T.C.
TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KENDİNİ KLONLAYAN KARINCA KOLONİSİ YAKLAŞIMIYLA OPTİMAL YOLUN BULUNMASI

Şenol Zafer ERDOĞAN

Doktora Tezi

Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. E. Murat ESİN

II. Danışman: Yrd. Doç. Dr. Erdem UÇAR

EDİRNE 2008

T.C.
TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KENDİNİ KLONLAYAN KARINCA KOLONİSİ YAKLAŞIMIYLA OPTİMAL YOLUN BULUNMASI

Şenol Zafer ERDOĞAN

DOKTORA TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez 11/01/2008 tarihinde aşağıdaki jüri tarafından kabul edilmiştir.

Yrd. Doç. Dr. E. Murat ESİN
(I. Danışman)

Yrd. Doç. Dr. Erdem UÇAR
(II. Danışman)

Prof. Dr. A. Mesut RAZBONYALI
(Üye)

Yrd. Doç. Dr. Nurşen SUÇSUZ
(Üye)

Yrd. Doç. Dr. Cavit TEZCAN
(Üye)

Yrd. Doç. Dr. Yılmaz KILIÇASLAN
(Üye)

Doktora Tezi
Trakya Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

ÖZET

Ağ yapısı içinde bir düğümden diğerine veri gönderilirken düğümden düğüme devam eden bir yol izlenir. Verinin hedef düğüme ulaşmak için sırayla geçeceği düğümlerin belirlenmesine yönlendirme adı verilir. Yönlendirme algoritmalarının temel amacı verinin en kısa yoldan gönderilmesinin sağlanmasıdır. Literatürde çok çeşitli yönlendirme algoritmalarına rastlanmaktadır. Bu çalışmada, bilinen yönlendirme algoritmalarına kısa bir bakışın ardından, bir yenilik olarak sunulan kendi yöntemimiz ifade edilmektedir.

Son yıllarda doğadaki canlılardan esinlenerek mühendislik alanlarında birtakım çalışmalar ortaya konmuştur. Bu çalışmalardan en önemlilerinden biri Karınca Koloni Sistemi'dir. Karınca davranışları incelenerek davranışları bir matematiksel model haline getirilmiştir. Sezgisel bir yaklaşım olan Karınca Koloni Sistemi pek çok alanda uygulanmıştır.

Bu tezde, karınca koloni sistemi temelinden hareketle yeni bir yaklaşım ortaya konmaktadır. Karınca davranışlarından esinlenerek özellikleri belirlenen ajanlar ağ içerisinde dolaşmakta ve geçtikleri yolları kendi yapıları içerisinde kayıt etmektedirler. Karınca kolonileri temelli yaklaşımlarda esas sorunlardan birisi, ortamda var olacak karıncaların optimum sayısı ve sistemde dolaşma süreleridir. Bu çalışmada karınca kavramı ile beraber sunulan klon kavramı, bu sayı ve zaman problemini de kendiliğinden çözmektedir. Klon, bir canlının tüm özellikleri ve bilgileri ile bir kopyasının yaratılmasıdır. Karıncalar ağ içerisinde karşılaştıkları durumlara göre ya kendilerini klonlamakta ya da kendilerini yok etmektedirler. Bu şekilde gereksiz olan karıncalar bir süre sonra ağ içerisinde kendilerini yok etmektedir. Bu yaklaşım farklı ağ yapıları içerisinde uygulanmış ve elde edilen sonuçlar çalışmada yer almıştır.

Tezin ilk bölümünde, yönlendirme temelleri, yönlendirme algoritmaları ve karınca koloni sistemi hakkında giriş niteliğinde bilgi verilmiştir. İkinci bölümde, graf üzerinde arama yöntemleri ve algoritmaları incelenmiştir. Üçüncü bölümde yönlendirme temelleri ve kullanılan önemli yönlendirme protokolleri hakkında bilgi verilmiştir. Dördüncü bölümde, Karınca Koloni sistemi ve matematiksel model detaylı şekilde gösterilmiş, karınca koloni sisteminin ağ uygulaması olan Antnet anlatılmıştır. Beşinci bölümde, bu tezin literatüre asıl katkısını oluşturan "kendini klonlayan karınca kolonisi yaklaşımı" ayrıntılarıyla açıklanmıştır. Sunulan yaklaşımın simülasyonları ve uygulamaları ise altıncı bölümde sunulmakta ve elde edilen sonuçlar gösterilmektedir. Yedinci bölüm sonuçların özetlenmesine ayrılmıştır.

Anahtar Kelimeler: Yönlendirme Algoritmaları, Karınca Koloni Sistemi, En Kısa Yol Algoritmaları

Doctorate Thesis
Trakya University Graduate School of
Natural and Applied Sciences
Department of Computer Engineering

ABSTRACT

A route is pursued while sending data between the nodes. The determination of the nodes that a data passes through consecutively in order to reach to the destination node is called routing. The main objective of the routing algorithms is to enable data to be sent in the shortest path. In the literature one faces a variety of routing algorithms. In this study following a brief look at the common routing algorithms, a method of our own is presented which is introduced as a contribution.

In recent years, some studies have been presented in the engineering fields inspired by the living organisms in the nature. One of the most important studies of these is Ant Colony System. In this study, the behavior of the ants is converted into a mathematical model by examining their behaviors. The ant colony system which is a metheuristic approach is applied in many fields.

In this thesis, a new approach is presented based on the behavior of the ant colony system. The agents whose qualities were determined inspired by the behavior of the ants, travel within the network and record the routes which they travel through within their own structures. One of the main problems confronted in the ant-based approaches is the optimum number of the ants existing in the environment and their duration of wondering within the system. The clon concept which is introduced together with the ant concept solves this number and duration problem by itself. Cloning refers to the creation of a copy of an organism with all of its qualities and data. The ants either clon or destroy themselves according to the circumstances they face within the network. Thus, redundant

ants destroy themselves within the network. This approach was carried out within the different networks and the results deducted were included in the study.

In the first chapter, introductory information is given about the basis of routing, routing algorithms and the ant colony system. Chapter two introduces the search methods on the graph. Chapter three gives information about the basics of the routing and the main routing protocols. In chapter four, ant colony system and mathematical model have been presented. In chapter five, the main contribution, Self Cloning Ant Colony Approach, has been examined in detail. In chapter six, the simulations and applications of the presented approach have been examined and the results have been presented and discussed. Chapter seven gives a summary of the results.

Keywords: Routing algorithms, ant colony system, shortest path algorithms.

TEŞEKKÜR

Bu tez çalışması sırasında değerli fikirleri ve yardımlarından dolayı tez danışmanım Sayın Prof. Dr. E. Murat ESİN' e teşekkür ederim.

Tez çalışma sürecinde her konuda destek olan ve ikinci danışman görevini üstelenen değerli hocam Sayın Yrd. Doç. Dr. Erdem UÇAR' a teşekkürlerimi sunmak isterim.

Tez izleme komitesinde yer alan ve bana tez çalışmam sürecinde sürekli destek veren Maltepe Üniversitesi Öğretim Üyesi Sayın Prof. Dr. A. Mesut RAZBONYALI Hocama teşekkür ederim.

Ayrıca çalışma hayatımda bana destek olan ve yardımlarını esirgemeyen Bilgisayar Mühendisliği bölüm hocalarıma ve çalışma arkadaşlarıma çok teşekkür ederim.

Son olarak bana yardımlarını esirgemeyen ve bana sonsuz destek veren annem, babam ve kardeşlerime teşekkür etmeyi bir borç bilirim.

Ocak 2008

İÇİNDEKİLER

ŞEKİLLER TABLOSU

ÇİZELGELER TABLOSU

BÖLÜM 1.

GİRİŞ

Günümüzde telekomünikasyon teknolojileri, geçmişte kullanılan yıldız mimarilerinin yerine yoğunlukla grid ağ mimarilerine dayandırılmaktadır. Bu mimaride, iletişim yapmak isteyen birimlere düğüm (node), düğümleri birbirine bağlayan iletişim hatlarına da bağlaç (*link*) adı verilir. Birbirleriyle doğrudan bağlantıları olmayan düğümler, ancak başka düğümler üzerinden bir yol oluşturarak iletişim kurar.

Bu yapıda doğal olarak bir düğümden diğerine iletim sağlayabilecek birden fazla yol tanımlanabilir. Bir düğümden diğerine veri iletmek üzere kullanılabilecek yollardan her birisi üzerinde yer alan, yani geçilmesi gereken düğümlerin listesine yol listesi (*path list*) adı verilir. Düğümler ve düğümleri birbirine bağlayan bağlaçlardan oluşan yapının modeli ağın topolojisi olarak adlandırılır ve birçok matematiksel ve mantıksal işlemin yapılmasına olanak sağlar.

Belli bir düğümden bir başkasına veri aktarımında bulunmak için arada geçilmesi gereken düğümler ve bunları bağlayan yolun belirlenmesi önemli bir problem olarak ortaya çıkar.

Literatürde bu tür problemlerin çözümü için birçok yöntem önerilmiş ve kullanılmıştır. Bu çalışmada, en uygun yolun bulunması ve bu yol bilgisinin dinamik olarak canlı tutulabilmesi için karınca kolonilerini esas alan bir yaklaşım önerilmektedir.

1.1. Graflar

Yol belirleme probleminin çözümünde graf teorisinden faydalanılmaktadır. Noktalar ve çizgilerin oluşturduğu topluluğa graf adı verilir. Graf üzerinde farklı alanlardaki problemler modellenmekte ve problemin çözümü sağlanmaktadır. Bu problemlerin başında birbirinden uzak iki nokta arasında mümkün yolların aranması ve bu mümkün yollar arasında en kısa olanın belirlenmesi gelmektedir. Graf üzerinde çeşitli arama algoritmaları geliştirilmiştir. Derinine Arama Algoritması (*Depth-First Search Algorithm*) ve Enine Arama Algoritması (*Breadth-First Search Algorithm*) en çok bahsedilen arama algoritmalarıdır (Nabiyev, 2007).

Derinine arama algoritmasında, graf üzerinde arama yapılırken öncelikle derinlemesine bakılır ve kökten itibaren en derinde yani uçta bulunan düğüme kadar gidilir. Ulaşılan düğümün hedef düğüm olmadığı anlaşılırsa, başka bir bağlacı olan üstteki bir düğüme geri dönülerek bu bağlaçtan devam edilir (Kruse, 1998).

Enine arama algoritmasında, köke aynı sayıda düğüm uzaklıkta olan düğümler aynı seviyede sayılır. Arama, kökün bir altındaki seviyeden başlar ve bu seviyedeki bütün düğümlerin aranan düğüm olup olmadığı test edilir. Aranan düğüm bulunamamışsa, bir sonraki seviye için aynı yöntem tekrarlanır. Algoritmanın çalışma yapısında İlk Giren İlk Çıkar (*First In First Out*) kuralı çalışır (Kruse, 1998). Bu durumda bir düğümden diğerine giderken var olan listedeki ilk düğümle devam edilir.

Graf üzerinde çözümü gerçekleştirilen önemli bir diğer problem ise en kısa yol problemidir. En çok bahsedilen en kısa yol algoritmalarından bazıları Dijkstra algoritması (Dijkstra, 1959) ve Floyd algoritmasıdır (Floyd, 1967). Dijkstra algoritmasında, negatif bağlaç değeri taşımayan bir grafta kaynak noktası için en kısa yol değeri hesaplanır (Barbehenn, 1998). Floyd algoritması, graf içindeki noktalar arasında en düşük maliyetli yolların bulunmasını sağlar. Düğümler arasındaki bağlaçların maliyet değerlerini gösteren bir komşuluk matrisi üzerinde işlem görür. Bu matrise göre hesaplamaları yaparak en düşük maliyetli yolları bulur.

Graf üzerinde yapılan çalışmalardan bir diğeri ise en küçük kapsayan yol ağacı algoritmalarıdır. Bu algoritmalara Kruskal (Kruskal, 1956), Prim (Prim, 1957) ve Sollin (Boruvka, 1926) algoritmaları örnek verilebilir. Bu algoritmalarda graf üzerinde yer alan

tüm düğümler kapsanır. Algoritmaların genel yapısı içerisinde iki düğüm arasında tek bir yol bulunur. Bu algoritmalar ile graf yapısı içinde yer alan düğümler ve bağlaçlar bir ağaç yapısı içerisinde yeniden düzenlenmektedir. Yaratılan bu ağaç yapısı ile iki nokta arasında, düşük maliyetli yol bulunabilmektedir.

Graf üzerinde geliştirilen bu algoritmalar ile düğümlerle yolların araştırılması ve düğüm çiftleri arasındaki en kısa yolun hesaplanması sağlanmaktadır.

Graf çizilirken düğümler noktalarla, bağlaçlar ise düğümleri birleştiren çizgilerle gösterilir. Graf üzerinde çeşitli arama algoritmaları ve en kısa yol algoritmaları geliştirilmiştir. Bu tezin 2. bölümü bu algoritmaların tanıtımına ayrılmıştır.

1.2. Yönlendirme Protokolleri

Ağ üzerinde bir düğümden diğerine ulaşmak için önerilen yolun elverişliliği bir takım parametrelere bağlıdır ve çoğu zaman en kısa erişim süresini kasteder. Erişim süresi ise; bağlacın uzunluğu, bağlacın band genişliği ile bağlaçtaki trafik yoğunluğuna bağlıdır. Bu parametrelerin bir ya da daha fazlasının aynı anda dikkate alınması ile en uygun yol hesaplamaları için kriter belirlenmiş olur. Bu kriterleri en iyi sağlayan yola en uygun (*optimal*) yol adı verilir.

Bilgisayar ağ topolojilerinde düğümler, ağ üzerinde veri gönderen, veri gönderilen veya üzerinden başkasına ait veriyi aktaran istasyonları temsil eder. Bir düğüm kendisine ait olmayan veriyi kendi yönlendirme tablosuna ve bir sonraki düğüme aktarır. Bu aktarma işini yapmak üzere tasarlanmış aygıtlara yönlendirici (*router*) adı verilir. Yönlendiriciler hedef düğüme doğru yola çıkan paketleri, en uygunluğu belirlenmiş yola yönlendirme işlemini gerçekleştirir. Yönlendirme kararını alırken yukarıda açıklanan metrik kriterler kullanılır. Yönlendiriciler metrik değerlerini elde etmek için ağ içerisinden düğümlerin durumları, düğümler arasındaki bağlaçların durumları, yoğunluk durumları gibi konularda bilgi sahibi olmak zorundadır. Bu bilgiyi elde etmek için

yönlendirme protokolleri kullanılır. Yönlendirme protokolleri, düğümlerin sahip oldukları bilgileri komşu düğümler ile paylaşmasını sağlar.

Yönlendirme protokolleri statik ve dinamik yönlendirme protokolleri olarak ayrılabilir. Statik yönlendirmede, yönlendirme tabloları belli bir tabloya göre oluşturulmakta ve bu tablo değerleri değiştirilmemektedir. Böyle bir yapıda yönlendirme bilgileri önceden bellidir ve yolda bir sorun olursa yönlendirme bilgisi değişmediği için bu yol üzerinden veri akışı gerçekleşemeyecektir. Dinamik yönlendirme protokolleri ise ağ topolojisinde oluşan değişimlere göre yönlendirme tabloları yeniden düzenlenir. Yol üzerinde bir sorun oluşursa yönlendirme tabloları dinamik olarak yeniden düzenlenir.

Dinamik yönlendirme protokolleri, uzaklık vektörü protokolleri ve bağlantı durum protokolleri olarak ayrılmaktadır. Uzaklık vektörü protokolleri için yönlendirme bilgi protokolü-RIP (Edward, 1979) ve IGRP (Cisco, 2001) örnek verilebilir. Bu algoritmalar günümüzde yoğunlukla kullanılmaktadır. Bağlantı durum protokollerine örnek olarak en kısa yol ilk-OSPF (OSPF, 1994) ve IS-IS (RFC, 1992) protokolleri verilebilir.

1.3. Doğal Hayatta Bulunan Canlılardan Esinlenen Karınca Koloni Sistemi

Günümüzde yol belirleme problemi için yeni algoritmalar ortaya çıkmıştır. Algoritmaların bazılarında doğal hayatta bulunan canlılar ilham kaynağı olmuştur. Bu canlıların davranışları modellenerek algoritmalar geliştirilmiştir. Bu algoritmalar içerisinde en çok kullanılanlarından birisi de karınca kolonisi optimizasyonudur (Dorigo, 1992). Gerçek karıncalardan esinlenerek geliştirilen bu algoritma Marco Dorigo tarafından geliştirilmiştir. Gerçek karıncalarla ilgili ilk deneysel çalışmalar 1989 yılında GOSS ve arkadaşları tarafından gerçekleştirilmiştir (Goss vd., 1989; Alaykıran ve Engin, 2005). Bir karınca kolonisi kolektif olarak yuvaları oluşturur ya da yiyecekleri arama işlevlerini yerine getirir (Bonabeau, vd., 2000, Dorigo, vd., 1999).

Karınca kolonisi optimizasyonu basit yapay ajanlar (*agents*) kullanır. Bu karıncalar topoloji üzerinde hareket ederler. Bu sırada topoloji üzerinde yer alan düğümler ve bağlaçlarla ilgili bilgileri toplarlar. Algoritma içerisinde yer alan her karınca bağımsız hareket eder. Karıncalar hareketleri sırasında iletim formu (*stigmergy*) kullanarak birbirleriyle iletişim kurarlar (Dorigo, vd., 1999). İletim formu, karıncaların kendi problem çözme aktivitelerini koordine etmek için kullandıkları indirekt iletim formudur (Dorigo, vd., 1999). Karıncalar, feromen (*pheromone*) adı verilen kimyasal bir maddeyi yere bırakarak iletimi sağlarlar. (Deneubourg, vd., 1990; Sim ve Sun, 2003) Böylece, ortam üzerinde hareket eden karıncalar değişimleri algılayabilirler.

Doğal hayatta bulunan bu karıncaların davranışları modellenerek optimizasyon problemlerine uygulanabileceğini ilk kez Marco Dorigo ve arkadaşları ortaya koymuştur. (Dorigo, 1992)(Gambardella ve Dorigo, 1995)(Dorigo ve Blum, 2005)

Karıncalar, bir düğümden gideceği bir sonraki düğümü bir olasılık fonksiyonu değerine göre seçer. Denklem 1.1.'de bu olasılık fonksiyonu verilmektedir.

$$p_{ij}^{k} = \begin{cases} \dfrac{\left[\tau_{ij}(t)\right]^{\alpha}.\left[\eta_{ij}\right]^{\beta}}{\sum_{k\in A_k}\left[\tau_{ik}(t)\right]^{\alpha}.\left[\eta_{ik}\right]^{\beta}} & k.izin.verilen.bir.seçimse \\ 0 & aksi.halde \end{cases} \qquad (1.1)$$

$\tau_{ij}(t)$: t anında (i,j) köşesindeki feromen izi miktarıdır.

η_{ij} :(i,j) köşeleri arasındaki görünürlük (*visibility*) değeridir. $\eta_{ij} = 1/d_{ij}$ denklemi şeklinde ifade edilebilir.

α : feromen ize verilen bağıl önemi gösteren parametredir.

β : görünürlük değerine verilen önemi gösteren parametredir.

A_k: Henüz seçilmemiş düğüm noktaları kümesidir.(izin verilen *Tabu-listesi*)

Düğümler arasındaki feromen yoğunluğu ve uzaklık değeri fonksiyonun olasılık değerini hesaplarken kullanılan parametrelerdir. Feromen yoğunluğunun yüksek olması o

yolun sık kullanıldığını gösterir ve bir daha seçilme olasılığını arttırır. Aynı şekilde uzaklık değerinin düşük olması o yolun seçilme olasılığını arttıran nedenlerden biridir.

Karınca kolonisi optimizasyonu, pek çok alanda uygulanmıştır. Bilgisayar ve telefon ağı problemlerinin çözümü için AntNET-Karınca Ağı (Caro ve Dorigo, 1998) algoritması geliştirilmiştir. Mobil gezgin ağlarda kullanılmak üzere, "Mobil Gezgin Ağlar için Karınca Yönlendirme Algoritması (ARAMA)" (Hussain ve Saadawi, 2003) geliştirilmiştir. Veri madenciliği uygulamaları için karınca kolonisi optimizasyonu algoritması kullanılmıştır (Parpinelli, vd., 2002; Admane, vd., 2004; Liu, vd., 2004) (Jiang, vd., 2005).

1.4. Kendini Klonlayan Karınca Kolonisi Yaklaşımı

Doğal hayatta bulunan karıncalardan esinlenilerek farklı bir karınca tipi tanımlanmaktadır. Yaklaşım içinde kullanılan karıncalar kendilerini klonlama yeteneğine sahiptir. Günümüzde klon kavramını sıklıkla duymaktayız. Bir canlı klonlandığı zaman klonlanan canlı ile aynı özelliklere sahip yeni bir klon canlı meydana gelmektedir. Bu algoritma içinde kullanılan karıncalar klonlama özelliğine sahiptir. Bu durumda bir karınca kendini klonladığı zaman kendisiyle aynı özelliklere sahip yeni bir klon karınca meydana gelir (Esin ve Erdogan, 2006; Erdogan ve Esin, 2006),

Kendini klonlayan karınca kolonisi yaklaşımıyla bilgisayar ağları üzerinde iki nokta arasındaki en uygun yol bulunabilir. Bunun için önceden ağın tamamının bilinmesine gerek yoktur. Algoritma içinde tanımlanan karıncalar kullanılarak en uygun yol bulunabilmektedir. Karıncalar bağımsız olarak hareket etmekte ve karşılaştıkları duruma göre kendilerini klonlamakta ya da öldürmektedir.

Algoritma, farklı alanlarda uygulanmıştır. İki düğüm arasında en uygun yolun belirlenmesi için kullanılmıştır (Esin ve Erdogan, 2006). Ayrıca yönlendirme tablolarının güncellemesi için kullanılmıştır (Erdogan ve Esin, 2006).

1.5. Tezin Kapsamı

Tez çalışmasının 2. bölümünde yol belirleme problemleri çözmek ve bir graf üzerinde bir düğüme ulaşan en az maliyetli yolu bulmak için ortaya konan algoritmalar ayrıntılı olarak açıklanmaktadır. Graf üzerinde bu algoritmalarının işleyiş adımları açıklanmaktadır. Bu algoritmalar içinde günümüzde de sıklıkla kullanılan Dijkstra Algoritmasının 6. Bölümde simülasyonu gerçekleştirilerek sonuçları tezin asıl çalışması olan "Kendini Klonlayan Karınca Kolonisi Yaklaşımı"yla karşılaştırılmaktadır.

3.bölümde, yönlendirme sürecinin temelleri ve yönlendirme protokolleri ayrıntılı olarak açıklanmaktadır. Yönlendirme protokollerinin neden gerekli olduğu, neleri sağladığı anlatılmaktadır. Yönlendirme protokollerinin sınıflandırılması ve bu sınıflar içinde yer alan en sık kullanılan protokoller ayrıntılı olarak açıklanmaktadır.

4. bölümde, karınca kolonisi optimizasyonu anlatılmaktadır. Karınca kolonisi optimizasyonunun matematiksel modeli ve algoritmanın davranışı ayrıntılı olarak açıklanmaktadır. Bu optimizasyon algoritması kullanılarak geliştirilen yeni uygulamalardan bahsedilmektedir.

5. bölümde, bu tez çalışmasının literatüre asıl katkısını sağlayan kendini klonlayan karınca kolonisi yaklaşımı ve bu yaklaşımla optimal yolun belirlenmesi yöntemi ele alınmaktadır. Yaklaşım içinde kullanılan karınca davranışları ve algoritmanın adımları ayrıntılı olarak anlatılmaktadır.

6. bölümde ortaya konan bu yeni yaklaşımın uluslar arası literatürde örnek olarak kullanılan çeşitli ağlar için, simülasyonları yapılmakta ve simülasyon sonuçları verilmektedir.

Tez çalışmasının son bölümünde, simülasyon sonuçlarının karşılaştırmalı tartışması yapılmaktadır.

BÖLÜM 2.

GRAF ÜZERİNDE ARAMA YÖNTEMLERİ

2.1. Giriş

Bu bölümde bir graf üzerinde, noktalar arasındaki arama yöntemlerinden bahsedilmektedir. En kısa yol algoritmalarından ve en küçük kapsayan ağaç yöntemleri ayrıntılı olarak açıklanmaktadır.

2.1.1. Derinine arama yöntemi (Depth-First Search-DFS)

Derinine algoritması bir graf üzerindeki dolaşma yöntemlerinden biridir (Çölkesen, 2000; Chen ve Shin, 1990; Ibrahim, vd., 2001). Bir graf üzerinde derinlemesine dolaşım gerçekleştirmektedir.

DFS algoritmasının verilen bir $G = (V, E)$ grafı için uygulaması gerçekleştirilmeden önce noktaların bir düzende oluşturulması gerekmektedir. Bu şekilde, eğer bir v noktasına komşu olan iki veya daha fazla nokta varsa ve bu noktaların hiçbiri ziyaret edilmemişse ilk hangi noktaya gidileceğine kesin olarak karar verilebilir (Grimaldi, 2003).

$G = (V, E)$ grafı döngüsüz, bağlı ve yönsüz (*loop-free, connected, undirected*) bir graftır. V, noktaları ve E ise kenarları ifade etmektedir. $|V| = n$ olmak üzere, noktalar $v_1, v_2, v_3, \ldots, v_n$ olarak sıralanmaktadır.

Algoritmanın uygulamasında bir başlangıç düğümü seçilir ve başlangıç düğümünden itibaren gidilebilecek en alt seviyedeki düğüme kadar gidilir. Ulaşılamayan

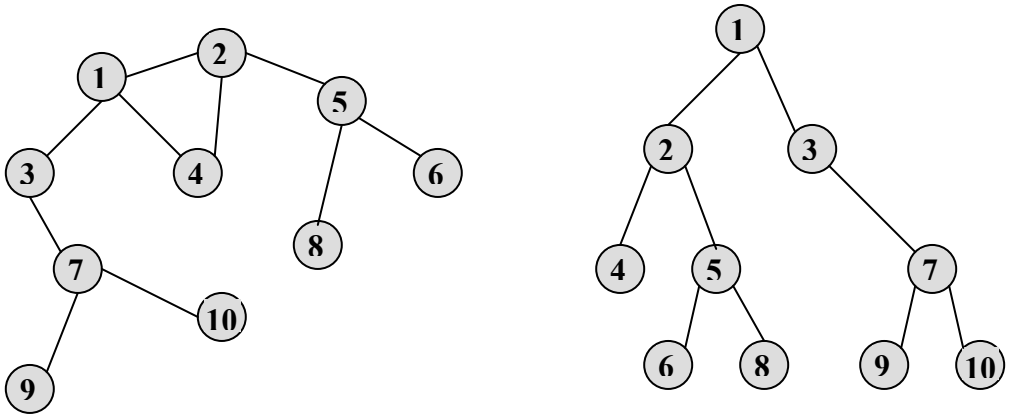
bir düğüm olduğu zaman geriye dönüş yapılarak (*backtrack*) başka bir düğümün ayrıtından itibaren devam edilir (Sakalli, vd., 2006; Stojmenovic, vd., 2000).

