T.C.
ERCİYES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DİSK PLANLAMA İÇİN KARINCA KOLONİ ALGORİTMASINA DAYALI YENİ BİR METOD

Tezi Hazırlayan
Selçuk ÖKDEM

Tezi Yöneten
Doç. Dr. Derviş KARABOĞA

Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Temmuz 2003
KAYSERİ

Bu çalışma, jürimiz tarafından Erciyes Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

23/07/2003

JÜRİ:

Üye : Prof. Dr. Mustafa ALÇI

Üye : Doç. Dr. Derviş KARABOĞA

Üye : Yard. Doç. Dr. Veysel ASLANTAŞ

ONAY:

Bu tezin kabulü Enstitü Yönetim Kurulunun 25.-07.-2003 tarih ve.....1.... sayılı kararı ile onaylanmıştır.

15./EKİM/2003

T.C.
Prof. Dr. İlhan ÖZTÜRK
Enstitü Müdürü
Mühür ve İmza

TEŞEKKÜR

Yüksek lisans tez çalışmamda bana destek olan sayın hocam Doç. Dr. Derviş Karaboğa'ya teşekkürü bir borç bilirim.

Ayrıca çalışmalarımda büyük katkıda bulunan Bilgisayar Mühendisliği Bölümü 3. sınıf öğrencilerinden Ayder Bulatov ve çalışmalarımda desteğini esirgemeyen arkadaşlarım Bilgisayar Mühendisi Hasan Kağan Akıcı ve Arş. Gör. Serkan Öztürk'e teşekkür ederim.

Bütün eğitim ve öğretim hayatım boyunca tüm imkanları ile bana destek olan ve her türlü fedakarlığı gösteren eşime, anneme, babama ve kardeşlerime şükranlarımı sunarım.

DİSK PLANLAMA İÇİN KARINCA KOLONİ ALGORİTMASINA DAYALI YENİ BİR METOD

ÖZET

Disk hızına nazaran işlemci hızı ve hafıza kapasitesinin kat kat hızlı gelişim içerisinde olmasından dolayı disk planlama problemi teorik ilgi ve pratik öneme sahip olmuştur. Disk hızlandırma teknolojisindeki yavaş ilerleme nedeniyle daha etkin bir şekilde disk kullanımı metotlarına gerek duyulmaktadır. Disk ve hafıza kapasitesi gelişimi disk hızı gelişiminden çok daha üst seviyede olmasına rağmen, daha verimli bir şekilde disk kullanımını sağlayan algoritmaların geliştirilmesi için az sayıda çalışma yapılmıştır. Tez çalışmasında, disk planlama problemi için karınca koloni algoritması üzerine kurulu yeni bir yaklaşım önerilmiştir ve bu yaklaşımın performansı değerlendirilmiştir.

Anahtar Sözcükler : Disk Planlama, Karınca Koloni Optimizasyonu

A NEW METHOD FOR DISK SCHEDULING BASED ANT COLONY OPTIMIZATION ALGORITHM

ABSTRACT

Disk scheduling problem has theoretical interest and practical importance in the case that processor speed and memory capacity have been increasing several times faster than disk speed. More efficient disk usage methods have been needed because of the slow evolution in disk speed technology. Although disk and memory capacity increment is much higher versus disk speed improvement, little studies have been made to develop disk usage algorithms in more efficient manner. In this work, a new approach based on ant colony algorithm is proposed for disk scheduling problem and its performance is evaluated.

Key Words : Disk Scheduling, Ant Colony Optimization

İÇİNDEKİLER

BÖLÜM 1

BÖLÜM 2

BÖLÜM 3

BÖLÜM 4

BÖLÜM 5

ŞEKİLLERİN LİSTESİ

BÖLÜM 1

GİRİŞ

1.1. Konunun Tanımı ve Önemi

Ağ yönlendirme, işlemci planlaması, hafıza organizasyonu ve disk planlaması gibi bilgisayar problemlerinin çözümünde kullanılan klasik tekniklerin yerine gelişime dayalı algoritmaların kullanımı çeşitli avantajlar sağlamaktadır. Gelişime dayalı algoritmalardan karınca koloni algoritması özellikle ağ yönlendirmesi probleminde tercih edilmektedir [1]. Bu çalışmada, disk planlama probleminin çözümü için karınca algoritmasına dayalı yeni bir metot geliştirilmiştir.

Sürekli gelişmekte olan bilgisayar teknolojisinde, bilgisayarın temel parçaları olan işlemci, ana hafıza ve ikincil hafıza olan harddisk bilgisayar performansını birinci derecede etkilemektedir. İşlemci hızındaki gelişmeye ve ana hafıza kapasite artışı ve hızındaki gelişmeye nazaran harddisk veri alış verişi hızındaki gelişme oldukça sönük kalmaktadır. Fiziksel olarak harddisk teknolojisindeki ilerlemenin yavaş olmasına mukabil, harddiskden veri alış-verişi organizasyonunun yazılımsal olarak geliştirilmesiyle işletim sistemi ve harddisk arasındaki haberleşme performansı artırılabilmektedir. Literatürde bu konuda yapılan bilimsel çalışma sayısının pek fazla olmaması bu problemin çözümüne yönelik yeni bir tekniğin geliştirilmesinin önemini artırmaktadır.

1.2. Tezin Amacı

Bu tezin maksadı işletim sistemi ile harddisk arasındaki haberleşme performansını artırmak için probleme uygun, kolay uyarlanabilir, etkin ve hızlı çalışan bir algoritma geliştirilmesidir.

1.3. Tezin İçeriği

İkinci bölümde gelişime dayalı algoritmalardan problem için uygun olduğu düşünülen karınca koloni optimizasyonu hakkında bilgi verilmiştir. Ayrıca bu bölümde gelişime dayalı algoritmaların en çok uyarlandığı bilgisayar problemlerinden olan yönlendirme optimizasyonu problemine karınca koloni algoritmasının uygulaması özetlenmiştir. Üçüncü bölümde disk planlama problemi tanıtılmış ve problem için daha önceden geliştirilmiş olan teknikler ve işleyiş düzenleri anlatılmıştır. Dördüncü bölümde disk planlama problemine karınca koloni optimizasyonu algoritmasının uyarlanması yapılmış ve algoritma probleme özgü şekilde düzenlenmiştir. Bu bölümde daha sonra problem için üretilen simülasyon verileri kullanılarak klasik teknikler ve karınca koloni optimizasyonu algoritması için sonuçlar alınmış ve bunlar şekiller üzerinde gösterilerek karşılaştırılmıştır. Son bölümde ise dördüncü bölümde alınan sonuçların değerlendirilmesi yapılarak geliştirilen yeni yaklaşım hakkında yorum yapılmıştır.

BÖLÜM 2

KARINCA KOLONİ OPTİMİZASYONU (KKO)

2.1. Giriş

Bu bölümde karınca kolonilerinin yiyecek temin etme davranışlarından esinlenerek, farklı optimizasyon problemleri için kullanılan 'karınca algoritması' üzerinde yapılan son çalışmalar incelenecek ve karınca koloni optimizasyon algoritması (KKO) tanıtılacaktır. Çalışmanın ilk bölümünde, gerçek karıncalar ile ilgili temel biyolojik bulgular incelenecek ve bunların suni karşılıkları tanımlanacaktır. Çalışmanın ikinci bölümünde KKO algoritmasının ayrık optimizasyonla ilgili uygulamaları ve haberleşme ağlarında kullanılmakta olan yol verme işlemi açıklanacaktır.

2.2. Temel Prensipler

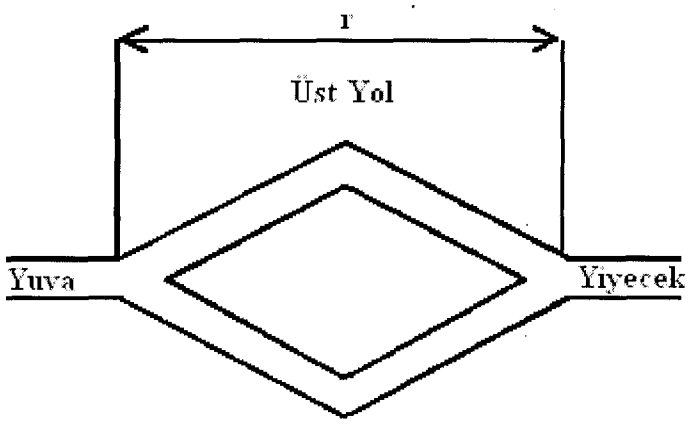
Karınca algoritması ilk olarak Dorigo ve meslektaşları tarafından, gezgin satıcı elemanı problemi (GSP) ve quadratik sıralama problemi (QSP) gibi zor kombinasyonel optimizasyon problemlerinin çözümü için bir yaklaşım olarak geliştirilmiştir [1][2]. Birçok zor ve farklı sahalarda optimizasyon problemlerine karınca tabanlı algoritmaların uygulanması için halen devam eden çok sayıda çalışma mevcuttur. Son uygulamalar, araç yönlendirme, ardışık sıralama, grafik renklendirme, haberleşme ağlarında yol verme gibi problemlere çözüm getirmeyi amaçlamaktadır.

Karınca algoritmasında, gerçek karınca kolonilerinin yiyecek temin etme davranışlarından ilham alınmıştır. Karıncalar, koloniler halinde yaşayan ve davranışları kolonin içinde tek bir birey olmaktan ziyade bir bütün halinde koloni olarak hayatta kalabilmek için yönlendirilen sosyal varlıklardır. Özellikle koloni bireylerinin bireysel basitliğine bakıldığı zaman karıncaların kolonilerini oluşturmalarındaki yüksek

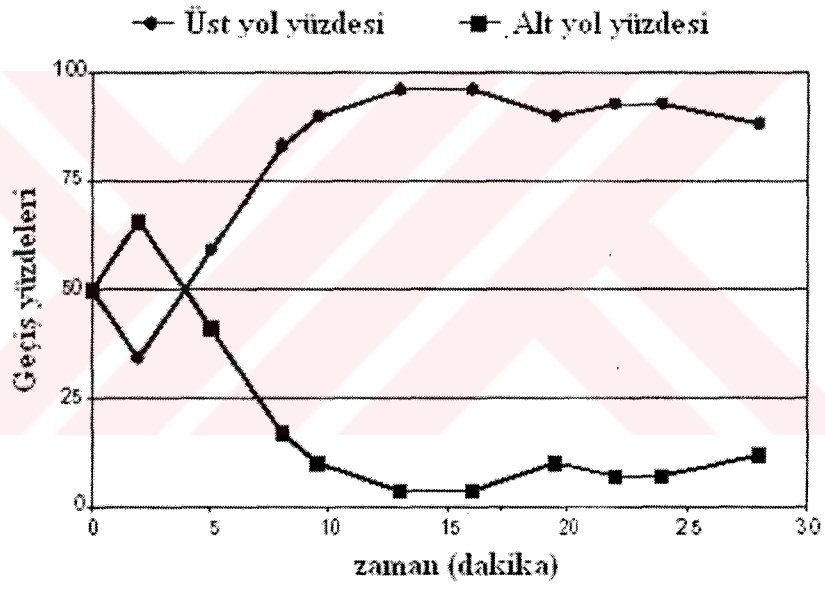
yapılanma seviyeleri bilim çevrelerinin dikkatini çekmiştir. Karınca kolonisinin en önemli ve ilginç olan davranışı, yiyecek bulmak için gösterdikleri çaba ve özellikle de yiyecek kaynağı ve yuvaları arasındaki en kısa yolu bulabilme özellikleridir.

Yiyecek kaynaklarından yuvalarına ve yuvalarından yiyecek kaynaklarına giderlerken, karıncalar geçtikleri yere feromon (phemorone) adı verilen kimyasal bir madde (koku) bırakırlar ve böylece feromon izi oluştururlar. Feromon izi karıncalara tekrar yuvalarına (yada yiyecek kaynağına) dönebilme imkanı sağlar. Ayrıca bu iz yuva arkadaşları tarafından bulunan yiyecek kaynaklarının yerini bulmak için diğer karıncalar tarafından da kullanılabilmektedir. Karınca kolonileri tarafından herhangi bir zamanda kullanılmış olan bu feromon izini takip etme davranışı en kısa yolun ortaya çıkmasına sebebiyet verir. Şöyleki; yuvadan yiyecek kaynağına birden fazla yol olduğu zaman, bir karınca kolonisi, bireysel karıncalar tarafından yapılmış olan feromon izlerinden, yuvadan yiyecek kaynağına ya da tersine en kısa yolu keşfetmek için istifade eder. Karıncaların yiyecek temin etme davranışındaki durumlarını incelemek için Deneubourg tarafından ikili(binary) köprü deney düzeneği kurulmuştur (Şekil 2.1.a.). Bu deneyde karınca kolonisinin yuvası ile yiyecek kaynağı birbirine eşit uzunlukta iki kolu olan bir köprü ile ayrılmıştır. Karıncalar yuvadan serbest bırakıldıktan sonra, zamana bağlı olarak iki koldan birini seçen karıncaların yüzdeleri incelenmiştir (Şekil 2.1.b.). Sonuçta; bazı osilasyonların görünebileceği ilk geçiş fazından (initial transitory phase) sonra karıncalar aynı yol üzerinde birleşme eğilimi göstermişlerdir.

Aşağıdaki deneyde ilk başta iki kolda da feromon yoktur, bundan dolayı iki kolda karıncalar tarafından aynı olasılıkla seçilmiştir. Bununla beraber ilk geçiş fazından sonraki rastgele düzensizlikler, daha fazla karıncanın bir kolu diğerinden daha fazla seçmesine sebep olmaktadır (üst kol). Çünkü karıncalar yürürlerken feromon bırakmaktadırlar. Üst koldaki karıncaların çokluğu, dönüşte daha fazla karıncanın o kolu seçmesini teşvik etmektedir. Bu ise o koldaki feromon miktarının daha fazla olmasını sağlamaktadır. Bu fenomeni tanımlayan olasılık modeli şu şekildedir: İlk olarak bir koldaki feromon miktarının o koldan daha önce geçmiş olan karıncaların sayısıyla orantılı olduğu farz edilir. Burada feromon buharlaşması hesaba katılmamıştır.



(a)



(b)

Şekil 2.1. Deneubourg ikili köprü deneyi

Yaklaşık olarak bir saatte sonlanan tipik bir deneyde, bu zaman periyodu içinde buharlaşan feromon miktarı ihmal edilebilmektedir. Bu modelde, belli bir zaman içinde bir kolun seçilme olasılığı o koldaki toplam feromon miktarına bağlıdır. Bu da aynı süre içinde o koldan geçen karıncaların sayısı ile orantılıdır. Köprüyü kullanan toplam karınca sayısı m, üst kolu kullanan karınca sayısı U_m, alt kolu kullanan karınca sayısı L_m ile gösterilirse ($U_m + L_m = m$), (m+1)'inci karıncanın üst kolu seçme olasılığı $P_U(m)$,

$$P_U(m) = \frac{(U_m + k)^b}{(U_m + k)^b + (L_m + k)^b} \tag{2.1}$$

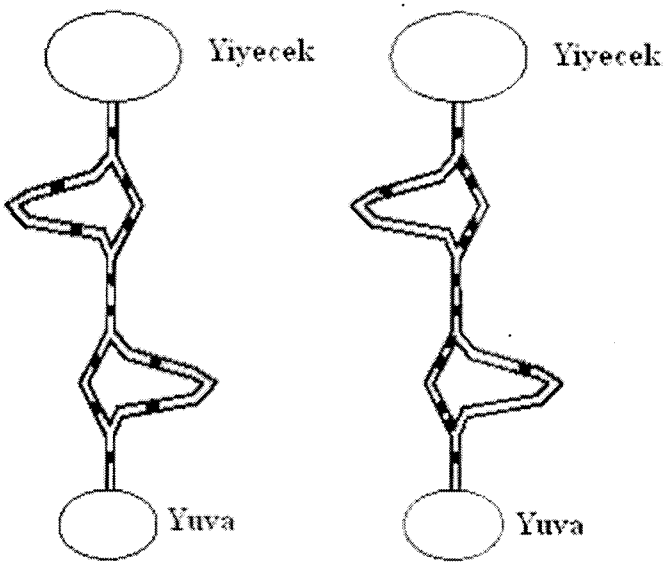
formülü ile ifade edilir. Alt kolu seçme olasılığı $P_L(m)$ ise $P_L(m)= 1-P_U(m)$ şeklinde olur. Bir kolu veya diğer kolu seçme olasılığının bu fonksiyonel ifadesi feromon izleme üzerine yapılan deneyler sonucunda bulunmuştur. b ve k parametreleri bu modeli deneysel verilere uygunlaştırma imkanı sağlamaktadır. Karınca seçim dinamiği şu şekilde ifade edilir: $P_U \geq \psi$ ise $U_{m+1}=U_m+1$, diğer durumlarda $U_{m+1}=U_m$. Burada ψ, aralık boyunca düzgün olarak değişen rastgele bir değişkendir.

Monte Carlo similasyonu bu model ile gerçek veriler arasındaki uygunluğu test etmek için kullanıldığında, parametreler, $k\approx20$ $b\approx2$'ye set edildiği zaman deneylerle gerçek karıncaların uyuştuğu gözlenmiştir.

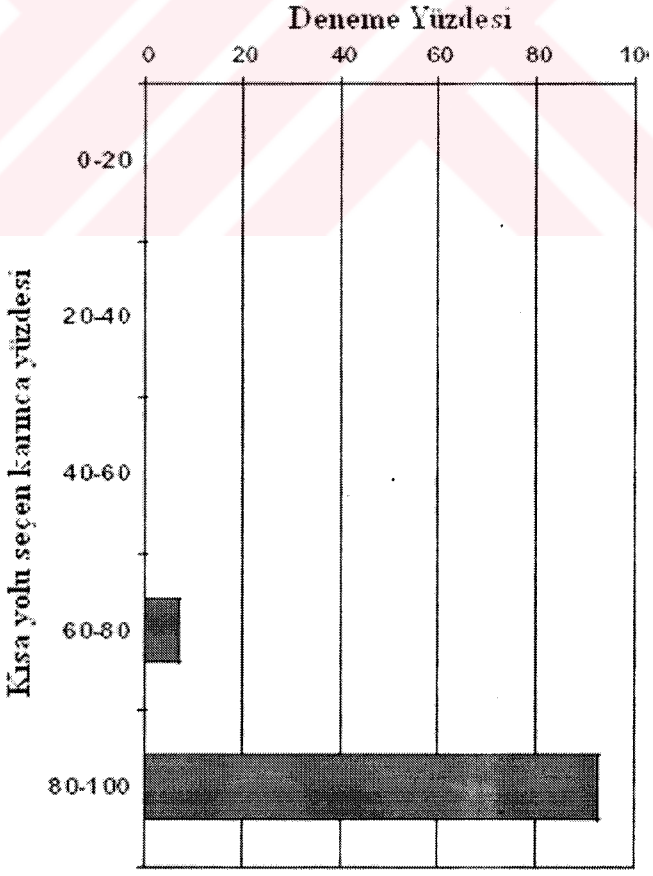
Deneyi, köprünün kol uzunluklarının farklı olması durumuna dönüştürmek (Şekil 2.2.) ve bu yeni durum için (2.1) eşitliğini genişletmek oldukça basittir. Bu yeni durumda feromon bırakma mekanizması bir önceki durumla aynı olduğundan dolayı en kısa yol daha fazla sıklıkla seçilecektir. Yiyecek kaynağına ulaşan ilk karıncalar, kısa yolu kullanmış olan karıncalardır ve bunlar dönüşte, kısa kollardaki feromon miktarı uzun kollardakinden daha fazla olduğu için yine kısa yolu tercih ederler. Bu durumda, başlangıçta meydana gelen rastgele düzensizliklerin önemi oldukça azalır ve karıncaların farklı kol uzunluklarında hedefe ulaşmak için ihtimale dayalı feromon yolunu takip etme davranışı bu çalışmada ana mekanizma olmaktadır.

Açıkça görülüyor ki yukarıda açıklanan yöntem, her bir karıncanın yapmış olduğu yardımlaşmadan oluşan bir çeşit dağıtılmış optimizasyon mekanizmasıdır. Prensipte bir karınca tek bir çözüm bulabilme yeteneğine sahip olabilmesine rağmen, karınca topluluğu yani karınca kolonisi 'en kısa yolu bulabilme' davranışını sergileyebilmektedir. Bir anlamda bu davranış karınca kolonisinde ortaya çıkan bir özelliktir. Karıncalar bu özel davranışı 'stigmergy' olarak adlandırılan, feromon bırakma vasıtasıyla oluşturulan, dolaylı haberleşmenin basit bir biçimini kullanarak gerçekleştirirler.

'Bellicositermes Natalensis' ve 'Cubitermes' yaptığı çalışmasında Grasse' nin tanımına göre stigmergy, karıncaların başarabildikleri en iyi performansla birbirlerini yönlendirme yetenekleri şeklinde ifade etmektedir [19].



(a) (b)



(c)

Şekil 2.2. Çift köprü deneyi

Grasse bu varlıkların, genetik olarak şifrelemiş bir tepkiyi harekete geçiren 'significant stimuli' olarak bilinen önemli uyarıya cevap verebilme yeteneğine sahip olduklarını ileri sürmüştür [19]. Sosyal varlıklara en iyi bilinen örneklerden olan beyaz karıncalar ve normal karıncalarda bu tepkilerin etkileri, hem bunu üreten böcek için hem de kolonideki diğer böcekler için yeni bir önemli uyarı olarak algılanır. Önemli bir uyarıya verilen tepkinin sonucu olarak uyarının ürünü olan aktivitelerin koordinasyonunun formu belirlenir ve buda dolaylı haberleşmenin bir şekli olarak yorumlanır. Örneğin; Grasse, Bellicositermes Natalensis ve Cubitermes, karıncaların yeni bir yuva inşa ederlerken işe toprak topaklarını rastgele ve koordinesiz bir biçimde bırakarak başladıklarını gözlemlemiştir. Fakat sınırlı bir bölgede bu toprak topakları belli bir yoğunluğa ulaşınca bunlar yeni bir önemli uyarı haline gelir ve bu uyarı da daha fazla beyaz karıncanın sütunlar ve kemerler için toprak topakları eklemesine sebep olur. Böylece bütün yuva inşa edilmiş olur. 'Stigmergy'yi diğer haberleşme vasıtalarından ayıran özellikler şunlardır: a) haberleşen böcekler tarafından bırakılan bilginin fiziksel doğası, b) bırakılan bilginin bölgesel doğasıdır.

Bu çalışmanın ana ilkesi haberleşmenin 'stigmergy'yi destekleyen bir modelinin oluşturulmasıdır. Bu model, zor optimizasyon problemlerinin çözümlerine uygulanan suni çoklu-ajan sistemler için ilginç bir modeldir. 'Stigmergy' nin yukarıda bahsedilen karakteristik özellikleri suni ajanlara şu yollarla kolaylıkla uygulanabilir: a) problem durumları ile uygun durum değişkenleri arasında ilişki kurularak b) bu değişkenlerin değerlerine suni ajanları vererek.

Örneğin yukarıda açıklanan karıncaların yiyecek temin etme davranışında kullandıkları 'stigmergy' haberleşmesi, karıncaların yürürlerken geçtikleri yere feromon bırakma işidir. Benzer olarak, suni karıncalar uygulandıkları optimizasyon problemlerine çözüm oluştururken, karşılaştıkları problem durumları ile ilişkilendirilen uygun 'feromon değişkenleri'ni değiştirmek suretiyle feromon bırakma işini taklit ederler.

Suni karıncaların faydalandığı, gerçek karıncaların yiyecek temin etme davranışlarının diğer önemli bir yönü ise pozitif geri besleme (autocatalysis) mekanizması sayesinde çözümler ile değerlendirme arasındaki koordinasyon yapısıdır. Değerlendirme ile

kastedilen unsur, suni karıncaların daha düşük maliyetli çözümü temsil eden kısa yolları uzun yollardan daha kısa sürede geçebileceği ve böylece ilave feromon'u daha hızlı bir şekilde bu yolların alacak olmasıdır. Pozitif geri beslemeyle birleştirilen çözüm değerlendirmesi, problem çözümünde etkili olabilir. Yol ne kadar kısa olursa, o kadar çok feromon bırakılır, böylece bu yoldan daha fazla karınca geçer. Eğer pozitif geri besleme uygun olarak kullanılırsa, popülasyon tabanlı optimizasyon algoritmalarında çok güçlü bir mekanizma olabilir. Gerçektende bu mekanizma araştırma sürecini yönlendirebilmesi için hızlı bir şekilde en iyi bireyleri tercih eder. Pozitif geri besleme kullanıldığı zaman erken yakınsamadan (durgunluktan) kaçınmak için bazı şeyler dikkate alınmalıdır. Erken yakınsama, bazı çok iyi olmayan bireylerin araştırma uzayının daha fazla incelenmesini engelleyerek popülasyonun kontrolünü ele geçirme durumudur. Feromon izi buharlaşması ve rastgele durum geçişleri pozitif geri beslemenin dezavantajlarını ortadan kaldıracaktır..

Aşağıda karınca algoritması çeşitleri ve karınca koloni optimizasyonun (KKO) sezgiselliği hakkında açıklamalar yer almaktadır.

2.3. Karınca Kolonisi Optimizasyonu'nun Gerçekleştirilmesi

Karınca kolonisi optimizasyonunda (KKO), zor ve farklı optimizasyon problemleri için en iyi yolun bulunmasında, suni karınca kolonileri kendi aralarında işbirliği yapar. İşbirliği, KKO algoritmasının anahtar konumundaki parçasıdır. İyi çözümler, bireyler (ajan) arası işbirliğinin doğal bir sonucudur.

KKO, gerçek karınca kolonilerinde gözlenen, en kısa yolu bulma davranışını sergilemektedir. Doğal karşılıkları olan ama gerçek karıncalarda bulunmayan bazı özellikler eklenerek algoritma zenginleştirilmiştir. KKO'nun mühendislik uygulamalarında zor optimizasyon problemlerinin çözümü için yazılım sistemlerinin oluşturulmasında ve çalıştırılmasında etkili olması istenir. İşte bu yüzden suni karıncalara bazı özellikler eklenerek daha etkili ve becerikli olmaları sağlanmıştır. Aşağıda suni karıncaların bazı özellikleri ve gerçek karıncalardan farklı yönleri anlatılmaktadır.

2.3.1. Gerçek Karıncalarla Farklılıkları ve Benzer Yönleri

KKO yaklaşımının temel prensipleri gerçek karıncalardan gelmektedir. Özellikle:

a) İşbirliği içinde olan bireylerden oluşan koloninin kullanımı,
b) Yerel 'stigmergetic' haberleşme için suni bir feromon yolunun kullanımı,
c) En kısa yolu bulmak için ardışık yerel hareket kullanımı,
d) İleriye bakmaksızın sadece yerel bilgiler kullanılarak en kısa yolu bulabilmek için tahmini karar verme yönteminin kullanılması.

İşbirlikçi bireyler kolonisi: Gerçek karınca kolonileri gibi karınca algoritması da bir popülasyondan (koloniden) ibarettir. Eşzamanlı ve eşzamanlı olmayan varlıkların hepsinin birden en iyi çözümü bulmak için işbirliği yaptığı bir popülasyondur. Her bir suni karınca mümkün olabilen bir çözüm bulabilir ama yüksek kaliteli çözümler, koloninin bütün bireyleri arasındaki işbirliğinin sonucu olarak ortaya çıkar. Karıncalar uğradıkları problem bölgelerinde aynı anda okuyup yazabildikleri bilgiler aracılığıyla işbirliği yaparlar.

Feromon izi ve stigmergy: Gerçek karıncaların, çevrede yaptığı değişimler gibi suni karıncalar da bulundukları çevrede bazı değişimler yaparlar. Gerçek karıncalar dünya üzerinde uğradıkları bölgelere kimyasal madde (feromon) bırakırlarken, suni karıncalar ise uğradıkları problem bölgelerinde yerel olarak depolanmış olan nümerik bilgileri değiştirirler. Bu bilgi, karıncanın o günkü performansını hesaba katarak hesaplanır ve o bölgeye giren herhangi bir karınca tarafından okunabilir veya yazılabilir. Bu nümerik bilgi "suni feromon izi" olarak adlandırılır. KKO algoritmasında yerel feromon izleri karıncalar arasındaki tek haberleşme kanalıdır. Haberleşmenin 'stigmergy'yi temel alan formu bilginin ortak kullanımında en büyük rolü oynamaktadır. Genellikle KKO algoritmasında buharlaşma mekanizması feromon bilgisini zamanla değiştirir. Feromon buharlaşması, koloninin geçmiş kararlarının zorlaması olmaksızın yeni yönlere doğru araştırmasını yönlendirebilmesi için geçmişi yavaş yavaş unutmasını sağlar.

En kısa yol arama ve yerel hareketler: Suni ve gerçek karıncalar ortak bir görev üstlenirler. Bu bir orijin (yuva) ile hedefi (yiyecek kaynağı) birbirine bağlayan en kısa yolun (minimum maliyetin) bulunmasıdır. Gerçek karıncalar atlama yapmaz, komşu bölgeler boyunca yürürler. Suni karıncalar da böyledir, problemin komşu durumları (state) boyunca adım adım ilerlerler.

Daha önce de bahsedildiği gibi suni karıncalar gerçek karıncalarda bulunmayan bazı karakteristiklere sahiptir. Bunlar:

- Suni karıncalar farklı bir dünyada yaşarlar ve hareketleri durumdan duruma farklılık arz eder.

