrd. Doç Dr. Şima ETANER UYAR

Diğer Jüri Üyeleri Yrd. Doç Dr. Şule GÜNDÜZ ÖĞÜDÜCÜ

Dç Dr. Zümray DOKUR ÖLMEZ

HAZİRAN 2007

ÖNSÖZ

Tez çalışmam öncesinde ve çalışma sırasında beni yönlendirdiği, her konuda yol gösterdiği ve yapıcı önerileri ile çalışmamı tamamlamamı sağladığı için danışmanım Sayın Yrd. Doç. Dr. Şima ETANER UYAR 'a teşekkür ederim

Her zaman bana destek olan annem Rukiye GEMİCİ ve babam Salih GEMİCİ 'ye ve çalışmamın en zor zamanlarında desteğini esirgemeyen eşim İsmail GÜNEŞ 'e teşekkür ederim.

Haziran 2007 **Özlem GEMİCİ**

İÇİNDEKİLER

KISALTMALAR

MPI	: Message Passing Interface
MPI.NET	: Message Passing Interface for .NET
OOMPI	: Object Oriented MPI
CLR	: Common Language Runtime
CLI	: Common Language Infrastructure
PELABS	: Parallel Environment for Lattice Based Simulations
MIMD	: Multiple Instruction Multiple Data

TABLO LİSTESİ

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

SEMBOL LİSTESİ

k_p	:	Nesneyi alma kararı için kullanılan eşik değeri
k_d	:	Nesneyi bırakmak için kullanılan eşik değeri
σ	:	Komşuluk boyutu
$d(i, j)$	:	i nesnesi ile j nesnesi arasındaki uzaklık
α	:	[0,1] aralığında bir değer
μ	:	Veriler arasındaki ortalama uzaklık
N_{items}	:	Nesne sayısı
N_{cells}	:	Grid üzerindeki hücre sayısı
c_{ij}	:	i ve j indisli grid hücresi
$p_{pick}(i)$	:	i nesnesinin alınması olasılığı
$p_{drop}(i)$	:	i nesnesinin bırakılması olasılığı
$f(i)$	:	i nesnesi çevresindeki yoğunluk

DEMETLEME PROBLEMİ İÇİN PARALEL KARINCA YAKLAŞIMI

ÖZET

Verilerin özelliklerine göre gruplara ayrılması veri madenciliği, görüntü işleme, örüntü tanıma, makine öğrenmesi gibi pek çok uygulama için önemli bir problemdir. Demetleme, verilerin belli karakteristik özelliklerinin benzerliğine göre gruplara ayrılmasıdır. Karakteristik özelliklerin benzerliğine ise belirlenen bir uzaklık ölçüsü hesaplanarak karar verilir.

Bu çalışmada öncelikle demetleme problemi çözümü için karınca demetleme yöntemi ardışıl olarak gerçeklenmiştir. Gerçeklenen yöntem karınca kolonisinin demetleme problemlerinde kullanımında karıncanın belleğe sahip olması, karıncanın demetleme yapılacak 2 boyutlu ortamda demetlenecek veri bulunmayan alanlarda dolaşmasının engellenmesi, karıncanın her adımda mutlaka bir veri alma veya bırakma işlemi gerçekleştirmesi ve algoritma parametrelerinin adaptif olması gibi özellikler içermektedir.

Karınca demetleme algoritmaları, demet oluşturma başarımı bakımından diğer demetleme algoritmaları ile benzer sonuçları vermektedir. Özellikle zaman ölçümleri bakımından diğer algoritmalardan daha kötü sonuçlar verdiği daha önce yapılan çalışmalarla gösterilmiştir. Ancak demetleme işlemi yapılacak olan veritabanı özelliklerini belirten parametrelerin sayısı arttıkça karınca demetleme algoritmasının zaman performansı diğer algoritmalara yaklaşmaktadır.

Gerçeklenen algoritma seçilen paralel kütüphane kullanılarak iteratif bir yaklaşım ile paralelleştirilmiştir. Paralelleştirme için MPI kütüphanesi seçilmiştir. Bu kütüphanenin ardışıl çözümün de gerçeklendiği .NET ortamında ve C# dili ile birlikte kullanılabilmesini sağlayan MPI.NET kütüphanesi paralelleştirme rutinleri için kullanılmıştır. Demetlenecek veriler ve demetleme ortamını oluşturan 2 boyutlu ortam ve toplam adım sayısı işlemciler arasında paylaştırılır. İşlemcilerden biri ana işlemci olarak belirlenir. Ana işlemci dahil her işlemci kendine atanan alanda ardışıl demetleme çözümünü çalıştırır ve çalışmasının sonlandığını ana işlemciye bildirir. Ana işlemci ardışıl demetleme çözümünü bütün verileri ve 2 boyutlu alanın tamamını kullanarak tekrar çalıştırır. Demetleme algoritmasının daha iyi bir noktadan başlayabilmesi amaçlanmaktadır.

Yapılan çalışmada gerçeklenen iteratif ve kısmi paralelleştirme işleminin özellikle algoritma sonlanma zamanında iyileşmeye sebep olup olmayacağı araştırılmıştır. Yapılan testlerde iki ve dört işlemci ile seçilen veritabanları için performans ölçüm sonuçları elde edilmiştir. Algoritmanın daha iyi bir noktadan başlamasını sağlamak ve atılacak toplam adım sayısının diğer işlemcilerle paylaşmasını sağlamak algoritma

çalışma zamanında iyileşmeyi sağlamıştır. Haberleşme maliyetinin diğer paralelleştirme yöntemine göre çok düşük olması da algoritmanın avantajlarından biridir.

PARALLEL ANT-BASED CLUSTERING

SUMMARY

Dividing datasets into groups regarding their properties is an important problem to solve in data mining, pattern recognition and machine learning. Clustering is classification of similar objects into different groups so that the data in each subset share common traits. Decision of similarity between two data items is due to a dissimilarity measure that computed using characteristic properties of the dataset.

In this study, sequential ant-based clustering is implemented. Implemented method has properties as ant has memory, ant does not walk around empty cells in the 2-dimensional clustering environments, and ants pick or drop a data object in every step and parameter settings are adaptive.

Performance measurements of ant based clustering are as successful as other clustering algorithms in literature. It is shown in earlier studies that ant based clustering performs worse regarding time measurement compared to other clustering algorithms. However, if the number of attributes that compose features space increases then time measurement has closer values to time measurements of other clustering algorithms like k-means or average link.

The implementation is parallelized using an iterative approach through the chosen parallelization library, which is MPI. MPI.NET library is used for parallelization routines. MPI.NET allows the programmer to use the MPI library in the .NET environment using the C# programming language, which also forms the development environment of the sequential algorithm. All the processors share the dataset to cluster and the 2-dimensional clustering environment. One of the processors is marked as master. Each of the processors, including the one denoted as master, runs the sequential algorithm and informs the master processor when the run ends. After all processors finish clustering the data assigned to them, the master processor runs the sequential algorithm again, but this time it uses all the items in the dataset and the whole 2-dimensional clustering environment. The aim of the first stage is to provide a better start point for the clustering algorithm performed by the master in the second stage.

In this study, it is analyzed if the implemented iterative and partially parallel clustering method causes a better time performance then its fully sequential version. Tests are executed for two and four processors on selected example datasets. As stated above, the aim is to make the second stage of the algorithm to start from a better initial situation for clustering by dividing the grid and the number of iterations among the processors. The aim of this study is to speed up the ant clustering method,

which is normally very time consuming. Thus, no steps are taken to improve the clustering solution. As a result of the experiments, it is seen that the proposed algorithm performs better based on time measurements. However, it does not improve other clustering performance measures. This is an expected outcome. The communication cost is very small compared to other ant clustering parallelization techniques found in literature.

1. GİRİŞ

Demetleme verilerin belli karakteristik özelliklerinin benzerliğine göre gruplara ayrılmasıdır. Karakteristik özelliklerin benzerliğine ise belirlenen bir uzaklık ölçüsü hesaplanarak karar verilir. Karakteristik özellikler; gözlemler, veri öğeleri veya özellik vektörleri ile temsil edilebilir [1].

Bu çalışmada, öncelikle karıncalarla demetleme konusunda yapılan çalışmalar incelenmiş ve bu algoritmalar arasından en iyi performansla çalışacağı düşünülen algoritma gerçeklenmiştir. Gerçeklenen yöntem karınca kolonisinin demetleme problemlerinde kullanımında karıncanın belleğe sahip olması, karıncanın demetleme yapılacak 2 boyutlu ortamda demetlenecek veri bulunmayan alanlarda dolaşmasının engellenmesi, karıncanın her adımda mutlaka bir veri alma veya bırakma işlemi gerçekleştirmesi ve algoritma parametrelerinin adaptif olması gibi özellikler içermektedir.

Gerçeklenen algoritma seçilen paralel kütüphane kullanılarak iteratif bir yaklaşım ile paralelleştirilmiştir. Paralelleştirme için MPI Kütüphanesi seçilmiştir. Bu kütüphanenin ardışıl çözümün de gerçeklendiği. NET ortamında ve C# dili ile birlikte kullanılabilmesini sağlayan MPI.NET kütüphanesi paralelleştirme rutinleri için kullanılmıştır.

Demetlenecek veriler ve demetleme ortamını oluşturan 2 boyutlu ortam ve atılacak toplam adım sayısı işlemciler arasında paylaştırılır. Her işlemci kendine atanan alanda ardışıl demetleme çözümünü çalıştırır ve çalışmasının sonlandığını ana işlemci olarak işaretlenen işlemciye bildirir. Bu işlemci ardışıl demetleme çözümünü bütün verileri ve 2 boyutlu alanın tamamını kullanarak tekrar çalıştırır. Demetleme algoritmasının daha iyi bir noktadan başlayabilmesi amaçlanmaktadır.

Yapılan çalışmada gerçeklenen iteratif paralelleştirme işleminin özellikle algoritma sonlanma zamanında iyileştirmeye sebep olup olmayacağı araştırılmıştır. Yapılan testlerde iki ve dört işlemi için seçilen veritabanları için başarım ölçüm sonuçları elde edilmiştir. Başarım ölçümü için oluşturulan demetlerin orijinal veritabanı ile ne kadar örtüştüğünü ve komşu demetlerden yeterince ayrık olup olmadığını gösteren indisler seçilmiş ve hesaplanmıştır. Ayrıca algoritma çalışma zamanı paralel program başarımının değerlendirilmesinde kullanılmıştır.

Algoritmanın daha iyi bir noktadan başlaması için ve karıcalar tarafından atılacak toplam adım sayısının diğer işlemcilerle paylaştırılması, algoritma çalışma

zamanlarında iyileşmeyi sağlamıştır. Haberleşme maliyetinin diğer paralelleştirme yöntemine göre çok düşük olması da algoritmanın avantajlarından biridir.

2. DEMETLEME

Demetleme probleminin değişik bilim dalları ile uğraşan araştırmacılar tarafından üzerinde düşünülen bir problem olması, veri analizi adımlarından biri olarak ne kadar önemli olduğunu gösterir. Sınıflar belli olmadığı için öğreticisiz bir yöntemdir [1]. Veri analizi yöntemleri iki dala ayrılır: Araştırma amaçlı ve onaylama amaçlı. Her iki dal için de önemli olan, ölçümlerin; önceden hazırlanan bir modele uygunluğunu inceleme veya analiz sırasında kümelere ayırma yöntemlerinden biri ile gruplandırılmasıdır.

Geçerli bir küme içindeki veriler birbirlerine farklı bir küme içindeki veriden daha çok benzemektedir.

Daha önce belirtildiği gibi demetleme öğreticisiz bir sınıflandırma şeklidir. Öğreticili sınıflandırmada etiketleri (sınıfları) önceden belli veriler mevcuttur. Problem bu veri grubuna yeni eklenecek olan henüz etiketlenmemiş olan verinin hangi sınıfa ekleneceğini yani hangi etiketle işaretleneceğini bulmaktır. Mevcut veriler sınıf etiketlerinin özelliklerinin öğrenilmesi için kullanılır. Öğreticisiz sınıflandırma (demetleme) işleminde ise eldeki bir grup etiketsiz verinin anlamlı kümelere ayrılması sağlanmaya çalışılır.

2.1 Demetleme Algoritmaları

Hiyerarşik ve kısmi olarak isimlendirilen 2 tipi vardır. Hiyerarşik olan algoritmalardan biri bu çalışmada da başarım ölçümünün yapılabilmesi için gerçeklenmiştir.

Hiyerarşik demetleme algoritmaları ile dendrogram adıyla bilinen küme hiyerarşisi veya kümeler ağacı oluşturur. Daha önce bulunmuş kümeleri kullanarak bir sonraki kümeyi bulur. Yığma yaklaşımında öncelikle her veri elemanının farklı bir kümede olduğu varsayılır. Belirlenen durma koşulu sağlanana kadar kümeler birleştirilerek sonuç elde edilir. Bölme yaklaşımında ise bütün veri elemanlarının aynı küme içinde olduğu varsayılır. Belirlenen durma koşulu sağlanana kadar kümeler bölünerek sonuç elde edilir.

Hiyerarşik algoritmalarda uzaklık ölçütünün belirlenmesi önemli bir adımdır. Veriler arasındaki uzaklık; verilerin özellik vektörünü oluşturan birimler arasındaki uzaklık

hesaplanarak bulunur. Kosinüs, Manhattan veya Öklit uzaklığı bu hesap için kullanılabilir. Bu algoritmanın iyi yönleri farklı veri özelliklerine kolayca uygulanabilmesi ve değişik uzaklık veya benzerlik tipleri ile çalışabilmesidir. Durma koşulunun belirsiz olması ve oluşturulan küme yapılarının iyileştirme amacıyla tekrar gözden geçirilmemesi bu algoritmanın kötü yönleri olarak sayılabilir.

Hiyerarşik algoritmalarda işleme tek veri elemanı içeren kümeler veya bütün veri elemanlarını içeren bir küme ile algoritmaya başlanır. İteratif olarak en uygun kümelerin bileştirilmesi veya bölünmesi ile durma koşulu sağlanana kadar uygun küme yapısı ede edilmeye çalışılır. Kümelerin birleştirme veya bölme işlemine uygun olması kümeler arası benzerlik veya farklılık değeri ile ilgilidir. Bu durum da kümelerin birbirine yakın veya benzer veri elemanlarından oluştuğu ön kabulünün sonucudur.

Alt kümelerin birleştirilmesi veya bölünmesi kararını verirken veri elemanları arasındaki uzaklık ölçütü kümeler arası uzaklığı ifade edecek şekilde genellenmelidir. Veri elemanları arası uzaklık kullanılarak türetilen bu ölçüte bağlama ölçütü denir. Bu ölçütün tipi hiyerarşik algoritmaları önemli ölçüde etkilemektedir. Kümeler arasında temel olarak en kısa uzaklık, ortalama uzaklık ve en büyük uzaklık ölçüleri kullanılmaktadır. Her iki küme içinde tüm veri elemanlarının ikili bütün kombinasyonlar için aralarındaki uzaklık hesaplanacaktır. Daha sonra seçilen ölçüte uygun şekilde iki küme arasındaki uzaklık hesaplanan ikili uzaklıkların en küçüğü, ortalaması veya en büyüğü olarak belirlenir [1,2].

Kısmi demetleme algoritmalarında ise bütün kümeler tek bir seferde, belirlenen kıstas optimize edilerek belirlenir. K-ortalama algoritması bu tip bir algoritmadır.

K-ortalama algoritması için akış aşağıda verilen şekilde olacaktır:

- Küme merkezleri olarak rasgele k tane nokta belirle
- Her noktayı kendine en yakın küme merkezine ata
- Yeni küme merkezlerini hesapla
- Belirlenen yakınsama kıstası sağlanana kadar tekrarla

Bu algoritmanın avantajı basit olması ve hızlı çalışması sebebiyle büyük veri kümelerinde de uygulanabilmesidir. Ancak her çalışmasında aynı sonucu vermez ve sonuçta oluşan kümeler başlangıçtaki rasgele atamalara bağlıdır.

Bulanık c-ortalama algoritmasında her noktanın kümelere ait olma derecesi vardır. Bulanık mantıkta olduğu gibi hiçbir nokta sadece tek bir kümeye tamamen ait değildir. Her x noktası için bu noktanın k kümesine ait olma derecesi $u_k(x)$ katsayısı

ile gösterilir ve genellikle bu katsayılar toplamı 1 ‘e eşittir. Başka bir deyişle $u_k(x)$ x noktasının k kümesine ait olma olasılığını göstermektedir.

K-ortalama gibi algoritmalar hızlı olmasına ve yerel optimuma çok çabuk yakınsamasına rağmen üzerinde çalışılan veri içindeki gürültüden etkilenir. Yoğunluk tabanlı algoritmalar ise gürültülü veri ile daha iyi sonuçlar üretebilmektedir. Ancak her veri için komşu alanlarda arama yapılması ve yoğunluk için eşik parametrenin belirlenmesi zordur. Demetleme nesnenin yoğunluğuna göre yapılır. Rasgele şekilde demetler oluşturabilir [1,2].

3. DOĞA ESİNLİ ALGORİTMALAR

Doğadan esinlenen, biyolojik materyaller ile oluşturulan ve doğal süreçler ile modellenen algoritmalardır. Bu tür algoritmaların örnekleri şunlardır:

- Yapay Sinir Ağları
- Evrimsel Algoritmalar
- Sürü Zekası
- Karınca Kolonisi Optimizasyonu

Doğa esinli algoritmalar arasında bu çalışma sırasında sürü zekası incelenecek ve gerçeklenecektir. Sürü zekası tek başına iken zeki olmayan bireylerin bir araya gelerek zekice davranışlar gösterebilmesi özelliğidir. Algoritmalar veya dağıtık problem çözme aygıtları böcek kolonileri veya diğer hayvan topluluklarından esinlenir. Arı, karınca veya termit toplulukları, balık, kurt, kuş sürüleri bu algoritmalara esin kaynağı olabilecek topluluklardır.

Sürü; bir kontrol merkezi, veri kaynağı veya bulunulan ortamın açık bir modelini içermez. Ancak ortamı algılayabilir ve ortamı değiştirebilir. Dağıtık programlama için uygun bir yapıdır. Her bireyin tek başına gösterebileceği davranış sınırlıdır. Hiçbir birey bütün koloninin durumunu öğrenemez.

3.1 Karınca Algoritmaları

Karınca kolonileri oluşturan bireylerin yapıları tek başlarına ele alındığında basittir. Buna rağmen grup olarak bir araya geldiklerinde merkezi bir koordinasyon olmaksızın yuva inşaatı, avlanma ve yiyecek temini gibi karmaşık işlemleri gerçekleştirebilirler [2].

Karıncalar arası iletişim stigmerjik olarak ifade edilen bir iletişim türüdür. Feromon denilen salgılar üzerinden gerçekleşir, dolaylıdır. Karıncalar doğrudan birbirleriyle haberleşmezler. Mesaj göndermek için ortama feromon bırakırken, mesaj almak için ortamdaki feromonu hissederler.