Derinine arama algoritmasının sözde kodu aşağıdaki gibidir (Grimaldi, 2003)

1. *v değişkenine v_1 ' i ata ve sadece bu noktayı içeren bir T ağacı oluştur, başlangıç durumuna getir. (v_1 noktası, geliştirilen kapsayan ağacın kök noktası olacaktır.)*
   v_1 noktasını ziyaret et.
2. *$2 \le i \le n$ için en küçük i elemanını seç, öyle ki $\{v, v_i\} \in E$ ve v_i daha önceden seçilmemiş olsun.*

   Eğer böyle bir eleman yoksa 3. adıma git. Aksi durumda, aşağıdaki alt durumları uygula.
   - *T ağacına $\{v, v_i\}$ kenarını ekle ve v_i noktasını ziyaret et.*
   - *v_i noktasını v' ye ata.*
   - *2. adıma dön.*
3. *Eğer $v = v_1$ ise T ağacı belirlenmiş sıra için kapsayan ağaçtır.*
4. *$v \neq v_1$ için, v noktasından T içindeki u noktasına geri dönüş yap. Daha sonra u noktasını v' ye ata ve 2. adıma dön.*

DFS algoritmasının adımları, aşağıda verilen Şekil 2.1. üzerinde açıklanmaktadır.



a) Uygulama Öncesi Graf b) Uygulama Sonrası Graf

Şekil 2.1. DFS uygulaması için bir örnek graf (Grimaldi, 2003)

Şekil 2.1.a)‘ ya bakıldığı zaman uygulama öncesi graf görülmektedir. Düğümlerin düzenleri ise 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 gibidir. Uygulamanın başında 1 numaralı düğüm kök düğüm olarak belirlenir ve *T* ağacı oluşturulur. Daha sonra ilk gelen düğüm 2 numaralı düğümdür ve daha önceden uğranılmamıştır. Bu düğüm ile bağlantılı olan kenar T ağacına eklenir ve 2 numaralı düğümden itibaren devam edilir. 2 numaralı düğümden sonra 4 numaralı veya 5 numaralı düğüme gidilir. Burada ilk önce 4 numaralı düğüm olduğu için ona gidilir. T ağacına 4 numaralı düğüm eklenir. 4 numaralı düğümden ileriye doğru gidiş kapalı olduğundan dolayı 2 numaralı ata düğüme geri dönüş yapar. 2 numaralı düğümden bu sefer 5 numaralı düğüme gidilir. Buradan ise sırasıyla 6 ve 8 numaralı düğümlere ulaşılır ve sonunda en yukarıda olan 1 numaralı kök düğüme geri dönülür. Bu noktadan sonra 3 numaralı düğüme gidilir. 3 numaralı düğümden sırasıyla 7, 9 ve 10 numaralı düğümler dolaşılarak grafın tümü üzerindeki dolaşım tamamlanmış olur. Şekil 2.1.b’ de uygulama tamamlandıktan sonraki yeni durum görülmektedir.

2.1.2. Enine arama yöntemi (Breadth-First-BFS)

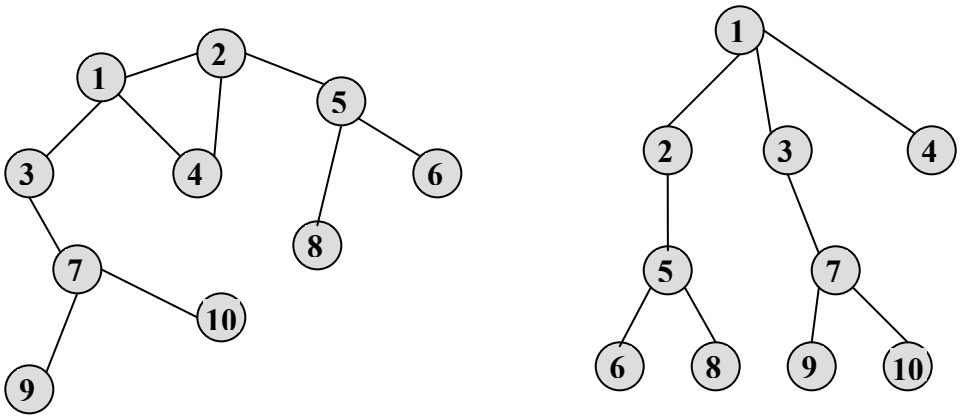
Enine arama algoritması, bir graf üzerinde var olan tüm düğümleri dolaşan bir yöntemdir (Yun, 2002). Enine arama algoritması graf üzerinde öncelikle genişlik araması yapmaktadır (Çölkesen, 2003; Dixon, 1996). Graf üzerinde seviye seviye hareket eder. Dolaşma sürecinde noktalara topolojik bir düzende yaklaşılmaktadır (Kruse, 1998, 2000; Das, vd.,1992).

Algoritmada bir başlangıç düğüm seçilir ve süreç bu düğümden itibaren başlar. Başlangıç düğümün sahip olduğu komşuluklar bulunur ve süreç sırayla bu komşu düğümlerden devam eder. Ara düğümlerde, başlangıç düğümünde olduğu gibi komşu düğümler bulunur ve sırayla daha önce ziyaret edilmemiş bu komşu düğümler ziyaret edilir. Düğüm isimleri FIFO (*First In First Out*- İlk Giren İlk Çıkar) yapısına sahip bir kuyruk içerisinde tutulmaktadırlar. Bu şekilde “ilk giren ilk çıkar” mantığına göre

düğümler ziyaret edilmektedir. BFS algoritmasının sözde kodu aşağıdaki gibidir (Grimaldi, 2003).

1. *v_1 düğümünü Q kuyruğunun sonuna ekle ve T ağacını sanki tek bir düğümden meydana geliyormuş gibi başlangıç durumuna getir. v_1 düğümünü ziyaret et.*
2. *Q kuyruğu boş olmadıkça kuyruğun ön tarafından v düğümünü sil. v düğümüne komşu olan v_i $(2 \le i \le n)$ düğümü eğer ziyaret edilmemişse aşağıdaki adımları uygula.*
    - *Q kuyruğunun sonuna v_i düğümünü ekle.*
    - *T ağacına $\{v, v_i\}$ kenarını ekle.*
    - *v_i düğümünü ziyaret et. (Eğer Q kuyruğundaki tüm düğümler kullanılmışsa ve yeni bir kenar yoksa T ağacı verilen sıra düzen için bir kapsayan ağaçtır.)*

Başlangıç düğümün komşuları sırası ile kuyruğun sonuna eklenir ve kuyruktaki ilk düğüm kuyruktan çıkarılarak başlangıç düğüm ile kuyruktan çıkan ilk düğümü birleştiren kenar eklenir. Düğümün komşuları kuyruğun sonuna eklenir. Daha sonra kuyruktaki bir sonraki düğüm kuyruktan çıkarılarak devam edilir. Algoritmanın uygulaması aşağıdaki Şekil 2.2.'de açıklanmaktadır.



a) Uygulama Öncesi Graf b) Uygulama Sonrası Graf

Şekil 2.2.BFS algoritması için örnek bir graf (Grimaldi, 2003)

Şekil 2.2.' de kullanılan graf DFS algoritmasında kullanılan şekil ile aynıdır. Graf içerisindeki düğümler sıralı olarak uygulamanın başında ele alınmaktadır. Grafta başlangıç düğümü olarak düğüm 1 seçilir ve komşuları düğüm 2, 3 ve 4 kuyruğa sırası ile eklenir. Daha sonra kuyruktaki ilk düğüm olan düğüm 2 kuyruktan çıkarılır ve düğüm 2'nin komşuları kuyruğun sonuna düğüm 3'ten sonra sırayla eklenir. Düğüm 2'nin komşusu olan düğüm 4 kuyruğa önceden eklendiği için bir daha yazılmaz. Bu şekilde düğüm 2'den sonra kuyruktaki bir sonraki düğüm olan düğüm 4 kuyruktan çıkarılır ve düğüm 4 için komşuları bulunur ve kuyruğa eklenir. Süreç kuyruktaki tüm düğümler bitene kadar devam eder ve sonunda BFS algoritmasına göre *T* ağacı oluşturulmuş olur.

2.2. En Kısa Yol Algoritmaları

Bir graf üzerindeki iki noktayı birbirine bağlayan birden fazla yol olabilir. Bu yolların her birinin birer maliyet değeri de bulunmaktadır. Böyle bir yapı içerisinde bir noktadan bir başka noktaya en kısa yolu kullanarak ulaşmak önemli bir problemdir. Bu amaca ulaşmak için maliyet değeri olarak en küçük olan yolların kullanımı tercih edilir. Bu bölümde literatürde en çok adı geçen bazı en kısa yol algoritmaları açıklanmaktadır.

2.2.1. Dijkstra algoritması

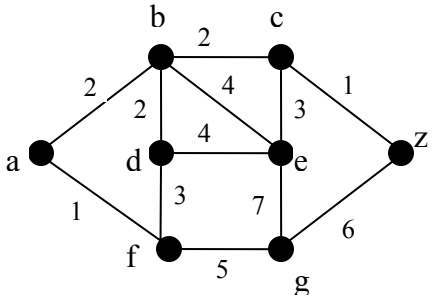
Dijkstra algoritması (Dijkstra, 1959), 1959 yılında E. W. Dijkstra tarafından bulunmuştur. Negatif bağlaç değerleri taşımayan bir graf G(V, E) için tek kaynaklı en kısa yolu hesaplayan bir algoritmadır (Barbehenn, 1998; Solka, vd., 1992).

Dijkstra algoritması, bir graf içerisindeki iki nokta arasındaki optimal yolun hesaplanmasını sağlar (Liu, vd., 1994; Garcia, vd., 2007). Burada ifade edilen optimal yol kavramı ile duruma göre ‘en kısa’, ‘en ucuz’ veya ‘en hızlı yol’ anlamları düşünülebilir. Bu durumda graf üzerinde yer alan bağlaçlar için kullanılan değerler önem taşımaktadır.

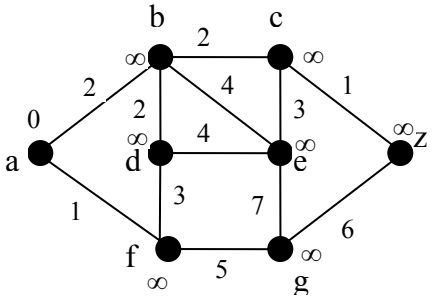
Dijkstra algoritmasında kaynak olarak bağlı, ağırlıklı ve ağırlık değerlerinin tamamı pozitif olan bir graf kullanılmaktadır. Algoritma sonuç değer olarak iki nokta arasındaki en kısa yolu vermektedir (Johnsonbaugh, 2005).

1. *Kaynak düğümü seç.*
2. *S düğümler dizisi tanımla ve boş diziye sıfırla. Algoritma süreci içerisinde en kısa yolu veren düğümler S dizisi içerisine kayıt edilecektir.*
3. *Kaynak düğümü 0 olarak etiketle ve S dizisi içerisine ekle.*
4. *Yeni eklenen düğüme bir kenarla bağlanmış olan S dizisi içerisinde olmayan her düğümü göz önünde bulundur. S içinde olmayan düğümü, yeni eklenen düğüm etiketi + kenar uzunluğuyla beraber etiketle.*
5. *S içinde olmayan düğümü, en küçük etiket değeriyle al ve S dizisine ekle.*
6. *Adım 4’e git (hedef düğüm S dizisi içerisinde olana kadar veya etiketsiz düğüm kalmayana kadar)*

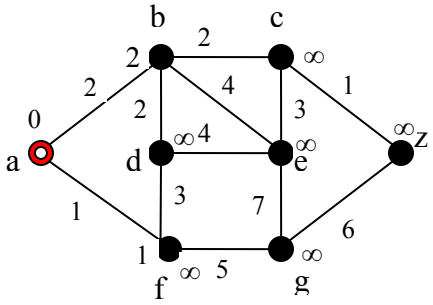
Aşağıda Şekil 2.3.’ de verilen graf üzerinde Dijkstra algoritması anlatılmaktadır.



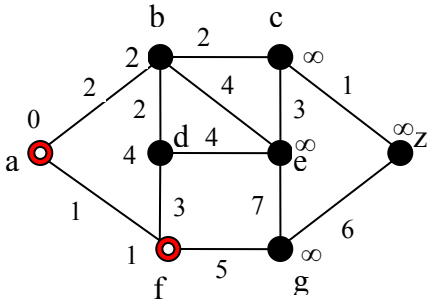
a) Dijkstra Algoritması için örnek graf yapısı



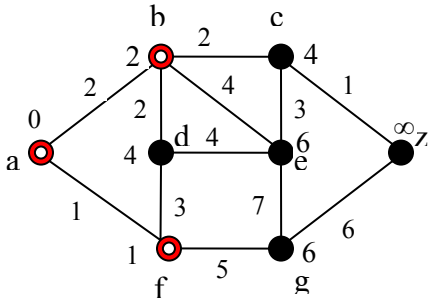
b) Dijkstra Algoritmasının Başlangıç safhası



c) Dijkstra Algoritmasında ilk iterasyon



d) Dijkstra Algoritmasında ikinci iterasyon



e) Dijkstra Algoritmasında üçüncü iterasyon

Şekil 2.3. Dijkstra algoritmasının adım adım açıklanması (Johnsonbaugh, 2005).

Şekil 2.3.' de Dijkstra algoritmasının adım adım uygulaması yapılarak a noktasından z noktasına en kısa yol bilgisinin nasıl belirlendiği gösterilmektedir. Şekil 2.3.a'da algoritmanın uygulanacağı graf yapısı gösterilmektedir. Şekil 2.3.b'de algoritmanın başlangıç safhasında tüm noktalar geçici noktalar olarak düzenlenmektedir ve ∞ sembolü ile görülmektedir. Şekil 2.3.c'de a noktası halka içerisine alınmaktadır. Bu şekilde a noktası kalıcı nokta olarak düzenlenmektedir. Daha sonra b ve f noktaları a noktasına komşu noktalar olarak düzenlenirler.

$$L(b) = \min\{\infty, 0+2\} = 2, \qquad L(f) = \min\{\infty, 0+1\} = 1$$

Bu noktalar arasında en küçük değeri veren f noktası Şekil 2.3.d'de görüldüğü gibi halka içine alınarak kalıcı hale getirilir, d ve g noktaları f noktasına komşu noktalar olarak düzenlenirler. Şekil 2.3.e' de görüldüğü gibi bir sonraki iterasyonda en küçük

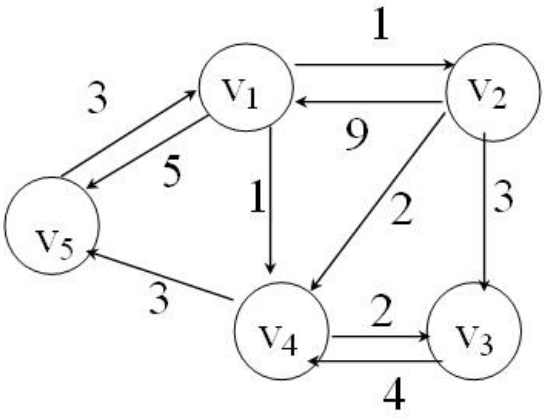
değeri veren b noktası halka içine alınır ve d noktası, c noktası ve e noktası komşuları olarak düzenlenir. Algoritma bu şekilde tekrarlı olarak devam ettirilir ve sürecin sonunda z noktası 5 değerini alarak düzenlenir. Bu değer a noktasından z noktasına en kısa yolun uzunluk değerini verir. En kısa yol bilgisi (a, b, c, z) şeklinde oluşmaktadır.

Dijkstra algoritması en iyi çözümü bulabilen bir algoritma olmasına rağmen hedef düğüme olan uzaklık değeri arttıkça etkinliği zayıflamaktadır (Noto, 2000). Buna rağmen Dijkstra algoritması uzun süredir en kısa yol problemlerinin çözümünde kullanılmaktadır (Fujita, vd., 2003).

2.2.2. Floyd algoritması

Floyd algoritması, bir graf içindeki tüm noktalar arasındaki en düşük maliyetli yolu bulmak için tasarlanmıştır (Darkridge, 2006). Algoritma, noktalar arasındaki kenar maliyetlerini gösteren bir matris üzerinde işlem yapar (Preiss, 2002; Wang, vd., 2007). Bu matrise komşuluk matrisi ismi verilmektedir.

Floyd algoritması çağrılmadan önce komşuluk matrisi oluşturulmalıdır ve bu matris genellikle iki boyutlu bir dizi içerisinde olmaktadır. Eğer graf içerisinde n adet düğüm varsa, matris (n x n) boyutunda olmaktadır. Matris içerisindeki her satır başlangıç düğümünü, matris içerisindeki her sütun ise bitiş düğümünü temsil eder. Eğer başlangıç düğümü (i) ile bitiş düğümü (j) arasında bir kenar varsa, bu kenar maliyet değeri matrisin (i, j) koordinatına yerleştirilir. Eğer iki düğüm arasında direkt kenar bağlantısı yoksa (i, j) koordinatına sonsuz değeri (∞) yerleştirilir. Bu değer, i'den j'ye direkt gidişin mümkün olmadığını ifade eder. Aşağıdaki Şekil 2.4.'de bir graf yapısı ve grafın komşuluk matrisi yer almaktadır.



Şekil 2.4. Yönlü graf , (Binghamton, 2006)

Çizelge 2.1. Komşuluk matrisi

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

Floyd algoritmasının sözde kodu aşağıdaki gibidir (Çölkesen, 2003).

```
Komşuluk matrisi üzerindeki maliyetleri başlangıç maliyeti olarak kabul et
Rota matrisine boş anlamında değerler yerleştir. (ROTA = -1)
Düğümlerin kendilerine ulaşım maliyetlerini sıfırla (UM[i][i]=0)
for (aradüğüm sayacı < düğüm sayısı) {
        for (satır sayacı < düğüm sayısı)  {
                for (sütun sayacı < düğüm sayısı )  {
                 if (ara düğüm üzerinden gitmek daha kısa ise) {
                        aradüğümlü maliyeti en küçük maliyet kabul et;
                        Rota bilgisini güncelle}
                }
        }
}
```

Sözde kod içerisinde yer alan ROTA kelimesi en kısa yolun rota bilgilerinin tutulduğu matrisi; UM kelimesi, sonucun tutulacağı uzaklık matrisini ifade etmektedir.

2.3. En Küçük Kapsayan Ağaç (Minimum Spanning Tree)

En küçük kapsayan ağaç yöntemlerinde bir graf üzerinde düğümler ve düğüm sayısı–1 tane bağlaç yer alır. Yöntem içerisinde tüm düğümler kapsanır. Genel yapısı içerisinde iki düğüm arasında tek bir yol vardır ve dolayısıyla halka bir yapıya sahip değildir.

2.3.1. Kruskal algoritması

Kruskal algoritması en küçük kapsayan ağacını belirlemek için kullanılan algoritmalardan birisidir. Algoritmanın başında graf üzerindeki düğümler, aralarında bağlantı olmayan n tane bağımsız küme gibi düşünülür. Bu bağımsız kümeler tek tek maliyet değeri en az olan kenarlarla birleştirilerek düğümler arasında bağlantı olan tek bir küme yaratılmaya çalışılır (Dai, ve Wu, 2003).

Küme birleştirilmeleri sırasında birleştirme işlemi en az maliyetli olan kenarlardan başlar. Daha sonra kalan kenarlar içerisinden en az maliyetli olanı seçilir. Aşağıda Kruskal algoritmasının sözde kodu gösterilmektedir (Çölkesen, 2004).

S dizisini oluştur. // Yol ağacını oluşturan kenarların tutulduğu dizi.
K dizisini oluştur. // Grafdaki kenarları içeren K dizisi.
Yol uzunluğuna başlangıç değerini ver.

While (K ≠ {O}) && yol uzunluğu < N)

{

K içerisinden en düşük maliyetli k_i kenarını al ve onu K' den sil.

if (k_i S ye eklendiğinde çevrim oluşturmuyorsa)

{

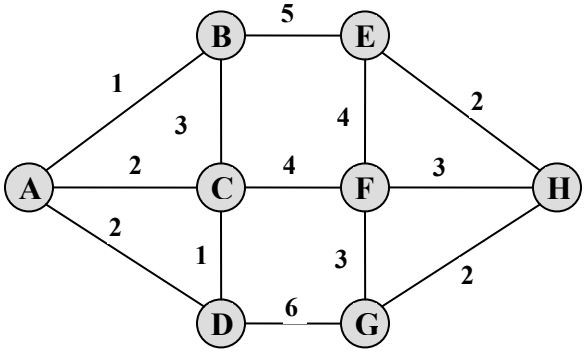
k_i' yi S' ye ekle.

Yol uzunluğu++;

}

}

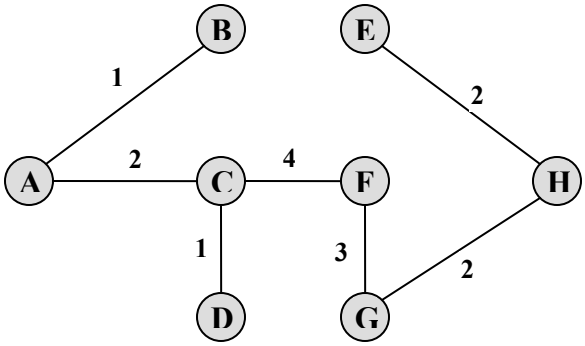
Sözde kod incelendiği zaman, S ve K dizileri oluşturulmakta ve kenarlar değerlerine göre sıralanarak K dizisi içerisine yerleştirilmektedir. Algoritmanın temeli while çevrimi üzerine oturtulmuştur. Döngü koşulu K'nın boş olmaması ve o ana kadar elde edilen yol uzunluğunun N'den küçük olmasıdır. Döngü içerisinde ilk önce K içerisinden en küçük maliyetli bağlaç alınır ve o bağlaç K dizisinden çıkarılır. Eğer alınan bağlaç, o andaki yol ağacı olan S'ye eklendiğinde bir çevrim oluşturmuyorsa S'ye eklenir, aksi durumda atlanır ve bir sonraki en az maliyetli bağlaça geçilir.



Şekil 2.5. Örnek graf yapısı

Şekil 2.5.'de algoritmanın uygulanacağı örnek bir graf görülmektedir. Uygulamanın başlangıcında düğümler arasında bir bağ bulunmadığı kabul edilmektedir. Öncelikli olarak en küçük maliyet değerine sahip olan düğüm çifti belirlenir. Bu durumda önce (A-B) ve daha sonra (C-D) düğüm çifti seçilir. Bu düğümler kendi içlerinde

birleştirilirler. Bir sonraki iterasyonda geriye kalan çiftler içerisinde en küçük maliyet değerine sahip olan düğüm çifti belirlenir. Bu durumda (A-C) çifti, (E-H) çifti ve (G-H) çiftleri birleştirilir. Bir sonraki iterasyonda (F-G) çifti ve en son iterasyonda (C-F) çifti birleştirilir. Son adım bitince dışarıda kalan düğüm olmadığı için işlem tamamlanmış olur.



Şekil 2.6. Uygulama sonucunda oluşan yol ağacı

Algoritmanın uygulanması sonucunda Şekil 2.6.'da görülen kapsayan ağacı oluşmaktadır. Oluşan en küçük kapsayan ağacının maliyet değeri ise 15 olmaktadır.

2.3.2. Prim algoritması

En küçük kapsayan ağacını oluşturabilmek için kullanılan diğer bir algoritma ise Prim algoritmasıdır. Prim algoritması veren Robert C. Prim tarafından 1957 yılında geliştirilmiştir (Prim, 1957). Ağırlık değerlerine sahip bir G grafı içerisinde en küçük yol ağacını hesaplamaktadır. Oluşturulan ağaç ile tüm kenarların ağırlıkları minimize edilmeye çalışılmaktadır. Graf bağlı bir graf olmalıdır. Eğer grafın tamamı birbirine bağlı değilse bu durumda sadece bağlı olan bileşenlerin en küçük yol ağacını bulacaktır (Eel, 2006).

Algoritma içerisinde iki küme kullanılmaktadır. Bu kümelerden biri işlem görmüş küme diğeri de işlem görmemiş kümedir. Algoritmanın başında sadece bir nokta işlem görmüş küme (P) içerisinde yer alır. Diğer tüm noktalar ise işlem görmemiş küme (N) içerisinde yer alır. Algoritmadaki her iterasyonda P kümesi bir artarken N kümesi bir azalır. İterasyon N kümesinde bir nokta olana kadar devam eder (Grimaldi,2003).

Prim algoritmasının sözde kodu aşağıdaki gibidir (Grimaldi, 2003):

Adım1: i =1 olarak düzenle ve rasgele v_1 noktasını ($v_1 \in V$) P setine ata. $N = V - \{v_1\}$ ve T=0 olarak tanımla.

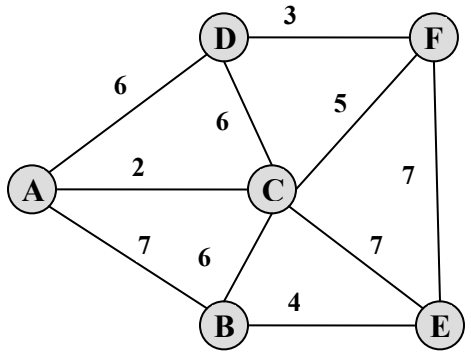
Adım2: $1 \le i \le n-1$ *için* $|V| = n$, $P=\{v_1, v_2, \ldots, v_n\}$, $T=\{e_1, e_2, \ldots, e_{i-1}\}$ *ve N= V-P. P kümesindeki x noktasıyla N kümesindeki y noktasını ($y=v_{i+1}$) birbirine bağlayan en kısa kenarı T ağacına ekle. Daha sonra y noktasını P kümesine ekle ve N kümesinden sil.*

Adım3: sayacı 1 arttır.

Eğer i=n ise G grafı için optimal ağaç oluşur.

Eğer i<n ise adım 2'ye geri dön.

Aşağıda Prim algoritmasının uygulaması adım adım gösterilmektedir.



Şekil 2.7. Prim algoritması için örnek graf.

Başlangıç safhası : P:{A}, N:{B, C, D, E, F}, T:{}

Birinci iterasyon : P:{A, C}, N:{B, D, E, F}, T:{A, C}

İkinci iterasyon : P:{A, C, F}, N:{B, D, E}, T:{{A, C}, {C, F}}

Üçüncü iterasyon : P:{A, C, F, E}, N:{B, D}, T:{{A, C}, {C, F}, {C, E}}

Dördüncü iterasyon : P:{A, C, F, E, B}, N:{D}, T:{{A, C}, {C, F}, {C, E}, {C, B}}

Beşinci iterasyon : P:{A, C, F, E, B, D}, N:{}, T:{{A, C}, {C, F}, {C, E}, {C, B}, {C, D}}

2.3.3. Sollin algoritması

En eski ve en basit en küçük kapsayan ağacı algoritması Sollin algoritması olarak bilinmektedir. Sollin algoritması 1926 yılında Boruvka tarafından ortaya konmuştur. Algoritma, 1960 yılında Sollin tarafından yeniden düzenlendiği için bu isimle anılmaktadır.

Algoritmanın uygulaması sırasında her bir düğüm bağımsız olarak düşünülmektedir. Bu düğümler uygulamanın süreçleri içerisinde birleşerek sonunda bir ağaç meydana getirmektedir.

Sollin algoritmasının sözde kodu aşağıdaki gibidir: (Çölkesen, 2003)

SK ← { Ø }; boş seçilen kenarlar (SK) dizisi oluştur.

Başlangıç anında her düğüm için En Küçük Maliyetli (EKM) kenarı seç ve sonuçta tüm düğümleri içeren altağaçları (ormanı) oluştur.

while (ormandaki ağaç sayısı > 1 VEYA kenar sayısı <N)

{

Her ağacı, genişletmek için, o ağaca herhangi bir yerden bağlı en az maliyetli kenarı al. O kenarı ilgili altağaca ekle.

Aynı kenar birden çok seçilmişse ilki dışındakileri önemseme.

}

Algoritmada her düğüm en küçük maliyet değerine sahip komşu düğümü belirler. Belirlediği bu düğümle kendisi arasında bir kenar oluşturur. Bu belirleme sürecinde aynı kenarın birden fazla seçildiği durumda, ilkinden sonraki durumlar grafın yönsüz graf olması nedeniyle önemsiz kabul edilir. Meydana gelen alt ağaçlar daha sonra kenar maliyeti değeri dikkate alınarak yeniden birleşirler. Uygulama, sonunda tek bir ağaç kalana kadar devam eder. Sollin algoritması, Prim ve Kruskal algoritmalarına göre daha az sayıda adımda sonuca ulaşabilmektedir. Bununla beraber ara işlem adımları daha fazla olabilir.