- Suni karıncalar bir içsel duruma (internal state) sahiptirler. Bu özel durum, karıncanın geçmiş hareketlerinin hafızasını içerir.
- Suni karıncalar bulunan 'çözümün kalitesinin' fonksiyonu olan bir miktar feromon bırakır.
- Suni karıncaların feromon bırakma zamanlaması probleme bağlıdır ve böylece gerçek karıncaların bu davranışını aynen yansıtmaz. Örneğin birçok durumda suni karıncalar bir çözüm bulduktan hemen sonra feromon izlerini düzenlemektedir.
- Tüm sistemin etkinliğini iyileştirmek için KKO algoritması ileriye bakma (lookahead), yerel optimizasyon, vb. gibi ekstra özellikler eklenerek zenginleştirilebilir.

Aşağıdaki bölümde suni karıncaların farklı optimizasyon problemlerine uygulanabilmesi için algoritmik bir ortamda nasıl çalıştığı anlatılacaktır.

2.3.2. KKO Sezgiselliği

KKO algoritmasında yukarıdaki bahsedilen özelliklere sahip sonlu büyüklükteki bir suni karınca kolonisi, incelenmekte olan optimizasyon problemine yardımlaşmayı esas alarak kaliteli çözümler üretmeye çalışır. Her karınca probleme bağlı kriterlere göre seçilmiş olan bir başlangıç durumundan başlayarak, bir çözüm veya çözümün bir parçasını üretir. Her karınca kendi çözümünü üretirken problem özelliklerinden ve kendi performasından bilgiler biriktirir ve bu bilgileri problemin ifadesini modifiye etmek için kullanır. Karıncalar işbirlikçi bir davranış sergileyerek eşzamanlı veya bağımsız bir şekilde hareket edebilirler. Doğrudan haberleşme kullanmazlar, karıncalar arası bilgi değişimini yöneten stigmergy'i kullanırlar.

Karıncalar mümkün olabilecek bir çözümü araştırmak için fazladan yardımcı uygulama kullanırlar. En iyi çözüm, problemin sınırlamalarına uygun olarak problem durumları içerisinde minimum maliyetli yol olarak ifade edilir. Her karınca kötü maliyetli de olsa bir çözüm bulabilir. Yüksek- kaliteli çözümler ancak, eşzamanlı olarak farklı çözümler üreten bireylerin oluşturduğu koloni içindeki genel işbirliğinin sonucu olarak ortaya çıkar. Probleme bağlı belirlenmiş bir komşuluk bilgisine göre her karınca sınırlı sayıda komşu bölgeler boyunca hareket ederek çözüm üretir.

Karıncanın dahili bilgi bölgesi, karınca hakkında geçmiş zamandaki bilgileri depolar. Bu depolama, oluşturulmuş olan çözümün değerini/iyiliğini hesaplamada kullanılan yararlı bilgileri taşımak için kullanılabilir. Dahası çözümlerin uygulama kabiliyetini yönetmede temel rol teşkil eder. Kombinasyonel optimizasyon problemi gibi problemlerde bazı hareketler karıncayı uygun olmayan duruma getirebilir. Bu durumdan karıncanın hafızasından yararlanılarak kaçınılabilir. Böylelikle karıncalar uygun çözümleri yerel bölge hakkındaki bilgilerden ve bu bölge içinde yapılabilen hareketlerin etkilerinden elde ederek oluşturabilir.

Genel bilgi bütün karıncalar tarafından araştırma sürecinin başından itibaren biriktirilen ve feromon izi üzerinde kodlanmış bulunan, probleme özgü sezgisel bilgileri, ve temel bilgileri (knowledge) içerir. Karıncalar tarafından oluşturulan zamana bağlı genel feromon bilgisi, karıncaların kararlarını etkileyen, paylaşılabilen uzun dönemli hafızadır. Bu kararlar, karıncaların ortama feromon bırakıp bırakmamaları, problemin özelliğine bağlı olarak ne kadar feromon bırakmaları gerektiği gibi kararlardır. Karıncalar çözüm oluşturdukları sırada ya da çözüm oluşturduktan sonra uğradıkları bölgelere tekrar dönerek feromon bırakabilirler. Pozitif geri besleme, KKO algoritmasının işleyişinde önemli bir rol oynar. Bir hareketi ne kadar çok karınca seçerse, bu hareket o kadar çok feromon içerir ve böylece bu hareket daha sonraki karıncalar için daha ilgi çekici hale gelir. Genelde bırakılan feromon'un miktarı bir karıncanın oluşturmuş olduğu çözümün iyiliği ile orantılıdır. Bu yolla, eğer bir hareket yüksek kaliteli çözümün üretilmesine yardım etmiş ise bu hareketin iyiliği yaptığı yardımla orantılı olarak arttırılacaktır.

Yerel olarak bulunan feromon ve sezgisel değerlerin, fonksiyonel bir birleşimi 'karınca karar tablosunu' oluşturur. Bu tablo araştırma uzayının en ilgi çekici bölgelerine doğru araştırmalarını yönlendirmek için karıncaların karar politikaları tarafından kullanılan olasılık tablosudur. Belli bir hareketi tercih etmedeki rastgele karar işlemi ve daha önce bahsedilen feromon buharlaşma mekanizması bütün karıncaların araştırma uzayında aynı bölgeye doğru sürüklenmesini önler. Bu rastgele işlemin seviyesi ve feromon izi içindeki güncellemelerin (update) gücü, durum uzayındaki yeni noktaları araştırma ve biriktirilmiş olan bilgileri kullanma arasındaki dengeyi sağlar. Eğer gerekli ve mümkünse karıncaların karar politikası probleme özgü parçalar eklenerek

zenginleştirilebilir. Karınca çözüm oluşturarak ve feromon bilgisi bırakarak görevini tamamladıktan sonra ölür yani sistemden çıkarılır.

Bütün KKO sezgiselliği yukarıda bahsedilen karıncaların jenerasyon, aktivite ve feromon buharlaşması gibi yerel perspektiflerinin yanı sıra global bilgileri de kullanır ve bazı ekstra bileşenler de içerebilir. Örneğin bir karıncanın davranışı ilave feromon bilgisi bırakılarak gözlenebilir. Yada karıncaların ürettiği tüm çözümleri inceleme prensibinden hareketle, probleme özgü yerel optimizasyon prosedürleri uygulanır ve ilave feromon bırakılabilir.

KKO algoritmasının 3 ana faaliyeti olan karınca jenerasyonu ve aktivitesi, feromon buharlaşması ve birey hareketleri bir çeşit senkronizasyona ihtiyaç duyabilir. Bu da aktiviteleri listeleme(schedule-activities) ile sağlanır. Genelde aktivitelerin tam manasıyla ardışık listelemesi özellikle global bilgiye herhangi bir anda kolayca ulaşılabildiği ve işlemlerin uygun şekilde senkronize edilebildiği dağıtılmamış (nondistributed) problemler için uygundur.

KKO sezgiselliğinin programında bazı bölümler ve davranışlar opsiyoneldir, yani birey davranışları isteğe bağlıdır. Ya da feromon'un ne zaman ve nasıl bırakılacağı gibi detayları gerektiren uygulamaya bağımlıdır.

Adaptif ve yardımlaşma doğasının bir neticesi olarak KKO algoritması, özellikle dağıtılmış rastgele problemler için uygundur. Örneğin haberleşme ve transportasyon ağları ile ilgili problemler aslında dağıtılmış olan ve sabit olmayan problemlerdir ve bazen temel teşkil eden değişkenliğin tam bir modelini elde etmek mümkün olmaz. Bunun aksine 'stigmergy' hem karınca haberleşme metodu olduğu hem de uzaysal olarak belirtildiği için, KKO algoritması bölgeler arası komşulukların büyük olduğu problemlerde çok verimli şekilde çalışamaz. Gerçekten de büyük komşuluklu bölgeleri ziyaret eden karıncalar çok fazla sayıda mümkün olan hareket seçenekleriyle karşılaşırlar. Bu yüzden bir çok karıncanın aynı bölgeyi ziyaret etme olasılığı oldukça azdır ve feromon izlerini kullanma yada kullanmama arasında bir fark yoksa aynı bölgeyi ziyaret etme olasılığı düşmektedir.

2.4. KKO Algoritması'nın Uygulamaları

Günümüzde KKO algoritmasının çok sayıda, farklı kombinasyonel optimizasyon problemlerinde başarılı uygulamalarını görmek mümkündür. Bu uygulamalar başlıca iki gruba ayrılabilir. Bunlar:

a) Statik kombinasyonel optimizasyon problemlerine uygulama,

b) Dinamik kombinasyonel optimizasyon problemlerine uygulama şeklindedir.

Statik problemlerde probleme ait karakteristikler başlangıçta verilir ve problem çözümü boyunca bunlar değişmez. Buna en temel örnek 'gezgin satıcı problemi' dir (Traveling Salesman Problem-TSP). Burada şehirlerin yeri ve aralarındaki mesafeler bellidir ve değişmezler. Bunun aksine dinamik problemler, değeri sistem dinamiği tarafından belirlenen bazı niceliklerin bir fonksiyonu olarak tarif edilir. Bu yüzden problem zamanla değişir ve optimizasyon algoritması da bu değişime ayak uydurabilecek kabiliyette olmalıdır. Bu problem türüne örnek olarak ise haberleşme sistemlerindeki yönlendirme (routing) problemi verilebilir.

2.4.1. Gezgin Satıcı Problemi

Karınca kolonisi optimizasyon (KKO) algoritmasının ilk uygulamasında gezgin satıcı problemi (TSP) test problemi olarak kullanılmıştır. TSP'nin seçilmesinin ana sebebi; TSP'nin karınca koloni tasarısının kolaylıkla uyarlanabileceği en kısa yolu bulma problemi olmasıdır. Çünkü TSP'de bu algoritmanın davranış açıklamaları çok fazla teknik terimlerle karmaşıklaştırılmamıştır ve böylece anlaşılması oldukça kolaydır.

TSP'nin en genel açıklaması şu şekilde yapılabilir. Şehirleri simgeleyen noktaları N ve bu noktaları birbirine bağlayan köprüleri de E ile gösterelim. d_{ij}, köprü(i,j)$\in E$ 'nin uzunluğudur yani i ve j şehirleri arasındaki mesafedir. Burada i,j$\in N$ 'dir. TSP, $G=(N,E)$ grafiği üzerinde minimum uzunluklu Hamiltonian dairesini (turunu) bulma problemidir. G grafiğinin Hamiltonian dairesi G'nin bütün $n=|N|$ noktalarına sadece bir kere uğrayarak elde edilen ve uzunluğu tüm köprülerin uzunluklarının toplamına eşit olan kapalı bir dairedir. Aşağıda karınca sistemi ile başlanarak TSP için kurulan KKO algoritmasına kısaca göz atılacaktır.

Karınca Sistemi (KS): Karınca sistemi (ant system) ilk KKO algoritmasıdır [2]. AS'nin asıl önemi çok sayıda başarılı ve ilginç uygulamaları bulmuş olan karınca algoritmasının prototipi olmasından kaynaklanmaktadır.

AS'de suni karıncalar problem grafiğinde bir şehirden başka bir şehire hareket ederek TSP'nin çözümlerini (turlarını) oluştururlar. Algoritma t_{max} tane iterasyon yapar (ileride bu t ile gösterilecektir). Her iterasyon esnasında m tane karınca olasılık karar (bölge geçişi) kurallarının uygulandığı n adet adım yaparak bir tur oluşturur. Pratikte karınca i noktasında iken gitmek için j noktasını seçer ve (i,j) bağlantısı tura eklenir.

Feromon yollarının güncelleme (update edilme) durumlarındaki farklılıklara dayanılarak üç AS algoritması tanımlanmıştır. Bu algoritmalar karınca-yoğunluğu (ant-density), karınca-miktarı (ant-quantity) ve karınca-devirli (ant-cycle) olarak adlandırılır. Karıncalar ilk ikisinde feromonlu çözüm oluştururken, üçüncüsünde ise turun tamamını oluşturduktan sonra bırakırlar. Belirli test problemleri üzerine yapılan çalışmalar karınca-devirli'nin performansının diğer ikisininkinden daha iyi olduğunu göstermiştir. Bu sebeple AS üzerine yapılan çalışmalar karınca-devirli'nin karakteristiklerinin daha iyi anlaşılma çabasına doğru yönlendirilmiştir.

Söylendiği gibi AS'de karıncalar turlarını tamamladıktan sonra geçtiği köprüleri sonraki karıncalar için daha çekici hale getirmek için geçtiği köprülerle ilgili olan feromon izi değişkenlerine feromon'larını bırakırlar ve sonra ölürler.

Problem çözümü sırasında karıncaların kazandıkları tecrübeyi yansıtmak için aynı esnada feromon izi bilgileri değiştirilir. Karıncalar ürettikleri çözümlerin kalitesi ile orantılı feromon miktarı bırakırlar. Bir karınca ne kadar kısa bir tur meydana getirirse, bu turu oluşturmak için kullandığı köprülere bıraktığı feromon miktarı da o kadar büyük olur. Bu iyi çözümlere doğru araştırmanın yönlendirilmesine yardımcı olur. Feromon buharlaşmasının ana rolü durgunluktan kaçınmaktır. Durgunluk, bütün karıncaların aynı turu yaptığı durumdur.

Tabu listesi olarak adlandırılan her karıncanın hafızası ziyaret edilmiş şehirleri depolar. Her k karıncası için bu hafıza i şehrinde bulunan o karıncanın ziyaret etmek zorunda olduğu şehirlerin grubunu tayin etmek için kullanılır. Böylece hafızadan yararlanmak

suretiyle bir k karıncası kesin bir durum uzayı grafiği ile mümkün olabilen sonuçları üretir. Ayrıca hafıza, karıncanın uğradığı köprülere tekrar gecikmeli bir şekilde feromon maddesi bırakması için aynı yola tekrar dönebilmesini sağlar.

i noktasının karınca-karar tablosu $A_i = [a_{ij}(t)]_{|N_i|}$ yerel feromon izi değerleri ve yerel sezgisel değerlerinin birleşimi ile bulunur. Şöyleki;

$$a_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \qquad \forall j \in N_i \tag{2.2}$$

burada $\tau_{ij}(t)$, t anında (i,j) köprüsü üzerindeki feromon izinin miktarıdır. $\eta_{ij} = 1/d_{ij}$, i noktasından j noktasına olan hareketin sezgisel değeridir. N_i , i noktasının komşularının grubudur. α ve β ise feromon izi ve sezgisellik değer ağırlıklarını kontrol eden parametrelerdir. k karıncasının t'inci algoritma iterasyonunda turunu oluştururken i şehrinden ($j \in N_i^k$) şehrine gitmeyi seçmesinin olasılığı:

$$p_{ij}^k(t) = \frac{a_{ij}(t)}{\sum_{l \in N_i^k} a_{il}(t)} \tag{2.3}$$

Burada $N_i^k \subseteq N_i$ ifadesi k karıncasının henüz gitmediği i noktasının komşuluğunda bulunan noktalar grubudur. N_i^k içindeki noktalar N_i grubu içinden karınca özel hafızası M^k kullanılarak seçilir.

α ve β parametrelerinin rolleri şöyle özetlenebilir; Eğer α=0 ise en yakın şehirlerin seçilmesi daha muhtemeldir. Bu klasik "stochastic greedy algoritmasına" benzer. Diğer yandan eğer β=0 ise feromon kuvvetlendirmesini sağlar. Bu metot durgunluk durumunun hızlı bir şekilde ortaya çıkmasına yol açar. Bu yüzden sezgisellik değeri ve iz yoğunluğu arasında karşılıklı etkileşim gerekmektedir.

Karıncaların tamamı turlarını tamamladıktan sonra bütün köprülerde feromon buharlaşması başlar ve sonra her k karıncası kullanmış olduğu her köprüye $\Delta\tau_{ij}^k(t)$ feromon miktarını bırakır.

$$\Delta\tau_{ij}^{k}(t)=\begin{cases}1/L^{k}(t) & if\,(i,j)\in T^{k}(t)\\ 0 & if\,(i,j)\notin T^{k}(t)\end{cases} \tag{2.4}$$

burada $T^k(t)$, t iterasyonunda k karıncası tarafından yapılan turdur ve $L^k(t)$ bunun uzunluğudur. Şunu vurgulamak gerekir ki; simetrik TSP köprüleri çift yönlü olarak farz edilir yani aslında aynı köprüler olan (i,j) ve (j,i) daima aynı zamanda güncellenir. Köprülerin farklı olduğu ve (i,j) , (j,i) köprüleri üzerindeki feromon izi seviyesi farklı olan asimetrik TSP'de durum farklıdır. Bu durumda, i noktasından j noktasına karınca hareket ettiği zaman sadece (i,j) köprüsü güncellenir, (j,i) köprüsü güncellenmez.

Eşitlik (2.4)'de görüldüğü gibi $\Delta\tau_{ij}^{k}(t)$ değeri karıncanın işini nasıl bir şekilde yerine getirdiğine bağlıdır yani yapılan tur ne kadar kısa olursa bırakılan feromon miktarı o kadar çok olmaktadır.

Pratikte, karıncalar tarafından yeni feromon ilavesi ve feromon buharlaşması bütün kollara uygulanan aşağıdaki kuralla yerine getirilir.

$$\tau_{ij}(t)\leftarrow(1-\rho)\tau_{ij}(t)+\Delta\tau_{ij}(t) \tag{2.5}$$

Burada $\Delta\tau_{ij}(t)=\sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t)$, m her iterasyondaki karıncaların sayısı ve $\rho\in(0,1]$ ise feromon izi zayıflama katsayısıdır. Başlangıç feromon miktarı $\tau_{ij}(0)$ bütün kollardaki aynı büyüklükteki τ_0 pozitif sabit değerine ayarlanır ve α, β, ρ parametreleri sırasıyla 1, 5, 0.5 değerinde iken karıncaların toplam sayısı $m=n$ olarak ayarlanır, bu durum Dorigo tarafından deneysel olarak en iyi durum olarak belirtilir.

Nispeten daha küçük TSP problemlerinde (30 ila 75 şehir içeren) AS, diğer genel amaçlı sezgisel tekniklerle karşılaştırılmıştır ve sonuç hem çok ilginç hem de çok hayal kırıklığına uğratıcı olmuştur. 30 şehirlik problem olan Oliver30 için AS genetik bir algoritmanın bulduğu en iyi çözümü bulabilmiştir ve karşılaştığı diğer genel amaçlı sezgisel tekniklerle aynı ya da daha iyi bir performans sergilemiştir [20]. İyi çözümlere karşı çabuk yakınsama sergilemesine rağmen malesef, büyük boyutlu problemler için müsaade edilen 3000 iterasyon içinde beklenen sonuçlara asla ulaşılamamıştır. Bu sonuçlar çok sayıda bilim adamını KKO optimizasyon uygulamaları üzerine daha fazla çalışmaya teşvik etmiştir ve bu etki birçok başarılı uygulamalarla sonuçlanmıştır [2].

2.4.2. Haberleşme Ağlarında Yönlendirme için KKO

Haberleşme ağlarında genel yönlendirme problemi, bazı ağ performans ölçümlerinin, ağ tipinin ve sağlanan servislerin fonksiyonlarının maksimize edilebilmesi amacına yönelik veri trafiğini yöneltmek için 'yol verme tablolarının' oluşturulması ve kullanılması problemi olarak tarif edilebilir.

Burada C, işlemsel noktalar grubu ve L, gerçek ağın haberleşme bağlantıları grubunu temsil etmektedir. Ayrıca G=(C,L) yönlendirilmiş grafiktir. Bu grafikte C grubunun içindeki her nokta bir ağ noktasına karşılık gelmekte ve L grubu içindeki her köprü bir yönlendirici iletim hattına (link) karşılık gelmektedir. Her link, kendisinin fiziksel özellikleriyle ve trafik akışını geçirmesiyle sınırlanan bir maliyet ölçümünü birleştirir. Ağ uygulamalarında, kaynaktan hedef noktalarına doğru veri akışı sağlanır. Ağdaki her nokta için, yerel yönlendirme elemanı, giren veriyi hedef noktalarına doğru yöneltmek için en iyi gidiş linkini seçmek için yerel yönlendirici tabloyu kullanır. i noktasının komşularının grubu N_i ile simgelenirse, genel bir i noktasının $R_i = [r_{ijd}]$ yönlendirme tablosu, i noktasına giren ve sonraki $j \in N_i$ noktasından geçerek d hedef noktasına doğru yönlendirilen veri paketlerini ifade eder. Yönlendirme tabloları iki boyutludur, çünkü i noktasına giren veri paketlerinin ilerleyebileceği komşu noktaların seçimi hedef d noktası paketinin bir fonksiyonudur. Karınca yönlendirme tabloları da aynı iki boyutlu yapıya sahiptir. Her bağlantıyla ilişkilendirilen feromon yolları N_C-1 değerlerinin vektörleridir. Yönlendirme için kullanılan bütün KKO uygulamalarında, karınca yönlendirme tabloları uygulama-bağımlı dönüşümler aracılığıyla yönlendirme tablolarının oluşturulmasında kullanılır. Bu vektörel feromon yolları TSP için kullanılan skaler yolların doğal uzantısıdır.

TSP uygulamaları ile diğer önemli farklılıklar, bu iki problemin farklı tabiatından kaynaklanır.(i) başlangıç-hedef noktalarının (s,d) çiftine her karınca atanır ve s-d arasında bir yol bularak her karınca ağdaki bütün (i,j) çiftleri arasındaki yollarla ilgili olarak problemin tam çözümünün sadece bir parçasının oluşturur. (ii) bağlantılarla ilgili olan maliyetler statik olarak tayin edilmez. Bu maliyetler bağlantıların fiziksel özelliklerine ve bağlantıların trafik geçişlerine bağlıdır.

AntNet: AntNet' de her karınca ağdaki nokta çiftleri arasındaki yolların minimum maliyetini belirler. Karıncalar her ağ noktasından trafik örneklerini karşılaştırmak için rastgele seçilen hedef noktalara dağıtılır. Her karıncanın bir s kaynak noktası ve bir d hedef noktası vardır ve her karınca d noktasına ulaşana kadar bir noktadan diğer bir noktaya hareket ederek s' den d' ye ulaşır. k karıncası i noktasında iken hareket etmek için sonraki j noktasını, karıncanın hafızası M^k ve yerel karınca yönlendirme tablosu A_i' nin bir fonksiyonu olan bir olasılık karar kuralına göre seçer.

Feromon yolları yine köprülerle birleştirilir fakat köprü hedef çiftleri ile ilgili olan değişkenlerle belirlenir. Yani yönlenmiş her (i,j) köprüsünün değerleri $\tau_{ijd} \in [0,1]$ olan N_C-1 tane yol vardır. Ayrıca her köprü bir final hedeften bağımsız olan sezgisel η_{ij} değeri ile bağlantılıdır. Sezgisel değerler şöyle ifade edilir.

$$\eta_{ij} = 1 - \frac{q_{ij}}{\sum_{l \in N_i} q_{il}} \tag{2.6}$$

Burada q_{ij}, i noktası ile komşusu olan j noktasını birleştiren linkin sıra (gönderilmeyi bekleyen bit sayısının) uzunluğudur.

AntNet'de yönlendirme problemleri için olan diğer çoğu KKO algoritmalarının uygulamalarında olduğu gibi, bunda da daemen bileşeni yoktur.

Yerel karınca yönlendirme tablosu A_i, yerel feromon yolları τ_{ijd} ve sezgisel değerler η_{ij} 'nin birleşik fonksiyonu ile elde edilir. Hedefe doğru yol oluştururlurken karıncalar verilere aynı link sırasını kullanarak hareket ederler. Bu yolla karıncaların yaşadıkları s noktasından d noktasına hareketleri esnasında oluşan geçen T_{sd} zamanı (veri paketleri gecikmeleri) yolun kalitesinin bir ölçümü olarak kullanılabilir. Baştan sona bir yolun iyiliği yolculuk zamanı T_{sd} ile ve her nokta için belirlenen yerel adaptif istatistiksel modelin sezgisel bir fonksiyonu ile değerlendirilebilir. Gerçekte yollar ağ durumuna bağlı olarak değerlendirilebilir. k karıncası bir yolu tamamladıktan hemen sonra, oluşturduğu yolun iyiliğiyle orantılı olan bir feromon miktarını ziyaret ettiği noktalara bırakır. Bu amaç için hedef noktasına ulaştıktan sonra karınca aynı yol boyunca tersine kaynak noktalarına doğru hareket eder (toplanan bilgilerin hızlı yayılımını sağlamak için öncelik sırasını belirler).

Bu, tersine yol boyunca (d' den s' e doğru) k karıncası s' den d' ye doğru hareketi sırasında daha önceden kullandığı her l_{ij} bağlantısının $\tau_{ijd}(t)$ feromon yolu değerini artırır. Feromon yolu yoğunluğu aşağıdaki kural uygulanarak artırılır.

$$\tau_{ijd}(t) \leftarrow \tau_{ijd}(t) + \Delta\tau^{k}(t) \tag{2.7}$$

Karıncanın dönüş yolculuğu boyunca feromon yollarını güncellemesinin sebebi şudur: karınca ziyaret edilen köprüler üzerine bırakmak için $\Delta\tau^{k}(t)$ feromon miktarını hesaplamadan önce, değerini tayin etmek için kaynaktan hedefe olan bir yolu tamamlamalıdır.

Ziyaret edilen köprüler üzerindeki feromon izleri güncellendikten sonra aynı i noktasının çıkan bütün bağlantıların feromon değerleri buharlaşır.

$$\tau_{ijd}(t) \leftarrow \frac{\tau_{ijd}(t)}{(1+\Delta\tau^{k}(t))}, \ \forall j \in N_i \tag{2.8}$$

$N_i = i$ noktasının koşullarının grubu söylendiği gibi i noktasının AntNet'i karınca yönlendirme tablosu $A_i = [a_{ijd}(t)]$, feromon yolu değerleri ve yerel sezgisel değerlerin birleşiminden bulunur.

$$a_{ijd}(t) = \frac{\varpi\tau_{ijd}(t) + (1-\varpi)\eta_{ij}}{\varpi + (1-\varpi)(|N_i| - 1)} \tag{2.9}$$

Burada $j \in N_i$, d hedef nokta $\varpi \in [0,1]$ ağırlık faktörü ve payda bir normalizasyon terimidir.

Karınca karar kuralı şu şekilde tarif edilir; t zamanında k karıncası i noktasına yerleştirilmiş ve d noktasına doğru yönlendirilmiş olsun, eğer $N_i \not\subset M^k$, yani eğer k karıncasının bulunduğu bölgedeki komşuluklarının içinde henüz ziyaret etmediği en azından bir şehir varsa, karınca bu $j \in N_i$, noktasını şu olasılıkla seçer.

$$p_{ijd}^{k}(t) = \begin{cases} a_{ijd}(t) & if(j \notin M^k) \\ 0 & if(j \in M^k) \end{cases} \tag{2.10}$$

Diğer yandan, karınca $j \in N_i$ şehrini düzgün olasılıkla seçer $(p_{ijd}^k(t) = 1/(|N_i|)$, Başka bir deyişle, karıncalar devirlerden kaçınmaya çalışır fakat i' nin komşuluğundaki bütün noktaların karınca tarafından ziyaret edilmiş olması durumunda karıncanın seçeneği kalmaz ve bir noktayı döngü meydana getirerek tekrar ziyaret etmek zorunda kalır. Bu durumda meydana getirilen döngü karınca hafızasından silinir [1].

BÖLÜM 3

DİSK PLANLAMA PROBLEMİ

3.1. Giriş

İşlemci hızı ve hafıza kapasitesinin disk hızından çok daha hızlı bir gelişim içinde bulunduğu göz önüne alınırsa, disk planlama problemi üzerinde teorik ilgi ve uygulamanın önemliliği artmaktadır. Teknolojinin gelişimiyle disk hızında önemli bir artış sağlanamaması, disk kullanımını daha etkin yapan metotların geliştirilmesini gerekli kılmıştır. Buna rağmen disk kullanımı algoritmalarında yapılan çalışma sayısı yeterli seviyeye ulaşmamıştır.

Bir disk sürücüsü üzerinde dairesel izler bulunduran ve sabit bir açısal hızla dönen dairesel plakalardan oluşur. Belli bir veriyi tutan izler, birbiri arkasına sıralı veya disk üzerine birbirinden uzak yerlere dağıtılmış durumda olabilirler. Disk kafası izler arasında hareket ederek disk plakası üzerindeki veriyi toplar. Disk kafasının istenen verileri bulunduran izlere ulaşması için temel olarak iki çeşit gecikme olur. Bunlar arama zamanı ve dönme gecikmesidir. Arama zamanı, disk kafasının istenen ize ulaşması için geçen süre, dönme gecikmesi ise bu iz üzerindeki istenen sektörün disk kafasının altına gelmesi için geçen süredir.

Disk istekleri ile ilgili belli bilgileri (parametre) dikkate alan bazı planlama algoritmaları, daha yüksek performans sağlamak üzere geliştirilmiştir. Disk planlamada bu disk isteği bilgilerinden "arama zamanı" ana parametre veya genelde tek parametre olarak alınmıştır. "Dönme gecikmesi" "arama zamanı"'ndan çok daha düşük değerlerde olduğu için çoğu algoritmada ihmal edilmektedir. Disk isteği ile ilgili diğer bazı önemli parametreler ise istek önceliği, son işlem zamanı ve istek tipidir.

Bu bölümde disk planlama problemin tanımına geçmeden önce disk işleyişinin donanımsal ve yazılımsal boyutu ele alınacak daha sonra probleme getirilen klasik teknikler anlatılarak bir örnek üzerinde açıklanacaktır.