Karınca temelli yaklaşımlar Dorigo tarafından önerilen Karınca Kolonisi Optimizasyonu algoritması ile dikkate alınmaya başlanmıştır. Bu çalışmada karıncalar gezgin satıcı probleminin çözümü için kullanılmıştır. Her karınca graf

üzerinde yürürken geçtiği yollara feromon bırakır. Karıncanın bıraktığı feromon miktarı sabit olduğundan kısa yollarda feromon yoğunluğu daha fazla olacaktır. Arkadan gelen karınca gideceği yolu seçerken feromon yoğunluğu daha çok olan yolları daha büyük olasılıkla seçer. Bu çözüm permütasyon şeklinde ifade edilebilen problemlerin çözümünde oldukça iyi sonuçlar vermektedir [3].

Çok sayıda birey ve görev koordinasyonunda merkezi olmayan yaklaşım, karınca kolonilerinin paralelleştirebilme, kendini yönlendirebilme ve hata toleransı gibi modern bilgisayar sistemlerinde aranan özellikleri barındırması demektir [4].

4. KARINCA DEMETLEME ALGORİTMALARI

İlk karınca demetleme algoritması Deneubourg tarafından önerilmiştir [4,5]. Çalışmasında robotlar kullanarak nesneleri kümelere ayırdı. Bu modelde çalışma uzayında rasgele yürüyen karıncalar önlerine çıkan nesnelerden birini alarak veya üzerinde taşıdığı nesneyi bırakarak kümelere ayırma işlemini yapmaya çalışıyordu. Karıncaların sınırlı algılama yetenekleri vardı. Belli bir anda etraflarında bulunan nesnelerin taşıdıkları nesneye benzeyip benzemediğini algılayabiliyorlardı. Bu algıya dayanarak nesne alma/bırakma işlemlerini gerçekleştiriyorlardı.

Gutowitz [6], karıncaların çevrelerindeki dağılımı (entropi) algılamasını sağlayarak bu algoritmayı geliştirdi. Çalışma alanındaki entropi seviyesi nesnelerin varlığı veya yokluğu ile belirlenir. Tamamen dolu veya tamamen boş olan alanlar en düşük entropiye sahip olurken damalı (bir hücre dolu bir hücre boş) yapıda olan alan en yüksek entropiye sahiptir. Entropi seviyesi karıncaların nesne alma/bırakma kararı vermesi için kullanılır. Karıncalar düşük entropi seviyeli alanlarda nesne alma/bırakma işlemlerini geçekleştirmezler; böylece karıncalar demetleme işlemine katkısı bulunmayan işlemlerden kaçınmış olur.

Şu ana kadarki algoritmalarda soyut veri kümeleri ile çalışılmıştır. Nesneler arasında benzerlik değil nesnelerin aynı veya farklı olması demetleme işlemi için kullanılmıştır. Lumer ve Faieta [7] bu algoritmayı karmaşık veri tipleri ile de kullanılabilir hale getirmiştir. Nesneler sürekli bir benzerlik ölçütüne göre birbirlerinden ayrılırlar. Karıncalar birbirleri ile haberleşemezken bulundukları ortamdaki nesnelerin benzerliklerini algılayabilirler. Nesne alma veya bırakma işlemi nesneler arasındaki benzerlik ölçütünün de parametre olarak yer aldığı bir olasılık fonksiyonuna göre gerçekleştirilir.

Monmarche [8], grid üzerindeki bir hücrede birden fazla eleman olmasını sağlayan bir algoritma önermiştir. Her hücre üzerinde taşıdığı bir veya daha fazla nesne ile bir "küme" temsil eder. Aynı zamanda bir karınca da birden fazla nesne taşıyabilir. Karıncalar tarafından oluşturulan kümelerdeki bazı hataları gidermek için k-ortalama algoritması kullanılır.

Ramos [9], karıncanın grid üzerindeki hareket yöntemini değiştirdi. Daha önceki çalışmalarda karınca tamamen rasgele hareket ederken bu algoritmada diğer karıncalar tarafından bırakılan feromon izlerini takip ederek hareket eder. Grid

üzerinde boş olan alanlara bırakılan feromon zamanla uçacak ve karıncanın o bölgelere gitmesi ihtimali zayıflayacaktır.

Hartmann [10], sistemdeki benzerlik hesaplama fonksiyonu ve hareket yönteminde değişiklik önerdi. Her karınca çevresinde bulunan nesneleri girdi olarak alan ve yapılması gereken işlemi (nesne al/bırak) çıktı olarak üreten bir yapay sinir ağına sahiptir.

De Castro [11], Lumer ve Faieta tarafından önerilen algoritmada bazı iyileştirmeler yaptı. İyi küme yapıları oluşturulduğu halde karıncaların demetleme işlemine devam ettiğini ve bu sebeple oluşturulan optimum küme yapılarının zamanla bozulabildiğini gözlemledi. Bu sorunun çözümü için olasılık hesaplama fonksiyonlarına bazı parametreler ekledi. Bu parametreler demetleme işlemi devam ettikçe zamanla nesne alma/bırakma işlemlerinin daha az yapılmasını sağladı. Böylece oluşturulan iyi küme yapılarının bozulması engellenmeye çalışıldı.

Handl [12 – 15], Lumer ve Faieta tarafından önerilen algoritmada bazı değişiklikler önerdi. Bunlardan ilki nesne alma/bırakma kararı için kullanılan olasılık fonksiyonları ve nesneler arası benzerlik hesaplamak için kullanılan fonksiyon parametrelerinin adaptif olmasıdır. Bu parametreler için başlangıç değeri, maksimum değer ve artış miktarları algoritma başında tanımlanmıştır. Algoritma içinde başlangıç değerinden başlayarak ve algoritma durumuna göre artış miktarına göre değişerek uygun değerlere ulaşmaları sağlanır. Algoritmada yapılan diğer değişiklik ise karıncaların hızlarının homojen olmamasıdır. Karıncalar değişik hızlara sahiptir. Karıncalar grid üzerinde hızları kadar hücreyi bir adımda geçebilir. Hızlı karıncalar küme yapılarını oluştururken daha yavaş olan karıncalar kümelerin daha yoğun olmalarını sağlar.

4.1 Temel Yaklaşım

Daha önce belirtildiği gibi demetleme bir veritabanının alt kümelere bölünmesidir. Her alt küme kendi içinde belli karakteristik özellikleri paylaşacak, bir anlamda birbirine benzeyen birimlerden oluşacaktır. Verileri kümelere ayırmak için verinin bir veya daha fazla özelliği kullanılır.

Karınca demetlemesinin diğer demetleme algoritmalarından farkı işlemin verilerin özelliklerini barındıran n boyutlu uzayda değil, bundan tamamen bağımsız olan 2 boyutlu toroidal grid üzerinde yapılmasıdır.

Demetleme algoritmasında ilk adım verilerin 2 boyutlu grid üzerine rasgele yerleştirilmesidir. Daha sonra belli sayıda karınca rasgele 2 boyutlu grid üzerine

yerleştirilir. Grid üzerindeki her hücre sadece bir adet nesne ve/veya karınca olabilir. Her karınca sadece bir adet nesneyi taşıyabilir.

Her adımda karıncaların her biri eğer üzerinde yük varsa bulunduğu koordinata üzerindeki yükü bırakmaya çalışır. Karınca üzerinde yük yoksa ancak bulunduğu hücrede bir nesne varsa karınca bu nesneyi almaya çalışır. Karıncanın bir nesneyi alması veya bırakması olasılıkları nesnelerin özellikleri arasındaki uzaklığa yani nesnelerin birbirinden ne kadar farklı olduğuna bakılarak hesaplanır. Bu hesap alınması/bırakılması düşünülen nesne ile bu nesnenin bulunduğu hücrenin komşu hücrelerindeki nesneler için hesaplanır.

Karınca, üzerindeki nesneye benzer nesnelerin arasında ise karınca üzerindeki nesnenin bırakılma olasılığı daha yüksektir. Eğer bir nesne çevresindeki diğer nesnelerden farklı ise bu nesnenin bulunduğu hücreye gelen karınca tarafından alınması ihtimali yüksektir.

Nesne alan veya bırakan karınca bulunduğu hücreye bitişik olan hücrelerden birine geçer.

Bu kuralları takip ederek özellik uzayında birbirine yakın olan nesneler grid üzerinde de birbirlerine yakın olacak şekilde yerleştirilecektir. Belli rasgele adım sonunda temel küme yapıları ortaya çıkar. Başlangıçta bu kümelerde çok az elman bulunacaktır. Ancak küme içinde yer alabilecek nesnelerin bu kümelerin yanına bırakılma olasılığı grid üzerinde başka bir alana bırakılma olasılığında daha yüksek olacağından kümelerin eleman sayısı zamanla artacaktır.

4.2 Komşuluk Boyutu

Olasılık fonksiyonlarında kullanılan komşu sayısı algoritma performansında etkili olan parametrelerden biridir. Bu boyutun küçük olması algoritmanın yakınsama süresini kısaltır. Komşuluk boyutunun büyük olması ise daha kaliteli yani daha homojen kümelerin oluşmasını sağlar. Ancak bu durumda algoritma daha geç yakınsar.

4.3 Nesne Bırakma/Alma Olasılığı Hesaplama Fonksiyonları

Karınca demetleme algoritmalarının temel fonksiyonlarıdır. Nesnelerin birbirinden farklılığını gösteren $f(i)$ fonksiyonu ile hesaplanır. Bu fonksiyon sonuç kümelerinin şeklini ve yoğunluğunu etkiler.

4.4 Karınca Hareketi

Orijinal algoritmada her zaman diliminde karınca bulunduğu hücrenin bitişiğindeki hücrelerden birine geçer. Karıncanın nesne alması/bırakması için uygun olmayan alanlarda geçirdiği zamanı kısaltan bir hareket politikası ile algoritmanın yakınsama zamanı kısaltılabilir. Bazı hareket politikalarında karınca nesne aldığı/bıraktığı noktalara feromon bırakarak diğer karıncaları da bu noktalara yönlendirebilir [12 - 15].

5. DEMETLEME PROBLEMİ İÇİN ARDIŞIL KARINCA TEMELLİ DEMETLEME YAKLAŞIMI

Algoritma, .NET ortamında C# ile gerçeklenmiştir. Bölümün ilerleyen kısımlarında ayrıntıları verilecek algoritma için öncelikle kullanıcı ara yüzü olan bir uygulama geliştirilmiş böylece karıncaların hareketi incelenerek demetleme işlemi görsel olarak da izlenebilir hale gelmiştir. Algoritma başarımının istenen seviyeye gelmesinden sonra kullanıcı ara yüzü çıkarılarak uygulama testler için uygun hale getirilmiştir.

5.1 Temel İşlemler

Karınca ortamı toroidal yapıda olan 2 boyulu bir grid şeklindedir. Kümelenecek veri elemanları bir dizide tutulur ve grid üzerinde rasgele hücrelere yerleştirilir. Her hücre sadece bir nesne tutabilir. Bu durumda c_{ij} şeklinde ifade edilen grid hücreleri aşağıdaki şekilde ifade edilir.

$$c_{ij} = \begin{cases} d_x & x \text{ indisli nesneyi tutuyor} \\ -1 & \text{boş hücre} \end{cases} \quad \textbf{(5.1)}$$

Karınca kolonisi tarafından gerçekleştirilen üç önemli işlem vardır. Bunlar:

Nesne alma: Yükü olmayan karınca nesne barındıran bir hücreye giderse bu nesneyi alabilir.

Nesne bırakma: Yükü olan bir karınca herhangi bir anda taşıdığı nesneyi bırakmaya karar verebilir. Nesneyi bırakmaya karar veren karıncanın bulunduğu hücre boş ise karınca nesneyi bu hücreye bırakır. Karıncanın bulunduğu hücre boş değil ise en yakındaki boş hücre bulunur ve karınca yükünü bu boş hücreye bırakır.

Yürüme: Karınca komşu hücrelerinden birine hareket edebilir.

Temel karınca temelli demetleme algoritması ilk olarak şu aşamalardan geçer:

- Kümelenecek veriler 2 boyutlu ve toroidal grid üzerine rasgele dağıtılır.
- Her karınca rasgele bir nesne seçerek o nesneyi alır.
- Her karınca grid üzerinde rasgele seçilen bir nesneye yerleştirilir.

Başlangıç aşamasından sonra demetleme işlemini gerçekleştiren kısım başlar. Bu aşamada aşağıdaki adımlar gerçekleştirilir:

- Rasgele bir karınca seçilir.
- Karınca grid üzerinde adım boyu kadar bir adım atarak yeni bir hücreye geçer.
- Karınca olasılık hesaplarını yaparak yükünü bırakıp bırakmamaya karar verir. Karınca yükünü bırakabileceği bir hücre bulana kadar grid üzerinde hareket eder. Yükünü bırakmaya karar verdiği hücre boş ise yükünü olduğu yere bırakır. Hücrenin dolu olması durumunda ise bu hücreye en yakın boş hücreyi bulur ve nesneyi oraya bırakır. Karıncanın yükünü bırakması işlemini gösteren akış diyagramı Şekil 5.1 'de verilmiştir. Yükünü bırakan karınca taşıyabileceği yeni bir nesne aramaya başlar. Bu işlem için herhangi bir karınca tarafından taşınmayan nesnelerin bir listesi tutulur. Bu listeden rasgele bir nesne seçilir. Karınca bu nesnenin bunduğu yerde benzerlik fonksiyonunu hesaplayarak nesnenin bulunduğu hücreden alınıp alınmayacağına karar verir. Karınca taşıyabileceği bir nesne bulana kadar bu işlem devam eder [12 – 15]. Karıncanın yük alması işlemini gösteren akış diyagramı Şekil 5.2 'de verilmiştir.

Nesne alma/ bırakma kararı için kullanılan denklemler aşağıda verilmiştir.

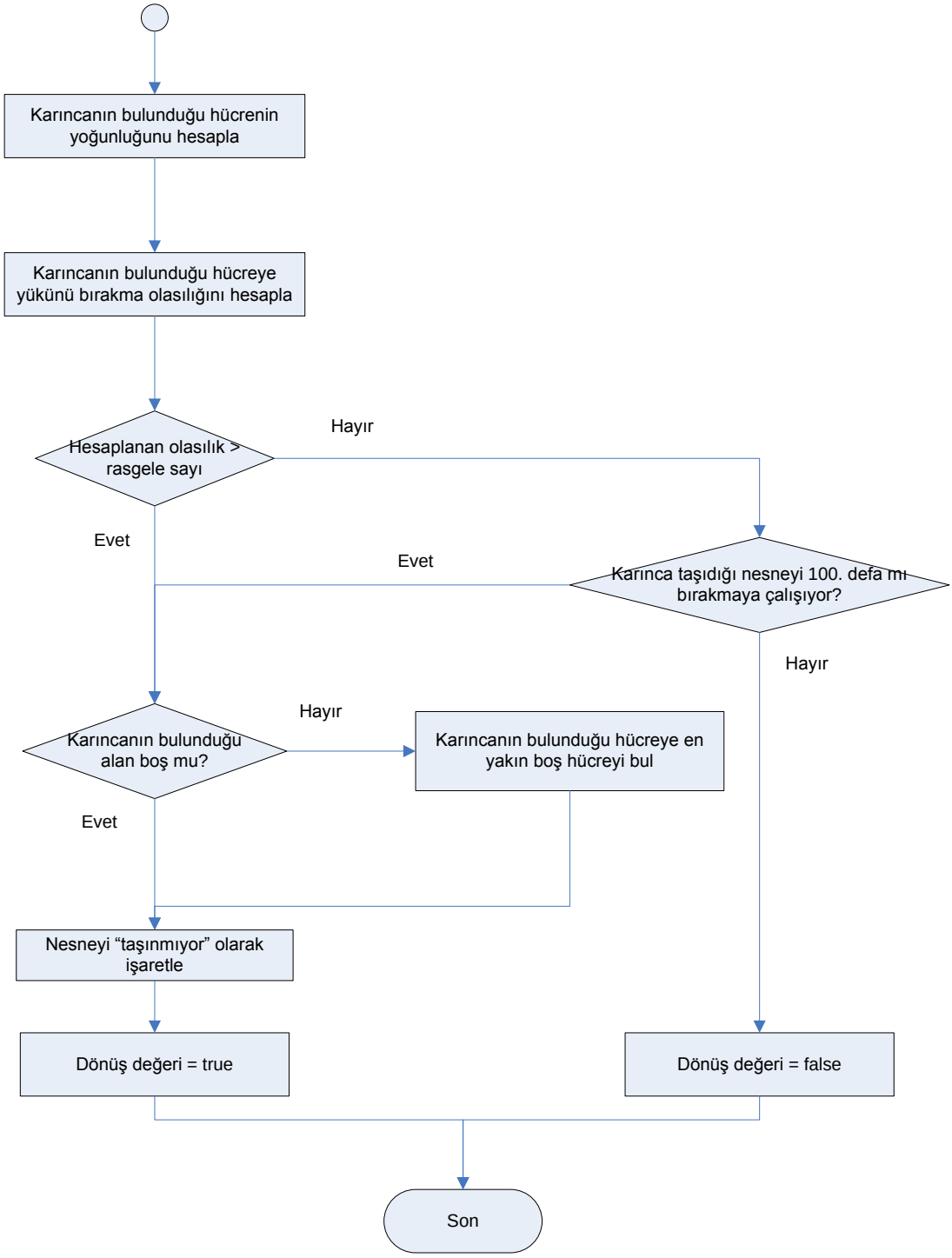
$$p_{pick}(i)=\left(\frac{k_p}{k_p+f(i)}\right)^2 \tag{5.2}$$