BÖLÜM 3.

YÖNLENDİRME TEMELLERİ

3.1. Giriş

Bilgisayar ağları ve telekomünikasyon sektöründe bir noktadan bir başka noktaya ulaşım ve iletişim kurmak için süreç içerisinde yönlendirmeler yapılmaktadır. Hızlı ve kısa bir zaman dilimi içerisinde son noktaya erişmek önemlidir. Bu bakımdan yönlendirme sürecinin temelleri ve ayrıntıları bu bölümde açıklanmaktadır.

3.2. Yönlendirme Bileşenleri

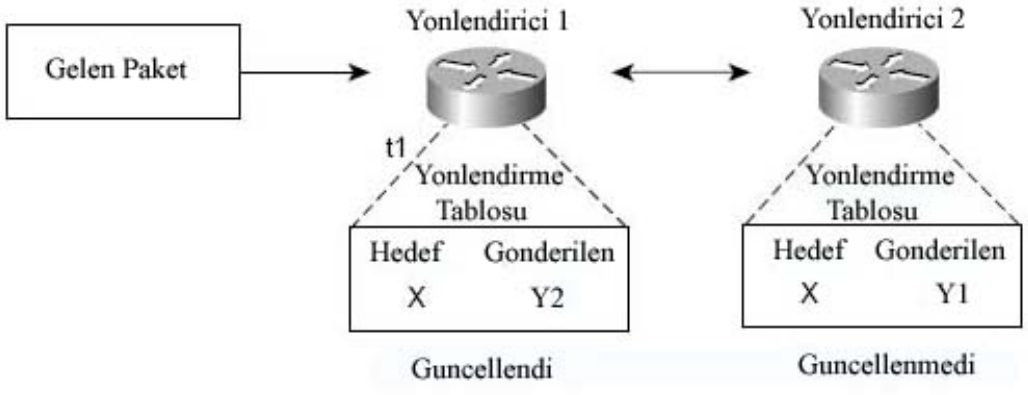
Yönlendirme süreci, iki temel aktiviteyi içerir. Bu aktiviteler en iyi yönlendirme kararını alabilme ve ağ içerisinde paketlerin iletimini gerçekleştirebilmedir. Yol belirleme süreci oldukça önemli ve karmaşık bir yapıya sahiptir.

3.2.1. Yol belirleme

Yönlendirme algoritmaları, bir paketin ağ içerisinde yolculuğu sırasında hedefe doğru giderken en iyi yolu kullanması için metrikleri kullanırlar. Metrikler birer ölçüm değerleridir. Örnek olarak bant genişliği, bekleme ve güvenilirlik verilebilir. Metrikler,

yönlendirme algoritmaları tarafından optimal yolun belirlenmesi için kullanılır. Yol belirleme sürecinde yönlendirme tabloları yönlendirme algoritmalarına yardımcı olurlar. Yönlendirme tabloları, yol bilgilerini içerir ve bu bilgiler kullanılan yönlendirme algoritmasına bağlı olarak değişiklik gösterebilir (Cisco, 2001).

Yönlendirme algoritmaları, yönlendirme tablolarını düğüm isimleri, çıkış arayüzü, hop sayıları gibi bilgilerle doldururlar. Hedef düğüm/sonraki düğüm birlikteliği, yönlendiriciye, belirli bir hedefe varacak olan optimal yol üzerindeki bir sonraki düğümü söyler. Bir yönlendirici kendisine gelen bir paketi aldığı zaman, hedef adresi kontrol eder ve sonraki düğümle bu adresi ilişkilendirir. Şekil 3.1.'de örnek bir hedef düğüm/sonraki düğüm ilişkisi gösterilmektedir.



Şekil 3.1. Örnek bir hedef düğüm/sonraki düğüm ilişkisi (Cisco, 2001)

Yönlendirme tabloları bir yolun tercih edilebilirliği hakkında da veriler içermektedir. Bu bilgiler doğrultusunda yolun tercih edilip edilmeyeceğine karar verilir. Yönlendiriciler metrikleri karşılaştırarak optimal yol kararını alırlar ve metrikler kullanılan yönlendirme algoritmasına bağlı olarak farklılık gösterebilirler.

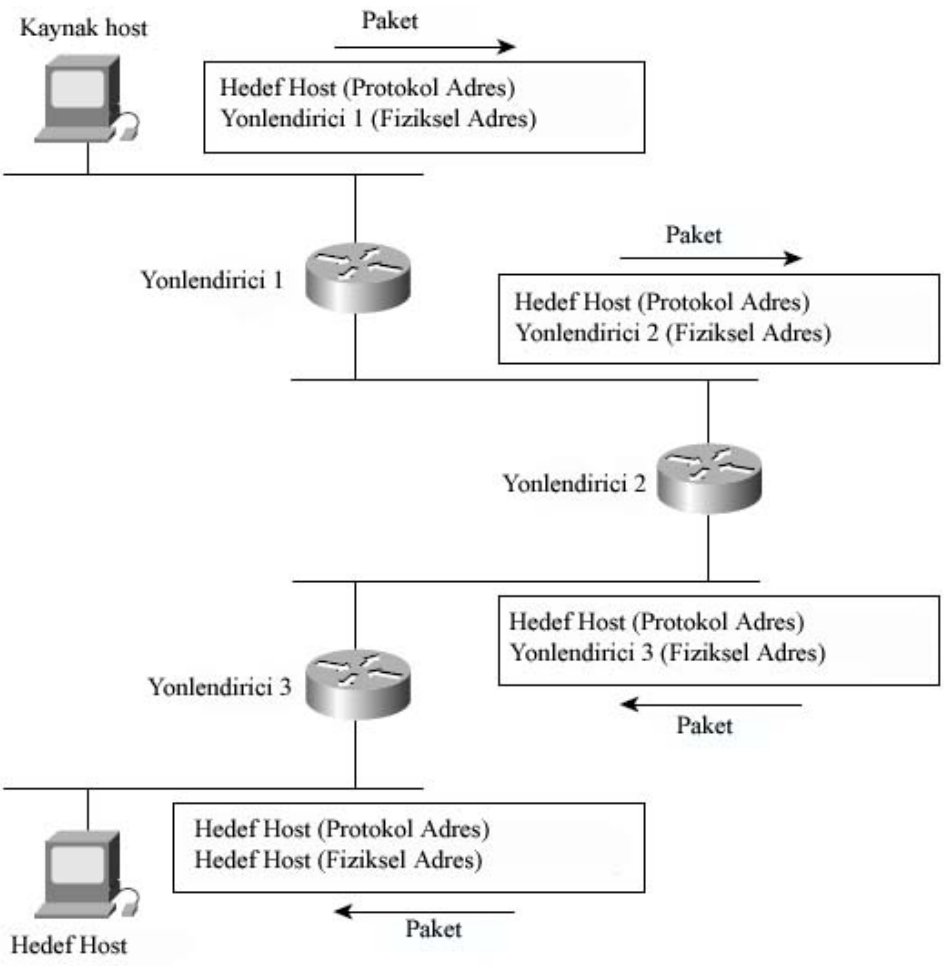
Yönlendiriciler diğer yönlendiricilerle iletişim halindedirler ve mesaj iletimine bağlı olarak kendi yönlendirme tablolarını düzenlerler. Yönlendirme güncelleme mesajı (*routing update message*) bu tip bir mesajdır ve genellikle yönlendirme tablosunun tamamını ya da bir kısmını içerir. Diğer tüm yönlendiricilerden gelen yönlendirme güncellemelerini analiz ederek, ağ topolojisinin detaylı resmi oluşturulabilir.

3.2.2. Anahtarlama

Anahtarlama, giriş arayüzünden alınan paketlerin uygun çıkış arayüzüne doğru aktarılmasına denir. Anahtarlama süreci genel olarak basit bir süreçtir. Çoğu durumda bir bilgisayar (*host*), diğer bir bilgisayara paket göndermeye karar verirse, yönlendiricinin adresine ihtiyaç duyar. Kaynak bilgisayar bir paketi yönlendiricinin fiziksel adresine (*Media Access Control-MAC*) gönderir, fakat paketin protokol adresi değişmez.

Yönlendirici, paketin hedef adresini aldığı zaman, paketin iletileceği sonraki düğümü bilip bilmediğine bakar. Eğer yönlendirici paketi nasıl ileteceğini bilmiyorsa, paketi siler. Eğer nasıl ileteceğini biliyorsa bu durumda paketin hedef fiziksel adresini sonraki düğümün fiziksel adresi olarak değiştirir ve paketi iletir.

Sonraki düğüm, son hedef düğüm olabilir. Eğer değilse bu durumda sonraki düğüm bir yönlendiricidir ve bu yönlendirici aynı anahtarlama sürecini devam ettirir. Paket ağ içerisinde hareket ederken fiziksel adresi sürekli değişir ama paketin protokol adresi daima sabit kalır. Aşağıda Şekil 3.2‘de anahtarlama süreci gösterilmektedir (Cisco, 2001).



Şekil 3.2. Anahtarlama süreci

3.3.3. Yönlendirme metrikleri

Yönlendirme algoritmaları, en iyi yolu belirleyebilmek için birçok farklı metrik değeri kullanmaktadır. Karmaşık yönlendirme algoritmaları, çoklu metrik değerlerini kullanır. Çoklu metrikleri birleştirerek tek bir metrik değeri gibi görmektedir. Aşağıda ağ

içerisinde en uygun yolun belirlenmesi için kullanılan metrikler yer almaktadır (Cisco, 2001).

- Yol uzunluğu (*path length*)
- Güvenilirlik (*reliability*)
- Gecikme (*delay*)
- Band genişliği (*bandwith*)
- Yük durumu (*load*)
- İletişim maliyeti (*communication cost*)

Yol Uzunluğu: En genel yönlendirme metriğidir. Bazı yönlendirme protokolleri ağ yöneticilerine her ağ hattı için rasgele maliyet değerleri atamasını sağlar. Bu durumda yol uzunluğu, geçilen her hattın maliyetlerinin toplamıdır. Diğer bazı yönlendirme protokolleri ise düğüm sayısını kullanır. Düğüm sayısı (*hop count*), ağ içerisinde geçilen düğümlerin sayısını verir. Düğüm ifadesi yönlendiricileri tanımlamaktadır ve bir paket kaynaktan hedefe doğru giderken yönlendiricilerin üzerinden geçer.

Güvenilirlik: Yönlendirme algoritmalarında, her ağ hattının bit hata oranlarına bakılabilir. Bazı ağ hatları diğerlerinden daha fazla veya daha sık çökebilir. Aynı şekilde bir ağ çöktükten sonra bazı ağ hatları daha kolay ya da daha çabuk tamir edilebilir. Bu nedenle güvenilirlik faktörleri güvenilirlik oranlarının değerlendirilmesinde dikkate alınır.

Gecikme: Gecikme, bir paketin kaynaktan hedefe doğru ağ içinden geçerken, yolculuğu sırasında gerekli olan zaman uzunluğudur. Gecikme, pek çok faktöre bağlıdır. Bunları hattın bant genişliği, yol üzerindeki yönlendiricilerin portları üzerindeki kuyruk, ağ hatlarındaki tıkanıklıklar ve fiziksel uzunluklar olarak söyleyebiliriz. Gecikme kavramı içerisinde pek çok önemli değişkenin birlikteliği söz konusu olduğu için genel ve yararlı bir metriktir.

Bant Genişliği: Bant genişliği, bir hattın kullanılabilir trafik kapasitesini göstermektedir. Bant genişliği ayrıca hat üzerindeki maksimum veri miktarı oranıdır. Büyük bant genişliğine sahip hatlar üzerindeki yollar yavaş dar bant genişliğine sahip hatlar üzerindeki yollardan daha iyidir. Hızlı bir hattın durumu yoğunsa ve hat meşgul durumdaysa hedefe gönderilen paketin ihtiyacı olan zaman da artacaktır.

Yük Durumu: Yük durumu, yönlendirici gibi bir ağ kaynağının meşguliyet derecesini göstermektedir. Yük durumu, çeşitli yollarla hesaplanabilir.

İletişim Maliyeti: İletişim maliyeti diğer bir önemli metriktir. Özellikle bazı firmalar performans yerine işletim masraflarına bakmaktadırlar. Hat gecikmeleri daha uzun sürmesine rağmen, kendi hatları üzerinden paket transferini sürdürmeyi düşünmektedirler (Cisco, 2001).

3.3. Yönlendirme Protokolleri (Routing Protocols)

Yönlendirme protokolleri, ağdaki düğümler arasında yönlendirme bilgilerinin değişimini sağlarlar. Bir ağ içindeki tüm düğümler kendilerine ait yönlendirme tablolarına sahiptirler ve bu yönlendirme tablolarını dinamik olarak düzenlerler. Bu düzenleme veya güncelleme işleminde yönlendirme protokollerini kullanırlar.

3.3.1. Statik yönlendirme

Statik yönlendirme protokollerinde, kullanılacak olan yönlendirme tabloları belli bir algoritmaya bağlı olarak önceden oluşturulmaktadır. Oluşturulan bu tablolar bir daha değiştirilmemektedir. Bu durumda, bir düğümden diğer bir düğüme giden yol önceden bellidir ve ağda oluşacak bir trafik ya da değişimlerden etkilenmemektedir.

Statik yönlendirme oldukça basittir ve güncelleme gerektirmez. Bununla beraber ağdaki tıkanıklık durumlarında alternatif ve daha iyi olan yolları seçmez ve önceden tanımlı yolu kullanmaya çalışır. Ağda hatlarda sorun olduğu zaman ya da hat koptuğu

zaman önceden tanımlı bu yollar değişemediği için düğümle olan bağlantısı kurulmayabilir.

3.3.2. Dinamik yönlendirme

Dinamik yönlendirme içerisinde, yönlendirme tabloları zaman içerisinde, değişimlere bağlı olarak yeniden düzenlenmektedir. Yönlendirme tabloları, ağdaki tıkanıklık durumları, düğümde oluşan yoğunluk, ağa yeni bir düğümün eklenmesi ya da hatlardan bazılarının kopması sonucunda oluşan değişimlerde güncellenir.

Dinamik yönlendirme statik yönlendirmeye göre daha karmaşıktır ve zaman içinde kendini güncellemektedir. Ağ içinde trafik yoğunluğunun artması veya hat kopması sonucu oluşan sorunlarda yönlendirme tabloları dinamik olarak güncellenir ve alternatif yollar oluşturulur.

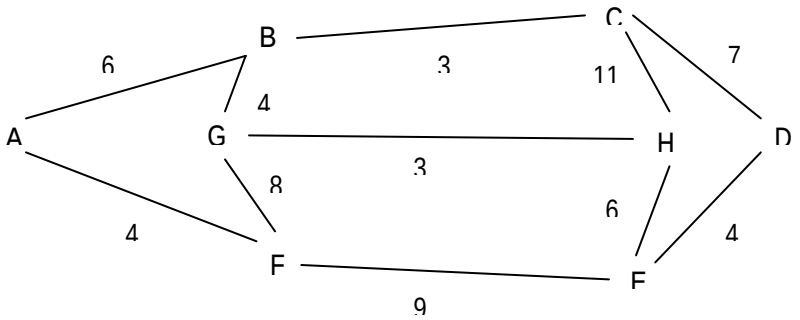
3.3.2.1. Uzaklık vektörü protokolleri

Uzaklık vektörü algoritması en popüler dinamik yönlendirme algoritmalarından birisidir. Bu algoritmanın çalıştığı sistem içerisinde her yönlendiricinin içinde barındırdığı yönlendirme tabloları bulunmaktadır. Algoritma her hedef düğüm için en iyi uzaklığı bulabilmek amacıyla bu tablolar üzerinde işlem yapar.

Uzaklık vektörü algoritmasında yolu hesaplamak için her zaman uzaklık ölçüsü kullanılmayabilir. Kullanılacak ölçü birimi olarak düğümden düğüme atlama sayısı, bekleme süresi, yol boyunca kuyruktaki toplam paket sayısı gibi ölçütler kullanılabilir (Tannenbaum, 2003).

Uzaklık vektörü algoritmasında, ağ içerisinde yer alan her düğüm (yönlendirici) belirli bir periyotla kendi yönlendirme tablosunu komşularına gönderir ve aynı şekilde komşu düğümlerin yönlendirme tablolarını da alır. Düğümler aldıkları yönlendirme tablolarındaki bilgileri kullanarak kendi tablolarını yeniden düzenlerler. Örneğin, bu bilgileri kullanarak kendilerine direkt olarak bağlı olmayan düğümler ile aralarındaki uzaklık ölçümünü veya gecikme sürelerini hesaplayabilirler veya kendilerine direkt bağlı olan düğümle aralarındaki gecikme süresinde oluşan değişimleri güncelleyebilirler.

Aşağıdaki Şekil 3.3.'de örnek bir ağ yapısı gösterilmektedir. Bu ağ yapısı kısa bir zaman önce çalışmaya başladığı düşünülürse bu durumda düğümlerin sahip olduğu yönlendirme tablolarında yalnızca komşu düğümlerin bilgisi yer alacaktır.



Şekil 3.3. Örnek ağ yapısı ve maliyet bilgileri

Sistem aktif hale geldiği ilk anda örnek olarak verilen B, F ve H düğümlerinin yönlendirme tablo bilgileri Çizelge 3.1.'deki gibi olacaktır ve bu düğümler tablo bilgilerini kendilerine komşu olan düğümlere göndereceklerdir.

Çizelge 3.1. Ağdaki örnek düğümlerin yönlendirme tablolarının başlangıç durumu

B Düğümü		
	Gecikme	**Bir sonraki düğüm**
A	6	A
B	0	-
C	3	C
D	∞	-
E	∞	-
F	∞	-
G	4	G
H	∞	-

F Düğümü		
	Gecikme	**Bir sonraki düğüm**
A	4	A
B	∞	-
C	∞	-
D	∞	-
E	8	E
F	0	-
G	7	G
H	∞	-

H Düğümü		
	Gecikme	**Bir sonraki düğüm**
A	∞	-
B	∞	-
C	11	C
D	∞	-
E	6	E
F	∞	-
G	3	G
H	0	-

Burada örnek verilen bu 3 düğüm için ortak olan komşu düğüm G düğümüdür. G düğümüne gönderilen bu bilgiler ile düğüm kendi yönlendirme tablosunu güncelleyecektir. G düğümünün güncellemeden önceki ve güncellemeden sonraki yönlendirme tablosunun durumu aşağıda Çizelge 3.2.'de gösterilmektedir.

Çizelge 3.2. G düğümünün güncellemeden sonraki durumu

G Düğümü

Güncellemeden Önce		
	Gecikme	**Bir sonraki düğüm**
A	∞	A
B	4	B
C	∞	-
D	∞	-
E	∞	-
F	8	F
G	0	-
H	3	H

Güncellemeden Sonra		
	Gecikme	**Bir sonraki düğüm**
A	4+6	B
B	4	B
C	4+3	B
D	-	-
E	3+5	H
F	8	F
G	0	-
H	3	H

Yukarıdaki Çizelge 3.2.'ye bakıldığı zaman G düğümünden A düğümüne gidilmek istenirse, F düğümü üzerinden (8+4) ya da B düğümü üzerinden (4+6) gidilebilir. Bu durumda B yolu daha kısa olduğu için bu yol tercih edilir.

Uzaklık vektörü algoritmasının bazı avantajları ve dezavantajları vardır. Algoritmanın en temel avantajı işletiminin kolay olmasıdır. Algoritmanın anlaşılması ve düğümler üzerinde kurulumu oldukça kolaydır. Bununla beraber bu yapı içerisinde yer alan yönlendirme tabloları oldukça büyük hacimlerde olmaktadır. Bunun sebebi ise büyük bir ağda yer alan düğüm sayısının ve bağlantı sayısının fazla olmasıdır. Bu durumda büyük tabloların ağda düğümler arasında gönderilmesi ağ üzerinde belirli oranda trafiğin oluşmasına sebep olacaktır.

3.3.2.1.1. Yönlendirme bilgi protokolü (RIP-Routing Information Protocol)

Yönlendirme bilgi protokolü (RIP), IP veya IPX ağlarda dinamik yönlendiriciler arasında yönlendirme bilgilerinin değişimi için kullanılır. RIP protokolü 1980 yılında Xerox Networks Systems (XNS) ile kullanılması için tasarlanmış olmakla birlikte günümüzde özellikle küçük ve orta ölçekli TCP/IP ağlarında kullanılmaktadır. RIP ilk olarak Berkeley BSD Unix platformunun 4.2 versiyonuna eklenmesiyle popüler olmaya başlamıştır.

RIP algoritmasının kullanıldığı yönlendiricilerde yönlendirme tabloları hedef düğümlere ulaşım sırasında geçilen düğüm sayısı temelinde hesaplanır. RIP yönlendiriciler yük durumu, bant genişliği, gecikme gibi ölçütleri kullanmazlar. TCP/IP ağındaki RIP kullanan yönlendiriciler kendi yönlendirme tablolarını her 30 saniyede bir UDP üzerinden yayınlarlar.

Bir RIP algoritmasının kullanıldığı yönlendirici ilk olarak çalışmaya başladığı zaman, genel bir RIP istek mesajı yayınlar ve komşu RIP yönlendiriciler kendi yönlendirme tablolarını gönderirler. Komşu yönlendiricilerden gelen bilgiler ise yönlendiricinin kendi yönlendirme tablosunu oluşturmasını sağlarlar.

RIP, maksimum 15 düğümü destekler. Eğer gönderilen paket 15 düğümü geçerse geriye ulaşılamaz bilgisi döndürülür. Bu yüzden RIP uygulaması daha çok küçük veya orta ölçekli ağlarda kullanılmaktadır.

RIP protokolünün en önemli dezavantajı, RIP kullanan yönlendiricilerin yönlendirme tabloları büyük olduğundan dolayı bu bilgilerin değişimi sırasında ağda bir yük oluşmasıdır. Bu yüzden RIP büyük ağlar için uygun değildir.

3.3.2.1.2. IGRP (Interior Gateway Routing Protocol) protokolü

IGRP protokolü Cisco Sistem (Cisco System) tarafından geliştirilen bir uzaklık vektörü protokolüdür. Protokol, yönlendiricilerin birbirleri arasında yönlendirme bilgilerinin değişimi için kullanılmaktadır. IGRP protokolünün temel amacı, otonom sistemler (Autonomous Systems-AS) arasında yönlendirme yapabilmektir (Charter, 1999).

1980'li yılların ortalarında, RIP protokolü küçük ve orta ölçekli ağlarda yönlendirme işlemleri için oldukça popüler ve kullanışlı olmuştur. Fakat, RIP protokolü maksimum düğüm sayısını 16 ile sınırlandırmaktadır ve tek bir ölçü olarak düğüm sayısını kullanmaktadır. Bu sınırlılık koşulları altında Cisco firması IGRP protokolünü geliştirmiştir. IGRP protokolü, bant genişliği, gecikme, güvenilirlik ve yük durumu ölçülerinden meydana gelen karma bir ölçü kullanmaktadır. Bunun anlamı ise tek bir ölçü birimi kullanmak yerine karma bir ölçü kullandığıdır. IGRP protokolü, 255 düğüm sayısına kadar yayılımı desteklemektedir (Black, 2000).

IGRP protokolünün kullandığı karma ölçüdeki yapılar aşağıdaki gibidir:

1. *Gecikme:* Kaynaktan hedefe bir paketi iletmek için gerekli olan zaman uzunluğudur.
2. *Bant Genişliği:* Hattın veri kapasitesidir. Kapasite 1200 bps'den 10 Gbps'e kadar olabilmektedir.

3. *Güvenilirlik:* Hattın hatalı bit oranıdır. 1–255 arasında bir değer alır.
4. *Yük Durumu:* Yönlendirici ya da hat gibi ağ kaynakları üzerindeki aktivite miktarıdır.

3.3.2.2. Hat durum protokolleri (Link State Protocols)

Hat durum protokolleri dinamik protokollerdir. Ağ topolojileri üzerinde oluşan değişimlere göre adapte olabilmekte ve yeni durumda oluşan alternatif yolları bulabilmektedir. Hat durum yönlendirmesi, en kısa yol algoritması olan Dijkstra algoritmasının uygulaması olarak düşünülebilir. Ağ içerisinde yer alan düğümler bağlantı durumlarını gösteren paketleri göndermektedirler. Bu paketler taşkın (*flooding*) yöntemiyle gönderilmektedir. Bu paketleri alan diğer düğümler kendi ağ topolojilerini oluştururlar ve en kısa yol algoritmalarını çalıştırarak en kısa yol hesaplamasını yaparlar. (Aktuğ, 2007) Taşkın yöntemi kullanılırken gereksiz trafikten kurtulmak için paketler üzerine sayaç konmaktadır. Burada tanımlı sayıya ulaşıldığı zaman paket kendini yok etmektedir.

3.3.2.2.1. OSPF protokolü

OSPF protokolünün temeli olan En kısa Yol İlk (*Shortest Path First*-SPF) algoritması 1978 yılında ARPANET için Bolt, Beranak ve Newman tarafından geliştirildi (Cisco, 2001). OSPF protokolü kısa zaman içerisinde endüstri standardı protokolü olarak kabul gördü. Kısa zaman içerisinde yaygın olarak kabul görmesinin sebebi ise geniş ağların gereksinimlerini karşılayabilmesidir. OSPF, topoloji değişimlerine hızlı cevap verebilmektedir ve OSPF yönlendiriciler zorunlu durumlarda veri trafiğini yeniden

yönlendirebilmektedir. OSPF, paket yük trafiğini azaltmaktadır. Bir değişim algılandığı zaman sadece değişim bilgileri gönderilmekte, tüm yönlendirme tablo bilgileri gönderilmemektedir.

OSPF, bir hat durumu yönlendirme protokolüdür. Protokol içerisinde bir topoloji veritabanı bulunmaktadır ve bu veritabanı otonom ağların hat durumlarıyla ilgili bilgileri barındırır. Bu bilgiler en kısa yolun hesaplanması için kullanılmaktadır.

OSPF protokolü içerisinde yer alan paket başlığı 24 byte uzunluğundadır ve paket başlığı yapısı aşağıdaki Şekil 3.4'de gösterilmektedir.

Versiyon Numarası	**Tipi**	**Paket Uzunluğu**	**Yönlendirici ID**	**Alan ID**	**Check-Sum**	**Yetkilendirme Tipi**	**Yetkilendirme**	**Veri**

Şekil 3.4. OSPF paket başlığı yapısı

Versiyon Numarası: OSPF uygulamasının tanımlayıcısı

Tipi: OSPF paketin tipini belirler

1. Hello: Periyodik olarak komşuluk ilişkilerini kurmak ve korumak için gönderilir.
2. Veritabanı tanımı: Yönlendiricinin topoloji veritabanının içeriğini tanımlar.
3. Hat durumu isteği: Komşunun topoloji veritabanının isteğini tanımlar.
4. Hat durumu güncelleme: Hat durumu isteği paketine yanıtı ifade eder.
5. Hat durumu onay: Hat durumu güncelleme onay paketi.

Paket Uzunluğu: OSPF paketinin uzunluğu

Yönlendirici ID: Paket kaynağını tanımlar

Alan ID: Paketin ait olduğu ağ alanını tanımlar

Checksum: İletim hata tespitini sağlar

Yetkilendirme Tipi: Yetkilendirme tipini içerir

Yetkilendirme: Yetkilendirme bilgisi içerir

OSPF protokolünde, bir yönlendirici (*router*) kendi varlığını mümkün komşularının her birine "Hello" mesajı göndererek duyurmaktadır. Periyodik olarak, her komşu düğüm kendi durumunu göstermek için bir "Hello" mesajı gönderir. Bu yüzden, yeni yönlendiriciler kendi komşularını belirli bir zaman geçtikten sonra öğrenmektedir.