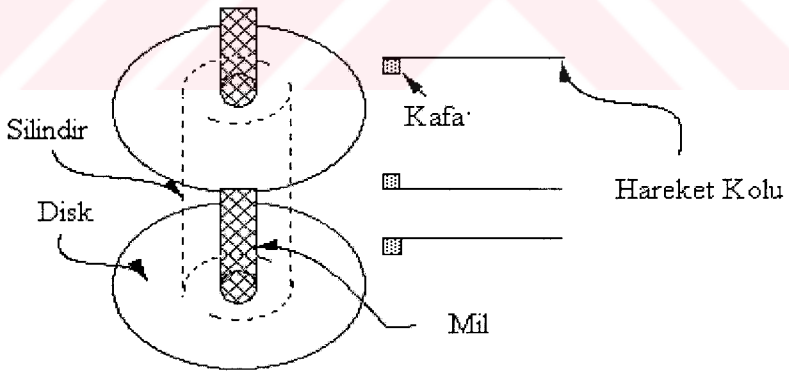
3.2. Disk İşleyişinin Donanımsal Boyutu

3.2.1. Disk Yapısı ve Bölümleri

Modern disk sürücüleri disk işlemlerini kontrol etmek amacıyla bir işlemci ve hafıza taşırlar. Sürücü aynı anda gelen birçok isteği kabul edebilme, bunları sıralama ve bu sırada bunları işleme özelliğine sahiptir. Bu özellik, istenen bütün veriler için gerekli olan toplam kafa hareketini en aza indirgemeye yarar.

3.2.1.1. Disk Yapısı

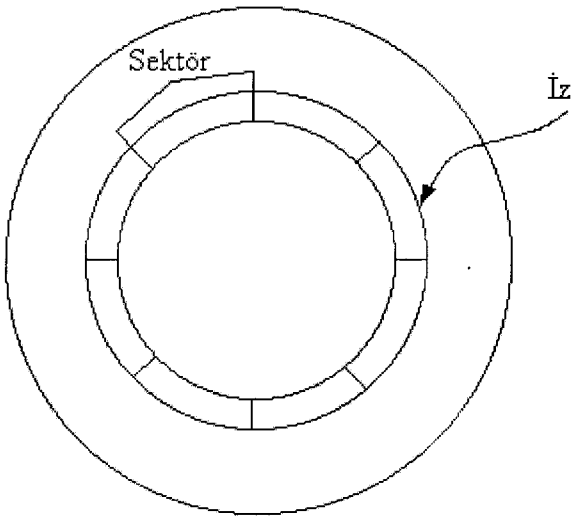
Bir harddisk fiziksel olarak bir mil üzerinde manyetik bir maddeyle yüzeyi kaplanmış disklerden oluşur. Milin dönmesiyle birlikte diskler arasında bulunan kafaların hareketiyle okuma veya yazma işlemi yapılır.



Şekil 3.1. Diskin fiziksel yapısı

3.2.1.2. Disk İzleri, Silindirleri ve Sektörleri

Bir disk izlere, silindirlere ve sektörlere bölünmüştür. Bir iz, bir kafanın disk üzerinde diskin dönmesiyle hareket ettiği alandır. Bu alan bir bit genişliğinde ve n bit uzunluğundadır. Bir silindir bütün kafaların birlikte gördüğü aynı hizadaki izlerin topluluğudur. Silindirler disk merkezine olan uzaklıklarına göre gruplandırılırlar. İzler, temel bir saklama birimi olan sektörlere bölünmüşlerdir.



Şekil 3.2. Disk plakasının fiziksel yapısı

Sun sisteminde bir sektör, başlık ve devam bilgisiyle birlikte 512 byte (1 disk bloğu)' dır. Buradaki devam bilgisiyle sektörlerin birbirleri arkasına birleştirilmesi ve bu birliktelik üzerine verilerin yazılması mümkündür. Kontrol mekanizması sektörleri izleyerek hataları görebilmekte ve bunları gerektiğinde düzeltebilmektedir. Kontrol mekanizmasına bağlı olarak bir disk sektörünün yapısı değişmektedir. Şekil 3.3. bize bu yapı hakkında bir fikir vermektedir.

Burada iki adet başlangıç ve bitiş alanları mevcuttur. Bu alanların büyüklükleri dönme hızı gibi sabitlere bağlı olarak değişebilir. Boşluk alanı kafanın hangi pozisyonda olduğunu belirtmek için, ECC alanı ise hata düzeltmeleri için kontrol mekanizmasına bilgi vermektedir.

İz üzerindeki sektör sayısı disk üzerinde bulunan izin çapı ile orantılıdır. En dıştaki iz en büyük olandır ve daha içteki olanlara nazaran daha fazla sektör içerir. Açısal hızın aynı olması nedeniyle dış bölgelerde bulunan izlerden birim zamanda daha fazla sektör okunması mümkündür. Disk blokları en dıştaki izden başlanarak numaralandırılmışlardır. Dolayısıyla ilk veriler en dıştan başlanarak yazılırlar.

Başlangıç 1	Başlık	Boş	Başlangıç 2	Boş	Veri Alanı	ECC	Bitiş
25 byte	8 byte	1 byte	25 byte	1 byte	512 byte	6 byte	22 byte

Şekil 3.3. Sektör yapısı

3.2.1.3. Silindir Grupları

İşletim sistemlerince hızlı bir dosya sisteminin gerçekleştirilmesi için silindirler gruplandırılırlar. Sun işletim sisteminde Berkeley hızlı dosya sistemi uygulanarak 32'li veya 16'lı gruplar oluşturulur. Her bir silindir grubu süper bloğun bir kopyasını içerir. Süper blok, indeks noktalarını (inode), elverişli bloklar listesini ve silindir içerisindeki veri bloklarının kullanım listesini içerir. Aynı dosyanın veri blokları, dönme gecikmelerini (rotational delays) en aza indirgemek amacıyla birbirlerine yakın bloklar içerisine konmaya çalışılırlar. Silindirleri bu şekilde gruplayarak bir dosyaya erişmek için ortalama olarak gerekli kafa hareketi düşürülmeye çalışılır. Dosyayı ve dosya verilerini tanımlayan indeks noktası bilgisi de diskin aynı fiziksel bölgesinde olmalıdır. Birden fazla süper blok olması ise gereksiz manada yer israf edildiği anlamında düşünülmemelidir, çünkü asıl süper blok bozulduğunda bu bilgi yedek süper bloklardan alınabilmektedir.

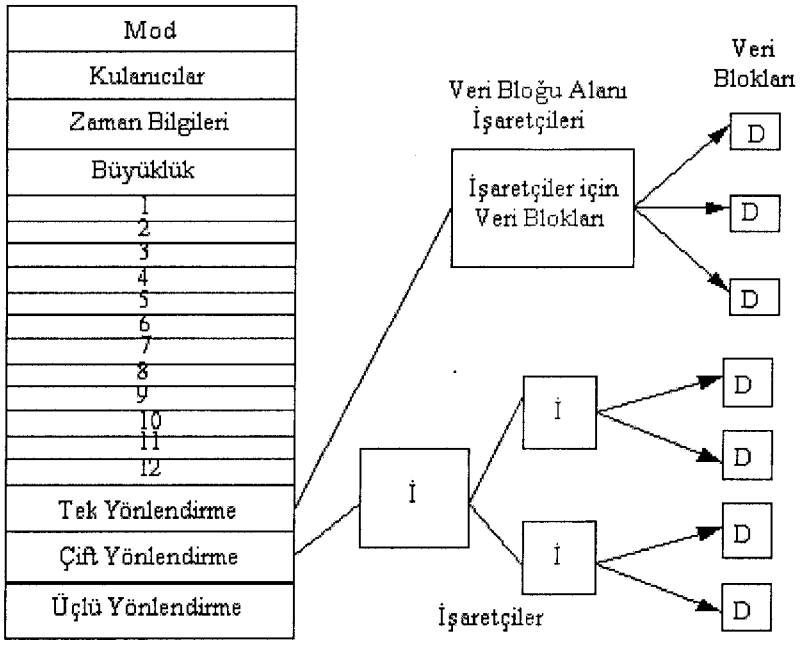
Unix dosya sisteminde bir dizi silindir grubu oluşturulmaktadır. Bunlar süper blok, silindir grubu özet bloğu, indeks noktaları dosyası ve veri bloğu alanını ihtiva eder. İndeks noktaları dosyası disk üzerindeki dosyaların yerini göstermektedir. İlk süper blok işlemlerde kullanılan süper bloktur, diğer silindirlerde bulunan süper bloklar yedek amaçlıdır. Silindir grubu özet bloğu aşağıdaki verileri tutmaktadır:

- dosya sisteminin büyüklüğü,
- indeks noktaları ve veri blokları sayısı ,
- kullanılan son blok ve indeks noktası işaretçileri,
- elverişli blok numaraları,
- indeks noktası haritası,
- boş indeks noktası haritası.

Her veri bloğu normalde 8192 byte uzunluğunda olup, 1024 byte uzunluğunda 8 bölüme ayrılmıştır.

İndeks noktaları her bir dosya için oluşturulmuşlardır. Her bir indeks noktası bloğu kullanıcı bilgisi; oluşturulma, değiştirilme ve erişim zamanları bilgisi; donanım bağlantı bilgileri ve o dosya için veri bloğunun yerini içermektedir. İndeks noktası bilgisi içerisinde dosyanın ismini bulundurmaz. Bu bilgi dizin tablosunda yer almaktadır [3].

Şekil 3.4. Mantıksal yapıda disk içeriği



Şekil 3.5. İndeks noktaları blokları içeriği

3.3. Disk İşleyişinin Yazılımsal Boyutu

3.3.1. Dosya Kavramı

Bilgisayarlar bilgiyi manyetik diskler, manyetik teypler ve optik diskler gibi farklı depolama aygıtlarında tutarlar. Bilgisayar sisteminin kullanımının kolaylaştırılması için işletim sistemi, bilgi depolamaların mantıksal bir çerçeveden bakış açısını sunar. İşletim sistemi fiziksel bir depolama aygıtında bulunan bilginin mantıksal bir bölümünü simgeleyen kısmın "*dosya*" olarak görüntülenmesini sağlar. Dosyalar fiziksel aygıtlara işletim sistemince yerleştirilmiş nesnelerdir. Bahsi geçen depolama aygıtları genelde "*kalıcılık*" özelliği taşırlar. Bu özellik sayesinde güç kesilmeleri veya yeniden sistem başlatmaları gibi durumlara karşı üzerlerindeki bilgiyi bozulmadan tutabilirler.

Dosyanın işlevli bir şekilde oluşturulabilmesi için dosyalar üzerine uygulanabilecek operasyonları tanımlamamız gerekmektedir. Bu tür operasyonlar oluşturma, yazma, okuma, yer değiştirme, silme ve boşaltma işlemleridir. Bunlar işletim sisteminin dosyalar üzerinde gerçekleştirdiği altı temel işlemdir. Bunlara benzer dosyayı yeniden adlandırma gibi işlemlerde söz konusu olabilmektedir.

Dosya Oluşturma: Dosya oluşturma iki adımda gerçekleştirilir. İlk olarak, dosya sisteminde oluşturulacak dosya için yer bulunmalıdır. Sonra, dizin içerisine yeni dosya

için bir giriş bilgisi yerleştirilmelidir. Dizin giriş bilgileri dosyanın adını ve dosya sistemindeki yerini tutar.

Dosya Yazma: Yazma işlemi için dosya adını ve yazılacak veriyi alarak işletim sisteminin ilgili sistem çağırmasını çalıştırmamız gerekir. Verilen dosya ismiyle sistem, dosyanın yerini bulmak için dizini araştırır.

Dosyadan Okuma: Okuma işlemi için dosya adını ve dosyadan okunacak bilgiyi, ana hafızada hangi adrese koyacağımızı belirten parametreleri işletim sisteminin ilgili sistem çağırmasına vererek gerçekleştiririz. Yazma ve okuma işlemleri gerçekleşirken dosyada son işlemden sonra kalınan pozisyonu göstermek için işaretçiler kullanılır.

Dosya İçerisinde Yer Değiştirme: Okuma ve yazma işlemleri için kullanılan işaretçilere yeni değer atama işlemidir.

Dosyayı Silme: Dosyayı silmek için verilen isimdeki dosyayı dizinde araştırırız. Dosyayla ilgili dizin girişini bularak bu dosyanın kullandığı alanı ileride başka dosyaların kullanması için serbest bırakırız ve dizin girişi bilgisini sileriz.

Dosyayı Boşaltma: Dosyayı silmek yerine içinde yazılı olan bilgiyi silerek dosyanın diğer özelliklerini muhafaza etmesini istiyorsak bu işlemi gerçekleştiririz.

Bu altı temel operasyon dosya işlemleri için gerekli olan minimum operasyon komut kümesini oluşturur. Burada bahsi geçen benzer işlemler dizinler için de geçerlidir.

Diskler dosya sisteminin tutulduğu ikincil bir depolama alanıdır. Ana depolama alanı hafızalar (RAM) çalışmakta olan programları ve dosyaları tutmayı sağlar ve işlemci ile birebir çalışma içerisinde bulunur. Disk I/O verimliliğini artırmak için disk ve hafıza arasındaki bilgi transferi bloklar ve bunları oluşturan üniteler şeklinde olur. Her blok bir veya daha fazla sektörden oluşur. Disk sürücüsüne bağlı olarak sektörler 32 byte'dan 4096 byte'a kadar değişir. Genelde sektörler 512 byte'dan oluşur. Çok sayıda dosyanın kullanışlı bir şekilde tutulması için diskler iki önemli özelliğe sahiptirler:

(a) **Tekrar Yazılabilirlilik:** Diskten bir dosya okuyup bunun üzerinde değişiklikler yapmak ve bunu tekrar aynı bölgeye yazmak mümkündür.

(b) **Direk Erişilebilirlilik:** Disk üzerindeki bilgi bloğuna direk olarak erişmek mümkündür. Bu sayede sıralı veya rasgele bir dosyaya erişme yada farklı bir

dosyaya erişme işlemi, okuma yazma kafasının hareketi ve diskin dönmesi için beklenmesi suretiyle gerçekleşir.

Yukarıda belirtilen dosya işlemlerinin gerçekleştirilmesi için öncelikle dosyalara ait verilerin disk üzerinde nerelerde tutulacağına karar verilmesi gereklidir. İşletim sisteminin bir görevi olan verilerin disk üzerindeki yerlerin tayini, ayırma metotları başlığı altında üç gruba ayrılmaktadır.

Bir dosyaya ait bilgileri toplamaya çalışan disk kontrol mekanizması, disk kafasını kullanılan bu metotlara bağlı olarak hareket ettirir ve dosya ile ilgili bütün verileri disk hafızasına topladıktan sonra dosya isteğinin gelmiş olduğu işletim sistemine bu verileri gönderir. Bu metotlardaki performans, direk olarak dosya bilgilerinin mümkün olduğunca dağıtık yerlerde olmamasının sağlanmasıyla elde edilebilmektedir. Bir dosyaya ait bilgilerin toplanması için disk kafasının disk üzerinde ne şekilde bir hareket yapacağı ayırma metotlarının incelenmesiyle anlaşılabilir.

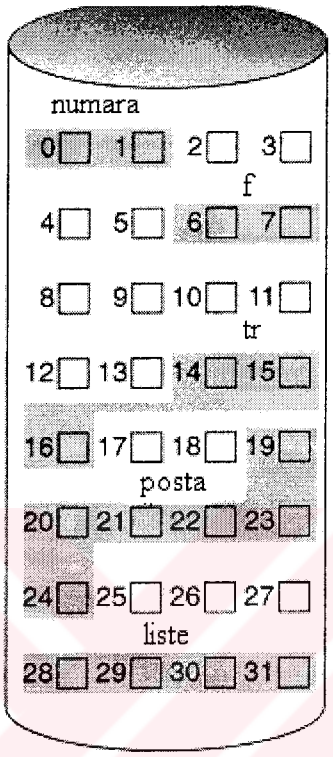
3.3.2. Ayırma Metotları

Disklerin direk erişim özelliği dosya uygulamalarında bize esneklik sunar. Hemen hemen her durumda birçok dosya aynı disk üzerinde tutulabilir. Burada ana problem verimliliği artırmak ve hızlı erişimi sağlamak için disk alanlarında ayrıştırma işleminin nasıl olacağıdır. Çoğunlukla kullanılan üç ana ayırma metodu vardır. Bunlar "ardışık", "bağlı" ve "indeksli" yapılardır. Bu metotların her biri kendine özgü avantaj ve dezavantajlara sahiptir. Bazı sistemler bu metotların üçünü birden kullanırken, çoğu sistemler bunlardan sadece birini kullanırlar.

3.3.2.1. Ardışık Ayırma

Ardışık ayırma metodu, her dosyanın disk üzerinde ardışık bir şekilde blokların tutulmasını gerektirir. Disk adresleri disk üzerinde doğrusal bir sıralamayı tanımlar. Bu tür bir sıralamayla, disk kafası hareketi olmadan blok b'den sonra blok b+1'e geçilerek bir dosya işlemi gerçekleştirilebilir. Kafa hareketi, ancak bir silindirin son sektöründen bir sonraki silindirin ilk sektörüne geçiş olduğunda olur. Bu geçiş bir iz olarak adlandırılır. Bu sayede sıralı ayırmalı dosyalarda disk araştırması (dosya bilgilerinin elde edilebilmesi için disk kafasının hareketi) minimum düzeydedir. Disk arama zamanı

bu işlem için geçen süredir. Ardışık ayırmalı dosya sistemini kullanan sistemlere, IBM VM/CMS işletim sistemi örnek olarak verilebilir.



dizin

dosya adı	baş	uzunluk
numara	0	2
tr	14	3
posta	19	6
liste	28	4
f	6	2

Şekil 3.6. Ardışık ayırmalı disk alanı

Bir dosyanın ardışık ayrılması, blok birimleri halinde disk adresi ve uzunluğuyla tanımlanır. Eğer dosya n blok uzunluğunda ve b pozisyonundan başlıyorsa, bu dosya $b, b+1, b+2, \ldots, b+n-1$ bloklarını kullanır. Her bir dosya için dizin girişi bu dosya için başlangıç bloğunun adresini ve uzunluğunu belirtir. Şekil 3.6.'da örnek dizin giriş bilgileri yer almaktadır.

Ardışık ayrılmış dosyaya erişim kolaydır. Sıralı erişimde, dosya sistemi en son bloğun referans alındığı disk adres bilgisini kullanarak bir sonraki bloğa erişim yapar. Blok-b 'den başlayan bir dosyanın blok-i 'sine yapacağımız doğrudan (direk) erişim için blok-b+1' erişim söz konusu olur. Dolayısıyla sıralı ve doğrudan erişimin her ikisi de ardışık ayırmada desteklenir.

Ardışık erişimde karşılaşacağımız güçlük yeni bir dosya için bulmamız gereken alandır. Boş alan bulma işlemi "boş alan yönetimi" sistemince gerçekleştirilir. Buradaki yönetim

sistemi, kullanılan tekniğe göre yavaş veya hızlı olabilir. Ardışık ayırmalı sistemde boş alan bulma işlemi diğer metotlara göre daha yavaş gerçekleşir. Bu metotda bir diğer problem ise oluşturulan bir dosyanın büyüklüğünün tahminidir. Eğer dosya büyüklüğü çok büyük tahmin edildiyse alan israf edilmiş olur, eğer küçük tahmin edildiyse dosya bir yerden sonra bu alana yerleşemeyeceği hatasıyla karşılaşır ve işlem sonlandırılır. Sonuçta dosya büyüklüğünün gerçek değerinden büyük bir değerde tahmin yapılması ve buna göre yer ayrılması gerekir. Bu ise Şekil 3.6.'da görüleceği üzere dosyalar arasında çok miktarda küçüklü büyüklü alanların israf edilmesi anlamına gelir.

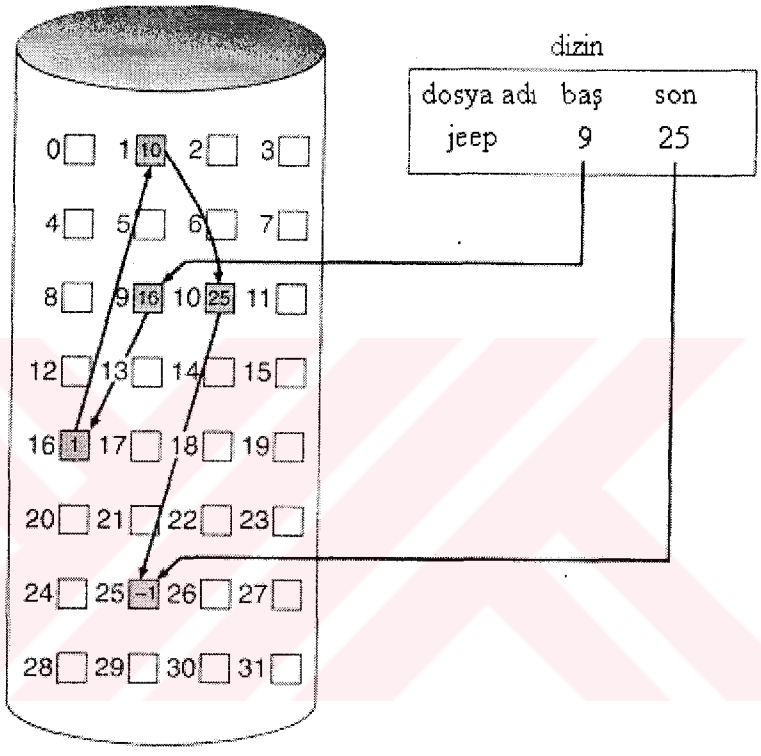
3.3.2.2. Bağlı Ayırma

Bağlı ayırma, ardışık ayırmadaki problemleri çözer. Bağlı ayırma ile disk üzerinde her hangi bir yerde bulunabilecek disk bloklarının birbirine bağımlı listesiyle oluşturulmuş yapıda dosyalar meydana getirilir. Dizin, dosyanın ilk ve son bloğunun yerini tutan bir işaretçi ihtiva eder. Örneğin, Şekil 3.7.'de görüldüğü üzere, beş blokluk bir dosya, blok 9'dan başlayabilir, devamında blok 16'da, blok 1'de, blok 10'da ve son olarak da blok 252'ye yerleşmiş olabilir. Bu işaretçiler kullanıcı tarafından görülmez. Eğer her blok 512 byte'dan oluşmuşsa ve disk adres bilgisi (işaretçi) 4 byte'a ihtiyaç duyuyorsa, kullanıcı blokları 508 byte olarak görecektir.

Yeni bir dosya oluşturmak için basit anlamda dizine yeni bir giriş yapmamız yeterlidir. Bağlı ayırmada her dizin girişi dosyanın ilk bloğunu gösteren bir işaretçiye sahiptir. Bu işaretçi başlangıçta boş dosyayı göstermek için liste sonu işaretçi değeri olan "nil" değerini içerir. Ayrıca, büyüklüğü gösteren alan da 0 değerine sahiptir. Dosyaya yazma işlemi boş alan yönetimince bulunan boş bloğun yazılmasını ve dosya sonuna bunun bağlanmasını sağlar. Dosya okuma ise işaretçilerin gösterdiği blokları bir bir okuma ile gerçekleşir.

Bağlı ayırmada, bahsedilen bir önceki metotta olduğu gibi alan israfı problemi yoktur. Çünkü herhangi bir boş blok devamındaki boş blokların sayısına ve büyüklüğüne bakılmaksızın listeye işaretçiler vasıtasıyla eklenebilir. Bir dosya diskte boş bloklar oldukça genişleyebilecek özelliktedir. Buna bağlı olarak, bağlı ayırma motodu dezavantajlara sahiptir. Ana problemi bu metodun sadece sıralı-erişimli dosyalarda verimli bir şekilde kullanılabilmesidir. Bir dosyanın *i.* bloğunu bulmak için dosyanın başlangıcından başlamamız ve *i.* bloğa kadar işaretçileri takip etmemiz gerekir.

İşaretçiye her bir erişim, bir disk okuması ve muhtemelen bir disk araması gerektirir. Sonuç olarak bağlı ayırmalı dosyalar için doğrudan erişim pek verimli değildir. Bir diğer dezavantaj işaretçiler için ihtiyaç duyulan alandır. Eğer bir işaretçi 512 byte'lık bir blokda 4 byte'lık bir alan kullanıyorsa, disk alanının % 0,78'inin veri alanı yerine işaretçi alanlarına tahsisi anlamına gelir.



Şekil 3.7. Bağlı ayırmalı disk alanı

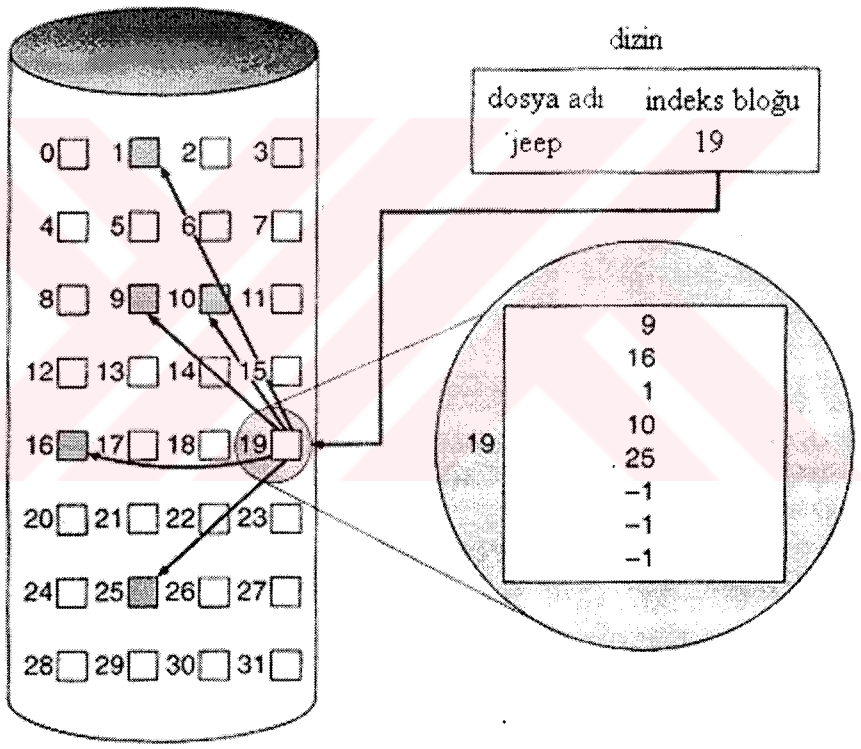
Bu problem için, blokları yapıştırarak (kümeler oluşturarak), bloklar yerine kümeleri ayrıştırarak genel bir çözüme gidilebilir. Böyle bir çözümle her bir blok yerine kümelere (cluster) işaretçiler atanarak işaretçilerin kullandığı alan yüzdesi düşürülebilir. Kümeler diğer metotlara göre disk erişim zamanında iyileştirme yaptığı için birçok işletim sistemince kullanılırlar.

Bağlı ayırmalı dosyalarda bir diğer problem ise disk üzerine yoğun miktarda dağıtılmış olan işaretçilerden herhangi birinin bozulması sonucu dosyanın geri kalanının tamamen bozulmasına neden olabilmesidir. MS-DOS ve OS/2 işletim sistemlerinde bağlı ayırma metodunun bir türü olarak (dosya ayrıma tablosu, FAT) yukarıda bahsedilen tekniklerle birlikte kullanılmaktadır. Diskin başlangıcında her bölümün (partition) bir tablosu yer

almaktadır. Tablo her bir disk bloğu için bir giriş bilgisini ve blok numaralarının indeks bilgilerini içerir.

3.3.2.3. İndeksli Ayırma

FAT'in olmadığı durumlarda bağlı ayırma metodu verimli bir doğrudan erişimi desteklemez. Çünkü blok işaretçileri, yine kendi içerlerinde bütün bir disk boyunca yayılmışlardır ve bir blok erişimi söz konusu olduğunda bütün bu işaretçilere sırayla erişmek gerekmektedir. İndeksli ayırma bu problemi bütün işaretçileri bir yerde (indeks bloğunda) toplayarak çözmektedir.



Şekil 3.8. Disk alanının indeksli ayırması

Her dosya disk blok adreslerinin dizisi olan bir indeks bloğuna sahiptir. İndeks bloktaki *i.* giriş dosyadaki *i.* bloğu işaret eder. Dizin indeks bloğunun adresini tutar (Şekil 3.8.). *i.* bloğu okumak için *i.* indeks bloğundaki işaretçi kullanılır.

Dosya oluşturulduğu zaman indeks bloktaki bütün işaretçilere "nil" değeri atanır. *i.* blok ilk defa yazılacağı zaman boş-alan yönetiminden alınan boş bir bloğun adresi *i.* indeks bloğu girişine yazılır [4].

3.4. Disk Planlaması

Bir disk aynı hizada hareket eden kafalardan ve belli sayıda silindirden oluşmaktadır. Silindirler en dıştan içe doğru izler topluluğundan meydana gelmektedir. Her bir iz ise belli sayıda sektörler içermektedir. Diske yazma veya okuma işlemi için yapılan her başvuru bir istek olarak adlandırılır. $< r >$ disk isteği r silindirine disk kafasının hareket etmesini istemektedir. *r1* ve *r2* silindirleri arasındaki mesafe $|r_1 - r_2|$'dir. Bir istek sırası $\alpha =< r_1, r_2, ... r_n >$, servis yapılmayı bekleyen diğer bir ifadeyle belirtilen noktalara kafa hareketinin gerçekleşmesini bekleyen bir diziyi ifade eder. Böyle bir dizi normalde ilk gelen ilk çıkar (FIFO) mantığıyla çalışır ve dizinin elemanları kuyruk diye adlandırdığımız bir kaydedicide tutulur. İstek kuyruğunun büyüklüğü(n), uygulamaya ve disk yapısına bağlıdır [5].

