EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(DOKTORA TEZİ)

TELSİZ DUYARGA AĞLARINDA DAĞITIK UYGULAMALAR İÇİN KÜME VE OMURGA TABANLI İLETİŞİM MİMARİLERİ

Orhan DAĞDEVİREN

Tez Danışmanı: Prof. Dr. Kayhan ERCİYEŞ

Uluslararası Bilgisayar Anabilim Dalı

Bilim Dalı Kodu: 619.02.04

Sunuş Tarihi: ……

Bornova-İZMİR

2010

Orhan DAĞDEVİREN tarafından DOKTORA TEZİ olarak sunulan "Telsiz Duyarga Ağlarında Dağıtık Uygulamalar için Küme ve Omurga Tabanlı İletişim Mimarileri" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve …… tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:	**İmza**
Jüri Başkanı :	**.....**
Raportör Üye :	**.....**
Üye :	**.....**
Üye :	**.....**
Üye :	**.....**

ÖZET

TELSİZ DUYARGA AĞLARINDA DAĞITIK UYGULAMALAR İÇİN KÜME VE OMURGA TABANLI İLETİŞİM MİMARİLERİ

DAĞDEVİREN, Orhan

Doktora Tezi, Uluslararası Bilgisayar Enstitüsü
Tez Yöneticisi: Prof. Dr. Kayhan ERCİYEŞ
Temmuz 2010, 218 sayfa

Telsiz duyarga ağı (TDA) çevreden algılama yapabilen ve kablosuz haberleşebilen düğümlerin sabit bir altyapı olmadan oluşturdukları bir ağdır. Habitat gözlemleme, askeri gözetim ve hedef takip TDAyla yapabilecek örnek uygulamalardır. Kümeleme ve omurga oluşturma uygulama paketlerinin etkin olarak yönetilmesi için kullanılan yöntemlerdir. Bu tezde, TDA için küme ve omurga tabanlı iletişim mimarileri çalışılmıştır.

Tezde ilk olarak güçlü iletişim kanallarını seçmeyi amaçlayan ağırlıklı çizge eşleme tabanlı dağıtık algoritmalar tasarlanmıştır. İkinci olarak enerjisi yüksek düğümleri seçmeyi amaçlayan merkezi ağırlıklı bağlı hâkim küme algoritmalarının dağıtık sürümleri tasarlanmıştır. Bu algoritmaların senkron ve asenkron sürümlerinin tasarımı verilmiştir. Son olarak sıradan düğümlerin az enerji harcamasını hedefleyen bir yaklaşım olarak, donanım etmeni tarafından merkezi işletilen konumlandırma ve kümeleme çerçevesi çalışılmıştır. Önerilen algoritmaların doğrulukları ve karmaşıklıkları analiz edilmiş, benzetimleri değişen parametrelere göre ve literatürdeki algoritmalarla karşılaştırılarak yapılmıştır. Ayrıca önerilen algoritmaların üzerine yapılabilecek geliştirmeler ve bu algoritmalara uygun örnek uygulamalar gösterilmiştir.

Anahtar sözcükler: Telsiz Duyarga Ağları, Kümeleme, Omurga Oluşturma, Çizge Eşleme, Bağlı Hâkim Küme, Donanım Etmeni.

ABSTRACT

CLUSTER AND BACKBONE BASED COMMUNICATION ARCHITECTURES FOR DISTRIBUTED APPLICATIONS IN WIRELESS SENSOR NETWORKS

DAĞDEVİREN, Orhan

Ph.D. in International Computing Institute
Supervisor: Prof. Dr. Kayhan ERCİYEŞ
July 2010, 218 pages

Wireless Sensor Network (WSN) is an infrastructureless network of nodes which are capable of sensing environment and communicating over wireless links. Habitat monitoring, military surveillance and target tracking are some application types of WSN. Clustering and backbone formation are methods to efficiently manage the application packets. In this thesis, cluster and backbone based communication architectures have been studied.

In this thesis, weighted graph matching based distributed algorithms which are aimed to select strong communication links have been studied firstly. Secondly, the distributed versions of central weighted connected dominating set algorithms which are aimed to select nodes with high energy have been designed. The synchronous and asynchronous versions of the proposed algorithms have been given. Lastly the framework executed centrally by an hardware agent have been studied as an approach which targets very low energy consumption of ordinary nodes. The correctness and complexities of the proposed algorithms have been analyzed, the simulations of these algorithms have been made with respect to varying parameters and by comparing with the algorithms in literature. Also the possible improvements and the example suitable applications of the proposed algorithms have been studied.

Keywords: Wireless Sensor Networks, Clustering, Backbone Formation, Graph Matching, Connected Dominating Set, Hardware Agent.

TEŞEKKÜR

Öncelikle doktora çalışmalarımda bana rehberlik yapan, yol gösteren, çalışmayı yönlendiren tez danışmanım Sn. Prof. Dr. Kayhan ERCİYEŞ'e çok teşekkür ederim. Yüksek Lisans çalışmamdan başlayarak benimle bilgilerini, deneyimlerini paylaşıp araştırmalarıma değerli fikirleriyle yön vermesi sayesinde bu çalışma meydana gelmiştir.

Doktora çalışmam sürecinde tez izleme toplantılarında bana rehberlik edip çalışmanın yönlendirilmesi konusunda yardımları olan hocalarım Sn. Prof. Dr. Turhan TUNALI ile Sn. Doç. Dr. İbrahim KÖRPEOĞLU'na teşekkür ederim.

Doktora eğitimimi aldığım Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü'ndeki tüm hocalarıma her türlü katkıları, destekleri ve yardımları için teşekkür ederim.

Bu çalışma sırasında fikirlerini paylaştığım ve yardım aldığım Sn. Dr. Ayşegül ALAYBEYOĞLU, Sn. İlker KORKMAZ ve Sn. Fatih TEKBACAK'a teşekkür ederim.

Bu çalışma sürecinde bana destek olan ve yardımlarını esirgemeyen İzmir Yüksek Teknoloji Enstitüsü'ndeki tüm hocalarıma ve çalışma arkadaşlarıma teşekkür ederim.

"Yurtiçi Doktora Başarı Bursu Programı" ile doktora çalışmamı maddi olarak destekleyen Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK)'a teşekkür ederim.

Tüm yaşamım boyunca bana destek olan ve yanımdan hiç ayrılmayan aileme çok teşekkür ederim.

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Şekil Sayfa

KISALTMALAR DİZİNİ

ABHK: Ağırlıklı Bağlı Hâkim Küme

ADC: Analog to Digital Converter (Sayısaldan Örneksele dönüştürücü)

ADS: Agent based Dominating Set Algorithm (Etmen tabanlı Hâkim Küme Algoritması)

ALOC: Agent based Localization Algorithm (Etmen tabanlı Konumlandırma Algoritması)

AST: Agent based Minimum Spanning Tree Algorithm (Etmen tabanlı En Küçük Kapsayan Ağaç Algoritması)

ASYNSET: Semi-asynchronous Weighted Dominating Set Algorithm (Yarı-asenkron Ağırlıklı Hâkim Küme Algoritması)

ASYNTREE: Semi-asynchronous Weighted Steiner Tree Algorithm (Yarı-senkron Ağırlıklı Steiner Ağacı Algoritması)

B-MAC: Berkeley Medium Access Control (Berkeley Ortama Erişim Kontrolü)

BFA: Backbone Formation Algorithm (Omurga Oluşturma Algoritması)

BFS-DSTA: Breadth-First Search based Distributed Spanning Tree Algorithm (Enine-Öncelikli Arama tabanlı Dağıtık Kapsayan Ağaç Algoritması)

BDÇ: Birim Disk Çizgesi

BHK: Bağlı Hâkim Küme

BĞLAN: BAĞLAN

BKL: BEKLE

CENTSET: Central Set Cover based Dominating Set Algorithm (Küme Örtüsü tabanlı Merkezi Hâkim Küme Algoritması)

CV: Coefficient of Variation (Varyasyon Katsayısı)

DAC: Digital to Analog Converter (Örnekselden Sayısala dönüştürücü)

DRM: DURUM

DS: Dominating Set Algorithm (Hâkim Küme Algoritması)

DSTA: Distributed Spanning Tree Algorithm (Dağıtık Kapsayan Ağaç Algoritması)

FLOODSET: Flooding based Synchronous Weighted Dominating Set Algorithm (Yayma tabanlı Senkron Ağırlıklı Hâkim Küme Algoritması)

FLOODTREE: Flooding based Synchronous Steiner Tree Algorithm (Yayma tabanlı Senkron Steiner Ağacı Algoritması)

GDI: Great Duck Island (Büyük Ördek Adası)

GHS: Gallagher, Humblet ve Spira'nın Dağıtık Kapsayan Ağaç Algoritması

GPS: Global Positioning System (Evrensel Konumlandırma Sistemi)

HE: Hâkim Eleman

HELMAN: HÂKİM_ELEMAN

HK: Hâkim Küme

KMO: Kulak Misafiri Olma

KYNŞTR: KAYNAŞTIR

LDR: LİDER

IEEE: The Institute of Electrical and Electronics Engineers (Elektik ve Elektronik Mühendisleri Enstitüsü)

IT: Iterative Trilateration (Yenilemeli Trilaterasyon)

İST: İSTEK

M-MAC: Mobility-Based Link Management Protocol for Mobile Sensor Networks (Gezgin Duyarga Ağları için Gezginlik-Tabanlı Bağlantı Yönetim Protokolü)

MAC: Medium Access Control (Ortam Erişim Kontrolü)

MASC: Matching based Synchronous Clustering Algorithm (Eşleme tabanlı Senkron Kümeleme Algoritması)

MCUBA: Matching based Asynchronous Clustering and Backbone Formation Algorithm (Eşleme tabanlı Asenkron ve Kümeleme ve Omurga Oluşturma Algoritması)

MIS: Chaterjee'nin Büyükçe Bağımsız Küme Tabanlı Ağırlıklı Hâkim Küme Algoritması

MOD-MASC: Modified Matching based Synchronous Clustering Algorithm (Değiştirilmiş Eşleme tabanlı Senkron Kümeleme Algoritması)

MODSET: Modified Central Set Cover based Dominating Set Algorithm (Değiştirilmiş Küme Örtüsü tabanlı Merkezi Hâkim Küme Algoritması)

RNT: RAUNT

RSS: Received Signal Strength

S-MAC: Sensor Medium Access Control (Duyarga Ortama Erişim Kontrolü)

SELMAN: SIRADAN_ELEMAN

SSET: Synchronous Weighted Dominating Set Algorithm (Senkron Ağırlıklı Hâkim Küme Algoritması)

STREE: Synchronous Weighted Steiner Tree Algorithm (Senkron Ağırlıklı Steiner Ağacı Algoritması)

TDA: Telsiz Duyarga Ağı

TRAMA: The Traffic-Adaptive Medium Access Protocol (Trafik-Uyumlu Ortama Erişim Protokolü)

UART: Universal Asynchronous Receiver/Transmitter (Evrensel Asenkron Alıcı/Verici)

WANG: Wang'ın Ağırlıklı Bağlı Hâkim Küme Algoritmasının İlk Fazı

WANG(2nd): Wang'ın Ağırlıklı Bağlı Hâkim Küme Algoritmasının İkinci Fazı

WMEWMA: The Window Mean with Exponentially Weighted Moving Average (Pencere Ortasıyla Üstel Ağırlıklı Değişen Ortalama)

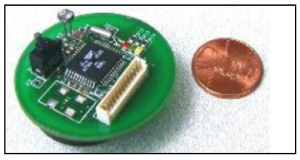
1. GİRİŞ

Bu bölümde öncelikle TDA kavramı ve uygulamaları teorik model ile birlikte anlatılacak, ikinci olarak kümeleme ve omurga oluşturma problemleri açıklanacak, bölümün devamında tezin katkıları ve tezin çerçevesi verilecektir.

1.1. Telsiz Duyarga Ağları

Kişisel bilgisayarların artık sıradan hale geldiği günümüzde, masaüstü bilgisayar ve sunucu mimarilerinden farklı tasarımlar ortaya çıkmaya başlamıştır. Bu tasarımlar daha küçük, daha ucuz, daha az güç kullanan özelliklere sahip donanımlardır. Bunun yanında sistemler artık bir entegre üzerinde, bütünleşik az güç kullanan haberleşme ünitesiyle birlikte tasarlanmaya başlanmışlardır. Bu tasarım teknikleri sayesinde bağlanmış duyargalar *(networked sensors*) kullanılmaya başlanmıştır. Temel mikroişlemci artık sadece bellek ve işlemciyi değil, sabit bellek, DAC, ADC, UART, kesici kontrolcüsü (*interrupt controller*) ve sayaç gibi arayüzleri barındırmaya başlamıştır. Kablolu, kısa mesafeli RF, kızılötesi, optik ve diğer haberleşme teknikleri kullanılmaya başlanmıştır. Duyargalar çevre ile iletişim kurarak ısı, ışık, hareket, pozisyon ve kimyasal oluşum gibi etkenleri tespit edip bu bilgileri haberleşme ile gönderebilir. Genel olarak bağlanmış duyargaların özelliklerini şu şekilde özetleyebiliriz:

- Ufak boyutlarda olma ve az güç harcama: Duyargalar kullanım kolaylığı açısından ufak boyutlarda olmalı, donanım ve yazılım olarak teknolojinin elverdiği ölçüde az güç harcayarak çalışmalıdır. Şekil 1.1'de örnek bir duyarga düğümü verilmiş ve madeni para ile boyut karşılaştırılması yapılmıştır.



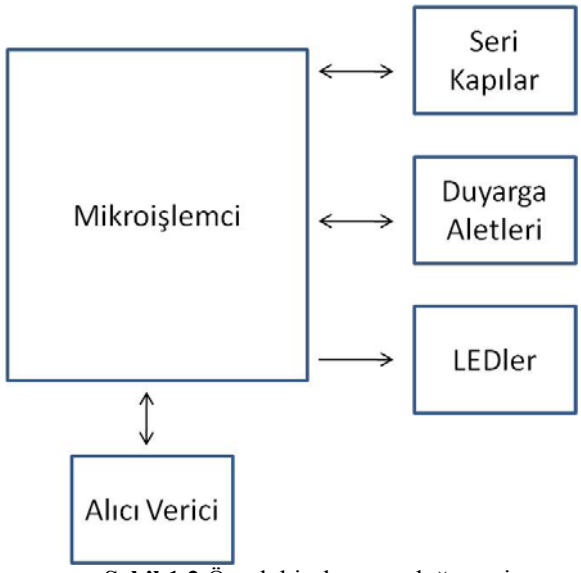
Şekil 1.1 Örnek bir duyarga düğümü

- Eşzamanlı çalışmaya uyumlu: Duyargalar aynı anda birden fazla iş yapabilmeli, örneğin çevreden veri toplanırken aynı anda başka veriyi işleyebilmelidir.

- Amaca yönelik tasarım: Duyargalar donanım olarak basit ve ihtiyaca yönelik olarak tasarlanmalıdır. Genel amaçlı donanımlardaki karmaşık kontrolcü hiyerarşileri kullanılmadan basit bir tasarım uygulanmalıdır.

Bu özellikleri verdikten sonra örnek bir duyarganın iç yapısını Şekil 1.2 'de görüyoruz. Mikroişlemciye bağlı olan duyarga aletleri, LEDler, düşük güç tüketimli

alıcı-verici (*transceiver*), duyarga aletleri ve seri kapılar (*port*) görülmektedir (Hill et al., 2000).

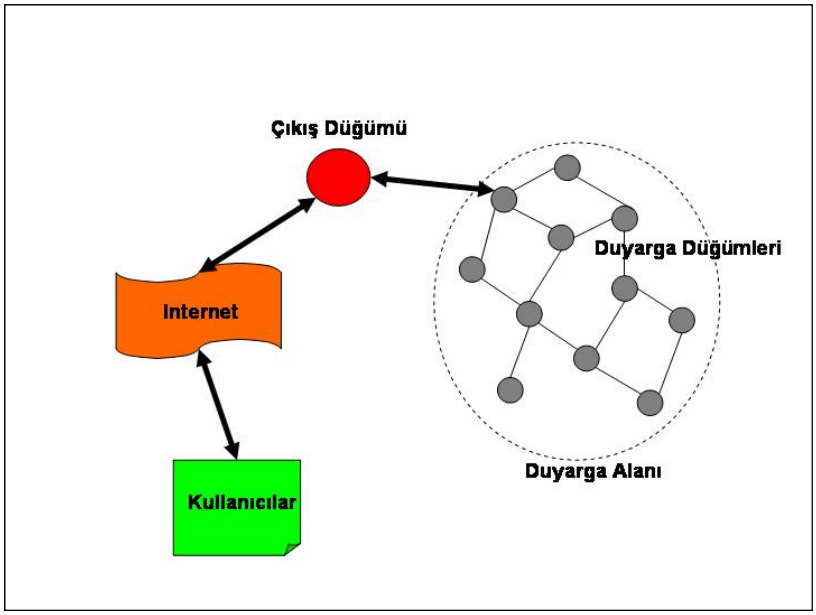


Şekil 1.2 Örnek bir duyarga düğümü iç yapısı

TDAlar çok sayıda duyarga düğümünün kendini yönetebileceği şekilde tasarsız bir yapıda kurulur. Örnek bir TDA yapısı Şekil 1.3'de verilmiştir. Bu şekilde görüldüğü üzere duyarga düğümlerinden herhangi birinin çevreden topladığı veri diğer düğümler üzerinden çok zıplamalı (*hop*) bir şekilde çıkış düğümüne (*sink node*) ulaşır. Çıkış düğümü, diğer düğümlerden gelen verilen toplandığı ve kullanıcıya yönlendirilmek üzere diğer ağ yapısına iletildiği ağ geçidir (*gateway*). Çıkış düğümü şekilde görüldüğü üzere Internet üzerinden verilerini gönderebileceği gibi uydu teknolojisi de kullanılabilir. Şekildeki çift taraflı oklardan da anlaşabileceği üzere bazı durumlarda kullanıcılar duyarga ağlarının ayarlarını değiştirmek için gerekli bilgileri gönderebilmektedir.

TDAların günümüzde birçok uygulama alanı vardır (Ning, 2005). En önemli kullanım alanı çevre gözlemleme (*habitat monitoring*) uygulamalarıdır. Great Duck Island (GDI) uygulamasında Great Duck Adasındaki Storm Petre isimli su kuşlarının yaşam döngüsü UCB ve Intel Araştırma laboratuarındaki araştırmacılar tarafından incelenmiştir (Mainwaring et al., 2002). Havai Üniversitesinde geliştirilen PODS (Biagioni and Bridges, 2002) araştırma projesi ile nesli tehdit olan bitki çeşitlerinin incelenmesi yapılmıştır. ALERT (http://www.alertsystems.org, 2010) sistemi sel tehlikesini tahminlemek için su seviyesi, ısı, rüzgâr duyargaları ile birlikte kurulmuş bir sistemdir. TDAların bir diğer kullanımı sağlık uygulamalarıdır. Loren ve arkadaşları (Schwiebert et al., 2001) mikro duyargaları kullanarak kör hastaların görmesini sağlayacak bir protez oluşturdular. Bunun yanında uzaktan hasta takibi ve ilaç kullanım yönetimi gibi konularda da uygulamalar vardır (Akyildiz et al., 2002). Ev ve ofis uygulamalarında da TDAlar kullanılır. Srivastava ve arkadaşlarının

(Srivastava et al., 2001) akıllı anaokulu projesiyle çocukların erken yaşta etkileşimli olarak eğitilmesi amaçlanmıştır.



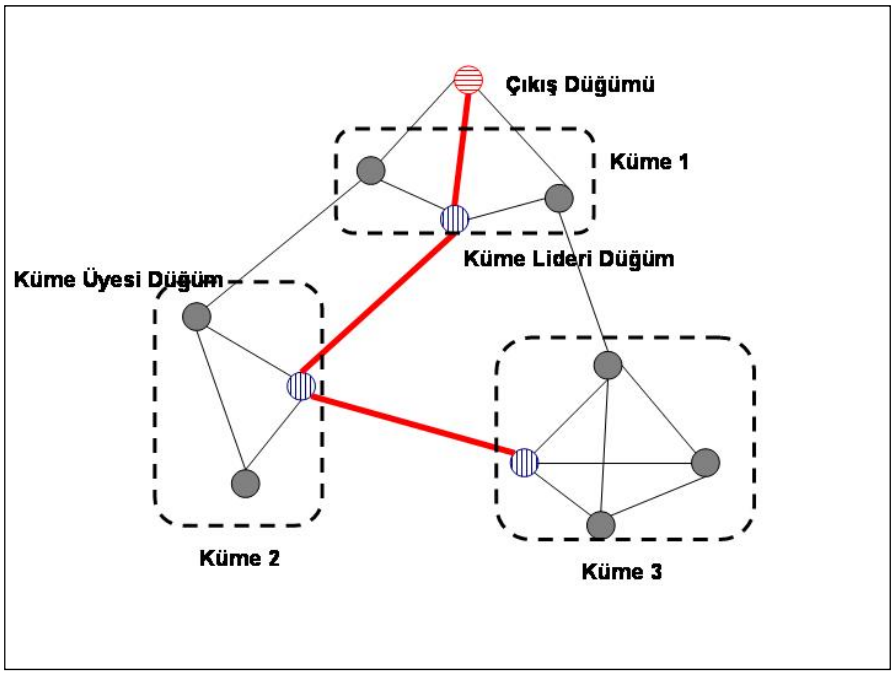
Şekil 1.3 Örnek bir Duyarga Ağı

1.2. Kümeleme ve Omurga Oluşturma Problemi

TDAlar üzerindeki en önemli problemlerden biri de uygulamalar için çevreden toplanan verilerin enerji etkili bir şekilde çıkış düğümüne gönderilmesini sağlayacak etkili bir ağ topolojisi oluşturmaktır. Bu şekilde bir ağ topolojisini oluşturmanın en önemli yöntemlerinden biri kümelemedir. Kümeleme en basit tanımıyla ağı parçalara bölüp her parçaya bir lider atama işlemidir. Kümeleme işlemi sayesinde TDA daha küçük parçalara bölünür ve problemlerimiz parçalanmış olur. TDA kümelenerek yönlendirme (*routing*), zaman senkronizasyonu (*time synchronization*), veri birleştirme (*data aggregation*), hata toleransı (*fault tolerance*), topoloji kontrolü (*topology control*), yük dengesinin (*load balancing*) sağlanması, ağın yaşam süresinin uzatılması ve güvenli iletişimin sağlanması işlemleri kolaylanmış olur.

Şekil 1.4'de kümeler kesik çizgiler ile gösterilmiş, TDA üç kümeye bölünmüştür. Küme üyesi düğümler içi gri dolu daireler ile küme lideri düğümlerse boyuna mavi çizgili daireler ile gösterilmiştir. Küme üyesi düğümler çevreden aldıkları verileri küme lideri düğüme gönderebilirler. Küme lideri düğümde toplanan verilerin çıkış düğümüne ulaşması için de bir yol tanımlanması gereklidir. Bu yolun adına omurga denir. Şekil 1.4'de kalın kırmızı çizgiler ile omurga gösterilmiştir. Çıkış düğümü kırmızı enine çizgileri olan ufak dairedir. Küme ve omurga yapılarıyla birlikte TDA üzerinde çıkış düğümüne doğru verilerin taşınması için etkin bir iletişim yapısı oluşturulur.

Kümeleme yaklaşımlarının amaçları çeşitlilik gösterir. Örneğin (Erciyes, et al. 2008)'deki algoritma ayrık kümeler oluşturmak için her düğümün sadece bir küme lideri olmasını sağlarken, (Youssef et al., 2007)'deki algoritma üst üste binen kümelerin hata toleransı, konumlandırma ve topoloji kontrolü için faydalı olacağını savunmuştur. (Cokuslu et al., 2006)'deki algoritma kümeleri ortada küme lideri olacak biçimde yıldız topolojisi olarak kurarken, (Erciyes et al. 2008)'deki algoritmada çok zıplamalı kümeler üretilmiştir. TDA için genel olarak kümeleme ve omurga oluşturma algoritmalarının hedefleri şu şekilde listelenebilir:



Şekil 1.4 Örnek Küme ve Omurga Mimarileri

1. Her düğümün, kümeleme ve omurga işlemlerine istediği zaman, yerel olarak (*locally*) başlaması istenir. Bundan dolayı bu işlemlerin dağıtık (*distributed*) ve asenkron (*asynchronous*) olması tercih edilir.

2. Kümeleme ve omurga oluşturma işlemlerinde, enerji etkin iletişim yapmak için güçlü iletişim kanalları seçilmelidir (Lal et al., 2003).

3. Küme liderleri, küme üyelerinin sunucularıdır. Her küme lideri, kendi üyelerinin çevreden algıladığı bilgileri topladıktan sonra bu bilgileri işler, birleştirir (*aggregate*), filtreler ve çıkış düğümüne doğru yönlendirir. Bundan dolayı küme liderlerinin enerjisi eğer kümesindeki eleman sayısı çoksa kısa zamanda tükenebilir. Buna karşın ağı çok küçük kümelere bölme işlemi maliyet etkin olmayabilir. Sonuç olarak, mümkün olduğunca enerjisi yüksek düğümler küme lideri seçilmeli, küme sayısı kontrol edilebilir olmalı ve kümeler dengeli olmalıdır.

4. Algoritmalar olabildiğince çok MAC (Medium Access Control) katmanı ve fiziksel katmana (IEEE Std 802.15.4_TM 2003; IEEE 802.11 StandardWorking Group, 2007; Ye et al., 2004; Choi et al.; 2009, Polastre et al., 2004; Rajendran, et al., 2003) arayüz sağlayabilmesi için bu katmanlardan bağımsız olmalıdır.

5. Algoritmalar düşük enerji tüketimi yapmalı bu yüzden mesaj karmaşıklığı (*message complexity*) olabildiğince düşük olmalıdır. Ayrıca gecikmeyi azaltmak için zaman karmaşıklığının (*time complexity*) düşük olması beklenir.

6. Üst üste binen kümeler hata toleransı sağlamak için iyi fikir olabilir fakat bir üye düğümün birden fazla lidere algıladığı veriyi göndermesi enerji etkin değildir. Kümelerin ayrık olduğu durumlarda hata toleransı başka metotlarla sağlanabilir.

Matematik ve bilgisayar biliminde çizge teorisi (*graph theory*) çizgelerin özellikleri üzerine çalışan bir daldır. Çizge $G=(V,E)$ şeklinde gösterilen bir çift olup, V düğüm kümesini, E kenar kümesini gösterir. Yönsüz çizge yönsüz kenarlar üzerinden birleştirilmiş çizgeye denir. Şöyle ki; A düğümünden B düğümüne doğru bir kenar varsa B düğümünden A düğümüne doğru bir kenar olmalıdır. Yönlü çizge yönlü kenarlar üzerinden birleştirilmiş çizgedir. Şöyle ki; A düğümünden B düğümüne doğru bir kenar varsa B düğümünden A düğümüne doğru bir kenar olmak zorunda değildir. TDAlar yönsüz ve yönlü çizge olarak modellenebilmektedir. TDAyı modellerken aşağıda listelenen maddeleri (Younis and Fahmy, 2004; Erciyes et al., 2008; Chen and Liestman, 2002; Wu and Li, 1999; Cokuslu et al., 2006)'de olduğu gibi varsaydık:

1. Her düğümün kendi kimliği (*id*) vardır.

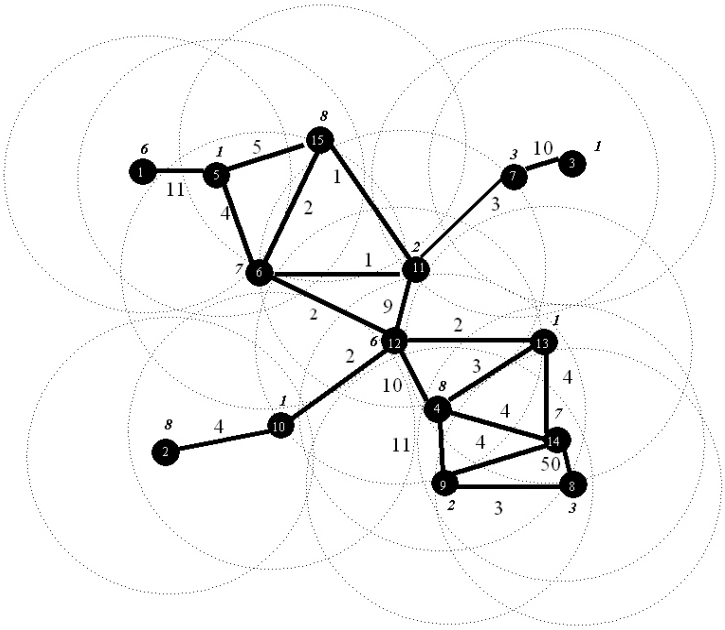
2. Düğümler hareketsizdir.