OSPF protokolünde RIP protokolünde olduğu gibi atlanan düğüm (hop) sayısı limiti yoktur. RIP protokolünde 15 düğüm sayısı sınırı vardır. Aynı şekilde OSPF yük dengeleme sorununu daha iyi çözer. OSPF, RIP protokolüne göre daha uyumludur çünkü yönlendirme değişimleri hızlı bir şekilde yayılır yani periyodik bir bilgi yayılımı yoktur.

3.3.2.2.2. IS-IS protokolü

IS-IS (*Intermediate System to Intermediate System*) protokolü OSPF protokolü ile benzer özelliklere sahip bir hat durum protokolüdür. DECNET ağında kullanılmak üzere DEC (*Digital Equipment Corporation*) firması tarafından geliştirilmiştir (Data network, 2007).

IS-IS protokolü aşağıda yazılı özellikler yönünden OSPF protokolü ile benzerlikler gösterir.

- Bir hat durum veri tabanı oluşturur.
- SPF ağacı oluşturmak için Dijkstra algoritmasını kullanır.
- *Hello* paketleri kullanarak komşulukları belirler.
- Yetkilendirme (*Authentication*) özelliği kullanılır.

IS-IS Protokolü Paket Formatı:

IS-IS protokolü, üç paket formatına sahiptir. Bunlar hello paketi, hat durum paketi (*Link State Packet-LSP*), sıra numarası paketidir. (*Sequence-Number Packet-SNP*) IS-IS paketleri 8 alan içerir. Aşağıdaki Şekil 3.5'de bu alanlar ve açıklamaları gösterilmektedir.

Şekil 3.5. IS-IS protokolü paket yapısı

Protokol Tanımlayıcısı: IS-IS protokolünü tanımlar.

Başlık Uzunluğu: Sabit başlık uzunluğunu içerir. 8 byte uzunluğundadır.

Versiyon: IS-IS protokolü için varsayılan 1 değerini içerir.

ID Uzunluğu: Kaynak ID alanının uzunluğunu belirtir.

Paket Tipi: IS-IS paketinin tipini gösterir.

Versiyon: Paket tipi alanından sonra tekrar eder.

Reserve: Kullanılmıyor.

Maksimum Alan Adresleri: 1 – 254 arasında bir değerdir. Bu alan içinde izin verilen adres numaraları yer alır.

Her IS-IS paket tipi içindeki tüm paketlerin başlık bilgileri aynıdır. IS-IS protokolü daha sonraları bütünleşmiş IS-IS (*Integrated IS-IS*) olarak yeni versiyonu geliştirilmiştir (Data network, 2007).

3.4. Yönlenen Protokoller (Routed Protocols)

Yönlenen protokoller, verilerin ağdaki taşınmalarıyla ilgilidirler. Yönlenen protokoller veriyi alarak bir noktadan bir noktaya iletilmesini sağlarlar. Yönlendiriciler yönlenen protokollerin yönlendirilmesini sağlarlar. Yönlenen protokollerin en genel kullanılanı internet ağının çekirdeğini oluşturan internet protokolüdür.

3.4.1. Internet protokolü

İnternet protokolü, OSI'nin 3. katmanında çalışan bir protokoldür. Adres bilgisi ile paketlerin yönlenmesini sağlayan birtakım kontrol bilgilerini içerir. IP protokolü RFC 791 standardı içerisinde tanımlanmaktadır. IP protokolü, internet protokolleri grubu içerisindeki birincil ağ katmanı protokolüdür. İletişim kontrol protokolüyle (TCP) beraber internet protokollerinin çekirdeğini oluşturur. IP protokolünün öncelikli iki sorumluluğu vardır. Bunlar bağlantısız olarak ağ içerisinde datagramların en iyi gönderimini sağlamak, farklı maksimum iletim birimi (MTU) boyutlarında veri hatlarını desteklemek için datagramları parçalamak ve yeniden birleşimini sağlamaktır.

Bir IP paketi pek çok bilgi tipini içerir. Aşağıdaki Şekil 3.6.'de IP paket formatı gösterilmektedir.

Şekil 3.6. IP protokolü paket formatı

*Versiyon*u	Geçerli kullanılan IP'nin versiyonunu tanımlar.
IP Başlık Uzunluğu	Datagramın başlık uzunluğunu tanımlar.
Servis Tipi	Geçerli datagramın üst seviye protokoller tarafından nasıl alacağı ve datagrama önem seviyesi atar.
Toplam Uzunluk	Veri ve başlık kısımlarının da dahil olduğu tüm IP paketinin uzunluğunu belirtir.
Kimlik	Geçerli datagramı tanımlayan bir tamsayı içerir. Bu alan parçalar halinde olan datagramları bir araya getirmek için kullanılır.
Bayraklar	3 bitlik bir alan içerir. Düşük bit paketin parçalanma durumunu belirtir. Orta bit, paketin parçalanmış paketler serisindeki son parça olup olmadığını tanımlar. Son bit yüksek seviye bitidir ve kullanılmamaktadır.
Parça Ofset	Parçanın orijinal datagramdaki pozisyonunu belirtir. Hedef IP için yeniden orijinal verinin oluşumuna olanak sağlar.
Yaşam Süresi	Sıfıra doğru azalan bir sayaçtır. Sıfır olduğu zaman paket silinir. Paketin sonsuz döngüye girmesini engeller.
Protokol	Hangi üst seviye protokolün gelen paketleri alacağını belirtir.
Başlık Sağlama Toplam	IP başlık bütünlüğünü sağlar.
Kaynak Adres	Gönderici düğümü tanımlar.
Hedef Adres	Alıcı düğümü tanımlar.
Seçenekler	Güvenlik gibi çeşitli seçenekleri IP'ye sağlar.
Veri	Üst seviye bilgiyi içerir.

3.4.2. IPX protokolü

IPX protokolü, OSI standardı içerisinde 3. katman protokolüdür. Paket gönderiminde garanti ya da başarılı vardığına dair bir bilgi vermez. IPX protokolü, verinin yerine başarılı bir şekilde ulaşıp ulaşmadığı bilgisine ihtiyaç olunmadığı

uygulamalarda kullanılır. IPX protokolünün düşük maliyete sahip olması dolayısıyla hız ve performans oranı yüksektir.

IPX adresleme şeması 3 adres bileşenine sahiptir. Bunlar aşağıdaki çizelgede kısaca tanımlanmıştır.

Çizelge 3.3. IPX bileşenleri

Adres	**Byte Uzunluğu**	**Açıklama**
Ağ	4 byte	IPX içindeki belirli bir ağı tanımlar.
Düğüm	6 bye	Ağdaki düğüm ya da bilgisayarı tanımlar.
Soket	2 byte	Bir düğümdeki süreç ya da fonksiyon işletimini tanımlar.

IPX protokolü, Novell firmasının istemci/sunucu yazılımı olan Netware için uyarlanmıştır ve geniş ölçekli ağlarda kullanılmaktadır.

BÖLÜM 4.

KARINCA KOLONİSİ SİSTEMİ

4.1. Giriş

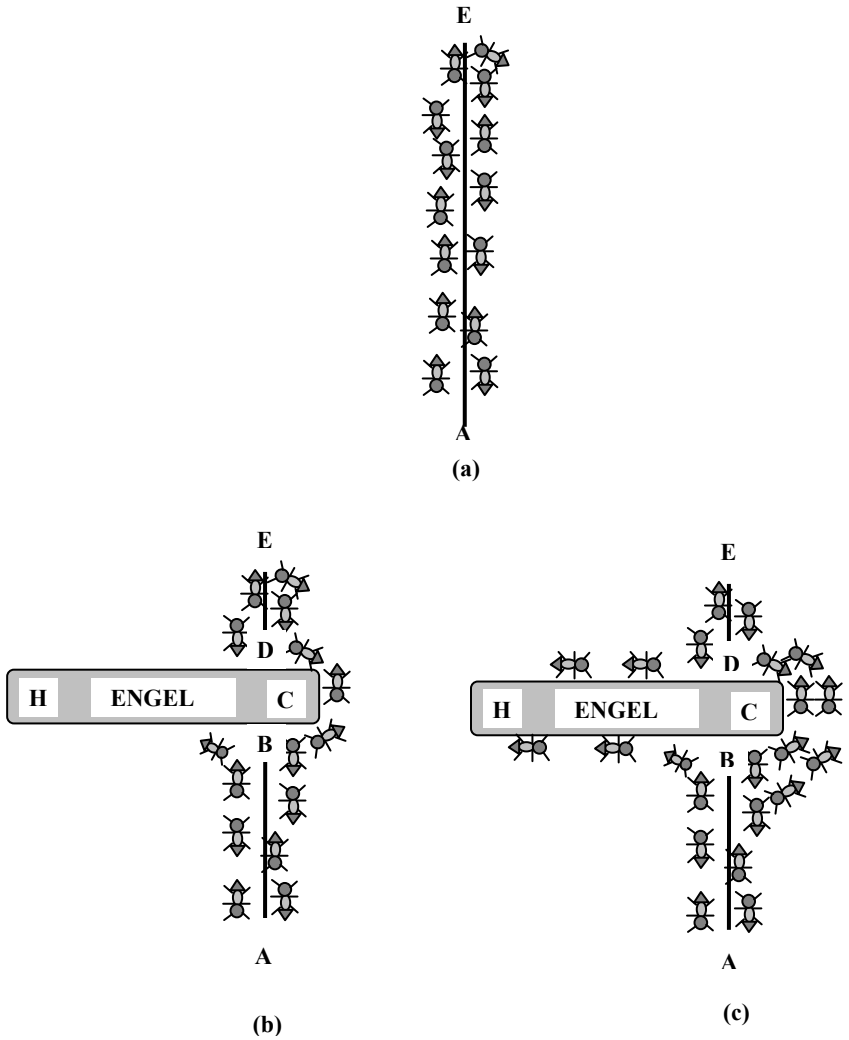
Gelişen ve büyüyen internet ve telekomünikasyon sektöründeki önemli konuların başında yönlendirme ve en uygun yolun belirlenmesi çalışmaları gelmektedir. Bu konuların gelişmesi sürecinde pek çok çalışma yapılmıştır ve yapılmaya da devam etmektedir. Gerçek karıncalarla ilgili ilk deneyler Goss ve arkadaşları (Goss, vd., 1989) tarafından 1989 yılında laboratuar ortamında yetiştirilmiş karınca kolonileri üzerinde yapılmıştır. Çalışmalardan bazıları doğal hayatta bulunan canlıların izlenmesi ve onlardan esinlenerek yeni yöntemlerin ve algoritmaların bulunmasıdır. Bu bölümde karıncaların davranışlarının izlenmesi ve analizi çalışmaları sonucunda geliştirilen *Karınca Kolonisi Optimizasyonu'*ndan bahsedilecektir. Karınca kolonilerinin davranışları, bu davranışların birer matematik modele dönüşümü ve bu algoritmanın ağ ortamındaki uygulamaları anlatılacaktır.

4.2. Doğal Hayatta Karıncaların Davranışları

Karıncalar, her biri tek bir birey olarak düşünüldüğü zaman karmaşık olmayan basit canlılardır. Bununla beraber bir topluluk (koloni) olarak ele alındıkları zaman karmaşık problemleri çözebilecek yeteneğe sahip olmaktadırlar. Bir karınca kolonisi kolektif olarak yuvalarını oluşturmak, yiyecek kaynağını aramak gibi görevleri yerine getirebilir (Bonabeau, vd., 2000; Dorigo, vd., 1999). Karıncalar yuvalarından yiyecek kaynağına doğru hareket ederler ve iki nokta arasındaki en kısa yolu bulabilirler. Bu sürecin dikkat çekici noktası ise bulunan bu kısa yol bilgisinin diğer karıncalarla da bir

iletim formu (stigmergy) sayesinde paylaşılmasıdır (Dorigo, vd., 1999; Ducatelle, vd., 2005; Baran ve Sosa, 2000). İletim formu (*stigmergy*), karıncalar tarafından kendi problem çözme aktivitelerini koordine etmek için kullandıkları indirekt iletim formudur. Karıncalar, geçtikleri yerlere feromen (Deneubourg, vd., 1990) adı verilen kimyasal bir maddeyi bulaştırarak iletimi sağlarlar (Sim ve Sun, 2003). Bu şekilde ortamda yer alan diğer karıncalar da değişimleri algılarlar.

Doğal hayatta yer alan karıncaların yuvaları ile yiyecek kaynağı arasında en kısa yolu bulabilmesini sağlayan davranışları aşağıdaki Şekil 4.1.'de gösterilmektedir (Dorigo, vd., 1996).



Şekil 4.1.(a) Karıncaların davranışları A – E arasındaki yolu izlemektedirler. (b) Yol üzerinde engel konulmuştur ve karıncalar rasgele yollarını belirlerler. (c) Kısa olan yol üzerindeki karınca sayısı ve feromen miktarı artar. (Dorigo, vd., 1996)

Şekil 4.1.(a)'ya bakıldığı zaman karıncaların düz bir yolda hareket ettikleri görülmektedir. A noktası karıncaların yuvası, E noktası yiyecek kaynağıdır. Bu düz yolda karıncalar hareket etmektedirler. Şekil 4.1.(b)'de ise bu düz yolda karşılaşılan bir engel görülmektedir. Bu durumda karıncaların ilerledikleri yol üzerinde karşılaştıkları engel nedeniyle yol kesilir. Karıncalar bu B noktasına geldikleri zaman sağ ya da sol yöne doğru hareket etmek için karar vermek zorundadırlar. B noktasına gelen ilk karıncanın bu iki yönden birisine gitmek ya da gitmemek için özel bir nedeni olmadığından eşit karar olasılıklarına sahiptir. Sonuçta BCD yolunu seçen ilk karınca, BHD yolunu seçen karıncadan önce D noktasına varacaktır. Şekil 4.1.(c)'ye bakıldığında, BCD yolunu seçerek önce D noktasına ulaşan karınca, E yiyecek kaynağından geriye dönerken D noktasına gelir ve hangi yoldan geri döneceğine karar vermek zorunda kalır. Bu durumda BCD yolu BHD yolundan daha kısa olduğu için bu yol üzerinden gelen karınca sayısı ve doğal olarak bıraktıkları feromen miktarı daha fazla olur. Bu nedenle karınca BCD yolunu kullanmaya karar verir.

Kısa yol üzerindeki feromen miktarı uzun yol üzerindeki feromen miktarına göre çok daha hızlı artar. Arkadan gelen karıncalarda feromeni daha yoğun olan yolu seçmek eğilimi olduğundan kısa olan yolun uzun olan yola göre seçilme olasılığını arttırır (Dorigo, 1992; Quyang ve Yan, 2004). Bu durumda kısa olan yol kendiliğinden belirlenmiş ve gittikçe daha fazla kullanılmaya başlanılmıştır.

4.3. Karınca Koloni Sistemi (ACS-Ant Colony System)

Doğal hayat içerisinde yer alan karıncaların bu davranışlarını ve karakteristiklerini başta en kısa yol problemleri olmak üzere farklı pek çok optimizasyon problemlerine uygulanabileceğini ilk kez Marco Dorigo ve arkadaşları 1992 yılında ortaya koymuştur. (Gambardella ve Dorigo, 1995; Dorigo ve Blum, 2005)

Karınca kolonisi optimizasyonunda (KKO) kullanılan karıncalar basit hareketli ajanlardır. Bu ajanlar, çeşitli tipteki problemleri çözmek için gerçek karıncalar gibi feromen maddesini kullanırlar. Bir düğüm üzerinden geçen her karınca birer iz bırakıyor gibi düşünerek feromen maddesi modellenmiş olur. Karınca, karar verirken feromen miktarının yoğunluğu ve gerçek karıncalarda olmayan farklı sezgisel parametreler etkili olmaktadır. (Sim, vd., 2002; Leith ve Takahara, 2003)

4.3.1. Karınca kolonisi sisteminin matematik modeli

Algoritmanın kullanıldığı problemlerde düğümler arasında uzaklık ölçüsü olarak Öklit (*Euclidian*) uzaklık ölçüsü de denilen mutlak uzaklık değeri kullanılmaktadır. Öklit uzaklık ölçüsüne göre i düğümü ile j düğümü arasındaki uzaklık aşağıdaki Denklem 4.1 ile hesaplanmaktadır.

$$d_{ij} = \left[\left(x_i - x_j \right)^2 + \left(y_i - y_j \right)^2 \right]^{1/2} \tag{4.1}$$

N tane düğümü olan ve E tane kenara sahip bir ağ yapısı G(N, E) olarak gösterilmektedir.

Denklem 4.2, t zamanında i düğümündeki karınca sayısını göstermektedir.

$$b_i(t)(i = 1,...,n) \tag{4.2}$$

Denklem 4.3, toplam karınca sayısını göstermektedir.

$$m = \sum_{i=1}^{n} b_i(t) \tag{4.3}$$

Ağ ortamında hareket eden her bir karınca aşağıdaki davranışları gerçekleştirir (Dorigo, vd., 1996):

- Karınca, gideceği bir sonraki düğümü bağlı kenar üzerindeki iz miktarının ve düğüm uzaklığının bir fonksiyonu olan olasılık değerine göre seçer.
- Karıncanın bir düğümden birden fazla kere geçmesine izin verilmez. Bu amaçla karıncanın geçtiği düğümler *tabu listesi* adı verilen bir veri kaydında tutulur. Bir düğümden daha önce geçilip geçilmediği bu listeye bakılarak kontrol edilir.
- Karınca turu tamamladığı sırada üzerinden geçtiği her (i, j) kenarına feromen karşılığı olarak bir kayıt bırakır.

$\tau_{ij}(t)$ İfadesi, t anında (i, j) kenarı üzerindeki feromen yoğunluğunu göstermektedir. Her karınca her t anında bir sonraki düğümü seçer ve (t +1) zamanında seçtiği düğümde olur. Bu durumda (t, t+1) aralığında m karınca tarafından m hareket gerçekleştirilir. Algoritmanın n iterasyonu sonucunda her karınca bir tur tamamlamış olur. Bu noktada aşağıdaki Formül 4.4'e göre feromen yoğunluğu güncelleştirilir.

$$\tau_{ij}(t+n) = \rho.\tau_{ij}(t) + \Delta\tau_{ij} \tag{4.4}$$

Denklem 4.4 içerisinde yer alan ρ bir katsayı değeridir ve $(1-\rho)$ değeri, t ile t+n zaman aralığında feromen miktarının buharlaşmasını ifade etmektedir $(0 < \rho < 1)$.

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \tag{4.5}$$

Denklem 4.5 'de gösterilen $\Delta\tau_{ij}^{k}$, k. karıncanın t ile t+n zaman aralığında (i, j) kenarı üzerinde bıraktığı iz miktarıdır.

Karıncanın kullandığı herhangi bir (i,j) bağlacı üzerinde bıraktığı feromen miktarı Denklem 4.6 ile hesaplanır. Dolayısıyla geçilmemiş bir bağlaçta feromen miktarı sıfır olur. Q parametresi sabit bir değerdir. L_k Parametresi ise k. karıncanın tur uzunluğunu ifade etmektedir.

$$\Delta\tau_{ij}^{k} = \left\{ \frac{Q}{L_k} \right\} \tag{4.6}$$

Algoritmada k. karıncanın i. düğümden j. düğüme geçiş olasılığını hesaplamak için Denklem 4.7 kullanılır. Denklem 4.7'de yer alan ifadelerin açıklamaları aşağıdaki gibidir:

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}.[\eta_{ij}]^{\beta}}{\sum_{k\in A_k}[\tau_{ik}(t)]^{\alpha}.[\eta_{ik}]^{\beta}} & k.izin.verilen.bir.seçimse \\ 0 & aksi.halde \end{cases} \tag{4.7}$$

$\tau_{ij}(t)$: t anında (i,j) köşelerindeki feromen izi miktarı.

η_{ij} : (i,j) köşeleri arasındaki görünürlük (*visibility*) değeridir. $\eta_{ij} = 1/d_{ij}$ denklemi şeklinde ifade edilebilir.

α : feromen ize verilen bağıl önemi gösteren parametredir.

β : görünürlük değerine verilen önemi gösteren parametredir.

A_k: Henüz seçilmemiş düğüm noktaları kümesidir.(izin verilen *Tabu-listesi*)

Ağ ortamı üzerinde yer alan tüm düğümler gezildiği zaman bir tur ya da bir iterasyon tamamlanmış olur ve Denklem 4.4'e göre feromen iz miktarı güncellenir.

4.3.2. Algoritmanın işleyişi

Algoritmanın başlangıcında zaman sıfırdır ve karıncalar farklı düğümlere rasgele dağılmışlardır. Bu arada her yolun üzerindeki feromen yoğunluğunun birbirine eşit olduğu kabul edilir ve uygun bir değer verilir. *Tabu listesi'nin* ilk elemanı algoritmanın başlangıç düğümüdür.

Başlangıç anından sonra her karınca Denklem 4.7'de verilen geçiş olasılık fonksiyonunu kullanarak i. düğümden seçilen j. düğüme doğru harekete başlar. Her t anında karıncalar hareket halindedirler ve (*i,j*) yolu üzerindeki bırakılan iz alınarak yeni değerlerin hesaplanması için kullanılır (Ngo, vd., 2004). Her karınca hareketi sırasında Denklem 5.4 ve Denklem 5.7'ye göre feromen güncellemesini ve geçiş olasılık değerlerini hesaplar.

Algoritmanın n. iterasyonu sonunda, her karıncanın *tabu-listesi* dolmuş olur. Her karıncanın sahip olduğu L_k değerleri karşılaştırılır ve bunlardan en küçüğü en kısa yol değeri olarak hesaplanır. Daha sonra *tabu-listelerin* içi boşaltılır. Bu süreç sonsuza kadar sürer. Ancak uygulamada iterasyon sayısı kullanıcı tarafından tayin edilir.

4.4. Karınca Ağı (ANTNET)

Kombinatoryal optimizasyon ve telefon ağı problemlerini çözmek için yapay karınca kolonileri tekniklerinden ilham alınarak AntNet adlı bir uygulama geliştirilmiştir (Caro ve Dorigo, 1998; Doi ve Yamamura, 2005) Bu tekniklerin ana fikirlerini aşağıdaki gibidir (Caro ve Dorigo, 1998):

1. Karıncalar olarak adlandırılan yapay ajan topluluğunun problemin yeni çözümlerini üretmek için tekrarlı ve eşzamanlı simülasyonlarla arama işlemleri yapması.

2. Çözüm yaratmak için stokastik yerel aramanın ajanlar tarafından yapılması.
3. Daha iyi çözümler için gelecekteki arama yönelimlerinde geçmiş simülasyonlardan toplanan bilgilerin kullanımı.

4.4.1. AntNet algoritmasının karakteristiği

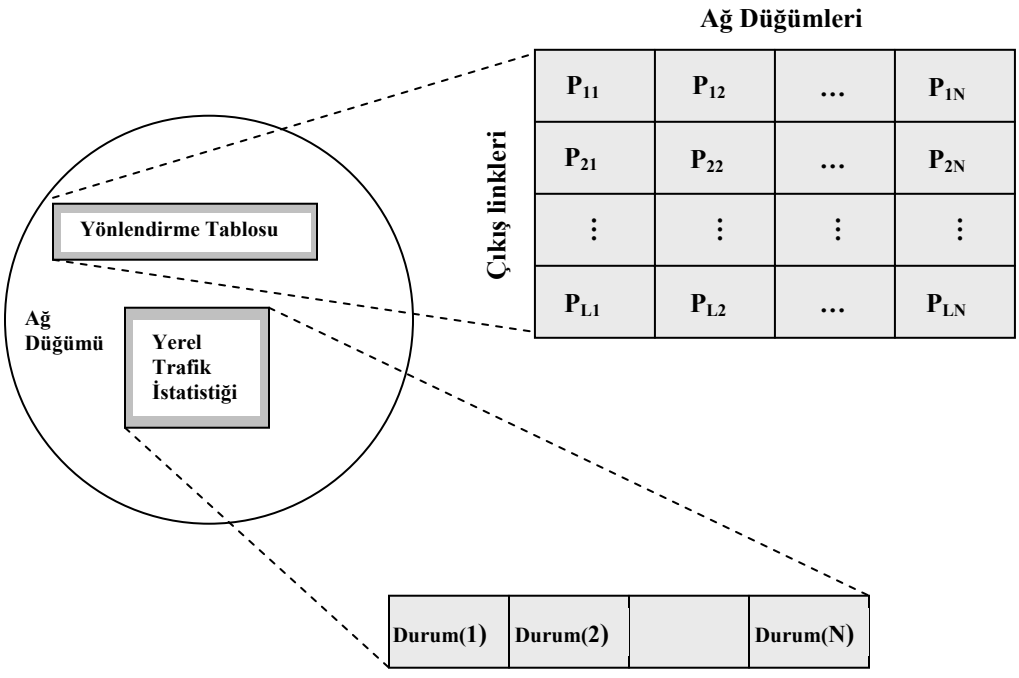
AntNet algoritmasında giden (*forward*) ve dönen (*backward*) karıncalar adı verilen iki tür karınca bulunmaktadır (Liang, vd., 2005; Yang, vd., 2002). Bu karıncalar aynı yapıdadırlar fakat farklı ortamlarda bulunabilmekte ve bağımsız olarak hareket etmektedirler. Bu şekilde farklı girdi değerleri elde edebilir ve sonunda farklı çıktı değerleri üretebilirler. Algoritmanın ana karakteristikleri aşağıdaki gibidir (Caro ve Dorigo, 1998):

- Düzenli aralıklarla ve veri trafiğiyle aş zamanlı olarak karıncalar her ağ düğümünden rasgele seçilen hedef düğümlere doğru asenkron olarak harekete geçerler.
- Karıncalar eş zamanlı ve bağımsız olarak hareket ederler, dolaylı yollardan iletişim kurarlar, düğümlerde bilgileri yerel olarak okur ve yazarlar.
- Her karınca, kaynak ve hedef düğümleri birleştiren minimum maliyetli yol için arama yapar.
- Her karınca hedef düğüme doğru adım adım hareket eder ve her ara düğümde "*stochastic greedy*" kuralını uygulayarak gidilecek bir sonraki düğümü belirler.
- Hareket ederken karıncalar zaman uzunluğu, tıkanıklık durumu ve takip edilen yolun düğüm tanımlayıcısı bilgilerini toplarlar.
- Hedef düğüme ulaştığı zaman, geldikleri yolun aynısını tam aksi yönde hareket ederek kaynak düğüme dönerler.

- Geri dönüş sırasında, karıncalar tarafından ağın durumunun yerel modeli ve her ziyaret edilen düğümün yerel yönlendirme tablosu güncellenir.
- Karıncalar kaynak düğüme döndükleri zaman ölürler.

4.4.2. AntNet algoritması

Algoritma içerisinde her birinin L komşusu olan N düğümlü bir ağda karıncalar kullanılmaktadır. Yönlendirme tablosu uzaklık vektörü (*distance-vector*) algoritmasındaki gibi organize edilir ama tabloda yer alan bilgiler olasılık değerleridir. Şekil 4.2.'de yönlendirme tablosu görülmektedir.