$$p_{drop}(i)=\left(\frac{f(i)}{k_d+f(i)}\right)^2 \tag{5.3}$$

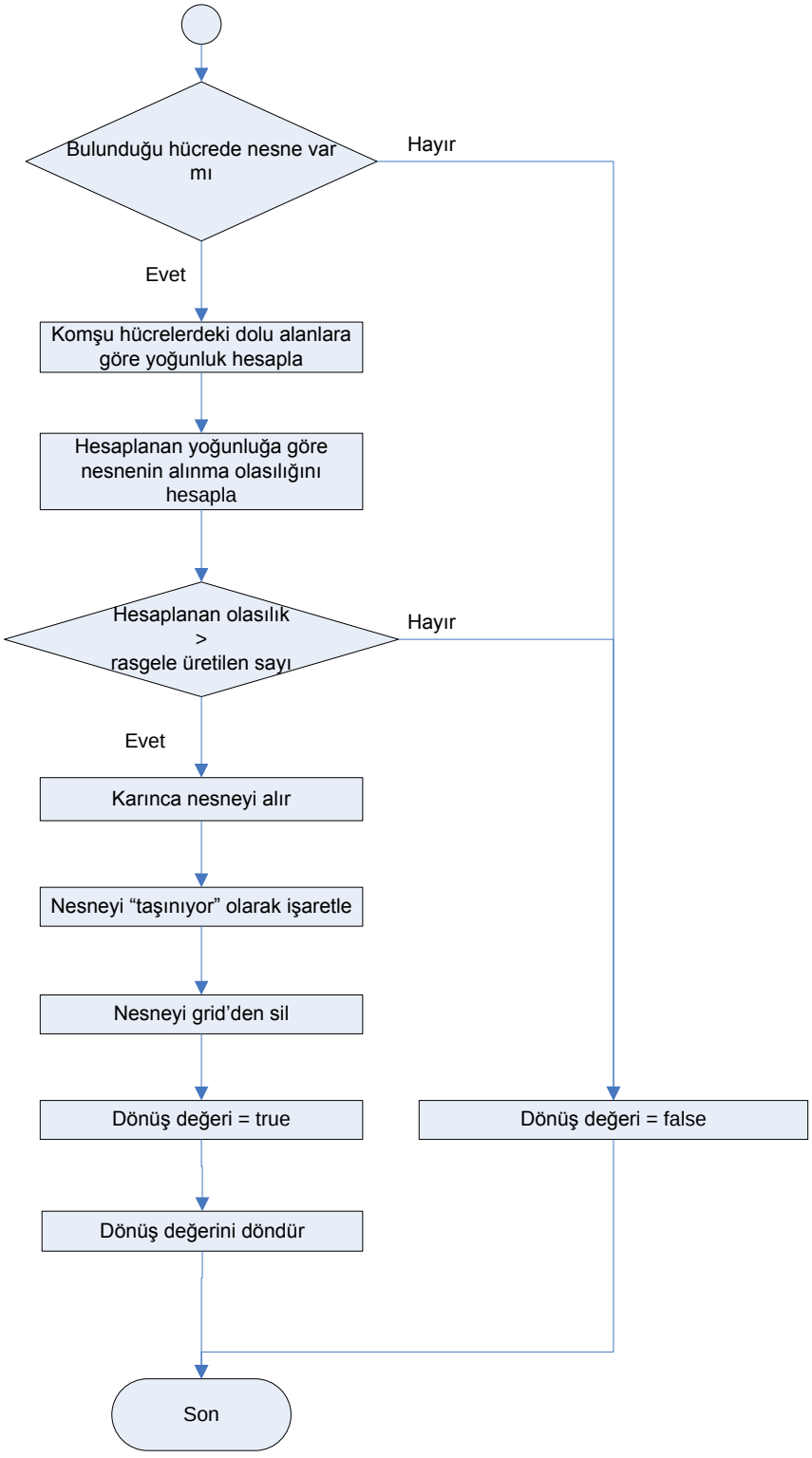
$$f(i)=\begin{cases}0 & if\ (f(i)<0 \vee \forall_j(1-\frac{d(i,j)}{\alpha\mu})<0)\\ \frac{1}{\sigma^2}\sum_{c_{ij}\neq -1}1-\frac{d(i,j)}{\alpha\mu} & aksihalde\end{cases} \tag{5.4}$$

$$\mu=\frac{2}{N(N-1)}\sum_{k=0}^{N}\sum_{l=0}^{k-1}d(k,l) \tag{5.5}$$

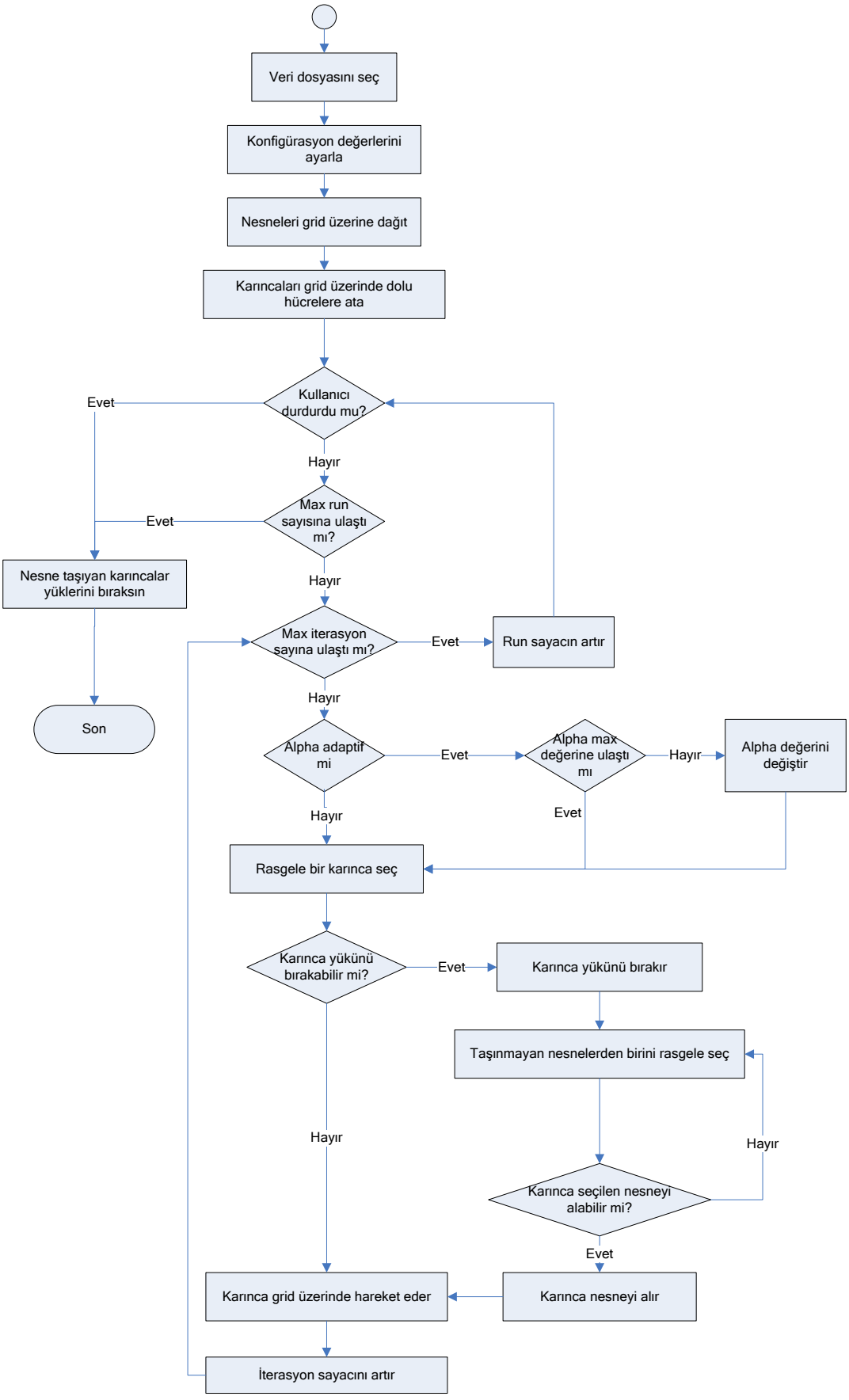
Algoritmanın karıncanın nesne alması/bırakması ve hareket etmesini işlemlerini de içeren tamamı için akış diyagramı Şekil 5.3 'te verilmiştir.



Şekil 5.1 **:** Nesne Bırakma İşlemi Akış Diyagramı



Şekil 5.2 : Nesne Alma İşlemi Akış Diyagramı



Şekil 5.3 **:** Genel Akış Diyagramı

5.1.1 Bellek

Algoritma performansını artırmak için her karıncaya sabit boyutlu dairesel bir bellek atanır. Karınca en son nesne bıraktığı n koordinatı belleğinde tutar.

5.1.2 Heterojen toplum

Topluluğu oluşturan karıncaların hızları aynı değildir. Bir karıncanın hızı $[0, V_{\max}]$ aralığında rasgele tam bir sayı olarak belirlenir. Yavaş karıncalar nesne seçme işleminde daha dikkatli davranırlar [12 – 15].

5.1.3 Adaptif strateji

Denklem 5.4 'teki α değeri kullanılarak uzaklıklar optimum seviyede ölçeklenebilir. α değerinin çok küçük seçilmesi grid üzerinde küme oluşumunu engelleyecektir. α değerinin çok büyük seçilmesi ise ayrı olması gereken kümelerin birleştirilmesine ve sonunda bütün nesnelerin aynı küme içinde gösterilmesine sebep olabilir. α 'nın uygun bir değer alması veritabanındaki nesnelerin ikili farklılıklarının dağılımına bağlıdır ve veritabanını dikkate almadan sabitlenmesi mümkün değildir.

Bu sebeple bu parametrenin değeri adaptif olarak belirlenir. Heterojen bir topluluk kullanılmıştır. Bütün karıncalar α parametresinin ilk değeri olarak 0.1 atar. Her karınca N_{active} adım attıktan sonra kendi parametre değerini gözden geçirir. Bu sürede karınca başarısız olan nesne bırakma işlemlerinin sayısını tutar [12 -15].

$$r_{fail} = \frac{N_{fail}}{N_{active}} \qquad \textbf{(5.6)}$$

N_{active} sayısı 100 olarak sabitlenmiştir. Parametrenin alabileceği değerler [0,1] aralığındadır.

$$\alpha \begin{cases} \alpha + 0.01 & if\ r_{fail} > 0.99 \\ \alpha - 0.01 & if\ r_{fail} <= 0.99 \end{cases} \qquad \textbf{(5.7)}$$

5.1.4 Ek kontroller

Bir karınca elindeki nesneyi bırakmada100 defadan fazla başarısız olursa bulunduğu hücrenin elindeki nesne için uygun olup olmadığına bakmadan nesneyi bırakır. Bu durum kümeler arasında bulunan aykırı özelliklere sahip nesneler sebep olur. Bu nesneler diğerlerinden çok farklı olduğu için karınca bu nesneyi diğerlerinin yanına bırakamaz.

5.1.5 Zaman yönetimi

Karınca grid üzerinde rasgele ilerler. Taşınacak nesne bulmak için grid üzerindeki boş alanlarda çok fazla zaman kaybeder. Bunu önlemek için herhangi bir karınca tarafından taşınmayan nesnelerin indisleri liste olarak tutulur. Elindeki nesneyi bırakan karınca indis listesinden rasgele birini seçerek bu nesnenin bulunduğu hücreye gider. Bu nesneyi almaya çalışır. İşlem başarılı olana kadar devam eder [12–15].

5.1.6 Parametreler

Karınca demetleme algoritmalarında algoritma başlangıcında ayarlanması gereken karınca sayısı, karıncaların bellek boyutu, grid boyutu gibi birçok parametre vardır.

Kümelenecek veri setinde N_{items} adet nesne oluğunu düşünelim. Bu durumda grid üzerinde N_{cells} adet hücre bulunacaktır. Hücre sayısı, karıncaların ellerindeki nesneleri daha hızlı bırakmasını sağlayacak şekilde atanmalıdır. Bunu sağlamak için denklem (5.8) ‘de verilen oran sağlanmalıdır. Bu oran deneysel olarak belirlenmiştir [12 – 17].

$$r_{occupied} = \frac{N_{items}}{N_{cells}} = \frac{1}{10} \qquad \textbf{(5.8)}$$

Grid yapısı kare şeklinde olacağından grid yüksekliği ve genişliği denklem (5.9a) ve (5.9b) ile hesaplanır [12 -17].

$$N_{cells} = 10 * N_{items} \qquad \textbf{(5.9a)}$$

$$\begin{aligned} Grid_{XSize} &= \sqrt{N_{cells}} \\ &= \sqrt{10 * N_{items}} \end{aligned} \qquad \textbf{(5.9b)}$$

Karıncanın adım sayısı bir harekette grid üzerinde mümkün olan herhangi bir noktaya ulaşmasını sağlayacak şekilde olmalıdır. Bunun için grid köşegen uzunluğu adım sayısı olarak denklem (5.10) ‘da görüldüğü gibi atanır [12 -17].

$$\begin{aligned} stepsize &= \left(\sqrt{10 * N_{items}}\right)^2 + \left(\sqrt{10 * N_{items}}\right)^2 \\ &= \sqrt{20 * N_{items}} \end{aligned} \qquad \textbf{(5.10)}$$

Algoritma için belirlenecek iterasyon sayısı kümelenecek veri sayısı ile değişmelidir. Lineer artış her bir grid hücresinin ziyaret edilme ortalamasını sabit tutacağından yeterli olacaktır [12 – 17].

$$iterations = 2000 * N_{items} \quad \textbf{(5.11)}$$

6. DEMETLEME PROBLEMİ İÇİN PARALEL KARINCA TEMELLİ DEMETLEME YAKLAŞIMI

Yüksek performans gerektiren hesaplamaların ve arama uzayı çok büyük olan optimizasyon problemlerinin çözümünde paralelleştirme çözümü kolaylaştırmak veya hızlandırmak için uygun bir yöntemdir. Topluluk yapısında olan doğa esinli algoritmalar paralelleştirme için çok uygundur. Topluluk, elde bulunan işlemcilere dağıtılır. Her işlemci elinde bulundurduğu toplum ile problem çözümü için gereken işlemleri yapar. Bu işlemler sırasında gerektiğinde diğer işlemcilerle mesajlar vasıtasıyla veri alıp gönderebileceği gibi toplumundaki bireyleri diğer işlemcilere de gönderebilir.

Permütasyon şeklinde ifade edilebilen optimizasyon problemlerinin çözümü için kullanılan karınca optimizasyonu algoritmasını da paralel çalıştırmak mümkündür. Örneğin gezgin satıcı problemi için her şehir için bir işlemci atanır. Karıncalardan her biri de şehirlerden birine rasgele atanır. Ana işlemci olarak işaretlenmiş olan işlemci feromon matrisinin güncellenmesinden sorumludur. Ayrıca uzaklık matrisi, başlangıç parametreleri gibi verilerin diğer işlemcilere dağıtılması görevini de yerine getirir. Gerekli parametreleri alan işlemciler buldukları yolu ve uzunluğunu master işlemciye göndererek feromon matrisinin güncellenmesini sağlar. Ancak haberleşme için geçen zaman çok fazladır. Bu durumu çözmek için kısmi asenkron, alternatif bir algoritma geliştirilmiştir. Bu algoritmada her işlemci ardışıl algoritmanın belli sayıda iterasyonunu gerçekleştirdikten sonra feromon matrisinin güncellenmesini sağlar. [18 – 20]. Bu algoritma haberleşme maliyetini düşürür ancak çözümün yerel optimuma takılmasına sebep olabilir.

Daha önce önerilen paralel çalışan karınca demetleme yönteminde paralel ve eşzamanlı güncelleme önerilmiştir [18]. Bu çalışmada kısıtlı da olsa karınca bulunduğu ortamı algılama, çevresindeki bilgiye erişme yeteneğine sahiptir. Deneubourg modelinde karınca kısıtlı zekaya yani belleğe sahip iken bu çalışmada karınca kendini çevreleyen bilgiyi kullanarak hareket eder.

Bu yaklaşımda da iki boyutlu ve kare şeklinde bir grid zemin olarak kullanılmıştır. Bir hücre komşuları bu hücreye bitişik olan 8 hücreden oluşur. Bütün karıncalar özdeş iken kümelenecek nesneler birkaç değişik türdedir. Bir hücrede farklı yönlere gitmeleri koşulu ile en fazla 8 adet karınca bulunabilir. Bir hücrede nesne varsa boş

olan karıncalardan en küçük indisli yöne gidecek olan karınca bu nesneyi alır. Nesne alan karınca boş bir hücreye ulaştığında nesnesini bırakır. Karınca hareketi difüzyon kuralına göre gerçekleşir. Karınca belli bir olasılık ile yönünü 45 derece çevirerek hareket eder. Karınca her adımda bir hücre ilerleyecektir. Karıncanın ilerlemesi yayılma aşaması olarak isimlendirilir. Nesne alan karınca bu durumunu gösteren bir bayrak taşımaya başlar. Bu bayrak mümkün olduğu kadar çabuk taşınan yükün bırakılmasını sağlayacaktır. Başlangıç durumunda karıncalar ve nesneler grid üzerine rasgele dağıtılmıştır.

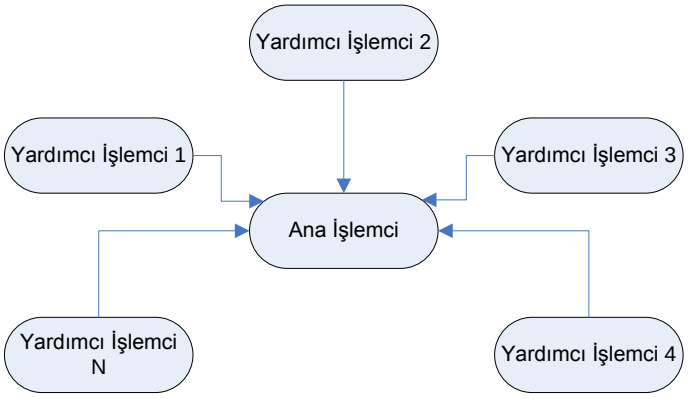
Paralelleştirme için yine aynı laboratuar tarafından paralel program yazımı amacıyla geliştirilen PELABS (Parallel Environment for Lattice Based Simulations) kütüphanesini kullanılmıştır. İki boyutlu grid bölümlere ayrılmış ve her bölüm bir işlemciye atanmıştır [19]. Her bölüm aynı zamanda komşularıyla olan sınırları da tutmaktadır. Böylece işlemci kendine ait olmayan bölüm için de komşuluk bilgisini tutabilecektir. Her zaman diliminde işlemciler birbirleriyle haberleşerek sınır verilirini günceller. Haberleşmenin en aza indirilmesi için grid in bölümlere ayrılması işlemi optimize edilmiştir. İşlemcilerin haberleşmesi kısmı yayılma safhasına denk getirilerek zaman tasarrufu sağlanmıştır.

Bütün denemeler için 290x290 boyutlu bir grid kullanılmıştır. Kümelenecek veri sayısı 1500 olarak belirlenmiştir. Veri sayısı ve karınca sayısı benzetim boyunca değişmemektedir. Kod optimize edilmediğinden sonuç olarak sadece belli bir durum için tek bilgisayar ve 8 bilgisayar kullanılması durumundaki benzetim zamanı verilmiştir. 8 adet Linux işletim sistemi kurulmuş olan Pentium 1 Ghz bilgisayar kullanılarak test gerçekleştirilmiş ve sonuç olarak 1000 karınca ile 10000 iterasyon 320 saniyede tamamlanmıştır. Aynı iterasyon sayısı ve karınca ile tek bir bilgisayar üzerinde işlemin tamamlanması 1600 saniye sürmektedir [19].