3. Düğümlerin arasındaki bağlantılar (*link*) yönsüzdür.

4. Düğümler koordinatlarını bilmez. GPS (Global Positioning System) gibi pozisyon izleyicilerine sahip değildir.

5. Bütün düğümlerin işlemci, iletişim, depolama gücü ve pil ömrü aynıdır. Uygulama çalıştıkça düğümler farklı miktarda pil tüketimi yapabilir.

6. Bütün düğümler komşularını ve komşularıyla arasındaki bağlantının kalitesini yani sinyal seviyesini bilir. Madde 3'de de söylendiği üzere bir bağlantının kalitesi iki ucundaki düğüm için aynıdır.



Şekil 1.5 Ağ modeli

Bu varsayımlara dayanarak ağı üç farklı şekilde modelleyebiliriz:

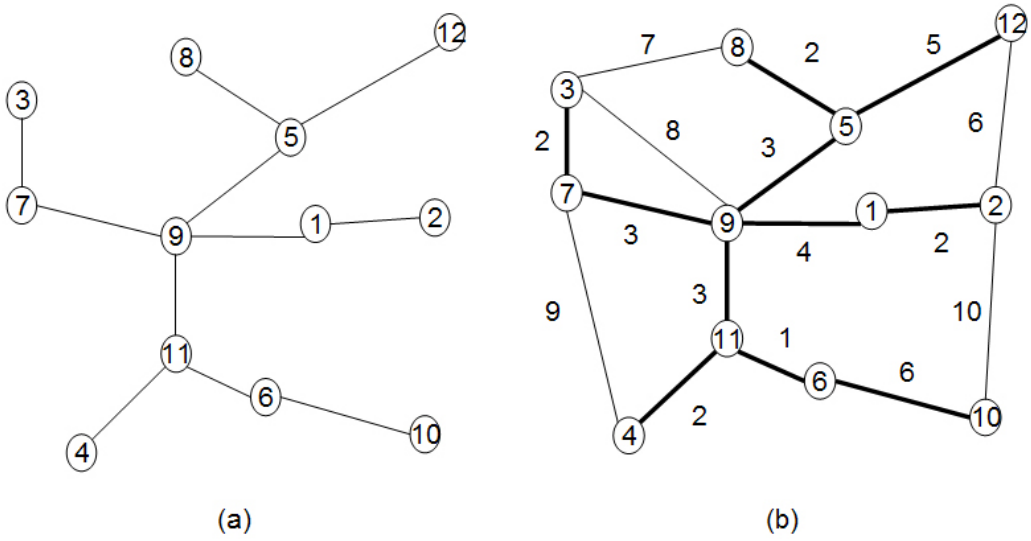
1. *Kenar Ağırlıklı Yönsüz Çizge*: Bu model $G_w(V, E_w)$ olup; V düğümlerin kümesi, E_w ağırlıklandırılmış kenarların kümesi olarak gösterilir. Kenarların ağırlık değerleri (IEEE Std802.15.4TM, 2003, IEEE 802.11 StandardWorking Group, 2007)'de verilen bağlantı kalite göstericileri (*link quality indicator*) yardımıyla atanır.

2. *Düğüm Ağırlıklı Yönsüz Çizge*: Bu model $G_w(V_w, E)$ olup; V_w ağırlıklandırılmış düğümlerin kümesi, E kenarların kümesi olarak gösterilir. Düğümlerin ağırlıkları 1/enerji değerleriyle temsil edilir.

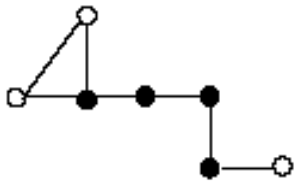
3. *Düğüm ve Kenar Ağırlıklı Yönsüz Çizge*: Bu model $G_w(V_w, E_w)$ olarak gösterilip; yukarıda anlatılan iki modelin birleştirilmiş türüdür.

Şekil 1.5'de örnek bir düğüm ve kenar ağırlıklı yönsüz çizge modeli verilmiştir. Bu modelde düğüm kimlikleri beyaz renkle içine, düğüm ağırlıkları siyah renkle ve yana yatık yazı tipiyle yanına, kenar ağırlıkları siyah renkle üzerine yazılmış; düğümlerin kapsama alanları noktalı çizgiler ile gösterilmiştir. Bu model üzerindeki düğümlerin ağırlıklarını silersek, kenar ağırlıklı yönsüz çizge modelini, kenar ağırlıklarını silersek, düğüm ağırlıklı çizge modelini elde ederiz. Çizge teorik yaklaşımlarda kullanılan diğer popüler model birim disk çizgesi (*unit disk graph*) modelidir. Bu modelde, her düğüm kendisi merkez olacak biçimde $r = 1$ yarıçaplı bir ağın merkezine konur. İki düğümün arasında bir bağlantı olabilmesi için aralarındaki uzaklığın en fazla 1 olması gerekir (Kuhn et al., 2007; Clark et al., 1990). Şekil

1.5'de gösterilen model üzerindeki düğüm ve kenar ağırlıklarını silersek birim disk çizgesi modeli elde edebiliriz. Birim disk çizgesi modeli, radyo iletişimini modelleyebildiğinden dolayı TDA için uygun bir modeldir (IEEE Std802.15.4TM, 2003; IEEE 802.11 StandardWorking Group, 2007) fakat düğümlerin enerjilerini ve bağlantıların kalitesini modelleyemez.

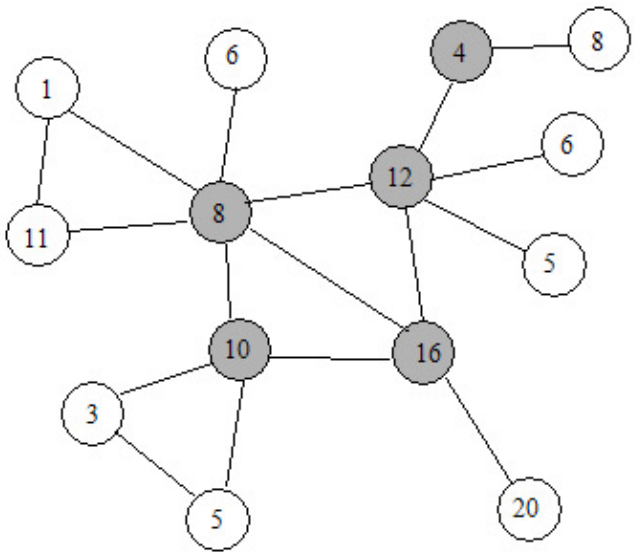


Şekil 1.6 (a) Kapsayan ağaç (b) En küçük kapsayan ağaç



Şekil 1.7 Örnek bağlı hâkim küme

Çizge teorik kümeleme ve omurga oluşturma algoritmaları ağın çizge olarak modellendiğini varsayıp bu özelliği kullanarak çizge teorik yapılarla ağı kümelere bölerler. Bu yapılardan en önemli iki tanesi kapsayan ağaç ve hâkim kümedir. Hâkim küme probleminde polinom zamanda en iyi (*optimum*) çözüm algoritmik olarak bulunamaz. Bu tür problemler NP-zor (*NP-hard*) ve NP-tam (*NP-complete*) sınıfına girer. Bu problemleri en iyinin altında ve ispatlanmış bir çözüm kalitesinde polinom zamanda çözmek için yaklaşım algoritmaları (*approximation algorithms*) tasarlanır. Yaklaşım algoritmalarının, çözüm kalitesi veya yaklaşım oranı hesaplanırken genellikle problemi kavrayacak bir sıkı örnek (*tight example*) verilir (Vazirani, 2001).



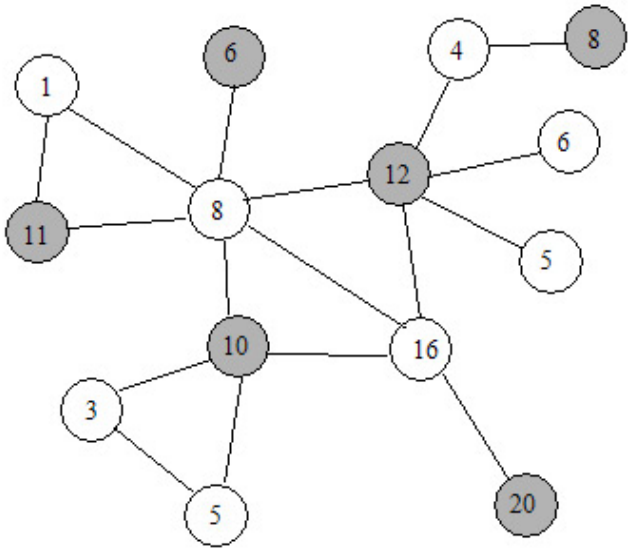
Şekil 1.8 Ağırlıklı bağlı hâkim küme

V_s=V olacak şekilde G_s = (V_s, E_s), G çizgesinin bir kapsayan alt çizgesidir. Kapsayan ağaç içinde döngü barındırmayan bir kapsayan alt çizgedir (Grimaldi, 1999). Şekil 1.6 (a)'da bir örnek kapsayan ağaç verilmiştir. Kenar ağırlıklı bir çizge üzerinde, toplam seçilen kenar ağırlıklarının en küçük olduğu kapsayan ağaca en küçük kapsayan ağaç (*minimum spanning tree*) denir. Şekil 1.6 (b)'de kalın kenarları gösterilmiş seçilen kenarlarla birlikte düğümler en küçük kapsayan ağacı oluşturur. En küçük kapsayan ağacı bulma problemini polinom zamanda çözen algoritmalar olduğundan bu problem *P* (polinom) karmaşıklık kümesi içindedir. Kapsayan ağaç kendi başına bir omurga gibi kullanılabilmesinin yanında kapsayan ağaç üzerindeki bazı düğümler küme lideri olarak seçilip ağ kümelere ayrılabilir.

Hâkim küme *S*, *G*'nin alt kümesi olan öyle bir kümedir ki, *G* de bulunan her düğüm *S* kümesinin elemanıdır veya *S* kümesindeki düğümlerden herhangi birine komşudur (West, 2001). *S* kümesi içinde düğümler de birbirleriyle bağlıysalar bu kümeye bağlı hâkim küme (BHK) denir. En küçük BHK bulma problemi NP-Tam'dır (*NP-Complete*). Şekil 1.7'de siyah ile içi doldurulmuş düğümlerle bir örnek BHK oluşturulmuştur. BHKlar tüme gönderim (*broadcast*), sanal omurga oluşturma (Stojmenovic et al., 2002) gibi ağ uygulamalarında çok önemlidir. Ayrıca BHKye ait olan düğüm küme lideri olarak seçilip ağ kümelere bölünebilir. Bunlara karşın BHK algoritmalarında istenmeyen sayıda küme lideri oluşturabilir.

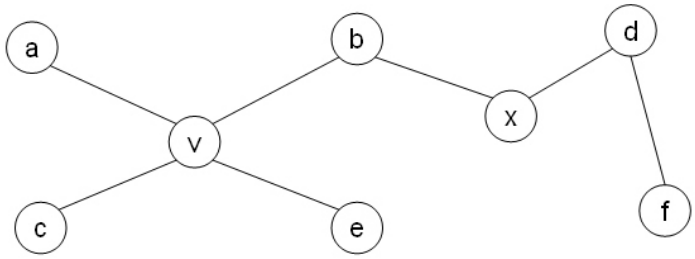
Ağırlıklı bir çizge üzerinde, düğümlerin ağırlıklarını en küçük yapacak BHKnin bulunması problemine ağırlıklı bağlı hâkim küme (ABHK) problemi denir (Boukerche, 2005). En küçük ABHK problemi, en küçük BHK problemi gibi NP-Tam'dır. Şekil 1.8'de örnek bir çizge üzerinde en küçük ağırlıklı bağlı hâkim küme gösterilmiştir. TDAda düğümlerin ağırlıklarını 1/ (düğümün enerjisi) olarak alırsak, enerji etkin bir omurga elde edip ağın yaşam süresini uzatabiliriz.

ABHK bulma veya hâkim küme problemine yapılan önemli bir yaklaşım ilk basamakta ağırlıklı bağımsız küme veya hâkim küme bulma, ikinci basamakta ağırlıklı Steiner ağacı bulmaktır. Birbirleri arasında herhangi bir bağlantı olmayan düğümlerin oluşturduğu kümeye bağımsız küme (*independent set*) denir (West, 2001). En büyük (*maximum*) bağımsız küme bulma problemi NP-Tam'dır. Düğümlerin ağırlıkları olduklarını kabul edersek, ağırlıklı bağımsız küme problemi birbirine bağlı olmayan ve toplam ağırlığı en büyük kümeye denir. Şekil 1.9'da bir örnek verilmiştir. Bu örnekte içleri gri ile dolu olanlar bağımsız kümenin içindeki düğümlerdir.



Şekil 1.9 Ağırlıklı bağımsız küme

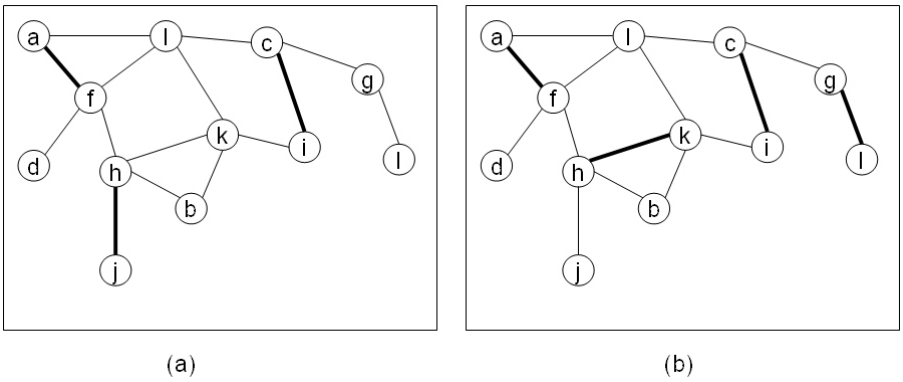
Bir sonraki basamakta ilk basamakta bulunan elemanlar birleştirilip Steiner Ağacı oluşturulur. Steiner ağacı bulma problemi en küçük kapsayan ağaç bulma problemine çok benzer. En küçük kapsayan ağaç problemi, V kümesi içindeki düğümleri, E kümesi içindeki kenarlardan en az sayıda kullanarak çember (*cycle*) oluşturmayacak şekilde bağlanması olarak tanımlanırken, Steiner ağacı problemi V kümesi içindeki düğümleri, V kümesi dışından düğümler ve E kümesi dışından da kenarlar kullanarak en az maliyetle birleştirilmesidir. En küçük Steiner ağacı bulma problemi NP-Tam'dır. Şekil 1.10'da V={a,b,c,d,e,f} kümesindeki düğümleri v ve x düğümleri kullanarak birleştiren bir Steiner ağacı görülmektedir. Bu tezde ABHKyı oluşturulan bir algoritma önerilmiş olup detaylı olarak önümüzdeki bölümlerde anlatılacaktır.



Şekil 1.10 Steiner Ağacı

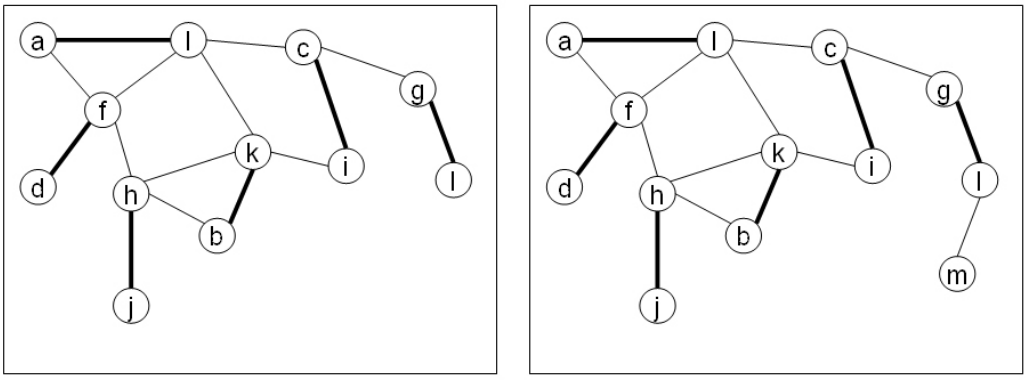
Yukarıda anlatılan kapsayan ağaç ve hâkim küme, omurga oluşturmak ve kümelemek için kullanılan çizge teorik yapılardır. Bu tezin kapsamında önerdiğimiz (Dagdeviren and Erciyes, 2010) ve literatürde kümeleme ve omurga oluşturma için kullanılmamış bir diğer çizge teorik yöntem çizge eşlemedir (*graph matching*). Eşleme problemi, bir *G* çizgesi üzerinde döngü oluşturmayan ve uçları ortak olmayan kenarların oluşturduğu bir kümedir. Şekil 1.11 (a)'da örnek bir eşleme verilmiştir; kalın ile gösterilen kenarlar eşleme kümesinin içindeki kenarlardır, bundan dolayı bu örnekteki eşleme kümesi M={e_{af}, e_{hj}, e_{ci}}'dır. *M* eşleme kümesindeki kenarlar bağlı olduğu düğümler tarafından doyurulmuş (*saturated*), kalan düğümler doyurulmamıştır. Şekil 1.11 (a)'da *a*, *f*, *h*, *j*, *c* ve *i* düğümleri doyurulmuştur.

Büyükçe eşleme, *M* eşleme kümesine yeni bir kenar eklenmesi mümkün olmayan eşlemedir. Şekil 1.11 (b)'de örnek büyükçe eşleme verilmiştir. En büyük eşleme bütün eşlemelerin içindeki en büyük kümeye denir. Bu problemler *P* karmaşıklık kümesi içindedir. Kusursuz eşleme (*perfect matching*) bütün düğümlerin doyurulması ile sağlanır (West, 2001). Kusursuz eşleme, en büyük eşlemedir fakat en büyük eşleme kusursuz eşleme olmayabilir. Örneğin, Şekil 1.12 (a)'da çizge üzerindeki kusursuz eşleme verilmiştir; bu çizgeye *l* düğümüne komşu olan *m* düğümü eklendiğinde çizgede toplam tek sayıda düğüm olacağı için kusursuz eşleme bulunamamış, fakat bulunan en büyük eşleme Şekil 1.12 (b)'de verilmiştir. İşe alım problemi ve evlilik problemi çizge eşleme ile çözülebilecek bazı örnek problemlerdir.

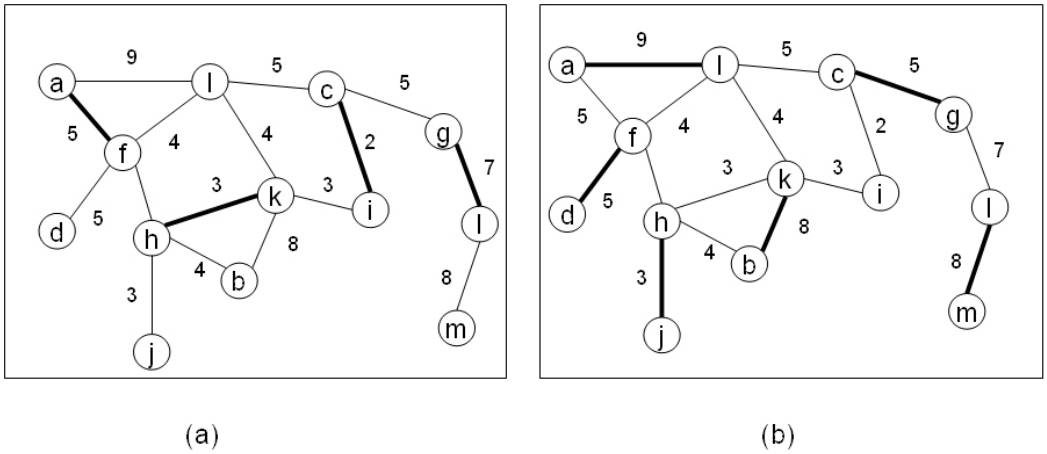


Şekil 1.11 (a) Eşleme (b) Büyükçe eşleme

Kenar ağırlıklandırılmış bir çizge üzerinde en büyük kenar ağırlığına ulaşacak şekilde en büyük ağırlıklı eşleme bulunabilir. Şekil 1.13 (a)'da örnek büyükçe ağırlıklı eşleme, Şekil 1.13 (b)'de örnek en büyük ağırlıklı eşleme kalın çizgiler ile belirtilen kenarlarla gösterilmiştir. En büyük ağırlıklı eşleme problemi *P* karmaşıklık kümesi içindedir.



Şekil 1.12 (a) Kusursuz eşleme (b) En büyük eşleme



Şekil 1.13 (a) Ağırlıklı büyükçe eşleme (b) Ağırlıklı en büyük eşleme

Bu tezde kümeleme ve omurga oluşturmak için aracı olarak kullanılan bir yöntem de konumlandırmadır (*localization*). TDAda konumlandırma, konumunu bilmeyen bir düğümün, konumunu bilen diğer düğümleri temel alarak konumunu öğrenmesi işlemidir (Alhmiedat and Yang, 2007). Birçok durumda, TDA içindeki bir düğümün konumunu bilmesi gereklidir. Örneğin hedef takip (*target tracking*) ve olay tespit (*event detection*) uygulamalarında konumlandırma işleminin önemi çok büyüktür. Büyük ölçekli duyarga ağı uygulamalarında, her düğümün konumunun kullanıcı tarafından verilmesi çok zordur. Bunun yanında, GPS çok pahalı bir tercihtir. Bu sebeplere bağlı olarak konumlandırma işleminin TDAda çok önemli olduğunu söyleyebiliriz (Karl and Willig, 2005). TDAda konumlandırma amaçları şu şekilde sıralanabilir:

1. Konumlandırmanın hata oranı olabildiğince düşük olmalıdır. Bir düğüm için hata oranı; *M* ölçülen nokta ile orijin arasındaki uzaklık ve *R* gerçek uzaklık ve orijin arasındaki uzaklık olduğunda $|M-R|/R$'ye eşittir.

2. Maliyeti azaltmak için üzerinde GPS taşıyan düğüm sayısı (kök düğüm) olabildiğince az olmalıdır.

3. Konumlandırma algoritması olabildiğince az enerji harcamalı ve alt katmanlardan bağımsız olmalıdır.

1.3. Tezin Katkıları

TDAda kümeleme ve omurga oluşturma algoritmalarının tasarımını, analizini ve uygulanmasını içeren bu tezin katkıları şu şekilde sıralanabilir:

1. Sinyal seviyesi güçlü kanalların (kenarların) seçilmesini öncelikle hedefleyen, literatürdeki çalışmalardan farklı olarak ilk kez ağırlıklı çizge eşleme tekniğini kümeleme ve omurga oluşturma sırasında kullanan senkron (*synchronous*) ve asenkron yaklaşım algoritmaları önerilmiştir (Dagdeviren and Erciyes, 2010). Senkron algoritma kümeleme yapabilirken, asenkron algoritma kümeleme ve omurga oluşturmayı birlikte yapacak şekilde tasarlanmıştır. Senkron algoritmanın toplam seçilen kenar ağırlığına yaklaşım oranı *r* raunt (*raund*) sayısı olmak üzere en iyi durumum 1/4'üne yaklaştığı ispatlanmıştır, asenkron algoritmanın *küme_üst_seviyesi* parametreleri eşitlendiğinde en kötü durumda aynı yaklaşım oranına sahip olduğu gösterilmiştir. Algoritmaların doğruluk analizlerini, zaman karmaşıklıklarını, mesaj karmaşıklılarını ve uzay karmaşıklıklarını yapılıp, benzetimleriyle birlikte literatürdeki algoritmalara göre üstünlükleri gösterilmiştir.

2. Literatürdeki algoritmaların oluşturduğu ABHKnın toplam ağırlığını düşürebilmek için *S* ABHKnın toplam ağırlığı olmak üzere Guha'nın merkezi 3 ln (*S*) algoritması (Guha and Khuller, 1998) dağıtık olarak tasarlanmıştır. Bu algoritma kendi içinde ln (*S*) yaklaşımlı ağırlıklı küme örtüsü tabanlı hâkim küme oluşturma algoritmasını ve 2 ln (*S*) yaklaşımlı Steiner ağacı oluşturma algoritmasını içermektedir. ln (*S*) yaklaşımlı merkezi hâkim küme oluşturma algoritmasını dağıtık olarak tasarlayabilmek için öncelikle bu algoritma üzerinde dağıtık yapmayı kolaylaştıracak değişiklikler yapıp aynı yaklaşım oranını sağlayan yeni bir merkezi algoritma elde edilmiştir. Bilgilerimize göre önerdiğimiz bu merkezi algoritma literatürde bulunmamaktadır. Daha sonra önerilen bu merkezi algoritma dağıtık ve senkron olarak tasarlanmıştır. 2 ln (*S*) yaklaşımlı Steiner ağacı algoritmasının da dağıtık ve senkron versiyonunu tasarlanıp dağıtık ve senkron 3 ln (*S*) yaklaşımlı ABHK oluşturan bir algoritmanın tasarımı tamamlanmıştır. Önerdiğimiz senkron

algoritmalarının yarı asenkron tasarımları da yapılmıştır. Algoritmaların doğruluk analizlerini yapılmış, zaman karmaşıklıkları, mesaj karmaşıklıkları ve uzay karmaşıklıkları analiz edilmiş ve benzetimleriyle birlikte literatürdeki algoritmalara göre üstünlükleri gösterilmiştir.

3. Yukarıda anlatılan 2 maddede önerdiğimiz algoritmalar dağıtıktır. TDAda merkezi kümeleme ve omurga oluşturma yapabilmek için bilinen yöntem çıkış düğümüne doğru komşuluk matrisinin toplanmasıdır (Bharghavan and Das, 1997). Bu yönteme alternatif olarak TDAda merkezi konumlandırma, kümeleme ve omurga oluşturmayı sağlayabilecek bir çerçeve (*framework*) önerilmiştir. Önerdiğimiz bu çerçeve bir donanım etmeni (*hardware agent*) ağda gezinerek düğümlerin önce konumlarını bulup, bu bilgileri kullanarak kümeleri ve omurgayı oluşturmuştur. Çerçevemiz düğüm başına sabit mesaj karmaşıklığına (3 mesaj) sahip olup, literatürdeki algoritmalara oranla daha az konumlandırma hatası yapmış ve daha kaliteli kümeler oluşturmuştur. Bilgilerimiz dâhilinde kümeleme ve omurga oluşturma işlemini düğüm başına sabit mesaj karmaşıklıyla yapabilen ilk çerçeve bizim tarafımızdan önerilmiştir.