Şekil 4.2. Ağda yer alan her düğüm için veri yapıları

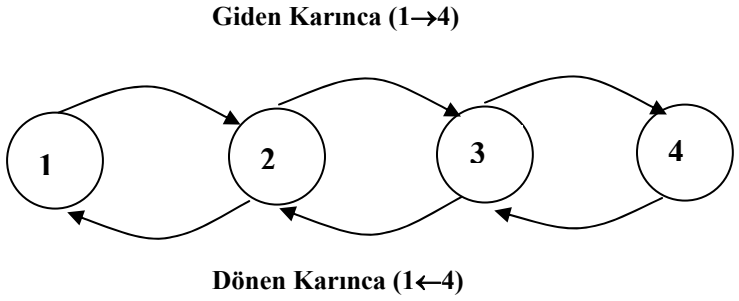
i. τ_k Yönlendirme tablosu, uzaklık vektörü algoritmasındaki gibi olasılık değerleri kullanılarak organize edilir. τ_k, o anki k. düğümde tahmini yönlendirme kuralını ifade eder ve mümkün her d hedef düğümü için ve her komşu n düğümü için, τ_k P_{nd} olasılık değerini saklar. P_{nd} seçilebilmeyi ifade eder. Geçerli yönlendirme kuralı altında d hedef düğümü için n düğümünün sonraki düğüm olarak seçilmesi Denklem 4.8 ile sağlanır.

$$\sum_{n \in N_k} P_{nd} = 1 \quad , \quad d \in [1, N], \quad N_k = \{komşular(k)\} \tag{4.8}$$

ii. $M_k(\mu_d, \sigma_d^2, W_d)$ Veri yapısı yerel k düğümünde ağ üzerindeki trafik dağılımı için basit parametrik istatistik model tanımlar. Bu model adaptiftir ve karıncalar tarafından tecrübe edilen dolaşım zamanları üzerinden hesaplanan ortalama ve varyans değerlerince tanımlanır. $W_{d,}$ karıncanın dolaşım süresini vermektedir. $W_{eniyi\ d}$, değeri karıncaların en iyi dolaşım zamanlarını tutar. Ağdaki her d hedef düğümü için, tahmini ortalama ve varyans, μ_d, σ_d^2, gidiş için beklenen zamanın gösterimini verir.

$$\begin{aligned} \mu_d &\leftarrow \mu_d + \eta(o_{k \to d} - \mu_d), \\ \sigma_d^2 &\leftarrow \sigma_d^2 + \eta((o_{k \to d} - \mu_d)^2 - \sigma_d^2), \end{aligned} \tag{4.9}$$

Denklem 4.9' da yer alan $o_{k \to d}$ ifadesi, k düğümünden d düğümüne yeni saptanan karınca yolculuk zamanıdır.



Şekil 4.3. Algoritma karıncalarının yol üzerindeki ilerlemeleri

AntNet algoritması aşağıdaki gibi açıklanabilir:

1. Δt düzenli aralıklarla her s ağ düğümünden bir karınca (giden karınca) $F_{s\rightarrow d}$, d hedef düğümüne doğru düşük maliyetli yolu ve ağın yük durumunu araştırmak için gönderilir (Şekil 4.3.). Hedefler, yerel yükler tarafından üretilen veri trafiklerine göre yerel olarak seçilir. f_{sd}, $s\rightarrow d$ veri akış ölçüsüdür. Hedef d düğümü için olasılık değeri Denklem 4.10'daki gibidir.

$$P_d = \frac{f_{sd}}{\sum_{d'=1}^{N} f_{sd'}} \tag{4.10}$$

2. Karıncalar hedef düğümlere doğru hareket ederken geçtiği düğümleri ve trafikle ilgili parametreleri belleklerinde tutarlar. Her varılan k. düğümün tanımlayıcısı ve hareket anından k. düğüme varış anına kadar geçen süre bilgileri bellek yığıtına, $S_{s\rightarrow d}(k)$, saklanır.
3. Her k düğümünde, her hareketli karınca ziyaret edilmemiş komşu düğümlerin arasından bir sonraki gidilecek n düğümünü seçer. Komşu n düğümü P'_{nd} olasılığıyla seçilir. P'_{nd} , P_{nd} ve l_n sezgisel doğrulama faktörü kullanılarak hesaplanır. l_n, n.bağlaç kuyruğunun uzunluğunu ifade eder. P'_{nd} değeri aşağıdaki Denklem 4.11.'deki gibi hesaplanmaktadır.

$$P'_{nd} = \frac{P_{nd} + \alpha l_n}{1 + \alpha\left(|N_k| - 1\right)} \tag{4.11}$$

l_n sezgisel doğrulama faktörü aşağıdaki denklem 4.12.'deki gibi hesaplanır. α değeri sezgisel doğrulamanın önemini yönlendirme tablosunda tutulan olasılık değeriyle ağırlıklandırır. Yapılan çalışmalarda α değeri 0,2 – 0,5 aralığında performansı etkilemez.

$$l_n = 1 - \frac{q_n}{\sum_{n'=1}^{|N_k|} q_{n'}} \tag{4.12}$$

q_n değeri, gönderilmeyi bekleyen bit uzunluğunu göstermektedir.

4. Karıncanın daha önceden ziyaret ettiği bir düğüme yeniden ulaşmasına döngü denir. Eğer bir döngü fark edilirse, döngü içinde yer alan düğümler karıncanın yığıtından çıkarılır.
5. Hedef d düğümüne varıldığı zaman, $F_{s\to d}$ giden karıncası $B_{d\to s}$ dönen karınca isminde farklı bir karınca yaratır ve tüm belleğini dönen karıncaya aktarır. Daha sonra kendini öldürür.
6. Dönen karınca, giden karıncanın takip ettiği yolu tam aksi yönde izler (Şekil 4.3.). Yol boyunca her k. düğümde bir sonraki gidilecek düğüm yığıtdan çıkarılır. Dönen karınca yüksek önceliğe sahiptir, çünkü bu tür karıncalar giden karınca tarafından toplanan bilgileri kullanarak düğümlerin yönlendirme tablolarını hızlıca güncellemektir.
7. Komşu düğümden k düğümüne varınca, dönen karınca iki önemli veri yapısını günceller. Bu veri yapıları μ_k ve τ_k'dır.
   i. μ_k, $S_{s\to d}(k)$ yığıttaki değerlerle güncellenir. O anki düğümden başlayan ve d′ hedef düğüme varmak için geçen zamanı, ortalama ve varyans tahminleri ile $W_{d'}$ değerlerini güncellemek için kullanır.
   ii. τ_k Yönlendirme tablosu, $P_{fd'}$ (hedef d′ olduğu zaman f komşu düğümün seçilme olasılığı) olasılık değerinin artması, azalması ve diğer $P_{nd'}$ olasılık değerleriyle değişir.

BÖLÜM 5.

KENDİNİ KLONLAYAN KARINCA KOLONİSİ YAKLAŞIMI

5.1. Giriş

Günümüzde telekomünikasyon teknolojileri, geçmişte kullanılan yıldız mimarilerinin yerine yoğunlukla grid ağ mimarilerine dayandırılmaktadır. Birbirleriyle doğrudan bağlantıları olmayan düğümler, ancak başka düğümler üzerinden bir yol oluşturarak iletişim kurar. Bu yapıda doğal olarak bir düğümden diğerine iletim sağlayabilecek birden fazla yol tanımlanabilir. Bir düğümden diğerine veri iletmek üzere kullanılabilecek yollardan her birisi üzerinde yer alan, yani geçilmesi gereken düğümlerin listesine yol listesi (*path list*) adı verilir.

Telekomünikasyon teknolojilerinde, yönlendirici (*router*) adı verilen ağ cihazı merkezi bir role sahiptir. Yönlendiriciler en uygun yolun belirlenmesini sağlayan ve yönlendirme kararını veren ağ katmanı cihazlarıdır. Yönlendirme kararını alabilmek için yönlendirme algoritmalarından faydalanılır. Bu algoritmalar ise yönlendiricilerin içine gömülü olarak çalıştırılır.

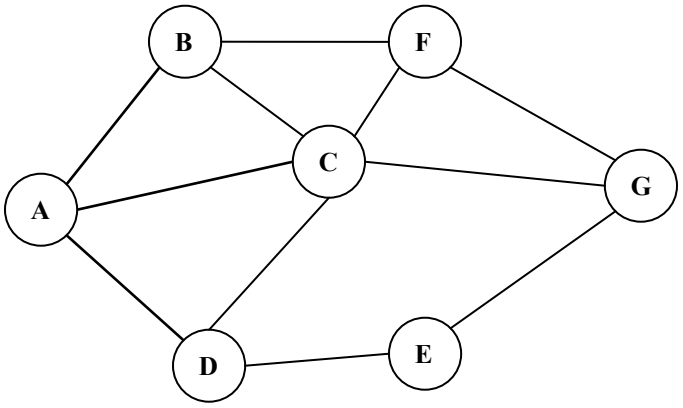
Bu bölümde Kendini Klonlayan Karınca adı verilen farklı bir tip karınca tanımlanmaktadır. Bu karıncalar alışılmış örneklerinden farklı olarak kendilerinden yeni klonlar üretmek şeklinde yeni bir özelliğe sahiptir. Klonlama işlemi sonucu, bir karıncadan aynı özelliklere sahip yeni bir karınca yaratılmaktadır. Yöntemde kullanılan bu karıncalar ağ içinde hareket ederken kendi kararlarını almaktadırlar. Karşılaştıkları duruma göre karıncalar ya kendilerini klonlamakta ya da yok etmektedirler. Böylece sistemde dolaşan karınca sayısı dolaşmasına gerek duyulan karınca sayısına eşit olmakta ve görevini tamamlayan karıncalar kendiliğinden sistemden çıkartılmaktadır.

5.2. Sistem Tanımı

Kendini klonlayan karınca kolonisi yaklaşımında yer alan karıncalar sentetik karıncalardır. Bu karınca tipi de aynı şekilde gerçek hayattaki karıncalar gibi bulundukları ortam hakkında hiçbir bilgiye sahip değildir. Yaklaşımda ortam olarak kast edilen ağ topolojisidir. Bu durumda Kendini Klonlayan Karıncanın ağ topolojisi hakkında bir bilgisi yoktur.

Ağ topolojisi, düğüm ve bağlaçlardan oluşur. Düğüm yönlendiricileri (*router*) ve bağlaç linkleri gösterir. Topoloji içindeki her düğüm en az bir düğümle mutlaka bağlantı halinde olmalıdır. Düğümler eşit yetkilere ve özelliklere sahiptir. Ağ topolojisinde merkezi bir otorite ya da ağ topolojisinin merkezi gibi bir kavram yoktur.

Topoloji içindeki düğümlerin her biri komşuluk tablosu (*neighborhood table*) adı verilen bir tabloya sahiptir. Bu tablo, düğümün komşularının isimlerini ve bağlantı kapılarını tutar. Aşağıdaki Şekil 5.1.'de örnek bir ağ topolojisi ve Çizelge 5.1'de komşuluk tablosu gösterilmektedir.



Şekil 5.1. Kendini klonlayan karınca kolonisi yaklaşımı için örnek ağ yapısı

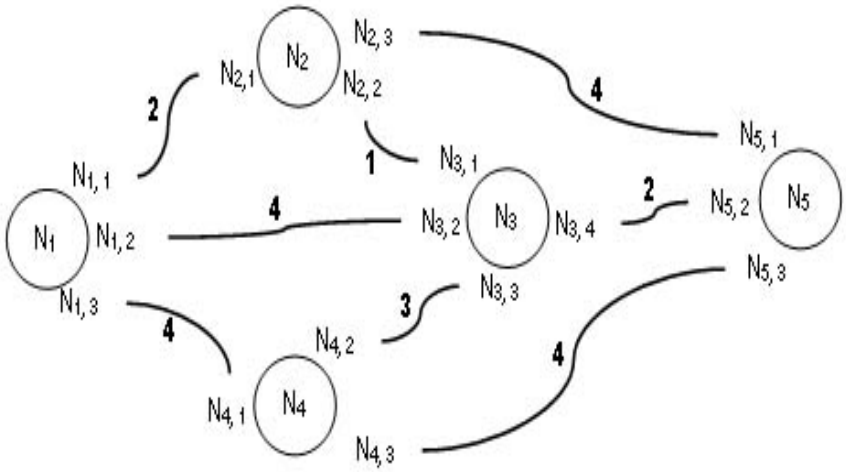
Çizelge 5.1. Komşuluk tablosu

	A	B	C	D	E	F	G
A	-	1	1	1	0	0	0
B	1	-	1	0	0	1	0
C	1	1	-	1	0	1	1
D	1	0	1	-	1	0	0
E	0	0	0	1	-	0	1
F	0	1	1	0	0	-	1
G	0	0	1	0	1	1	-

Çizelge 5.1.'de gösterilen komşuluk tablosunda, eğer iki düğüm arasında direkt bir bağlantı varsa çizelge değeri 1, direkt bağlantı yoksa çizelge değeri 0 olur.

Çizelge 5.1'e bakıldığı zaman A düğümünün B, C ve D düğümleri ile direkt bağlı olduğu görülmektedir. Aynı şekilde C düğümünün A, B, D, F ve G düğümleri ile direkt bağlı olduğu görülmektedir.

Örnek olarak n tane düğümden oluşan bir ağ topolojisi düşünülecek olursa, (i = 1, 2, 3, … n), bu durumda düğüm isimleri N_1, N_2, N_3, … N_n olarak gösterilir. Her düğüm en az bir, en fazla (n-1) tane bağlantıya sahip olacaktır. Bu durumda $N_{i,j}$ ifadesi, i. düğümün j. kapısını ifade eder. Her kapı bilinen diğer bir düğümün bir kapısına atanmaktadır.



Şekil 5.2. Ağ üzerinde düğümler, kapılar ve maliyet değerleri

Şekil 5.2.' de örnek olarak $N_{2,3}$ ifadesinin 2. düğümün 3. kapısını temsil ettiği ve $N_{3,4}$ ifadesinin 3. düğümün 4. kapısını temsil ettiği görülebilir. Bağlaçlar üzerinde yer alan değerler ise bağlaçların maliyetlerini gösterir.

Topoloji içindeki düğümlerin her biri feromen durum bayrağı (*pheromone status flag*) adı verilen bir bayrak (*flag*) taşır. Bu bayrak ile feromen maddesi benzer özellikler taşırlar. Doğal hayattaki karıncalar geçtikleri yolları işaretlemek için feromen maddesini yere bırakırlar böylece bu yolu kullandıklarını anlarlar. Bu bayrağın işaretlenmesi ile de benzer özellikli diğer bir karıncanın önceden bu düğüme ulaştığı anlaşılır. Eğer bayrak işaretlenmemişse daha önceden benzer özellikli diğer bir karınca bu düğüme ulaşmamıştır. Bu durumda düğüme gelen karınca, düğüme ilk ulaşan karıncanın kendisi olduğunu anlayacaktır.

Yaklaşım içinde bahsedilen ağ topolojisinde, düğümler sadece kendilerine komşu düğümlerin bilgilerine sahiptir. Topolojinin nasıl olduğu ya da düğümler arasındaki uzaklık bilgisi, zaman bilgisi ve tıkanıklık durum bilgisi gibi herhangi bir bilgi bu yaklaşım içerisinde bilinmemektedir.

5.3. Kendini Klonlayan Karıncanın Davranışı

Kendini klonlayan karınca kolonisi yaklaşımı arama ve yönlendirme süreçlerinde kullanılmak üzere tasarlanan bir yaklaşımdır. Ağ topolojisinin bilinmediği durumlarda ya da ağ topolojisinin sürekli değişiklik gösterdiği durumlarda, iki nokta arasındaki en uygun yolun belirlenmesi için kullanılmaktadır. Kendini klonlayan karınca kolonisi yaklaşımında kendini klonlayan karınca adı verilen bir tür sentetik karınca kullanılmaktadır. Bu tip karıncalar karşılaştıkları farklı durumlara göre bir değerlendirme yapar ve sonra ya kendilerini klonlar ya da yok ederler.

Bu yaklaşım içinde kullanılan karıncaların tümü, yol listesi (*path list*) bilgisine sahiptir. Bu listede, karınca tarafından ziyaret edilen düğümlerin isimleri ve iki düğüm

arasında geçen süre bilgisi tutulur. Yaklaşım içinde yer alan klonlanan karıncaların yapısı aşağıdaki gibidir:

<kaynak_id> <hedef_id> <yol listesi>

Karıncanın yapısında yer alan kaynak_id, kaynak düğümün ismini tanımlar. Hedef_id alanı hedef düğümün ismini gösterir. Yol listesi (*path list*) alanı yolculuk sırasında geçilen düğümlerin isimlerini ve geçen süreyi vermektedir. Kendini klonlayan karıncanın yapısında, Perkins ve arkadaşlarının (Perkins ve Royer, 1999) ortaya koyduğu yaklaşımdaki maksimum düğüm sayısı (*Hop Count*) alanı yer almamaktadır. Maksimum düğüm sayısı alanı ile tanımlanan düğüm sayısı değerine ulaşılırsa bu durumda yolculuk sona ermektedir ve hedef düğüm ağ topolojisi içerisinde yer alsa bile ulaşılamamaktadır. Bu tez içerisinde ortaya konulan yaklaşım da ise eğer hedef düğüm ağ topolojisi içerisinde yer alıyorsa karınca mutlaka hedefe ulaşacaktır. Karıncanın yapısında, maksimum düğüm sayısı alanı yer almadığından belirli bir sayıya kadar gidip dönmek diye bir kural yoktur. Eğer belirtilen isimde bir düğüm topoloji içerisinde yer almıyorsa süreç içerisinde karıncaların tamamı yok olmuş olacaktır.

Kendini klonlayan karınca, s kaynak düğümünden d hedef düğümüne ulaşmak için kaynak düğümü üzerinde yaratılır ve karıncanın yol listesinin başına kaynak düğümün ismi eklenir. Daha sonra karınca, kaynak düğümün sahip olduğu bağlaç sayısına bakar ve bağlaç sayısının bir eksiği kadar karıncayı klonlar. Klon karıncaların tümü klonlanan karınca ile aynı özelliklere sahiptir, dolayısıyla tümü klonlanan karıncayla aynı yol listesi bilgisine sahiptir. Klonlanan karıncaların her biri düğümün sahip olduğu ayrı bağlaçlardan gönderilir.

Topoloji içerisinde düğümler ve bağlaçlardan geçen karıncalar karşılaştıkları durumlara göre farklı davranışlar sergilerler.

Bağlaç üzerinde hareketlerine devam eden klon karıncalar bir düğüme ulaştıkları zaman öncelikle düğümün feromen durum bayrağını kontrol ederler. Eğer bayrak işaretlemiş ise klon karınca bu düğüme kendisinden önce bir başka klon karıncanın vardığını anlar. Bu durumda kendisinin takip ettiği yoldan daha kısa olan bir başka yolun olduğunu anlar ve bundan sonra gereksiz bir trafik yaratarak ağ içinde yük oluşturmamak için kendisini öldürmeye karar verir. Eğer bayrak işaretlenmemiş ise klon karınca

düğüme ilk olarak kendisinin ulaştığını anlar. Klon karınca düğümün ismini ve önceki düğüm ile vardığı düğüm arasındaki geçen zaman bilgisini kendi yol listesine ekler. Daha sonra klon karınca vardığı düğümün hedef düğüm olup olmadığına bakar. Eğer ulaştığı düğüm hedef düğüm ise arama süreci tamamlanmış olur ve klon karınca yol listesinde yer alan düğümlere doğru ters yönde harekete geçerek yolculuğunu kaynak düğüme doğru devam ettirir. Kaynak düğüme ulaştığı zaman elde ettiği yol listesi bilgisini kaynak düğüme aktarır. Aktarım işlemi bittiği zaman klon karınca kendisini öldürür. Klon karıncanın ulaştığı düğüm hedef düğüm değil ise klon karınca düğüm ismini kendi yol listesine ekler ve düğümün sahip olduğu bağlaç sayısının bir eksiği kadar kendisini klonlar. Klonlanan karıncaların tümü düğümün sahip olduğu bağlaçlardan ayrı ayrı gönderilir. Bu süreç klon karıncalardan birinin hedef düğüme ulaşmasına kadar devam eder. Klonlanan diğer karıncalar ise zaman içerisinde kendilerini öldürürler.

Kendini klonlayan karınca kolonisi yaklaşımının adımları aşağıdaki gibi açıklanmaktadır (Esin ve Erdogan, 2006).

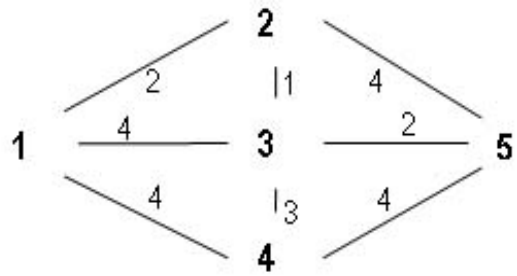
1. *Varılan düğümün feromen durum bayrağını oku.*
2. *Eğer bayrak işaretlenmiş ise 11. adıma git.*
3. *Düğüm ismini oku.*
4. *Düğüm ismini kendi yol listene ekle.*
5. *Önceki düğüm ile varılan düğüm arasındaki geçen zaman bilgisini yol listesine ekle.*
6. *Eğer düğüm ismi hedef düğüm ise feromen durum bayrağını işaretle ve kaynak düğüme doğru dönüş yoluna çık.*
7. *Eğer düğüm ismi hedef düğüm değilse feromen durum bayrağını işaretle ve düğümün komşuluk tablosunu oku.*
8. *Düğümün sahip olduğu komşuluk kadar karıncayı klonla.*
9. *Yeni düğüme varmak için karıncaları yola gönder*
10. *Adım 1'e git.*
11. *Son.*

Kendini klonlayan karınca kolonisi yaklaşımında kullanılan karıncalar birbirlerinden bağımsız olarak yollarına devam ederler ve kendilerini klonlama ya da

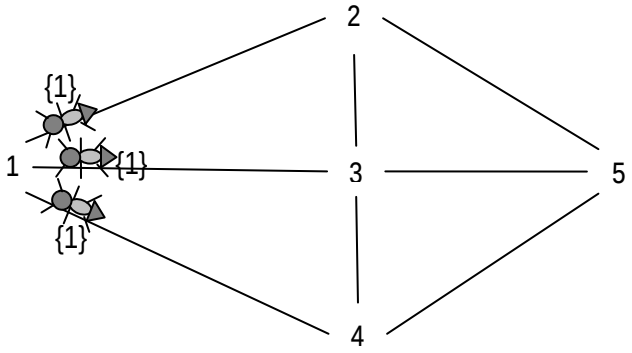
öldürme kararını bağımsız olarak alırlar. Karıncalar ayrık zamanlı ve paralel olarak hareketlerine devam ederler.

5.4. Kendini Klonlayan Karınca Kolonisi Yaklaşımının Uygulama Süreci

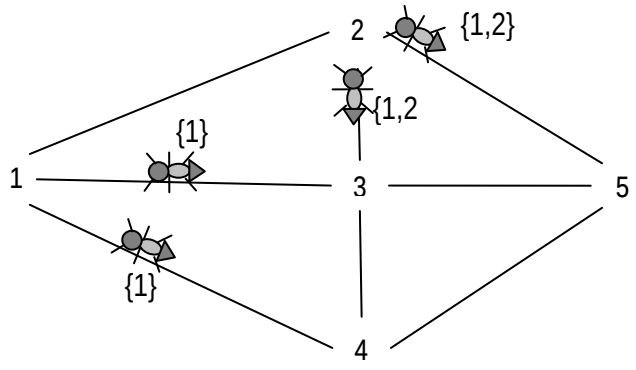
Kendini klonlayan karınca kolonisi yaklaşımının uygulama süreci aşağıda verilen örnek bir ağ topolojisi ile açıklanmaktadır. Şekil 5.3'te gösterilen örnek ağ topolojisinde, düğümler ve düğümlerin birbirlerine bağlanmasını sağlayan bağlaçlar görülmektedir. Bağlaçların üzerinde uzunluk bilgileri bulunmaktadır. Yaklaşımın uygulanması sırasında uzunluk bilgisine ihtiyaç yoktur fakat yaklaşımın kolay anlaşılması için örnek ağ topolojisi üzerinde uzunluk bilgileri de verilmektedir.



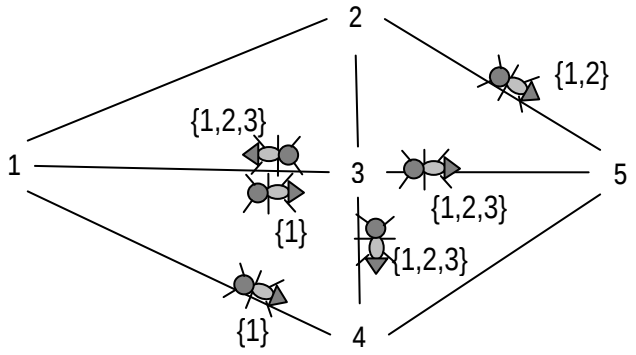
Şekil 5.3. Örnek ağ topolojisi ve uzaklık bilgisi



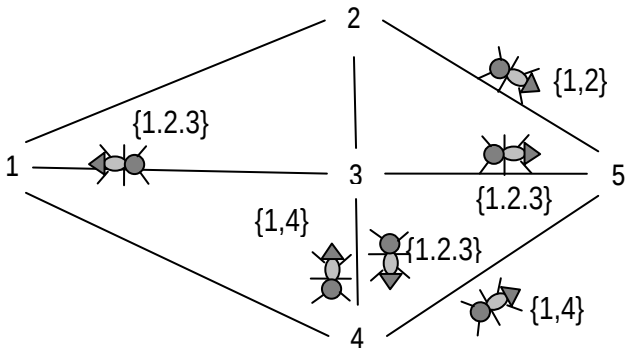
Şekil 5.3.a. Uygulamanın başlangıç anı



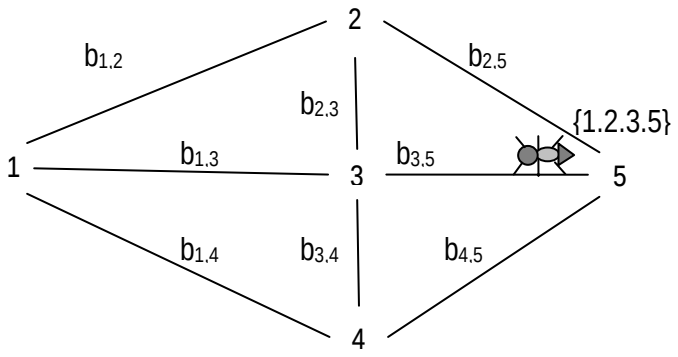
Şekil 5.3.b. İki birim zaman sonraki uygulama anı



Şekil 5.3.c. Üç birim zaman sonraki uygulama anı



Şekil 5.3.d. Dört birim zaman sonraki uygulama anı



Şekil 5.3.e. Beş birim zaman sonraki uygulama anı

Yaklaşımın adım adım gerçekleştirdiği işlemler yukarıdaki Şekillerde gösterilmektedir. Uygulamada yer alan karıncaların birim zamanda bir birimlik mesafe aldıkları kabul edilmektedir. Yaklaşımın işlem adımları aşağıda anlatılmaktadır.

Uygulamanın başlangıç anı Şekil 5.3.a.'da gösterilmektedir. 1 numaralı düğüm kaynak ve 5 numaralı düğüm hedef düğüm olsun. Başlangıç anında bir tane klon karınca yaratılır. Karıncanın yol listesine kaynak düğümün ismi eklenir. Daha sonra karınca, düğümün komşuluk tablosuna bakar. Komşu sayısı kadar karınca klonlanır. Klonlanan her karınca aynı özelliklere sahip olduğundan dolayı aynı yol listelerine ve aynı hedef düğüm bilgisine sahiptir. Karıncaların her biri ayrı yollardan yolculuklarına başlar.

En genel şekliyle i. düğüm ile j. düğümü bağlayan bağlaç b_{ij} ile gösterilebilir. Bu durumda 3. düğümü 4. düğüme bağlayan bağlaç $b_{3,4}$ gösterimi ile ifade edilebilir. Şekil 5.3.b'de, 2 birim zaman geçtikten sonraki karıncaların durumları görülmektedir. $b_{1,2}$ yolunu kullanan klon karınca 2 numaralı düğüme ulaşır. 2 numaralı düğüme ulaşan klon karınca, düğümün feromen durum bayrağını kontrol eder. Eğer bayrak işaretli ise bu durumda başka bir klon karıncanın düğüme daha önceden ulaştığını anlar ve kendisini öldürür. Şekil 5.3.b'de klon karıncanın ilk olarak düğüme ulaştığı görülmektedir ve klon karınca ulaştığı düğümün ismini okur. Düğümün ismini yol listesine ekler ve feromen durum bayrağına işaret koyar. Daha sonra düğümün komşuluk tablosunu bakar. Düğümün sahip olduğu komşu sayısı kadar karınca klonlanır ve her bir karınca bu ayrı yollardan yolculuklarına devam eder.