Diğer doğadan esinlenen ve topluluk yapısında olan algoritmaların paralelleştirilmesindeki yöntemin izlenmesi durumunda haberleşme maliyetinin çok yüksek olacağı öngörülmektedir. Bu yöntemde karıncalar ve 2 boyutlu grid alanı farklı işlemcilere dağıtılacak ve algoritma her işlemci tarafından komşu işlemcilerle sürekli haberleşerek çalıştırılacaktır. İşlemciler arasında nesneler ve karıncalar hareket edebilecektir. Ayrıca karıncanın nesne alma/bırakma kararı verirken hesapladığı yoğunluk fonksiyonu için komşu işlemcilerle sınır olan bölgelerin de içinde bulunduğu bir hesap yapılması durumunda komşu işlemcilerle veri alışverişi yapılması gerekecektir.

Oluşturulan mimaride ana işlemcinin ortada bulunduğu ve diğer işlemcilerin doğrudan bu işlemci ile haberleşebileceği, Şekil 6.1 ‘de görülen, yıldız şeklinde bir yapı oluşturulmuştur. Kullanılan bilgisayarlardan her biri P4 2 Ghz işlemciye ve 2

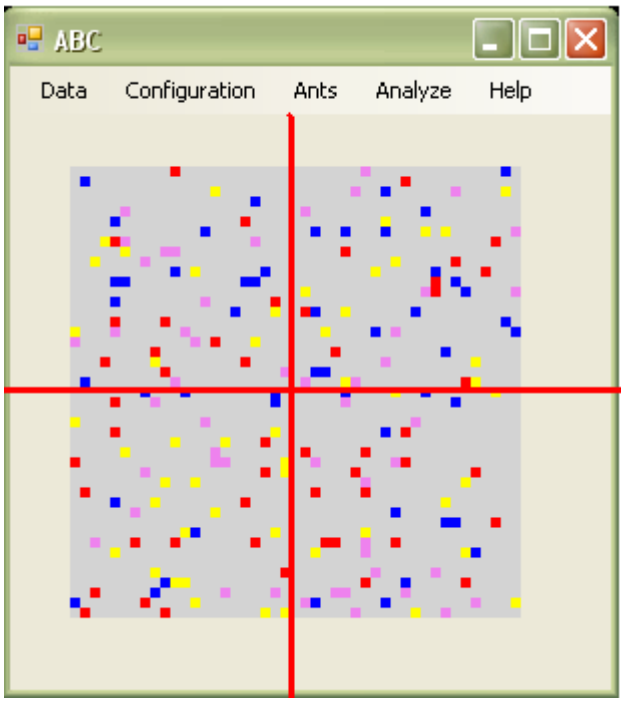
GB belleğe sahiptir. Bilgisayarlar aynı anahtar üzerinden ağ kartı ve UTP kablo ile birbirine bağlıdır. Ayrıca bilgisayarların her biri Windows XP Pro işletim sistemi ile çalışmaktadır. Bilgisayarlarda ayrıca .NET 2.0, MPI 1.2.5 kurulmuştur.



Şekil 6.1 : Kullanılan Bilgisayar Ağı Mimarisi

Öngörülen haberleşme maliyeti dikkate alınarak bu çalışmada paralelleştirme için iteratif bir yöntem izlenmiştir. İşlemcilerden biri ana işlemci olarak işaretlenecek ve işlem sonucunun elde edilmesi ve başarımın hesaplanmasından bu işlemci sorumlu olacaktır. Haberleşme maliyetinin biraz daha azaltmak için başlangıçta veri setinin bütün işlemcilerde bulunduğu varsayılmaktadır. 2 boyutlu grid ve veri kümesi elemanları işlemciler arasında eşit olarak paylaştırılır. Her işlemci veri kümesi içinde kendi payına düşen kısımdaki verileri alarak grid üzerinde kendine ait olan kısma rasgele dağıtır.

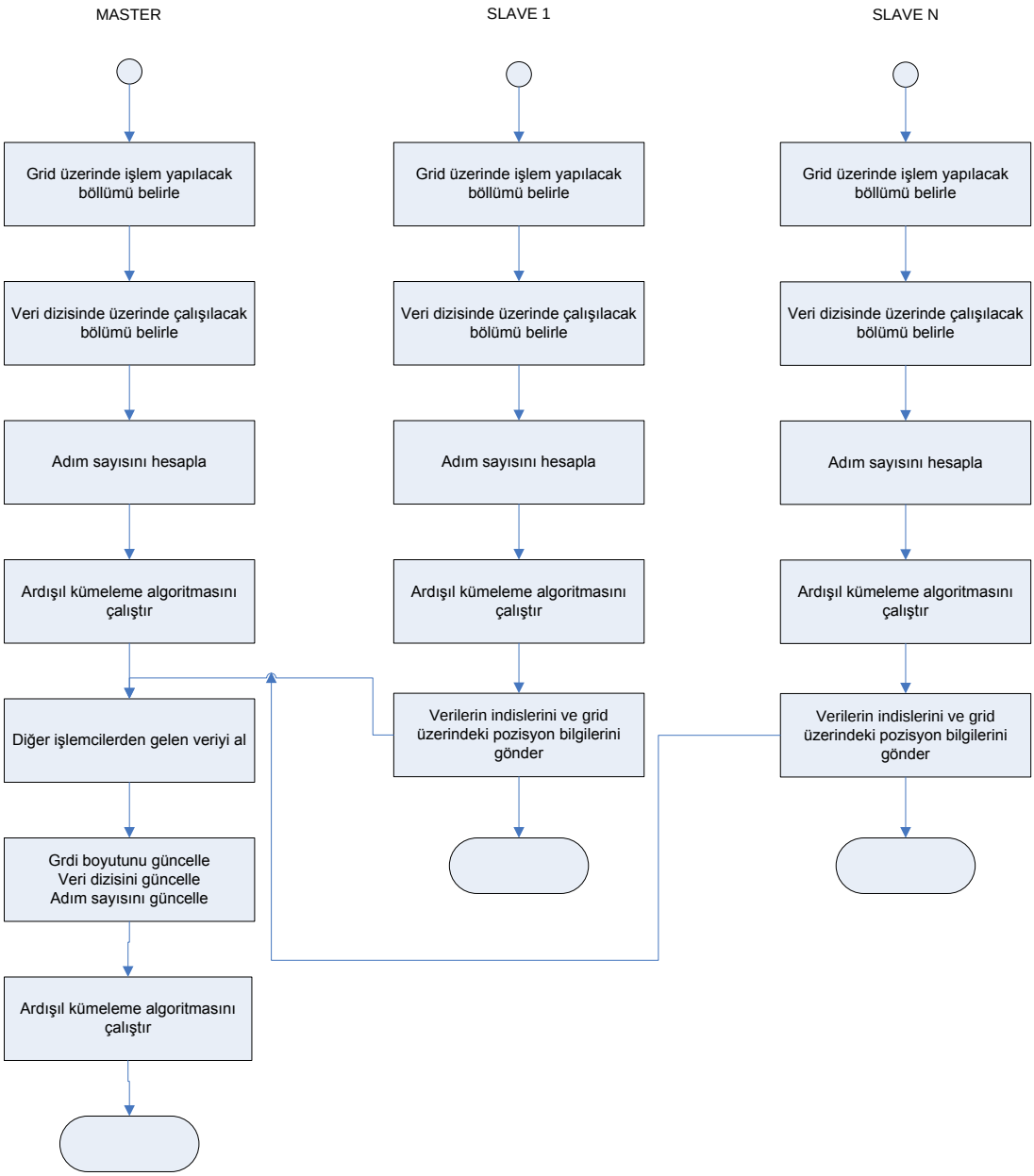
Nesnelerin işlemcilere paylaştırılması için örnek bir dağılım Şekil 6.1 ‘de verilmiştir. Burada verilerin 4 işlemciye paylaştırılması işlemi görülmektedir. Grid birbirine eşit 4 parçaya atanır. Bu parçalar (x,y) koordinatları ile ifade edilebilen hücrelerden oluşan grid üzerinde belirlenmiştir ve yine her biri (x,y) koordinatları ile ifade edilebilir. Her işlemci algoritma başlangıcında öncelikle gerçek grid üzerinde kendine düşen parça için minimum ve maksimum (x,y) koordinatlarını hesaplar. Böylece kendine ait yerel grid yapısını oluşturur.



Şekil 6.2 : Grid Yapısının İşlemcilere Bölünmesi Durumundaki Görünümü

Veri kümesi elemanlarının işlemcilere dağıtılması işlemi de grid yapısının dağıtılmasına benzer şekilde gerçekleşir. Her işlemci veritabanına erişebilir durumdadır. İşlemciler sırasıyla 0,..n-1 indislerine sahiptir. 0 indisli işlemci ana işlemci olarak işaretlenmiştir. Veritabanındaki veriler n parçaya bölünür ve her işlemci bir veritabanı parçasını alır.

Bütün işlemciler ardışıl algoritmayı çalıştırır. İşlemciler tarafından atılan toplam adım sayısı ardışıl algoritmada atılan adım sayısına eşittir. Master olmayan işlemciler iterasyon sayısı tamamlandığında kendine ait olan her veri için verinin indisini (veri kümesi içindeki sıra sayısı) ve grid üzerindeki konumunu bildiren veri yapısını ana işlemciye gönderir. Master işlemci kendi elindeki verileri ve grid yapısını diğer işlemcilerden gelen bilgilerle genişletir. Master işlemci ardışıl algoritmayı bütün grid üzerinde, veri kümesindeki bütün verileri kullanarak ve ardışıl algoritmanın adım sayısının kendini payına düşen kısmı için çalıştırır. İterasyon adımları tamamlandığında başarım analiz sonuçlarını oluşturur ve sonlanır. Her iki işlemci türü için de akış diyagramı Şekil 6.2 ‘de verilmiştir. Şekilde de görüleceği gibi tüm işlemciler kendi paylarına düşen grid parçası için kendi verileriyle küme yapısını oluşturmaya çalışırken paralel çalışırlar. Master işlemcinin diğer işlemcilerin oluşturduğu veri yapısını alması ile master dışındaki işlemciler çalışmayacaktır.



Şekil 6.3 : Paralel programın her iki işlemci türü için akış diyagramı

6.1 Gerçekleme Ortamı

Çözülmek istenen problemin zorluk derecesi arttıkça problem çözümü için harcanan zaman da artacaktır. Çözümü hızlandırmak için yapılan kod gözden geçirme veya derleyici parametreleri kullanılarak yapılan hızlandırmanın yeterli olmadığı noktada problemin bölünebilir olup olmadığına bakılmalıdır. Problem bölünmeye uygun ise birden fazla işlemcisi olan bir bilgisayarda çalıştırılabilecektir. Tek makinede birden fazla işlemci içeren bilgisayarlar çok yüksek bellek ve depolama kapasitesine sahiptir ve süper bilgisayar olarak adlandırılır. Ancak maliyeti oldukça yüksektir. Maliyet engelinden kurtulmak için problem bilgisayar ağı ortamında çözülebilir. Ağdaki her

bilgisayar birer işlemci gibi düşünülebilir. Ancak bu durumda da program parçalarının haberleşmelerinde dikkatli davranılmalıdır. Program parçaları arası haberleşmeye problem çözümü için harcanandan daha az zaman harcamalıdır

6.1.1 Mpi

Dağıtık bellekli sistemler için paralel program yazmak için kullanılabilen bir kütüphane standardıdır. Dağıtık bellekli, birden fazla düğümden oluşan sistemde düğümler arası iletişim için kullanılan standart bir protokoldür [20]. Bilgisayarlar arası iletişim mesajlaşmaya dayalıdır. C/C++ ve Fortran için sürümleri bulunmaktadır.

Bu kütüphanenin amacı birbirlerine mesaj göndererek haberleşen programlar için bir standart oluşturmaktır. Bu sebeple kütüphane ara yüzü mesaj gönderme için pratik, etkin, taşınabilir ve esnek bir ara yüz sunmalıdır

Mesaj gönderme/alma özellikle dağıtık bellekli paralel makinelerin bulunduğu sistemlerde yaygın olarak kullanılan bir modeldir. Pek çok çeşidi olmasına rağmen proseslerin mesajlarla haberleşmesindeki temel kavramlar bellidir. Son yıllarda bu modele uygun uygulamaların geliştirilmesindeki ilerlemeler ile mesajlar ile haberleşen sistemlerin kullanabileceği etkin ve taşınabilir olan birkaç değişik sistem geliştirilmiştir. Bu sebeple kütüphane metotları için sözdizimi standartlarını belirlemek gerekmiştir [20].

MPI tasarımında belli bir sistemin seçilip standart haline getirilmesi yerine, var olan birkaç mesajlaşma sistemi incelemiş ve bunların en etkili yönleri araştırılmıştır. MPI özellikle IBM T. J. Watson Research Center, Intel NX/2, Express, nCUBE Vertex, p4 ve PARMACS gibi çalışmalardan esinlenmiştir. Bunların yanında Zipcode, Chimp, PVM, Chameleon ve PICL ürünlerinden de yararlanılmıştır [20].

MPI(Message Passing Interface) standartlaştırma çalışmalarına geneli ABD ve Avrupa'da bulunan 40 firmadan 60 kişi katılmıştır. Üniversite araştırma grupları, devlet laboratuarları ile önemli paralel bilgisayar firmaları da bu çalışma içinde yer almıştır. Standartlaştırma çalışması dağıtık bellekli sistemlerde var olan mesajlaşma sistemlerinin incelenmesini amaçlayan bir çalıştay ile başlamıştır. Bu çalıştay sırasında kurulan araştırma gurubu standartlaştırma işlemine devam ederek 1992 yılında ilk taslak, 1993 yılında da gözden geçirilen sürüm MPI1 ismiyle tamamlandı. MPI1 mesajlaşma sisteminde olması gereken temel özellikleri bütün olarak belirtiyordu [20].

MPI tasarımının amaçlarını listeleyecek olursak:

- Etkin haberleşmeye desteklemek.

- Homojen olmayan ortamlarda kullanılacak uygulamaları desteklemek. .
- Kullanıcının bağlantı hataları ile uğraşmamasını, bu tip hatalar ile haberleşme altyapısının ilgilenmesini sağlamak.
- Var olan uygulamalardan çok farklı olmayan bir ara yüz tanımlamak ancak daha esnek kullanımı sağlayan ek özellikler sunmak.
- Haberleşme altyapısı ve sistem yazılımında çok fazla değişiklik gerektirmeden değişik platformlarda gerçeklenebilen bir ara yüz tanımlamak.
- Tanımlanan ara yüzün kullanılan dile bağlı olmamasını sağlamak.
- Ara yüz çok thread içeren uygulamalar tarafından da güvenli şekilde kullanılabilmeli.

Mesajlaşma modelinin çok tutulmasının sebebi büyük oranda taşınabilir olmasıdır. Bu modeli kullanan uygulamalar Dağıtık bellekli çok işlemcili bilgisayarlarda, ağ ile birbirine bağlı olan bilgisayarlarda ve bunların değişik birleşimleri üzerinde çalışabilmektedir. Ayrıca paylaşılan bellek kullanan uygulamaların geliştirilmesi de mümkündür.

6.1.2 Mpich

Ücretsiz ve taşınabilir olan bir MPI uygulamasıdır. Kararlı ve esnek bir uygulamadır. MPI için program geliştirme, çalıştırma ve gerekli kütüphaneleri sunan bir sistemdir. Unix işletim sistemi ve pek çok türevi ile Windows işletim sisteminde çalışabilmektedir. Ağ ile birbirine bağlanmış makinelerde çalışabildiği gibi tek bilgisayar üzerinde de paralel programlamayı simüle edebilmektedir [20].

MPI standartları oluşturulurken kullanım ve uygulama konusunda geri besleme üretme amaçlı olarak geliştirilmiştir. MPI sürümünün yayınlanması ile piyasadaki mesaj geçme sistemlerinin yerini alması için tasarlanmıştır. MIMD (Multiple Instruction Multiple Data) programlamayı ve homojen olmayan ortamları çalışabilmeyi desteklemektedir [20].

6.1.3 Mpi.net

.NET Microsoft tarafından sunulmuş olan ve pek çok servis ve uygulamayı destekleyen bir platformdur. Ortak Dil Çalışma Zamanı yapısı standart ve programlama dilinden bağımsız bir kod yorumlayıcısı ve CLI uyumlu programlama dilleri tarafından paylaşılan bir kütüphanedir. C# ise nesne yönelimli ve Microsoft tarafından özellikle. NET için önerilmiş bir dildir. Ancak bu ortamın yüksek

performanslı hesaplamalarda kullanılıp kullanılamayacağı hala açık değildir. Hindistan Üniversitesi Bilgisayar Bilimleri bölümünde yapılan çalışmada MPI kütüphanesinin C# ve Ortak Dil Çalışma Zamanı yapısı ile çalışması için ara yüzler tasarlanmış ve gerçeklemiştir [22]. Önerilen ara yüzler iki katman halindedir:

- Alt seviye: C++ ile MPI kullanımına benzer bir yaklaşımdır.
- Üst seviye: C# dilinin modern programlama dili özellikleri kullanılabilir (MPI.NET).

MPI paralel bilgisayarlarda mesajlaşma için belirlenmiş bir standarttır. Paralel bilgisayarlarda dağıtık bellek kullanımını basitleştirir.

.NET programlama ortamı ve C# C/C++ ile yazılmış olan kütüphanelere ara yüzler oluşturabilir. Bu ara yüzler kütüphane içindeki bir metodun C# programındaki bir metot gibi kullanılmasını sağlar. Kullanılan parametrelerin kütüphane içindeki veri tipleri ile .NET veri tipleri arasındaki dönüşümü .NET ortamı tarafından otomatik olarak gerçekleştirilir.

Kütüphane içindeki bir metoda ara yüz oluşturmak için öncelikle kütüphane içinde bu metot dışarıdan kullanılabilir hale getirilmelidir. Bu metodun C# tarafından çağrılabilmesi için her parametrenin. NET karşılığına çevrilmesi gerekmektedir. Sayı tipindeki parametrelerin farklı dillere çevrilmesi kolaydır. Ancak işaretçi tipindeki parametrelerin dönüşümü zordur. Bu dönüşümü gerçekleştirmek için

- İşaretçi boyutlu tamsayıların (IntPtr)
- C# içindeki işaretçi tipi
- Parametrenin MarshalAs özelliği

kullanılabilir. Bu yöntemlerin hepsinde kullanıcının müdahalesi gerekmektedir. Üst seviye ara yüzde bu işlemler otomatik olarak yapılmaktadır [22].

MPI metotları genellikle C dilinde void * olarak belirtilen tipte tanımlanmış veri tamponları kullanır. C# içinde bu tip işaretçilerin kullanımını sağlamak için unsafe kelimesi kullanılır. Bu kelimenin kullanımını sağlamak için derleyici bu kullanıma izin verecek şekilde ayarlanmalıdır. .NET nesneleri GC tarafından silinebilir. Nesnelerin MPI tarafından kullanılıyorken silinmesi engellenmelidir. Bu problemleri çözmek için CLR içindeki GCHandle sınıfı kullanılır. Bu sınıfın kullanım amacı nesnelerin bellekteki yerlerini sabitlemek ve bellekteki bu alanlara sadece güvenli C# kodları ile erişimi sağlamaktır [22].