1.4 Tezin Çerçevesi

Tezin ikinci bölümünde literatür özeti verilmiştir. Bu bölüm kendi içinde çizge teorik kümeleme algoritmaları, çizge eşleme algoritmaları ve konumlandırma algoritmaları olarak ayrılmıştır. Çizge teorik kümeleme algoritmaları bölümünde literatürdeki kapsayan ağaç algoritmaları ve hâkim küme algoritmaları gösterilmiştir. Çizge eşleme bölümünde literatürdeki algoritmalar sınıflandırılmış ve bu tezde kullanılan yönsüz çizge üzerinde ağırlıklı eşleme konusundaki çalışmalar detaylandırılmıştır. Konumlandırma algoritmaları bölümünde literatürdeki önemli konumlandırma yaklaşımları verilmiştir.

Tezin üçüncü bölümünde tarafımızdan önerilen eşleme tabanlı kümeleme ve omurga oluşturma algoritmaları detaylı olarak anlatılmıştır. Bu bölümde ilk olarak algoritmaların tasarımında temel alınan Hoepman'ın algoritmasının TDAya uyarlanması gösterilmiştir. Bu adımdan sonra senkron algoritmanın genel fikri tasarımı ve örnek uygulaması verilmiştir. Takiben asenkron algoritmayla ilgili aynı bilgiler detaylı olarak sunulmuştur. Algoritmalarla ilgili bu bilgiler verildikten sonra doğrulukları, kümeleme kaliteleri, seçilen kenarların kaliteleri mesaj karmaşıklıkları, zaman karmaşıklıkları ve uzay karmaşıklıkları analiz edilmiştir. Algoritmaların analizlerini yaptıktan sonra ns2 benzetim ortamında kodlayıp, literatürdeki çizge teorik kümeleme algoritmalarınla kümeleme ve omurga oluşturma performanslarını karşılaştırılmıştır. Son olarak bu algoritmaların üzerinde çalışabilecek örnek uygulamalar ve algoritmalara yapılabilecek eklentiler tartışılmıştır.

Tezin dördüncü bölümünde önerdiğimiz ağırlıklı bağlı hâkim küme algoritmaları anlatılmıştır. Bu algoritmalar iki ana bölüme ayrılmıştır: Ağırlıklı hâkim küme algoritmaları ve ağırlıklı Steiner ağacı algoritmaları. Öncelikle ağırlıklı bağlı hâkim küme algoritmalarının genel fikirleri, tasarımları ve örnek uygulamaları verilmiştir. Takiben Steiner ağacı algoritmalarının genel fikirleri, tasarımları ve örnek uygulamaları verilmiştir. Algoritmaların genel fikirlerini verdikten sonra doğruluklarının, mesaj karmaşıklıklarının, zaman karmaşıklıklarının ve uzay karmaşıklıklarının analizi verilmiştir. Bir sonraki bölümde algoritmaların performans değerlendirmeleri, hâkim düğümlerinin ağırlığı, bağlayıcı düğümlerinin ağırlığı, hâkim düğümlerin sayısı, bağlayıcı düğümlerin sayısı, tüketilen enerji miktarları, mesaj gönderimleri ve çalışma zamanları ölçülerek ve literatürdeki algoritmalarla karşılaştırılarak yapılmıştır. Son olarak algoritma bir uygulamayla birlikte çalıştırılmış ve literatürdeki algoritmaların oluşturduğu omurgaların enerji tüketiminle kıyaslanmıştır.

Tezin beşinci bölümünde etmen tabanlı konumlandırma ve kümeleme çerçevesi verilmiştir. Bu bölümde öncelikle ağ modeli anlatılmıştır. Ağ modeli anlatıldıktan sonra bu modelin üzerinde çalışacak çerçevemizin genel fikri, konumlandırma algoritması ve kümeleme algoritmaları gösterilmiştir. Çerçevemizin çalışma prensipleri anlatıldıktan sonra çerçevemizin analizi verilmiş ve performans değerlendirmesi yapılmıştır. Çerçevemizin performansını değerlendirirken konumlandırma ve kümeleme sonuçlarıyla birlikte enerji tüketimleri ölçülüp literatürdeki çalışmalarla karşılaştırılmıştır. Son olarak çerçevemizin üzerinde çalışabileceği örnek bir uygulama olarak hedef takip uygulaması önerilmiş ve bu uygulamanın çerçevemizle birlikte kullanıldığındaki maliyeti literatürdeki çalışmalarla karşılaştırılarak incelenmiştir. Tezin altıncı bölümünde sonuçlar ve çalışmayı ileriye götürebilecek eklentiler verilmiştir.

2. LİTERATÜR ÖZETİ

Bu bölümde literatürdeki çizge teorik kümeleme algoritmaları, çizge eşleme algoritmaları ve konumlandırma algoritmaları verilecektir.

2.1. Çizge Teorik Kümeleme ve Omurga Oluşturma Algoritmaları

Çizge teorik kümeleme algoritmaları genel olarak hâkim küme tabanlı ve kapsayan ağaç tabanlı olarak ayrılabilir.

2.1.1. Kapsayan Ağaç Algoritmaları

Prim, Kruskal ve Borouvka merkezi en küçük kapsayan ağaç algoritmaları önermişlerdir (Jungnickel, 2005). Prim'in algoritmasında herhangi bir düğümden başlayarak en küçük kenarlar üzerinden diğer düğümler bir küme içine alınır ve bu işlem tekrarlanır.

Prim Algoritması (G,w;T)
$g(1) \leftarrow 0$, $S \leftarrow 0$, $T \leftarrow 0$
Döngüye i=2'den başla ve n'ye kadar $g(i) \leftarrow \infty$ döngüyü bitir.
$S \neq V$ olduğu sürece
  Öyle bir i seç ki; $g(i)$ en küçük olacak şekilde $i \in V \setminus S$ olsun ve $S \leftarrow S \cup i$ yap.
  Eğer $i \neq 1$ ise $T \leftarrow T \cup \{e(i)\}$
  $j \in A_i \cap (V \setminus S)$ olduğu sürece
    Eğer $g(v) > w(ij)$ ise $g(v) \leftarrow w(ij)$ ve $e(v) \leftarrow ij$ olsun
  döngüyü bitir.
döngüyü bitir.

Şekil 2.1 Prim'in algoritması

İlk basamakta başlangıç düğümünden çıkan en küçük kenar seçilir ve ucundaki düğüm ile birlikte başlangıç düğümü aynı küme içine alınır. İkinci basamakta bu kümenin içindeki iki düğümden küme dışına çıkan en küçük kenar seçilir ve ucundaki düğüm küme içine alınır. Bu işlem tekrarlamalı bir şekilde bütün düğümler aynı küme içine girdiğinde algoritma tamamlanır. Şekil 2.1'de Prim'in algoritması verilmiştir. Bu algoritmaya G çizgesi, w kenar ağırlık kümesi ve T seçilen kenar kümesi parametre olarak verilir. $g(i)$ kümesi merkez düğümden i. düğüme ağırlığı tutar. Algoritma düğüm 1den itibaren başladığı için $g(1)$ değeri 0'a eşittir. S kümesi bir düğümden itibaren başlar, algoritma V=S olana kadar devam eder. A_i kümesi i düğümünün komşularının kümesidir. Prim'in algoritmasının zaman karmaşıklığı

$O(V^2)$'dir. Yığın (*heap*) veri yapısı kullanılarak algoritmanın zaman karmaşıklığı $O(E+V\ log_2(V))$'ye düşürülür.

Kruskal'ın algoritmasında öncelikle kenarlar küçükten büyüğe doğru sıralanır. Her düğüm ayrı bir küme içine koyulur. En küçük kenardan itibaren başlanır, en küçük kenarın ucundaki düğümler aynı küme içine konur. Her basamakta birleştirilen kenarların ucundaki düğümler aynı kümenin içine konur. Eğer bir kenarın ucundaki iki düğüm aynı küme içindeyse işlem yapılmaz. Algoritma bütün kenarları kontrol ettiğinde biter. Kruskal'ın algoritması Şekil 2.2'de verilmiştir. Öncelik kuyruğu (*priority queue*) kullanarak algoritmanın zaman karmaşıklığı $O(E\ log_2(E))$'ye düşürülür.

Kruskal Algoritması (G, w; T)
$T \leftarrow 0$
Döngüye i=2'den başla ve n'ye kadar
  Eğer e_k, T'ye ait olan kenarlar ile döngü oluşturmuyorsa e_k'yı T'ye ekle.
döngüyü bitir.

Şekil 2.2 Kruskal'ın algoritması

Borouvka Algoritması(G, w; T)
Döngüye i=1'den başla ve n'ye kadar $V_i \leftarrow \{i\}$ döngüyü bitir.
$T \leftarrow 0$
$M \leftarrow \{V_1,\ldots,V_n\}$
$|T| < n$-1 olduğu sürece
  $U \in M$ olduğu sürece
    Bütün $e' = u'v'$, $u' \in U$, $v \notin U$ kenarları için öyle bir $e = uv$ bul ki $u \in U$, $v \notin U$ ve $w(e) < w(e')$ olsun.
    v'yi içeren U' bileşenini bul.
    $T \leftarrow T \cup \{e\}$
  döngüyü bitir
  $U \in M$ olduğu sürece U ve U' bileşenlerini kaynaştır.
döngüyü bitir

Şekil 2.3 Boruvka'nın algoritması

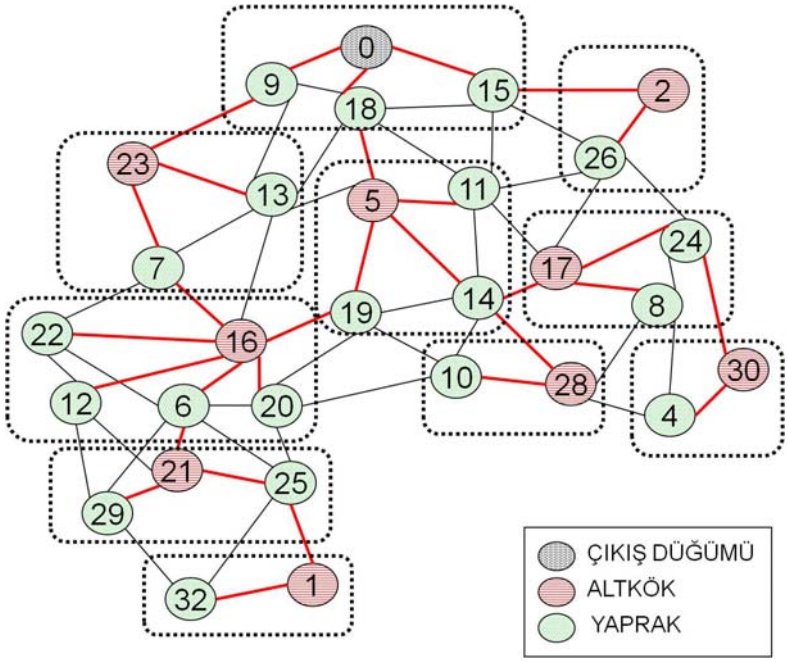
Boruvka'nın algoritması, Prim'in algoritmasına benzer fakat Prim'in algoritmasında tek bir kümenin büyüyüp bütün düğümleri içine alırken, Boruvka'nın algoritmasında her düğüm bir küme içinde başlayıp her tekrarda kendinden çıkan en küçük kenarla birlikte başka bir kümeye bağlanır. Böylelikle her tekrarda birden fazla kenar seçilip, birden fazla küme birleşebilir. Algoritma bütün düğümler aynı küme içine girince sonlanır. Boruvka'nın algoritması Şekil 2.3'de verilmiştir. Algoritmanın zaman karmaşıklığı $O(V\ log_2(E))$'dir.

Gallagher'in (Gallagher et al., 1983) algoritması (GHS), bir yönsüz ağırlıklı çizge üzerinde dağıtık olarak en küçük kapsayan ağacı (EKKA) bulan öncü bir algoritmadır. Gallagher'in algoritması Boruvka'nın algoritmasının dağıtık sürümüdür. Algoritmanın amacı küçük parçaları birleştirip daha büyük parçaları dış kenarlar üzerinden birleştirmektir. Parça denilen yapı EKKAya ait olan bir alt ağaçtır. Dış kenar, parçacıkları birbirlerine birleştiren en küçük kenarlardır. Parçaların birleşmesi bazı kurallar çerçevesinde gerçekleşmektedir. İçinde tek bir düğüm içeren parça, sıfır seviyesine sahiptir. *L* seviyesindeki *F* ve *L'* seviyesindeki *F'* parçacıklarının birleşecek olduğunu varsayalım. Aşağıdaki kurallar çerçevesinde birleşme gerçekleşecektir.

- Eğer *L*<*L'* ise *F*, *F*'nin içine dâhil olur. Yeni seviye *L'* olur.
- Eğer *L*=*L'* ise *F*, *F'*nin içine dâhil olur. Yeni seviye *L*+1 olur.
- Eğer bu durumlardan bir değilse *F*, *F'*nin belli bir seviyeye erişmesini bekler.

Yukarıdaki kurallar ışığında, üzerinde birleşilen kenara, parçacığın yeni çekirdeği denir. Çekirdek üzerinde bağlı olan 2 düğüm mesaj değişerek, üzerinde birleşilecek yeni kenara karar verecektirler. En kötü durumda toplam mesaj sayısı, N düğüm sayısı, E kenar sayısı olmak üzere $5N\log_2(N)+2E$ olacaktır. Algoritmanın zaman karmaşıklığı $O(E+N\log_2(N))$ olarak verilebilir. Awerbuch (Awerbuch et al., 1987), Garay (Garay et al., 1993) ve Peleg (Peleg and Rubinovich, 2000)'ın tarafından bu algoritma geliştirilmiştir. Banerjee (Banerjee and Khuller, 2001) kablosuz ağlarda hiyerarşik yönlendirme yapmak için kapsayan ağaç tabanlı bir protokol önermiştir. Kümeler üst üste bindirilmiş ve kümelerin eleman sayısına üst sınır getirilmiştir.

Erciyes (Erciyes et al., 2008) dağıtık kapsayan ağaç tabanlı bir kümele algoritması önermiştir (*distributed spanning tree algorithm*, DSTA). DSTA'da derinlik parametresi ile kümelerin enleri (*diameter*) ayarlanabilir. Çıkış düğümü periyodik olarak *ANA(hop sayısı)* (*PARENT(nhops)*) mesajını komşu düğümlerine algoritmayı tetiklemek için gönderir. Her düğüm *ANA(hop sayısı)* mesajını aldıktan sonra *ANA((hop sayısı*+1*) mod derinlik))* mesajını komşularına gönderir. *ANA(hop sayısı*=0*)* olarak mesajını alan düğümler *ALTKÖK* (*SUBROOT*), *ANA*(*hop sayısı*<*derinlik*) mesajını alan düğümler *ORTA* (*INTERMEDIATE*), *ANA*(*hop sayısı*=*derinlik*) mesajını alan düğümler de *YAPRAK* (*LEAF*) durumuna geçer. Örnek DSTA (*derinlik*=1) uygulaması Şekil 2.4'te verilmiştir; şekilde kalın gösterilen kenarlar DSTA'da seçilen kenarlardır. Noktalı çizgiler ile gösterilen bölgelerin içinde kalan düğümler aynı kümenin içindeki elemanlardır.



Şekil 2.4 DSTA örnek uygulama

2.1.2. Hâkim Küme Algoritmaları

Guha, $|S|$ en küçük BHK içindeki eleman sayısı olmak üzere $\log_2(S)$ yaklaşımlı BHK algoritması önermiştir (Guha and Khuller, 1998). Bu algoritmada derecesi en büyük düğümden itibaren başlayarak bir T ağacı büyür. İlk basamakta en büyük dereceli düğüm bulunur, bu düğüm siyaha komşuları griye boyanır. Daha sonra her basamakta işaretlenmemiş (beyaz renkli düğümler) komşu sayıları en büyük olacak şekilde bir adet gri ve bir beyaz olacak şekilde iki adet düğüm seçilir ve bu düğümler siyaha boyanır. Bütün düğümler işaretlenene kadar algoritma çalışır. Şekil 2.5'de Guha'nın algoritması verilmiştir.

Wu'nun iki aşamalı dağıtık BHK algoritmasında öncelikle her düğüm kendini F olarak işaretliyor (Wu and Li, 1999). İlk fazda, bağlı komşusu olmayan düğümler kendini T olarak işaretler. İkinci aşamada daha gelişmiş kurallar kullanarak ilk fazda elde edilen eleman sayısı azaltmaya çalışıyor. Dai (Dai and Wu, 2004), Nanuvala (Nanuvala, 2006) ve Cokuslu (Cokuslu et al., 2006) Wu'nun algoritmasına ekstra kurallar ekleyerek BHKdaki eleman sayısını azaltmayı amaçladılar. Bunların dışında literatürde azımsanmayacak sayıda dağıtık BHK algoritması bulunmaktadır (Alzoubi et al., 2002; Ni et al., 2005; Thai et al. 2007; Yuanyuan et al., 2007; Schneider and Wattenhofer, 2008). Bunlara rağmen araştırmalar devam etmekte günümüz itibariyle de geliştirilmeler bulunmaktadır (Misra and Mandal, 2010).

Bütün düğümleri beyaz olarak işaretle. Derecesi en çok olan düğümü siyah, komşularını gri olarak işaretle. Tarama işleminin tanımı: *u* gri ve *v* komşu beyaz düğüm olsun. Önce *u*'yu siyah

yap ve *u*'nun komşularını gri yap, daha sonra *v*'yi siyah yap ve komşularını gri yap.
Tarama işlemi sırasında kazancın tanımı: Tarama işlemi sırasında griye boyanan toplam düğüm sayısı.
Beyaz düğüm olduğu sürece
Kazancı en büyük olacak şekilde tek bir düğüm veya bir çift düğümü tara.
döngüyü bitir.

Şekil 2.5 Guha'nın BHK Algoritması

Guha'nın merkezi algoritması 2 basamaklı ABHK algoritması önermiştir (Guha and Khuller, 1998). İlk basamakta (Chvatal, 2002)'de verilen algoritmayı kullanarak aç gözlü (*greedy*) küme örtüsü (*set cover*) tabanlı HK bulur. Bu algoritmanın yaklaşım oranı *S* en küçük hâkim kümenin ağırlığı olmak üzere ln (*S*) dir. İkinci basamakta aç gözlü (Klein and Ravi, 1995)'de verilen algoritmayı kullanarak düğüm ağrılıklı Steiner ağacı algoritmasıyla ilk basamakta bulduğu HK düğümlerini birleştirir. Bu algoritmanın yaklaşım oranı 2 ln (*S*) dir. Bu iki basamağın yaklaşım oranını toplarsak 3 ln (*S*) olur. Guha'nın algoritmasının basamakları Şekil 2.6'da verilmiştir.

Basamak 1. Ağırlık küme örtüsü tabanlı yaklaşım algoritmasını kullanarak hâkim kümeyi bul.
Basamak 2. Hâkim küme içerisindeki elemanları birleştirmek için Klein ve Ravi tarafından önerilen ağırlıklı Steiner ağacı algoritmasını kullan.

Şekil 2.6 Guha'nın ağırlıklı bağlı hâkim küme algoritması

Guha'nın ilk basamakta kullandığı küme örtüsü tabanlı HK algoritması Şekil 2.7'de verilmiştir. Algoritma genel olarak bir tekrarda, her düğüm için ağırlık oranını, (düğümün maliyeti) / (düğümle birlikte örtülmemiş komşularının maliyeti) olarak hesaplayıp, en küçük ağırlıklı düğümü hâkim küme içine aldıktan sonra bu düğümün komşularını örtüp ağırlıklarını günceller.

1. Tüm düğümler örtüldüyse algoritmayı bitir.
2. Ağırlık oranı = (Düğümün maliyeti / Düğümle birlikte örtülmemiş komşularının maliyeti) denklemiyle en küçük ağırlıklı düğümü seç.
3. Seçilen düğümü ve komşularını ört.
4. Seçilen düğümün komşularının ağırlıklarını güncelle.
5. Birinci basamağa dön.

Şekil 2.7 Merkezi Küme Örtüsü Tabanlı HK Algoritması

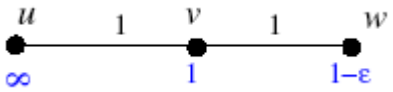
1. *V* kümesi içindeki düğümlerin ağırlık oranlarını 0 yap.
2. *V* kümesi içinde birbirlerine komşu olan düğümleri aynı ağaca koy, hiç komşusu olmayan düğümleri farklı ağaçlar içine koy.
3. *V* kümesi dışındaki düğümlerden ağırlık oranı = ((düğümün maliyeti+ düğümün

ağaçlara olan uzaklığı) / *V* kümesi içinden bağlanacak ağaçlardaki toplam düğüm sayısı) olacak şekilde en küçük ağırlıklı düğümü bağlayıcı olarak seç.
4. *V* kümesinden bağlayıcı düğümle bağlanacak ağaçlar üzerindeki düğümleri bağla, bağlayıcı ile birlikte aynı ağacın içine koy.
5. *V* kümesi içindeki düğümler aynı ağaç üzerindeyse algoritmayı bitir, değilse 3. basamağa dön.

Şekil 2.8 Klein ve Ravi'nin 2 ln (*S*) Yaklaşımlı Merkezi Düğüm Ağırlıklı Steiner Ağacı Algoritması

Bulunan bu hâkim kümenin birleştirilip bağlı hâkim küme yapılması için düğüm ağırlıklı Steiner ağacı (*node weighted Steiner tree*) yapısı kullanılır. Steiner ağacı bulma probleminde düğümler ve kenarlar ağırlıklı olabilir. Düğümlerin ağırlıklı olduğu Guha'nın ABHK bulurken kullandığı, Klein ve Ravi tarafından önerilen merkezi 2 ln (*S*) yaklaşımlı Steiner ağacı algoritması (Klein and Ravi, 1995) Şekil 2.8'de verilmiştir. Öncelikle *V* kümesi içinden birbirlerine komşu olan düğümler aynı ağaç içine koyulur, eğer *V* kümesi içindeki bir düğüm hiçbir düğüme komşu değilse tek başına bir ağaç oluşturur. Bu algoritmada bağlanacak olan *V* kümesi içindeki düğümlerin ağırlıkları 0 olarak kabul edilir. Dikkat edilirse başlangıçtaki *V* kümesi, Şekil 2.7'de verilen merkezi küme örtüsü tabanlı algoritma tarafından üretilen hâkim elemanlardır. *V* kümesinin dışında kalan her düğüm için ağırlık oranı (düğümün maliyeti / bağlanacak ağaçlardaki toplam düğüm sayısı) olarak hesaplanır. Ağırlık oranı en küçük olan düğüm bağlayıcı (*connector*) olarak seçilir ve bu düğüm üzerinden bağlanabilecek ağaçlar bağlanıp, *V* kümesi içine dâhil edilir. Bağlayıcı seçimi *V* kümesi içindeki bütün düğümler bağlanana kadar devam eder. Yani *V* kümesi içindeki düğümler tek bir ağaç olana kadar devam eder. Algoritma sonlandığında *V* kümesi ağırlıklı bağlı hâkim kümedir.

Bilgilerimiz dâhilinde literatürdeki dağıtık ABHK algoritmalarının sayısı (Chatterjee et al., 2002; Song et al., 2005; Wang et al.; 2006), dağıtık BHK algoritmanın sayısına göre çok daha azdır. Bağlı olmayan ağırlıklı hakim küme oluşturmanın yöntemlerinden bir tanesi büyükçe (*maximal*) bağımsız küme kurmaktır (Chaterjee et al., 2002). (Chatterjee02 et al., 2002)'ye göre ağırlıklı büyükçe bağımsız küme bulmak için şöyle bir yol izlenir: Öncelikle bütün düğümlerin *BEYAZ* durumunda olduğunu düşünelim. Ağırlığı komşularından küçük olan bir *u* düğümü komşularına *HâkimElemanım* mesajı göndersin ve *HâkimEleman* olsun. *HâkimElemanım* mesajını komşusundan alan bir *v* düğümü komşularına *SıradanElemanım* mesajı gönderiyor ve *SıradanEleman* olur. *SıradanElemanım* mesajını alan bir düğüm *BEYAZ* komşuları içinde en büyük ağırlığa sahip olduğu an *HâkimElemanım* mesajını gönderip *HâkimEleman* duruma geçer. Şekil 2.9'da ideal algoritmanın seçeceği düğüm sadece *v* ve ağırlık 1 olduğunda bu yöntem *u* ve *w* düğümlerini seçip; toplam ∞ ağırlık seçer.



Şekil 2.9 Büyükçe bağımsız küme ile hakim küme bulma yönteminin en iyiden çok uzak sonuç bulduğu örnek çizge

Bao ağırlıklı bağlı hakim küme bulmak için bir yöntem önermiştir (Bao and Garcio-Luna-Aceves, 2003). Bu yöntemde düğümler iki kurala bakarak hakim kümeyi kurarlar. Bu algoritmanın yaklaşım oranı yoktur. Bao'nun algoritması ilk safhada hakim kümeyi bulur ve kuralları şöyledir:

1. Eğer bir düğüm tüm komşularından daha düşük ağırlığa sahipse.

2. Eğer bir düğümün ağırlığı herhangi bir komşusunun tüm komşularından azsa.

Gezgin ağlarda, ağ paketlerinin güvenliğini sağlamak için Song tarafından ağırlıklı bağlı hakim küme tabanlı bir algoritma önerilmiştir (Song et al., 2005). Algoritmanın amacı düğümlerin bazılarını gözlemleyici olarak seçip, ağ üzerinde genel bir ihlal tespit sistemi (*intrusion detection system*) oluşturmaktadır. Düğümlerin ağırlığının enerjilerine bağlı olabileceğini söylemişlerdir. Algoritmanın mesaj karmaşıklığı O(N), zaman karmaşıklığı Δ düğüm derecesi olmak üzere O($N\Delta\log_2(\Delta)$)'dir. Algoritmanın yaklaşım oranı yoktur.

Wang, duyarga ağlarında omurga oluşturmak amacıyla iki fazlı bir algoritma önermiştir (Wang et al., 2006). İlk safha kendi içinde ikiye ayrılır. İlk safhada öncelikle ağırlıklı büyükçe bağımsız küme bulunur. Ağırlıklı büyükçe bağımsız küme elemanları daha sonra küme örtüsü algoritmasını çalıştırarak hakim kümeyi bulurlar. İlk safhanın işleyişi şu şekildedir: Algoritmanın başında bütün düğümler *BEYAZ* konumundadır. Maliyeti komşularına göre en küçük düğümler komşularına *HâkimElemanOlmakİstiyorum* mesajı gönderip *OlasıHâkimEleman* konumuna geçerler. Bu mesajı komşusundan alan düğüm *SıradanEleman* konumuna geçip, *SıradanElemanım* mesajını gönderir. *SıradanElemanım* mesajını komşusundan alan *BEYAZ* durumundaki bir düğüm komşusunu *BEYAZ* komşular listesinden siler ve maliyeti *BEYAZ* komşularından küçükse veya tüm komşuları *SıradanEleman* ve kendisi izole kaldıysa, *HakimElemanOlmakİstiyorum* mesajı gönderir; *OlasıHâkimEleman* konumuna geçer. İlk safhanın ikinci kısmında *OlasıHâkimEleman* durumundaki düğümler, en fazla iki zıplama uzaklıktaki bütün komşularının ağırlıklarını öğrenir, kendilerini ve bir zıplama uzaklıktaki komşularını örtecek, şekilde ağırlıklı küme örtüsü yöntemini kullanırlar. *OlasıHâkimEleman* durumundaki bir düğüm, kendinin ve komşularının en fazla iki zıplama uzaklıktaki diğer başka komşuları tarafından daha az maliyetle örteceğini bulursa, küme örtüsü

içine giren düğümlere *SenHâkimElemansın* mesajı gönderir. Eğer bulamazsa kendi *HâkimEleman* olur. İlk safhanın genel olarak basamakları Şekil 2.10'da verilmiştir. İlk safha sonucunda oluşan hakim elemanlar aynı zamanda küme lideridir.