Şekil 5.3.c.'de, 3 birim zaman geçtikten sonraki karıncaların durumları görülmektedir. $b_{2,3}$ yolunu kullanan klon karınca 3 numaralı düğüme ulaşır. Aynı şekilde düğümün feromen durum bayrağını kontrol eder. Bayrak işaretli olmadığı için klon karınca düğüm ismini kendi yol listesine ekler ve düğümün feromen durum bayrağına işaret koyar. Daha sonra düğümün komşuluk tablosuna bakar ve kendisini düğümün komşuları sayısı kadar klonlar. Klon karıncaların her biri ayrı yollardan yolculuklarına devam ederler.

Şekil 5.3.d.'de, 4 birim zaman geçtikten sonraki karıncaların durumları görülmektedir. $b_{1,3}$ ve $b_{1,4}$ yollarını kullanan klon karıncalar sırasıyla 3 numaralı ve 4 numaralı düğümlere varırlar. 3 numaralı düğüme varan klon karınca düğümün feromen durum bayrağını kontrol eder. Bayrağın işaretli olduğunu görür ve düğüme daha önceden ulaşıldığını anlar. Bu durumda klon karınca kendisini öldürür. 4 numaralı düğüme ulaşan klon karınca düğümün feromen durum bayrağını kontrol eder. Düğümün bayrağını işaretler. Düğümün ismini kendi yol listesine ekler. Düğümün komşuluğunun bir eksiği kadar kendini klonlar ve klon karıncaların her biri ayrı yollardan yolculuklarına devam ederler.

Şekil 5.3.e.'de, 5 birim zaman geçtikten sonraki karıncaların durumları görülmektedir. $b_{3,5}$ yolunu kullanan klon karınca 5 numaralı düğüme ulaşır. 5 numaralı düğüm hedef düğümdür. Klon karınca öncelikle düğümün feromen durum bayrağını kontrol eder. İşaretli olmadığını görür ve düğümün feromen durum bayrağını işaretler. Daha sonra bayrağın ismini okuyarak kendi yol listesine ekler. Düğümün hedef düğüm olduğunu görür ve klonlama süreci tamamlanır. Klon karınca kendi yol listesini aksi yönde takip etmeye başlar ve kaynak düğüme ulaşır. Diğer klon karıncalar zaman içerisinde kendilerini öldüreceklerdir.

Sürecin sonunda kaynak düğüme ulaşan klon karınca sahip olduğu yol listesini kaynak düğüme aktarır. Aktarma işlemi tamamlandıktan sonra klon karınca kendini öldürür.

5.5. Kendini Klonlayan Karınca Kolonisi Yaklaşımı ile Yönlendirme Tablolarının Güncellenmesi

Yönlendirme kararlarını almak için yönlendiriciler yönlendirme tabloları kullanırlar. Her yönlendirici kendi yapıları içerisinde yönlendirme tablosu saklamaktadır. En uygun yolun hesaplanması için bu tablolar içerisindeki bilgileri kullanırlar. Bu yüzden yönlendirme tabloları içerisinde yer alan bilgilerin sürekli olarak güncel olması gerekir. Yönlendirme tabloları belirli periyotlarla güncellenmelidir. Kendini klonlayan karınca kolonisi yaklaşımını kullanarak yönlendirme tablolarının güncellemesi yapılabilmektedir.

Yönlendirme tablolarının güncellemesi için klon karıncaların davranışlarında da bir farklılık yaratılmaktadır. Kendini klonlayan karınca kolonisi yaklaşımı anlatılırken daha önceden başka bir klon karınca tarafından ziyaret edilen bir düğüme bir klon karınca ulaşırsa karınca kendisini yok etmektedir. Burada ise klon karınca, düğümün başka bir klon karınca tarafından önceden ziyaret edildiğini anlarsa kendini öldürmez ve sadece yolculuğunu durdurur. Daha sonra klon karınca o düğümü kendi yol listesine eklemeden yol listesindeki bilgilerine bakarak kaynak düğüme geri döner.

Güncelleme sürecinin başında, güncelleme yapacak olan düğüm, bir klon karınca yaratır ve karıncanın yol listesine kendi ismini ekler. Bundan sonra klon karınca, düğümün sahip olduğu bağlaçlar kadar kendini klonlar. Her klon karınca ayrı bağlaçlardan yolculuklarına başlarlar.

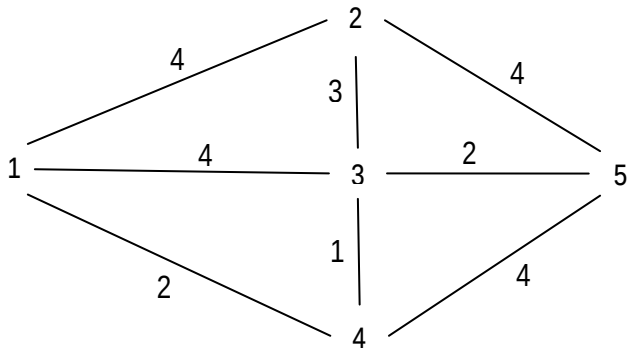
Her ulaşılan düğümde klon karıncalar düğümün feromen durum bayrağını kontrol ederler. Eğer bayrak işaretli ise klon karınca düğümün ismini kendi yol listesine eklemez ve geriye kaynak düğüme doğru yolculuğa başlar. Eğer bayrak işaretli değilse klon karınca düğümün ismini kendi yol listesine ekler. Bundan sonra klon karınca düğümün sahip olduğu bağlaçlar kadar kendini klonlar ve her klon karınca ayrı bağlaçlardan yolculuklarına başlar. Düğüme ulaşan karınca ise geriye kaynak düğüme doğru harekete başlar. Kaynak düğüme ulaştığı zaman kendi yol listesi bilgisini kaynak düğüme aktarır. Diğer klon karıncalar yolculuklarına devam ederler.

Yönlendirme tablosu güncelleme algoritması aşağıdaki gibidir: (Erdoğan, Esin 2006)

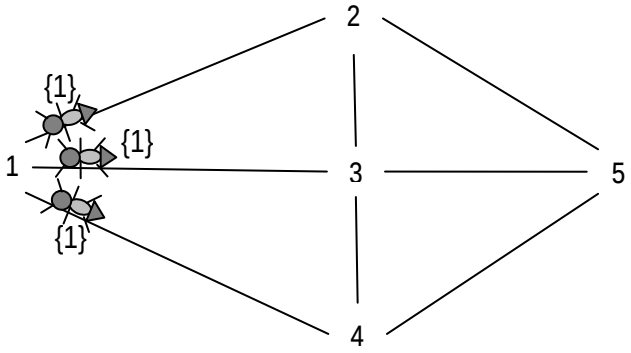
1. *Varılan düğümün feromen durum bayrağını kontrol et*
2. *Eğer bayrak işaretli ise kaynak düğüme geri dön*
3. *Düğüm ismini oku*
4. *Düğüm ismini ve geçen süre bilgisini yol listesine ekle*
5. *Düğüm durum bayrağını işaretle*
6. *Düğümün komşuluk tablosunu oku*
7. *Düğümün yol sayısı kadar karınca klonla*
8. *Klon karıncalaır ayrı yollara gönder ve ilk klon karınca kaynak düğüme dön*
9. *adım 1' e git*

Her düğüm kendi yönlendirme tablosunu güncellemek için bir klon karınca yaratır ve yukarıda anlatıldığı gibi süreci çalıştırırlar.

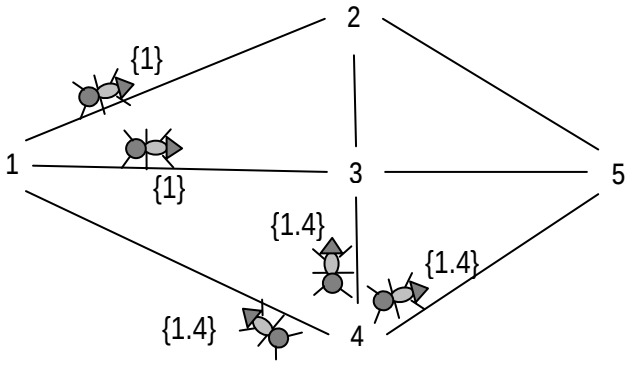
Kendini klonlayan karınca kolonisi yaklaşımıyla düğümlerin (yönlendiricilerin) yönlendirme tablolarını güncelleme süreci aşağıda anlatılmaktadır. Sürecin anlatıldığı basit ağ topolojisi Şekil 5.4'de gösterilmektedir. Uygulamanın çalışması için düğümlerin olması yeterlidir ama süreci açıklamak için düğümler arası maliyet değerleri şekil üzerinde gösterilmektedir.



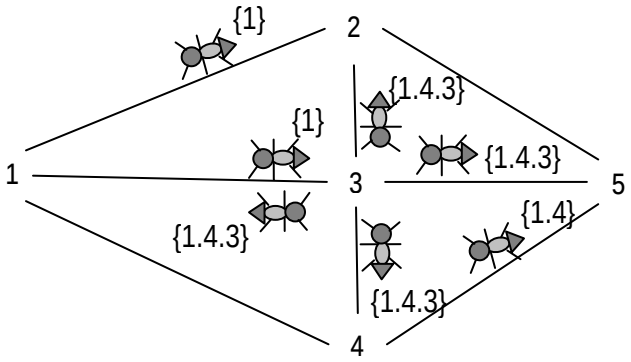
Şekil 5.4. Örnek ağ topolojisi



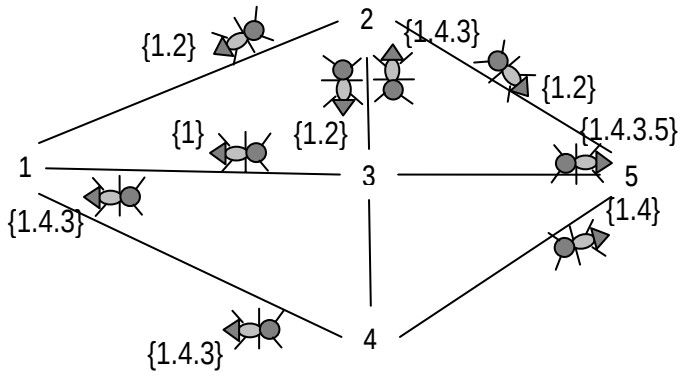
Şekil 5.4.a. Uygulamanın başlangıç anı



Şekil 5.4.b. İki birim zaman sonraki uygulama anı



Şekil 5.4.c. Üç birim zaman sonraki uygulama anı



Şekil 5.4.d. Beş birim zaman sonraki uygulama anı

Şekil 5.4.a, uygulamanın başlangıç durumunu göstermektedir. Düğüm 1, kaynak düğüm olarak kabul edilmektedir, yani 1 numaralı düğümün yönlendirme tablosunun güncelleme süreci anlatılmaktadır. Şekil 5.4.a'da 1 numaralı düğüm bir klon karınca yaratır ve kendi düğüm ismini klon karıncanın yol listesine ekler. Bundan sonra, klon karınca düğümün sahip olduğu bağlaç sayısı kadar kendini klonlar. Her klon karınca ayrı bağlantılardan gönderilir.

Şekil 5.4.b, iki birim zaman geçtikten sonraki karıncaların durumlarını göstermektedir. (1 – 4) yolunu kullanan klon karınca 4 numaralı düğüme ulaşır. Düğümün feromen durum bayrağını kontrol eder ve düğümün ismini kendi yol listesine ekler. Feromen durum bayrağını işaretler. Daha sonra, klon karınca kendisini düğümün sahip olduğu yol sayısı kadar klonlar. Her klon karınca ayrı yollardan gönderilir. Kendini klonlayan ilk karınca ise kaynak düğüme döner. (1 – 2) ve (1 – 3) yollarını kullanan klon karıncalar yollarına devam ederler.

Şekil 5.4.c, üç birim zaman geçtikten sonraki karıncaların durumlarını göstermektedir. (4 – 3) yolunu kullanan klon karınca, 3 numaralı düğüme varır. Klon karınca feromen durum bayrağını kontrol eder. Düğümün ismini kendi yol listesine ekler. Feromen durum bayrağını işaretler kendisini düğümün sahip olduğu yol sayısı kadar klonlar. Her klon karınca ayrı yollardan gönderilir. Düğüme varan karınca ise kaynak düğüme döner.

Şekil 5.4.c, beş birim zaman geçtikten sonraki karıncaların durumlarını göstermektedir. (3 – 5) yolunu kullanan klon karınca, 5 numaralı düğüme varır. Klon karınca feromen durum bayrağını kontrol eder. Düğümün ismini kendi yol listesine ekler. Feromen durum bayrağını işaretler.

Kaynak düğüme geri dönen klon karıncalar, yol listelerini kaynak düğüme aktarırlar. Bu şekilde kaynak düğüm, klon karıncalardan gelen bilgileri kullanarak kendi yönlendirme tablosunu güncellemiş olur. Aktarım işlemi tamamlandıktan sonra klon karıncalar kendilerini öldürürler.

Çizelge 5.2. 1 Numaralı düğüm için yönlendirme tablosu

Hedef Düğüm	**Yönlendirme Listesi**
1	-
2	1 – 2
3	1 – 4 – 3
4	1 – 4
5	1 – 4 – 3 – 5

Çizelge 5.2’ de, 1 numaralı düğüm için yönlendirme tablosu gösterilmektedir. Çizelgede yer alan her satır bir hedef düğüm için en uygun yol listesi bilgisini vermektedir.

BÖLÜM 6.

KENDİNİ KLONLAYAN KARINCA KOLONİSİ YAKLAŞIMININ SİMÜLASYONLARI

6.1. Simülasyon Programı

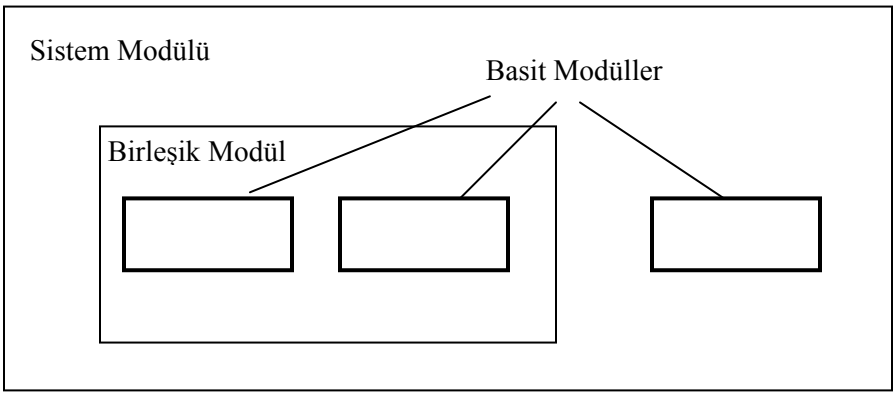
Kendini klonlayan karınca kolonisi yaklaşımının simülasyonu gerçekleştirilirken OMNET++ simülasyon programından faydalanılmıştır. OMNET++, nesne yönelimli, modüler, ayrık olay tabanlı ağ simülatörüdür. (Omnet, 2006) OMNET++ programı aşağıdaki uygulamaları gerçekleştirmek için kullanılmaktadır.

- Telekomünikasyon ağlarının trafik modellemeleri
- Protokol modellemeleri
- Ağlarda kuyruk modellemeleri
- Çoklu işlemcilerin ve diğer donanım sistemlerinin modellemesi
- Kompleks yazılım sistemlerinin performanslarının gelişimi
- Ayrık olay yaklaşımının gerçekleşebileceği diğer sistemlerin modellemeleri

OMNET++ programı, hiyerarşik olarak birbirinin içine geçmiş modüllerden oluşur. İç içe geçmiş bu modüllerin derinliğinin ne kadar olacağı konusunda bir sınır bulunmamaktadır. Bu sayede kullanıcılara mevcut sistemin mantık yapısını model içerisine yansıtma olanağı sağlamaktadır. Modüller, birbirleriyle mesajlaşarak iletişim kurarlar. Mesajlar, karmaşık veri yapıları içerebilir. Modüller, mesajları ya direkt olarak ya da kapılar (*gates*) ve bağlantılardan (*connections*) kurulu önceden tanımlı bir yol üzerinden hedefe gönderebilir.

Her modül birtakım parametrelere sahiptir. Bu parametreler, modül davranışlarını özelleştirmek için kullanılır ve modülün yapısının parametrik şekle çevrilmesini sağlar.

Modül hiyerarşisinin en alt seviyesindeki modüller, davranışları özetler. En alt seviyedeki bu modüllere basit modüller adı verilir. Birden fazla basit modülün içinde yer aldığı diğer bir modül tipi ise birleşik modüllerdir. Aşağıdaki Şekil 6.1.' de basit ve birleşik modüller görülmektedir. Modül yapısı, NED dili kullanılarak OMNET++ programı içinde tanımlanmaktadır. NED, OMNET++ programına özgü ve modül yapısı tanımlamak için kullanılan bir dildir. Modüllerin içerikleri ise simülasyon programının kütüphaneleri kullanılarak C++ programlama diliyle kodlanır.



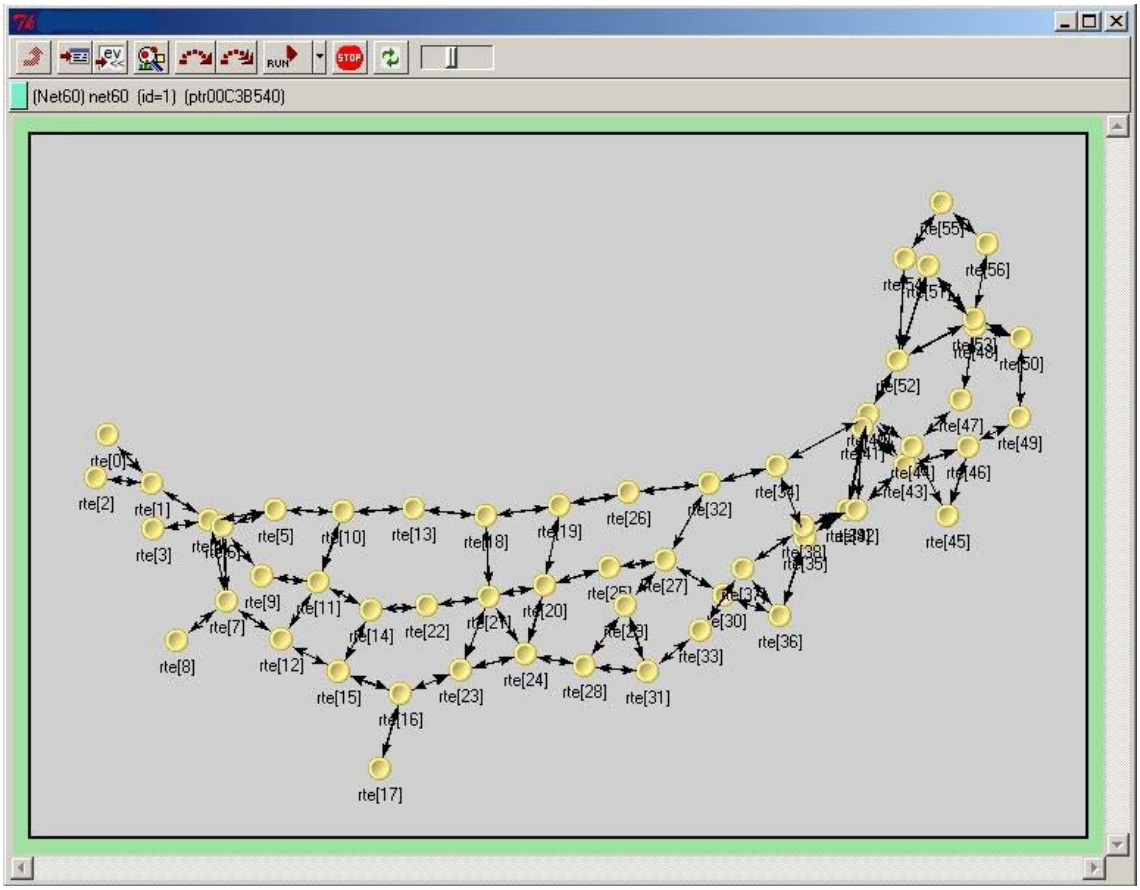
Şekil 6.1. Basit ve birleşik modüller

OMNET++ programı, paralel ve dağıtık simülasyonları da desteklemektedir. OMNET++, paralel ve dağıtık bir simülasyonun parçaları arasındaki iletişim için pek çok mekanizmayı da kullanabilmektedir. Örnek olarak boru hatları (*pipe lines*) verilebilir.

OMNET++ programı haricinde farklı ağ simülasyon programları da bulunmaktadır. Örnek olarak NS-2 (*Network Simulator*), J-Sim (*Java-Simulator*), Opnet, NS (*Network Simulator*) verilebilir. Bu yaklaşımın gerçekleştirilmesinde OMNET++ programının kullanılmasını aşağıdaki maddelerle açıklayabiliriz.

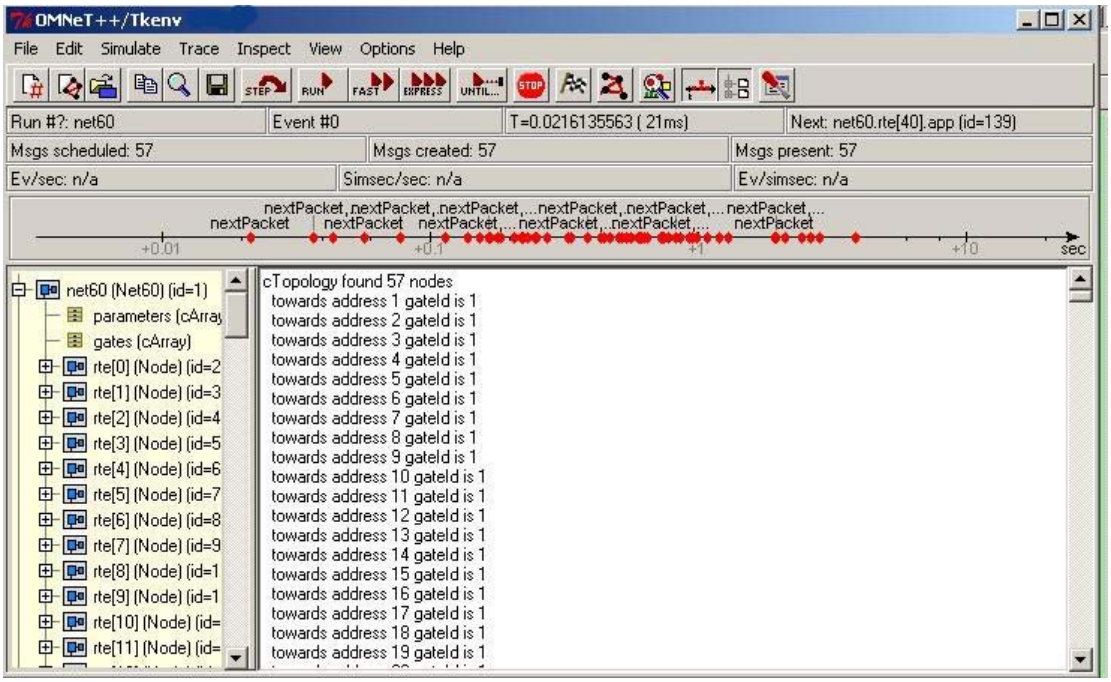
- Genel Kamu Lisansına (GPL- *General Public License*) sahiptir. Bu şekilde akademi lisansı ücretsizdir.
- Kodlama sırasında programın arka kısmında C++ programlama dili kullanılmaktadır. C++ programlama dili yoğun olarak kullanılan ve desteklenen bir dildir.

- OMNET++ programı içerisinde pek çok ağ modeli yer alır. Sahip olduğu kütüphaneler programın kodlanmasını kolaylaştırır.
- Yardıma ihtiyaç olduğu durumlarda, birden fazla alternatif bulmak oldukça kolaydır. Gerek iyi hazırlanmış yardım içerik rehberi gerekse açılmış olan mektup listeleri (*mail-list*) aracılığıyla sorunlara hızlı şekilde yanıt alınabilir.



Şekil 6.2. Simülasyon gösterim ortamı

Şekil 6.2.' de simülasyonun gösterim ortamı görülmektedir. Şekil 6.2'de 57 düğümden oluşan bir ağ ortamı görülmektedir. *rte[]* olarak ifade edilen şekiller düğümleri göstermektedir ve düğümler birbirlerine bağlaçlarla bağlanmaktadır.



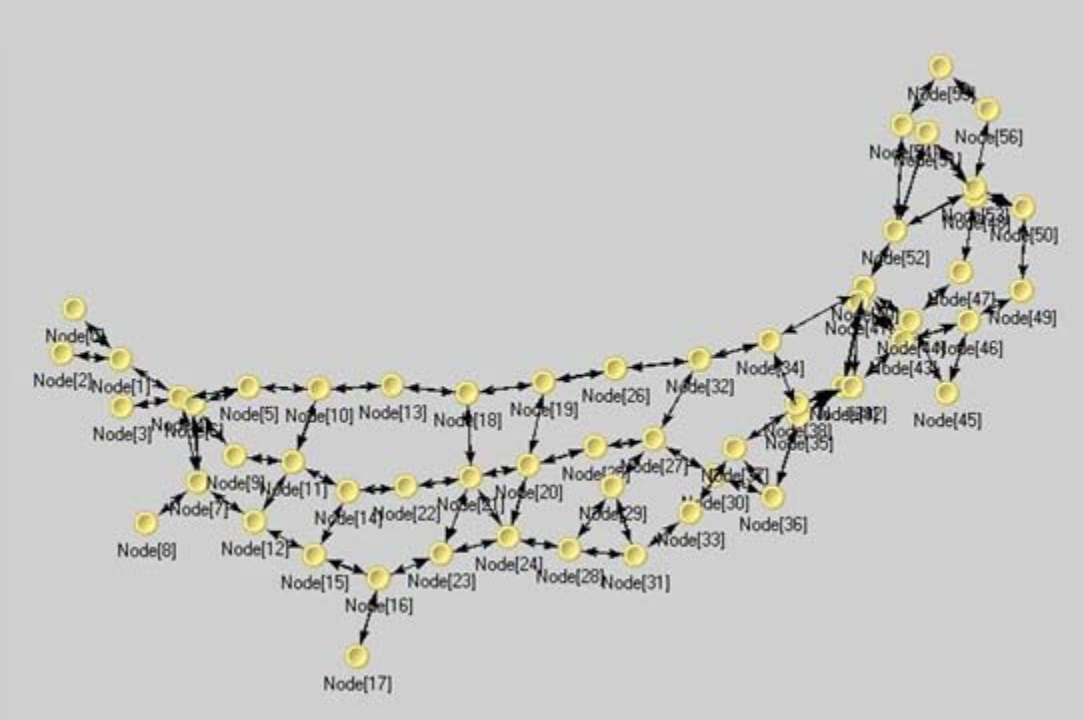
Şekil 6.3. Simülasyon programının işlem adımlarının görüldüğü ekran

OMNET++ programında, modeli oluşturulan simülasyonun çalıştırılması sırasında iki pencere açılmaktadır. Bunlardan biri simülasyonun göründüğü penceredir. İkincisi ise Tkenv penceresidir. Şekil 6.3'de Tkenv ekranı görülmektedir. Bu ekranda simülasyonun işlem adımları ve zamanlama tablosu (*Schedule*) bilgileri yer almaktadır.

6.2. Kendini Klonlayan Karınca Kolonisi Yaklaşımının NTTNET Ağı Üzerinde Uygulaması

Bu bölümde, kendini klonlayan karınca kolonisi yaklaşımı, NTTNET ağı üzerinde uygulanmaktadır.