BOŞ	BOŞ	R_{n-2}	....	R_3	R_2	R_1
n	n-1	n-2		3	2	1

Şekil 3.9. Kuyruk yapısı

Tek disk kafalı örnek bir disk modelinin parametreleri Tablo 3.1'de verilmiştir.

Tablo 3.1. Disk parametreleri

Parametre	**Anlamı**	**Değeri**
İz	Disk Üzerindeki İz sayısi	1000
Arama Faktörü	Arama Zamanını Gösteren Faktör	0,6 ms
Disk Sabiti	Dönme Gecikmesi + Transfer Zamanı	15,0 ms

Disk istek kümesine servis yapılırken, harddisk kafasının daha önceden mevcut bulunduğu pozisyon hesaba katılmalıdır. Mevcut kafa pozisyonundan n adet iz mesafeli bir disk isteğine kafa hareketi için erişim zamanı eşitlik 3.1 ifade etmektedir.

$$Erişim_n = Arama_n + Dönme\ \ Gecikmesi + Transfer\ \ Zamanı \qquad (3.1)$$

Bu modelde dönme gecikmesi ve transfer zamanı tek bir parametre olan disk sabitine indirgenmiştir. $Arama_n$, disk faktörüne ve iz numarasına bağlı olan doğrusal olmayan bir eşitlilikle ifade edilir.

$$Disk\ Sabiti = Dönme\ Gecikmesi + Transfer\ Zamanı \tag{3.2}$$

$$Arama_n = Arama\ Faktörü \times \sqrt{n} \tag{3.3}$$

$$Erişim_n = Arama\ Faktörü \times \sqrt{n} + Disk\ Sabiti \tag{3.4}$$

Fazla sayıda iz içeren diskler için arama zamanı ($Arama_n$) eşitlik 3.4 'de çok daha fazla önem kazanmaktadır. Böyle bir durumda disk sabiti ihmal edilebilir. Çoğu disk planlama algoritması ana parametre olarak sadece iz numarasını kullanmaktadır.

İşletim sisteminin görevlerinden birisi donanımı verimli bir şekilde kullanmaktır. Disk sürücüleri için bu, hızlı erişim zamanı ve yüksek disk bant genişliği anlamına gelmektedir. Erişim zamanı, eşitlik 3.4'den de görüleceği üzere arama zamanı ve disk sabiti olmak üzere iki ana unsurdan oluşur. Arama zamanı, istenen sektörü içeren silindire disk kolunun hareketi için gerekli süredir. Dönme gecikmesi, istenen sektörü disk kafasının altına getirmesi için diskin dönmesiyle geçen süredir. Disk bant genişliği birim zamanda diskten gönderilen veri sayısını ifade eder ve transfer zamanı bu özellik ile ilişkilidir. Erişim zamanı için disk I/O isteklerini uygun bir şekilde sıralayarak performans artışı sağlanabilir [6].

Sisteme gelen bir istek eğer sistem kaynakları boş ise cevaplandırılır, aksi halde cevaplanmayı bekleyen istekler kuyruğuna yerleştirilir. Bir isteğin cevaplanmasının ardından sistem, kuyrukta bekleyen istekler arasında bir seçim yapma şansına sahiptir. Gerek isteklerin kuyrukta sıralanması işlemi, gerekse de kuyrukta servis yapılmayı bekleyen isteklerin seçilmesi işlemi işletim sistemince veya disk aygıtı üzerindeki kontrol mekanizması ile yapılabilir. Kontrol mekanizması, disk üzerindeki motorları kontrol eden , veri alış verişini yapan , işlemci ve hafıza içeren bir yapıdır.

İstemci/Sunucu dağıtılmış sistemlerde I/O kaynakları verimli bir şekilde kullanılabilmeli, farklı kullanıcılara farklı önceliklerle bu kaynaklar tahsis

edilebilmelidir. Bu yüzden bu tür sistemlerde isteğin iz numarasına ilave olarak, gelen bir diğer parametre de ''istek önceliği'' dir [6].

Gerçek zamanlı uygulamalar, disk I/O isteklerinin belli bir zaman aralığında yanıtlanmasını isterler. Gerçek zamanlı istekleri kaçırmamak için ''son işlem zamanı'' kısıtlaması alınmalıdır. Gerçek zamanlı sistemlerde, isteğin son işlem zamanı önemli bir parametredir [7].

Yukarıda bahsedilen parametrelerden biri veya bir grubu disk planlama algoritmasında girdi olarak kullanılabilir. Algoritma, en iyi performansı üretecek uygun istek sırasını bulmayı amaçlamaktadır. Performans aşağıdaki kriterler dikkate alınarak değerlendirilir;

Ortalama Yanıt Zamanı: Bir isteğin kuyruğa geliş zamanı ile servis yapılmaya başlandığı zaman arasındaki süre o isteğin yanıt zamanıdır. Başka bir ifade ile isteğin kuyruğa geldiği an ile disk kafasının isteğin belirttiği ize gittiği an arasındaki zaman "yanıt zamanı" olarak nitelendirilir. Kuyruktaki bütün isteklerin yanıt zamanlarının ortalaması ise bir performans değerlendirme kriteri olarak alınabilir.

Ortalama Bekleme Zamanı: Bir isteğin kuyruğa geliş zamanı ile servis işleminin bittiği zaman arasındaki zaman dilimi o isteğin bekleme zamanıdır. Ancak arama zamanının, dönme gecikmesi ve transfer zamanının toplamından çok daha büyük olması nedeniyle bekleme zamanı ve yanıt zamanı hemen hemen birbirine eşit olmaktadır. Bu nedenle ortalama bekleme zamanı değerlendirmelerde pek kullanılmamaktadır.

Maksimum Yanıt Zamanı: Bir isteğin uzun süre kuyrukta beklemesi durumunda açlık (starvation) problemi ile karşılaşılmaktadır. Böyle bir durum, bir prosesin(süreç) uzun süre cevap vermemesine, kitlenmesine veya isteği oluşturan prosesin devamında gelecek olan proseslerinde uzun süre bekletilmesine neden olacaktır. İşletim sistemince sıkıntıya girilecek bir durum olduğu için bu problemden mümkün olduğunca kaçınmak gerekmektedir. Açlık probleminin ne derece olup olmadığını en büyük yanıt zamanlı olan istekten anlayabiliriz. Kuyruktaki isteklerden en büyük yanıt zamanlı isteğin yanıt zamanı ne kadar düşükse açlık kriteri açısından performans o derece iyi demektir. Yani kuyruk içerisindeki maksimum yanıt zamanı kriterinin en aza indirgeme çalışması, bizim için performans iyileştirilmesi anlamına gelecektir.

İşlem Zamanı (Overhead): Kuyruktaki isteklerinin servis yapılacak sıralamaya getirilmesi için planlama algoritmasının toplam harcadığı işlem zamanıdır.

Özetlenecek olursa planlama algoritması belli parametreleri girdi olarak kabul ederek yine belli değerlendirme kriterlerini kullanarak en uygun sıralamayı yapmaya çalışmaktadır.

Tablo 3.2. Parametre ve kriterler

Parametreler	**Kriterler**
İz numarası	Ortalama Yanıt Zamanı
Öncelik	Ortalama Bekleme Zamanı
En son işlem Zamanı	Maksimum yanıt zamanı /İşlem zamanı

3.4.1. Klasik Disk Planlama Teknikleri

Bu teknikleri iki grupta toplayabiliriz. Bunlardan ilki arama optimizasyonunu yaparken sadece iz bilgisi parametresini kullanmaktadır. İkinci grup ise bu parametreye ilave olarak istek önceliği veya son işlem zamanı gibi parametreleri kullanmaktadır. İlk grupta yer alan bilinen bazı algoritmalar konunun devamında açıklanmaktadır.

3.4.1.1. İlk Gelen İlk Alınır (First Come First Served -FCFS) Planlaması

Disk planlamanın en basit halidir, disk yükü hafif olduğu zaman tercih sebebidir. İstekler kuyruğa geliş sırasına göre alınmaktadır. Gerçek manada bir sıralama işlemi olmadığı için algoritmanın işlem zamanı çok düşüktür. Ancak önemli bir kriter olan ortalama yanıt zamanı oldukça yüksek değerdedir. Bunu bir örnek üzerinde görmeye çalışalım. Disk kuyruğunda aşağıdaki numaralı silindirlerdeki veri bloklarına erişmek isteyen I/O (Giriş/Çıkış) istekleri olsun;

98, 183, 37, 122, 14, 124, 65, 67

Bu isteklerin kuyruktaki bulunma sırası numaraların veriliş sırasında olsun. Eğer disk kafası başlangıcı itibariyle 53. silindirde bulunuyorsa, ilk hareket 53'den 98'e olacaktır. Sonra 183,37,122,14,124,65 ve son olarak 67 olacaktır. Bu durumda disk kafa hareketinin bir izden bir sonraki ize geçmesi için gerekli süre olan arama zamanlarının toplamı (iz bazında) :

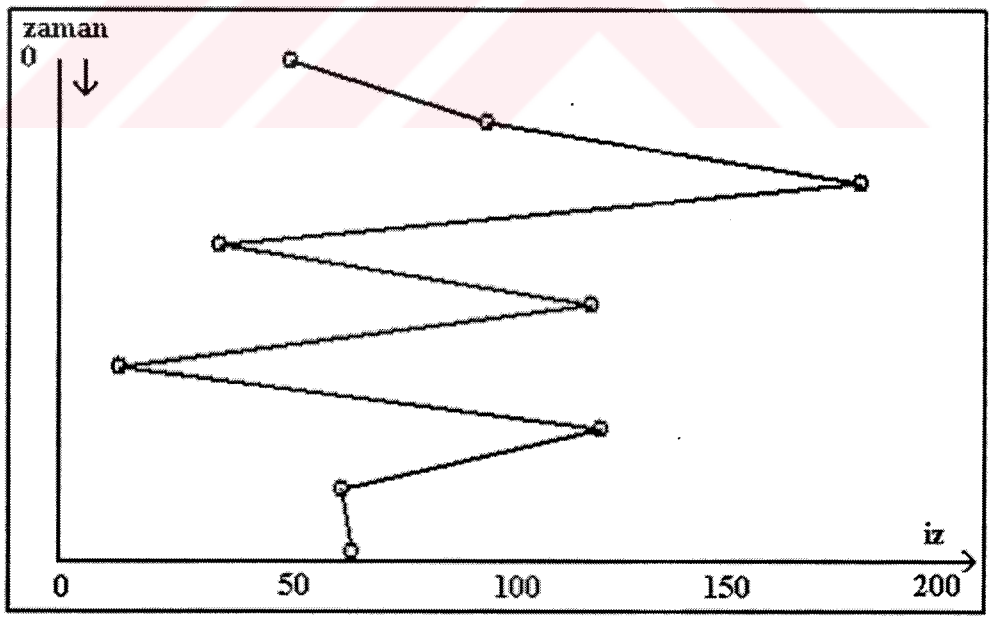
$$Arama = \sum_{i=1}^{n} | r_{i-1} - r_i | \qquad (3.5)$$

$Arama$ = |53-98 |+|98-183|+|183-37 |+ ...+|65-67| Toplam 640 iz olacaktır.

Bu değer, isteklere FCFS algoritmasıyla verilen sırada servis yapılırsa toplam kafa hareketi 640 iz mesafesine eşdeğer mesafe de olacağı anlamına gelmektedir. Disk kafasının hareket edeceği iz sayısı bilgisinden yararlanarak Eşitlik (3.3)'den bu hareketin ne kadar zaman alacağı elde edilebilir. Eşitlik (3.3) bir disk için örnek verilen bir eşitlik olup disk modeline göre değişmektedir.

Sonuç itibariyle disk kafasının toplam katettiği iz mesafesi ne kadar yüksek olursa istekler için ortalama yanıt zamanı da o kadar yüksek olacaktır.

Bu şekildeki bir planlama ile Şekil 3.10.'da görüleceği üzere 14'den 124'e gidip gereksiz mesafe katedilmiştir. Halbuki 37 ve 14 önce servis yapılıp daha sonra 122 ve 124 servis yapılsa idi kafa hareketi ciddi anlamda azalacak ve performans artacaktır.



Şekil 3.10 FCFS disk planlaması

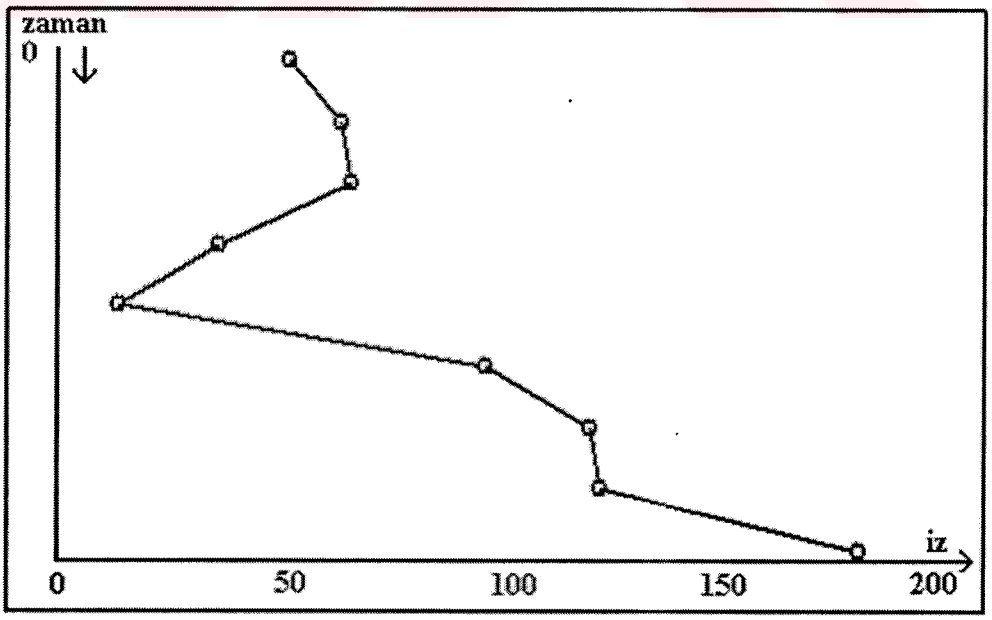
3.4.1.2. En kısa Arama Zamanlı Olan İlk Planlaması (Shortest Seek Time First – SSTF)

Daha uzak isteklere disk kafasını hareket ettirmeden önce kafanın mevcut bulunduğu pozisyona yakın noktalardaki isteklere servis yapmak daha mantıklı bir yaklaşımdır. Böyle bir kabul, en kısa arama zamanlı olan ilk (SSTF) algoritmasının temelini oluşturur. SSTF algoritması, mevcut kafa pozisyonuna en az arama zamanlı olan isteği seçer. Arama zamanı kafanın geçtiği silindir sayısı ile orantılı olduğu için SSTF, beklemede olan isteklerden kafanın mevcut pozisyonuna en yakın olanı alır.

Bir önceki algoritmada örnek verdiğimiz istek kuyruğunu ele alacak olursak, başlangıç pozisyonuna (53) en yakın silindir 65'dir. 65. silindire gelindikten sonra bir sonraki adımda en yakın silindir olan 67'e gidilir. Bu noktada 37. silindir 98. silindire nazaran daha yakın olduğu için 37. silindire hareket edilir. Devamında 14, 98, 122, 124 ve son olarak da 183. silindire gidilir (Şekil 3.11.). Bu metotla iz bazında toplam arama zamanı (eşitlik 3.5) 'den;

$$\text{Arama}=|53-65|+|65-67|+|67-37|+\ldots.|124-83|$$

toplam olarak 236 silindir veya iz elde edilir.



Şekil 3.11. SSTF disk planlaması

Bulunan değer bir önceki algoritma olan FCFS'ye göre üçte birinden daha az bir değerdir. Bu algoritma toplam yanıt zamanı veya ortalama yanıt zamanı performansında kayda değer bir artış sağlamaktadır.

SSTF planlaması, işlemcilerde kullanılan en kısa iş ilk (shortest job first-SJF) planlamasının bir türüdür. SSTF planlaması da bazı istekler için SJF planlamasında da görülen açlık (starvation) problemine yol açabilmektedir. Disk isteklerinin kuyruğa her an gelebileceği düşünülürse, kuyrukta bulunan 14 ve 186 silindir istekleri için bu algoritma 14'ü ilk servis için seçecektir. 14'üncü silindire hareket halinde iken 14'üncüye yakın bir istek geldiğini düşünelim. Bir sonraki adımda bu yeni isteğe servis yapılacaktır. 186 silindir isteği bekleyecektir. Bir sonraki adımda 186'ıncı silindirden daha yakın bir istek gelebilir. Böyle bir durumda 186'ıncı silindir isteği çok uzun süre kuyrukta beklemeye tabi tutulabilir. Bu örnekte de olduğu gibi mevcut kafa pozisyonuna çok uzak uç noktalardaki silindir isteklerinin uzun süre bekletilmeleri bu algoritmayla mümkündür. Eğer kuyruk uzunluğu büyük ise bu durumla karşılaşılma ihtimali daha da yüksektir. Bu durum daha öncede bahsedildiği üzere "açlık" olarak adlandırılmaktadır.

3.4.1.3. Tarama (SCAN) Planlaması

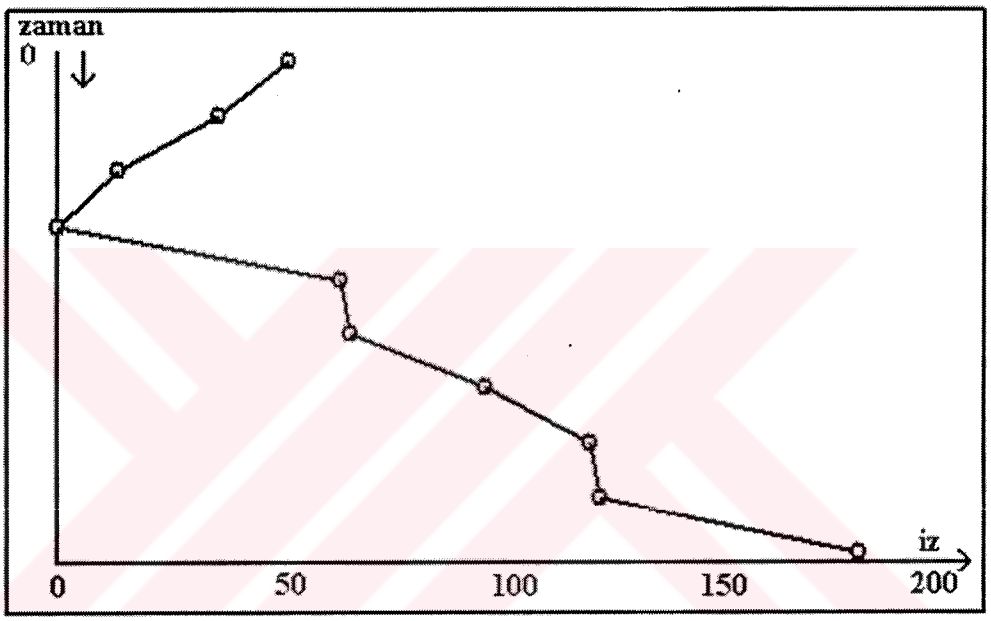
SCAN algoritmasında, disk kafası diskin bir ucundan başlayarak diğer uca doğru hareket eder ve güzergah boyunca isteklere servis yapar. Uç noktaya gelinince kafa hareketinin yönü değişir ve diğer uca doğru ilerlenerek yol boyunca isteklere servis yapılmaya devam edilir. Disk kafası sürekli olarak disk boyunca ileri ve geri yönde silindirleri tarar.

SCAN planlamasını uygulamak üzere tekrar aynı örneğimiz üzerinde inceleme yapacak olursak; isteklerimiz 98, 183, 37, 122, 14, 124, 65 ve 67 nolu silindir istekleri idi. Mevcut kafa pozisyonumuz ise 53 nolu silindirdeydi. Eğer disk kolu başa doğru ilerleme yönünü seçerse 37'nci ve 14'ncü silindirlere servis yapılır. Silindir 0'da disk kolu yön değiştirir ve diğer uca doğru ilerlemeye başlar.

Bu yönde 65, 67, 98, 122, 124 ve 183'ncü silindirlere servis yapılır (Şekil 3.12.). Herhangi bir anda kafanın önüne bir istek gelirse bu isteğe servis hemen yapılır,

arkasına bir istek gelirse, disk kafasının diğer uca gidip yönünü değiştirip gelmesi için bu istek bekletilir.

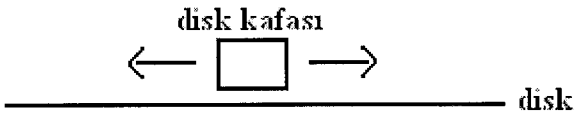
SCAN algoritması asansör (elevator) algoritması olarak da isimlendirilir. Çünkü çalışma mantığı asansörde yukarı çıkanları sırayla uğramak sonra aşağı hareket halindeyken aşağı inenlere sırayla uğramaktır. Bu ise SCAN algoritmasının işleyiş düzenidir.



Şekil 3.12. Scan disk planlaması

3.4.1.4. Tek Yönde Tarama (Circular SCAN-CSCAN) Planlaması

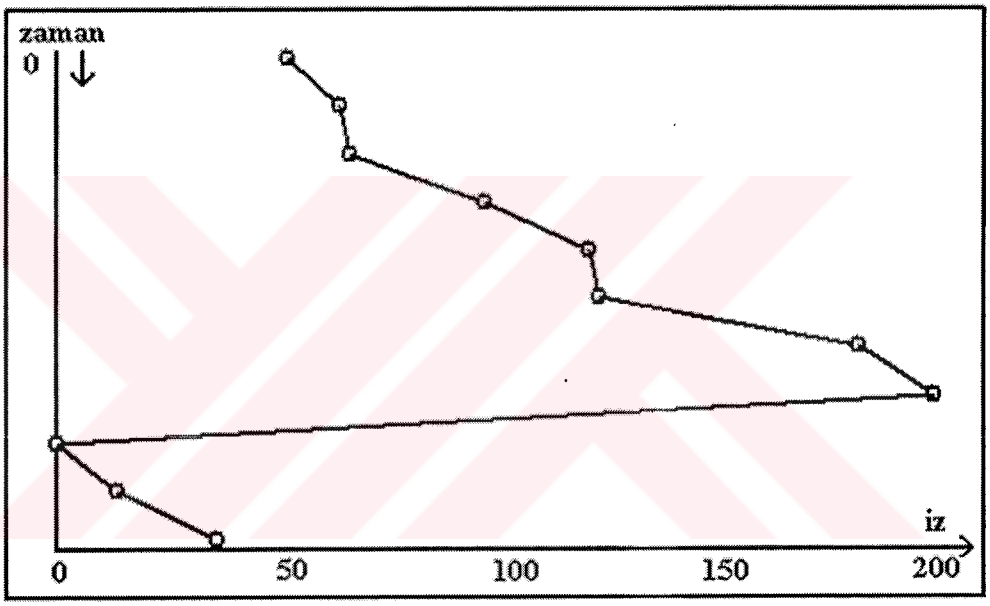
SCAN algoritmasında disk kafası disk üzerinde belli bir peryotla ileri-geri yönde hareket etmektedir.



Şekil 3.13. SCAN algoritması için disk kafa hareketi

Dikkat edilecek olursa böyle bir hareketle disk merkezindeki izler uç noktalardaki izlere nazaran iki kat daha sıklıkla disk kafasıyla karşılaşmaktadırlar. Merkez bölgelerdeki

istekler uç bölgelerdeki isteklerden daha kısa zamanda servis alacaklardır. Bu durum disk isteğinin iz numarasının büyüklüğü ve küçüklüğüne göre gereksiz manada öncelik farklılıklarının doğmasına neden olacaktır. Bu durumun önüne geçilmesi amacıyla SCAN algoritmasının bir türevi olan CSCAN algoritması geliştirilmiştir. CSCAN algoritması tek yönde isteklere cevap vermektedir. Disk sonuna gelindiğinde hiçbir isteğe cevap verilmeyerek başa dönülür ve tekrar baştan sona doğru istekler cevaplandırılır. Bu şekildeki bir çalışma ile disk isteklerindeki yanıt zamanlarındaki farklılıklar en aza düşürülmüş olunur.

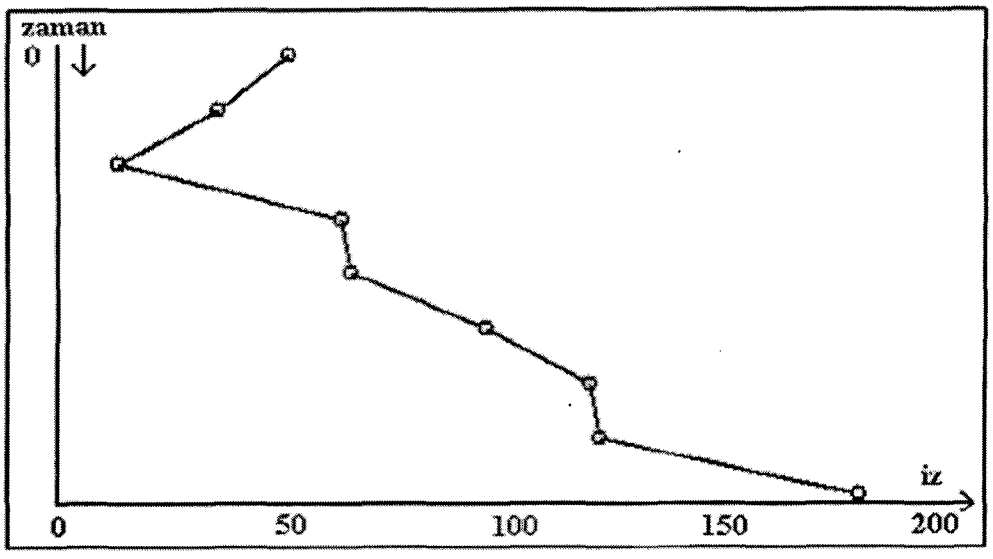


Şekil 3.14. CSCAN disk planlaması

3.4.1.5. Arama (LOOK) Planlaması

SCAN ve CSCAN algoritmalarında disk kafasının tüm disk yüzeyini taradığına dikkat edilmesi gerekir. Pratikte algoritma uygulamaları bu şekilde olmaz. Disk kafasını her iki yönde de diskin uç noktalarına hareket ettirmek yerine isteklerin belirttiği en son noktalardaki silindirlere hareket ettirmek mantıklıdır.

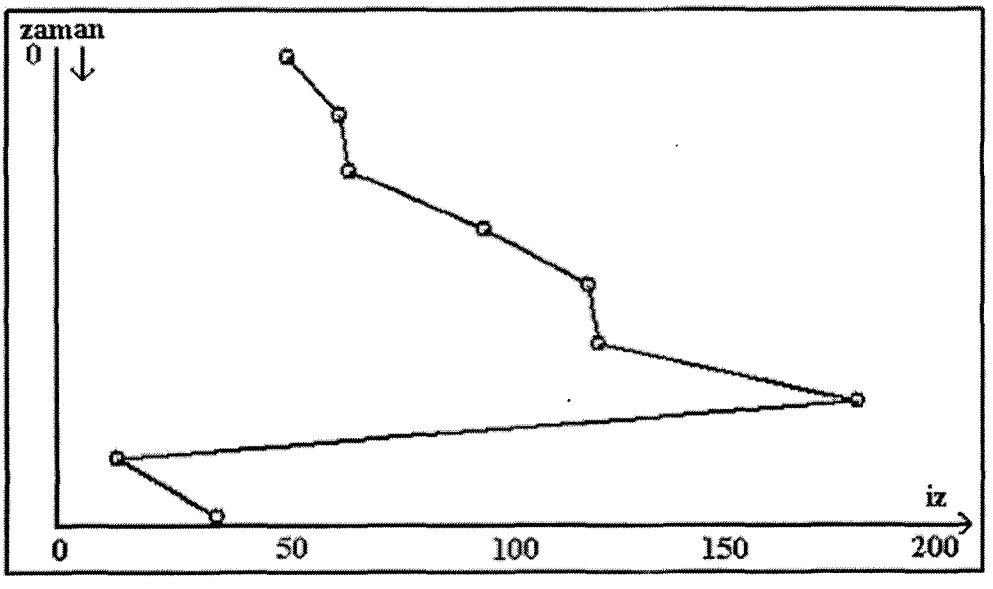
LOOK Algoritması SCAN algoritmasının işleyişiyle aynıdır. Çift yönde isteklere cevap verir, ancak verilen yönde servis yapacağı bir istek yoksa hareket yönünü değiştirir.



Şekil 3.15. Arama(LOOK) disk planlaması

3.4.1.6. Tek Yönlü Arama (CLOOK) Planlaması

Bu algoritmada da LOOK algoritmasında olduğu gibi disk kafası tüm disk yüzeyini taramamaktadır. Hareket yönünde servis yapılmayı bekleyen bir istek yoksa yön değiştirilmektedir. En baş ve son uç noktalara gereksiz hareket yoktur. Ancak bu algoritmada CSCAN algoritmasında olduğu gibi tek yönde servis işlemi vardır [4][8]. CLOOK algoritması yanıt zamanında iyileştirme için LOOK (arama) tekniğinin ve gereksiz öncelik farklılıklarının yok edilmesi içinde CSCAN (tek yönde tarama) tekniğinin birleşiminden faydalanılmıştır.