1. Chaterjee'nin yöntemine benzer olarak en küçük bağımsız kümeyi bul, en küçük bağımsız kümenin içerisine giren düğümler hakim eleman olsun.
2. Hakim eleman olan her düğüm iki zıplama uzaklığa kadarki tüm düğümleri kullanarak kendisini ve bir zıplama uzaklıktaki komşularını ağırlıklı küme örtüsünü kullanarak daha az maliyet ile örtebiliyorsa kendisi hakim eleman olmaktan vazgeçer ve küme örtüsü içindeki düğümlere hakim eleman olması için mesaj gönderir.

Şekil 2.10 Wang'ın algoritmasının birinci safhasının kodu

İlk safhada bulunan küme bir hakim kümedir fakat bağlı bir hakim küme değildir. İkinci safhada, birinci safhada oluşturulan hakim küme birleştirilip bağlı hakim küme elde edilir. HEler birbirlerine en fazla 2 zıplama uzaklıktaki diğer HE lere olan maliyetlerini öğrenirler. Bu öğrenme işlemi diğer sıradan düğümler üzerinden olur. Bir düğümün diğer bir düğüme olan maliyeti aradaki düğümlerin toplam maliyeti kadardır. Düğümler bir zıplama uzaklıktaki HE listesini *TekZıplamadakiHâkimElemanListesi* mesajı ile gönderirken, iki zıplama uzaklıktaki HE bilgisini *ÇiftZıplamadakiHâkimElemanListesi* ile gönderir. İkinci safhanın ilk işlemi yukarıda özetlediğimiz işlemlerdir. Daha sonra HEler bu verileri kullanarak dağıtık en küçük kapsayan ağaç algoritması çalıştırırlar. Şekil 2.11'de Wang'ın algoritmasının ikinci safhasının basamakları verilmiştir.

1. Birbirine en fazla 2 zıplama uzaklıkta olan hakim elemanlar aralarındaki en az maliyetli yolu öğrenirler.
2. Hakim elemanlar bu verileri kullanarak en küçük kapsayan ağaç algoritması çalıştırırlar.

Şekil 2.11 Wang'ın algoritmasının ikinci safhasının kodu

Wang'ın Algoritmasını veri yapılarının bulunması açısından incelersek karşımıza genel olarak şu denklem çıkar:

ABHKnin Oluşturulması İşlemi = (Ağırlıklı Büyükçe Bağımsız Kümenin Bulunması İşlemi) + (Küme Örtüsünün Bulunması) + (Ağırlıklı En Küçük Kapsayan Ağacın bulunması)

Algoritma 1 ilk safha, Algoritma 2 ikinci safhayı göstersin. $\alpha(G)$, G çizgesi içindeki en büyük bağımsız eleman sayısını göstersin. $\alpha^{[k]}(G)$ G çizgesi içinde birbirlerine en fazla k zıplama uzaklıktaki en büyük bağımsız eleman sayısını göstersin. Birim disk çizgesi (BDÇ) için $\alpha^{[1]}(\text{BDÇ})=5$ ve $\alpha^{[2]}(\text{BDÇ})=18$ olduğu

bilinmektedir (Wang et al., 2006). Algoritma 1'in birim disk çizgesi üzerinde kurduğu HEnin ağırlığı d en büyük düğüm derecesi, s birbirlerine komşu düğümlerin ağırlıkları arasındaki en büyük oran olmak üzere $\min(18\log_2(d), 4s+1)$'den küçüktür. Genel bir G çizgesi için bu değer $\min(\alpha^{[2]}(G)\log_2(d), (\alpha^{[1]}(G)-1)s+1)$'dir. Algoritma 2'nin seçtiği bağlantı düğümlerin birim disk çizgesi üzerindeki ağırlığı, en iyi ağırlık A olmak üzere $10A$ ile sınırlanmıştır. Genel bir G çizgesi için bu değer $2\alpha^{[1]}(G)$'ye eşit olur. Böylelikle Wang'ın algoritmasının seçtiği düğümlerin toplam ağırlığı, BDÇ üzerinde $\min(18\log_2(d), 4s+1)+10A$'dan küçüktür. E kümesi, iletişim kanallarının oluşturduğu küme, N düğüm sayısı olmak üzere, Wang'ın algoritmasının mesaj karmaşıklığı Algoritma 2'deki dağıtık ağırlıklı en küçük kapsayan ağacın bulma işlemine eşit olup, $O(N\log_2(N)+E)$'dir. Algoritmanın zaman karmaşıklığı Algoritma 1'e eşit olup $O(N^2)$ dir.

2.2. Çizge Eşleme Algoritmaları

Genel olarak dağıtık eşleme problemlerini şu kategorilere ayırabiliriz:

1. Yönsüz çizgeler üzerinde en büyük eşleme (Czygrinow and Hanckowiak, 2006)

2. Yönsüz çizgeler üzerinde en büyük ağırlıklı eşleme (Hoepman, 2004)

3. İki kısımlı (*bipartite*) çizgeler üzerinde eşleme ve ağırlıklı eşleme (Chattopadhyay et al., 2002)

4. Ağaçlar üzerinde eşleme ve ağırlıklı eşleme (Karaata and Saleh, 2000)

5. Yönlü Çizge Üzerinde Eşleme (Hsu et al., 2004)

Karaata (Karaata and Saleh, 2000) öz kararlı, dağıtık ağaçlar üzerinde $O(N^4)$ zamanda çalışan, 4. kategoriye koyabileceğimiz bir eşleme algoritması önermiştir. Chattopadhyay (Chattopadhyay et al., 2002) büyükçe eşleme problemini ağırlıksız çizgeler üzerinde çözmek için öz kararlı bir algoritma önermiştir. Önerilen algoritma iki kısımlı çizge üzerinde $O(N^2)$ zamanda çalışan ve 3. kategoriye giren bir algoritmadır. Aynı zamanda algoritmaları içinde çevrim (*cycle*) olan ağlarda da büyükçe eşlemeyi bulabilir. Hsu büyükçe eşleyen çok-kanallı MAC protokolünü tasarsız ağlar için geliştirmiştir (Hsu et al., 2004). Bu algoritma dağıtık ve öz kararlı olup, ağı yönlü bir çizge şeklinde modellemesi üzerine tasarlanmıştır ve 5. kategoriye koyulabilinir. Algoritmanın genel fikri büyükçe eşleme kullanıp bant genişliğinin etkin kullanılmasıdır. Algoritmanın kilitlenme (*deadlock*) yapmadığı gösterilmiştir. Czygrinow, 1. kategoriye giren yönsüz bir çizge için polilogaritmik zamanda çalışan en büyük eşleme algoritması tasarlamıştır (Czygrinow and Hanckowiak, 2006).

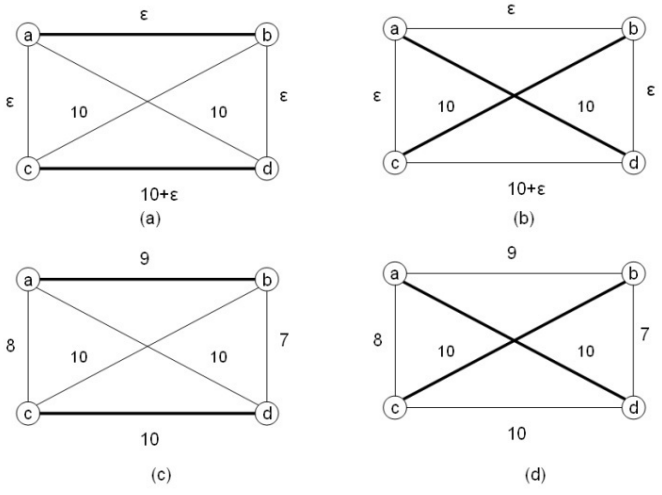
Bu tezde eşleme problemini kümeleme için kullanmak istiyoruz. Kümelenin amacı aralarında benzerlik oranı yüksek verilerin dengeli bir şekilde aynı grup içine alınmasıdır. TDAlar için bu durumu ele aldığımızda aralarında iletişim kanalları yüksek düğümleri aynı küme içine almak, rastgele almaktan daha avantaj sağlayacaktır. Bu durumu göz önüne alırsak dağıtık eşleme yaparken 2. kategoriyi seçmenin yani yönsüz çizgeler üzerinde en büyük ağırlıklı eşleme yapmanın diğer yöntemlere göre daha avantajlı olacağını söyleyebiliriz.

2.2.1. Yönsüz Çizgeler Üzerinde Ağırlıklı Eşleme

En büyük ağırlıklı eşleme problemi *P* içindedir. Buna rağmen daha iyi performanslar elde etmek için yaklaşım algoritmaları da önerilmiştir. Ağırlıklı eşleme algoritmalarını çalışma prensiplerine göre dağıtık ve merkezi olarak ayırabiliriz. Algoritmaların bir kısmı en büyük ağırlıklı eşleme yaparken bir kısmı da en büyük ağırlıklı eşlemeye yaklaşırlar. Gabow (Gabow, 1990) $O(|V||E|+ |V| \log_2(|V|)$ zamanda en büyük ağırlıklı eşleme yapan, Avis (Avis, 1983) $O(|E|\log_2(|V|))$ zamanda çalışan 1/2 yaklaşımlı, Preis (Preis, 1999), $O(|E|)$ zamanda çalışan 1/2 yaklaşımlı (Şekil 2.12'de kodu verilmiştir), Drake 2/3-ε yaklaşımlı (Drake and Hougardy, 2003), Petite (Petite and Sanders, 2004) 2/3-ε yaklaşımlı merkezi algoritmalar önermişlerdir.

```
Preis'ın Algoritması (E)
M(G) = 0 /* Eşleme kümesi */
E ≠ 0 olmadığı sürece
    E içinde en en büyük kenarı(e) seç
    e'yi M(G)'ye ekle.
    e'yi ve ona bitişik olan bütün kenarları E kümesinden sil
döngüyü bitir.
```

Şekil 2.12 Preis'ın algoritması



Şekil 2.13 (a), (c) Preis'ın uygulamaları (b), (d) En büyük eşleme uygulamaları

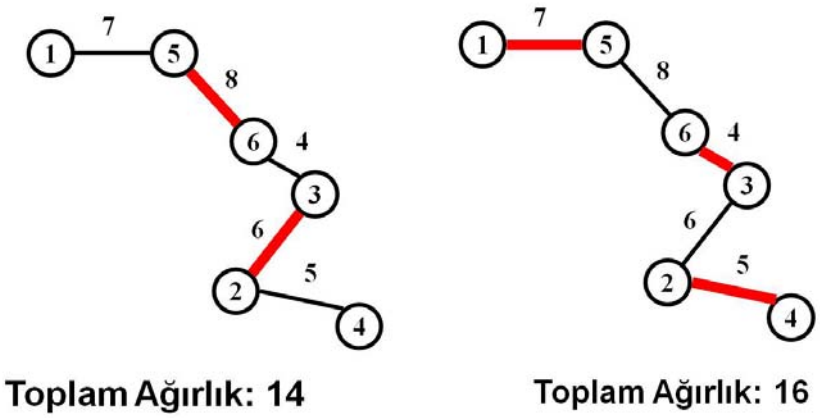
Uehara (Uehara and Chen, 2000) 1/*d* (d: En büyük düğüm komşu sayısı veya derece) yaklaşımlı, Wattenhofer (Wattenhofer and Wattenhofer, 2004) 1/5 yaklaşımlı, Hoepman (Hoepman, 2004) 1/2 yaklaşımlı, Lotker (Lotker et al., 2008) 1/2, Nieberg 1-ε yaklaşımlı (Nieberg, 2008) dağıtık algoritmalar önermişlerdir.

Preis'ın algoritması Şekil 2.12'de verilmiştir. Preis'ın algoritması açgözlü bir algoritmadır. Bu algoritmada kenarlar büyükten küçüğe doğru sıralanır. En büyük kenar seçilir, eşleme kümesinin içine konur ve kenarın bağlı olduğu düğüme bağlı olan bütün kenarlar silinir. Çizgede kenarların hepsi bitene kadar algoritma çalışır. Şekil 2.13'de Preis'ın algoritmasının ve en büyük eşlemenin karşılaştırılması örneklerle gösterilmiştir; sol taraftaki örnekler üzerinde Preis'ın algoritması uygulanmış, sağ taraftaki örnekler üzerinde en büyük ağırlıklı eşleme uygulanmıştır. Şekil 2.13 (a)'da Preis'ın örnek bir çizge üzerinde çalıştırılması gösterilmiştir. İlk önce e_{cd} düğümü seçilmiş, daha sonra da e_{ab} düğümü seçilmiş, böylece toplamda 10+2ε ağırlığında kenar seçilmiştir. Bu duruma karşın Şekil 2.13 (b)'de gösterildiği üzere en büyük ağırlıklı eşlemenin değeri 20'dir. Preis'ın algoritmasının bu örnek üzerindeki yaklaşım oranı yaklaşık olarak 1/2 dir. Bu örnek Preis için en kötüye yaklaşan durumu gösteren örneklerden biridir. Şekil 2.13 (c)'deki örnekte Preis'ın algoritmasının seçtiği kenarların toplamı 19 dur. Buna karşılık gelen Şekil 2.13 (d)'de en büyük eşlemenin seçtiği toplam kenar ağırlığı 20 olduğu için Preis'ın algoritmasının bu örnek üzerindeki yaklaşım oranı 19/20'dir.

Drake'in 2/3 yaklaşımlı merkezi algoritması Şekil 2.14'de verilmiştir. Bu algoritma arttırılan patikalar (*augmenting paths*) kavramını kullanarak toplam seçilen kenar ağırlığını arttırmayı hedefler. Arttırılan patika kavramı Şekil 2.15'de örnek bir uygulaması verilmiştir, sol tarafta düğüm 1'den düğüm 4'e doğru kalın ile gösterilen seçilen kenarların ağırlığı toplamda 14 olurken aynı patika üzerinde sağ tarafta başka kenarlar seçilmiş ve toplam ağırlık olarak 16 elde edilmiş ve böylece patika üzerindeki ağırlık arttırılmıştır.

Drake'in algoritması(G=(V,E), w, M)
M'yi maksimal yap.
$M' \leftarrow M$
$e \in M$ olmak üzere bütün e kenarları için
  Eğer M''de e'nin merkez olduğu bir β-arttırılmış yolu varsa M''nü bu şekilde arttır.
döngüyü bitir
M'yi döndür

Şekil 2.14 Drake'in algoritması



Şekil 2.15 Drake'in algoritması

Hoepman'ın algoritması yönsüz çizgeler üzerinde ağırlıklı eşleme yapan 1/2 yaklaşımlı dağıtık bir algoritmadır (Hoepman, 2004). Bu algoritma Preis'ın merkezi algoritmasının dağıtık tasarlanmış sürümüdür. Hoepman'ın algoritması, her düğümün kendine bitişik olan kenarlarının ağırlığını bildiği varsayar. Her düğümün bitişik olduğu en ağır kenar üzerindeki komşusuna eşleme adayı (*candidate*) denir. Algoritma her düğümün adayına *İSTEK* (*REQUEST*) göndermesiyle başlar. Birbirlerinin adayı olan iki düğüm birbirlerine istek gönderirse eşlenirler. Eşlenmiş bir düğüme istek mesajı gelirse *RED* (*DROP*) mesajı ile cevap verir. Algoritmanın mesaj karmaşıklığı O(N), zaman karmaşıklığı O($|E|$)'dir. Algoritmanın kodu Şekil 2.16'da verilmiştir.

Hoepman'ın algoritması
$R \leftarrow 0$
$N \leftarrow \Gamma(v)$
$c \leftarrow$ aday(v,N)
Eğer $c \neq Ø$'se c'ye *İSTEK* gönder.
$N \neq Ø$ olmadığı sürece
  Mesaj almayı bekle (u düğümünden m mesajını al)
  Eğer m=*İSTEK*'se, $R \leftarrow R \cup \{u\}$
  Eğer m=*RED*'se $N \leftarrow N \setminus \{u\}$
    Eğer $u = c$'se $c \leftarrow$ aday(v,N)
      Eğer $c \neq Ø$'se c'ye *İSTEK* gönder.
  Eğer $c \neq Ø$'se ve $c \in R$ ise bütün $w \in N \setminus \{c\}$ için w'ya *DÜŞME* gönder.
    N $\leftarrow$0
döngüyü bitir.
{ if $c \neq Ø$'se ise $(v,c) \in M$ }

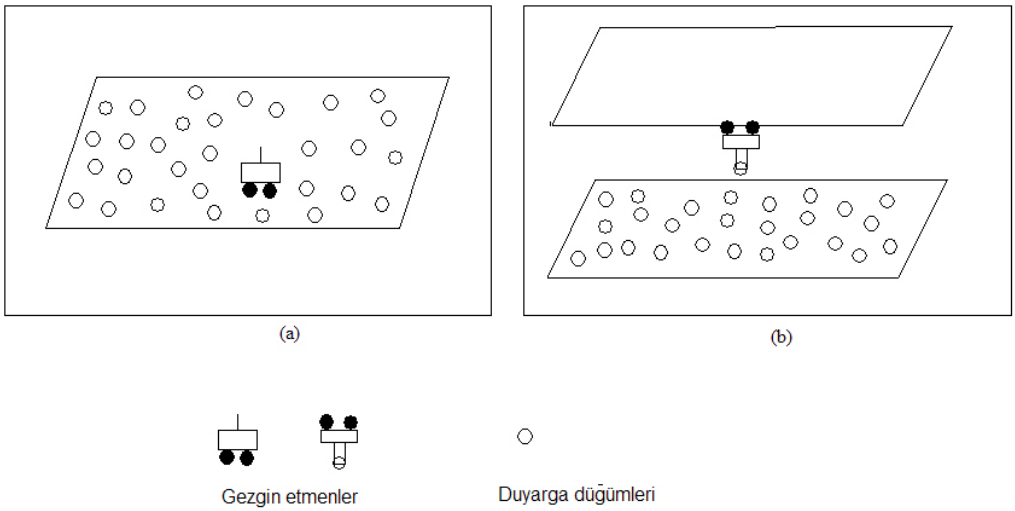
Şekil 2.16 Hoepman'ın algoritması

2.3. Konumlandırma Algoritmaları

Konumlandırma algoritmaları dağıtık ve merkezi olarak iki grupta ayırabiliriz. Niculescu'nun algoritması (Niculescu and Nath, 2001), Savvides'in algoritması (Savvides et al., 2002), Savarese'nin algoritması (Savarese et al., 2002) çok zıplamalı TDA için üç önemli yaklaşımdır (Langendoen and Reijers, 2003). Niculescu'nun yöntemine göre kök düğümler ağa konum bilgilerini yayar. Kök düğümler birbirleri arasındaki en kısa yolu bu mesajlar sonucundan öğrendikten sonra ağa tekrardan bir mesaj yayıp bu bilgiyi gönderirler. Sıradan düğümler bu bilgiyi kullanarak çoklulaterasyon (*multilateration*) tekniğiyle konumlarını bulurlar. Çoklulaterasyon tekniğinde üç veya daha fazla kök düğümden gelen zaman veya sinyal şiddeti bilgileri uzaklığa çevrilip konum bilgileriyle birlikte konumu bilinmeyen düğümün konumu hesaplanır.

Savvides, Niculescu'dan farklı olarak sadece bir grup kök düğümü referans olarak kullanmamış aynı zamanda konumunu hesaplayan her sıradan düğümünde referans olabileceğini önermiştir (Savvides et al., 2002). Bu tekniğe yenilemeli çoklulaterasyon (iterative multilateration) denir. Savarese bu tekniğe çok benzer bir yöntem önermiştir (Savarese et al., 2002). Savarese'nin yönteminde bir düğümün referans düğümü olabilmesi için iletişim kanalları kesişmeyen en az üç farklı kök veya referans düğümden konum bilgisini alması gerekir.

Merkezi algoritmalar kendi içinde iki gruba ayrılabilir. (Doherty et al., 2001) ve (Shang et al., 2003)'da önerilen ilk grup merkezi algoritmalarda sıradan düğümlerden çıkış düğümüne doğru veriler gönderilir ve veriler çıkış düğümünde işlenip, konum bilgileri ağa yayılır. İkinci gruptaki algoritmalarda çıkış düğümünün görevini gören; ağı gezme, kablosuz haberleşme üzerinden veri toplama, topladığı verileri kaydetme ve işleme kabiliyetleri bulunan, bunların yanında yüksek enerjiye sahip gezgin donanım etmenleri (*Mobile Hardware Agents*) (Pathirana et al., 2005) sıradan düğümleri konumlandırır. Pathirana donanım etmenlerinin kullanılmasıyla birlikte düğümlerin üzerinde konumlandırma işlemi için çalışan işlemlerin azaltılacağını belirtmiştir (Pathirana et al., 2005). Ayrıca etmen hareketli olduğu için tükenme gürültüsünün (*fading noise*) zaman içerisindeki ölçümlerle azaltılacağını belirtmiştir. Bunlara ek olarak konumlandırma işlemi etmen tarafından çalışacağı için sıradan düğümler üzerinde çalışan işlemlerden çok daha karışık olabilir. Bu bilgilerden yola çıkarak Pathirana, kalman filtresi tabanlı konumlandırma algoritması tasarlamıştır. Algoritma çok az sayıda düğüm için çalıştırılmış ve gerçek konum bilgilerine göre yaklaşık 1m'lik hata üretmiştir. Şekil 2.17 (a)'da Pathriana'nın yaklaşımı görülmektedir (Dagdeviren et al., 2010).



Şekil 2.17 (a) Pathirana'nın yaklaşımı (b) Lee'nin yaklaşımı

Lee, TDAda konumlandırma yapmak için görüntü tabanlı bir teknik önermiştir (Lee et al., 2006). Pathriana'nın yaklaşımından farklı olarak, Şekil 2.17 (b)'de etmenin hareket düzlemi düğümlerin üzerine oturduğu düzleme paraleldir. Ayrıca Pathirana'nın yaklaşımından farklı olarak donanım etmenin üzerinde görüntü alabilmesi için bir kamera bulunması gerekir. İki farklı yaklaşım ele alınmıştır: etmenin genel (*global*) konumunu bildiği durum ve bilmediği durum. Bunların yanında komşu keşif yaklaşımları önerilmiş ve ağın topolojisinin bulunmasında kullanılmıştır.

3. EŞLEME TABANLI DAĞITIK KÜMELEME VE OMURGA OLUŞTURMA ALGORİTMALARI

Bu bölümde tasarladığımız senkron ve asenkron dağıtık kümeleme ve omurga oluşturma algoritmaları verilmiştir. Öncelikle Hoepman'ın algoritmasının TDA üzerinde uyarlanmış versiyonu gösterilip, daha sonra bu algoritmayı kullanarak tasarladığımız eşleme tabanlı kümeleme ve omurga oluşturma algoritmalarının tasarım detayları, analizleri, performans değerlendirmeleri ve uygun olabilecek TDA uygulamaları verilmiştir.

3.1. Hoepman'ın Algoritmasının TDAya uygulanması

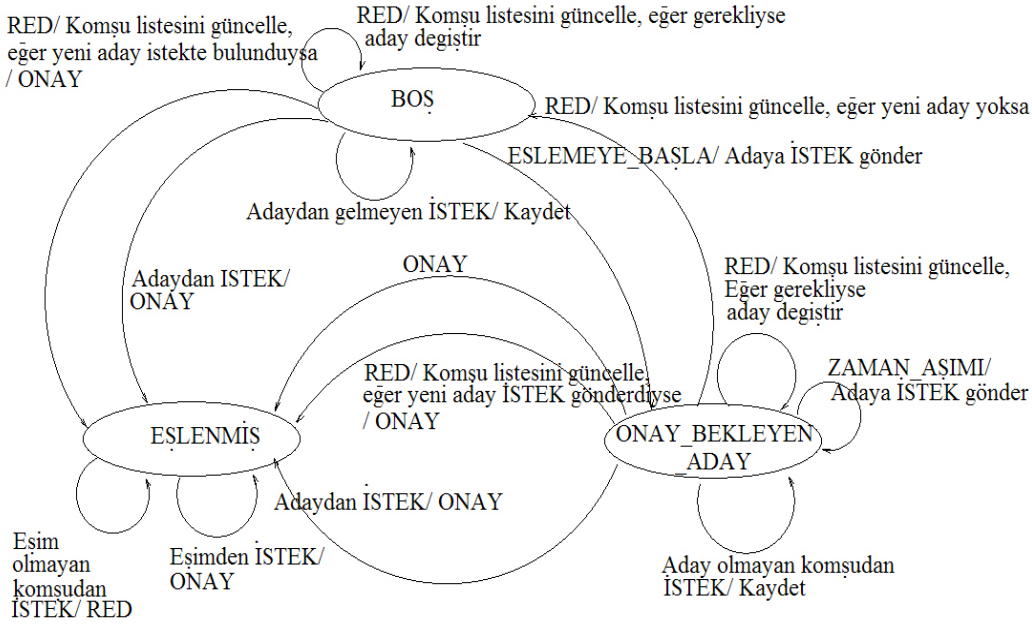
Eşleme tabanlı kümeleme algoritmalarını tasarlamadan önce Hoepman'ın algoritmasını TDA için tasarladık. Bu algoritmanın basamakları Şekil 2.16'da verilmiştir. Şekil 3.1'de TDA ortamı için uyarladığımız Hoepman'ın algoritmasının sonlu durum makinesi görülmektedir. Hoepman'ın algoritmasında eşlenmiş düğüm dışındaki bütün komşulara *RED* mesajı gönderilmesi gerekebilir. Kablosuz ortam için uyarlanmış bu tasarım sayesinde mesajın tüm komşulara yayılma avantajını göz önüne alıp tek bir *RED* mesajına dönüştürdük. Durumları ve mesajları sırayla şu şekilde inceleyebiliriz:

BOŞ (IDLE): Bu durumdaki düğüm üst katmandan gelen *EŞLEMEYE_BAŞLA (START_MATCH)* mesajıyla birlikte eşleme aday düğümüne *İSTEK(düğüm_kimliği, hedef_düğüm_kimliği) (REQUEST)* mesajı gönderdikten sonra *ONAY_BEKLEYEN_ADAY (CANDIDATE_WT_ACK)* durumuna geçip algoritmayı çalıştırmaya başlar. *EŞLEMEYE_BAŞLA (START_MATCH)* mesajı gelmeden önce *RED(düğüm_kimliği) (DROP)* mesajı gelirse mesajın kaynağını komşu listesinden siler, *İSTEK* mesajı gelirse de kaydeder.

ONAY_BEKLEYEN_ADAY: Bu durumdaki düğüm aday düğümünden *ONAY* bekler. Eğer aday düğümünden *ONAY(düğüm_kimliği, hedef_düğüm_kimliği)* alırsa *EŞLENMİŞ (MATCHED)* durumuna geçer. *RED* mesajı alırsa, bu mesajın kaynağını komşu listesinden siler ve yeni aday düğüm seçip, ona *İSTEK* gönderir. *RED* mesajı alan düğümün adayı kalmazsa *EŞLENMİŞ* durumuna geçer.

EŞLENMİŞ: Bu durumdaki düğüm adayıyla birlikte eşlenmiş düğümdür. Çevreden gelen *İSTEK* mesajlarına *RED* mesajıyla cevaplar.

Bu algoritmanın ns2 benzetim sonuçları Bölüm 3.5.1'de verileceği üzere çok iyi çıkmıştır. Bundan dolayı bu algoritmanın diğer algoritmalarımızın tasarım temelinde olması uygun görülmüştür.