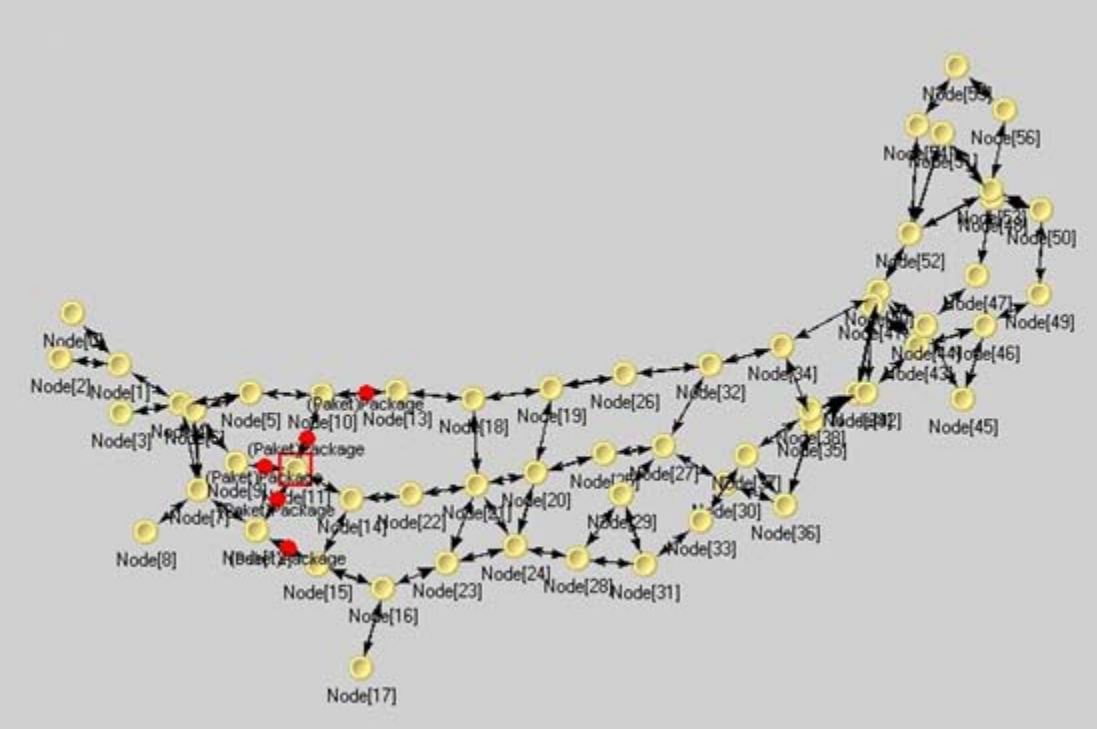
NTTNET (*Nippon Telephone and Telegraph Company*) ağı, Japonya'nın ağ omurgasıdır. Ağ içerisinde 57 düğüm ve 162 çift yönlü hat bulunmaktadır. Her hattın bant genişliği 6 Mbit / saniye'dir. Ağ topolojisi aşağıda Şekil 6.4.'de görülmektedir.



Şekil 6.4. NTTNET – Düğümler

Simülasyon uygulamasında kaynak düğüm olarak 1 numaralı düğüm ve hedef düğüm olarak ise 56 numaralı düğüm seçilmektedir. Simülasyonun başlangıcında bir tane kendini klonlayan karınca 1 numaralı düğüm üzerinde yaratılmaktadır. Daha sonra bu karıncanın yol listesine kaynak düğümün adı ve hedef düğüm bilgisi yazılır ve karınca kendini düğümün sahip olduğu yol sayısı kadar klonlar. Her klon karınca bu ayrı yollardan gönderilirler.

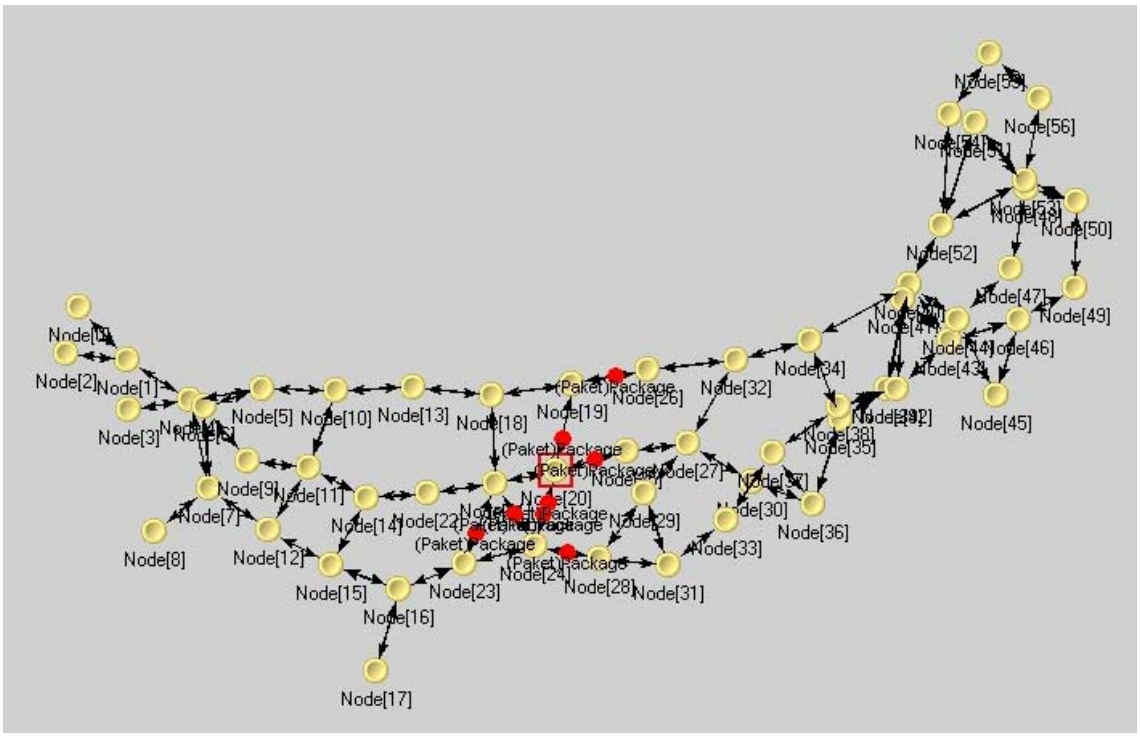
Simülasyon içerisinde, klon karıncalar 5. bölümde anlatılan klon karıncaların davranışlarını sergileyerek yollarına devam ederler. Şekil 6.5.'de klon karıncaların ağ üzerindeki ilerlemelerinin ekran görüntüsü görülmektedir.



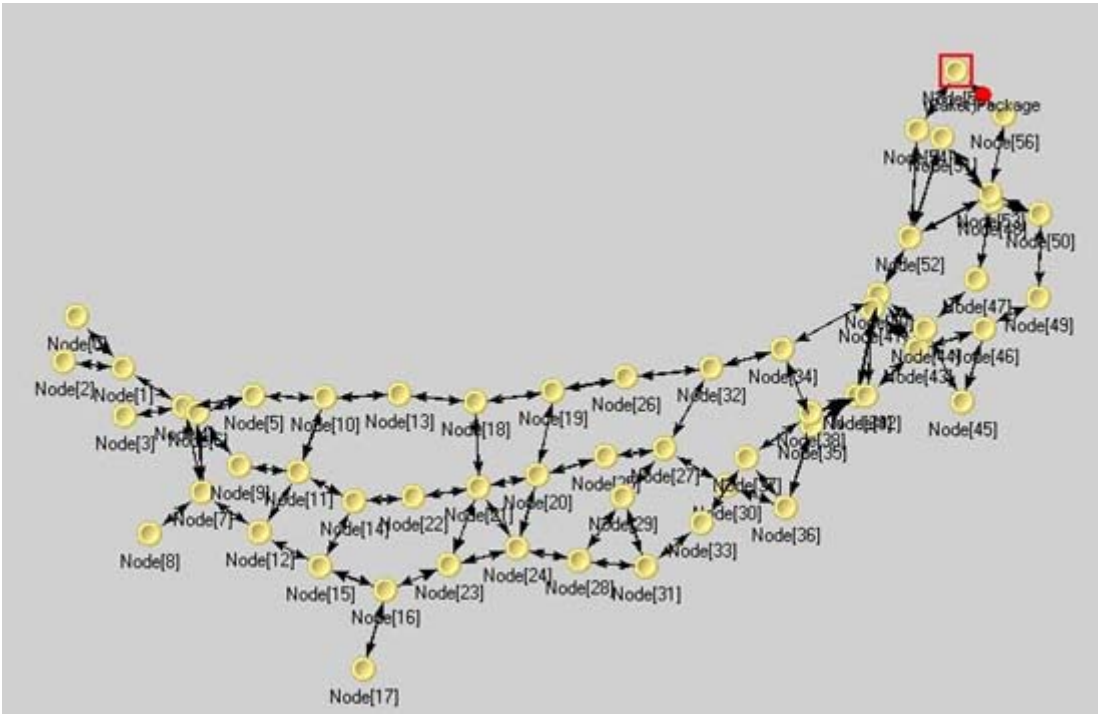
Şekil 6.5. Klon karıncaların ağ üzerindeki ilerlemeleri

Simülasyonun sonunda sadece bir tane klon karınca hedef düğüme ulaşır. Diğer klon karıncalar zaman içerisinde kendilerini yok ederler. Hedef düğüme ulaşan klon karınca kendi yol listesinde, geçtiği düğümlerin isimlerini saklar. Bu yol listesi ise kaynak düğüm ile hedef düğüm arasındaki optimal yolu gösterir. Simülasyonun sonucunda hedefe ulaşan karıncanın yol listesi: ***1 -2 -5 -6 -7 -10 -12 -15 -23 -22 -21 -26 -28 -33 -35 -41 -53 -55 -56*** gibidir. Klon karınca hedef düğüme 36 milisaniye sonra ulaşır.

Şekil 6.6'da simülasyon süreci içerisinde yer alan bir adımın ekran görüntüsü gösterilmektedir. Şekil 6.7'de ise simülasyon sürecinde hedef düğüme varılma anı gösterilmektedir.



Şekil 6.6. Simülasyon sürecinde bir adım



Şekil 6.7. Hedef düğüme varılma anı

Simülasyonda 27 tane klon karınca yaratılmaktadır ve bu 27 klon karınca tüm ağı taramaktadır. Simülasyonun sonunda ise sadece bir klon karınca hedefe varır ve diğerleri

zaman içerisinde kendilerini yok ederler. Tüm ağın taranması ise 47 milisaniye sürmüş ve 47 milisaniye sonra ağda hiçbir klon karınca kalmamıştır. Aşağıda Çizelge 6.1'de bazı örnek hedef düğümler için optimal yol listesi ve süre bilgisi gösterilmektedir.

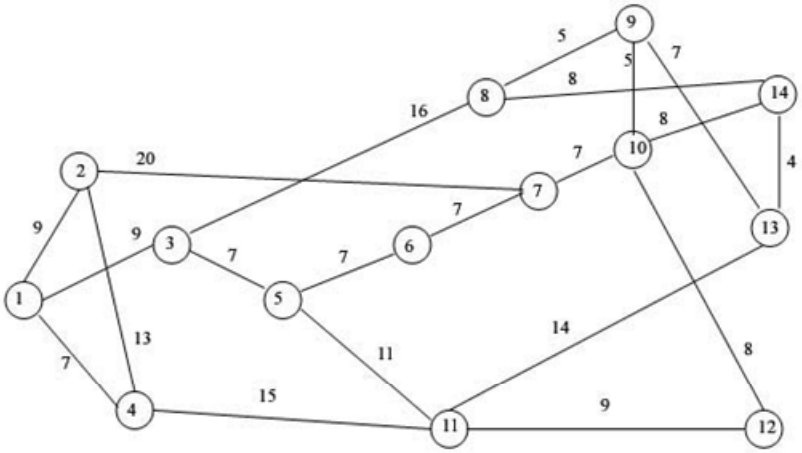
Çizelge 6.1. Örnek düğümler için optimal yol listesi

Kaynak Düğüm	**Hedef Düğüm**	**Yol Listesi**	**Süre Bilgisi**
Düğüm 1	Düğüm 23	1 -2 -5 -6 -7 -10 -12 -15 -23	15 ms
Düğüm 1	Düğüm 38	1 -2 -5 -6 -7 -10 -12 -15 -23 -22 -21 -26 -28 -31 -37 -38	26 ms
Düğüm 1	Düğüm 27	1 -2 -5 -6 -11 -14 -19 -20 -27	22 ms
Düğüm 1	Düğüm 46	1 -2 -5 -6 -7 -10 -12 -15 -23 -22 -21 -26 -28 -33 -35 -41 -45 -46	32 ms
Düğüm 1	Düğüm 13	1 -2 -5 -8 -13	8 ms
Düğüm 1	Düğüm 51	1 -2 -5 -6 -7 -10 -12 -15 -23 -22 -21 -26 -28 -33 -35 -41 -44 -47 -50 -51	36 ms

6.3. Kendini Klonlayan Karınca Kolonisi Yaklaşımının NSFNET Ağı Üzerinde Uygulaması

Bu bölümde kendini klonlayan karınca kolonisi yaklaşımı NSFNET üzerinde uygulanmaktadır.

NSFNET, Amerika Birleşik Devletleri'nin ağ omurgasıdır. Omurga 14 düğümden ve 21 çift yönlü hattan oluşmaktadır. Her hattın bant genişliği 1.5 Mbit / sn.'dir. NFSNET' in ağ topolojisi ve maliyet değerleri aşağıda Şekil 6.8'de gösterilmektedir.



Şekil 6.8. NFSNET topolojisi ve maliyet değerleri

Simülasyonun başlangıç safhasında kaynak düğüm olarak düğüm 1 ve hedef düğüm olarak düğüm 14 belirlenmiştir. Simülasyon bir önceki simülasyonda olduğu gibi çalışmakta ve simülasyon sonunda tek bir klon karınca hedefe varmaktadır. Hedefe varan klon karıncanın geçtiği yolların isimleri yol listesinde tutulmaktadır. Bu listeye göre yol listesinde ***1 – 3 – 8 – 14*** düğümleri tutulmakta ve karıncanın hareketi sırasında geçen süre 33 ms.'dir. Bu liste bilgisi, kaynak düğüm ile hedef düğüm arasındaki optimal yol bilgisini verir. Aşağıdaki tabloda farklı hedef düğümler için bulunan optimal yol bilgileri gösterilmektedir.

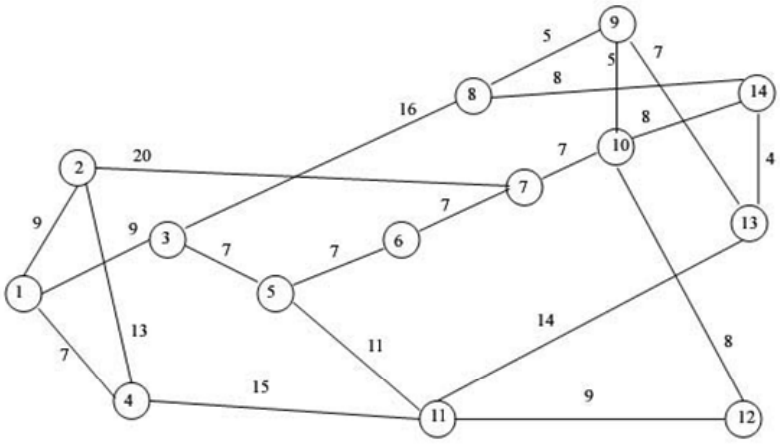
Çizelge 6.2. Örnek düğümler için optimal yol listesi

Kaynak Düğüm	**Hedef Düğüm**	**Yol Listesi**	**Süre Bilgisi**
Düğüm 1	Düğüm 14	1–3–8–14	33 ms.
Düğüm 1	Düğüm 7	1 – 2 – 7	29 ms
Düğüm 1	Düğüm 13	1 – 4 – 11 – 13	36 ms
Düğüm 1	Düğüm 9	1 – 3 – 8 – 9	30 ms
Düğüm 1	Düğüm 11	1 – 4 – 11	22 ms

6.4. Kendini Klonlayan Karınca Kolonisi Yaklaşımı Kullanılarak Yönlendirme Tablolarının Güncellenmesi

Yönlendirme kararı alınırken düğümler kendi yönlendirme tablolarına bakarlar. Yönlendirme tablolarındaki bilgiler ne kadar güncel olursa alınan yönlendirme kararları da o denli doğru olmaktadır. Bu bölümde kendini klonlayan karınca kolonisi yaklaşımı kullanılarak yönlendirme tablolarının güncelleme simülasyonu ve simülasyon sonuçları gösterilmektedir.

Simülasyonda Şekil 6.9.'da gösterilen NFSNET ağ topolojisi kullanılmaktadır. NSFNET ağ topolojisinin ayrıntıları bölüm 6.3.'te anlatılmaktadır.



Şekil 6.9. NFSNET topolojisi ve maliyet değerleri

Simülasyonun başlangıç safhasında, kaynak düğüm olarak düğüm 1 seçilmektedir. Klon karıncalar 5. bölümde açıklandığı şekilde hareket ederler. Simülasyon daha sonra kaynak düğüm olarak düğüm 6, düğüm 9, düğüm 12 ve düğüm 14 kabul edilerek yeniden çalıştırılmıştır. Simülasyon sonucunda elde edilen yönlendirme tablo değerleri aşağıda Çizelge 6.3'de görüldüğü gibidir.

Çizelge 6.3. Güncel yönlendirme tabloları

Kaynak Düğüm 1

Hedef Düğüm		Yol Listesi	Süre
	1	1	0 ms
	2	1–2	9 ms
	3	1–3	9 ms
	4	1–4	7 ms
	5	1–3–5	16 ms
	6	1–3–5–6	23 ms
	7	1–2–7	29 ms
	8	1–3–8	25 ms
	9	1–3–8–9	30 ms
	10	1–3–8–9–10	35 ms
	11	1–4–11	22 ms
	12	1–4–11–12	31 ms
	13	1–4–11–13	36 ms
	14	1–3–8–14	33 ms

Kaynak Düğüm 6

Hedef Düğüm		Yol Listesi	Süre
	1	6–5–3–1	24 ms
	2	6–7–2	27 ms
	3	6–5–3	14 ms
	4	6–5–3–1–4	30 ms
	5	6–5	7 ms
	6	6	0 ms
	7	6–7	7 ms
	8	6–7–10–9–8	24 ms
	9	6–7–10–9	19 ms
	10	6–7–10	14 ms
	11	6–5–11	18 ms
	12	6–7–10–12	22 ms
	13	6–7–10–9–13	26 ms
	14	6–7–10–14	22 ms

Kaynak Düğüm 9

Hedef Düğüm		Yol Listesi	Süre
	1	9–8–3–1	30 ms
	2	9–10–7–2	32 ms
	3	9–8–3	21 ms
	4	9–13–11–4	36 ms
	5	9–10–7–6–5	26 ms
	6	9–10–7–6	19 ms
	7	9–10–7	12 ms
	8	9–8	5 ms
	9	9	0 ms
	10	9–10	5 ms
	11	9–13–11	21 ms
	12	9–10–12	13 ms
	13	9–13	7 ms
	14	9–13–14	11 ms

Kaynak Düğüm 12

Hedef Düğüm		Yol Listesi	Süre
	1	12 – 11 – 4 – 1	31 ms
	2	12 – 10 – 7 – 2	35 ms
	3	12 – 11 – 5 – 3	27 ms
	4	12 – 11 – 4	24 ms
	5	12 – 11 – 5	20 ms
	6	12 – 10 – 7 – 6	22 ms
	7	12 – 10 – 7	15 ms
	8	12 – 10 – 9 – 8	18 ms
	9	12 – 10 – 9	13 ms
	10	12 – 10	8 ms
	11	12 – 11	9 ms
	12	12	0 ms
	13	12 – 10 – 9 – 13	20 ms
	14	12 – 10 – 14	16 ms

Kaynak Düğüm 14

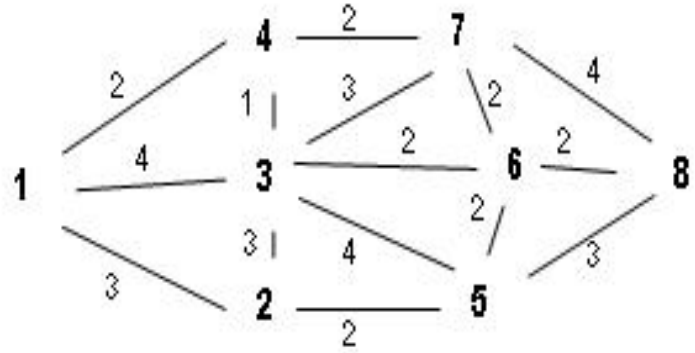
Hedef Düğüm		Yol Listesi	Süre
	1	14 – 8 – 3 – 1	33 ms
	2	14 – 10 – 7 – 2	35 ms
	3	14 – 8 – 3	24 ms
	4	14 – 13 – 11 – 4	33 ms
	5	14 – 13 – 11 – 5	29 ms
	6	14 – 10 – 7 – 6	22 ms
	7	14 – 10 – 7	15 ms
	8	14 – 8	8 ms
	9	14 – 13 – 9	11 ms
	10	14 – 10	8 ms
	11	14 – 13 – 11	18 ms
	12	14 – 10 – 12	16 ms
	13	14 – 13	4 ms
	14		0 ms

Çizelge değerlerinde her bir satır bir düğümü ifade etmektedir. Bu satırlar seçilen kaynak düğüm ile o satırdaki düğüm arasındaki en kısa yol bilgisini ve süre bilgisini gösterir.

6.5. Dijkstra Algoritması ile Kendini Klonlayan Karınca Kolonisi Yaklaşımının Karşılaştırılması

Bu bölümde, iki farklı en kısa yol algoritmasının aynı ağ topolojisi üzerinde uygulaması gerçekleştirilmektedir. Bu algoritmalar Dijkstra algoritması ve kendini klonlayan karınca kolonisi yaklaşımıdır.

Simülasyon için kullanılmak üzere basit bir ağ topoloji yaratılmıştır. Bu ağ topoloji 8 düğümden ve 15 çift yönlü hattan oluşmaktadır. Ağ topolojisi ve maliyet değerleri aşağıda Şekil 6.10‘de gösterilmektedir.



Şekil 6.10. Basit ağ topolojisi ve maliyet değerleri

Simülasyonda kaynak düğüm olarak düğüm 1 ve hedef düğüm olarak düğüm 8 seçilmektedir. Dijkstra algoritması ve kendini klonlayan karınca kolonisi yaklaşımı ayrı ayrı bu topoloji üzerinde uygulanmıştır.

Simülasyonda Dijkstra algoritmasının çalıştırılabilmesi için önceden yol maliyet değerlerinin girilmesi gerekir. Bu nedenle simülasyonun başlangıcında bu değerler girilmiştir. Kendini klonlayan karınca kolonisi yaklaşımının çalıştırılması için yol bilgisi değerlerinin girilmesine gerek yoktur.

Simülasyonun sonunda her iki yöntem iki düğüm arasındaki optimal yolu hesaplamaktadırlar. Düğüm 8 için optimal yol listesi ***1 – 4 – 3 – 6 – 8*** olmuştur. Geçen süre ise 7 ms.' dir. Aşağıdaki Çizelgede her düğüm için hesaplanan optimal yol bilgileri verilmektedir. Her iki yöntem aynı değerleri elde etmemizi sağlamıştır.

Çizelge 6.4. Düğüm 1 için yönlendirme çizelgesi

Kaynak Düğüm	**Hedef Düğüm**	**Yol Listesi**	**Süre**
Düğüm 1	Düğüm 2	1–2	3 ms
Düğüm 1	Düğüm 3	1–4–3	3 ms
Düğüm 1	Düğüm 4	1–4	2 ms
Düğüm 1	Düğüm 5	1–2–5	5 ms
Düğüm 1	Düğüm 6	1–4–3–6	5 ms
Düğüm 1	Düğüm 7	1–4–7	4 ms
Düğüm 1	Düğüm 8	1–4–3–6–8	7 ms

Simülasyon sürecinde klon karıncalar kendilerini klonlamakta ya da kendilerini öldürmektedir. Aşağıdaki Çizelge 6.5'de zaman içerisinde klonlanan ve ölen karınca sayıları gösterilmektedir.

Çizelge 6.5. Her ayrık zamanda klonlanan ve ölen karıncalar – düğüm 1 için

	Klonlanan Karınca	**Ölen Karınca**	**Toplam Klon Karınca**
0. ms	3	0	3
2. ms	1	0	4
3. ms	5	0	9
4. ms	2	1	10
5. ms	4	0	14
6. ms	0	4	10
7. ms	0	7	3
8. ms	0	2	1
9. ms	0	1	0

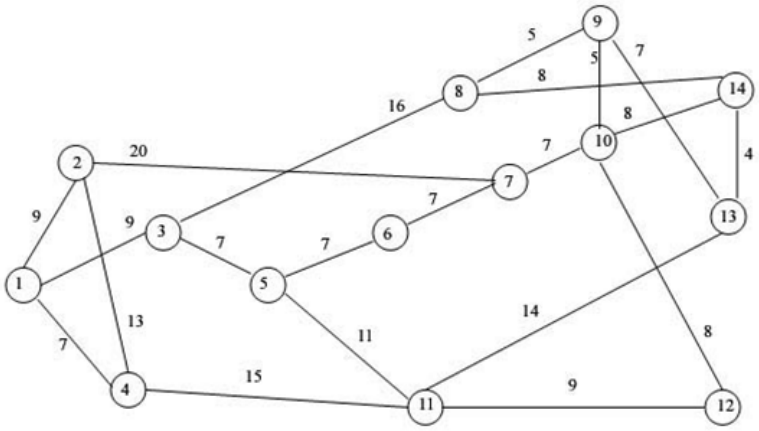
Çizelge 6.5'e bakıldığı zaman 5. ms.'de toplam klon karıncanın 14 olduğu görülüyor. Daha sonra zaman içerisinde bu klon karıncalar kendilerini yok etmektedirler. 9. ms.'de ise topoloji üzerinde klon karınca kalmamıştır.

6.6. Ağ Topolojisinin Değişmesi Durumunda Optimal Yolun Yeniden Bulunması

Geniş ağ topolojileri içerisinde pek çok düğüm ve bağlaç bulunmaktadır. Zaman içerisinde topolojiler üzerinde dinamik değişimler de gerçekleşebilmektedir. Örneğin ağ içinde yer alan düğümler, yönlendirici olarak isimlendirilen elektronik cihazlardır ve bu cihazlar birtakım sorunlardan dolayı bozulabilir ya da işlem yapamayabilir. Aynı şekilde düğümleri birbirine bağlayan hatlarda da bazı sorunlar gerçekleşebilir ve hatlar işlevlerini yerine getiremeyebilirler. Bu gibi durumlarda topoloji üzerinde birtakım değişimler kaçınılmaz olmaktadır. Düğümler, kendi yönlendirme tablolarını yeniden düzenleyerek bu yeni duruma adapte olmalıdırlar.

Bu bölümde gerçekleştirilen simülasyon uygulamasında örnek bir ağ topolojisi yaratılmakta ve kullanılmaktadır. Simülasyonun ilk aşamasında topoloji içerisinde yer alan tüm düğümler ve hatlar doğru şekilde çalışmaktadır. Bu durumda düğümler kendi yönlendirme tablolarını kendini klonlayan karınca kolonisi yaklaşımını kullanarak düzenlemektedirler. Simülasyonun ikinci aşamasında ise rasgele bir düğüm kapatılmaktadır. Bu durumda o düğüm işlem yapamaz hale gelir. Simülasyon içindeki düğümler bu yeni duruma karşı kendi yönlendirme tablolarını yeniden düzenlemek için klon karıncaları kullanırlar. Simülasyon sonucunda düğümler yeni duruma uygun güncel yönlendirme tablolarını oluşturmuş olmaktadırlar.