MPI.NET C++ için nesne yönelimli MPI kütüphanesi olan OOMPI(Object Oriented MPI) ara yüzleri esas alınarak tasarlanmıştır [22]. OOMPI; MPI için temel olan port ve message sınıflarını içermektedir. Port; mesajın kaynağı veya hedefini temsil

etmektedir. Message ise gönderilecek veya alınacak mesajın başlangıç adresi, uzunluğu ve veri tipini içeren tanımlamadır. OOMPI kullanıcı tanımlı veri tiplerinin gönderilip alınması için basit yöntemler sunar. MPI.NET; OOMPI tarafından sunulan bu özellikleri korurken C# kütüphanelerinde kullanılan isimlendirme standartlarına daha uygun biçimde isimlendirir.

7. PARALEL VE ARDIŞIL ALGORİTMA PERFORMANSLARININ KARŞILAŞTIRILMASI

Karıncalar tarafından 2 boyutlu grid üzerinde görsel olarak oluşturulan kümeleme çözümünün belirlenmesi için yığma yaklaşımı ile çalışan bir algoritma kullanılacaktır. Her nesne öncelikle farklı bir küme içinde gibi düşünülür. Daha sonra her adımda belirlenen ölçüte göre birbirine en yakın olan 2 küme birleştirilir. Burada yakınlık nesnelerin grid üzerinde bulunduğu koordinatlar arasındaki uzaklık ile belirlenir. Bu döngü belirlenen bir durma koşulu gerçeklenene kadar devam eder.

Grid üzerinde C_i ve C_j şeklinde 2 küme verildiğini düşünelim. Bu kümelerin grid uzayındaki uzaklıları küme içindeki elemanların birbirine uzaklıklarının ortalaması olarak tanımlanır.

7.1 Karşılaştırılacak Başarım Kriterleri

7.1.1 F-ölçütü

Orijinal veritabanındaki her i kümesi n_i adet eleman içerir. Algoritma tarafından üretilen j kümesinde ise n_j adet eleman bulunur. n_{ij} i sınıfına ait olup j sınıfında da bulunan elemanların sayısıdır [24,25]. $n_i = n_j = n_{ij}$ olması durumunda yani bütün nesnelerin doğru etiketlenmesi durumunda en büyük değeri olan 1 değerini alır.

$$F = \sum_i \frac{n_i}{n} \max_j \{F(i, j)\} \quad \textbf{(7.1)}$$

$$F(i, j) = \frac{2 * p(i, j) * r(i, j)}{p(i, j) + r(i, j)} \quad \textbf{(7.1a)}$$

$$p(i, j) = \frac{n_{ij}}{n_j} \quad \textbf{(7.1b)}$$

$$r(i, j) = \frac{n_{ij}}{n_i} \quad \textbf{(7.1c)}$$

n veritabanının toplam eleman sayısıdır. F; [0,1] aralığındadır ve maksimize edilmelidir.

7.1.2 Rand indeksi

Kümeler arası benzerliği gösteren bir ölçüttür. Orijinal veritabanındaki U kümesi ile algoritma tarafından üretilen V kümesi için aşağıdaki denklem ile hesaplanır. [0,1] aralığında değerler alır ve maksimize edilmelidir [24,25]. Doğru etiketlenen nesne sayısının veritabanı boyutuna oranı ile hesaplanır. Bütün elemanların doğru etiketlenmesi durumunda en büyük değeri olan 1 değerini alır.

$$\frac{a+d}{a+b+c+d} \tag{7.2}$$

a: U ve V 'de aynı kümede olan eleman sayısı
b: U'da aynı V 'de farklı kümede olan eleman sayısı
b: U'da farklı V 'de aynı kümede olan eleman sayısı
a: U ve V 'de farklı kümede olan eleman sayısı

7.1.3 Küme içi değişim (Inner cluster variance)

Küme merkezinden sapmaların kareleri toplamını hesaplar. μ_c küme merkezini gösterirken $\delta(i,\mu_c)$ her küme içindeki her veri elemanının merkezden uzaklığını gösterir. İyi bir küme yapısının oluşturulmuş olması için bu rakamın minimize edilmesi gerekmektedir [24,25]. Küme elemanları mümkün olduğunca merkeze yakın olmalıdır.

$$I = \sum_{c\in C}\sum_{i\in c}\delta(i,\mu_c)^2 \tag{7.3}$$

7.1.4 Siluet indeksi

a(i); i nesnesinin kendisi ile aynı kümede bulunan diğer elemanlarla olan faklılığının ortalamasıdır. b(i) ise i nesnesinin kendisi ile aynı kümede bulunmayan diğer elemanlarla olan faklılığının en küçük değeridir. [-1,1] aralığında değerler alır [24]. Küme içindeki her noktanın aynı kümedeki noktalara ne kadar yakın, farklı kümelerdeki noktalara ne kadar uzak olduğunu gösteren bir indekstir. 1 e yakın olması kümenin komşu kümelerden tamamen soyutlanmış olduğunu ve küme içindeki noktaların birbirine çok yakın olduğunu gösterir. Bu indeks değerinin negatif olması istenmeyen bir durumdur. Çünkü bu durumda ilgili nokta komşu

kümelerdeki noktalardan birine kendi kümesindeki noktalardan daha yakındır. a(i)>b(i) durumu bu indeksin negatif hesaplanmasına sebep olur.

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \quad \textbf{(7.4)}$$

7.1.5 Hızlanma

Bunlardan ilki hızlanma ölçümüdür. (7.5) ile denklemi verilmiştir. Burada T(1) tek işlemci kullanılarak geçen zaman, T(N) ise N adet işlemci kullanılması durumunda geçen zamandır. İşlemci sayısındaki artış ile ne kadar hızlanma elde edildiği ölçülmektedir.

$$S(N) = \frac{T(1)}{T(N)} \quad \textbf{(7.5)}$$

7.1.6 Verimlilik

Verimlilik hesaplanması (7.6) numaralı denklem kullanılarak yapılacaktır. Burada amaç işleci başına elde edilen hızlanmayı hesaplamaktır.

$$E(N) = \frac{S(N)}{N} \quad \textbf{(7.6)}$$

7.1.7 Etkinlik

Etkinlik hesaplanası için (7.7) ile verilen denklem kullanılacaktır. İşlemci sayısına göre değişen etkinlik grafiğinde ekinlik değerinin en yüksek değer aldığı nokta optimum işlemci sayısının seçilmesinde referans olarak kullanılabilir.

$$n(N) = E(N) * S(N) \quad \textbf{(7.7)}$$

7.2 Testler İçin Kullanılan Veritabanları ve Özellikleri

Tablo 7.1 ‘de testler için kullanılan veritabanları için genel özellikler verilmektedir.

Square1 veritabanı sentetik bir veritabanıdır. Seçilen 4 merkez etrafında normal dağılan verilere sahip nesnelerden oluşan bir veritabanıdır. Her küme eşit sayıda eleman içerir. Küme içindeki her nesne 2 boyutlu özellik vektörü ile temsil edilir. Özellikler merkeze bağlı olarak hesaplanan reel sayı değerleri alır. Kümeler lineer ayrılabilirdir.

Wine veritabanı 3 küme ve 178 nesneden oluşur. Kümelerdeki eleman sayıları birbirinden farklıdır. Her nesne 13 boyutlu özellik vektörü ile tanımlanır. Özellikler reel sayı değerleri alır.

Iris veritabanı 3 küme ve 150 nesneden oluşur. Kümeler 50 ‘şer nesne içerir. Her nesne 4 boyutlu özellik vektörü ile temsil edilir. Vektör elemanları reel sayı değerleri almaktadır. Kümelerden ikisi lineer olarak ayrılabilir değildir.

Tablo 7.1 : Bir İşlemci ile Square1 Veritabanı için Elde Edilen Sonuçlar

İsim	**Küme Sayısı**	**Nesne sayısı**	**Her kümedeki nesne sayısı**	**Özellik Sayısı**	**Tip**
Iris	3	150	50, 50, 50	4	Sürekli
Wine	3	178	59, 71, 48	13	Sürekli
Square1	4	400	100,100,100,100	2	Sürekli

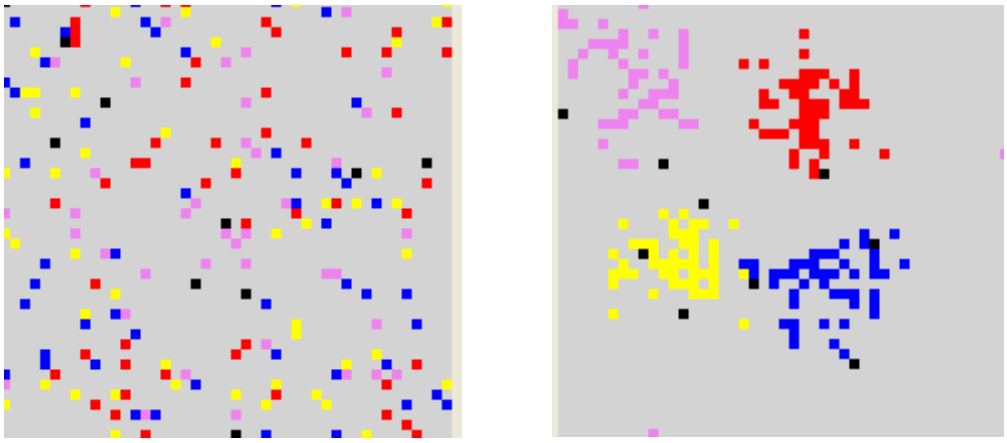
7.3 Belirlenen Performans Kriterleri İçin Elde Edilen Sonuçlar

Algoritma içinde karınca sayısı dışında bütün parametreler adaptiftir veya demetleme yapılacak veritabanı özelliklerine göre hesaplanmıştır. Bu sebeple 4 farklı karınca sayısı için testler yapılmıştır. Her veritabanı için farklı karınca sayısı ve işlemci sayısı ile bağımsız 40 koşum yapılarak elde edilen sonuçlar aşağıda raporlanmıştır.

Tablolarda verilen değerler ilgili kıstas için hesaplanan ortalama ve parantez içinde verilen ortalama hata değerleridir. Elde edilen sonuçların daha anlaşılır şekilde incelenebilmesi için farklı karınca sayılarına göre performans ölçümleri için grafikler çizilmiştir.

7.3.1 Square1 veritabanı için elde edilen sonuçlar

Square1 veri kümesi, karınca demetleme algoritmalarında sık kullanılan bir veritabanıdır. İki boyutludur ve kare şeklinde düzenlenmiş 4 kümeden oluşur. Seçilen dört merkez çevresinde normal dağılacak şekilde üretilmektedir. Şekil 7.1 ‘de bu veritabanı için örnek demetleme işlemi gösterilmektedir.



Şekil 7.1 : Square1 Verisi için Örnek Demetleme İşlemi

Square1 veritabanı kullanılarak yapılan 1,2 ve 4 işlemci için elde edilen demetleme başarımını gösteren sonuçlar sırasıyla Tablo 7.2, Tablo 7.3 ve Tablo 7.4 'te verilmiştir.

Tablo 7.2 : Bir İşlemci ile Square1 Veritabanı için Elde Edilen Sonuçlar

	1	5	10	20
F-Ölçütü	0,9189(0,0035)	0,9243(0,0034)	0,933(0,003)	0,9378(0,002)
Siluet	0,9017(0,0026)	0,9029(0,0033)	0,902(0,0034)	0,9102(0,0028)
Rand	0,9223(0,0028)	0,9322(0,0022)	0,9352(0,002)	0,9375(0,0027)
Variance	1,7459(0,0641)	1,7246(0,0569)	1,729(0,0572)	1,7053(0,062)
Time	1,6526(0,5969)	1,6487(0,602)	1,6376(0,5945)	1,6319(0,6093)

Tablo 7.3 : İki İşlemci ile Square1 Veritabanı için Elde Edilen Sonuçlar

F-Ölçütü	0,9213(0,0038)	0,9212(0,0037)	0,9311(0,0033)	0,9351(0,0021)
Siluet	0,902(0,0029)	0,9045(0,0034)	0,9105(0,0034)	0,9161(0,0028)
Rand	0,9212(0,0031)	0,9213(0,0022)	0,9218(0,002)	0,9242(0,0027)
Variance	1,6423(0,0705)	1,5423(0,0626)	1,5346(0,0629)	1,5345(0,0681)
Time	1,4954(0,5982)	1,4529(0,6032)	1,4349(0,5957)	1,4377(0,6106)

Tablo 7.4 : Dört İşlemci ile Square1 Veritabanı için Elde Edilen Sonuçlar

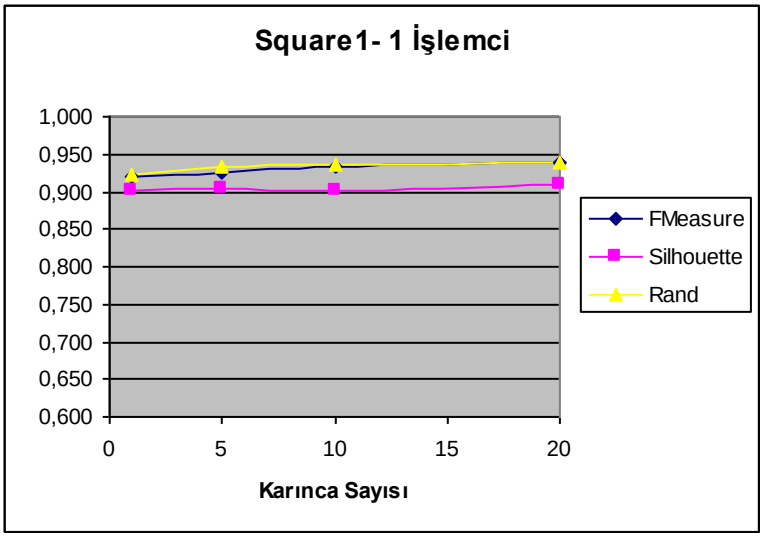
	1	5	10	20
F-Ölçütü	0,9459(0,0042)	0,9659(0,0041)	0,9686(0,0036)	0,9786(0,0024)
Siluet	0,9615(0,0032)	0,9798(0,0034)	0,979(0,0034)	0,9797(0,0027)
Rand	0,9687(0,0034)	0,9795(0,0024)	0,9797(0,0021)	0,9874(0,0027)
Variance	1,5058(0,0708)	1,4651(0,0628)	1,4577(0,0686)	1,4469(0,075)
Time	1,289(0,5994)	1,2879(0,6045)	1,2769(0,5969)	1,2757(0,6119)

Şekil 7.2 'de 1 işlemci için ortamda 1,5,10 ve 20 karınca olması durumunda daha önce anlatılan indekslerin aldığı değerler işaretlenmiştir. Karınca sayısındaki değişimin algoritma başarımını çok fazla etkilememesi beklenmektedir. Bu durum grafikte de açıkça görülmektedir. Karınca sayısının algoritma başarımında hafif bir

artışa sebep olduğu gözlenmektedir. Bu, karıncaların belleğe sahip olması ve belleğin karınca hareketi için kullanılması sebebiyledir. Karınca sayısı arttıkça grid üzerinde bulunan ve bellekte tutulan hücre sayısı artmaktadır. Bu da karıncanın grid yapısının daha büyük bir kısmını aklında tutarak buna göre hareket etmesini sağlamaktadır. Ancak karınca sayısındaki çok büyük artış karınca belleğinin güncelliğinin diğer karıncalar tarafından bozulmasına ve karınca hareketinin tamamen rasgele olmasına sebep olacağından başarımın düşmesine sebep olacaktır.

Şekil 7.2 'de f-ölçütü ve rand değerlerinin benzer davranış gösterdiği gözlenmektedir. Çünkü her iki parametre de bulunan küme yapısının gerçek küme yapısı ile ne kadar örtüştüğünü göstermektedir.

Şekil 7.2 ' de görülen siluet indeks değeri ile Şekil 7.5 'te görülen Variance1 değeri aynı küme içindeki elemanların birbirine yakın, diğer küme elemanlarına uzak olmasını ölçen değerlerdir. Siluet indeksindeki karınca sayısıyla gözlenen hafif artış variance1 değerinde hafif azalma olarak gözlenmektedir.

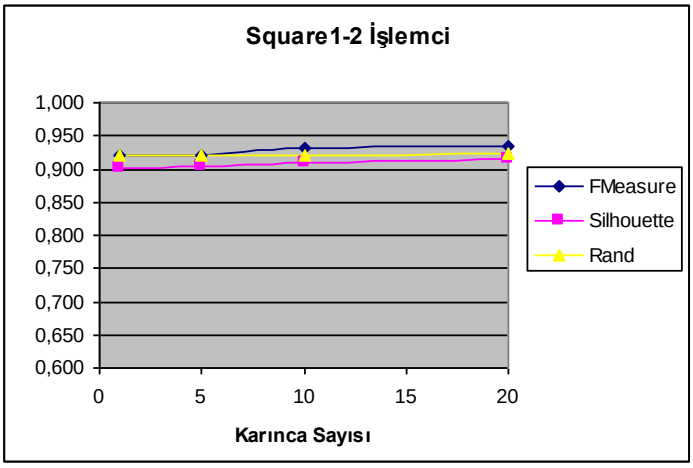


Şekil 7.2 : Square1 Veritabanı ve 1 İşlemci ile Elde Edilen Sonuçlar

Şekil 7.3 'te başarım ölçümleri 2 işlemci için grafik haline getirilmiştir. Şekilde karınca sayısındaki artışın performans kıstasları üzerindeki etkisi görülmektedir. Nesnelerin özellik sayısı az ve nesneler birbirinden ayrık olduğu için bu veritabanı üzerindeki demetleme işlemi basit bir problemdir. Üzerinde çalışılan veritabanı ile başarımın zaten çok yüksek olduğu göz önünde bulundurulmalıdır. F-ölçütü ve rand değerlerinin yine benzer davranış gösterdiği gözlenmektedir.

Şekil 7.3 'te siluet indeksinin karınca sayısıyla çok az arttığı gözlenmektedir. Bu artış küme içi elemanların birbirine uzaklığının diğer küme elemanlarına olan

uzaklıklarından daha az olduğunu göstermektedir. Bu durumda beklenen varyans değerinin de karınca sayısıyla beraber az da olsa azalmasıdır. Şekil 7.5 'te variance2 değerinde bu durum gözlenmektedir.