Şekil 3.1 Hoepman'ın algoritmasının TDA için tasarlanmış sonlu durum makinesi

3.2. Eşleme Tabanlı Senkron Kümele Algoritması

Bu bölümde eşleme tabanlı senkron kümeleme algoritmasının (*matching based synchronous clustering algorithm* (*MASC*)) (Dagdeviren and Erciyes, 2010) genel fikri, tasarımı ve örnek uygulaması verilmiştir.

3.2.1. Genel Fikir

MASC, kümeleme işlemini büyükçe ağırlıklı maksimum eşleme kullanarak yapar. Algoritmanın en önemli hedefi yönsüz kenar ağırlıklı çizge modelindeki güçlü iletişim kanallarını seçerek küme içi iletişim kalitesini arttırmaktır. MASC algoritması rauntlara bölünmüş olarak çalışır ve yeni bir raunt başlamadan önce algoritmayı çalıştıran bütün düğümler eski raunt içinde yapmaları gereken görevi tamamlar. Her rauntta küme liderleri maksimum kenar ağırlığı ile bağlı olduğu kümeyle birleşmek için istekte bulunur. Her rauntta düğümlerin ağır kenarlar üzerinden birleşmesi sayesinde yalnızca küme içi iletişim kalitesi yükseltilmez aynı zamanda özellikle BHK algoritmalarının işletilmesi sonucunda karşımıza sıkça çıkan gereksiz kümelerin sayısı azaltılmış olur. Kümeleme algoritmamızda her rauntta çalıştırdığımız ağırlıklı eşleme algoritması Hoepman'ın TDA için uyarlanmış ağırlıklı eşleme algoritmasının genişletilmiş versiyonudur. Hoepman'ın algoritması sadece yerel (*local*) mesaj değişimlerine ihtiyaç duyduğundan dolayı TDA için uygundur. Ayrıca yaklaşım oranı 1/2 olmasına rağmen, az sayıda mesaj tüketimi ve zaman harcamasıyla birlikte rastgele üretilen TDAlar için benzetim sonucunda elde edilen yaklaşım oranlarının çok yüksek olduğu Bölüm 3.5.1'de gösterilmiştir. Nieberg'in algoritmasının yaklaşım oranı 1-ε olmasına rağmen; bu algoritmada

maksimum ağırlıklı eşlemenin bulunabilmesi için öncelikle bağımsız hakim küme bulunması gerekmektedir. Bu durumdan dolayı duyarga ağlarında kümeleme için bu algoritmayı tekrarlı bir şekilde uygulandığında zaman ve mesaj karmaşıklığı çok artabileceğinden bizim tasarımımızda çok uygun görünmemektedir.

3.2.2. Tasarım

MASC'ı tasarlarken Bölüm 1'de verilen varsayımlara ek olarak düğümlerin rauntlara aynı zamanda başlayabilmeleri için zaman senkronizasyonu yaptıklarını kabul ettik. Kenar ağırlıkları alınan sinyal gücü (RSS) değerleriyle temsil edildi. Algoritmamızda düğümleri birbirine bağlayan kenarlar üzerinden mesajların alınması ve bu mesajların işlenmesi ile çalışıyor. Algoritmamızın sonlu durum makinesi Şekil 3.2'de verilmiştir.

Bir kümeleme oturumunda, kaynak düğüme en ağır kenar ağırlığı ile bağlı olduğu hedef düğüme *İSTEK* mesajı gönderir. Kaynak düğüme en ağır kenardan bağlanan hedef düğüm kaynak düğümün kümeleme işleme için adayıdır. Aynı zamanda kaynak düğüm de hedef düğümünün adayı ise, hedef düğüm kaynak düğüme *ONAY* mesajı gönderir ve böylece kümeleme işlemi tamamlanmış olur. Eğer hedef düğüm aynı raunt içinde başka bir düğümle daha önceden kümelenmişse kaynak düğüme *RED* mesajı gönderir. Kaynak düğüm *RED* mesajını alınca yeni bir aday belirler ve aynı işlemleri tekrardan yapar. Bu mekanizma Hoepman'ın algoritmasıyla benzerdir. Bizim eklentimiz ve katkımız düğümleri küme lideri ve küme üyesi olacak şekilde sınıflandırmak ve türlerine uygun şekilde sorumluluklar atamak ve böylece güçlü bir küme topolojisi oluşturmaktır.

Algoritmanın durumları ve mesajları aşağıda madde listelenmiştir:

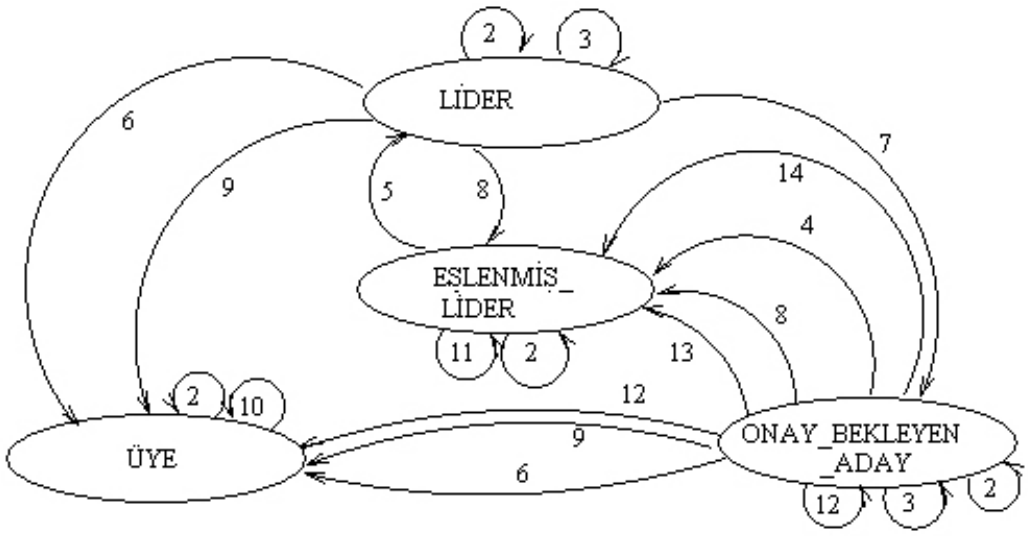
- *LİDER* (*LEADER*): *LİDER* durumundaki bir düğüm kendi kümesinin başıdır. Algoritma çalıştırılmadan önce her düğüm *LİDER* durumundadır. Her düğümde yeni raunt başladığı zaman *YENİ_RAUNT* (*NEW_RAUNT*) kesmesi (*interrupt*) olur. *LİDER* durumdaki bir düğüme *YENİ_RAUNT* olursa adayına *İSTEK(düğüm_kimliği, hedef_düğüm_kimliği, kümeye komşu elemanların RSS listesi, küme elemanlarının listesi)* mesajı gönderir. *LİDER* durumundaki bir düğüm adayından *İSTEK* mesajı alırsa, *ONAY(düğüm_kimliği, hedef_düğüm_kimliği, kümeye komşu elemanların RSS listesi, küme elemanlarının listesi)* ile cevap verir ve yeni küme oluşturulur. Bu kümeleme işleminin sonucunda düğümlerden biri *EŞLENMİŞ_LİDER* (*MATCHED_LEADER*) durumuna geçerken, diğer düğüm *ÜYE* (*MEMBER*) durumuna geçer. Düğümler geçecekleri duruma şu algoritmayı işleterek karar verir:

1. RSS_i $düğüm_i$'nin en büyük ikinci RSS değeri, RSS_j'ise $düğüm_j$'nin en büyük ikinci RSS değeri olsun. $Düğüm_i$'nin kimliği $kimlik_i$, $düğüm_j$'nin kimliği $kimlik_j$ olsun.

2. Büyük RSS değerine sahip düğüm *EŞLENMİŞ_LİDER* durumuna geçerken diğer düğüm *ÜYE* durumuna geçer.

3. RSS değerleri eşitse, büyük kimlikli düğüm *EŞLENMİŞ_LİDER*, diğer düğüm *ÜYE* durumuna geçer.

Bu lider seçim metodunu kullanmamızın nedeni, bir sonraki aşamada, RSS değeri daha büyük olan düğüm çevresinde kümelenme işleminin olmasının sezilmesidir. *LİDER* düğüm kendi kümesindeki diğer bütün üyelerin komşularını bilir ve her bir kümeleme işleminden sonra listesini işleme katılan diğer düğüm ile kaynaştırır (*merge*).



1: RED/ Komşu listesini güncelle
eğer gerekliyse aday değiştir, İSTEK

2: Eşimden İSTEK/
ONAY

3: Aday olmayan düğümden
İSTEK/ Kaydet

4: RED/ Komşu listesini gönder
Eğer yeni aday daha önce İSTEK
göndermişse ve lidersem, ONAY

5: RAUNT_SONU

6: RED/ Komşu listesini güncelle
Eğer yeni aday İSTEK göndermişse, ONAY

7: YENİ_RAUNT/ Önceki RED bilgisini
sil, adaya İSTEK

8: Adaydan İSTEK /
Eğer lidersem, ONAY

9: Adaydan İSTEK/
ONAY

10: Eşim olmayan düğümden
ISTEK/ İSTEK'i
liderime ilet.

11: Eşim olmayan düğümden
İSTEK/ RED

12: Daha büyük RSS'li
ONAY

13: Daha küçük RSS'li
ONAY

14: RED/ Komşu listesini güncelle
eğer aday yoksa

Şekil 3.2 MASC'ın sonlu durum makinesi

- *EŞLENMİŞ_LİDER*: *EŞLENMİŞ_LİDER* durumundaki düğüm, yeni raunt başlayana kadar kendine gelen kümeleme isteklerini *RED(düğüm_kimliği)* mesajı göndererek yanıtlar. Yeni raunt başladığında, *LİDER* durumuna geçer.

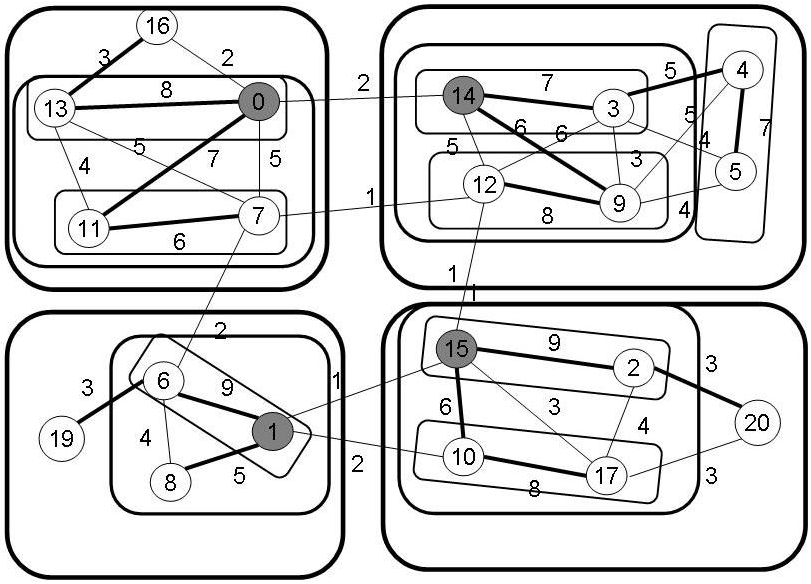
- *ÜYE*: Algoritmamızda küme üyeleri ÜYE durumundadır ve kendilerine gelen küme isteklerini liderlerine yönlendirirler.

- *ONAY_BEKLEYEN_ADAY*: Bir raundun başında, her *LİDER* düğüm kendi adayına *İSTEK* mesajı gönderir ve ONAY_*BEKLEYEN_ADAY* durumuna geçer. Eğer bir *RED* mesajı alırsa, yeni aday seçip *İSTEK* mesajı gönderir. Eğer başka bir aday bulunmazsa, *EŞLENMİŞ_LİDER* durumuna geçiş yapar.

MOD-MASC (Modified MASC) algoritması ağırlıklı eşleme tekniğini yenilemeli olarak kullanarak kümeleri güçlü kanallar seçerek oluşturur. Ağırlıklı eşleme tekniğiyle ağırlıksız eşleme tekniğinin kümeleme işlemi üzerindeki etkisini göstermek amacıyla MASC algoritmasının değiştirilmiş versiyonu olan MOD-MASC algoritması önerilmiştir (Dagdeviren and Erciyes, 2010). MOD-MASC algoritmasında ağır kenarlar seçilmek yerine düğüm kimliklerine bağlı olarak eşleme yapılmıştır. MOD-MASC algoritmasında bir düğüm eşleme yaparken en küçük kimlikli komşusunu aday olarak seçer.

3.2.3. Örnek Uygulama

Şekil 3.3'de 20 düğümlük bir duyarga ağı, kenar ağırlıkları ve düğüm kimliklerinle birlikte verilmiştir. Bu ağ üzerinde MASC algoritmasını kullanarak 3 raunt süren örnek bir kümeleme işlemi yapacağız. İlk rauntta, 13 kimlikli *LİDER* durumdaki düğümde *YENİ_RAUNT* kesmesi oluşur, adayı olan düğüm 0'a *İSTEK* mesajı gönderir ve *ONAY_BEKLEYEN_ADAY* durumuna geçer. Düğüm 0, düğüm 13'den gelen *İSTEK* mesajını almadan önce *LİDER* durumundadır ve adayı düğüm 13'dür, bundan dolayı düğüm 0 *ONAY* mesajı ile düğüm 13'e cevap verir. Düğüm 13'un ikinci RSS değeri 5 olduğunda, düğüm 0'ın ikinci RSS değeri 7 dir, bundan dolayı düğüm 0 *EŞLENMİŞ_LİDER* durumuna geçiş yaparken, düğüm 13 küme lideri durumuna geçiş yapar. Bu işlemlere eş zamanlı olarak düğüm 11 düğüm 0'a *İSTEK* mesajı gönderir. Düğüm 11'in *İSTEK* mesajı düğüm 0 tarafından *RED* mesajı gönderilerek reddedilir. Düğüm 14 ve düğüm 3, düğüm 4 ve düğüm 5, düğüm 12 ve düğüm 9, düğüm 15 ve düğüm 2, düğüm 10 ve düğüm 17, düğüm 6 ve düğüm 1, düğüm 11 ve düğüm 7 birinci rauntta yukarıda anlatılan işleme benzer şekilde kümeleme işlemini yaparlar. Bunun yanında düğüm 16 ve düğüm 8'in bütün istekleri adayları tarafından düşürülür, bundan dolayı düğüm 16 ve düğüm 8 kümeleme işlemi yapabilecekleri başka bir düğüm bulamazlar.



Şekil 3.3 MASC'ın örnek bir uygulaması

İkinci raundun başında, *LİDER* durumunda olan düğüm 12 düğüm 3'e *İSTEK* mesajı gönderir. *ÜYE* durumundaki düğüm 3 *İSTEK* mesajını aldıktan sonra kendisinin lideri olan düğüm 14'e iletir. Düğüm 14, *İSTEK* mesajını *ONAY* ile cevaplar çünkü kendi kümesinden çıkan en ağır kenarın ucunda düğüm 12 vardır. Düğüm 12, *ONAY* mesajını aldıktan sonra yeni kümesinin içindeki elemanlar şu şekildedir: {14, 3, 12, 9}. İkinci raundun sonunda, diğer kümeler {13, 0, 11, 7}, {6, 1, 8}, {15, 10, 2, 17} şeklinde oluşmuş olur. Üçüncü raunttaki kümeleme işlemleri de ilk iki raunttaki işlemlere çok benzerdir ve kümeler şu şekilde oluşulur: {14, 3, 12, 9, 4, 5}, {6, 1, 8, 19}, {13, 0, 16, 11, 7}, {15, 10, 2, 17, 20}. En son durumda Şekil 3.3'de küme liderleri gri ile boyanmış olup, iletişim için seçilen kenarlar diğer kenarlara göre daha kalın olarak çizilmiştir.

3.3. Eşleme Tabanlı Asenkron Kümele ve Omurga Oluşturma Algoritması

Bu bölümde eşleme tabanlı asenkron kümeleme algoritmasının (*matching based asynchronous clustering algorithm* (*MCUBA*)) (Dagdeviren and Erciyes, 2010) genel fikri, tasarımı ve örnek uygulaması verilmiştir.

3.3.1. Genel Fikir

MASC algoritması ağırlıklı çizge eşleme tekniğini kullanarak güçlü kenarları kümeleme işlemi sırasında literatürdeki diğer çizge teorik kümeleme algoritmalarından farklı olarak seçmiştir. Ayrıca MASC'da üretilen küme sayısı raunt sayısıyla kontrol edilebilirdir ve bu yönüyle BHK algoritmalarına göre avantajlıdır. MASC'ın literatürdeki çizge teorik algoritmalara göre bütün avantajları

Bölüm 3.4 ve 3.5'de gösterilmiştir. Buna karşın MASC algoritmasının eksik olan yönleri şunlardır:

1. Algoritma senkron olarak çalışmaktadır. Bu durumun sağlanabilmesi için düğümlerin zaman senkronizasyonu (Ganeriwal et al., 2003) yapması veya senkronizer arakatman (Rao, 2008) kullanması gerekmektedir.

2. Algoritmanın tükettiği enerji miktarı literatürdeki çalışmalara oranla daha fazla olduğu Bölüm 3.5'te gösterilmiştir.

3. Omurga oluşumu üst katmandaki bir protokole bırakılmıştır.

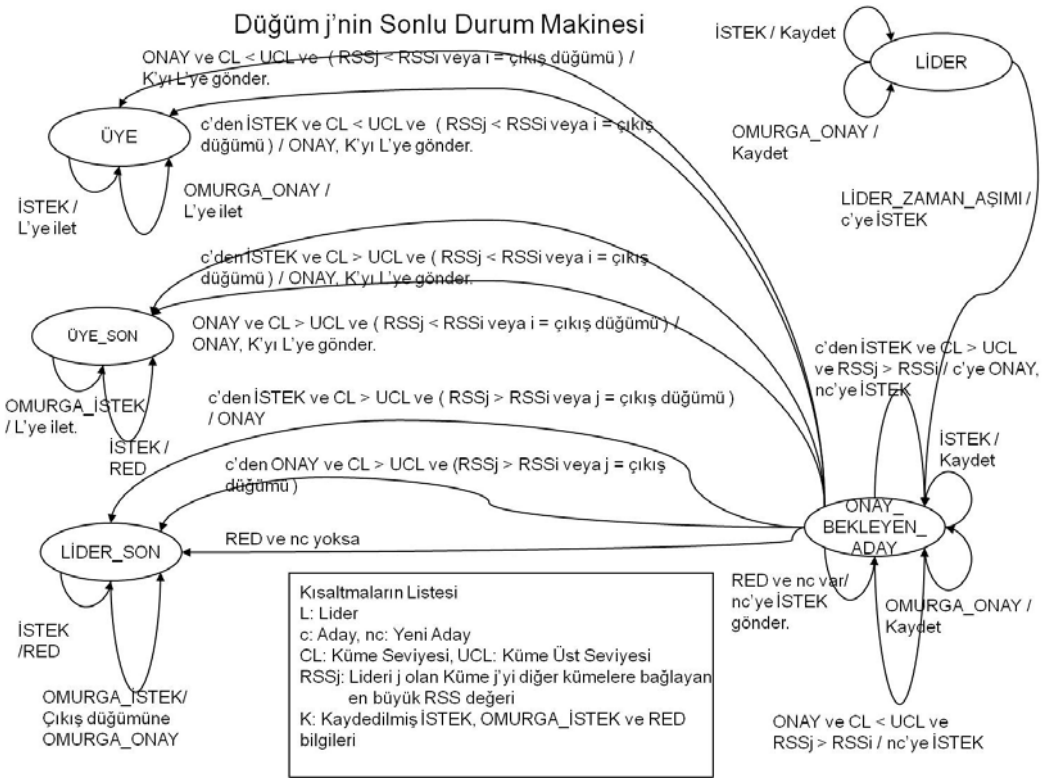
MASC algoritmasının üstünlüklerini koruyarak, eksiklerini gidermek için MCUBA'yı öneriyoruz (Dagdeviren and Erciyes, 2010). MASC algoritmasında ağırlıklı eşleme ana fikri koruyoruz, asenkron çalışma için gerekli algoritmik eklentiler yapıyoruz, enerji tüketimini düşürmek için yöntemler ekliyoruz ve omurga oluşturmak için kümelemenin çok benzeri bir fikir sunuyoruz. Algoritmanın detaylı açıklaması bir sonraki bölümde gösterilmiştir.

3.3.2. Asenkron Küme ve Omurga Oluşturma

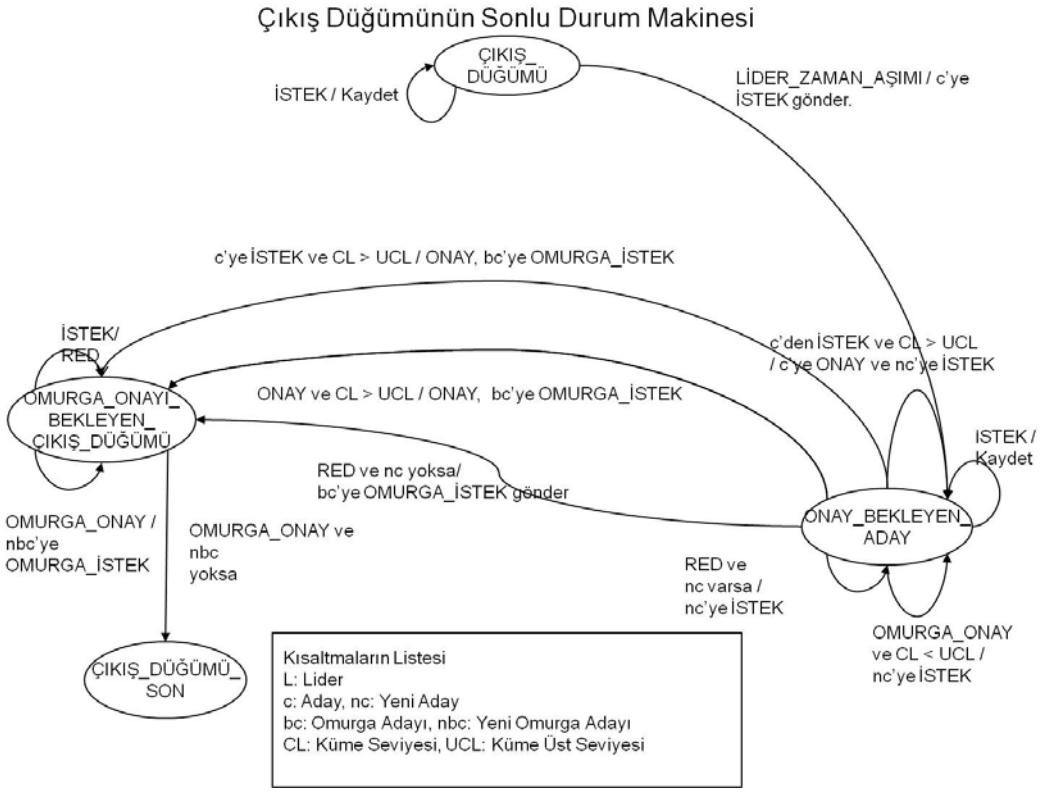
MCUBA algoritmasında asenkron kümeleme ve omurga oluşturma işlemini yapabilmek için iki adet sonlu durum makinesi öneriyoruz. Şekil 3.4'de görülen birinci sonlu durum makinesini sıradan düğümler eşleme tabanlı kümeleme yapmak için kullanırlar. Algoritmada küme dengesinin sağlanması için *küme_üst_seviyesi* (*upper_cluster_level*) parametresi eklenmiştir; bir kümenin içindeki eleman sayısı o kümenin seviyesidir. Öncelikle her düğüm *LİDER* durumundadır. Düğümlerde *LİDER_ZAMAN_AŞIMI* (*LDR_TOUT*) oluştuktan sonra en güçlü kenarla bağlı oldukları adaylarına *İSTEK* gönderip *ONAY_BEKLEYEN_ADAY* durumuna geçer. Aday düğüm de kendi adayından *İSTEK* mesajı alırsa *ONAY* ile cevaplar. *ONAY* mesajını alan düğüm *ONAY_BEKLEYEN_ADAY* durumundan *LİDER*, *LİDER_SON* (*LEADER_END*), ÜYE veya *ÜYE_SON* (*MEMBER_END*) durumlarına geçebilir. *LİDER* seçim kıstası MASC ile aynıdır. LİDER düğümün küme seviyesi, küme içerisindeki eleman sayısıdır.

Düğümler adayları dışında *İSTEK* mesajı alırsa bunu bir tabloya kaydeder. Çünkü küme oluşumundan sonra düğüm kendine yeni bir aday bulacaktır. Bu aday düğüm de önceden *İSTEK* göndermiş olabilir. Önceden *İSTEK* göndermiş yeni adaya *ONAY* ile cevap verilir ve yeni kümeleme işlemi yapılmış olur. *LİDER* bir düğümün yeni adayı ona önceden *İSTEK* göndermemişse yukarıdaki paragrafta anlatılan işlem sırası takip edilir. Bu yaklaşım MASC'a göre farklıdır. MASC'da *LİDER* düğüm bir raunt içinde eşlendikten sonra *EŞLENMİŞ_LİDER* konumuna geçer ve kendisine gelem bütün *İSTEK*leri reddeder. Hâlbuki MCUBA'da bu *İSTEK*ler bir tabloya

kaydedilir ve lider düğüm, kümesi *küme_üst_seviyesi* ne gelene kadar kümeleme işlemine devam eder. Bu seviyeye geldikten sonra gelen bütün kümeleme isteklerini düşürür. Bu yaklaşım tekrarlanan *İSTEK*leri ve RED mesajlarını eleyerek enerji etkinliği sağlar.



Şekil 3.4 MCUBA sıradan düğümün sonlu durum makinesi



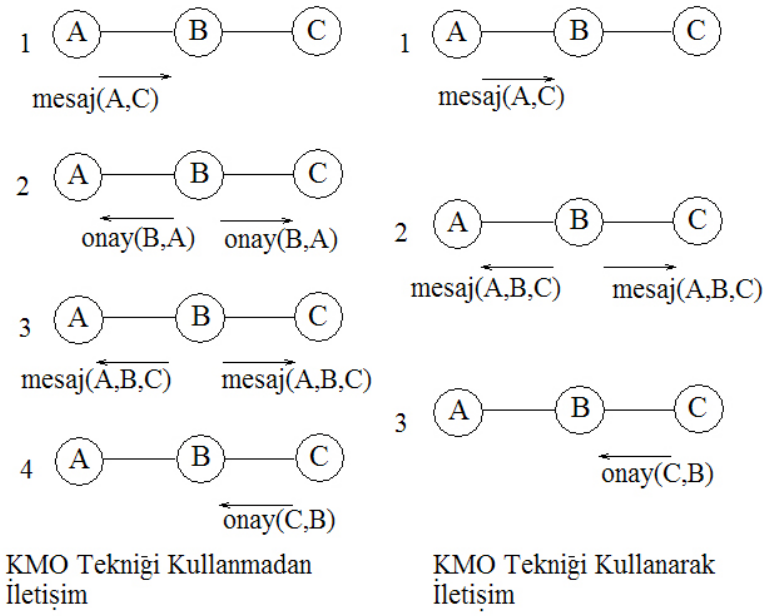
Şekil 3.5 MCUBA çıkış düğümün sonlu durum makinesi

Omurga oluşturma işlemi kümeleme işlemine çok benzemektedir. Kümeleri birleştirip en üst seviyede tek bir bölüm (*segment*) yapılması fikrine dayanmaktadır, bu sırada kümelerin oluşma işlemi asenkron olarak devam edebilir. Şekil 3.5'de görülen ikinci sonlu durum makinesinde çıkış düğümünün lider düğümlerle küme ve omurga oluşturmasını göstermektedir. Çıkış düğümü sıradan düğümler gibi kümeleme işlemine katılır. Çıkış düğümünün kümeleme işleminin diğer düğümlerin kümelemesinden tek farkı çıkış düğümünün kümeleme sırasında da sürekli lider olmasıdır. Çıkış düğümünün kümesi belirli bir seviyeyi aştıktan sonra, çıkış düğümü omurga oluşturma işlemine başlar. Bu sırada diğer düğümler kümeleme işlemine devam edebilirler. Çalışma tamamen asenkrondur. Çıkış düğümü omurga oluşturma işlemine *OMURGA_İSTEK* (*BACKBONE_REQUEST*) mesajını en ağır kenarından göndermekle başlar. *OMURGA_İSTEK* mesajını alan düğüm eğer *LİDER_SON* durumuna ulaşmışsa *OMURGA_ONAY* (*BACKBONE_REQUEST_ACK*) ile cevap verir. Eğer ulaşmamışsa bekler. Çıkış düğümüne *OMURGA_ONAY* mesajı ulaştıktan sonra çıkış düğümü kendi kümesi ile birlikte cevabını aldığı kümeyi omurgaya dâhil eder ve bir sonraki aşamada en ağır kenarı üzerinden *OMURGA_İSTEK* mesajı gönderir. Bu işlem bütün kümeleri kapsayana kadar devam eder. En son durumda bütün kümeler bir bölümün içine dâhil olur. MCUBA'nın sonlu durum makinesinde detaylı sadece önemli geçişler gösterilmiştir.