Şekil 3.16. CLOOK disk planlaması

Parametre olarak disk istek izi haricinde parametre kullanan bazı algoritmalar aşağıda verilmiştir. Bu algoritmaların bir kısmı birden fazla parametre kullanmaktadır.

3.4.1.7. En Erken Son İşlem Zamanlı İlk (Earliest Deadline First -EDF) Planlaması

EDF algoritması FCFS algoritmasının bir benzeridir. Burada da iz parametresinin bir etkisi yoktur. Bunun yerine isteklerin son işlem zamanları dikkate alınmaktadır. Disk istekleri kuyrukta son işlem zamanları değerlerine göre sıralanır ve bu sıraya göre servis yapılır. Son işlem zamanı en küçük olan istek en önce cevaplandırılır. Bu algoritma gerçek zamanlı sistemlerde kullanışlıdır. Burada bir disk isteğinin son işlem zamanı o isteğin bu zamandan önce yanıtlanması gerektiğini ifade etmektedir.

3.4.1.8. Taramalı En Erken Son İşlem Zamanlı İlk (SCAN-EDF) Planlaması

Bu tür bir planlama SCAN ve EDF planlamasının bir kombinasyonudur. En erken son işlem zamanlı isteğe her zaman önce servis yapılmaktadır. Ancak aynı son işlem zamanına sahip istekler varsa bunlar kendi aralarında gruplandırılır ve gruplar içerisinde SCAN algoritması uygulanır. Bu algoritmada son işlem zamanı (deadline) ve iz bilgisi olmak üzere iki parametre kullanılmaktadır.

3.4.1.9. Dinamik İstek Öncelikleri Kullanarak Disk Planlaması (Disk Scheduling with Dynamic Request Prioirties -DSDRP)

İstemci / Sunucu sistemlerde sunucuların performans sağlayabilmeleri için disk I/O kaynaklarını çoklu istemciler arasında verimli ve etkin kullanabilmeleri ve yönetebilmeleri gerekmektedir. Sayısal multimedya sistemlerde istemciler farklı önceliklere sahiptir. Yüksek öncelikli istekler en önce planlamaya tabi tutulur, düşük öncelikli istekler ancak yüksek öncelikli kuyruklar boş olduğu zaman işleme alınır. Bu algoritmada kullanılan bu yaklaşım öncelik optimizasyonunu yapmasına rağmen arama (seek) optimizasyonunu dikkate almamaktadır. Yani algoritmaya giriş parametresi olarak istek önceliği bilgisini almakta, iz numarasına bakmamaktadır. Algoritma iz bilgisini de dikkate alacak şekilde değişime tabi tutulabilir. Böyle bir değişim için istekler öncelik numaralarına göre gruplandırılarak bu gruplar içerisinde SCAN algoritması uygulanabilir [6].

3.4.1.10. Ağırlıklandırılmış En Kısa Toplam Zamanlı İlk (Weighted Shortest Total Time First - WSTTF) Planlaması

Disk erişimi planlamasında hareketli disk kafalarının icadından beri birçok teknik geliştirmiştir. Bu teknikler büyük hafızalı ve muhtemel uzun disk kuyruklu sistemlere uygulanmıştır. Disk kafa hareketini optimize etmeye çalışan ve yanıt zamanındaki açlık (starvation) problemini ortadan kaldırmayı amaçlayan klasik teknikler uzun disk kuyruklarında analiz edilmiştir. Burada erişim gecikmesi dikkate alınarak " ilk gelen ilk alınır –FCFS" algoritmasının dört katı iyileştirme yapan bir algoritma önerilmektedir.
Bu algoritmada belli bir zaman diliminde isteklere cevap vermeyi garantileyen "dosya sunucuları"'nda kullanılması uygun olan bir yapı oluşturulmuştur. Algoritma standart "En kısa arama zamanlı ilk-SSTF" planlaması tekniğine uygulanır ancak işlem zamanını belirleyen fonksiyon kullanılır. Her bir SSTF hesabında ağırlık değeri w ile gerçek I/O zamanı çarpılır. w, müsaade edilen maksimum yanıt zamanından önce ne kadar zaman kaldığının hesaplanmasıyla bulunur.

T_w : Ağırlıklandırılmış Zaman

T_{real} : Gerçek I/O zamanı

M :Müsaade edilen maksimum yanıt zamanı

E :İstek kuyruğa geldikten sonra harcanan zaman

$$T_w = T_{real} \frac{M - E}{M} \tag{3.6}$$

E zamanı disk isteğinin ne kadar süredir kuyrukta beklediğini göstermektedir ve bu süre arttıkça T_w ağırlık faktörü küçülür, bu sayede isteğe servis yapılma şansı artar. Çünkü kuyrukta ağırlık faktörü en düşükten başlanarak servis yapılmaktadır.
Bu algoritma ile elde edilen avantajlar şunlardır;

- Açlık "starvation" probleminden uzaklaşır.
- Maksimum yanıt zamanı dikkate alınır.
- Büyük disk kuyruklarında performans elde edilir [9].

BÖLÜM 4

KKO ALGORİTMASININ DİSK PLANLAMA PROBLEMİNE UYGULANMASI VE DİĞER TEKNİKLERLE SONUÇLARIN KARŞILAŞTIRILMASI

4.1. Giriş

Disk planlama probleminin anlatıldığı bölümden de gürüleceği üzere bu problemde amaçlanan çözüm, belirli parametreler ve kısıtlamalar gözönüne alınarak istenen kriterler doğrultusunda en uygun sıralamanın sağlanmasıdır. Tanımdan yola çıkılacak olunursa bu problemin bir optimizasyon problemi olduğu görülür.

Disk planlama probleminde disk kafasının belirli noktalara hareket etmesi ancak bu hareketin minumum mesafe sağlayacak şekilde yapılması hedeflenmektedir. Problem bu haliyle gezgin satıcı problemine (travelling salesman problem – TSP) benzemektedir. Disk üzerinde gidilecek iz pozisyonları gezgin satıcı problemindeki şehirlerle eşleştirilebilir, ancak disk planlama probleminde kafa hareketi mevcut pozisyondan itibaren başlar ve gezgin satıcı probleminde olduğu gibi tekrar başlangıç pozisyonuna gelmek durumunda da değildir. Gezgin satıcı problemi ve disk planlama problemlerinin her ikisi de karınca koloni sisteminin kolaylıkla uyarlanabileceği en kısa yol problemi olması nedeniyle, bu tür problemlerin çözümünde karınca koloni algoritması tercih sebebi olmuştur. Disk planlama problemindeki algoritma davranışlarının karınca koloni algoritmasındaki karınca davranışlarının benzerliği neticesinde problemdeki ve karınca algoritmasındaki matematiksel ifadelerin eşleştirilmesi kolaylıkla yapılabilmektedir.

4.2. Problem için Oluşturulan KKO Algoritması

4.2.1. Algoritma Tanımı

Gerçek karıncalar yuvalarından yiyecek kaynağına olan en kısa yolu bulurlar. Bunu yaparken gözlerinden faydalanmazlar. Eğer buldukları kısa yol çevresel etkenlerden dolayı değişirse belli bir zaman sonra yeni bir kısa yol bulurlar. Bir karınca feromon isimli kimyasal bir madde taşır ve bunu yürüdüğü yol üzerine bırakır. Kısa yolu tercih eden karınca uzun yolu tercih eden karıncalara nazaran daha çok madde bırakır. Çünkü birim mesafeye düşen karınca sayısı kısa yolda daha fazla olur. Karınca sayısı arttıkça madde miktarında da artma meydana gelir. Zamanla bu pozitif geri besleme etkisiyle kısa yolu tercih eden karınca sayısı çoğalır. Yukarıda açıklanan karıncaların davranışı gezgin satıcı probleminin çözümünün simulasyonunda kullanılabilir. KKO algoritması gerçek karıncaların bu davranışını taklit etmektedir. KKO algoritmasında yapay karıncaların farklı nesilleri, problem için iyi çözümler aramaktadır.

Disk planlama; iz numarası, öncelik, son işlem zamanı gibi istek parametrelerini dikkate alarak disk istekleri için en iyi sıralama yapmayı amaçlar. Disk planlama problemi gezgin satıcı problemine benzemektedir. Benzetmede, karıncaların en kısa yiyecek yolu, istek kuyruğundaki isteklerin optimum sıralamasını temsil etmektedir. Bu çalışmada, KKO algoritması, gezgin satıcı probleminin biraz değiştirilmiş hali olan optimum disk planlaması için kullanılmıştır.

Her bir karınca, daha önceden ziyaret edilen disk pozisyonlarını tutmak için bir hafızaya (M_k) sahiptir. Hafıza her yeni turun başlangıcında silinir ve her yeni disk pozisyonunda güncellenir. Disk pozisyonundaki k yapay karıncası hareket etmek için M_k hafızasında olmayan bir sonraki disk pozisyonunu aşağıdaki olasılık formülü kullanarak seçer.

$$P_k(r,s) = \begin{cases} \dfrac{\tau(r,s)^{\alpha}\,\eta(r,s)^{\beta}}{\sum_{s \notin M_k} \tau(r,s)^{\alpha}\,\eta(r,s)^{\beta}} \\ 0 \end{cases} \qquad (4.1)$$

Eşitlikte $\tau(r,s)$ ifadesi (r,s) pozisyonundaki feromon maddesi miktarını, η_{rs} ifadesi sezgisel bir yaklaşım olan izler arasındaki mesafenin tersini, α ve β değerleri ise feromon miktarının sezgiselliğe olan etkisini kontrol etmeye yaramaktadır. Eşitlik, k

karıncasının mevcut r pozisyonundan s pozisyonunu seçmesi ihtimalini $P_k(r,s)$ olarak vermektedir.

Başlangıçta karıncalar mevcut disk kafası pozisyonundan başlarlar. k karıncası bütün disk pozisyonlarının seçimi işlemini yaptıktan sonra uzaklık Yol-Uzunluk değişkeninde ziyaret edilen izler arasındaki mesafelerin toplamıyla elde edilir. Feromon maddesi miktarı olan değer aşağıdaki eşitlikle saptanır.

$$\Delta\tau_k(r,s) = (Yol - Uzunluk)^{-1} \tag{4.2}$$

Kolonideki bütün karıncalar kendi turlarını tamamladıktan sonra (r,s) pozisyonu için feromon miktarı aşağıdaki eşitlikle güncellenir.

$$\tau(r,s) = \sum_{k=1}^{n} \Delta\tau_k(r,s) + \alpha\tau_0 \tag{4.3}$$

Eşitlikte n değeri (r,s) pozisyonunu seçen karıncaların sayısını, T_0 değeri, (r,s) pozisyonundaki bir önceki feromon maddesi miktarını ve α ise feromon maddesinin buharlaşma katsayısını vermektedir.

4.2.2. KKO Algoritma Yapısı

Disk problemi için oluşturulan karınca algoritması, gezgin satıcı probleminin uygulanmasına benzer bir şekilde oluşturulmuştur. Gezgin satıcı probleminde, karıncalar en iyi yolu bulma çabasında iken katettikleri mesafeye göre davranış değişimi yaparlar. Eğer bir grup karınca yiyeceğe ulaşmak için genele nazaran daha fazla yol gitmişse bir sonraki nesilde bu gruptaki karıncaların sayısında azalma görülecektir. Gezgin satıcı problemi için kullanılan tek parametre şehirlerin pozisyonlarıdır. Karınca algoritmasında, karıncaların aldıkları mesafeler bu pozisyonlar arasındaki farkların toplamının hesaplanmasıyla bulunur. Disk probleminde geçerli plan istek izlerinin pozisyonları, gezgin satıcı problemindeki şehir pozisyonlarına eşleştirilirse, gezgin satıcı problemi için geliştirilen karınca algoritması en temel anlamda disk problemine uyarlanmış olacaktır. Bu manada bir karınca algoritmasının özet yapısı aşağıdaki gibi olacaktır.

- Adım 1 : Başlangıç
  - Her bir (i,j), i noktasından j'ye olan yol için, başlangıç bir feromon maddesi değeri atanır,
  - Bölgedeki her bir noktaya n adet karınca yerleştirilir (eğer başlangıç noktası belirli ise bütün karıncalar bu noktaya yerleştirilir),
  - Her bir karınca için ilk nokta tabu listesine alınır,
- Adım 2 : Her bir i noktası için tabu listesi dolana kadar bu adım tekrarlanır :
  - i noktasında henüz hareket etmemiş her bir k karıncası için:
    - Olasılık fonksiyonuna bağlı olarak bir sonraki j noktası seçilir,
    - k karıncası bir sonraki j noktasına hareket ettirilir,
    - k karıncasının tabu listesine j noktası eklenir,
    - i noktasından j noktasına feromon maddesinin yoğunluğunun değişimi hesaplanır,
  - Bütün (i, j) geçişleri için yeni feromon maddesi yoğunluk miktarı hesaplanır,
- Adım 3 : En kısa yol hafızaya alınır,
  - Eğer iterasyon sayısı, maksimum iterasyon sayısından küçük ise;
    - Bütün tabu listeleri boşaltılır,
    - Her bir karınca mevcut pozisyonunu tabu listesine ekler,
    - 2. adıma gidilir,
  - değilse;
    - En kısa yol sonuç olarak verilir.

Disk planlama probleminde, harddisk kafasının daha önceden bulunduğu pozisyon belirli olduğundan disk isteklerine gidilirken hali hazırdaki bu pozisyondan hareket edilecektir. Bundan dolayı bu problem için KKO algoritmasında, bütün karıncalar bu pozisyonu temsil eden noktadan itibaren hareket ederler. Bir karınca, bir noktadan diğer bir noktaya (i,j) harekete geçtiğinde belirli miktarda feromon maddesi bırakır. Her bir karınca bir sonraki hareket noktasını seçerken belli bir olasılık fonksiyonunu kullanır. Bu fonksiyon, hareket edilecek noktanın uzaklığına ve o nokta için mevcut feromon miktarına göre değer üretmektedir. Her bir karınca, gittiği noktalara tekrar gitmemek için gittiği noktaları yazdığı bir liste tutar. Pozisyon sayısını gösteren n adımdan sonra her bir karınca bütün noktalardan bir şekilde geçen bir güzergaha sahip olur. Bu karıncalar içerisinde en kısa olan güzergah kaydedilir ve yeni bir iterasyon daha başlatılır.

Yukarıda özetlenen karınca algoritması, gezgin satıcı ve benzeri problemlerin çözümü için yeterlidir. Bu algoritma bazı düzenlemelerle dinamik problemlerin çözümünde de yeterli olabilmektedir [10][11].

4.2.2.1. Algoritmanın İlk Adımı

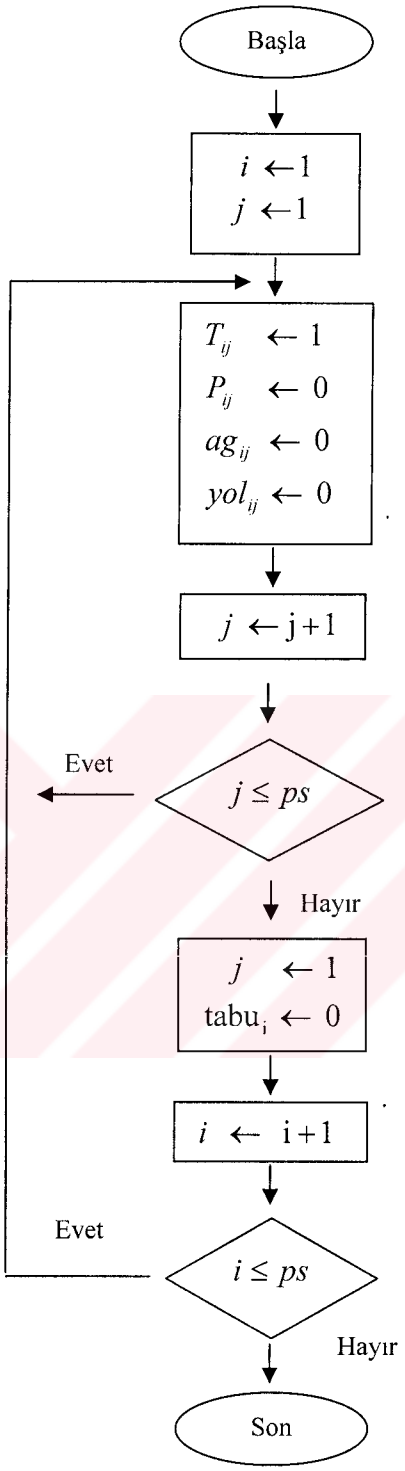
Algoritmada kullanılan feromon maddesinin karşılığı olan değişken T_{ij}, her bir noktadan noktaya geçiş yolu için başlangıç değeri olan 1'e eşitlenir. Daha sonraki adımlarda bu değer, bu yolun uygunluk(fitness) değerine ve buharlaşma katsayısına göre değişim içerisinde olur. Her bir karıncanın başlama noktası disk kafasının mevcut pozisyonu olan noktadan başlaması gerekir. Bu durum burada göz önüne alınarak hiçbir karınca için noktalara dağıtılma işlemi yapılmaz. Bir sonraki adımda bütün karıncalar sabit bir noktadan başlatılır ve bu nokta tabu listesine eklenir. Dolayısıyla burada tabu listesi içinde bir başlangıç işlemi gerçekleştirilmesine gerek yoktur. Başlangıç adımında tabu listelerinin boşaltılması, ileride kullanılacak olan olasılık fonksiyonunun başlangıç değerinin, pozisyonlar arası mesafelerin ve katedilen yolun mesafesini gösteren değişkenlerin sıfırlanması yapılır. Bu durumda algoritmada ilk adımımız için akış şemamız Şekil 4.1.' deki gibi olacaktır.

Akış şemasında geçen ifadeler; i ve j pozisyon indislerini, T_{ij} feromon maddesi değişkenini, P_{ij} olasılık fonksiyonunun değerini, ag_{ij} pozisyonlar arası mesafeyi, yol_{ij} karıncanın katettiği yol uzunluğunu, ps pozisyon sayısını, tabu ise karıncanın tabu listesini göstermektedir.

İlk adımın akış şemasında pozisyonların birbirlerine olan uzunlukları çift boyutlu bir matrisle i ve j indisleri kullanılarak yapılır. Bu yoldaki feromon maddesi ise bu indislerin gösterdiği T_{ij} ile ifade edilir. Başlangıç değerleri, çift boyutlu bu matrisin elemanları kadar adım sayısı ile verilir.

Pozisyonların birbirlerine olan uzaklıkları çift boyutlu bir matrisle i ve j indisleri kullanılarak verilir. Bu yollardaki feromon maddesi ise bu indislerin gösterdiği T_{ij} ile ifade edilir. Başlangıç değerleri, matrisin elemanları adedince adım sayısı ile verilir.

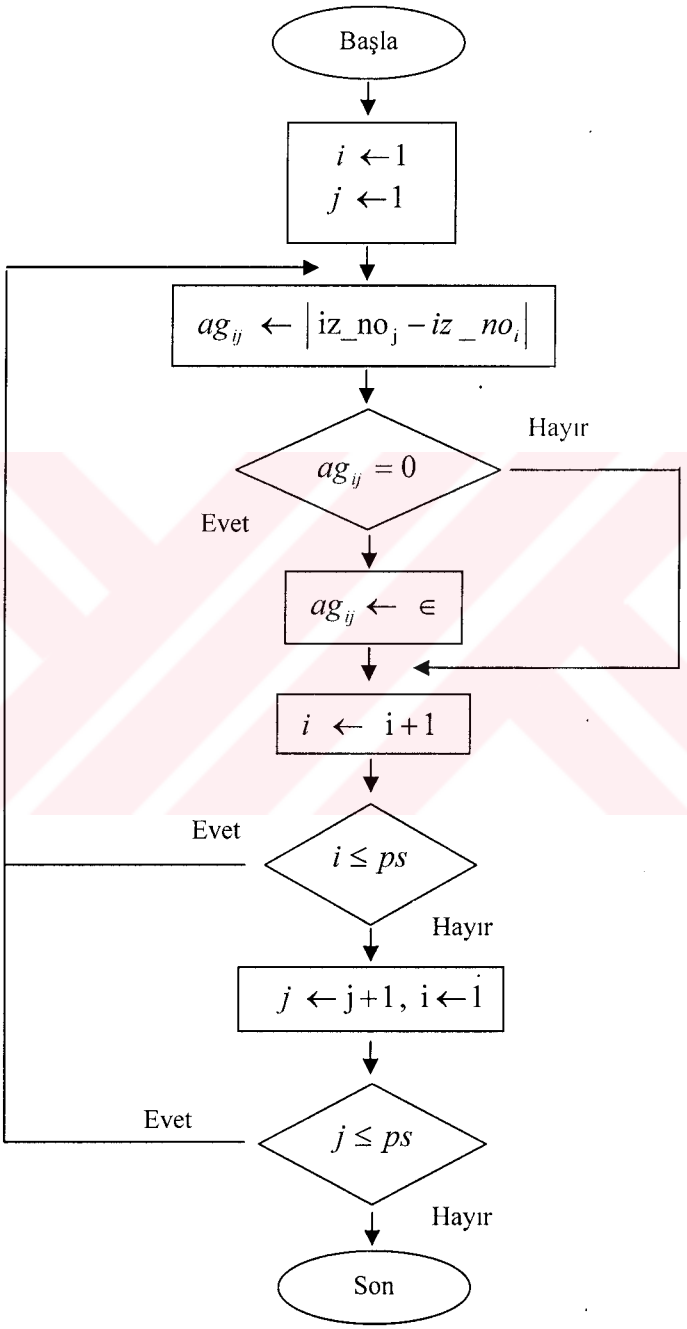
Pozisyonlar arası mesafelendirme bu adımda yapılan bir işlemdir. Disk probleminin karınca algoritmasına uyarlamasında bu adım önem kazanmaktadır. Algoritmada disk isteği iz numaraları, öncelik ve son işlem zamanları değerlerine bağlı parametreler dikkate alınmaktadır.



Şekil 4.1. Başlangıç değerlerinin atanmasının akış şeması

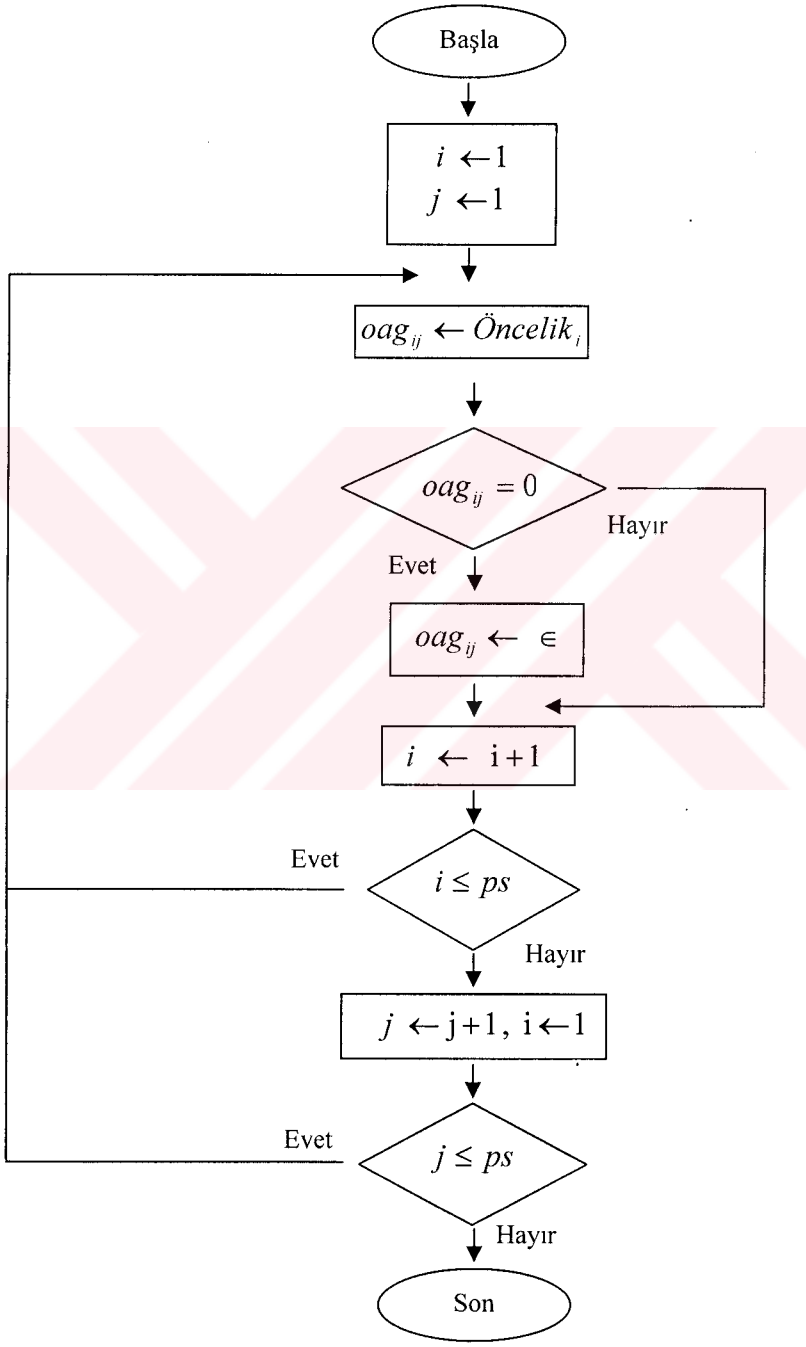
Disk isteklerinin iz numaraları, verinin disk plakası üzerindeki pozisyonları gösterdiği için bu parametreye bağlı mesafelendirme işlemi gezgin satıcı probleminde olduğu gibi Şekil 4.2.'deki akış şemasıyla gerçekleştirilir. Buna göre disk isteğinin iz numaraları arasındaki fark ag_{ij} mesafe değişkenine direk olarak atanır. Değer atama işlemi,

pozisyonların birbirlerine göre mesafe farklarını ifade etmek için çift boyutlu matris halinde yapılır. Eğer bu fark 0 değerinde ise ileriki adımlarda "0'a bölme hatası" ile karşılaşmamak için bu değer 0'a çok yakın $\in$ değeri olarak değiştirilir.



Şekil 4.2. Disk isteklerinin iz numaralarına göre mesafelendirme işleminin akış şeması

Karınca algoritmasında öncelik ve son işlem zamanlarına göre de mesafelendirme işlemi yapılmaktadır. Bu sayede disk planlama probleminde iz numaraları ile birlikte üç parametre algoritmaya girdi olarak verilebilmektedir.



Şekil 4.3. Disk isteklerinin önceliklerine göre mesafelendirme işleminin akış şeması

Disk isteklerine önceliklerine göre servis yapılması için bu isteklerin öncelik değerlerine göre sıralanması gerekmektedir. Literatürde öncelik (priority) değerleri küçük olanlar, öncelik değerleri büyük olan isteklere göre daha öncelikli farz edilmektedir [12]. Bu çalışmada da istekler öncelik değerlerine bakılarak, küçük olanlara olumlu yönde (daha önce servis edilme anlamında) yaklaşım geliştirilecektir.

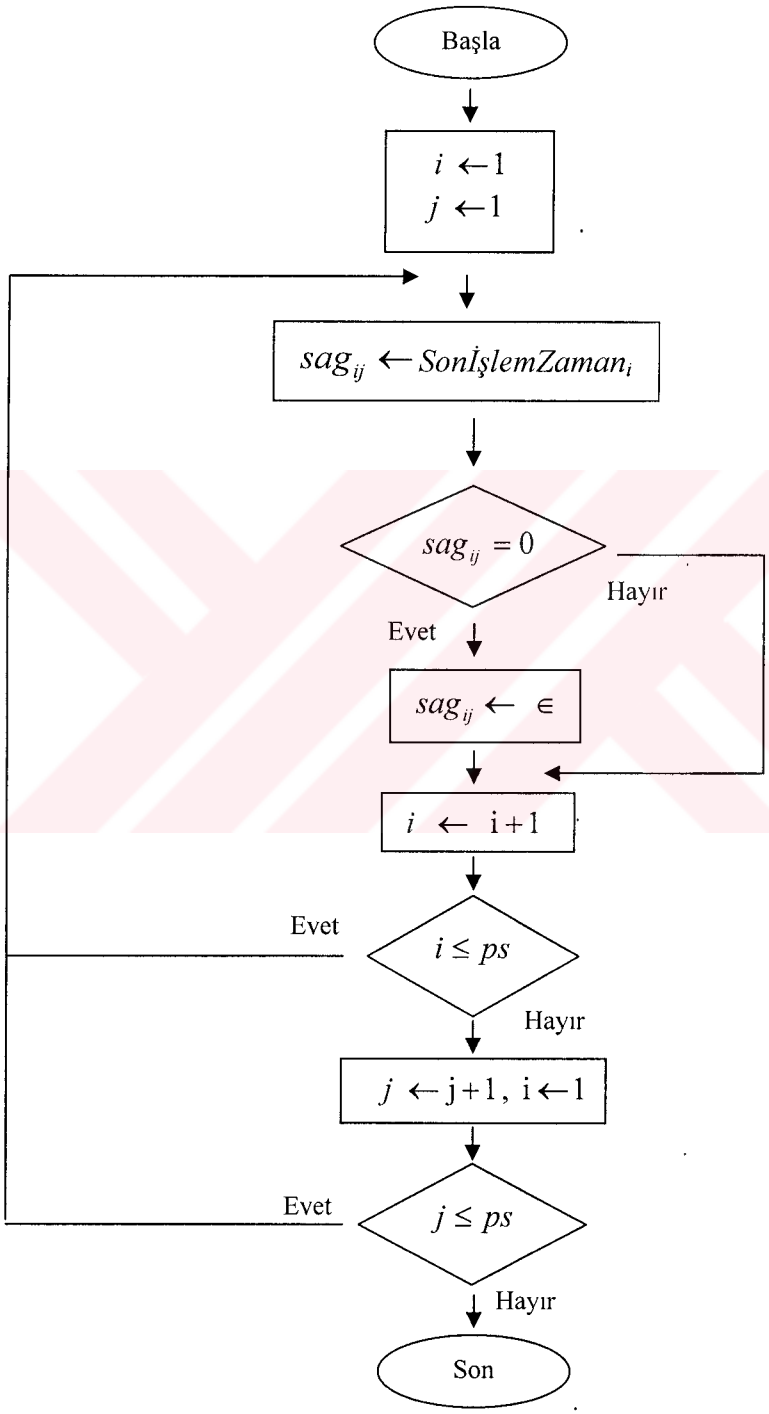
Önceliklere bakılarak yapılacak mesafelendirme işlemi gidilmesi düşünülen bir sonraki adımın öncelik değerine bakılarak yapılır. Karıncalar harekete belirli bir pozisyondan başlamak zorundadır. Çünkü disk kafasının başlangıçta bulunduğu pozisyon hesaplamalarda dikkate alınmalıdır. Bu pozisyon bir disk isteği pozisyonu olmamasına rağmen başlangıç için hesaplamalara katılan bir noktadır. Bu başlangıç noktasının iz numarası belirli olmasına rağmen, diğer disk istekleri gibi öncelik ve son işlem zamanları parametrelerine sahip değildir. Ancak disk isteklerini tutan dizi değişkeni tanımlamasında başlangıç pozisyonu bu dizinin ilk elemanı olmaktadır. Öncelik ve son işlem zamanı olarak 0 değeri verilmektedir. Disk isteklerinin önceliklere göre yapılacak sıralamasında bu pozisyon en ilk sırada olacaktır. Disk isteklerinin önceliklerine göre yapılan sıralamanın bulunmasıyla, birbirine yakın öncelikli değerler, algoritmanın ileriki adımlarında bir araya getirilmeye çalışılacaktır. İlk pozisyon öncelik değeri 0 olduğu için bunun devamında 1, 2, 3 ve birbirine yakın değerli öncelikliler bir araya gelecektir. Sonuçta istekler öncelik değerlerine göre sıralanmaya çalışılacaktır. Bu sayede disk isteğinin öncelik parametresiyle bizden istenen önceliklere göre servis yapılma işlemi gerçekleştirilmiş olacaktır.