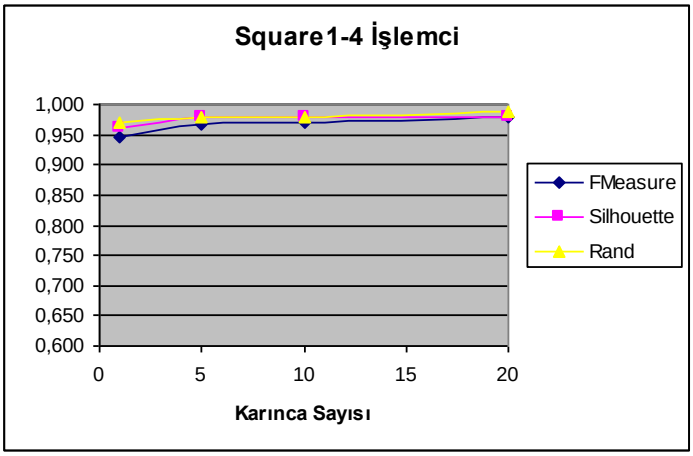


Şekil 7.3 : Square1 Veritabanı ve 2 İşlemci ile Elde Edilen Sonuçlar

Şekil 7.4 'te başarım ölçümleri 4 işlemci için grafik haline getirilmiştir. Şekilde karınca sayısındaki artışın performans kriterleri üzerindeki etkisi görülmektedir. Yine beklendiği gibi f-ölçütü ve rand değerleri benzer davranış göstermekte ve karınca sayısındaki artış ile bir miktar artış gözlenmektedir.

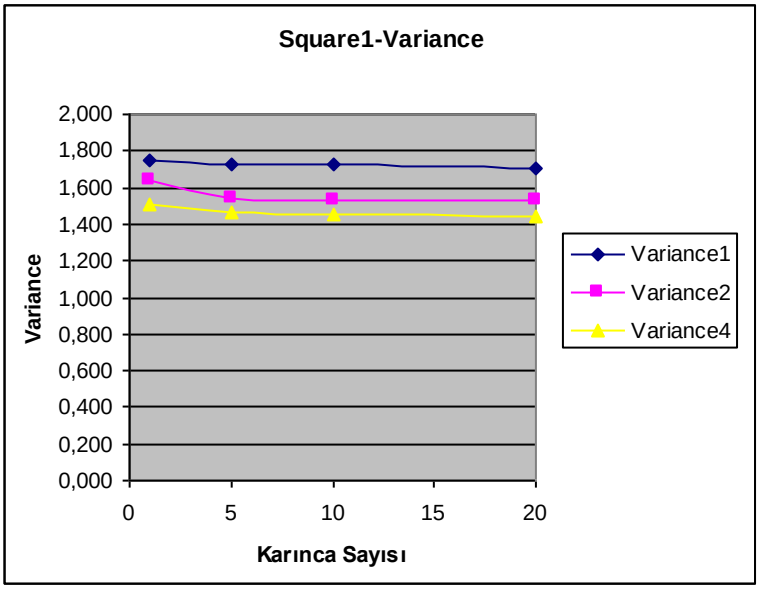
Şekil 7.4 'te görülen siluet değeri ile Şekil 7.5 'te görülen variance4 değeri beklendiği gibi birbirlerinin tersi davranış göstermektedir. Artan siluet değerine karşılık varyans değerinde azalma gözlenmektedir.

Ayrıca 4 işlemci ile çalışmada f- ölçütü, siluet, rand değerleri 1 ve 2 işlemciye göre daha yüksek değerler almaktadır. Yani daha iyi yapıdaki kümeler elde edilmiştir.



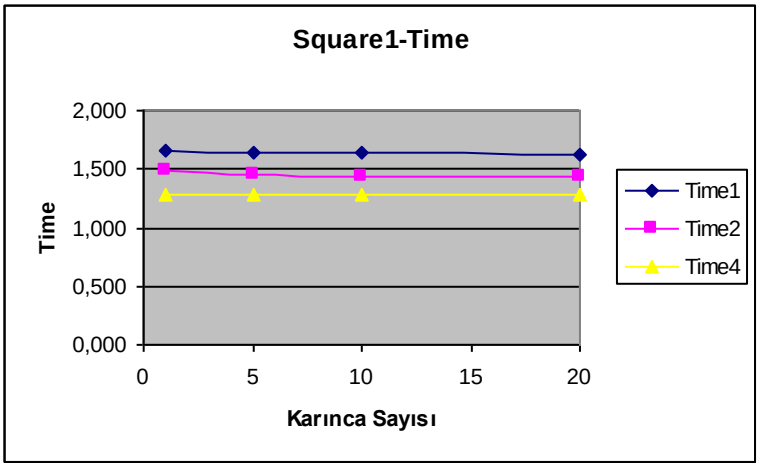
Şekil 7.4 : Square1 Veritabanı ve 4 İşlemci ile Elde Edilen Sonuçlar

Şekil 7.5 ‘te farklı karınca ve işlemci sayıları için varyans değerindeki değişim gözlenmektedir. İşlemci sayısındaki artışın varyans değerinde azalmayı sağladığı görülmektedir.



Şekil 7.5 **:** Square1 Veritabanı ve Farklı İşlemci Sayıları ile Varyans Değişimi

Şekil 7.6 ‘da farklı karınca ve işlemci sayıları için algoritma çalışma zamanındaki değişim gözlenmektedir. İşlemci sayısındaki artışın zaman değerinde azalmayı sağladığı görülmektedir. Ana işlemci olarak işaretlenen işlemci tarafından atılacak adım sayısı diğer işlemcilerle paylaşıldığından zaman değerinin azalması zaten beklenen bir durumdur.

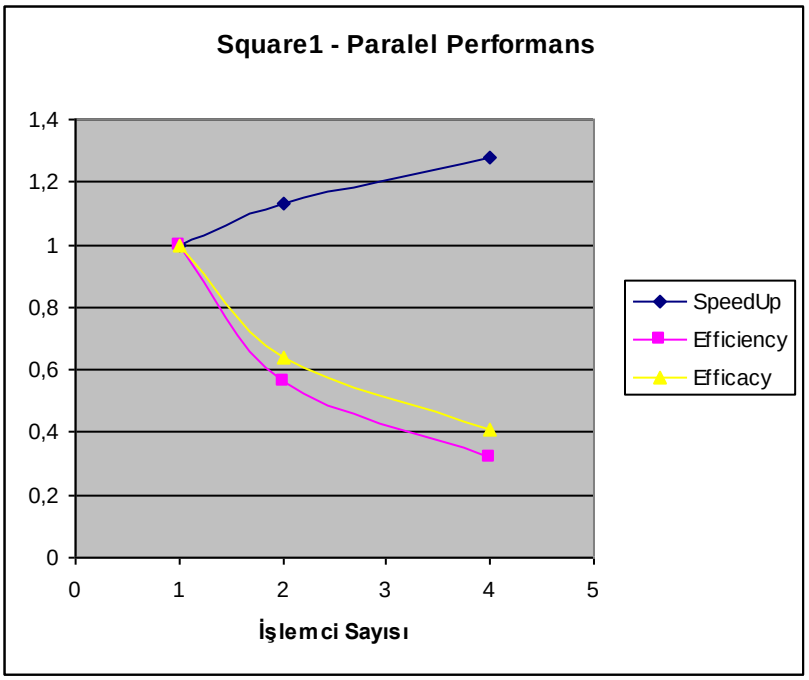


Şekil 7.6 **:** Square1 Veritabanı ve Farklı İşlemci Sayıları ile Zaman Değişimi

Şekil 7.7 ‘de hızlanma, verimlilik ve etkinlik değerlerinin farklı işlemci sayıları için değişimi verilmektedir.

2 işlemci kullanılması durumunda yaklaşık 1,13 kat daha hızlı sonuç elde edilmektedir. İşlemci başına ise 0,56 kat hızlanma sağlanmaktadır. 4 işlemci kullanıldığında ise ortalama 1,3 kat hızlanma sağlanmaktadır. İşlemci başına elde edilen hızlanma ise 0,3 civarındadır.

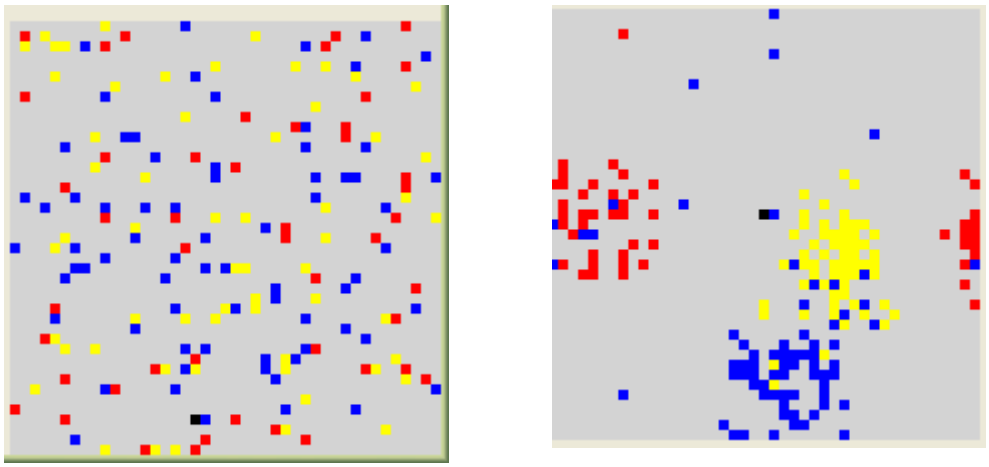
Verimlilik grafiğinde tepe ilk tepe noktası optimum işlemci sayısının belirlenmesinde kullanılan bir parametredir. Şekil 7.7 ' de bu tepe noktasının 1 işlemci ile yakalandığı gözlenmektedir. Ancak bu tespitin yapılması için sadece algoritma zamanındaki değişimin esas alındığı unutulmamalıdır. Daha önceki incelemelerde 4 işlemci kullanıldığında algoritma başarımında az da olsa artış gözlendiği belirtilmişti. Bu durumda algoritma başarımının iyileşmesi ve algoritma zamanında beklendiği kadar olmasa da hızlanmanın sağlanmış olması sebebiyle 4 işlemci ile problemin çözülmesi uygun sayılabilmektedir.



Şekil 7.7 : Square1 Veritabanı ve 1,2 ve 4 İşlemci ile Elde Edilen Paralel Program Performans Ölçümleri

7.3.2 Wine veritabanı için elde edilen sonuçlar

178 elemandan oluşan veritabanı 3 küme içermektedir. Kümelerde sırası ile 59,71,48 eleman bulunmaktadır. Elemanlardan her biri 13 boyutlu özellik vektörü ile tanımlanmaktadır. Özellikler sürekli değer almaktadır. Bu veritabanı ile işlemler yapılırken veriler arası uzaklık hesaplamak için kosinüs uzaklığı kullanılmıştır. Bu veritabanı için örnek bir kümelenme işlemi Şekil 7.8 ' de görülmektedir.



Şekil 7.8 : Wine Verisi için Örnek Demetleme İşlemi

Wine veritabanı kullanılarak yapılan 1,2 ve 4 işlemci için elde edilen demetleme başarımını gösteren sonuçlar sırasıyla Tablo 7.5, Tablo 7.6 ve Tablo 7.7 'de verilmiştir.

Tablo 7.5 : Bir İşlemci İle Wine Veritabanı İçin Elde Edilen Sonuçlar.

	1	5	10	20
F-Ölçütü	0,7059(0,0186)	0,7672(0,0533)	0,7944(0,0165)	0,7685(0,0232)
Siluet	0,7819(0,0344)	0,8144(0,0376)	0,8923(0,0405)	0,8123(0,0307)
Rand	0,7399(0,0177)	0,7413(0,0165)	0,7543(0,013)	0,7262(0,0181)
Variance	3,4431(0,456)	2,6851(0,4471)	2,3398(0,4655)	2,515(0,2531)
Time	5,4494(0,3595)	5,1949(0,3842)	4,9401(0,4352)	5,2644(0,3171)

Tablo 7.6 : İki İşlemci ile Wine Veritabanı için Elde Edilen Sonuçlar.

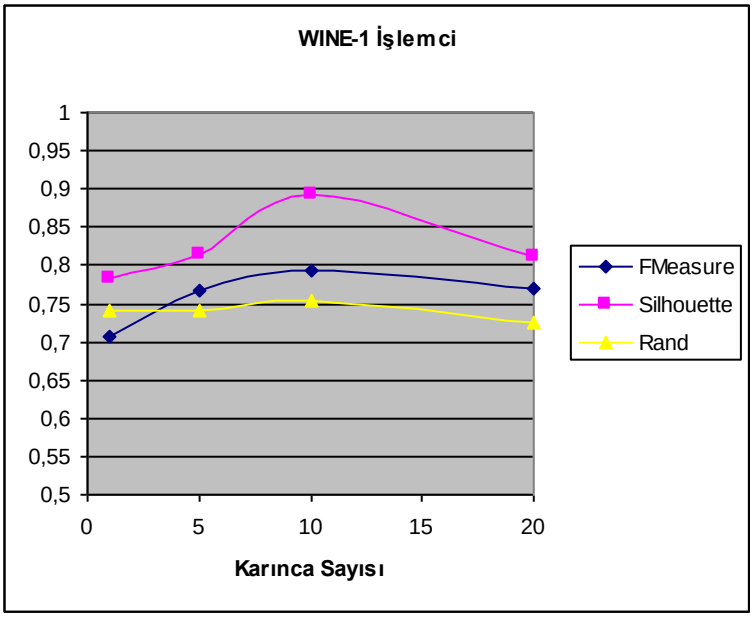
	1	5	10	20
F-Ölçütü	0,724(0,0256)	0,7423(0,0711)	0,8252(0,022)	0,7377(0,0309)
Siluet	0,783(0,0459)	0,7819(0,0501)	0,8138(0,054)	0,8065(0,0483)
Rand	0,724(0,0236)	0,7321(0,022)	0,763(0,0174)	0,7454(0,0241)
Variance	3,3431(0,7691)	2,8512(0,7267)	2,4834(0,6206)	2,5149(0,5965)
Time	4,8595(0,4699)	4,6559(0,5031)	4,2521(0,4745)	4,5115(0,7452)

Tablo 7.7 : Dört İşlemci ile Wine Veritabanı için Elde Edilen Sonuçlar.

	1	5	10	20
F-Ölçütü	0,7017(0,0523)	0,7255(0,0404)	0,7825(0,0353)	0,7638(0,0361)
Siluet	0,7444(0,0231)	0,7566(0,021)	0,8031(0,0248)	0,7223(0,0307)
Rand	0,7262(0,0042)	0,7671(0,0139)	0,7862(0,008)	0,7454(0,0533)
Variance	3,7311(0,4069)	3,4685(0,3558)	3,3398(0,3633)	3,515(0,3388)
Time	4,2087(0,4595)	4,1296(0,5218)	3,9483(0,5805)	4,1483(0,5097)

Şekil 7.9 ‘da farklı karınca sayıları için performans ölçüm değerlerindeki değişim görülmektedir. Bu veritabanında karınca sayısı artışının algoritma performansına etkisi gözlenebilmektedir. 10 civarında karınca sayısının bu veritabanı için optimum sayı olduğu görülmektedir.

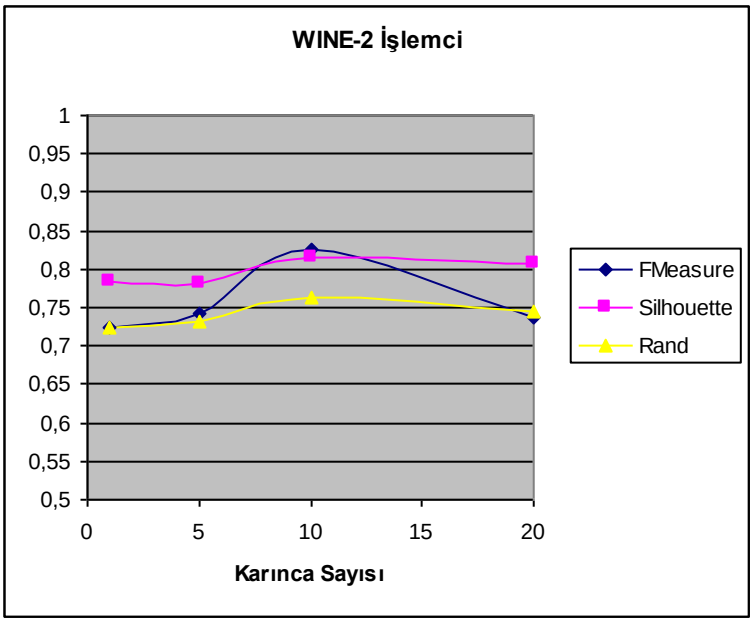
Karınca sayısının 1 ‘den 5 ‘ çıkarılması siluet, f- ölçütü ve rand değerlerinde artışa sebep olmuştur. Daha çok karınca grid üzerinde daha çok alanın bellekte tutulması ve karıncaların taşıdığı nesnelerin daha doğru yerlere bırakılması demektir. Bu sebeple algoritma başarımının artması beklenen bir durumdur. Karınca sayısının 10 olması durumunda ölçümü yapılan kriterlerin tümü en büyük değerlerini almışlardır. Karınca sayısının 20 ‘ye çıkarılması ölçümü yapılan kriterlerin değerlerinde azalmaya sebep olmuştur. Optimum bir seviyeden sonra karınca sayısındaki artış daha çok karıncanın grid üzerinde hareket etmesi ve karınca belleklerinin diğer karıncalar tarafından bozulmasına sebep olmaktadır.



Şekil 7.9 : Wine Veritabanı ve 1 İşlemci ile Elde Edilen Sonuçlar

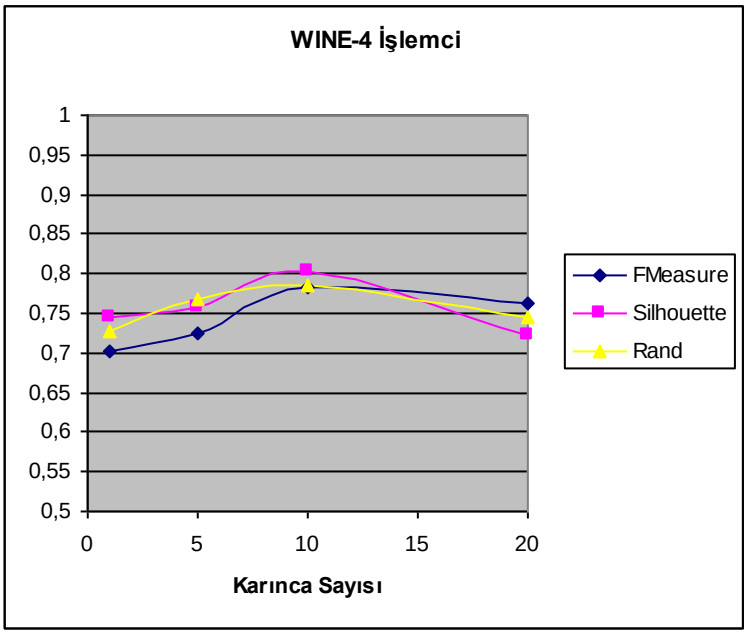
Şekil 7.10 ‘da farklı karınca sayıları ve 2 işlemci için performans ölçüm değerlerindeki değişim görülmektedir. İşlemci sayısının artışı performans ölçümlerinde artışa sebep olmamıştır. Karınca sayısının 1 ‘ den 5 ‘e çıkarılması siluet değerinde değişime sebep olmamıştır. Rand ve f- ölçütü değerlerinde ise bir miktar artışa sebep olmuştur. Yine 10 karınca sayısı ile ölçümlerin en büyük değerlerine ulaştığı, karınca sayısının tekrar artırılması ile ölçümlerin değerlerinin azaldığı görülmektedir. Siluet 1 işlemciye göre daha düşük değerler almaktadır.2 işlemci ile oluşturulan kümelerin 1 işlemciye göre daha geniş çapta ve/veya diğer kümelere daha yakın olduğu anlamına gelmektedir. Ancak etiketleme performansına

bakıldığında f-ölçütü ve rand ölçüm değerlerinin çok değişmediği gözlenmektedir. Bu durum da aynı kümedeki nesnelerin grid üzerinde daha dağınık olduğunu gösterir ancak diğer kümelerle iç içe geçmedikleri için etiketleme işleminde 1 işlemci ile aynı başarım gözlenmiştir. Şekil 7.12 'de elde edilen varyans değerlerine bakacak olursak 2 işlemci ile elde edilen varyans değerlerinin 1 işlemci ile elde edilenden daha büyük değerler aldığı gözlenmektedir. Bu durum da siluet değerinde elde edilen davranış ile tutarlıdır.