3.3.3. Enerji Tüketimindeki Azaltmalar

MASC algoritmasında mesajların güvenli iletimi için *ACK* (*ALT_SEVİYE_ONAY*) mesajları kullanılmıştır. MCUBA algoritmasında *ALT_SEVİYE_ONAY* mesajlarını azaltmak ve böylece algoritmanın enerji tüketimi düşürmek için kulak misafiri olmak (*overhearing*) tekniği kullanılmıştır. Kulak misafiri olma (KMO) tekniğinde pasif onay mesajları kullanılarak toplam onay mesajlarının sayısı azaltılır. KMO tekniğinin onay mesajlarını elemek için kullanılması Şekil 3.6'da gösterilen kablosuz ağ topolojisi üzerine anlatılmıştır. Topolojiye göre *A* düğümünün komşusu *B* düğümü, *B* düğümünün komşuları *A* ve *C* düğümleri, *C* düğümünün komşusu da sadece *B* düğümüdür. İletim ortamı kablosuz olduğu için bir düğümden çıkan mesaj tüm komşulara ulaşır. Şekil 3.6'da sol taraftaki bölümde KMO tekniği kullanmadan *A* düğümünden çıkan ve hedefi *C* olan bir mesajın *C* düğümüne iletilmesi sırasındaki basamaklar görülmektedir. İlk adımda *A* düğümü *B* düğümüne mesajını gönderir. İkinci adımda *B* düğümü bu mesajı aldığına dair onay mesajını *A* ve *C* düğümlerine gönderir ve bir sonraki adımda *A* düğümünden aldığı mesajı *C* düğümüne iletir. Son adımda *C* düğümü, *B* düğümüne mesajı aldığına dair cevap gönderir. Şekil 3.6'nın sağ bölmesinde KMO tekniği kullanarak aynı mesajlaşmanın azaltılması gösterilmiştir. İlk adımda yine *A* düğümü *B* düğümüne, *C* düğümüne iletilecek mesajını gönderir. *B* düğümü *ONAY* mesajı göndermeden *C* düğümüne mesajı iletir. *B* düğümü *C* düğümüne mesajı iletirken *A*

düğümü de mesajı almış olur. *A* düğümün aldığı bu mesaj aynı zamanda onay mesajı yerine geçer ve böylece sol taraftaki mesajlaşma basamaklarından bir adet eksilmiş olur. En son adımda *C* düğümü *B* düğümüne onay göndermek zorundadır.



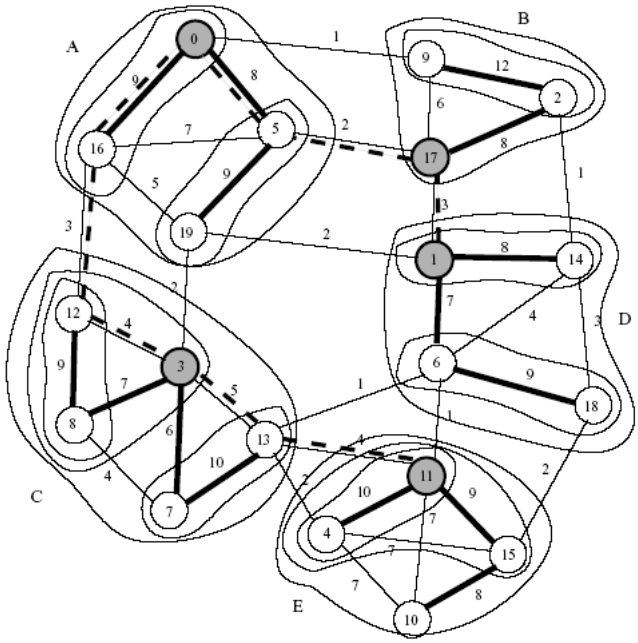
Şekil 3.6 Kulak misafiri olma yöntemi

3.3.4. Örnek Uygulama

Şekil 3.7'de 20 düğümlük bir duyarga ağı, kenar ağırlıkları ve düğüm kimliklerinle birlikte verilmiştir. Bu ağ üzerinde MCUBA algoritmasını kullanarak kümeleme ve omurga oluşturma işleminin göstereceğiz. MCUBA algoritmasının *küme_üst_seviyesi* değeri 4 olarak verilmiştir. Düğüm 0 çıkış düğümü, diğer düğümler sıradan düğümlerdir. Başlangıçta tüm düğümler *LİDER*, düğüm 0 *ÇIKIŞ_DÜĞÜMÜ (SINK)* düğümü durumundadır. Düğüm 0, en ağır kenarı üzerinden adayı olan düğüm 16'ya *İSTEK* mesajı gönderir. Düğüm 16'nın da aday düğümü 0 olduğu için *ONAY* mesajı ile cevap verir. Düğüm 0 çıkış düğümü olduğu için küme lideri olmaya devam eder. Düğüm 5 ve düğüm 19'da benzer şekilde kümeleme işlemini yaparlar. Düğüm 5'in ikinci RSS değeri 8, düğüm 19'un 5 olduğu için düğüm 5 bu kümelemenin sonucunda lider olur. Düğüm 9 ve düğüm 2, düğüm 1 ve düğüm 14, düğüm 6 ve düğüm 18, düğüm 11 ve düğüm 4, düğüm 12 ve düğüm 8, düğüm 13 ve düğüm 7 benzer şekilde iki seviyelik kümeler oluştururlar.

Düğüm 17, kümeleme işlemi yaparken düğüm 2'ye *İSTEK* mesajı göndermiştir. Düğüm 2'nin daha ağır bir kenarı olduğu için düğüm 17'ye *ONAY* mesajı göndermez fakat bu isteği kaydeder. Düğüm 2 kümeleme işlemini lider olarak bitirdikten sonra düğüm 17'ye kümelenmek üzere *ONAY* mesajını gönderir. Böylece düğüm 17'nin liderliğinde {17, 2, 9} kümesi oluşur. Benzer şekilde düğüm 3'ün liderliğinde {3, 8, 12} kümesi ve düğüm 11'in liderliğinde {11, 4, 15} kümeleri

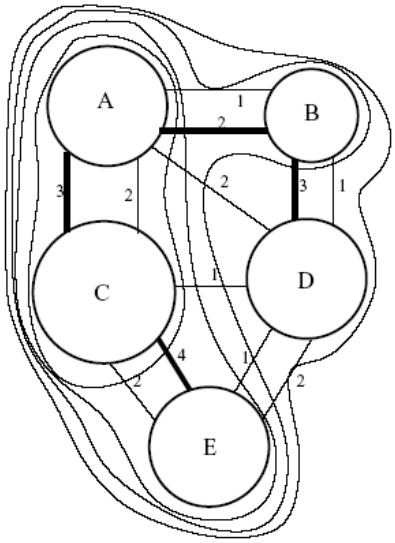
oluşur. Bu sırada düğüm 1, düğüm 6'ya *İSTEK* mesajı gönderir. Düğüm 6'nın adayı düğüm 1 olduğu için *ONAY* mesajı ile cevap verir, böylece düğüm 1'in liderliğinde {1, 14, 6, 18} kümesi oluşturulur. Düğüm 1 *küme_üst_seviyesi* ne ulaştığı için *LİDER_SON* durumuna geçer. *LİDER_SON* durumdaki düğümler kendilerine gelen bütün küme isteklerini reddederler. Benzer bir şekilde düğüm 0'da kümeleme işlemi yaparak düğüm 5 ile kümelenir, böylece çıkış düğümü olan düğüm 0'ın liderliğinde {0, 5, 19, 16} kümesi oluşur. Düğüm 0 çıkış düğümü olduğu ve üst seviyeye ulaştığı için omurga oluşturma işlemine başlar. Düğüm 0'ın kümesinden çıkan en ağır kenar 3 olduğu için ve bu kenarın ucunda düğüm 12 olduğu için düğüm 12'ye *OMURGA_İSTEK*'i gönderir. Düğüm 12 *ÜYE* durumda olduğu için mesajı lideri olan düğüm 3'e yönlendirir. Düğüm 3, kümeleme işlemini bitirmediği için bu mesajı kaydeder ve kümeleme işlemine devam eder.



Şekil 3.7 MCUBA örnek uygulama

Düğüm 0'ın liderliğindeki küme ve düğüm 1'in liderliğindeki kümeler üst seviyeye ulaştıkları için kümeleme işlemini bitirmişlerdir. Bundan dolayı düğüm 17'den gelen *İSTEK*leri *RED* mesajı ile geri çevirirler. Düğüm 17, bütün komşularından *RED* mesajı aldığı için *LİDER_SON* durumuna geçer. Bu sırada düğüm 3 ve düğüm 7 ile birleşerek, düğüm 3'ün liderliğinde {3, 12, 8, 13, 7} kümesini oluşturur. Düğüm 11'de düğüm 10'unu kümesine dâhil ederek son durumda {11, 4, 15, 10} kümesi oluşur. Böylelikle düğüm 0'ın liderliğindeki A kümesi {0, 16, 5, 19} olarak, düğüm 17 liderliğindeki B kümesi {17, 12, 9} olarak, düğüm 11 liderliğinde {11, 4, 15, 10} olarak, düğüm 3 liderliğindeki C kümesi {3, 12, 8, 13, 7} olarak, düğüm 1 liderliğindeki D kümesi {1, 14, 6, 18} olarak, düğüm 11 liderliğindeki E kümesi {11, 4, 15, 10} olarak tamamlanmıştır. Şekil 3.7'de seçilmiş kenarlar kalın biçimde gösterilmiştir. C kümesindeki kümeleme işlemi bittikten sonra düğüm 3, düğüm 0'a *OMURGA_ONAY* mesajı gönderir, böylece A ve

C kümeleri omurganın içine dâhil olurlar. Daha sonra omurga bölümünden çıkan en ağır kenar düğüm 13 ve düğüm 11'i birleştiren kenar olduğu için çıkış düğümü, düğüm 11'e *OMURGA_İSTEK* mesajı gönderir. Düğüm 11, düğüm 0'a *OMURGA_ONAY* gönderir ve böylece A, C ve E kümeleri omurga bölümünde birleşirler. Benzeri bir şekilde önce B daha sonra da D kümesi omurga bölümüne dâhil olur. Omurga işlemi sırasında seçilen kenarlar Şekil 3.7'de kesikli çizgiler ile gösterilmiştir. Omurga oluşumu Şekil 3.8'de özetlenmiştir. Omurga oluşumu sırasında kümeler arasında seçilen kenarlar Şekil 3.8'de kalın çizgiler ile gösterilmiştir.



Şekil 3.8 MCUBA örnek omurga oluşturma uygulaması

3.4. Analiz

Bu bölümde MASC ve MCUBA'nın doğruluk analizi, küme kalitesi, seçilen kenarların kalitesi, mesaj karmaşıklığı, zaman karmaşıklığı ve uzay karmaşıklığı analizleri verilecektir. Ayrıca MCUBA'nın MASC'daki enerji tüketimini nasıl azalttığını gösteren bir analiz verilecektir. Uzay karmaşıklığı yapılırken farklı veri tiplerinin sabit ve birbirleriyle yakın miktarda bellek alanları tuttukları farz edilmiştir.

3.4.1. Doğruluk İspatı

Gözlem (Observation) 3.1. MASC'da ve MCUBA'da iki düğüm arasındaki bir kümeleme oturumu birinci düğümün *İSTEK*, ikinci düğümün *ONAY* veya *RED* mesajlarını göndermesinden oluşur.

Gözlem 3.2. MASC'da *ÜYE* durumundaki düğümler küme üyeleri, *EŞLENMİŞ_LİDER* durumundaki düğümler küme liderleridir. MCUBA'da *ÜYE* ve *ÜYE_SON* durumundaki düğümler küme üyeleri, *LİDER_SON* durumundaki

düğümler küme liderleri, *ÇIKIŞ_DÜĞÜMÜ_SON (SINK_END)* durumundaki düğüm küme lideri olan çıkış düğümüdür.

Gözlem 3.3. MCUBA'nın bir omurga oluşturma işleminde çıkış düğümü tarafından *OMURGA_İSTEK* mesajı gönderilir, *OMURGA_İSTEK* mesajını alan lider düğüm *OMURGA_ONAY* ile cevap verir.

Ön Teorem (Lemma) 3.1. MASC ve MCUBA'da bir düğüm adayına *İSTEK* gönderirse, adayından *RED* veya *ONAY* mesajı alır.

İspat. Teoremin ispatı için karşımıza çıkabilecek durumları sırayla sonlu durum makinesi ve algoritmik gösterim üzerinden inceleyelim (*proof by cases*).

Durum (Case) 1: İSTEK alan düğüm *LİDER* veya *ONAY_BEKLEYEN_ADAY* durumundaysa *ONAY* ile cevap gönderir.

Durum 2: İSTEK alan düğüm *EŞLENMİŞ_DÜĞÜM* durumundayken *RED* gönderir.

Durum 3: İSTEK alan aday *ÜYE* durumundaysa liderine iletir. İletilen lider Gözlem 3.2'e göre *ÜYE* durumunda olamaz. Liderin bulunabileceği kalan tüm durumlar *durum 1* ve *durum 2* içine girer.

MASC'da ve MCUBA'da yukarıda listelenen durumlar Şekil 3.2, Şekil 3.4 ve Şekil 3.5'de verilen sonlu durum makineleriyle verilmiştir. Başka bir *durum* yoktur, tüm *durum*lar ele alınmıştır. ■

Ön Teorem 3.2. MASC 'da *ÜYE* durumundaki düğümler başka hiçbir duruma geçiş yapamazken, *EŞLENMİŞ_LİDER* durumundaki düğümler de aynı raunt içinde başka duruma geçiş yapmazlar. MCUBA'da *ÜYE*, *ÜYE_SON*, *LİDER_SON* ve *ÇIKIŞ_DÜĞÜMÜ_SON* durumlarından başka durumlara geçiş yapmazlar.

İspat. MASC'da *EŞLENMİŞ_LİDER* durumundaki düğüm *RAUNT_SONU* kesmesi oluştuğunda *LİDER* durumuna geçer. MASC'da ve MCUBA'da diğer durumlardaki düğümlerin durum değişikliği yapmayacağı Şekil 3.2, Şekil 3.4 ve Şekil 3.5'de verilen sonlu duruma makineleriyle verilmiştir. ■

Ön Teorem 3.3. MCUBA'da bir düğüm adayına *OMURGA_İSTEK* gönderirse, adayından *OMURGA_ONAY* mesajı alır.

İspat. Hedef düğüm küme lideri, küme üyesi veya kümeleme işlemine devam eden bir düğüm olabilir. Hedef düğüm için bütün durumları sırayla incelersek:

Durum 1: Hedef düğüm *LİDER_SON* durumunda olabilir. Bu durumda hedef düğüm *OMURGA_ONAY* mesajı gönderir.

Durum 2: Hedef düğüm *ÜYE* veya *ÜYE_SON* durumlarında olabilir. Bu durumda hedef düğüm *OMURGA_İSTEK* mesajını liderine iletir.

Durum 3: Hedef düğüm *LİDER* veya *ONAY_BEKLEYEN_ADAY* durumlarında olabilir. Bu durumda hedef düğüm *OMURGA_İSTEK* mesajını kaydeder. Hedef düğüm *ÜYE* veya *ÜYE_SON* durumuna geçerse kaydettiği *OMURGA_İSTEK* mesajını komşularına iletir. Eğer hedef düğüm *LİDER_SON* durumuna geçiş yaparsa *OMURGA_ONAY* mesajını kaynak düğüme gönderir.

Durum 4: Hedef düğüm *ÇIKIŞ_DÜĞÜMÜ, ÇIKIŞ_DÜĞÜMÜ_SON veya OMURGA_ONAYI_BEKLEYEN_ÇIKIŞ_DÜĞÜMÜ (SINK_WT_FOR_ BACKBONE)* durumlarında olabilir. Bu durumda hedef düğüme *OMURGA_İSTEK* mesajı alması mümkün değildir. Şekil 3.4 ve Şekil 3.5'te verilen sonlu durum makinelerine göre bu durumun oluşması mümkün değildir.

Başka bir *durum* yoktur, tüm *durum*lar ele alınmıştır. ■

Teorem 3.1. MASC'ın çalışması sonucunda her düğüm küme lideri veya bir kümeye ait üye düğüm olabilir.

İspat. Teorem doğruysa, algoritmayı çalıştıran düğümler Gözlem 3.2'e göre *ÜYE* ve *EŞLENMİŞ_LİDER* durumunda olabilir. Aksinin doğru olduğunu farz edip, teoremi ispat edelim (*proof by contradiction*). Böylece algoritmayı sonlandıran bir düğüm *LİDER* veya *ONAY_BEKLEYEN_ADAY* durumlarının herhangi birinde olabilir. Bu durumları sırayla inceleyelim:

- *LİDER*: Bir düğüm *LİDER* durumunda bitiyorsa adayından *İSTEK* mesajı almıyor veya kendisi adayına bir *İSTEK* mesajı göndermiyor olmalıdır. Çünkü bunlardan herhangi birini yaparsa durum değiştirir. Eğer *İSTEK* mesajı almazsa, *LİDER_ZAMAN_AŞIMI* olup adayına *İSTEK* mesajı gönderir, *ONAY_BEKLEYEN_ADAY* durumuna geçer. Ön Teorem 3.1'e göre adayından *İSTEK* mesajı alırsa *ONAY* gönderir, şartlara bağlı olarak ÜYE veya *EŞLENMİŞ_LİDER* durumuna geçer. Düğümün adayı yoksa *LİDER_ZAMAN_AŞIMI* olup *EŞLENMİŞ_LİDER* durumuna geçer.

- *ONAY_BEKLEYEN_ADAY*: Bu durumdaki düğümün adayından mesaj almıyor olması gerekir. Çünkü adayından *İSTEK* mesajı alırsa *ONAY* gönderip *EŞLENMİŞ_LİDER* veya *ÜYE* durumuna geçer, adayı ile birlikte tüm komşularından

RED alırsa *EŞLENMİŞ_LİDER* durumuna geçer. Ön Teorem 3.1'e göre adayına *İSTEK* mesajı gönderirse, adayından *DÜŞME* veya *ONAY* mesajı alır. Düğüm bu durumdan mutlaka ayrılır.

Düğüm her koşulda bu iki durumdan *EŞLENMİŞ_LİDER* veya *ÜYE* durumlarına geçiş yapar. Ön Teorem 3.2'e göre de tersi mümkün değildir. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Teorem 3.2. MCUBA'nın çalışması sonucunda her düğüm küme lideri, çıkış düğümü veya bir kümeye ait üye düğüm olabilir.

İspat. Teorem doğruysa, algoritmayı çalıştıran düğümler Gözlem 3.2'e göre *ÜYE*, *ÜYE_SON*, *LİDER_SON* veya *ÇIKIŞ_DÜĞÜMÜ_SON* durumunda olabilir. Aksinin doğru olduğunu farz edip, teoremi ispat edelim. Böylece algoritmayı sonlandıran bir düğüm *LİDER*, *ONAY_BEKLEYEN_ADAY*, *ÇIKIŞ_DÜĞÜMÜ* veya *OMURGA_ONAYI_BEKLEYEN_ÇIKIŞ_ DÜĞÜMÜ* durumlarının herhangi birinde olabilir. Bu durumları sırayla inceleyelim:

- *LİDER*: Bir düğüm *LİDER* durumunda bitiyorsa adayından *İSTEK* mesajı almıyor veya kendisi adayına bir *İSTEK* mesajı göndermiyor olmalıdır. Çünkü bunlardan herhangi birini yaparsa durum değiştirir. Eğer *İSTEK* mesajı almazsa, *LİDER_ZAMAN_AŞIMI* olup adayına *İSTEK* mesajı gönderir, *ONAY_BEKLEYEN_ADAY* durumuna geçer. Adayından *İSTEK* mesajı alırsa *ONAY* gönderir, şartlara bağlı olarak *ÜYE*, *ÜYE_SON*, *LİDER_SON* veya *ONAY_BEKLEYEN_ADAY* durumuna geçer. Düğümün adayı yoksa *LİDER_ZAMAN_AŞIMI* olup *LİDER_SON* durumuna geçer.

- *ÇIKIŞ_DÜĞÜMÜ*: *LİDER* düğümü ile aynı koşullara sahiptir. Düğüm bu durumdan mutlaka ayrılıp *ONAY_BEKLEYEN_ADAY*, *OMURGA_ONAYI_BEKLEYEN_ÇIKIŞ_DÜĞÜMÜ* veya *ÇIKIŞ_DÜĞÜMÜ_ SON* durumuna geçiş yapar.

- *ONAY_BEKLEYEN_ADAY*: Bu durumdaki düğümün adayından mesaj almıyor olması gerekir. Çünkü adayından *İSTEK* veya *DÜŞME* mesajı alırsa *küme_üst_seviyesi*ne ulaşıp *LİDER_SON* durumuna geçecek veya küme üyesi olup *ÜYE* veya *ÜYE_SON* durumuna geçecektir. Son olasılık doğruysa komşularının hepsinden *DÜŞME* alıp *LİDER_SON* duruma geçecektir. Ön Teorem 3.1'e göre adayına *İSTEK* mesajı gönderirse, adayından *DÜŞME* veya *ONAY* mesajı alır. Düğüm bu durumdan mutlaka ayrılır.

- *OMURGA_ONAYI_BEKLEYEN_ÇIKIŞ_DÜĞÜMÜ*: Bu durumdaki düğümün *OMURGA_İSTEK* mesajlarına cevap almıyor olması gerekmektedir. Hâlbuki Ön Teorem 3.3'e göre *OMURGA_İSTEK* mesajlarının hepsine *OMURGA_ONAY* ile cevap verilir. Düğüm her koşulda *ÇIKIŞ_DÜĞÜMÜ_SON* durumuna geçer.

Düğüm her koşulda bu dört durumdan *ÜYE*, *ÜYE_SON*, *LİDER_SON* veya *ÇIKIŞ_DÜĞÜMÜ_SON* durumlarına geçiş yapar. Ön Teorem 3.2'e göre de tersi mümkün değildir. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Sonuç Teoremi (Corollary) 3.1. MASC ve MCUBA'da kilitlenme ve açlık (*starvation*) yoktur.

İspat. Aksinin doğru olduğunu farz edelim. Böylece MASC'ın veya MCUBA'nın kümeleme veya omurga oluşturma işlemlerinin herhangi birinde kilitlenme veya açlık vardır. Gözlem 3.1'e göre bir kümeleme oturumu birinci düğümün *İSTEK*, ikinci düğümün *ONAY* veya *RED* mesajlarını göndermesiyle oluşur. Gözlem 3.2'e göre bir omurga oluşturma işleminde çıkış düğümü tarafından *OMURGA_İSTEK* mesajı gönderilir, *OMURGA_İSTEK* mesajını alan lider düğüm *OMURGA_ONAY* ile cevap verir. Bu mesaj akışlarının herhangi birinde bir problem olması gerekmektedir. Fakat Ön Teorem 3.1, Ön Teorem 3.2, Teorem 3.1 ve Teorem 3.2'e göre bu kümeleme ve omurga oluşturma oturumları sorunsuz biter. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

3.4.2. Kümeleme Kalitesi

Teorem 3.3. r rauntta çalışan MASC'ın oluşturabileceği maksimum küme seviyesi 2^r'dir.

İspat. Tüme varım ile ispat edelim.

Başlangıç Basamağı: r=1 olduğunda düğümler 1 seviyede eşlenip 2 seviyede kümeler oluştururlar. 2^1=2

Tümevarım Basamağı: r rauntta 2^r seviyede kümelerin oluştuğunu kabul edelim. (r+1). rauntta 2^r seviyedeki 2 küme birleşip 2^r 2 = 2 $^{(r+1)}$ seviyede kümeler oluşturabilir. ■

Teorem 3.4 Rastgele eşit yoğunluklu düğümlerin dağıldığı bir ağda uygulanan MCUBA algoritmasının sonucunda bir kümenin içindeki maksimum düğüm sayısı *küme_üst_seviyesi* = s olduğunda (2s-2) 'dir.

İspat. Küme seviyesi *s*'e ulaşana kadar düğümler *İSTEK* mesajlarını *RED* ile cevaplarlar. Bir küme içindeki maksimum küme seviyesine ulaşmak için iki kümenin de seviyesinin *s*-1 olması gerekmektedir. Böylece, 2(*s*-1) = 2*s* -2 'dir. ■

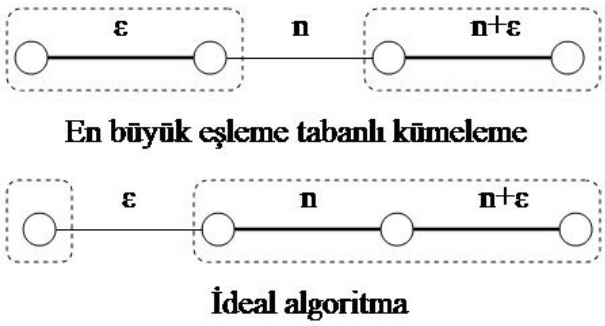
3.4.3. Seçilen Kenarların Kalitesi

Bu bölümde MASC ve MCUBA için kümeleme ve omurga oluşturma sırasında seçtiği toplam kenarların ağırlıklarının alt sınırlarının (yaklaşım oranlarının) analizi verilmiştir. Bu bölümde kullanılacak değişkenler şu şekildedir:

- *K*: En büyük kenarları seçen ve *k* adet küme oluşturan ideal algoritma tarafından *r* rauntta kümeleme için seçilen kenarların toplam ağırlığı.
- *B*: En büyük kenarları seçen ve *k* adet küme oluşturan ideal algoritma tarafından *r* rauntta omurga oluşturma için seçilen kenarların toplam ağırlığı.
- *W*: En büyük eşleme kullanılarak ve *k* adet küme oluşturan algoritma tarafından *r* rauntta kümeleme için seçilen kenarların toplam ağırlığı.
- *A*: MASC tarafından *r* rauntta seçilen kenarların toplam ağırlığı.
- *S*: MCUBA tarafından *s* = *küme_üst_seviyesi* ve $s=2^r$ parametreleriyle kümeleme için seçtiği kenarların toplam ağırlığı.
- *C*: MCUBA tarafından *s* = *küme_üst_seviyesi* ve $s=2^r$ parametreleriyle omurga oluşturma için seçtiği kenarların toplam ağırlığı.