Öncelik mesafelendirilmesinde, disk isteklerinin öncelik değerlerinin birbirlerine göre kıyaslaması yapılır eğer disk isteğinin kendi öncelik değerinden daha küçük değerdeki bir öncelik değerli disk isteği varsa mesafe '0' olmalıdır. Çünkü kendinden daha öncelikli bir isteğe sıra verilmesi için bu yaklaşımın gerçekleştirilmesi gerekir. Bu yaklaşım;

$$Eğer\ öncelik_i > öncelik_{i+1}\ ise\ mesafe=0$$

ifadesiyle gerçekleştirilir.

Şekil 4.3.' de önceliklere göre yapılan mesafe işleminin akış şeması yer almaktadır. Burada '0' değerli olan değişkenler için daha sonra '0' 'a bölme hatası ile karşılaşılmaması için bu değişkenlere sıfıra çok yakın ε değeri ataması yapılır.



Şekil 4.4. Disk isteklerinin son işlem zamanlarına göre mesafelendirme işleminin akış şeması

Akış şemasında yer alan *i* ve *j* matris indislerini, oag_{ij} *i* ve *j* indisli istekler arasındaki öncelik mesafe bilgisini ve ps ise pozisyon sayısını göstermektedir.

Son işlem zamanı için yapılan mesafelendirme, öncelikler için yapılan mesafelendirmenin çok benzer bir yapısıdır. Bilindiği üzere isteklere servis yapılırken son işlem zamanlarına bakılır ve en erken son işlem zamanlı istek önceliğe alınır. Başka bir ifadeyle; öncelik parametrelerine bakılıp küçük değerli olanların tercih sebebi olması işlemine benzer şekilde isteklerinin son işlem zamanlarına bakılarak küçük değerlilerin öne alınmasına çalışılır.

Şekil 4.4.'de önceliklere göre yapılan mesafelendirme akış şemasının son işlem zamanlarına göre uyarlanmış hali yer almaktadır. Burada i ve j matris indislerini; *son işlem* $_i$ *i.* disk isteğinin son işlem zamanı; sag_{ij} *i* ve *j*. İstekler arasındaki son işlem zamanlarının mesafe bilgisini ve *ps*'de pozisyon sayısını ifade etmektedir.

4.2.2.2. Algoritmanın İkinci ve Üçüncü Adımı

Her bir tabu listesi dolana kadar veya her bir pozisyon ziyaret edilene kadar bu adım tekrarlanır. Şekil 4.5.'deki akış şemasında tekrarlama işlemini n=1 'den n = pozisyon sayısına kadar yapılmaktadır. En son adımda tabu listesi dolmuş olacaktır. Akış şemasında gösterilen adımlar her bir karınca için yapılır ve karıncaların buldukları en iyi sıralama, algoritmanın üçüncü adımında hafızaya alınır. Buraya kadar yapılan işlem daha önceden belirlenen maksimum iterasyon sayısı adedince tekrarlanır ve bu sayede bu sayının büyüklüğüne bağlı olarak iyi sonuçların bulunma olasılığı değişir.

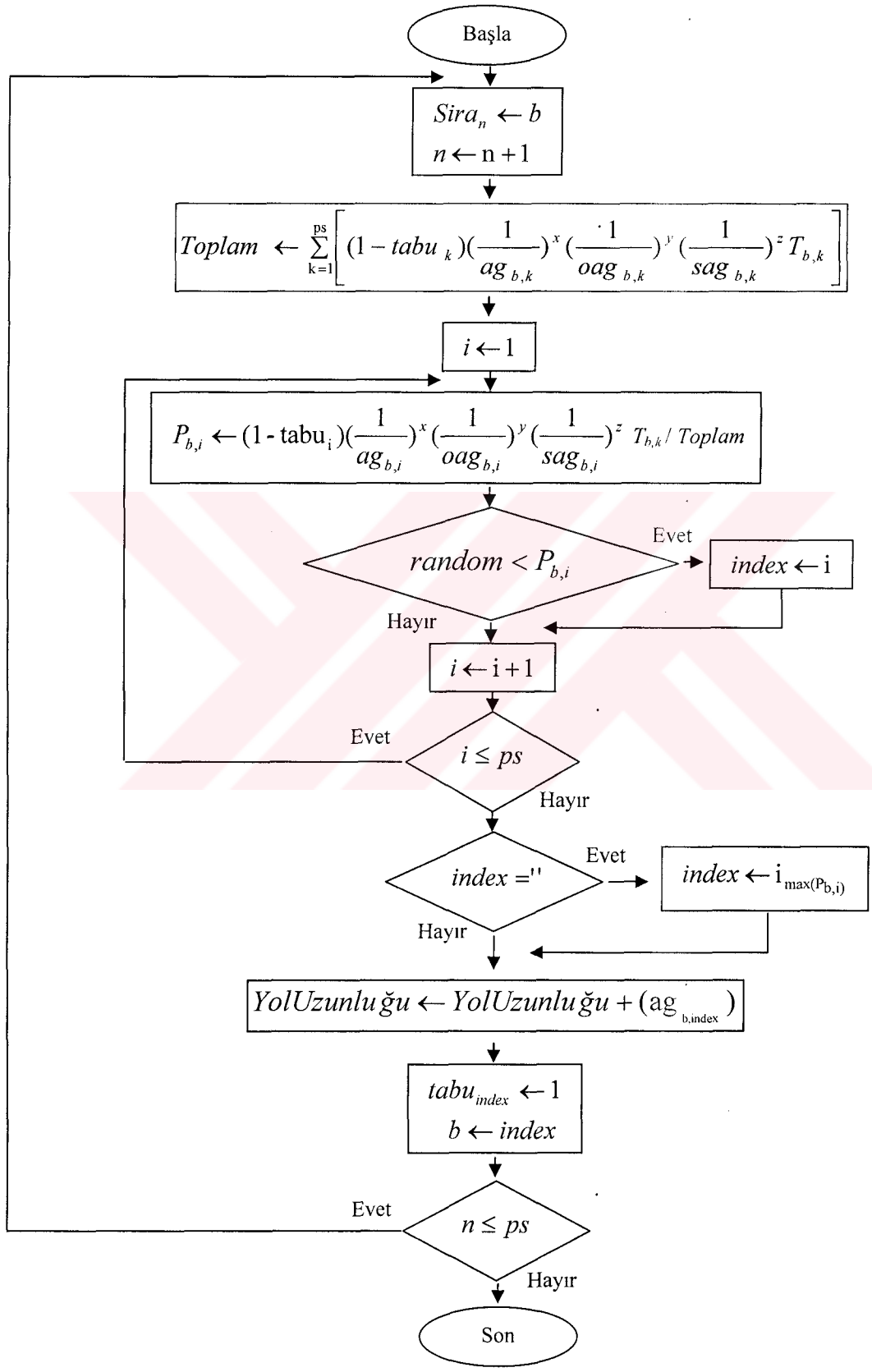
Algoritmanın ikinci adımının işleyişi, her bir karıncanın çift boyutlu bir matris dizisini ifade eden *i, j* yolarındaki feromon maddesine bağlı olarak değer üreten bir olasılık fonksiyonunu kullanarak ilerleyeceği yolları bulma çabasıyla gerçekleşir. Sonuçta bir karıncanın hareket ettiği güzergah bir noktalar dizisinden ibarettir ve bu dizi algoritmamızda *Sıra* $_n$ değişkeninde tutulmaktadır, *b* değişkeninde karıncanın mevcut pozisyonu tutulur. Karınca bu mevcut pozisyondan hareket edeceği pozisyonu (*index*) seçmeye çalışmaktadır. Ancak seçeceği pozisyon daha önceden gitmediği bir pozisyon olmalıdır yani tabu listesinde olmamalıdır. Seçim için öncelikle *b*'den gidebileceği pozisyonlara olan yollardaki ($ag_{b,k}$) sezgisel ifade ile feromon miktarının çarpımlarının toplamı hesaplar. Bu hesaplama aşağıdaki eşitlikle yapılır;

$$Toplam \leftarrow \sum_{k=1}^{ps} \left[(1 - tabu_k)(\frac{1}{ag_{b,k}})T_{b,k} \right] \qquad (4.4)$$

Eşitlikteki $tabu_k$ değişkeni k. pozisyonun tabu listesinde olup olmadığını göstermektedir. Eğer bu eleman tabu listesinde değilse $tabu_k$ 0 değerinde, tabu listesinde ise 1 değerindedir. Tabu listesinde olan k pozisyonu için eşitlik sıfır değerini üretmekte yani feromon miktarı hesabını yapmamaktadır, $ag_{b,k}$ ifadesi yol uzunluğunu vermekte ve eşitlik 4.1'de verilen sezgisel ifade η fonksiyonu olarak bu uzunluğun tersini almaktadır. Yani bizim burada η sezgisel fonksiyonumuz $(\frac{1}{ag_{b,k}})$ şeklindedir. η sezgisel fonksiyonunun ağırlığı olan β ifadesi bu algoritmada x değişkenine karşılık gelmektedir. $T_{b,k}$ değişkeni ise bu yoldaki mevcut feromon miktarını vermektedir.

$$Toplam \leftarrow \sum_{k=1}^{ps} \left[(1 - tabu_k)(\frac{1}{ag_{b,k}})^x (\frac{1}{oag_{b,k}})^y (\frac{1}{sag_{b,k}})^z T_{b,k} \right] \qquad (4.5)$$

Eşitlik 4.5 'de öncelik değerlerini dikkate alan sezgisel fonksiyon $(\frac{1}{ag_{b,k}})^y$ ve son işlem zamanlarını dikkate alan sezgisel fonksiyonda $(\frac{1}{sag_{b,k}})^z$ ifadeleridir. Bunların ağırlıkları olan x, y ve z değerlerinde değişim yapılarak, algoritma işleyişi istenen parametreye daha çok önem göstererek çalışabilmektedir. Toplam ifadesi bulunduktan sonra b pozisyonundan hareket edebileceği (b,i) yollarındaki feromon maddelerine $(T_{b,i})$ göre ve sezgisel ifadelere $((\frac{1}{ag_{b,i}})^x (\frac{1}{oag_{b,i}})^y (\frac{1}{sag_{b,i}})^z)$ göre değer alan bir fonksiyon sonucuyla [0,1] aralığında üretilen rastgele bir değer karşılaştırılır. Eğer sonuç olumlu ise $(random < P_{b,i})$ i noktası *index* olarak belirlenir yani bir sonraki hareket noktası olarak seçilir. Eğer indeks olarak seçilen hiçbir nokta olmazsa maksimum olasılık değerli i noktası indeks olarak alınır. Daha sonra yol uzunluğu ileride diğer karıncaların bulduğu çözümlerle karşılaştırılmak üzere hesaplanır. Artık, b mevcut pozisyon değişkenine indeks pozisyon değeri yüklenerek karıncanın bu pozisyona hareketi sağlanır, ve bu pozisyon tabu listesine eklenir.



Şekil 4.5. Karınca algoritmasının 2. adımının akış şeması

Algoritmada pozisyon sayısı *ps*, adedince pozisyon seçme ve seçilen pozisyona hareket etme işlemi gerçekleştirilir.

Karıncaların bulduğu çözümler karşılaştırılmak için algoritmanın üçüncü adımında uygunluk (fitness) değerlerinin hesabı yapılarak karşılaştırılır. *k* karıncasının takip ettiği yolların uygunluk değeri yol uzunluğu değişkenlerinin toplamının tersiyle elde edilir. Her bir karınca için alınan mesafelerin toplamının 1'e bölümünden elde edilen değerlerin en büyüğü bize en iyi yolu bulmamızı sağlayacaktır. Daha sonra bu yol hafızada tutulmaktadır. Her bir iterasyonda (nesilde) yollardaki $T_{i,j}$, *sigma* değişkeninin içeriğine bağlı olarak kısmen buharlaştırılır.

Karınca algoritmasının bu bölümde anlatılan işleyiş düzeninin disk problemine uyarlanmış yapısı delphi kodları halinde test programı olarak oluşturulmuştur.

4.3. Tek Parametre kullanılarak KKO Algoritmasının Disk Planlama Problemine Uygulanması

Disk planlama probleminde ilk göze çarpan parametre disk kafasının hareket etmesi gereken iz pozisyonları ve bunun neticesinde performans değerlendirmesi yapacağımız kriterde kafanın hareket ettiği yerlerin toplam mesafesidir. Bu mesafenin kısalığı bize performansın iyilik derecesini verecektir.

r_i, i nolu bir servis isteği olarak disk kafasının r_i'nin belirttiği iz pozisyonuna hareket etmesini istemektedir. $\alpha =< r_1, r_2, ..., r_n >$ ise kuyrukta servis yapılması yani disk kafasının r_i'nin gösterdiği pozisyonlara gitmesi için bekleyen bir dizidir. Kuyruk içerisindeki bu dizi her r_i için öyle sıralanmalıdır ki istenen kriter doğrultusunda en iyi performans alınabilmelidir. Bu işlem yapılırken en az mesafeli kafa hareketini sağlayan dizi sıralaması işlemi $< r_i >$ isteğinin belirttiği r pozisyonu dikkate alınmaktadır. Buradan anlaşılacağı üzere kriterimiz disk kafasının katettiği toplam mesafe bilgisi, parametremiz ise r pozisyonunun bulunduğu iz numarasıdır. Mesafe bilgisi direk olarak bu mesafenin katedilmesi için gerekli süre ile ilişkilidir. Buradaki herbir süre bir sonraki istek için yanıt zamanı olmaktadır.

$$\text{Toplam Yanıt Zamanı} = \sum f(n) \tag{4.6}$$

Formülünde $f(n)$ kafanın mevcut bulunduğu izle bir sonraki adımda gitmesi gereken iz arasındaki iz sayısı n'nin ne kadar sürede katedileceğii belirleyen formüldür. Eşitlik 4.6 genel bit eşitlik olup, disk modeline göre değişmektedir.

Algoritmada disk istekleri bir matris dizisi halinde tutulmaktadır. r_i disk isteği iz numarası, öncelik ve son işlem zamanı bilgilerini içeren i. isteği tanımlayan üç boyutlu bir matristir. Karınca algoritmasında bu isteklerin özelliklerinden şekil 4.2.'deki gibi iz numarası bilgisi kullanılırsa, algoritma yanıt zamanını en aza indirmeye çalışan tek parametreli bir yapı olmaktadır. Aynı şekilde sadece öncelik (Şekil 4.3.) veya son işlem zamanı (Şekil 4.4.) bilgileri kullanılarak da tek parametreli bir yapı oluşturulabilir. Böyle bir durumda algoritma disk isteklerini önceliklerine veya son işlem zamanlarına göre küçükten büyüğe sıralayacaktır.

Disk isteği iz numaralarına göre algoritma sonuçları konunun devamında verilmiş olup diğer tekniklerle mukayese amacıyla klasik algoritma verileri de bir örnek üzerinde Şekil 4.7 - Şekil 4.26'da gösterilmiştir.

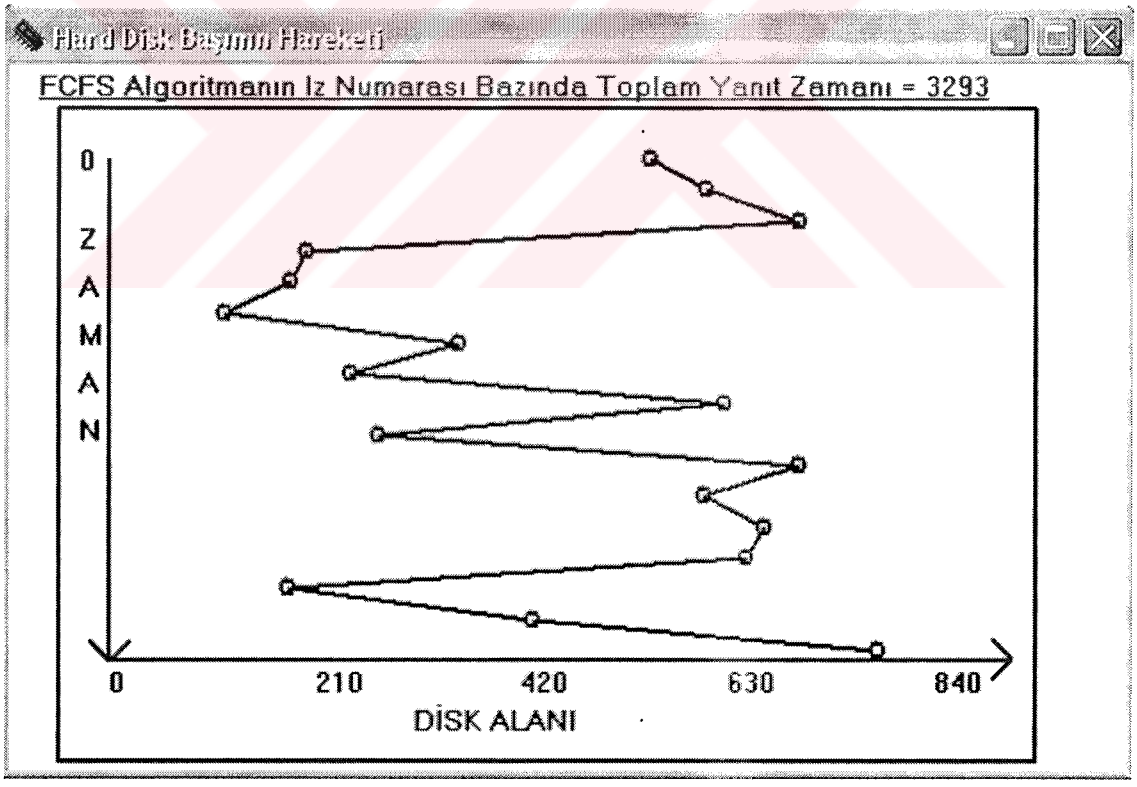
Şekil 4.6.'da gösterildiği düzende disk kuyruğunun büyüklüğü $q = 16$, son işlem zamanları [1000-2000] aralığında rastgele değerler, öncelikler ise [0-10] aralığında rastgele değerler alınmıştır. Model olarak aldığımız fujitsu M2361A disk sürücüsü için disk üzerindeki maksimum iz numarası da 840 olarak alınmıştır [13]. Şekildeki geliş sırası için hard disk kafa hareketi Şekil 4.7 - Şekil 4.26'da her bir teknik için gösterilen hareket tarzında olacaktır. Burada yapılan toplam yanıt zamanı hesabı model olarak aldığımız disk özelliklerinden;

$$YanitZaman_x = \begin{cases} 0 & \text{if } x = 0 \\ 4.6 \times 10^{-3}\ \text{sec} + \sqrt{x} \times 0.87 \times 10^{-3}\ \text{sec} & \text{if } x \leq 239 \\ 18 \times 10^{-3}\ \text{sec} + (x - 239) \times 0.028 \times 10^{-3}\ \text{sec} & \text{if } x > 239 \end{cases} \tag{4.7}$$

eşitliğini kullanılmaktadır. Burada x ifadesi hareket edilecek iki pozisyon arasındaki iz sayısını vermektedir.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı(msn)
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	(1)	574	1553	9	11,052
2 - Kuyruk[2]:	(2)	662	1898	0	23,813
3 - Kuyruk[3]:	(3)	190	1968	6	48,337
4 - Kuyruk[4]:	(4)	174	1985	2	56,417
5 - Kuyruk[5]:	(5)	110	1248	5	67,977
6 - Kuyruk[6]:	(6)	336	1565	8	85,656
7 - Kuyruk[7]:	(7)	232	1271	8	99,129
8 - Kuyruk[8]:	(8)	591	1385	7	120,489
9 - Kuyruk[9]:	(9)	259	1019	8	141,093
10 - Kuyruk[10]:	(10)	663	1249	1	163,713
11 - Kuyruk[11]:	(11)	572	1918	3	176,612
12 - Kuyruk[12]:	(12)	629	1218	6	187,78
13 - Kuyruk[13]:	(13)	612	1055	0	195,967
14 - Kuyruk[14]:	(14)	171	1871	1	219,623
15 - Kuyruk[15]:	(15)	407	1478	8	237,589
16 - Kuyruk[16]:	(16)	738	1873	8	258,165

Şekil 4.6. İstek listesi

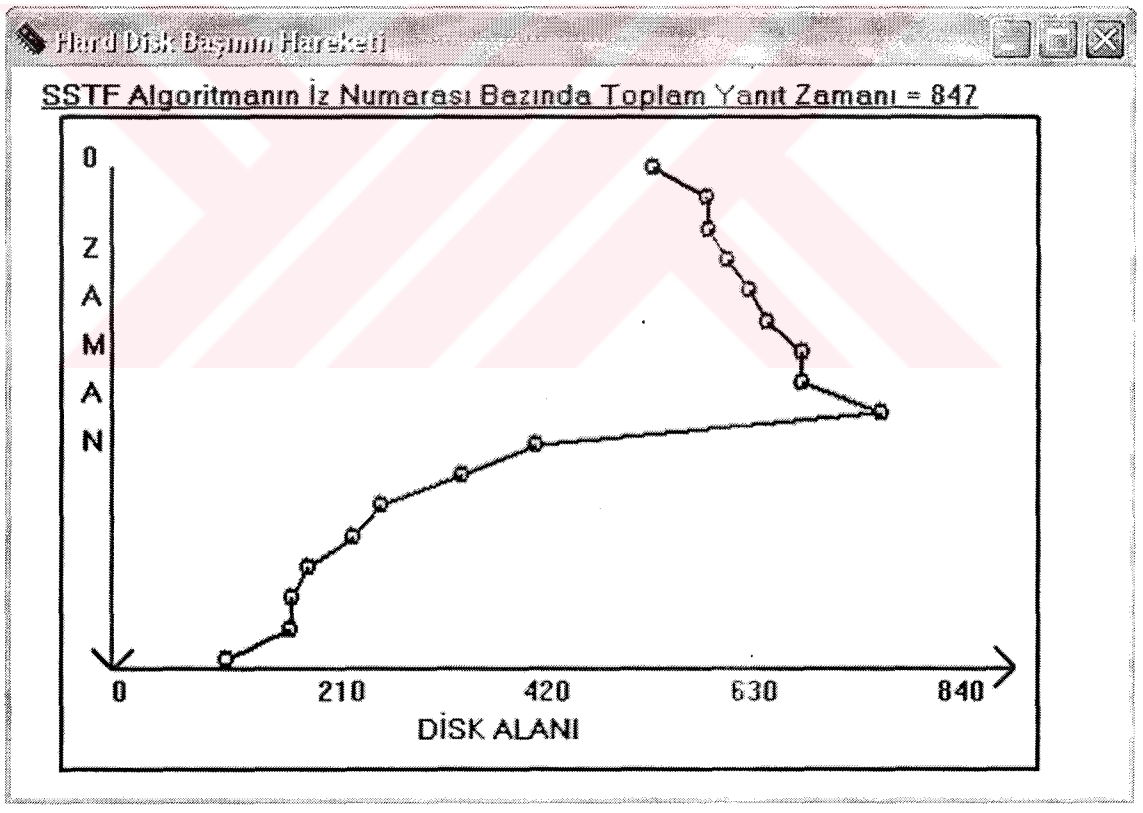


Şekil 4.7. FCFS algoritmasına göre disk kafası hareketi

FCFS algoritmasında disk isteklerinin yanıt zamanı toplamları 2093 ms. olmaktadır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı(msn)
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	(11)	572	1918	3	10,934
2 - Kuyruk[2]:	(1)	574	1553	9	16,764
3 - Kuyruk[3]:	(8)	591	1385	7	24,951
4 - Kuyruk[4]:	(13)	612	1055	0	33,538
5 - Kuyruk[5]:	(12)	629	1218	6	41,725
6 - Kuyruk[6]:	(2)	662	1898	0	51,323
7 - Kuyruk[7]:	(10)	663	1249	1	56,793
8 - Kuyruk[8]:	(16)	738	1873	8	68,927
9 - Kuyruk[9]:	(15)	407	1478	8	89,503
10 - Kuyruk[10]:	(6)	336	1565	8	101,434
11 - Kuyruk[11]:	(9)	259	1019	8	113,668
12 - Kuyruk[12]:	(7)	232	1271	8	122,789
13 - Kuyruk[13]:	(3)	190	1968	6	133,027
14 - Kuyruk[14]:	(4)	174	1965	2	141,107
15 - Kuyruk[15]:	(14)	171	1871	1	147,214
16 - Kuyruk[16]:	(5)	110	1248	5	158,609

Şekil 4.8. SSTF algoritmasına göre disk isteklerinin diziliş sırası

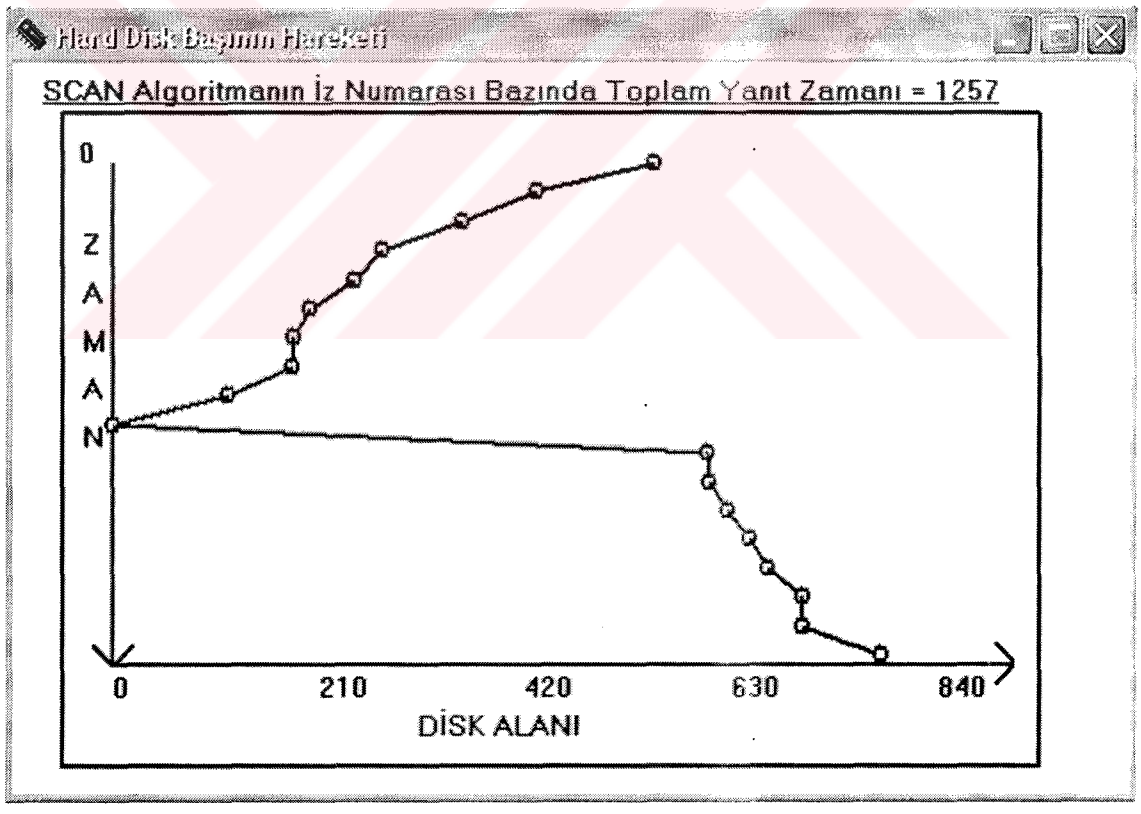


Şekil 4.9. SSTF algoritmasına göre disk kafası hareketi

SSTF algoritmasında disk isteklerinin yanıt zamanı toplamları 1312 ms. olmaktadır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı(msn)
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	[15]	407	1478	8	13,807
2 - Kuyruk[2]:	[6]	336	1565	8	25,738
3 - Kuyruk[3]:	[9]	259	1019	8	37,972
4 - Kuyruk[4]:	[7]	232	1271	8	47,093
5 - Kuyruk[5]:	[3]	190	1968	6	57,331
6 - Kuyruk[6]:	[4]	174	1985	2	65,411
7 - Kuyruk[7]:	[14]	171	1871	1	71,518
8 - Kuyruk[8]:	[5]	110	1248	5	82,913
9 - Kuyruk[9]:	[11]	572	1918	3	107,157
10 - Kuyruk[10]:	[1]	574	1553	9	112,987
11 - Kuyruk[11]:	[8]	591	1385	7	121,174
12 - Kuyruk[12]:	[13]	612	1055	0	129,761
13 - Kuyruk[13]:	[12]	629	1218	6	137,948
14 - Kuyruk[14]:	[2]	662	1698	0	147,546
15 - Kuyruk[15]:	[10]	663	1249	1	153,016
16 - Kuyruk[16]:	[16]	738	1873	8	165,15