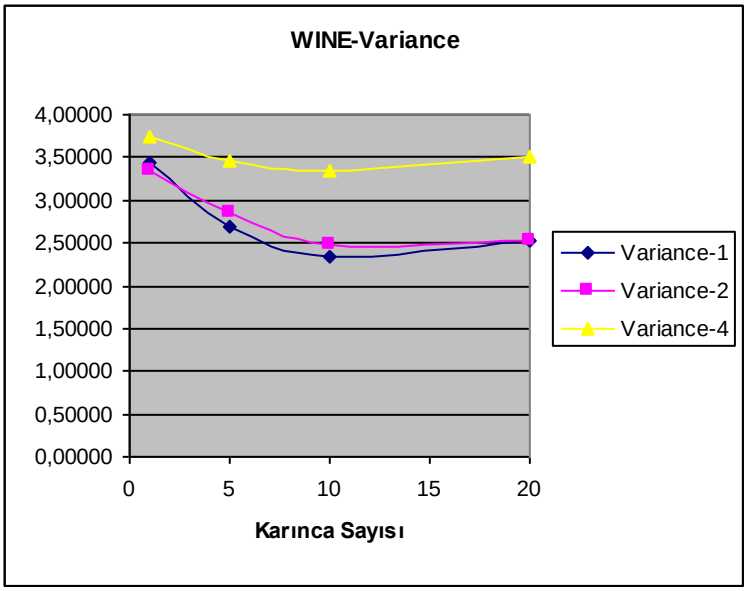
Şekil 7.10 : Wine Veritabanı ve 2 İşlemci ile Elde Edilen Sonuçlar

Şekil 7.11 'de farklı karınca sayıları ve 4 işlemci için performans ölçüm değerlerindeki değişim görülmektedir. İşlemci sayısının artışı performans ölçümlerinde hafif bir düşüşe sebep olmuştur. Rand ve f-ölçütü değerleri yine 10 civarındaki karınca sayısı için en yüksek değerini almıştır. Ancak 1 ve 2 işlemci için elde edilen değerlerden daha düşük değerler elde edilmiştir. Siluet değeri de 10 civarında karınca sayısı için en yüksek değerine ulaşırken tüm işlemciler için yapılan ölçümlerde en düşük değerlerini almıştır. Benzer şekilde Şekil 7.12 'de de varyans değerinin 4 işlemci için yapılan ölçümlerde diğerlerine göre daha yüksek değerler aldığı gözlenmektedir. Bu durum 4 işlemci ile daha dağınık kümelerin oluşturulduğunu ve kümelerin birbirinden tam olarak ayrılamadığını gösterir.



Şekil 7.11 : Wine Veritabanı ve 4 İşlemci ile Elde Edilen Sonuçlar

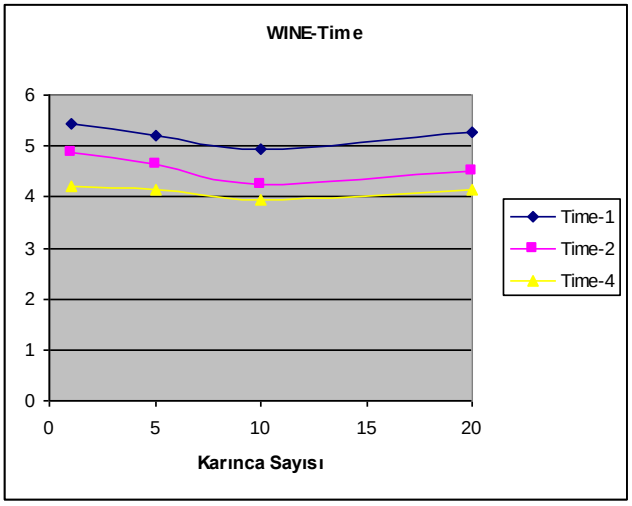
Şekil 7.12 'de farklı karınca ve işlemci sayıları için varyans değerindeki değişim gözlenmektedir. İşlemci sayısındaki artışın varyans değerlerinde artışa sebep olduğu gözlenmektedir. Ancak her durumda 10 civarında karınca sayısı için varyans değerinin en düşük değerini aldığı gözlenmektedir.



Şekil 7.12 : Wine Veritabanı ve Farklı İşlemci Sayıları ile Varyans Değişimi

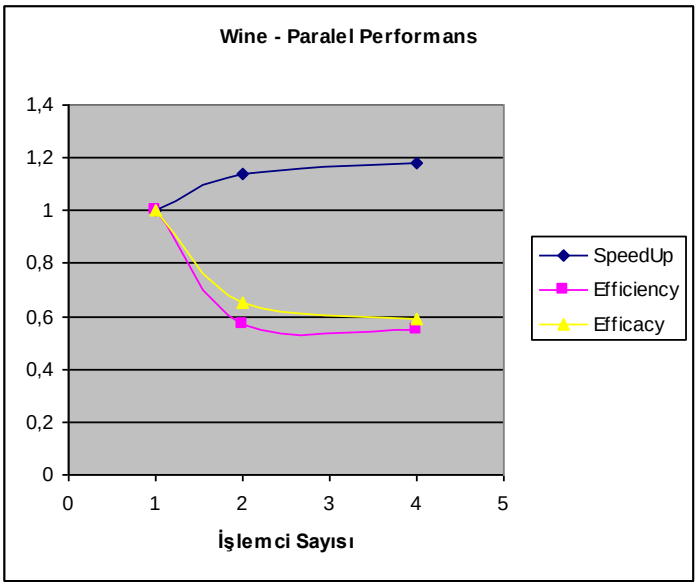
Şekil 7.13 'te farklı karınca ve işlemci sayıları için zaman değerindeki değişim gözlenmektedir. İşlemci sayısındaki artışın zaman değerlerinde azalmaya sebep olduğu gözlenmektedir. Atılacak toplam adım sayısının diğer işlemcilerle paylaştırılması sebebiyle bu zaten beklenen bir durumdur. 4 işlemci ile en düşük

çalışma zamanı elde edilmesine rağmen performans ölçümlerinde en düşük değer de yine 4 işlemci ile elde edilmiştir.



Şekil 7.13 : Wine Veritabanı ve Farklı İşlemci Sayıları ile Zaman Değişimi

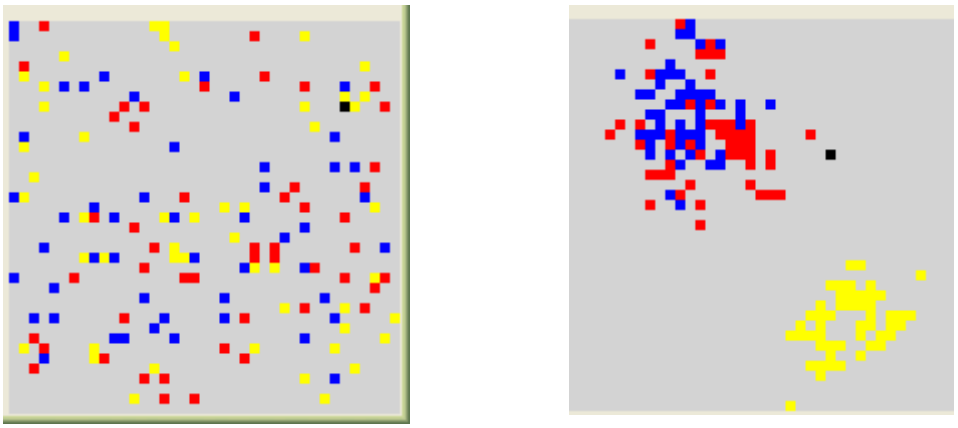
Şekil 7.14 'te farklı işlemci sayıları ile hızlanma, verimlilik ve etkinlik değerleri verilmiştir. İşlemci sayısının artması algoritma çalışma zamanlarında iyileşmeye sebep olmuştur. Ancak işlemci başına elde edilen hızlanma değerleri düşüktür. Ayrıca optimum işlemci sayısını belirleme ölçütü olan verimlilik değeri de işlemci sayısındaki artış ile azalmaktadır. Ancak işlemci maliyetini de göz önünde bulundurarak çalışma zamanındaki hızlanmayı sağlamak için 2 işlemci ile çalışmanın bu problem çözümü için uygun olacağı söylenebilir. Ancak hem en düşük başarım ölçümlerinin yapılmış olması hem de işlemci başına yeterli hızlanmayı sağlamaması sebebiyle 4 işlemci ile çalışma bu problem için uygun görünmemektedir.



Şekil 7.14 : Wine Veritabanı ile Elde Edilen Paralel Program Performans Ölçümleri

7.3.3 Iris veritabanı için elde edilen sonuçlar

Iris veritabanı 4 özellik ile belirtilen 150 elemanlı bir veritabanıdır. Her küme 50 adet veri içerir. Kümelerden ikisi lineer olarak ayrılamamaktadır. Bu veritabanı ile işlemler yapılırken veriler arası uzaklık hesaplamak için kosinüs uzaklığı kullanılmıştır. Şekil 7.15 'te örnek bir demetleme sonucu gözlenmektedir.



Şekil 7.15 : IRIS Verisi için Örnek Demetleme İşlemi

Bu veritabanı kullanılarak 1,2 ve 4 işlemci için elde edilen sonuçlar Tablo 7.8, Tablo 7.9, Tablo 7.10 'da listelenmiştir.

Tablo 7.8 : Bir İşlemci ile İris Veritabanı için Elde Edilen Sonuçlar.

	1	5	10	20
F-Ölçütü	0,7004(0,1239)	0,7641(0,0567)	0,798(0,0364)	0,7685(0,0549)
Siluet	0,8866(0,0474)	0,8928(0,0966)	0,9058(0,0587)	0,8923(0,0728)
Rand	0,7678(0,0099)	0,782(0,033)	0,7855(0,019)	0,7276(0,1264)
Variance	2,3235(0,9645)	2,0346(0,8435)	1,8589(0,8611)	1,9405(0,803)
Time	3,5023(1,0892)	3,4749(1,2368)	3,4361(1,376)	3,5354(1,2081)

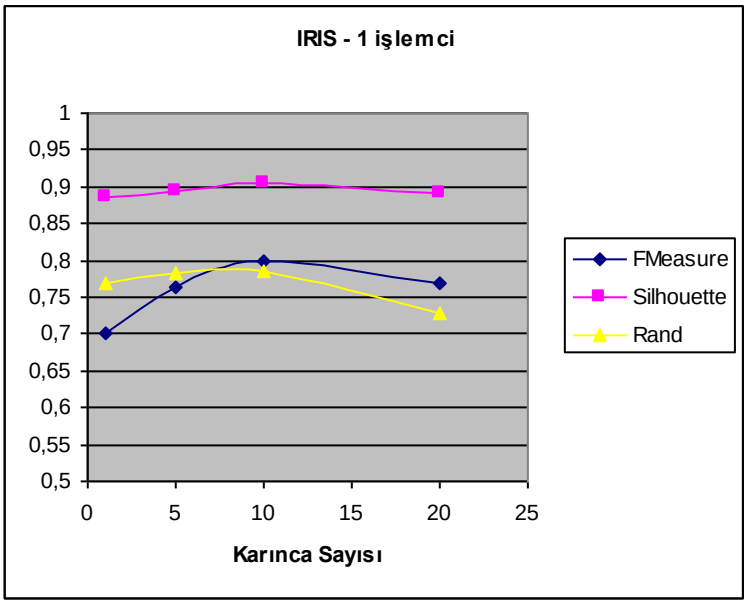
Tablo 7.9 : İki İşlemci ile İris Veritabanı için Elde Edilen Sonuçlar.

	1	5	10	20
F-Ölçütü	0,7195(0,0566)	0,7947(0,1313)	0,795(0,0738)	0,7418(0,1159)
Siluet	0,8585(0,0963)	0,8634(0,1044)	0,8677(0,0711)	0,8505(0,0886)
Rand	0,7925(0,0384)	0,8257(0,0126)	0,8411(0,0107)	0,7709(0,0146)
Variance	2,5857(0,4043)	2,0636(0,9682)	2,0583(0,3924)	2,4171(0,3402)
Time	2,6561(0,7986)	2,6836(0,8027)	2,1353(0,4641)	3,3724(1,172)

Tablo 7.10 : Dört İşlemci ile İris Veritabanı için Elde Edilen Sonuçlar.

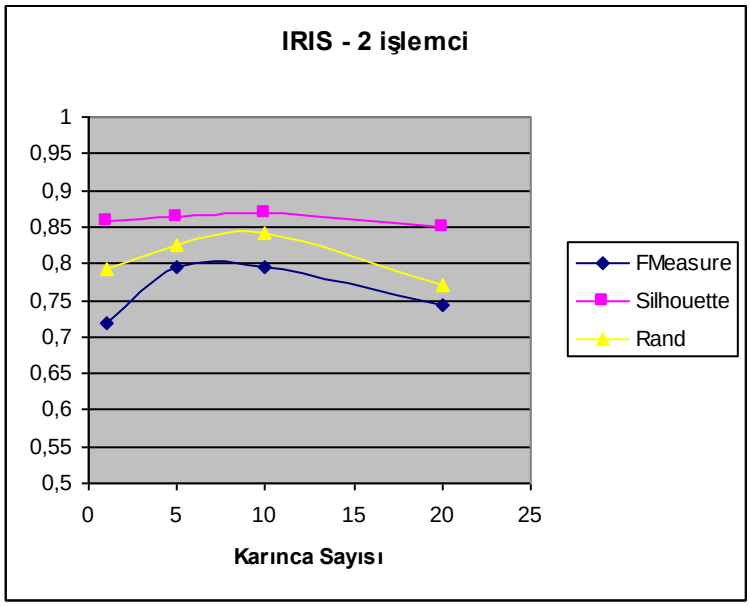
	1	5	10	20
F-Ölçütü	0,7047(0,1316)	0,7059(0,1265)	0,7944(0,039)	0,7685(0,0549)
Siluet	0,8772(0,0474)	0,8866(0,0966)	0,9076(0,0587)	0,8922(0,0728)
Rand	0,7678(0,0099)	0,782(0,033)	0,7855(0,019)	0,7276(0,1264)
Variance	2,3235(0,9645)	2,0346(0,8435)	1,9405(0,8611)	1,8589(0,803)
Time	2,2653(0,7115)	2,2849(0,7824)	2,1402(0,8436)	2,986(1,3248)

Şekil 7.16 ‘da farklı karınca sayıları için performans ölçüm değerlerindeki değişim görülmektedir. 10 civarında karınca sayısının bu veritabanı için optimum sayı olduğu görülmektedir. Tüm ölçüm kriterleri için yaklaşık 10 adet karınca ile en büyük değerler elde edilmektedir. F-ölçütü ve rand değerleri benzer davranış sergilemektedir. Şekil 7.19 ‘da siluet değerlerinin davranışı ile tutarlı varyans değeri davranışı gözlenmektedir. Yaklaşık 10 karınca civarında siluet en büyük değerini alırken varyans değeri en küçük değeri almaktadır.



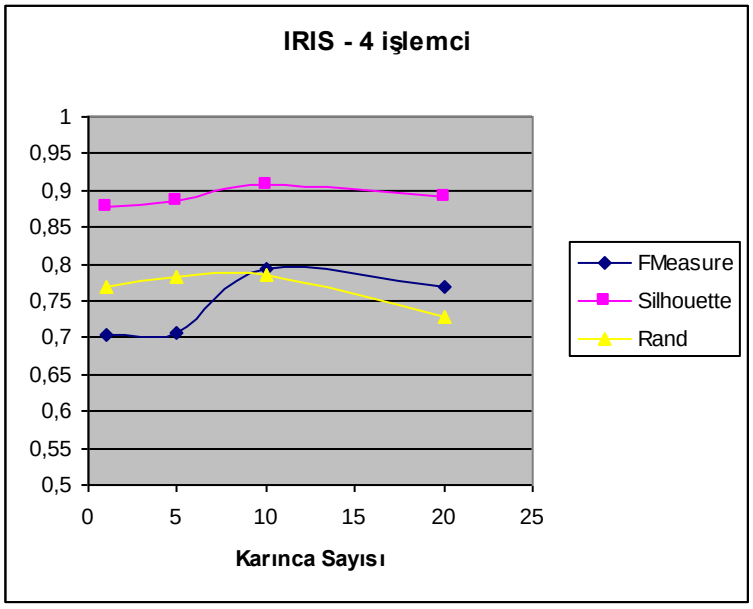
Şekil 7.16 : IRIS Veritabanı ve 1 İşlemci ile Elde Edilen Sonuçlar Grafiği

Şekil 7.17 ‘de farklı karınca sayıları ve 2 işlemci için performans ölçüm değerlerindeki değişim görülmektedir. 10 civarında karınca sayısının bu veritabanı için optimum sayı olduğu görülmektedir. 2 işlemci ile küme içi elemanların birbirine yakın olduğu ancak komşu kümelerle uzaklığın daha az olduğu yapılar elde ediliyor olabilir. Ancak kümelerin birbiri içine geçmemesi durumunda kümelerin çapının geniş olması f- ölçütü ve rand değerlerini değiştirmeyebilir. Siluet değerinin daha düşük değerler almasına karşılık Şekil 7.19 ‘da görüldüğü gibi varyans daha büyük değerler almaktadır. Bu durum aynı zamanda 2 işlemci ile varyans değerinin azalmasını da açıklamaktadır.