Ön Teorem 3.4. *r*=1 olduğunda, *W* için alt sınır $W \geq 1/2\ K$ 'dır.

İspat. Alt sınır değeri (Feo et al., 1992)'de verilmiştir. Şekil 3.9'da görüldüğü üzerinde görüldüğü üzere en büyük eşleme ile seçilen kenarların ağırlığı *n*+ε olduğunda ideal algoritmanın seçtiği toplam kenar ağırlığı 2*n*+ε dir. ■



Şekil 3.9 En büyük eşleme tabanlı kümeleme algoritması *r*=1 için en kötü durum

Ön Teorem 3.5. *W* için alt sınır $W \geq 1/2\ K$ 'dır.

İspat. İdeal algoritmanın seçtiği kenar sayısı ilk rauntta 1/2 *K*'dır. Daha sonra ideal algoritma raundlar ilerledikçe 1/4 *K*, 1/8 *K*, 1/16 *K* olarak kenar seçmeye devam etsin. Bu durumu göz önüne alıp tümevarım ile ispat edelim (İdeal algoritmanın raundlara göre seçtiği kenar ağırlığı sonucu değiştirmez sadece işlemlerimizde kolaylık olması amacıyla bu şekilde seçtiğini varsayıyoruz).

Başlangıç Basamağı: $r = 1$ olduğunda Ön Teorem 3.4'de alt sınır verilmiştir:

$$W \geq 1/2\ K$$

r=2 olduğunda ideal algoritma tarafından seçilen kenar ağırlığı (1/2+1/2) *K*'dır. En kötü durumumda eşleme tabanlı algoritma (1/4+1/4) *K* kadar kenar ağırlığı seçer.

$$W \geq (1/4 + 1/4)\ K$$

$$W \geq 1/2\ K$$

Tümevarım Basamağı: $r = n$ olduğunda doğru olduğunu kabul edelim. $r = n + 1$ olduğunda ideal algoritmanın seçtiği kenar ağırlıklarının en kötü durumunda 1/2 si kadar seçilecektir.

$$W = \sum_{i=2}^{n} (1/2)^i K \ + (1/2)^{n+1}\ 1/2\ K$$

$$2\left(\sum_{i=2}^{n} (1/2)^i K + (1/2)^{n+1}\ 1/2\ K\right) \geq \sum_{i=1}^{n+1} (1/2)^i K$$

$$2\sum_{i=2}^{n} (1/2)^i K \ + (1/2)^{n+1}\ K \geq \sum_{i=1}^{n+1} (1/2)^i K$$

$$2\sum_{i=1}^{n} (1/2)^i K \ - K + (1/2)^{n+1}\ K \geq \sum_{i=1}^{n+1} (1/2)^i K$$

$$\sum_{i=1}^{n+1} (1/2)^i K + \sum_{i=1}^{n} (1/2)^i K - K \geq \sum_{i=1}^{n+1} (1/2)^i K$$

n, ∞'a yaklaşırken $\sum_{i=1}^{n+1} (1/2)^i K = K$ olacağından dolayı teoremimiz doğrudur.

$$W \geq 1/2\ K \ \blacksquare$$

Ön Teorem 3.6. *A* için alt sınır $A \geq 1/2\ W$'dır.

İspat. İspat Ön Teorem 3.5'e çok benzerdir. En büyük eşleme tabanlı algoritmanın seçtiği kenar ağırlıklarının artan rauntlara göre 1/2 *W*, 1/4 *W*, 1/8 *W* olarak sıralandığını kolaylık olması için varsayalım Tüme varım ile ispat edelim.

Başlangıç Basamağı: $r = 1$ olduğunda DETSKA, Hoepman'ın eşleme algoritmasıyla aynı çalışır. Hoepman'ın eşleme algoritması 1/2 yaklaşımlı olduğu için

$$A \geq 1/2\ W$$

$r = 2$ olduğunda DETSKA, ilk rauntta Hoepman'ın eşleme algoritmasıyla aynı çalışır. İkinci rauntta en büyük eşlemenin yarısı kadar kenar ağırlığı seçer.

$$A \geq (1/4 + 1/4)\ W$$

$$A \geq 1/2\ W$$

Tümevarım Basamağı: $r = n$ olduğunda doğru olduğunu kabul edelim. $r = n + 1$ olduğunda ideal algoritmanın seçtiği kenar ağırlıklarının en kötü durumunda 1/2 si kadar seçilecektir.

$$A = \sum_{i=2}^{n} (1/2)^i W + (1/2)^{n+1}\ 1/2\ W$$

$$2\left(\sum_{i=2}^{n} (1/2)^i W + (1/2)^{n+1}\ 1/2\ W\right) \geq \sum_{i=1}^{n+1} (1/2)^i W$$

$$\sum_{i=1}^{n+1} (1/2)^i W + \sum_{i=1}^{n} (1/2)^i W - W \geq \sum_{i=1}^{n+1} (1/2)^i W$$

n, ∞'a yaklaşırken $\sum_{i=1}^{n+1} (1/2)^i W = W$ olacağından dolayı teoremimiz doğrudur.

$$A \geq 1/2\ W \blacksquare$$

Teorem 3.5. A için alt sınır $A \geq 1/4\ K$ 'dır.

İspat. Ön Teorem 3.6'e göre $A \geq 1/2\ W$ ve Ön Teorem 3.5'e göre $W \geq 1/2\ K$'ye eşittir. Bu iki denklemi birleştirirsek $A \geq 1/4\ K$ olur. ■

Ön Teorem 3.7. S değeri için alt seviye $S \geq 1/2\ W$'dir.

İspat. Tümevarım ile ispat edelim.

Başlangıç Basamağı: $r = 1$ olduğu durumda s=2 dir. MCUBA s=2 ile çalıştırıldığında en büyük küme seviyesi $2s$-2 = 2 'dir. Bu durumda MCUBA ile MASC kümelemede aynı sonucu verirler. MASC'ın kümeleme için seçtiği kenar ağırlığı:

$$W\ 1/2 \leq (S{=}A)$$

Tümevarım Basamağı: $r = n$ olduğunda doğru olduğunu kabul edelim. Bu durumunda $2s$-2'den $s=2^{n+1}$-1 olur. $r = n + 1$ olduğunda MASC algoritmasının Teorem 3.3'e göre ulaşabileceği küme seviyesi $s = 2^{n+1}$ dir. MCUBA'nın ulaşabileceği üst küme seviyesi Teorem 3.4'e göre $2s$-2 dir. Bu durumda:

$$\sum_{i=2}^{n}(1/2)^i W \leq S = \sum_{i=2}^{2^{n+1}-2}(1/2)^i W$$

$$\sum_{i=2}^{n}(1/2)^i W + 1/2^{n+1} W \leq \sum_{i=2}^{2^{n+1}-2}(1/2)^i W + 1/2^{n+1} W \leq \sum_{i=2}^{2^{n+2}-2}(1/2)^i W$$

$n \geq 2$ olduğu durumda $\sum_{i=2}^{n+1}(1/2)^i W \leq \sum_{i=1}^{2^{n+2}-2}(1/2)^i W$ olduğundan

$$\sum_{i=1}^{n}(1/2)^i W \leq S. \quad \blacksquare$$

Teorem 3.6. S değeri için $S \geq 1/4\ K$ 'dir.

İspat. Ön Teorem 3.7'e göre $S \geq 1/2\ K$ ve Teorem 3.5'e göre

$A \geq 1/2\ K$ dır. $S \geq A$ için $S \geq 1/4\ K$. ■

Sonuç Teoremi 3.2. $S+C$ değerleri $1/4\ K \leq S+C \leq K+B$

İspat. Teorem 3.6'ya göre $S \geq 1/4\ K$ dir. $0 \leq C \leq B$ olduğu için $1/4\ K \leq S+C \leq K+B$'dir. ■

3.4.4. Mesaj, Zaman ve Uzay Karmaşıklıkları

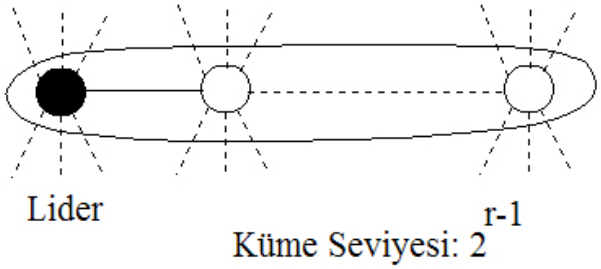
Teorem 3.7. MASC algoritmasının mesaj karmaşıklığı Δ düğüm derecesi, r de raunt sayısı olduğunda düğüm başına alt sınırda $\Omega(r)$, üst sınırda $O(\Delta 2^r)$ 'dır.

İspat. En kötü durumda, 2^{r-1} seviyedeki bir kümenin lideri kümesindeki bütün elemanların komşuları tarafından reddedilir. Bu durum Şekil 3.10'da verilmiştir. Bu durumdaki toplam mesaj değişimi:

$$\frac{2\Delta + 4\Delta + 6\Delta + ... + (2^r + 2)\Delta}{2^{r-1}}$$

$$= \frac{2\Delta(1 + 2 + 3 + ... + 2^{r-1} + 1)}{2^{r-1}}$$

$$= \frac{\Delta(2^{r-1} + 1)(2^{r-1} + 2)}{2^{r-1}} \in O(\Delta 2^r)$$



Şekil 3.10 MASC'ın en kötü mesaj karmaşıklığı için topoloji

Alt sınırda bir düğüm diğerine *İSTEK* gönderir, bu mesajı alan düğüm de *ONAY* ile cevap verir. Bu durumda toplamda 2*r* mesaj değişimi olur. Bundan dolayı alt sınır Ω(*r*) dır. ■

Sonuç Teoremi 3.3. MASC algoritmasının mesaj karmaşıklığı alt sınırda Ω(*rN*), üst sınırda O(Δ2^r*N*) 'dır.

İspat. Teorem 3.7'e göre mesaj karmaşıklığı düğüm başına alt sınırda Ω(*r*), üst sınırda O(Δ2^r) 'dır. *N* düğüm için mesaj karmaşıklıkları alt sınırda Ω(*rN*), üst sınırda O(Δ2^r*N*) 'dır. ■

Teorem 3.8. MASC algoritmasının zaman karmaşıklığı *r* raunt sayısı olduğunda O(Δ2^r*r*) 'dır.

İspat. En kötü durumdaki zaman karmaşıklığı, Şekil 3.10'daki topolojide oluşur ve mesaj karmaşıklığıyla aynı olup O(Δ2^r)'e eşittir. *r* raunt için zaman karmaşıklığı O(Δ2^r*r*)'e eşittir. ■

Teorem 3.9. MCUBA algoritması *s=küme_üst_seviyesi* parametresiyle mesaj karmaşıklığı düğüm başına alt sınırda Ω(2), üst sınırda k üretilen küme sayısı olmak üzere O(Δ*s*+*k*) 'dır.

İspat. En kötü durum Şekil 3.10'da verilen topolojiyle aynıdır; fakat bu topolojiden farklı olarak küme seviyesi *r*'ye değil *s*'e bağlıdır. Buna ek olarak toplamda 2*k* kadar omurga oluşturma mesajı gönderebilir. En kötü durumdaki hesaplamalar aşağıdaki gibi hesaplanır:

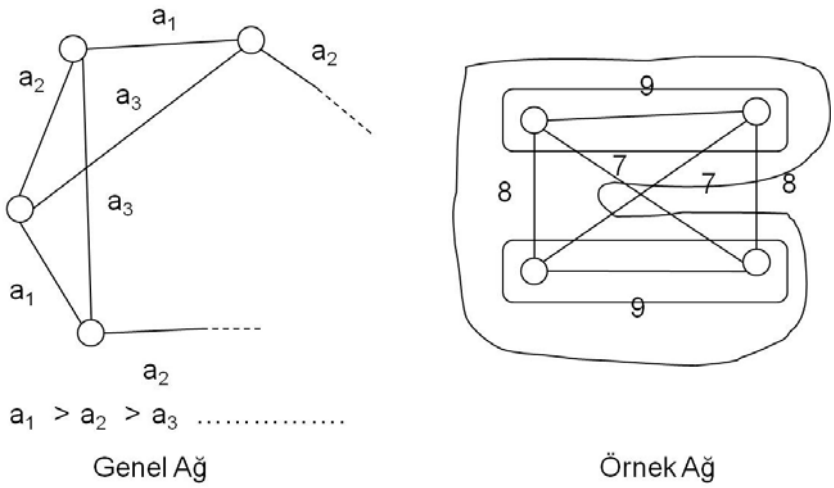
$$\frac{2\Delta + 4\Delta + 6\Delta + ... + (2s-2)2\Delta + 2\Delta}{2s-2} + 2k$$

$$= \frac{2\Delta(1 + 2 + 3 + ... + (2s-1))}{2s-2} + 2k$$

$$= \frac{(\Delta)(2s-1)(2s)}{2s-2} + 2k \in O(\Delta s + k)$$

En iyi durum Şekil 3.11'de verilen K_n tam çizgesinde (*complete graph*) olur. Bu durumda *N* düğüm *İSTEK* ve *ONAY* mesajlarını değişirler. Bu durumda zaman karmaşıklığı:

$$\frac{n(1+1/2+1/4+...)}{n} \leq 2 \blacksquare$$



Şekil 3.11 MCUBA'nın en iyi mesaj karmaşıklığı için topoloji

Sonuç Teoremi 3.4. MCUBA algoritması *s=küme_üst_seviyesi* parametresiyle ve $\Delta s>k$ olduğunda mesaj karmaşıklığı alt sınırda $\Omega(N)$, üst sınırda $O(\Delta sN)$ 'dır.

İspat. Teorem 3.9'e göre MCUBA algoritması *s=küme_üst_seviyesi* parametresiyle düğüm başına alt sınırda $\Omega(2)$, üst sınırda $O(\Delta s)$ 'dır. N düğüm için mesaj karmaşıklığı alt sınırda $\Omega(N)$, üst sınırda $O(\Delta sN)$ 'dır. ■

Teorem 3.10. MCUBA'nın zaman karmaşıklığı alt sınırda $\Omega(\log_2(N))$, üst sınırda $O(s \Delta \log_2(s))$'dir.

İspat. N adet duyarga düğümümüz olduğunu düşünelim. En iyi ve en kötü durumu ele alırken küme seviyesini N olarak kabul edip kümeleme ve omurga oluşturmayı birlikte yapacağız. En iyi durum Şekil 3.11'deki ağ tipinde meydana gelir. En iyi durumda önce seviye 1 kümeler birleşip seviye 2 kümeleri, seviye 2 kümeler birleşip seviye 4 kümeleri oluşturur. Küme seviyesi n olana kadar bu işlem devam eder. Bundan dolayı alt sınır $\Omega(\log_2(N))$ dir. En kötü durumda Teorem 3.8'e göre $s=2^r$ için $O(s \Delta \log_2(s))$'dir. ■

Teorem 3.11. MASC ve MCUBA algoritmalarında en büyük boyutlu mesajın uzay karmaşıklığı $O(N)$'e eşittir.

İspat. MASC ve MCUBA algoritmasında *İSTEK* mesajının içindeki RSS listesi veya küme içindeki elemanların listesi N olabileceğinden dolayı $O(N)$'e eşittir. ■

Teorem 3.12. MASC ve MCUBA algoritmasında bir düğümün kullanması gereken belleğin uzay karmaşıklığı $O(N)$'e eşittir.

İspat. MASC ve MCUBA algoritmasında lider düğüm RSS listesini ve küme içindeki elemanların listesini bellekte tutması gerektiği için ve bu bellek alanı en büyük *N* olabileceğinden dolayı O(*N*)'e eşittir. ■

3.4.5. Enerji Tüketimindeki Eksilmeler

Teorem 3.13. Düğümlerin rastgele ve eşit yoğunlukta dağıldığını varsayalım. Ayrıca enerji tüketimlerinin önemli ölçüde mesaj gönderimleri ve alımları sırasında oluştuğunu varsayalım (iletişim dışındaki enerji tüketimini ihmal ediyoruz). Kulak misafiri olma tekniği MCUBA'da *küme_üst_seviyesi* ≥ 8 olduğunda sadece kümeleme oturumu yapan düğümlerin ortalama bir kümeleme oturumunda % 40'dan fazla enerji tasarrufu sağlar.

İspat. *s=küme_üst_seviyesi* olsun. Bu durumda küme seviyeleri 1 ile 2*s*-2 arasında eşit olasılıkla değişebileceği için ortalama küme seviyesi *s* olur. Böylelikle ortalama küme çapı (*diameter*) *s*/2 olur. Şekil 3.12'de iki küme arasındaki ortalama bir kümeleme oturumu gösterilmiştir. *ALT_SEVİYE_ONAY* (*CONTROL_ACK*) mesajının güvenli iletişim için alt katmanlar tarafından kullanıldığını varsayalım. KMO tekniği olmadan, MCUBA'daki her mesaj değişiminde *ALT_SEVİYE_ONAY* mesajı gerekeceğinden dolayı toplam mesaj değişim sayısı (*M*):

M = (*İSTEK* mesajları + *ONAY (REPLY)* mesajları + *ALT_SEVİYE_ONAY* mesajları)

= ((*s*/2 + 1) + (*s*/2 + 1) + (*s* + 2))

= 2*s* + 4

KMO tekniğini kullanarak en son iki *ALT_SEVİYE_ONAY* mesajı dışındaki mesajlara ihtiyaç yoktur; KMO kullanarak toplam mesaj sayısı

= ((*s*/2 + 1) + (*s*/2 + 1) + 2)

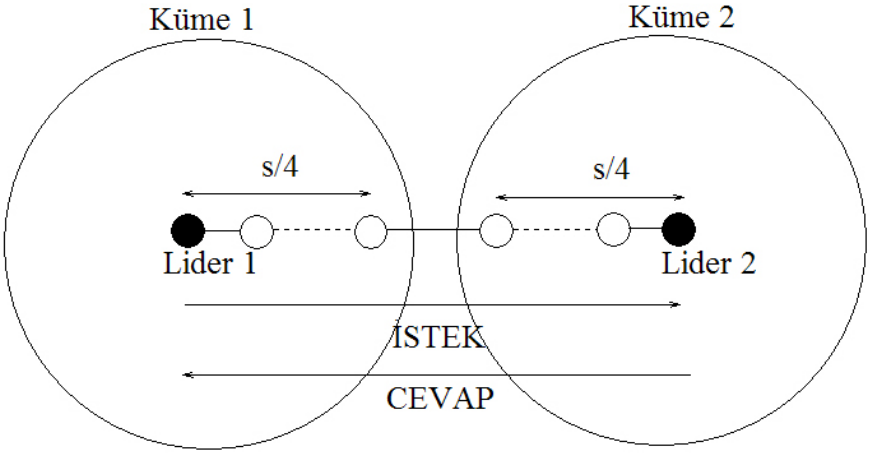
= *s* + 4

Bu durumda

$$\frac{M - T}{M} = \frac{s}{2s + 4}$$

s ≥ 8 için

$$\frac{s}{2s + 4} 100 \geq \% 40.$$ ■



Şekil 3.12 MCUBA'daki ortalama bir kümeleme oturumu

3.5. Performans Değerlendirmesi

Öncelikle Hoepman'ın dağıtık eşleme algoritmasının performansını görmek için ns2 2.31 sürümü üzerinde uyguladık (Fall and Varadhan, 2010). Hoepman'ın algoritmasının yaklaşım oranını hesaplamak için Gabow'un merkezi en büyük ağırlıklı eşleme algoritması uygulandı (Gabow, 1990). Daha sonra MASC ve MCUBA algoritmalarını ns2 benzetim ortamında uyguladık. Bunların yanında eşleme için kenar ağırlıkları yerine düğüm kimliklerini kullanan MOD-MASC algoritması uygulandı. MOD-MASC algoritmasında bir düğümün adayı bağlı olduğu en küçük kimlikli komşusu olarak alındı. MOD-MASC algoritması MASC ile aynı zaman ve mesaj karmaşıklığına eşit olsa da rastgele seçim yaptığından dolayı kenar ağırlığına yaklaşım oranı çok farklılık gösterebilir. Algoritmamızı literatürdeki çizge teorik kümeleme algoritmalarıyla karşılaştırmak için dağıtık kapsayan ağaç tabanlı kümeleme algoritması (DSTA) (Erciyes et al., 2008) ve dağıtık bağlı hâkim küme algoritmasını (DS) (Wu and Li, 1999) uyguladık. DSTA algoritması çıkış düğümüne doğru Bölüm 2.1.1'de gösterilen biçimde bir kapsayan ağaç ile birlikte kümeleri oluşturur. Deneylerimizde DSTA algoritmasının derinlik parametresini 3 olarak ayarladık. DS algoritması BHK oluşturur ve Wu'nun algoritmasına çok benzer kurallar içerir. DS algoritması BHK bulmak için aşağıdaki iki kuralı uygular:

1. Bir *v* düğümün bağlı olmayan iki komşusu varsa kendini BHK içine alır.

2. Bir *v* düğümünün kendinden büyük kimlikli düğümleri *v* düğümün bütün komşularını kapsıyorsa düğüm kendini BHK içine almaz.

Rastgele eşit yoğunlukta yayılan 100 düğümden 400 düğüme kadar düğüm içeren ağlar ürettik. IEEE 802.11 radyo ve MAC katmanı standartları alt seviyedeki protokoller olarak kullanıldı (IEEE 802.11 StandardWorking Group, 2007). Radyo dalgasının yayılma modeli olarak iki-yönlü yer modeli (*two-way ground model*) seçildi. Her düğüme 1kJ enerji içeren bir pil konuldu. İletim gücü 0.660mW olarak,

alım gücü 0.395mW olarak ayarlanırken kapsama alanı 250m'ye sabitlendi. Algoritmaların performansı 4,5 ve 6 olarak değişen ortalama düğüm derecelerine göre değiştirildi. Ortalama düğüm derecelerinin değiştirebilmesi için Şekil 3.13'de verilen değişen yüzey alanları kullanılmıştır.

Düğüm Sayısı/Derece	4	5	6
100	2700x1200	2520x1200	2340x1040
200	5100x1200	4760x1120	4420x1040
300	7800x1200	7280x1120	6760x1040
400	10200x1200	9520x1120	8840x1040

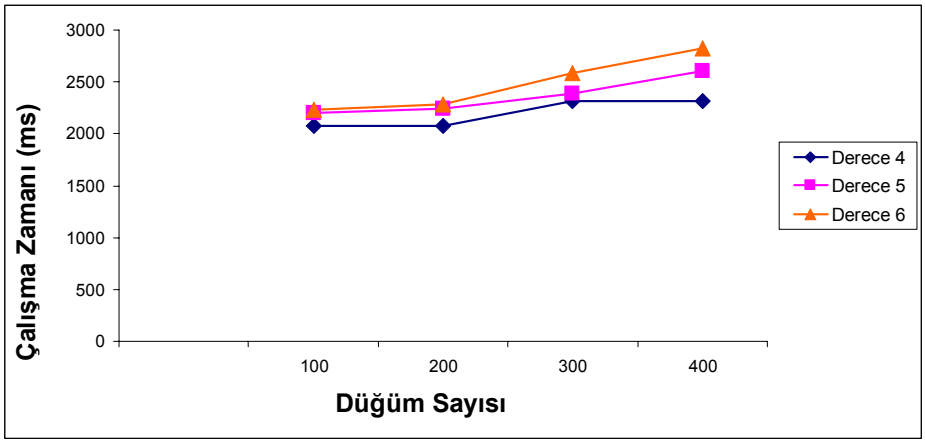
Şekil 3.13 Yüzey alanlarının boyutları (X*Y (m))

3.5.1. Hoepman'ın Algoritmasının Testleri

Hoepman'ın algoritmasının yaklaşım oranlarının değişen düğüm sayısı ve düğüm derecelerine göre ölçüldü.

Düğüm Sayısı/ Yaklaşım Oranı	4	5	6
100	0.9986	0.9982	0.9977
200	0.9986	0.9983	0.9983
300	0.9992	0.9987	0.9984
400	0.9991	0.9988	0.9986

Şekil 3.14 Hoepman'ın algoritmasının yaklaşım oranları

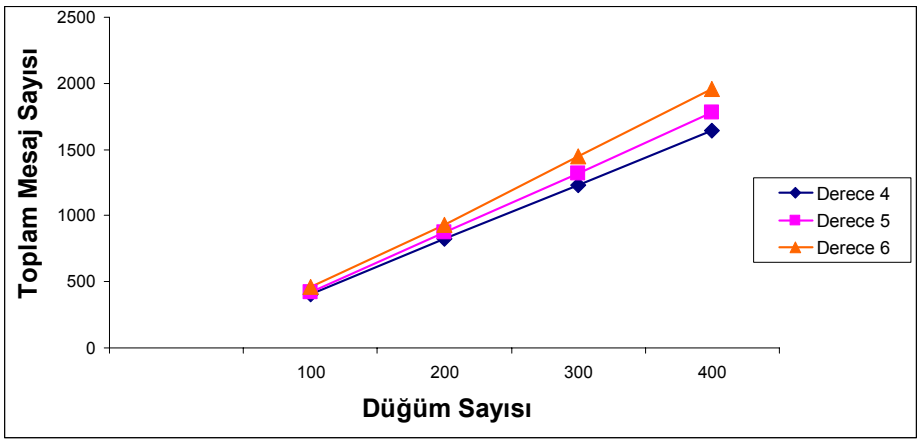


Şekil 3.15 Hoepman'ın algoritmasındaki toplam mesaj sayısı

Yaklaşım oranını ölçmek için, V düğümlerin algoritması, E_w de kenarların RSS değerleri kümesi olmak üzere $G_w = (V, E_w)$ olarak kenar ağırlıklı bir çizge kullanılmıştır. Gabow'un algoritması merkezi olduğu için G_w kümesini girdi olarak alıp E_M en büyük eşleme kümesini çıktı olarak verir. E_H kümesinin Hoepman tarafında üretilen eşleme kümesi olduğunu, T_H kümesinin E_H kümesindeki kenarların toplam ağırlıkları, T_M kümesinin de E_M kümesindeki kenarların toplam ağırlıkları olduğunu varsayalım; bu durumda Hoepman'ın algoritmasının yaklaşım oranı T_H/T_M olur. Şekil 3.14'te Hoepman'ın algoritmasının yaklaşım oranları verilmiştir. Hoepman'ın algoritmasının ölçülen yaklaşım oranları 0.99'dan büyüktür ve düğüm

sayısından ve ortalama düğüm derecesinden bağımsızdır. Teorik olarak yaklaşım oranı 0.5 olmasına rağmen, rastgele yerleştirilen duyarga düğümlerinin oluşturduğu ağlar üzerinde 1'e çok yakın sonuçlar ölçülmüştür.

İkinci olarak, Hoepman'ın algoritmasının toplam mesaj sayıları ve çalışma zamanı ölçümleri verilmiştir. Şekil 3.15'te Hoepman'ın algoritmasının kullandığı mesaj sayıları verilmiştir. Toplam mesaj sayısı düğüm sayısı ile doğrusal olarak artış gösterir ve ortalama düğüm derecesiyle az bir değişim gösterir. Ortalama bir düğüm algoritmanın çalıştırılması ve güvenli iletim için 4 mesaj gönderir. Şekil 3.16'da görüldüğü üzere çalışma zamanları 2s ile 3s arasında değişim gösterir ve çalışma zamanları toplam mesaj sayısında olduğu gibi düğüm derecesi değişimiyle birlikte kararlı değerler gösterir. Sonuç olarak Hoepman'ın algoritmasının yaklaşım oranları çok yüksektir, zaman ve mesaj ölçümlerinden dolayı yüksek sayıda duyarga düğümü içeren TDAlar için ölçeklenebilirdir. Bundan dolayı, Hoepman'ın algoritması TDA için tasarladığımız yenilemeli kümeleme algoritmalarına temel olarak kullanmak için çok uygundur.