Şekil 4.10. SCAN algoritmasına göre disk isteklerinin diziliş sırası

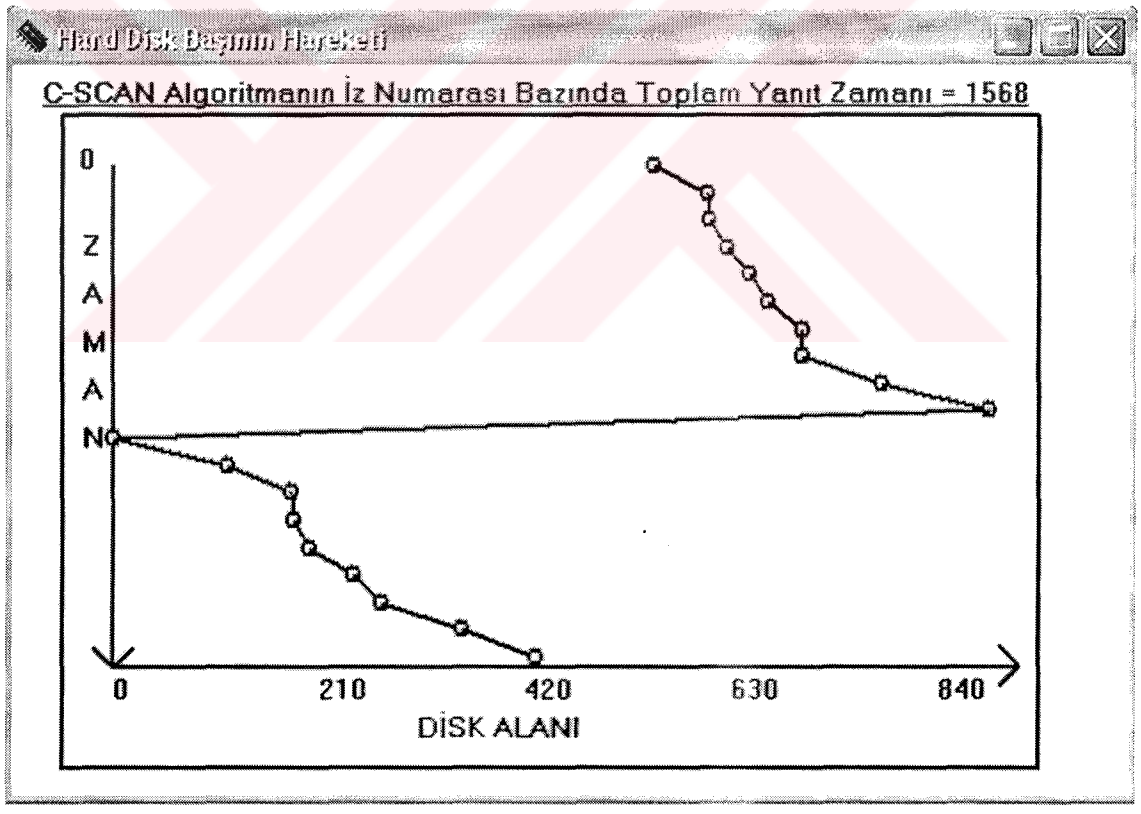


Şekil 4.11. SCAN algoritmasına göre disk kafası hareketi

SCAN algoritmasında disk isteklerinin yanıt zamanı toplamları 1477 ms. olmaktadır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı(msn)
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	[11]	572	1918	3	10,934
2 - Kuyruk[2]:	[1]	574	1553	9	16,764
3 - Kuyruk[3]:	[8]	591	1385	7	24,951
4 - Kuyruk[4]:	[13]	612	1055	0	33,538
5 - Kuyruk[5]:	[12]	629	1218	6	41,725
6 - Kuyruk[6]:	[2]	662	1898	0	51,323
7 - Kuyruk[7]:	[10]	663	1249	1	56,793
8 - Kuyruk[8]:	[16]	738	1673	8	68,927
9 - Kuyruk[9]:	[5]	110	1248	5	97,819
10 - Kuyruk[10]:	[14]	171	1871	1	109,214
11 - Kuyruk[11]:	[4]	174	1965	2	115,321
12 - Kuyruk[12]:	[3]	190	1968	6	123,401
13 - Kuyruk[13]:	[7]	232	1271	8	133,639
14 - Kuyruk[14]:	[9]	259	1019	8	142,76
15 - Kuyruk[15]:	[6]	336	1565	8	154,994
16 - Kuyruk[16]:	[15]	407	1478	8	166,925

Şekil 4.12. CSCAN algoritmasına göre disk isteklerinin diziliş sırası

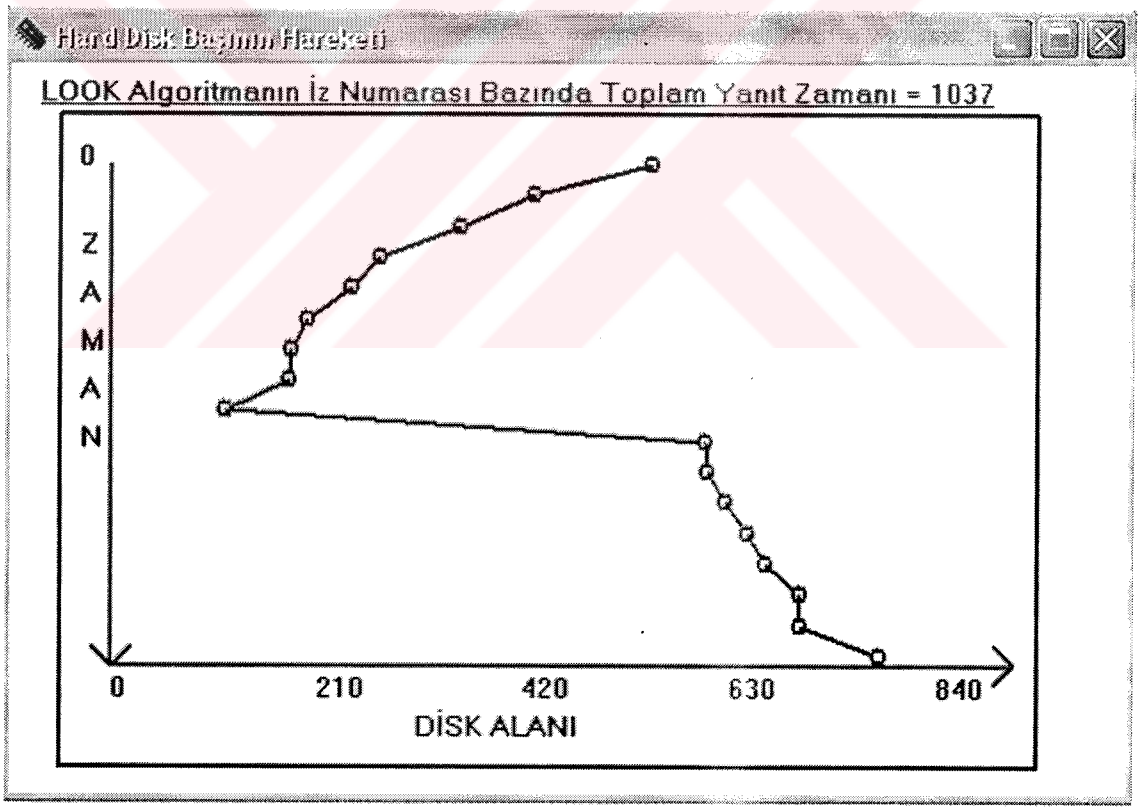


Şekil 4.13. CSCAN algoritmasına göre disk kafası hareketi

CSCAN algoritmasında disk isteklerinin yanıt zamanı toplamları 1349 ms. olmaktadır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı[msn]
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	[15]	407	1478	8	13,807
2 - Kuyruk[2]:	[6]	336	1565	8	25,738
3 - Kuyruk[3]:	[9]	259	1019	8	37,972
4 - Kuyruk[4]:	[7]	232	1271	8	47,093
5 - Kuyruk[5]:	[3]	190	1968	6	57,331
6 - Kuyruk[6]:	[4]	174	1985	2	65,411
7 - Kuyruk[7]:	[14]	171	1871	1	71,518
8 - Kuyruk[8]:	[5]	110	1248	5	82,913
9 - Kuyruk[9]:	[11]	572	1918	3	107,157
10 - Kuyruk[10]:	[1]	574	1553	9	112,987
11 - Kuyruk[11]:	[8]	591	1385	7	121,174
12 - Kuyruk[12]:	[13]	612	1055	0	129,761
13 - Kuyruk[13]:	[12]	629	1218	6	137,948
14 - Kuyruk[14]:	[2]	662	1898	0	147,546
15 - Kuyruk[15]:	[10]	653	1249	1	153,016
16 - Kuyruk[16]:	[16]	738	1873	8	165,15

Şekil 4.14. LOOK algoritmasına göre disk isteklerinin diziliş sırası

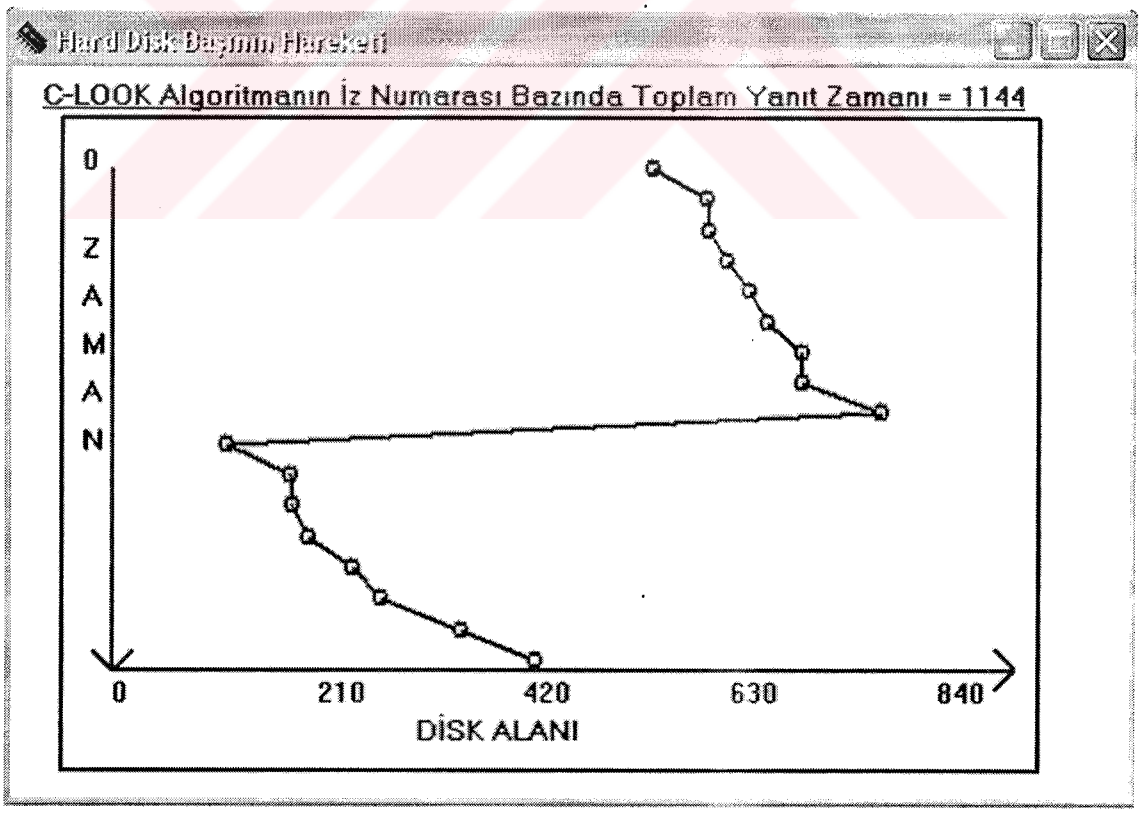


Şekil 4.15. LOOK algoritmasına göre disk kafası hareketi

LOOK algoritmasında disk isteklerinin yanıt zamanı toplamları 1477 ms. olmaktadır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı[msn]
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	[11]	572	1918	3	10,934
2 - Kuyruk[2]:	[1]	574	1553	9	16,764
3 - Kuyruk[3]:	[8]	591	1385	7	24,951
4 - Kuyruk[4]:	[13]	612	1055	0	33,538
5 - Kuyruk[5]:	[12]	629	1218	6	41,725
6 - Kuyruk[6]:	[2]	662	1898	0	51,323
7 - Kuyruk[7]:	[10]	663	1249	1	56,793
8 - Kuyruk[8]:	[16]	738	1873	8	68,927
9 - Kuyruk[9]:	[5]	110	1248	5	97,819
10 - Kuyruk[10]:	[14]	171	1871	1	109,214
11 - Kuyruk[11]:	[4]	174	1985	2	115,321
12 - Kuyruk[12]:	[3]	190	1958	6	123,401
13 - Kuyruk[13]:	[7]	232	1271	8	133,639
14 - Kuyruk[14]:	[9]	259	1019	8	142,76
15 - Kuyruk[15]:	[6]	336	1565	8	154,994
16 - Kuyruk[16]:	[15]	407	1478	8	166,925

Şekil 4.16. CLOOK algoritmasına göre disk isteklerinin diziliş sırası

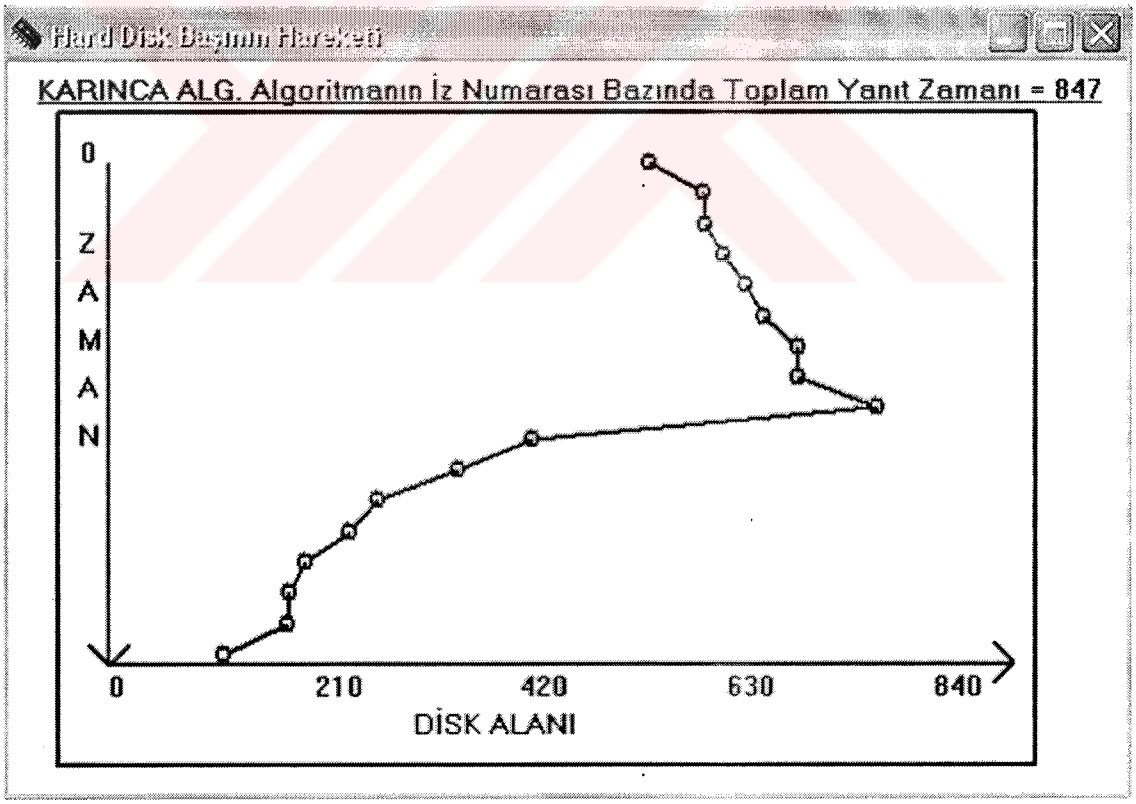


Şekil 4.17. CLOOK algoritmasına göre disk kafası hareketi

CLOOK algoritmasında disk isteklerinin yanıt zamanı toplamları 1349 ms. olmaktadır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı[msn]
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	(11)	572	1918	3	10,934
2 - Kuyruk[2]:	(1)	574	1553	9	16,764
3 - Kuyruk[3]:	(8)	591	1385	7	24,951
4 - Kuyruk[4]:	(16)	738	1873	8	40,099
5 - Kuyruk[5]:	(10)	663	1249	1	52,234
6 - Kuyruk[6]:	(2)	662	1898	0	57,704
7 - Kuyruk[7]:	(12)	629	1218	6	67,302
8 - Kuyruk[8]:	(13)	612	1055	0	75,489
9 - Kuyruk[9]:	(15)	407	1478	8	92,545
10 - Kuyruk[10]:	(6)	336	1565	8	104,476
11 - Kuyruk[11]:	(9)	259	1019	8	116,71
12 - Kuyruk[12]:	(7)	232	1271	8	125,831
13 - Kuyruk[13]:	(3)	190	1968	6	136,069
14 - Kuyruk[14]:	(4)	174	1985	2	144,149
15 - Kuyruk[15]:	(14)	171	1871	1	150,256
16 - Kuyruk[16]:	(5)	110	1248	5	161,651

Şekil 4.18. Karınca algoritmasına göre disk isteklerinin diziliş sırası

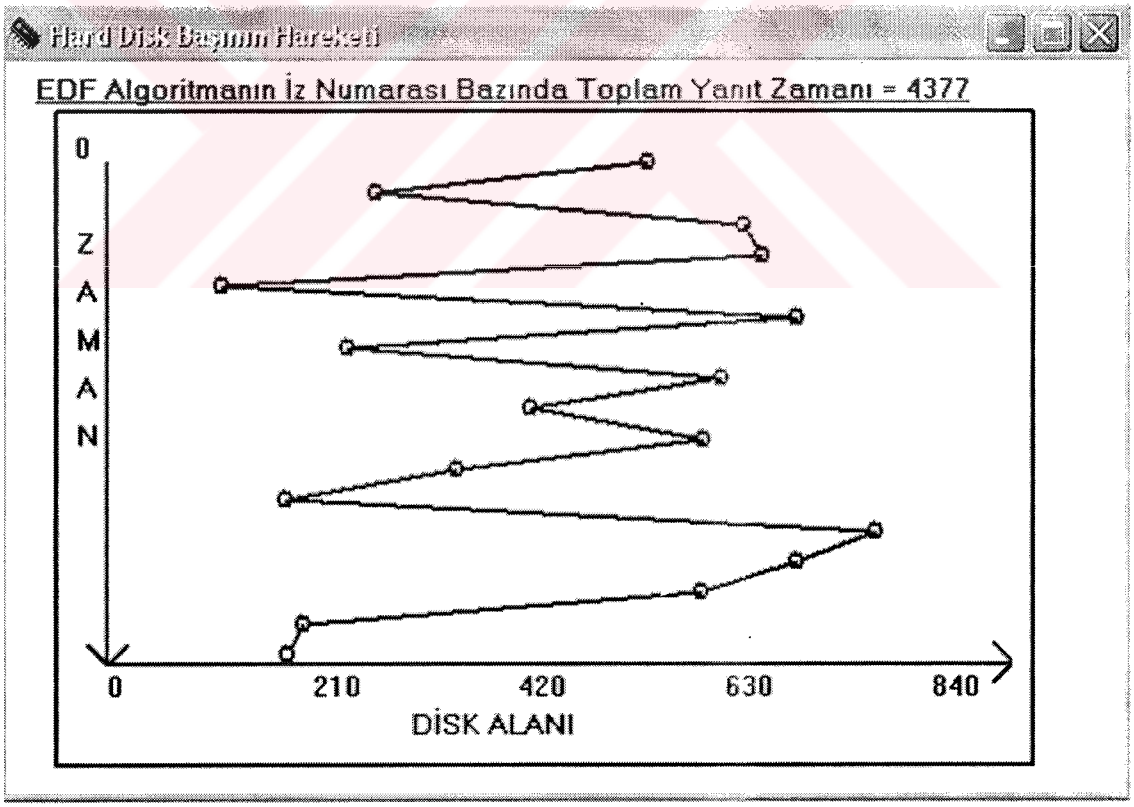


Şekil 4.19. Karınca algoritmasına göre disk kafası hareketi

Karınca algoritmasında disk isteklerinin yanıt zamanı toplamları 1312 ms. olmaktadır. Verilen örnek için karınca algoritmasının parametreleri *KarıncaSayısı*=50, *BuharlaşmaKatsayısı*=0,8 *ve İterasyonSayısı*=50 olarak alınmıştır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı(msn)
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	[9]	259	1019	8	18,588
2 - Kuyruk[2]:	[13]	612	1055	0	39,78
3 - Kuyruk[3]:	[12]	629	1218	6	47,967
4 - Kuyruk[4]:	[5]	110	1248	5	73,807
5 - Kuyruk[5]:	[10]	663	1249	1	100,599
6 - Kuyruk[6]:	[7]	232	1271	8	123,975
7 - Kuyruk[7]:	[8]	591	1385	7	145,335
8 - Kuyruk[8]:	[15]	407	1478	8	161,736
9 - Kuyruk[9]:	[1]	574	1553	9	177,579
10 - Kuyruk[10]:	[6]	336	1565	8	195,601
11 - Kuyruk[11]:	[14]	171	1871	1	211,376
12 - Kuyruk[12]:	[16]	738	1873	8	238,56
13 - Kuyruk[13]:	[2]	662	1898	0	250,745
14 - Kuyruk[14]:	[11]	572	1918	3	263,598
15 - Kuyruk[15]:	[3]	190	1968	6	285,602
16 - Kuyruk[16]:	[4]	174	1985	2	293,682

Şekil 4.20. EDF algoritmasına göre disk isteklerinin diziliş sırası

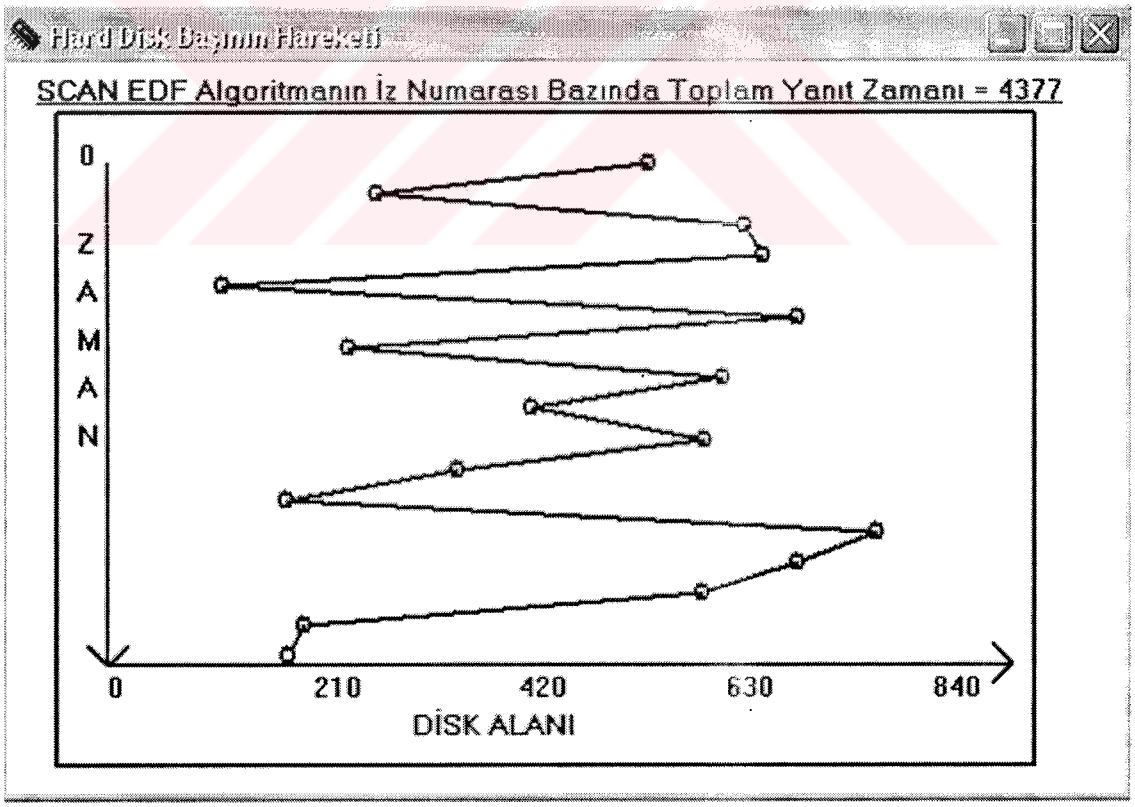


Şekil 4.21. EDF algoritmasına göre disk kafası hareketi

EDF algoritmasında disk isteklerinin yanıt zamanı toplamları 2093 ms. olmaktadır.

Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı(msn)
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	[9]	259	1019	8	18,588
2 - Kuyruk[2]:	[13]	612	1055	0	39,78
3 - Kuyruk[3]:	[12]	629	1218	6	47,967
4 - Kuyruk[4]:	[5]	110	1248	5	73,807
5 - Kuyruk[5]:	[10]	663	1249	1	100,599
6 - Kuyruk[6]:	[7]	232	1271	8	123,975
7 - Kuyruk[7]:	[8]	591	1385	7	145,335
8 - Kuyruk[8]:	[15]	407	1478	8	161,736
9 - Kuyruk[9]:	[1]	574	1553	9	177,579
10 - Kuyruk[10]:	[6]	336	1565	8	195,601
11 - Kuyruk[11]:	[14]	171	1871	1	211,376
12 - Kuyruk[12]:	[16]	738	1873	8	238,56
13 - Kuyruk[13]:	[2]	662	1898	0	250,745
14 - Kuyruk[14]:	[11]	572	1918	3	263,598
15 - Kuyruk[15]:	[3]	190	1968	6	285,602
16 - Kuyruk[16]:	[4]	174	1985	2	293,682

Şekil 4.22. SCANEDF algoritmasına göre disk isteklerinin diziliş sırası

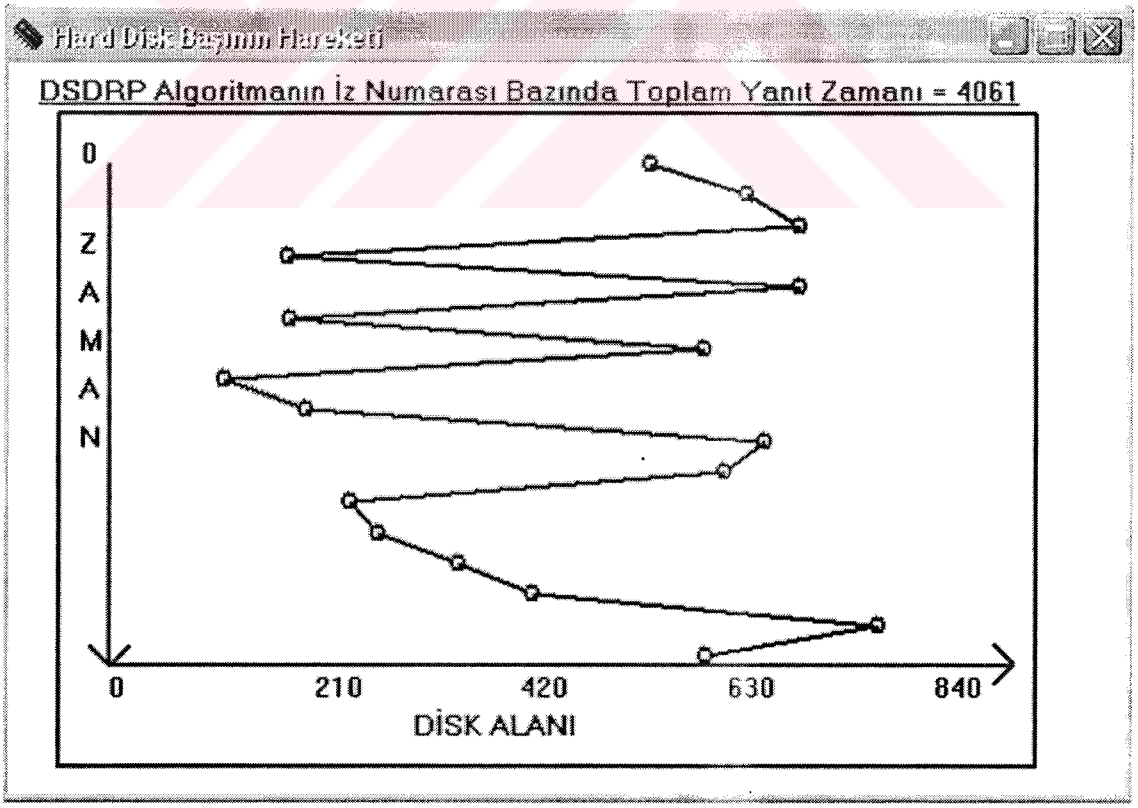


Şekil 4.23. SCANEDF algoritmasına göre disk kafası hareketi

SCANEDF algoritmasında disk isteklerinin yanıt zamanı toplamları 1522 ms. olmaktadır.