Simülasyon uygulamasında NSFNET ağ topolojisi kullanılmaktadır. Aşağıdaki Şekil 6.11'de ağ topolojisi ve maliyet değerleri gösterilmektedir.



Şekil 6.11. NFSNET topolojisi ve maliyet değerleri

Simülasyonun ilk aşamasında ağ topolojisi içindeki düğümler ve hatlar sorunsuz şekilde çalışmaktadır. Bu durumda rasgele olarak seçilen örnek düğümlerin yönlendirme tabloları aşağıdaki Çizelge 6.6'da gösterilmektedir.

Çizelge 6.6. Örnek düğümlerin yönlendirme tabloları

Kaynak Düğüm 1

Hedef Düğüm		Yol Listesi	Süre
	1	1	0 ms
	2	1–2	9 ms
	3	1–3	9 ms
	4	1–4	7 ms
	5	1–3–5	16 ms
	6	1–3–5–6	23 ms
	7	1–2–7	29 ms
	8	1–3–8	25 ms
	9	1–3–8–9	30 ms
	10	1–3–8–9–10	35 ms
	11	1–4–11	22 ms
	12	1–4–11–12	31 ms
	13	1–4–11–13	36 ms
	14	1–3–8–14	33 ms

Kaynak Düğüm 6

Hedef Düğüm		Yol Listesi	Süre
	1	6–5–3–1	24 ms
	2	6–7–2	27 ms
	3	6–5–3	14 ms
	4	6–5–3–1–4	30 ms
	5	6–5	7 ms
	6	6	0 ms
	7	6–7	7 ms
	8	6–7–10–9–8	24 ms
	9	6–7–10–9	19 ms
	10	6–7–10	14 ms
	11	6–5–11	18 ms
	12	6–7–10–12	22 ms
	13	6–7–10–9–13	26 ms
	14	6–7–10–14	22 ms

Kaynak Düğüm 9

Hedef Düğüm		Yol Listesi	Süre
	1	9–8–3–1	30 ms
	2	9–10–7–2	32 ms
	3	9–8–3	21 ms
	4	9–13–11–4	36 ms
	5	9–10–7–6–5	26 ms
	6	9–10–7–6	19 ms
	7	9–10–7	12 ms
	8	9–8	5 ms
	9	9	0 ms
	10	9–10	5 ms
	11	9–13–11	21 ms
	12	9–10–12	13 ms
	13	9–13	7 ms
	14	9–13–14	11 ms

Kaynak Düğüm 12

Hedef Düğüm		Yol Listesi	Süre
	1	12 – 11 – 4 – 1	31 ms
	2	12 – 10 – 7 – 2	35 ms
	3	12 – 11 – 5 – 3	27 ms
	4	12 – 11 – 4	24 ms
	5	12 – 11 – 5	20 ms
	6	12 – 10 – 7 – 6	22 ms
	7	12 – 10 – 7	15 ms
	8	12 – 10 – 9 – 8	18 ms
	9	12 – 10 – 9	13 ms
	10	12 – 10	8 ms
	11	12 – 11	9 ms
	12	12	0 ms
	13	12 – 10 – 9 – 13	20 ms
	14	12 – 10 – 14	16 ms

Simülasyonun ikinci aşamasında topoloji içerisinde yer alan 8 numaralı düğüm kapatılmaktadır. Bu durumda 1 numaralı düğümden 14 numaralı düğüme gönderilen paket hedef düğüme ulaşamamaktadır. Paket kaynak düğüme geri döndüğü zaman, kaynak düğüm yol üzerindeki düğümde ya da hatta sorun olduğunu düşünecektir. Bu

durumda ağ değişimleri nedeniyle yönlendirme tablosunu yeniden düzenlemek için bir klon karınca yaratarak karıncayı ağın içerisine doğru gönderecektir.

Simülasyonun ikinci aşamasının sonunda, düğümler kendi yönlendirme tablolarını düzenlemektedirler. Aşağıda Çizelge 6.7'de seçilen düğümlerin yeniden düzenlenmiş yönlendirme tabloları gösterilmektedir.

Çizelge 6.7. Örnek düğümlerin yönlendirme tabloları

Kaynak Düğüm 1

Hedef Düğüm		Yol Listesi	Süre
	1	1	0 ms
	2	1 – 2	9 ms
	3	1 – 3	9 ms
	4	1 – 4	7 ms
	5	1 – 3 – 5	16 ms
	6	1 – 3 – 5 – 6	23 ms
	7	1 – 2 – 7	29 ms
	8	-	-
	9	1 – 2 – 7 – 10 – 9	41 ms
	10	1 – 2 – 7 – 10	36 ms
	11	1 – 4 – 11	22 ms
	12	1 – 4 – 11 – 12	31 ms
	13	1 – 4 – 11 – 13	36 ms
	14	1 – 4 – 11 – 13 – 14	40 ms

Kaynak Düğüm 6

Hedef Düğüm		Yol Listesi	Süre
	1	6 – 5 – 3 – 1	23 ms
	2	6 – 7 – 2	27 ms
	3	6 – 5 – 3	14 ms
	4	6 – 5 – 3 – 1 – 4	30 ms
	5	6 – 5	7 ms
	6	6	0 ms
	7	6 – 7	7 ms
	8	-	-
	9	6 – 7 – 10 – 9	19 ms
	10	6 – 7 – 10	14 ms
	11	6 – 5 – 11	18 ms
	12	6 – 7 – 10 – 12	22 ms
	13	6 – 7 – 10 – 9 – 13	26 ms
	14	6 – 7 – 10 – 14	22 ms

Kaynak Düğüm 9

Hedef Düğüm		Yol Listesi	Süre
	1	9 – 10 – 7 – 2 – 1	41 ms
	2	9 – 10 – 7 – 2	32 ms
	3	9 – 10 – 7 – 6 – 5 – 3	33 ms
	4	9 – 13 – 11 – 4	36 ms
	5	9 – 10 – 7 – 6 – 5	26 ms
	6	9 – 10 – 7 – 6	19 ms
	7	9 – 10 – 7	12 ms
	8	-	-
	9	9	0 ms
	10	9 – 10	5 ms
	11	9 – 13 – 11	21 ms
	12	9 – 10 – 12	13 ms
	13	9 – 13	7 ms
	14	9 – 13 – 14	11 ms

Kaynak Düğüm 12

Hedef Düğüm		Yol Listesi	Süre
	1	12 – 11 – 4 – 1	31 ms
	2	12 – 10 – 7 – 2	35 ms
	3	12 – 11 – 5 – 3	27 ms
	4	12 – 11 – 4	24 ms
	5	12 – 11 – 5	20 ms
	6	12 – 10 – 7 – 6	22 ms
	7	12 – 10 – 7	15 ms
	8	-	-
	9	12 – 10 – 9	13 ms
	10	12 – 10	8 ms
	11	12 – 11	9 ms
	12	12	0 ms
	13	12 – 10 – 9 – 13	20 ms
	14	12 – 10 – 14	16 ms

Yukarıdaki Çizelge 6.7.'ye bakıldığı zaman önceki yol listesinde düğüm 8'i tutan satırlar yeni durumda güncellenip düzenlenmişlerdir. Düğüm 8'in kullanılmadığı ve hedefe varan alternatif yeni yol listeleri oluşturulmuştur. Örneğin, kaynak düğüm 1'den 14 numaralı düğüme gidebilmesi için 1 – 3 – 8 – 14 numaralı yolu kullanması gerekirken 8 numaralı düğümün kapatılması sonucu oluşan yeni durumda yol bilgisi 1 – 4 – 11 – 13 – 14 olarak güncellenmiştir. Bu durumda kendini klonlayan karınca kolonisi yaklaşımı kullanılarak ağ içinde oluşan değişimlere göre düğümlerin yol listeleri tabloları güncellenip düzenlenebilmektedir.

BÖLÜM 7.

SONUÇLAR

Bu tez çalışmasında bilgisayar ağları alanında önemli bir yer teşkil eden yönlendirme algoritmaları ve yönlendirme protokolleri açıklanmaktadır. Bu açıdan konu ele alındığı zaman geçmişten günümüze uzanan eski ve yeni yöntemler üzerinde durulmuştur.

Yönlendirme problemlerinin ele alındığı başlangıç durumlarında ağın statik bir yapı içerisinde olduğu düşünülmekteydi. Bu varsayıma dayanan algoritmalar yönlendirme kararını almaktaydılar. Zaman içerisinde, her an oluşabilecek değişimlerde de uygun yönlendirme kararı almaya çalışan algoritmalar gelişmektedir. Bu açıdan bakıldığı zaman doğal hayat içerisinde bulunan canlılardan esinlenerek dinamik yönlendirme algoritmaları veya protokolleri ortaya konmaktadır.

Yönlendirme algoritmaları için kullanılabilecek diğer bir yaklaşım Karınca Koloni Sistemi'dir. Karınca koloni sistemi içerisinde karınca olarak ifade edilen ajan yapılar sayesinde ağ üzerindeki değişimler takip edilmekte ve yönlendirme kararı alınmaktadır. Sezgisel bir yöntem olan karınca koloni sistemi süreç içerisinde en uygun olan çözüm yolunu vermektedir. Bu yöntemin bilgisayar ağlarına uygulanması ise AntNet algoritmasının geliştirilmesi ile mümkün olmuştur.

Tez çalışmasında karınca kavramı kullanılarak yeni bir yöntem ortaya konmaktadır. Bu yöntem içerisinde iki önemli kavram ortaya konmaktadır. Bu kavramlar, karınca olarak ifade edilen ajan yapısı ve klonlama mantığıdır. Bu iki yapı bu yaklaşım içerisinde bir araya getirilmiştir. Karınca olarak ifade edilen ajanlar kendi yapıları içerisinde klonlama kararını alabilmektedir. Bu şekilde kendileri ile aynı özelliklerde yeni klon karıncalar ortaya çıkarmaktadırlar. Bu karıncalar ağ içerisinde kendi kararlarını da

almaktadırlar ve karşılaştıkları durumlara göre ya kendilerini klonlayarak çoğaltmakta ya kendilerini yok etmekte ya da kaynağa geri dönmektedirler. Uygulanan topoloji içinde eğer ismi belirtilen bir düğüm varsa kesinlikle bu düğüme ulaşılacaktır ve o düğüm için en uygun yol bilgisi elde edilecektir. Eğer bu düğüm ismi kullanılan topoloji içinde yer almıyorsa bu durumda süreç tüm karıncaların yok olması ile sona erecektir.

Yaklaşım farklı ağ topolojileri üzerinde uygulanmıştır ve sonuçlar iki nokta arasındaki en uygun yolun bulunabildiğini göstermektedir. Ortaya konan yaklaşım ile çok eskiden beri kullanılan Dijkstra en kısa yol algoritmasının karşılaştırılması yapılmıştır ve ikisinin de sabit bir ağ yapısı içerisinde hedef düğümler için aynı değerleri verdiği görülmüştür. Fakat ağ yapısı dinamik olarak değişimlere açık olduğu zaman Dijkstra algoritmasının yeniden en kısa yolu belirleyebilmesi için güncel bilgileri elde etmesi gerekir. Bu durum ise zaman alacaktır.

Ağ içerisinde trafik, tıkanıklık, ağ cihazlarının bozulması, yazılım problemleri gibi birtakım sorunlar olabilir. Bu gibi sorunlarda dinamik olarak en uygun yolun bulunması ortaya konulan yaklaşım ile gerçekleştirilmektedir.

Tez çalışmasında sunulan bu yaklaşım ileriki zamanlarda üzerinde yoğun çalışmaların olduğu kablosuz sensör ağlarda uygulanabilir. Sensör düğümler üzerinden en uygun yolun bulunarak kullanılması ve bu süreç içerisinde de enerji verimliliğinin sağlanması üzerinde durulması gereken önemli fırsatlar olabilir.

KAYNAKLAR

1. Admane, L., Benatchba, K., Koudil, M., Drias, H., Gharout, S., Hamani, N., 2004, "Using Ant Colonies to Solve Data-Mining Problems", IEEE International Conference on Systems , Man, and Cybernetics, s. s. 3151 – 3157.
2. Aktuğ, S., 2007, http://www3.itu.edu.tr/~oktug/BH/notlar/bolum2.pdf, Erişim tarihi: 23.09.2007
3. Alaykıran, K., Engin, O., 2005, "Karınca Kolonileri Metasezgiseli ve Gezgin Satıcı Problemleri Üzerinde bir Uygulaması", Gazi Üniversitesi Müh. Mim. Fak. Dergisi, Cilt 20, No.1, s.s. 69-76.
4. Binghamton, http://www.cs.binghamton.edu/~dima/cs333/floyd.ppt, 31.08.2006
5. Baran, Benjamin ve Sosa, Ruben, 2000, "A New Approach for AntNet Routing", International Conference on Computer Communication and Networks IEEE ICCCN-2000, Las Vegas - Estados Unidos.
6. Barbehenn, M., 1998, "A Note on the Complexity of Dijkstra's Algorithm for Graphs with Weighted Vertices", IEEE Transactions on Computers, vol. 47, No: 2.
7. Black, U., 2000, "IP Routing Protocols", Prentice-Hall.
8. Bonabeau, E., Dorigo, M., Theraulaz, G., 2000, "Inspiration for optimization from social insect behavior," *Nature*, vol. 406, s.s. 39–42.
9. Boruvka, O., 1926, "English *On a certain minimal problem*",
10. Charter, S., MARK, A., 1999, "IP Routing Fundamentals", Indianapolis: Cisco Press.
11. Chen, Ming-Syan, Shin, Kang G., 1990, "Depth-First Search Approach for Fault Tolerant Routing in Hypercube Multicomputers", IEEE Transactions on Parallel and Distributed Systems, Vol.1, No:2, ss. 152-159.
12. Çölkesen, R., 2000 "Veri Yapıları ve Algoritmalar", Papatya Yayıncılık, s.s. 346.
13. Çölkesen, R., 2003,"Veri Yapıları ve Algoritmalar", Papatya Yayıncılık, s.s. 348-349.
14. Cisco, 2001, "Internetworking Technologies Handbook", Cisco Pres, Bölüm 5.2.

15. Caro, G., D., Dorigo, M., 1998, "AntNet: Distributed Stigmergetic Control for Communications Network", Journal of Artificial Intelligence Research, s.s. 317 – 365.
16. Dai, Quing, Wu, Jie, 2003, "Computation of Minimal Uniform Transmission Power in Ad HocWireless Networks", Proceedings of the 23 rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), ABD.
17. Dai, Shijin, Jing, Xiaorong, Li, Lemin, 2005, "Research and analysis on routing protocols for wireless sensor networks", The Proceedings of the International Conference on Communications, Circuits and Systems, s.s. 407-411.
18. Darkridge, http://www.darkridge.com/~jpr5/archive/alg/node88.html, 01.09.2006.
19. Das, Sajal K., Chen, Calvin C. Y., 1992 "A new parallel algorithm for breadth-first search on interval graphs", Proceedings of the Sixth International Parallel Processing Symposium, ABD, s.s. 150-153.
20. Data network, 2007, www.rhyshaden.com/isis.htm, Erişim tarihi: 07.08.2007
21. Deneubourg, J. L., Aron, S., Goss, S., Pasteels, J., M., 1990, "The self-organizing exploratory pattern of the argentine ant," *J. Insect Behav.*, vol. 3, s.s. 159–168.
22. Dijkstra, E., W., 1959, "A Note on Two Problems in Connexion with Graphs", Numerische Mathematik, vol. 1, pp. 269–271.
23. Dixon, Clare, 1996, "Temporal resolution: a breadth-first search approach", TIME '96: Proceedings of the 3rd Workshop on Temporal Representation and Reasoning (TIME'96), IEEE Computer Society, ABD.
24. Doi, S., Yamamura, M., 2005, "Congestion Detection and Clearing History of Trip Time in AntNet", Evolutionary Computation, IEEE Congress, s.s. 1621 – 1628.
25. Dorigo, M., Caro, G. D., Gambardella, L. M., 1999, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, s.s. 137–172.
26. Dorigo, M., Blum, C., 2005, "Ant Colony Optimization theory: A Survey", Theoretical Computer Science, Elsevier, s.s. 243 – 278.
27. Dorigo, M., Maniezzo, V., Colorni, A., 1996, "Ant System: Optimization by a Colony of Cooperating Agents", IEEE Transactions on Systems, Man and Cybernetics-Part-B, vol. 26, no.1, s.s. 29–41.

28. Dorigo, M., 1992, “Optimization, Learning and Natural Algorithms”, Doktora Tezi, Dipartimento di Elettronica e Informazione, Politecnico di Milano, İtalya.
29. Ducatelle, F., Caro, G., D., Gambardella, L., M., 2005, “Ant Agents for Hybrid Multipath Routing in Mobile Ad Hoc Networks”, Proceedings of the Second Annual Conference on Wireless On-demand Network Systems and Services (WONS’05).
30. Edward, A. T., 1979, “Gateway Information Protocol”, Xerox Parc, Palo Alto.
31. Eel, 2006, http://lcm.csa.iisc.ernet.in/dsa/node183.html, Erişim tarihi: 14.12.2006
32. Erdogan, S. Z., Esin, E., M., 2006, "An Application of Two Different Algorithms on the Same Network", Proceedings of the International Symposium on Communucations and Information Technologies 2006 (ISCIT2006), Tayland.
33. Erdogan, S., Z., Esin, E., M., 2006, "Routing Table Updating By Using Self Cloning Ant Colony Approach", Proceedings of the 5th International Symposium On Intelligent Manufacturing Systems, IMS2006, Sakarya, Türkiye.
34. Esin, E., M., Erdogan, S., Z., 2006, “Self Cloning Ant Colony Approach and Optimal Path Finding”, Proceedings of the Euro American Conference on Telematics and Information Systems, EATIS 2006, Kolombiya, ISBN: 958-8166-38-1
35. Floyd, R. W., 1967, “Non-Deterministic Algorithms”, J. Assoc. Computing, s.s. 636 – 644.
36. Fujita, Yusuke, Nakamura, Yoshihiko, Shiller, Zvi, 2003, “Dual Dijkstra Search for paths with different topologies”, Proceedings of the International Conference on Robotics and Automation ICRA '03, IEEE, Taywan.
37. Gambardelle, L., M., Dorigo, M., 1995, “Ant-Q: A Reinforcement Learning Approach to the Travelling Salesman Problem”, In Proceedings of the Eleventh International Conference on Machine Learning, Morgan Kaufman, s.s. 252-260.
38. Garcia, Nuno M.; Lenkiewicz, Przemyslaw; Freire, Mario M.; Monteiro, Paulo P., 2007, “On the Performance of Shortest Path Routing Algorithms for Modeling and Simulation of Static Source Routed Networks -- an Extension to the Dijkstra Algorithm”, Second International Conference on Systems and Networks Communications ICSNC 2007, Fransa.
39. Grimaldi, Ralph P., 2003, “Discrete and Combinatorial Mathematics”, Addison Wesley, 5.Baskı

40. Goss, S., Aron S., Deneubourg J. L., and Pasteels J. M., 1989, "Self-organized Shortcuts in the Argentine Ant", Naturwissenschaften, 76:579-581.
41. Hussain, O., Saadawi, T., 2003, "Ant routing algorithm for mobile ad-hoc networks", Performance, Computing and Communications Conference Proceedings of the 2003 IEEE International, s.s. 281 – 290.
42. Ibrahim, Muhammed A.M., Xinda, Lu, Rwakarambi, J.M., 2001, "Parallel Execution of An Irregular Algorithm Depth First Search (DFS) on Heterogeneous Clusters of Workstation", The Proceedings of International Conferences on Info-tech and Info-net ICII 2001 – Beijing, Çin.
43. Is-is, 1992, RFC 1195.
44. Jiang, Wei-Jin, Xu, Yu-Hui, Xu, Yu-Sheng, 2005, "A Novel Data Mining Algorithm Based On Ant Colony System", Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, s.s. 1919 – 1923.
45. Jin, Xin, Wang, Hongbo, Zhang, Yaoxue, Chang, Dong, 2003, "A Dynamic Self-adaptive Routing Update Algorithm for MANET", International Conference on Communication Technology Proceedings ICCT 2003.
46. Johnsonbaugh, Richard, 2005, "Discrete Mathematics", Pearson Prentice Hall Publisher, 6. Baskı.
47. Kruse, Robert L, Ryba, Alexander J., 1998, "Data Structures and Program Design, Prentice Hall", s.s. 577.
48. Kruse, R. L., Ryba, A. J., 2000, "Data Structures and Program Design", Prentice Hall, s.s. 581- 582.
49. Kruskal, J. B., 1956, "On the Shortest Spanning Subtree and the Traveling Salesman Problem", Proceedings of the American Mathematical Society, s.s. 48 – 50.
50. Larsson, T., Hedman, N., 1998, "Routing Protocols in Wireless Ad-Hoc Networks-A Simulation Study", Master Tezi, Lulea University of Technology.
51. Leith, Chris, Takahara Glen, 2003, "A Control Framework for Ant-Based Routing Algorithms", Evolutionary Computation, CEC '03.
52. Liang, S., Zincir-Heywood A. N., Heywood M. I., 2005, "Adding more intelligence to the network routing problem: AntNet and GA-agents", Journal of Applied Soft Computing, Elsevier.

53. Liu, Bing, Choo, Siew-Hwee, Lok, Shee-Ling, Leong, Sing-Meng, Lee, Soo-Chee, Poon, Foong-Ping, Tan, Hwee-Har, 1994, "Integrating case-based reasoning, knowledge-based approach and Dijkstra algorithm for route finding", Proceedings of the Tenth Conference on Artificial Intelligence for Applications, Texas, ABD.
54. Liu, Shang, Dou, Zhi-Tong, Li, Fei, Huang, Ya-Lou, 2004, "A New Ant Colony Clustering Algorithm Based on DBSCAN", Proceedings of the Third International Conference on Machine Leaming and Cybernetics, Shanghai, Çin.
55. Nabiyev, Vasfi V., 2007, "Algoritmalar Teoriden Uygulamalara", Seçkin Yayınevi, Ankara, Türkiye.
56. Ngo, Son-Hong, Jiang, Xiaohong, Horuguchi, Susumu, 2004, "Adaptive Routing and Wavelength Assignment Using Ant-Based Algorithm", IEEE International Conference on Networking (ICON2004), Singapur.
57. Noto, M., Sato, H., 2000, "A Method for the Shortest Path Search by Extended Dijkstra Algorithm", IEEE,
58. http://www.kirupa.com/developer/actionscript/depth_breadth_search4.htm, 08.09.2006.
59. Ospf NSSA Option, 1994, RFC 1587.
60. Quyang, Jun, YAN, Gui-Rong, 2004, "A Multi-Group Ant Colony System Algorithm for TSP", Proceedings of the 3. International Conference on Machine Learning and Cybernetics, Shanghai, 2004.
61. Parpinelli, R. S., Lopes, H., S., Freitas, A. A., 2002, "Data Mining With an Ant Colony Optimization Algorithm", IEEE Transactions on Evolutionary Computing, Vol. 6, No. 4, s. s. 321 – 332.
62. Perkins, Charles E., Royer, Elizabeth M., 1999, "Ad-Hoc On-Demand Distance Vector Routing", Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, s.s. 90 – 100, ABD.
63. Preiss, B. R., 2002, "Data Structures and Algorithms with Object-Oriented Design in Java", Waterloo Üniversitesi.
64. Prim, R. C., 1957, "Shortest connection networks and some generalizations", Bell System Technical Journal, s.s. 1389 – 1401.

65. Sakallı, Mustafa, Pearlman, W. A., 2006, “SPIHT Algorithms Using Depth First Search with Minimum Memory Usage”, Conference on Information Sciences and Systems, CIPR Technical Report TR-2006-10.
66. Sim, K., M., Sun, W., H., 2003, “Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions”, IEEE Transactions on Systems, Man and Cybernetics.
67. Sim, K., M., Sun, W., H., 2002, “Multiple Ant-Colony Optimization for Network Routing”, Proceedings of the First International Symposium on Cyber Worlds (CW’02).
68. Solka, Jeffrey L., Perry, James C., Poellinger, Brian R., Rogers, George W., 1992, “Autorouting using a parallel Dijkstra algorithm with embedded constraints”, International Joint Conference on Neural Networks IJCNN, Maryland, ABD.
69. Stojmenovic, Ivan, Russell, Mark, Vukojevic, Bosko, 2000, “Depth First Search and Location Based Localized Routing and QoS Routing in Wireless Networks”, ICPP '00: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing, IEEE Computer Society, ABD.
70. Tanenbaum, A., S., 2002, “Computer Networks, Fourth Edition”, Prentice Hall, ISBN: 0130661023.
71. Microsoft, (http://www.microsoft.com/technet/prodtechnol/widows2000serv/reskit/intwork/inac_uni_szam.mspx?mfr=true, 23 eylül 2006)
72. Kirupa, http://www.kirupa.com/developer/actionscript/depth_breadth_search2.htm, 04 Ocak 2007.
73. Wang, Jialei; Sun, Ying; Liu, Zuojun; Yang, Peng; Lin, Tao, 2007, “Route Planning based on Floyd Algorithm for Intelligence Transportation System”, IEEE International Conference on Integration Technology, ICIT '07, s.s. 544-546.
74. Yang, Y., Zincir-Heywood A. N., Heywood M. I., Srinivas S., 2002, “Agent Based Routing Algorithms on a LAN”, Proceedings of the IEEE Canadian Conference of Electrical and Computer Engineering, CCECE, ISBN 0-7803-7514-9, pp. 1442-1447, Winnipeg, Kanada.
75. Yun, Liu, 2002, “The application of breadth first search in the dispatching system of the China Railway Communication network”, The Proceedings of the IEEE Region

10 Conference on Computers, Communications, Control and Power Engineering TENCON '02.

TEZ SIRASINDA YAPILAN ÇALIŞMALAR

Uluslararası Hakemli Dergide Yayınlanan Makaleler

1. Erdoğan, Şenol Zafer, Esin, E. Murat, “Using Self Cloning Ant Colony Approach for Updating Routing Tables”, Journal of Intelligent Manufacturing, Springer, Kabul edildi.

Uluslararası Kongre ve Sempozyum Bildirileri

2. Erdoğan, S. Z., E. ,Murat Esin, 2006, "An Application of Two Different Algorithms on the Same Network", Proceedings of the International Symposium on Communucations and Information Technologies 2006 (ISCIT2006), Tayland (Bangkok), 18 - 20 October 2006.

3. Erdoğan, S. Z., E., Murat Esin, 2006, "Routing Table Updating By Using Self Cloning Ant Colony Approach", Proceedings of The 5th International Symposium On Intelligent Manufacturing Systems, IMS2006, Sakarya, Türkiye.

4. E., Murat Esin, Erdoğan, S. Z., 2006 , "Self Cloning Ant Colony Approach and Optimal Path Finding", Proceedings of The Euro American Conference on Telematics and Information Systems, EATIS 2006, Kolombiya, ISBN: 958-8166-38-1

ÖZGEÇMİŞ

Şenol Zafer ERDOĞAN, 06 Ekim 1978 yılında İstanbul'da doğdu. Lise öğrenimini Kabataş Erkek Lisesi'nde tamamladıktan sonra 1997 yılında Trakya Üniversitesi Mühendislik Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü'nü kazandı. Bu bölümden 2001 yılında mezun oldu. 2001 yılı Ekim ayında Maltepe Üniversitesi Mühendislik Fakültesi'nde Araştırma Görevlisi olarak çalışmaya başladı. Aynı sene İstanbul Üniversitesi Sayısal Yöntemler Bilim Dalı'nda yüksek lisans çalışmalarına başladı. 2004 yılında yüksek lisansı (bir yıl bilimsel hazırlık) başarıyla tamamladı. 2004 yılından itibaren doktora çalışmalarına başladı. 2006 yılında Maltepe Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde Öğretim Görevlisi kadrosuna atandı. Şenol Zafer ERDOĞAN, 2006 yılından beri Öğretim Görevlisi olarak görevine devam etmektedir.