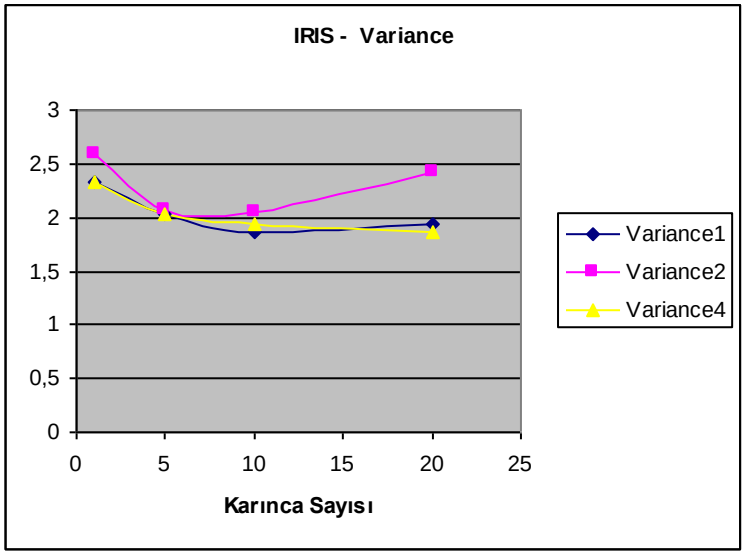
Şekil 7.17 : IRIS Veritabanı Ve 2 İşlemci ile Elde Edilen Sonuçlar

Şekil 7.18 'de farklı karınca sayıları ve 4 işlemci için performans ölçüm değerlerindeki değişim görülmektedir. 10 civarında karınca sayısının bu veritabanı için optimum sayı olduğu görülmektedir. Siluet değeri 2 işlemci için hesaplanan değerden daha yüksektir. Buna karşılık Şekil 7.19 ' da varyans değerinin azaldığı gözlemektedir. Rand ve f- ölçütü değerleri benzer davranış göstermektedir. 4 işlemci ile elde edilen sonuçlar 1 işlemci ile elde edilenlere yakın değerlerdedir.



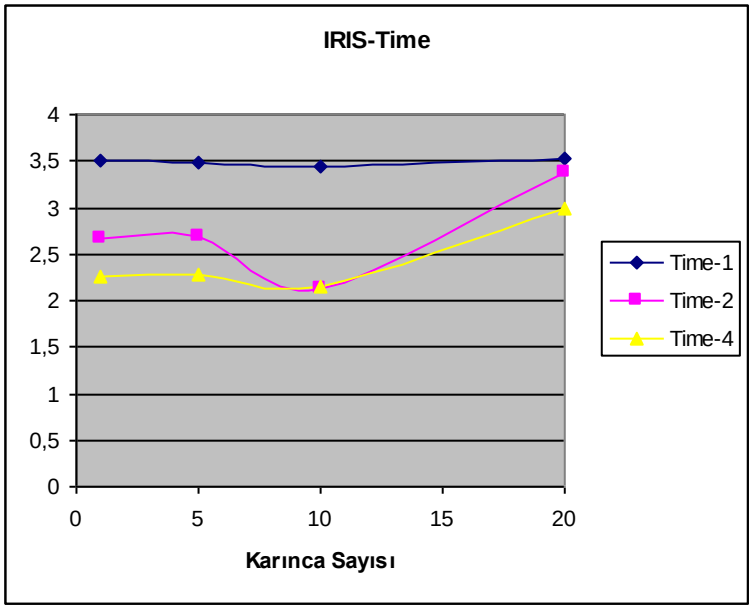
Şekil 7.18 : IRIS Veritabanı Ve 4 İşlemci ile Elde Edilen Sonuçlar

Şekil 7.19 ‘de farklı karınca ve işlemci sayıları için varyans değerindeki değişim gözlenmektedir. 2 işlemci ile en düşük varyans değeri elde edilirken 1 ve 4 işlemci ile 2 işlemciden daha yüksek ve birbirine yakın varyans değerleri elde edilmektedir.



Şekil 7.19 : IRIS Veritabanı için Değişik İşlemci Sayıları ile Varyans Değişimi

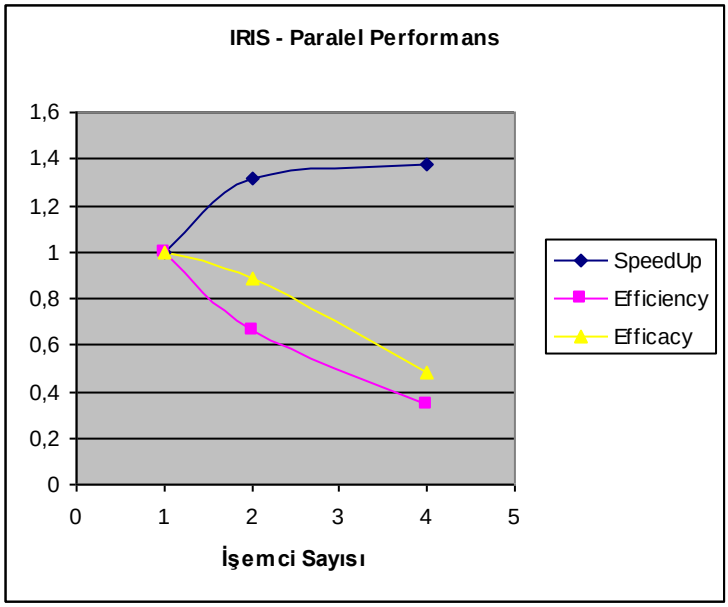
Şekil 7.20‘de farklı karınca ve işlemci sayıları için zaman değerindeki değişim gözlenmektedir. İşlemci sayısındaki artış algoritma zamanının azalmasına sebep olmaktadır.



Şekil 7.20 : IRIS Veritabanı için Değişik İşlemci Sayıları ile Zaman Değişimi

Şekil 7.21‘de 2 işlemci ve farklı karınca sayıları için hızlanma, etkinlik ve verimlilik değerlerindeki değişim gözlenmektedir. 2 işlemci ile daha düşük siluet, daha yüksek

varyans değeri elde edilirken algoritma zamanında da hızlanma görülmektedir. 4 işlemci ile siluet için 2 işlemci ile elde edilenden daha yüksek, 1 işlemci ile elde edilene yakın değerler elde edilirken algoritma zamanında hızlanma gözlenmektedir. Hem başarım kriterlerinin daha yüksek olması hem de algoritma çalışma zamanında iyileşmenin görülmesi sebebiyle 2 işlemci ile çalışma daha uygun olarak değerlendirilebilir.



Şekil 7.21 : IRIS Veritabanı için Paralel Program Performans Ölçümleri

8. SONUÇLAR VE ANALİZ

Daha önceki bölümlerde anlatıldığı gibi karınca kısıtlı bir zekaya yani belleğe sahiptir. Bu bellekte en son nesne bıraktığı, belirlenen sayıdaki hücrenin koordinatını ve o koordinata bırakılan nesneyi tutar. Karınca hareketi sırasında ve algoritma sonlanması durumunda taşıdığı yükü bırakmak için bu belleği kullanır.

Herhangi bir problem için karınca sayısı en az 1 ve en çok kümelenecek veritabanı boyutu kadar olabilecektir. Karınca sayısının optimum bir değer üstüne çıkması durumunda karıncaların belleklerinde tuttuğu verilerin güncel olmaması daha büyük bir olasılık olacaktır. Çünkü her iterasyon adımında rasgele bir karınca hareket eder, nesne alır veya bırakır. Karınca sayısının artması durumunda bir karıncanın seçilme aralığı artacak ve karınca tekrar adım atabilene kadar belleğinde tuttuğu hücrelerin verileri diğer karıncalar tarafından değiştirilmiş olabilecektir. Bu sebeple karınca sayısının çok büyük olması durumunda çözüm kalitesinin düştüğü gözlenmiştir.

Testler sırasında adaptif olmadığı ve kullanıcının girişi ile belirlendiği için farklı karınca sayıları kullanılmıştır. Karınca sayısı artışı başarımın artmasında etkilidir ancak bu artış sürekli değildir. Optimum bir değerden sonra karınca sayısının artırılması başarımın düşmesine sebep olmaktadır. Bu durum yapılan testler sırasında gözlenmiştir.

Karınca sayısının optimize edilmesi için daha önce yapılan bir çalışmada kümelenecek veritabanı boyutunun 0,07 ile çarpılması ile yaklaşık bir değer bulunabildiği bildirilmiştir [27]. Yapılan çalışmada Square1 veritabanı için optimum karınca sayısı 400*0,07 = 28 olacaktır. Square1 veritabanı için elde edilen başarım ölçüm grafiklerinde karınca sayısı artışı ile başarım kriterlerinde az da olsa artış gözlenmiştir. Testler için en fazla 20 karınca kullanıldığı için bu durum gözlenememiştir ancak 20 ‘den daha büyük bir karınca sayısı ile maksimum başarım ölçüm değerlerinin elde edileceği görülmüştür.

Wine ve iris veritabanları için ise literatüre göre sırasıyla 178*0,07 = 12 ve 150*0,07=10 karınca sayısı optimum değerlerdir. Bu durum her iki veritabanı başarım ölçüm grafiklerinde açıkça gözlenmiştir. Her iki veritabanı da 10 civarında karınca sayısı ile işlemci sayısından bağımsız olarak maksimum değerlerini almıştır.

Algoritmanın oluşturduğu küme yapısının oluşturulması için ortalama uzaklık ölçütü ile küme uzaklıklarını belirleyen hiyerarşik bir algoritmanın (average link)

kullanıldığı daha önce belirtilmişti. Görsel olarak kümelerin oluşturulduğu gözlenmektedir. Ancak başarım ölçümünün otomatik yapılabilmesi için program tarafında da kümelerin bilinmesi sağlanmalıdır. Average link ile durma koşulu olarak küme sayısı kullanılmıştır. Bu işlem karınca demetleme algoritması çalışması sonlandıktan sonra başarım ölçümü hesaplanması için yapılmaktadır. İleriki çalışmalarda durma koşulu olarak küme sayısı yerine oluşturulan dendogramda bir önceki seviye ile fark kullanılabilir.

Karınca demetleme algoritmasını diğer demetleme algoritmalarıyla karşılaştırdığımızda en büyük eksiği çalışma zamanının çok uzun olmasıdır. Algoritma süresi, üzerinde çalışılan veritabanındaki özellik vektörünün boyutuna göre değişmekte ve bu boyut büyüdükçe karınca demetleme algoritması çalışma zamanı diğer kümle algoritmaları çalışma zamanlarına yaklaşmaktadır [28]. Kümelenecek veri ile ilgili herhangi bir bilgiye sahip olmadan veri demetleme işlemini gerçekleştiren bu algoritmanın en zayıf noktası olan çalışma zamanının iyileştirilmesine çalışılmıştır. Bu amaçla, gerçeklenen ardışıl algoritma paralelleştirilmiştir. Karınca demetleme algoritmasının paralelleştirilmesinde akla ilk gelen yöntem ile haberleşme maliyetinin çok yüksek olacağı öngörülmektedir. Bu yöntemde grid ve karıncalar işlemcilere paylaştırılacak, karıncalar ve nesneler işlemciler arasında gezebileceklerdir. Bu durumun haberleşme maliyetinin yüksek olacağı açıktır. Gerçeklenen iteratif yaklaşımlı paralel algoritmada işlemciler arası iletişim tek bir noktada gerçekleştiğinden haberleşme maliyeti oldukça düşüktür.

İşlemci sayısındaki artış algoritma çalışma zamanında lineer bir azalmaya sebep olmamaktadır. Çalışma zamanını, kullanılan işlemci sayısı ile orantılı olarak azaltmasa bile gerçeklenen yöntem, algoritma çalışma zamanında iyileşmeyi sağlamaktadır. Ancak işlemci sayısı artışı algoritma performansının iyileşmesini sağlamamaktadır. Seçilen veritabanı eleman sayılarının az olması sebebiyle tek işlemci ile de hem performans ölçümleri hem de algoritma çalışma zamanı için iyi sayılabilecek sonuçlar alınmaktadır. Bu sebeple önerilen yapının başarımını daha iyi gözlemleyebilmek için hem eleman sayısı daha çok olan hem de özellik vektörü boyutu daha büyük olan bir veritabanı ile de testler tekrarlanabilir. Ancak bunun için performans ölçümü yapan küme analizi kısmındaki algoritmanın iyileştirilmesi gerekmektedir. Bu bölümde karmaşıklığı yüksek olan indislerin hesaplanması arka arkaya yapıldığından analiz sonuçlarının elde edilmesi için geçen zaman farklı karınca sayıları için bütün testlerin yapılmasını zorlaştırmaktadır.

Büyük veritabanları ile de bu çalışmada elde edilene benzer iyileşmenin görülmesi durumunda veritabanı hakkında herhangi bir bilgiye ihtiyaç duymayan karınca

demetleme algoritma diğer demetleme algoritmaları ile karşılaştırılacak başarıma erişebilecektir.

KAYNAKLAR

[1] **Jain, A. K., Murty, M. N. and Flynn, P. J.**, 1999. Data Clustering: A Review, *ACM Computing Surveys*, **31**, 264-323.

[2] **Luo, J., Wang, M., Hu, J., Shi, Z.,** 2007. Distributed Data Mining on Agent Grid: Issues, Platform and Development Toolkit, *Future Generation Computer Systems*, **23**, 61-68.

[3] **Stützle, T., Dorigo, M.,** 1999. ACO Algorithms for the Traveling Salesman Problem in *Evolutionary Algorithms in Engineering and Computer Science*, 163-183, John Wiley & Sons.

[4] **Aranha, C.,** 2006, A Survey on Using Ant-Based Techniques for Clustering, *IBA institute's Seminar*, The University of Tokyo, Tokyo, Japan, January 2006.

[5] **Deneubourg, J. L., Gross, S., Franks, N. R., Sendova-Franks A., Detrain C., Chretien L.,** 1991.The Dynamics of Collective Sorting: Robot-like Ants Ant-like Robots. *In Proceedings of the First International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, Paris, France, 356-363.

[6] **Gutowitz, H. A.**, 1993. Complexity-Seeking Ants. *In Proceedings of Third European Conference on Artificial Life*, Granada, Spain, June 1993.

[7] **Lumer H. A. E., Faieta.** 1994. Diversity and Adaptation in Populations Of Clustering Ants, *In Proceedings of the Third International Conference on the Simulation of Adaptive Behavior: From Animals to Animats 3*, Brighton, UK, 501-508.

[8] **Monmarché N., Slimane M., and Venturini G.,** 1999. On Improving Clustering in Numerical Databases with Artificial Ants. *Fifth European Conference on Artificial Ants*, Lausanne, Switzerland, September 1999, 626-635.

[9] **Ramos, V., Muge, F., Pina, P.,** 2002. Self-Organized Data and Image Retrieval as a Consequence of Inter-Dynamic Synergistic Relationships in Artificial Ant Colonies, *Second International Conference on Hybrid Intelligent Systems*, Santiago, Chile, December 2002, 500-509.

[10] **Hartmann, V.,** 2005. Evolving Agent Swarms for Clustering and Sorting, *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, Washington DC, USA, 217-224.

[11] **Vizine, A., de Castro, L. N., Hruschka, E. R., Gudwin, R.R.,** 2005. Towards Improving Clustering Ants: An Adaptive clustering algorithm, *Infomatica Journal*, **29**, 143-154.

[12] **Handl, J.,** 2003. Ant-based methods for tasks of clustering and topographic mapping: extensions, analysis and comparison with alternative methods, *Masters Thesis*, Chair of Artificial Intelligence, University of Erlangen-Numberg, Germany.

[13] **Handl, J., Meyer, B.,** 2002. Improved Ant Based Clustering and Sorting in a Document Retrieval Interface. *In Proceedings of the Seventh International Conference on Parallel Problem Solving from Nature*, Granada, Spain, September 2002, 913-923.

[14] **Handl, J., Knowles, J.,** 2006. An Evolutionary Approach to Multi-objective Clustering, *IEEE Transactions on Evolutionary Computation*, **11**, 56-76.

[15] **Handl, J.,** 2001. Visualizing Internet Queries using Ant-based Heuristics, *BS Thesis*, School of Computer and Software Engineering, Monash University.

[16] **Handl, J.,** 2003. Ant-based Methods for Tasks of Clustering and Topographic Mapping: Improvements, Evaluation and Comparison with Slternative Methods, *Masters Thesis*, Universität Erlangen-Nürnberg.

[17] **Handl, J., Knowles, J., and Dorigo, M.,** 2003. Ant-based Clustering: a Comparative Study of its Relative Performance with Respect to K-means, Average-link and 1d-som, *Technical Report TR/IRIDIA/2003-24*, IRIDIA, Universit e Libre de Bruxelles.

[18] **Stützle, T.,** 1998. Parallelization Strategies for Ant Colony Optimization, *In Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature*, Amsterdam, The Netherlands, September 1998, 722-731.

[19] **Albuquerque, P., Dupuis A.,** 2002. A Parallel Cellular Ant Colony Algorithm for Clustering and Sorting, *Proceedings of the 5th International Conference on Cellular Automata for Research and Industry*, Geneva, Switzerland**,** October 2002, 220-230.

[20] **Bonabeau E., Dorigo M.,** 1999. Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, Inc., New York.

[21] URL : http://www-unix.mcs.anl.gov/mpi/mpich1/mpich-nt/ , Instructions to get MPICH up and running, 2007.

[22] **Willcock, J., Lumsdaine, A., Robison A.**, 2002. Using MPI with C# and Common Language Infrastructure, *Indiana University Computer Science Department Technical Report 570*.

[23] **Bolshakova, N., Azuaje, F.**, 2002. Cluster Validation Techniques for Genome Expression Data, *Signal Processing*, **83**, 825-833.

[24] **Bolshakova, N., Azuaje, F.**, 2003. Improving Expression Data Mining Through Cluster Validation, *Fourth Annual IEEE EMBS Special Topic Conference on Information Technology Applications in Biomedicine*, New York, USA, 825-833.

[25] **Kovács, F., Legány, C., Babos, A.**, 2005. Cluster Validity Measurement Techniques, *Sixth International Symposium of Hungarian Researchers on Computational Intelligence*, Budapest, Hungary, November 2005.

[26] **Dunn, J.,** 1974. Well Separated Clusters and Optimal Fuzzy Partitions, *Journal of Cybernetics*, **4,** 95-104.

[27] **Ekola, T., Laurikkala, M., Lehto, T., Koivisto, H.**, 2004. Network Traffic Analysis Using Clustering Ants, *World Automation Congress*, **17**, 275- 280.

[28] **Handl, J., Knowles, J., Dorigo, M.,** 2003. On the performance of ant-based Clustering, *In Proceedings of the Third International Conference on Hybrid Intelligent Systems*, Melbourne, Australia, December 2003.

ÖZGEÇMİŞ

1981 İstanbul doğumlu olan Özlem GEMİCİ; orta ve lise öğrenimini Manisa Demirci Ortaokulu ve Lisesi ‘nde tamamlamıştır. 1999 yılında lise mezuniyeti sonrasında İstanbul Teknik Üniversitesi Bilgisayar Mühendisliği Bölümünü kazanmış ve 2003 yılında bu bölümden lisans derecesini almıştır. 2004 yılında Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Yüksek Lisans Programı’na kaydolmuş ve Mayıs 2007 itibariyle yüksek lisans tezini sunmaktadır.