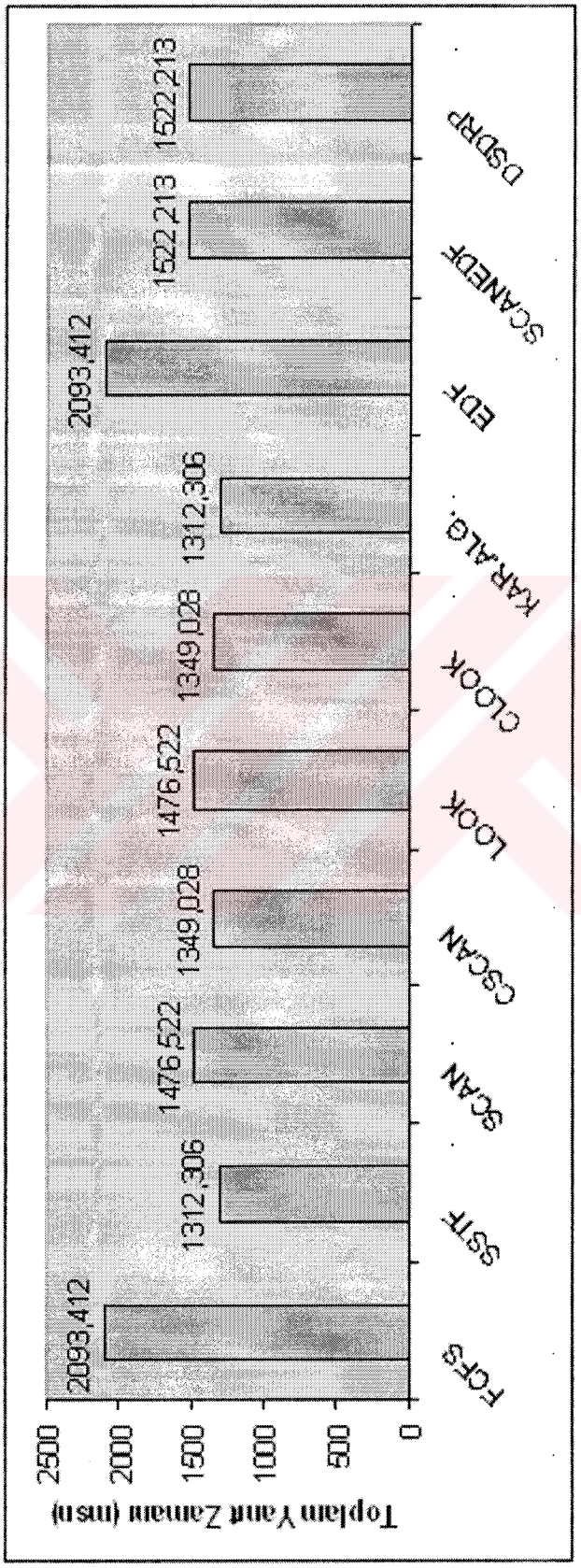
Yeni sıra	Geliş sırası	İz no'su	Son İşlem Zaman değeri	Öncelik değeri	Yanıt zamanı(msn)
Disk kafasının ilk pozisyonu:		519			
1 - Kuyruk[1]:	[13]	612	1055	0	12,99
2 - Kuyruk[2]:	[2]	662	1898	0	23,742
3 - Kuyruk[3]:	[14]	171	1871	1	48,798
4 - Kuyruk[4]:	[10]	663	1249	1	73,882
5 - Kuyruk[5]:	[4]	174	1985	2	98,882
6 - Kuyruk[6]:	[11]	572	1918	3	121,334
7 - Kuyruk[7]:	[5]	110	1248	5	145,578
8 - Kuyruk[8]:	[3]	190	1968	6	157,959
9 - Kuyruk[9]:	[12]	629	1218	6	181,559
10 - Kuyruk[10]:	[8]	591	1385	7	191,522
11 - Kuyruk[11]:	[7]	232	1271	8	212,882
12 - Kuyruk[12]:	[9]	259	1019	8	222,003
13 - Kuyruk[13]:	[6]	336	1565	8	234,237
14 - Kuyruk[14]:	[15]	407	1478	8	246,168
15 - Kuyruk[15]:	[16]	738	1873	8	266,744
16 - Kuyruk[16]:	[1]	574	1553	9	282,485

Şekil 4.24. DSDRP algoritmasına göre disk isteklerinin diziliş sırası



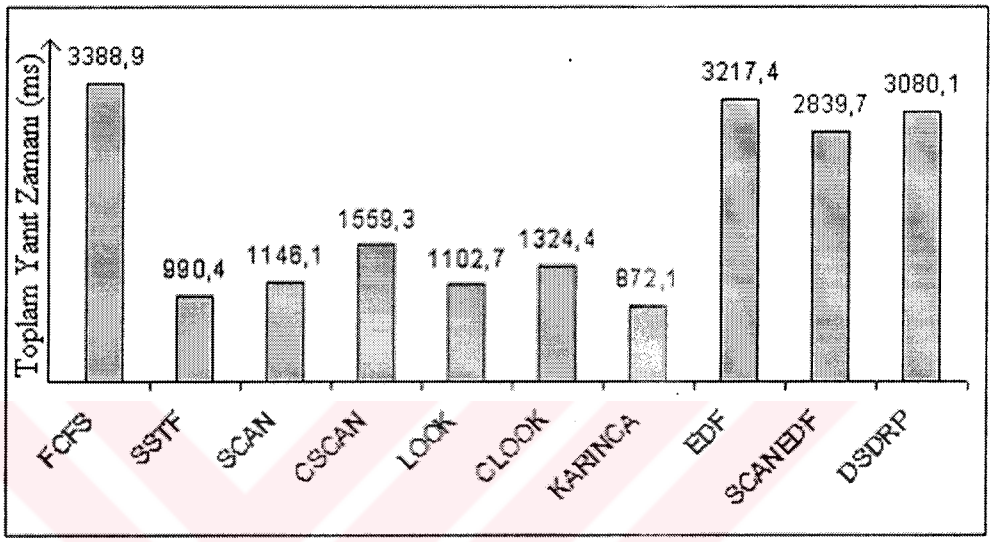
Şekil 4.25. DSDRP algoritmasına göre disk kafası hareketi

DSDRP algoritmasında disk isteklerinin yanıt zamanı toplamları 1522 ms. olmaktadır.

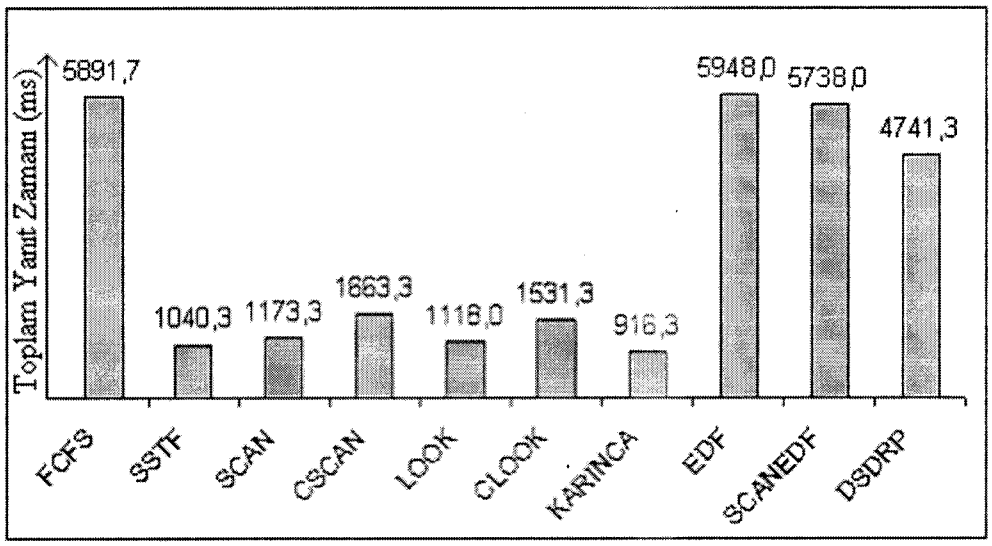


Şekil 4.26. Verilen örnek için algoritma sonuçlarının karşılaştırılması

Disk istek kuyruğunu büyüklüğünü küçük, orta ve büyük olmak üzere üç ana sınıfta gruplayabiliriz [9]. Bu durumda algoritma sonuçlarını almak için küçük, orta, büyük ölçek olarak tanımlayabileceğimiz 10, 20, 50 istek uzunluğundaki disk kuyruklarını kullanabiliriz.



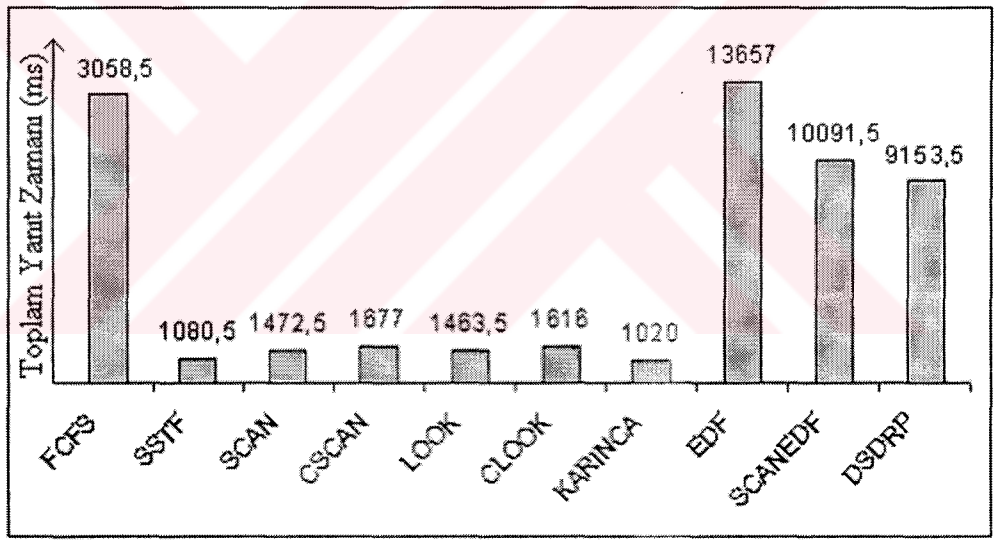
Şekil 4.27. 10 Disk isteği uzunluğundaki kuyruk için algoritma sonuçlarının ortalaması



Şekil 4.28. 20 Disk isteği uzunluğundaki kuyruk için algoritma sonuçlarının ortalaması

Karınca algoritması her koşulduğunda birbirine yakın olsa da farklı değerler verebilmektedir. Bu durum göz önüne alınarak çalışma sonuçları 10 koşmanın ortalaması olarak alınmıştır. Kuyruğa gelen disk isteklerinin iz numarası, öncelik değerleri ve son işlem zamanları model aldığımız disk için 10 adımda rastgele değerler üretilerek yapılmıştır.

Şekil 4.27. ve 4.28.'deki sonuçlar için karınca algoritmasında parametreler: *KarıncaSayısı*=50, *BuharlaşmaKatsayısı*=0,8 *ve İterasyonSayısı*=50 olarak alınmıştır. Şekil 4.29.'da ise bu değerler sırasıyla 75, 0,8 ve 75 olarak alınmıştır. Bu parametreler, aynı disk istekleri için karınca algoritmasının farklı koşulmalarındaki verdiği sonuçlar arasındaki farkın yüzde 5'i geçmemesini sağlayan yaklaşık değerlerin seçilmesiyle elde edilmiştir.



Şekil 4.29. 50 Disk isteği uzunluğundaki kuyruk için algoritma sonuçlarının ortalaması

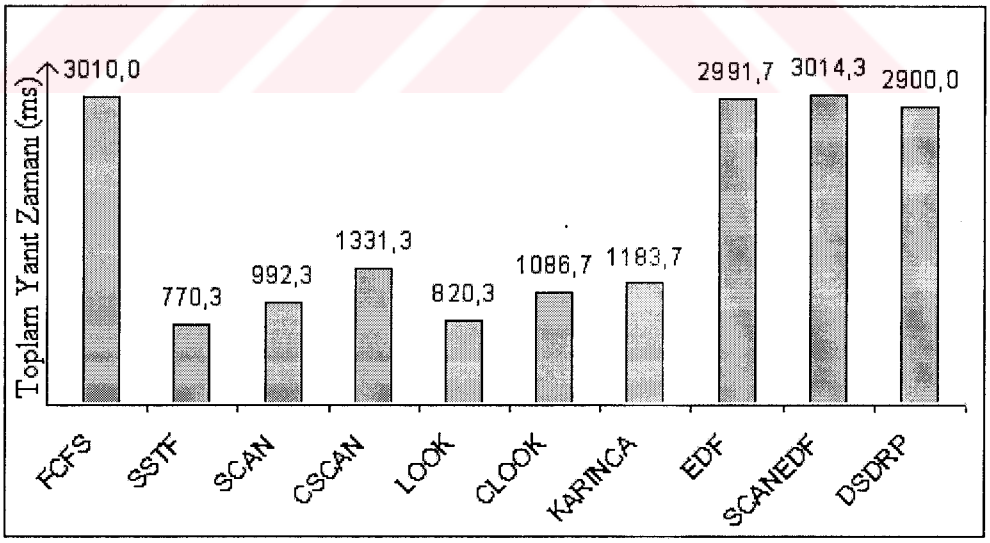
4.4. Birden Fazla Sayıda Parametre kullanılarak KKO Algoritmasının Disk Planlama Problemine Uygulanması

Disk planlama probleminde disk isteklerini tanımlayan, iz numarası haricinde diğer önemli parametreler istek önceliği ve son işlem zamanıdır. Problem çözümüne getirilen klasik tekniklerin çoğu bu iki parametreyi dikkate almamaktadır [14]. Bu parametre değerlerini çözüme yansıtmak için literatürde probleme özgü teknikler geliştirilmiştir

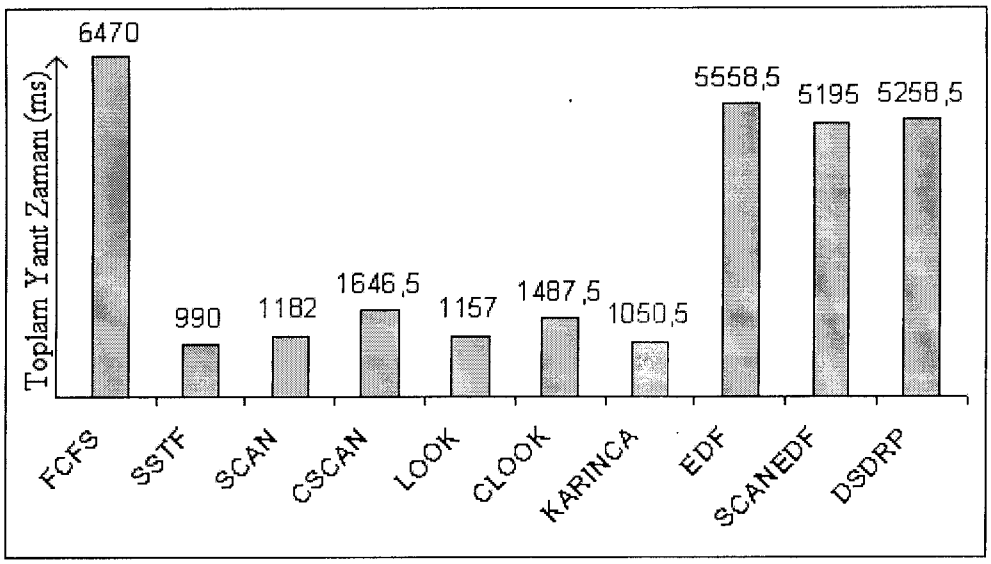
[15][16][17][18]. Ancak bu çözümlerin çoğu dosya sunucuları, gerçek zamanlı sistemler, veri tabanı sunucuları gibi belli amaçlara hizmet edecek türde geliştirilen algoritmalar olup genel ve esnek bir yaklaşımdan uzaklaşmışlardır.

Karınca algoritmasının disk planlama problemine uyarlanmasıyla disk isteğinin iz numarasının yanında öncelik ve son işlem bilgisini de dikkate alarak yeni bir çözüm tarzı geliştirilmiştir. Eşitlik 4.5' de verilen *x, y* ve *z* ağırlıklarına dinamik yapıda değişen değerler verilerek disk isteğinin belirtilen üç parametresinden istenilenler birbirlerine göre daha etkin veya zayıf hale getirilebilirler. Eğer disk istek öncelikleri bizim için disk kafasının aldığı toplam mesafeden daha az önem taşıyorsa, öncelik bilgisinin ağırlığını ifade eden *y* değişkeni diğerlerine nazaran daha küçük değere yüklenir. Bu bölümde disk isteklerinin parametreleri eşit önemde kabul edilerek sonuçlar alınmaktadır. Bunun için karınca algoritmasında *x, y* ve *z* ağırlıklarına eşit değerler verilmiştir..

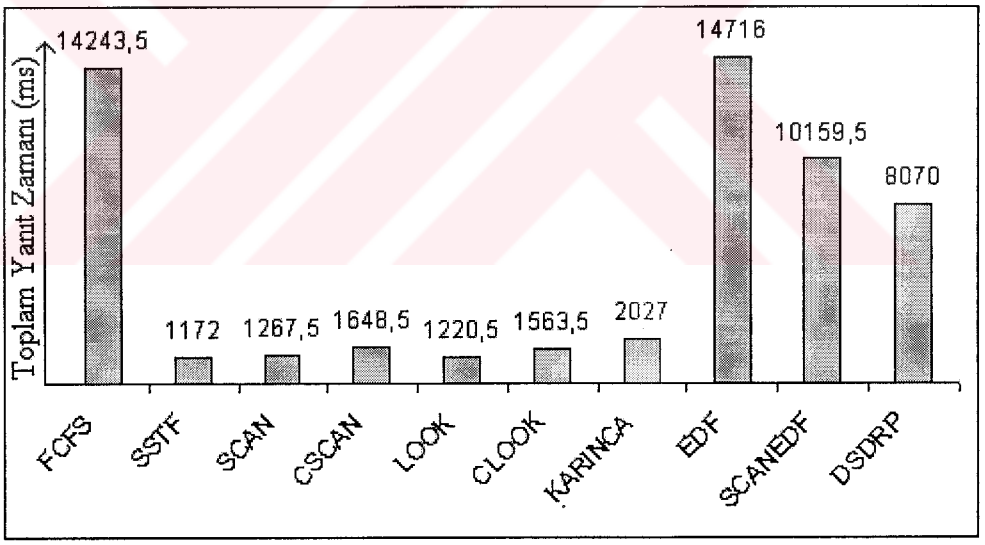
Yine disk istek kuyruğu büyüklükleri küçük, orta ve büyük olmak üzere üç ana sınıfta gruplanarak, 10 farklı koşmanın ortalaması alınmaktadır.



Şekil 4.30. 10 Disk isteği uzunluğundaki kuyruk için algoritma sonuçlarının ortalaması



Şekil 4.31. 20 Disk isteği uzunluğundaki kuyruk için algoritma sonuçlarının ortalaması



Şekil 4.32. 50 Disk isteği uzunluğundaki kuyruk için algoritma sonuçlarının ortalaması

Şekil 4.30. ve 4.31.'deki sonuçlar için karınca algoritmasında parametreleri *KarıncaSayısı*=50, *BuharlaşmaKatsayısı*=0,8 *ve İterasyonSayısı*=50 olarak alınmıştır. Şekil 4.29.'da ise bu değerler sırasıyla 75, 0,8 ve 75 olarak alınmıştır. Yine bu parametreler, aynı disk istekleri için karınca algoritmasının farklı koşulmalarındaki verdiği sonuçlar arasındaki farkın yüzde 5'i geçmemesini sağlayan yaklaşık değerlerin seçilmesiyle elde edilmiştir.

BÖLÜM 5

SONUÇLAR

Bu çalışmada disk planlama problemine çözüm getirilmesi amacıyla karınca kolonisi optimizasyon algoritmasına dayalı yeni bir yaklaşım geliştirilmiştir.

Disk planlama probleminde amaçlanan, istenen parametreler dikkate alınmak suretiyle yine istenen kriterler doğrultusunda en uygun sıralama işleminin yapılmasıdır. Disk kuyruğu üzerinde disk isteklerinin en iyi sıralanması işlemi bir optimizasyon problemidir. Problem yapısına uygun bir gelişime dayalı optimizasyon algoritması olan karınca kolonisi algoritması, problemin getirdiği ilgili parametre ve kriterlere uyarlanarak yapılandırılmıştır. Algoritma başarısını test etmek amacıyla literatürde taranan ve toplanan en yaygın disk planlama teknikleriyle birlikte aynı veriler üzerinde sonuçlar alınmıştır. Bu sonuçlar Delphi ortamında geliştirilen simülasyon programı üzerinde, belirtilen aralıklarda üretilen rastgele disk isteği verileri ile elde edilmiştir. Genel bir performans analizi için bu verilerin on ayrı koşmada elde edilen ortalama sonuçları alınmıştır. Performans testleri küçük, orta ve büyük disk isteği kuyruğu şeklinde üç ayrı grupta toplanmıştır.

Tek parametre kullanılarak çalıştırılan yapıda, klasik teknikler içinde bulunan en iyi sonucun; küçük disk istek kuyruklu problemlerde %12, orta disk istek kuyruklu problemlerde %12 ve büyük disk istek kuyruklu problemlerde ise %6'sı oranında daha iyi sonuç elde edilmiştir (Şekil 4.27, Şekil 4.28, Şekil 4.29). Çok parametreli bir yapıda ise yine klasik tekniklere nazaran küçük, orta ve büyük disk kuyruklu problemler için daha iyi performans sonuçları elde edilmiştir.

Klasik teknikler içerisinde disk isteği ile ilgili tüm parametreleri kullanan genel bir yaklaşım geliştirilememesi, karınca algoritmasını bu konuda da avantajlı bir duruma getirmektedir. Literatürde genelde problemin bir boyutu alınarak sadece bir parametreyi optimize eden algoritma kullanılmaktadır. SCANEDF gibi birden fazla parametreli teknikler ise performansta önemli ölçüde kayıplar vermektedir. Karınca algoritmasının performansı aynı parametreleri kullanan diğer tekniklere göre oldukça yüksek değerdedir (Şekil 4.30, Şekil 4.31, Şekil 4.32). Algoritmada parametre sayısı kolayca artırılabildiği ve hatta bu parametrelerin çözüme etkilerinin ağırlıklarının ayarlanabildiği de göz önüne alınırsa, disk planlama problemi için bu algoritmanın esneklik ve kolay uyarlanabilirlik özelliğinden dolayı uygunluğu anlaşılmaktadır.

Sonuç olarak harddisk performansını artırmak üzere disk kafası ve plakaları için fiziksel bir gelişme yapmaksızın verilerin ne tür bir yaklaşımla alınacağı konusunda geliştirilen metotlara ilave olarak probleme uygun, kolay uyarlanabilir, daha iyi performans sağlayan bir optimizasyon algoritması, karınca koloni optimizasyonu kullanarak geliştirilmiştir. Algoritma işletim sistemi üzerinde harddisk sürücüsü yazılımına veya harddisk aygıtı üzerinde kontrol mekanizması yazılımına ilave olarak yüklenebilir. Gelecek çalışmalarda, bu algoritmanın daha hızlı bir şekilde çalışmasının sağlanabilmesi için algoritmanın donanım üzerine yüklenerek çalıştırılması ve bu işleminde harddisk işlemci ünitesine veya alan programlanabilir kapı dizilerine(FPGA) yüklenerek gerçekleştirilmesi konusunda araştırma yapılması ve çalışmaların uygulamaya dökülmesi hedeflenmektedir.

KAYNAKLAR

1. Dorigo, M., Di Caro, G. and Gambardella, L.M., Ant Algorithms for Discrete Optimization. Artificial Life, 5,2, pp. 137-172, 1999.
2. Dorigo, M., and Di Caro, G., Ant colony optimization a new meta-heuristic, Congress on Evolutionary Computation, pp. 1470-1477, July 6-9, 1999.
3. Unix System Administration, University Technology Services, The Ohio State University, 1971 Neil Avenue, Columbus, OH 43210, http://wks.uts.ohio-state.edu/sysadm_course/html/sysadm-27.html#HEADING27-0, Aug. 1996
4. Silberschatz, A., Galvin, P.B., Operating System Concepts, pp.373-438, John Wiley & Sons, Inc., ISBN 0-471-36414-2, New York USA, 1997
5. Yeh, T., Kuo, C., Lei, C. and Yen, H., Competitive analysis of on-line disk scheduling, technical report, Dept. of Electircal Engineering, National Taiwan University, 1995.
6. Li, Y., Tan, S.M., Chen, Z. and Campbell, R.H., Disk scheduling with dynamic request priorities, technical report, University of Illinois at Urbana- Champaign, IL, August 1995.
7. Gopalan, K. and Chiueh, T., Real-time Disk Scheduling Using Deadline Sensitive SCAN, technical report TR-92, Experimental Computer Systems Labs, Dept. of Computer Science, SUNY at Stony Brook, Stony Brook, NY, USA, Jan 2001.
8. Worthington, G., Ganger, R. and Patt, Y. N., Scheduling algorithms for modern disk drives, Proceedings of the ACM Sigmetrics, pp. 241-251, May 1994.
9. Seltzer, M., Chen, P. and Ousterhour, J., Disk scheduling revisited. Winter USENIX Technical Conference Washington, DC, 22 26 January 1990.
10. Dorigo, M., Maniezzo, V., et al., Ant System An Autocatalytic Optimizing Process., Dipartimento di Elettronica e Informarzione, Politecnico di Milano. 1991
11. Schoonderwoord, R., Holland, O., et al., Ant-based load balancing in telecommunications Networks, Hewlett-Packard Laboratories Bristol, 1996.

12. Silberschatz, A., Galvin, P.B., Operating System Concepts, pp.133-135, John Wiley & Sons, Inc., ISBN 0-471-36414-2, New York, USA, 1997.
13. Fujitsu Limited, M2361A Mini Disk Drive Engineering Specifications, Fujitsu Limited, 1984.
14. Chen, S. And Towsley, D., Scheduling Customers in a Non-Removal Real-Time Systems with an Application to Disk Scheduling, Real-Time Systems, vol 6, pp. 55-72, 1994.
15. Turton, B. C. H . and Arslan, T., A paralel genetic VLSI architecture for combinatorial real-time applications - disc scheduling, In IEE/IEEE Sheeld '95 [851], pp. 493-498, yconf.prog ga95aTurton.
16. Gemmell, D.J., Vin, H.M., Kandlur, D.D., Rangan, P.V. and Rowe, L.A., Multimedia storage servers, A tutorial, IEEE Computer, vol. 28, pp. 40-49, May 1995.
17. Ghandeharizadeh, S., Zimmermann, R., Shi, W., Rejaie, R., Ierardi, D. and Li, T.W., Mitra, a scalable continuous media server, In Multimedia Tools and Applications, pp. 79-108, Kluwer Academic Publishers, July 1997.
18. Bruno, J., Brustoloni, J., Gabber, E., McShea, M., Ozden, B. and Silberschatz, A., Disk Scheduling with Quality of Service Guarantees, In Proceedings of ICMCS99, June 1999.
19. White, T., Pagurek, B., Artificial Life, Adaptive Behavior, Agents Application Oriented Routing with Biologically Inspired Agents, In Proceedings of the Genetic and Evolutionary Computation Conference (GECCQ-99), July 1999.
20. Colorni, A., Dorigo, M. and Maniezzo, V., An Investigation of some Properties of an Ant Algorithm. In Proc of PPSN-92. Elsevier Publishing, 1992.

ÖZGEÇMİŞ

05.07.1977 tarihinde Gemerek'de doğdu. İlk öğrenimini Fatih İlkokulunda, orta öğrenimini İbrahim Yüzbaşıoğlu Ortaokulunda tamamladı. Lise öğrenimini Erzurum Fen Lisesi'nde tamamladı. Lisans eğitimini 1996-2001 yılları arasında Erciyes Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde tamamladı. 2001-2002 öğretim yılında Erciyes Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans eğitimine başladı. Kasım 2001 tarihinde Erciyes Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü'ne Araştırma Görevlisi olarak atandı. Halen bu görevde çalışmaktadır.

İletişim Bilgileri :

Adres : Erc. Üniv. Mühendislik Fak. Bilgisayar Müh. Bölümü Talas Cad. 38039, Kayseri, Türkiye.

Tel. : (90) (352) 4374901-32527

e-posta: okdem@erciyes.edu.tr