Şekil 3.16 Hoepman'ın algoritmasının çalışma zamanları

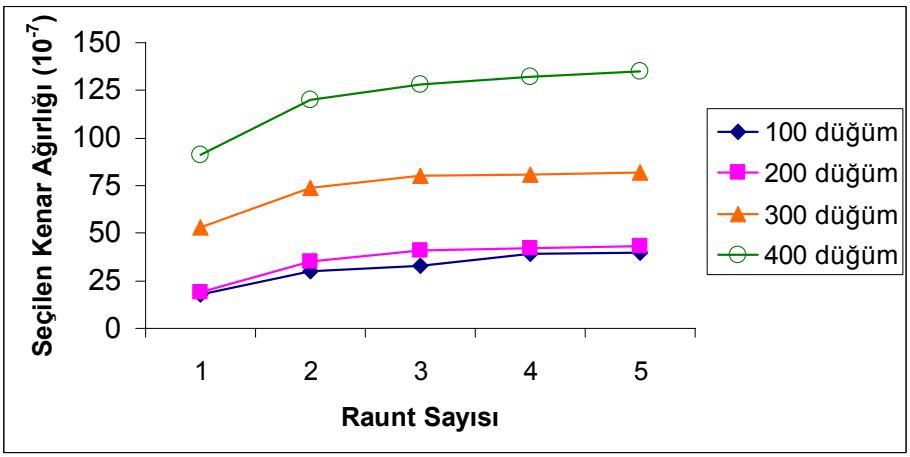
3.5.2. Seçilen Kenar Ağırlıkları

Kümeleme ve omurga oluşturma sırasında güçlü kenarların seçilmesi tasarladığımız algoritmalar için çok önemli bir hedeftir. Bundan dolayı MASC ve MCUBA algoritmalarının seçtiği toplam kenar ağırlıklarını ölçtük. Kenar ağırlıkları IEEE 802.11 standartlarında tanımlanan RSS değerleriyle tanımlanmıştır (IEEE 802.11 StandardWorking Group, 2007).

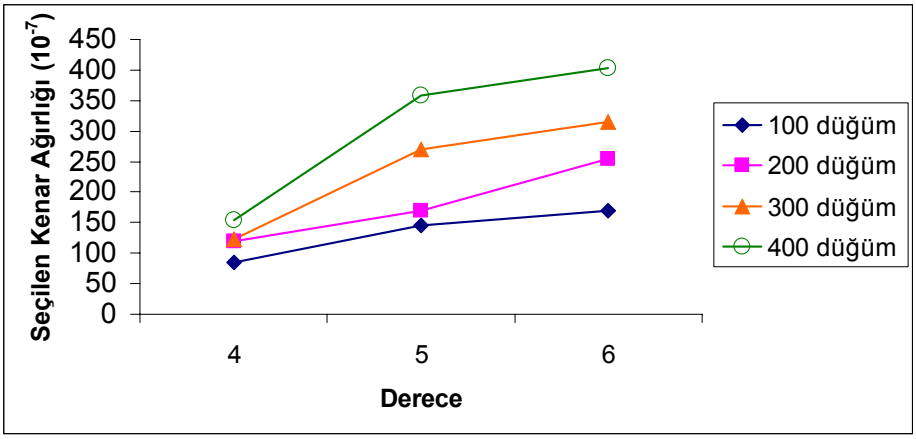
Şekil 3.17'de MASC algoritmasının seçtiği kenar ağırlıklarının artan rauntlara göre değişimi verilmiştir. *r*. rauntta seçilen kenar ağırlığı (*r*+1). rauntta seçilen kenar ağırlığından daha fazladır, bundan dolayı fonksiyonun artışı azalan eğimledir. Bundan dolayı Şekil 3.17'de görüldüğü üzere raunt sayısı arttıkça MASC tarafından seçilen toplam ağırlık doğrusal olarak ve azalan eğimle artar. MCUBA

algoritmasının *küme_üst_seviyesi*=10 olduğu durumda seçtiği toplam kenar ağırlığını ölçtük. Ortalama düğüm derecesi arttıkça güçlü kenarların zayıf kenarlara oranı sabit kalmasına rağmen toplam güçlü kenar sayısı artar. MCUBA algoritması rastgele kenar seçmek yerine güçlü kenarları seçtiği için Şekil 3.18'de görüldüğü üzere düğüm derecesi arttıkça MCUBA tarafından seçilen toplam kenar ağırlığı artar.

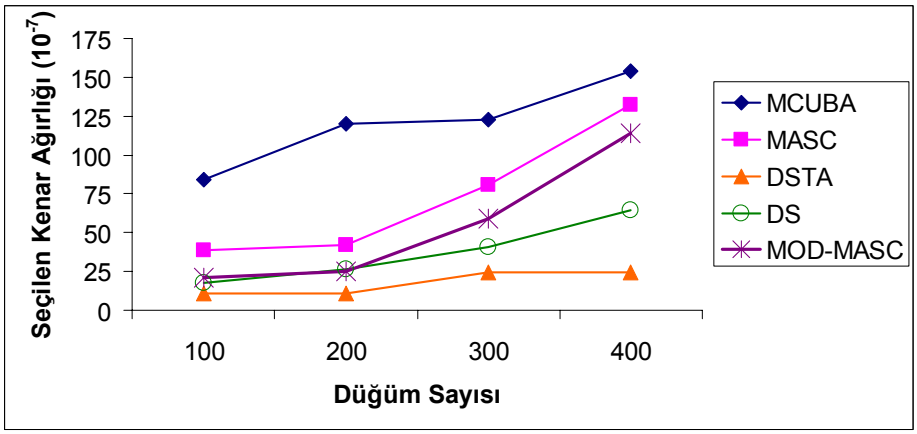
Algoritmaların seçtiği toplam kenar ağırlıklarını Şekil 3.19'da gösterilmiştir. Algoritmalarımızı şu şekilde sınıflandırabiliriz: Ağırlıklı eşleme tabanlı algoritmalar (MCUBA ve MASC), eşleme tabanlı algoritmalar (MOD-MASC), hâkim küme tabanlı algoritmalar (DS) ve kapsayan ağaç tabanlı algoritmalar (DSTA). MCUBA algoritmasını *küme_üst_seviyesi*=4 olarak, MASC ve MOD-MASC algoritmalarını da 4 raunt çalıştırdık. Algoritmalar böyle çalıştırılınca üretilen kümelerin seviyeleri yaklaşıktır. Şekil 3.19'da görüldüğü üzere ağırlıklı eşleme tabanlı ve eşleme tabanlı algoritmalar, hâkim küme tabanlı ve kapsayan ağaç tabanlı algoritmalara göre daha ağır kenarları seçmişlerdir. Bu noktadan yola çıkarak kenar kalitesini göz önüne alırsak çizge eşleme tekniğinin literatürdeki kümeleme tekniklerine göre daha iyi sonuç verdiğini söyleyebiliriz. Ağırlıklı eşleme tabanlı algoritmalar seçilen toplam kenar ağırlığını en büyük yapmayı hedeflerken, rastgele eşleme yapan algoritmalar kenar ağırlıklarını dikkate almadan seçim yaparlar. Bu gerçeği göz önüne alırsak, Şekil 3.19'da görüldüğü üzere MCUBA ve MASC'ın toplam seçtiği kenar ağırlığı MOD-MASC'dan daha fazladır. Ortalama olarak DSTA'nın seçtiği toplam kenar ağırlığına W dersek, DS'nin seçtiği toplam kenar ağırlığı $2.3W$, MOD-MASC'ın kenar ağırlığı $3.7W$, MASC'ın ağırlığı $4.6W$ ve MCUBA'nın ağırlığı $7W$'dir.



Şekil 3.17 MASC'ın seçtiği toplam kenar ağırlıkları



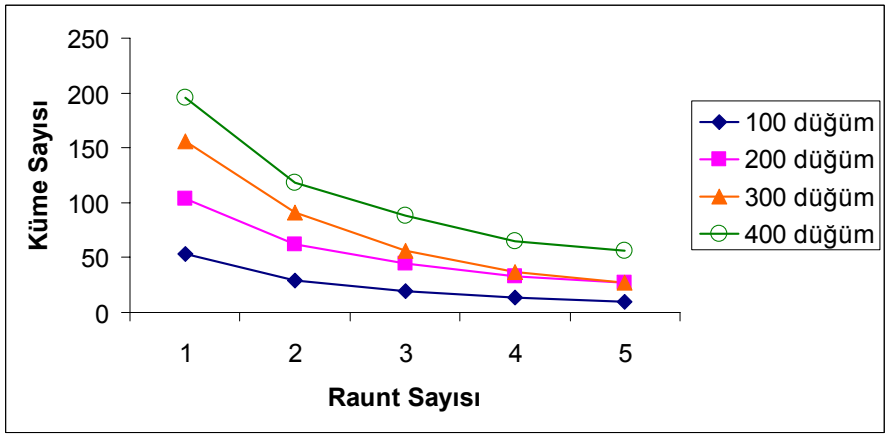
Şekil 3.18 MCUBA'nın seçtiği toplam kenar ağırlıkları



Şekil 3.19 Algoritmaları seçtiği toplam kenar ağırlıkları

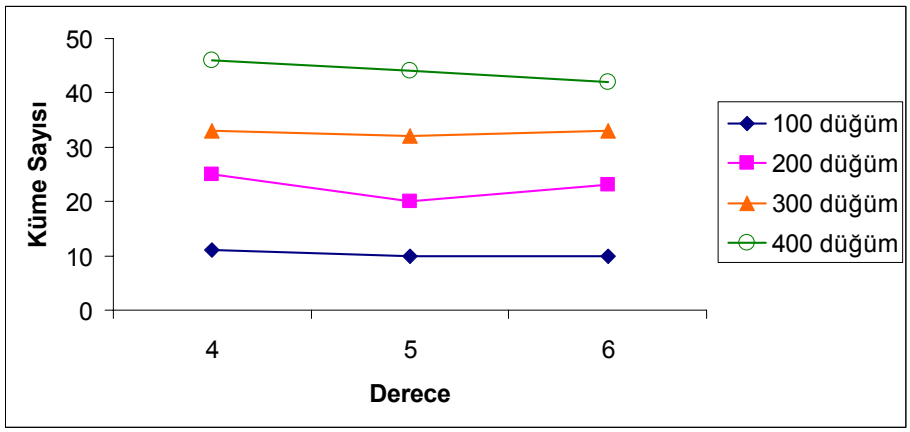
3.5.3. Kümeleme Kalitesi

Kümeleme kalitesini değerlendirmek için iki adet ölçüt kullandık: küme sayısı ve kümelerin içindeki düğüm sayısı. Kümeleme algoritmalarında kümelerin sayısının kontrol edilebilir olması istenir. MASC'da her rauntta alt seviyedeki kümeler birleşip üst seviyede kümeler oluştururlar ve böylelikle raunt sayısı arttıkça küme sayısı azalmış olur. Şekil 3.20'de görüldüğü üzere MASC'ın ilk rauntta ürettiği kümeler toplam düğüm sayısının yaklaşık yarısına eşittir ve raunt sayısı arttıkça toplam küme sayısı azalmıştır.

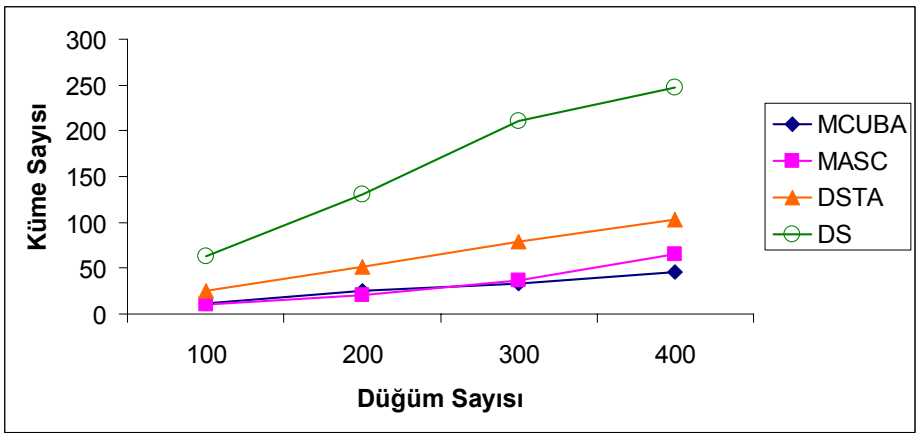


Şekil 3.20 MASC'ın ürettiği toplam küme sayıları

MCUBA'da küme seviyeleri *küme_üst_seviyesi* parametresi ile ayarlanabilir; böylelikle kümelerin sayısı ayarlanabilir. MCUBA algoritmasının *küme_üst_seviyesi*=10 olduğunda ürettiği kümelerin sayısı değişen düğüm sayısına ve derecesine göre ölçülmüştür. Şekil 3.21'de görüldüğü üzere MCUBA'nın ürettiği küme sayısı sabit ve kontrol edilebilirdir. MCUBA'nın *küme_üst_seviyesi* parametresi değiştirilerek küme dengesi ölçülmüştür. Şekil 3.22'de görüldüğü üzere *küme_üst_seviyesi* 5, 7, 10 ve 20 olarak değiştirildiğinde sırayla 19, 16, 10 ve 6 küme elde edilmiştir. Bu ölçümlerle birlikte MCUBA'nın *küme_üst_seviyesi* parametresinin küme sayısını kontrol ettiğini söyleyebiliriz.



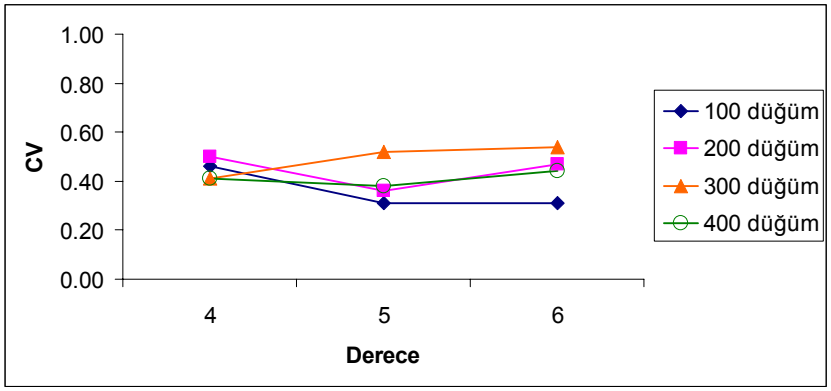
Şekil 3.21 MCUBA'nın ürettiği toplam küme sayıları



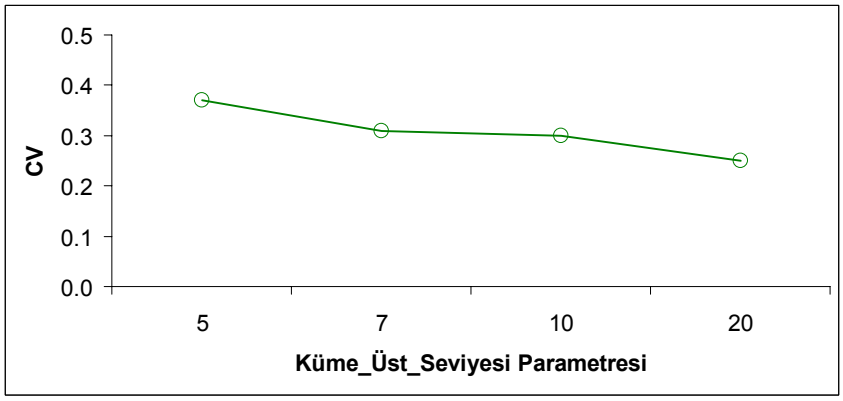
Şekil 3.22 Algoritmaların ürettiği toplam küme sayıları

Algoritmaların ürettiği küme sayılarının karşılaştırması Şekil 3.22'de verilmiştir. DS algoritması BHK üretmek zorunda olduğu için küme sayısı kontrol edilebilir değildir. Diğer taraftan DSTA'da *derinlik* parametresiyle küme seviyeleri ayarlanabilir. Şekil 3.22'de gösterilen algoritmalar içinde DS en kötü performansa sahiptir. DSTA *derinlik*=3 parametresiyle çalıştırıldığında üretilen küme sayıları doğrusal olarak az bir eğimle artar. 400 düğümlük ve ortalama düğüm derecesi 4 olan bir ağ üzerinde MCUBA *küme_üst_seviyesi*=10 ile çalıştırıldığında 46 küme üretir, MASC 4 rauntta 65 küme üretir, DSTA 102 küme üretir ve DS 247 küme

üretir. MCUBA ve MASC algoritmalarında küme sayıları düğüm sayılarının değişimlerine rağmen kararlı değerler gösterdiği için algoritmaların iyi performans gösterdiğini söyleyebiliriz. Bu algoritmalar alt seviyedeki kümeleri kaynaştırıp üst seviyedeki kümeleri oluşturdukları için küme sayıları kontrol edilebilir.

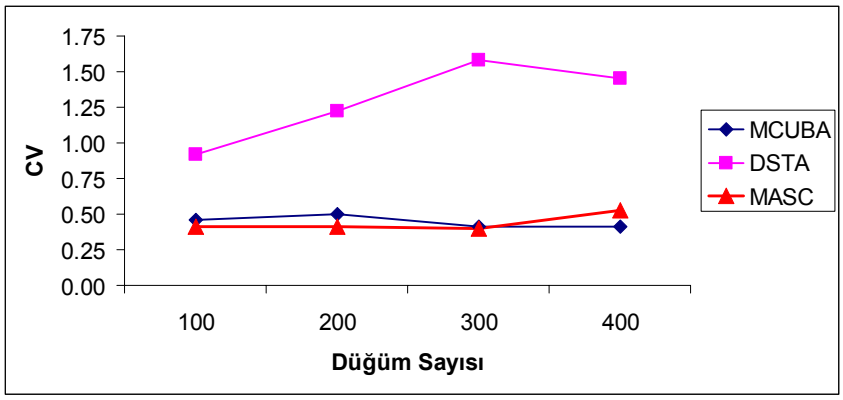


Şekil 3.23 MCUBA'nın ürettiği CV değerleri



Şekil 3.24 MCUBA'nın *küme_üst_seviyesi*ne karşı ürettiği CV değerleri

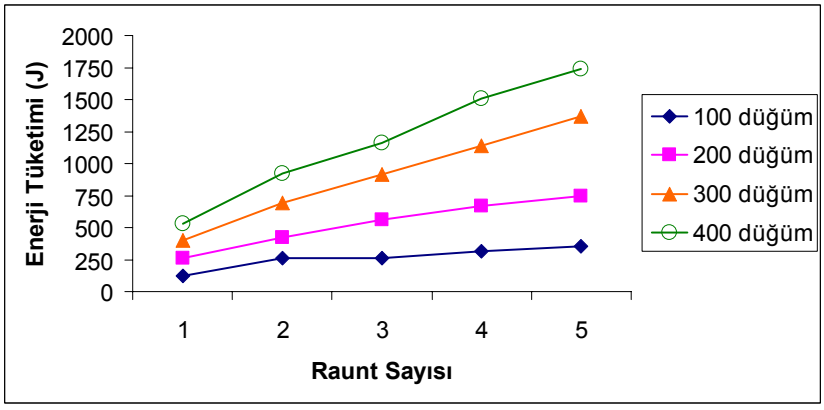
Algoritmaların kümeleme performansını ölçmek için ikinci olarak kümelerin dengesini inceledik. Bunu yapmak için varyasyon katsayısı (*coefficient of variation* (CV)) değerlerini ölçümlerimize göre hesapladık. Varyasyon katsayısı standard sapma / ortalama olarak hesaplanır. Eğer CV değeri 1 den küçükse dağılımın varyansı düşük, değilse varyansı yüksektir. MCUBA'nın CV değerlerini *küme_üst_seviyesi*ne göre ölçtük. Şekil 3.23'de görüldüğü üzere MCUBA'nın *küme_üst_seviyesi*=10 olduğunda ölçülen CV değerleri, ortalama düğüm derecesi değişimleriyle kararlıdır. Bu ölçümlerdeki ortalama CV değeri 0.43'tür. Ortalama düğüm derecesini 4'e sabitleyip MCUBA'nın *küme_üst_seviyesi*ni 5, 7, 10 ve 20 olarak değiştirip CV değerlerini ölçtük. Şekil 3.24'de görüldüğü üzere CV değerleri kararlılık gösterirler ve 0.40 ile 0.25 arasındadırlar. MCUBA'nın *küme_üst_seviyesi*=10, MASC'ın raunt sayısı 4, DSTA'da *derinlik*=3 olduğunda alınan ölçümler Şekil 3.25'te gösterilmiştir. DS algoritması kontrol edilebilir sayıda küme oluşturmadığı için CV değerleri ölçülmemiştir. Ortalama CV değerleri MASC için 0.44, MCUBA için 0.45, DSTA için 1.29'dur.



Şekil 3.25 Algoritmaların CV değerlerinin karşılaştırılması

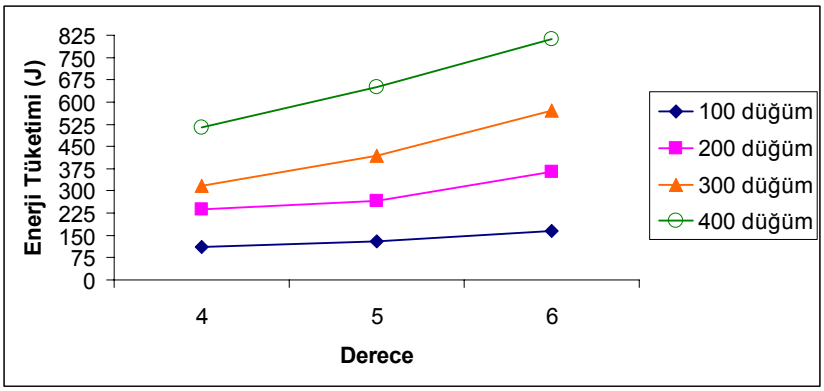
3.5.4. Enerji Tüketimleri

Algoritmaların enerji etkin çalışması TDA için önemli hedeflerdendir. Algoritmaların kümeleme ve omurga oluşturma sırasındaki enerji etkinliklerini ölçtük. Enerji tüketimlerinin mesaj transferleri sırasında oluştuğunu varsaydık. Şekil 3.26'da görüldüğü üzere MASC'ın enerji tüketimi raunt sayısı ve düğüm sayısı arttıkça doğrusal olarak artar. Şekil 3.27'de görüldüğü üzere MCUBA'nı enerji tüketimi *küme_üst_seviyesi*=10 olduğunda düğüm derecesi ve düğüm sayısıyla doğrusal olarak artış gösterir.

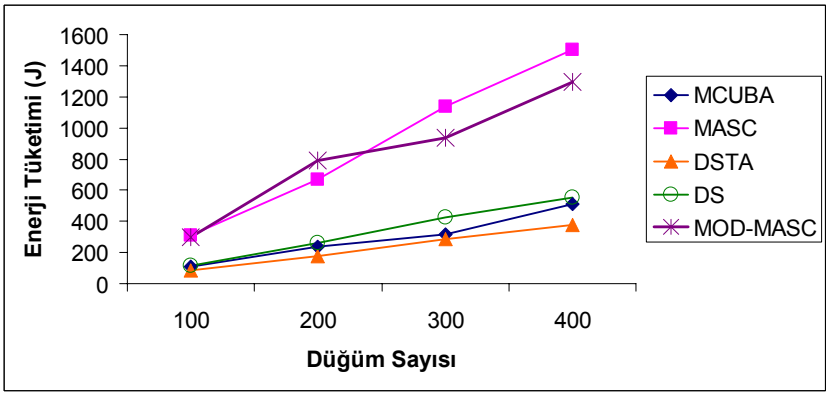


Şekil 3.26 MASC'ın enerji tüketimleri

Şekil 3.28'de algoritmaların enerji tüketimlerinin karşılaştırılması verilmiştir. DS ve DSTA algoritmaları MASC ve MCUBA'dan daha az mesaj kullandıkları için daha az enerji harcarlar. Bu enerji tüketiminin ana sebebi küçük seviyedeki kümeleri birleştirip büyük seviyede kümeler elde etmektir. MCUBA'nın enerji tüketimini azaltmak için KMO tekniği uygulanmıştır. Şekil 3.28'de görüldüğü üzere MCUBA'nın enerji tüketimi, DS ve DSTA ile çok yakın çıkmıştır. Ortalama olarak DSTA 0.90J, MCUBA 1.16J, DS 1.32J, MOD-MASC 3.28J, MASC 3.53J düğüm başına enerji harcamıştır.



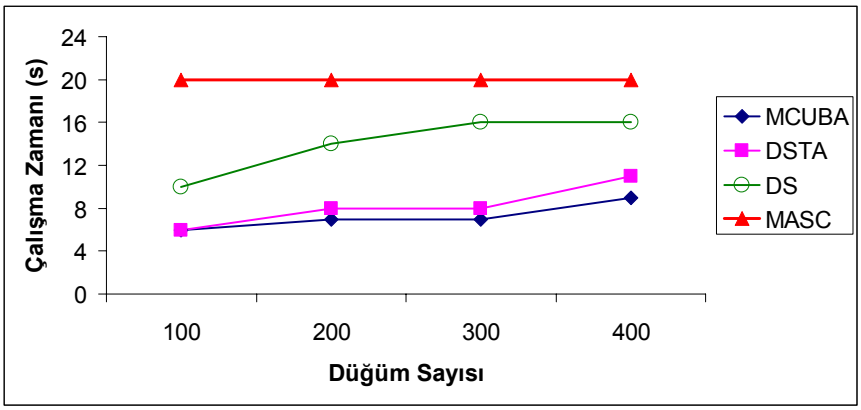
Şekil 3.27 MCUBA'nın enerji tüketimleri



Şekil 3.28 Algoritmaların enerji tüketimleri

3.5.5. Çalışma Zamanları

MCUBA'nın *küme_üst_seviyesi*=10, MASC raunt sayısı 4, DSTA'da derinlik=3 olduğunda ve DS algoritmalarının çalışma zamanlarını (*run time, wall clock time*) ölçtük. Şekil 3.15'de görüldüğü üzere en kötü durumda dağıtık eşleme yaklaşık 3s çalışır. Bizim ölçümlerimizde, MASC'ın bir raundu en kötü durumda 5s çalışır, bu durumdan dolayı bir raunt süresini 5s olarak çalıştırdık. MASC algoritmasını çalıştıran düğümler işleri bittikten sonra boşta olmalarına rağmen raunt sonunu beklerler. MCUBA'da düğümler asenkron çalıştıkları için bütün düğümler için aynı anda bekleme zamanı yoktur. 4 raunt çalışan MASC algoritmasının çalışma zamanı sabit 20s'dir. DSTA'da her düğüm aldığı mesajı sadece bir kez komşularına iletir. DS'de bir düğüm durumunu bütün komşularından gelen mesajlara göre karar verir. Bu tür algoritmalarda düğümler yaklaşık aynı zamanda komşularıyla haberleştikleri için mesaj çarpışmaları sıklıkla olabilir. Bu çarpışmaları engellemek için IEEE 802.11 MAC katmanı zamanlayıcı kullanarak paket gönderim zamanlarını belirler; fakat bu durum çalışma zamanını yükseltir. Şekil 3.29'da görüldüğü üzere algoritmalar içinde MCUBA en iyi performansı gösterirken, DSTA iyi performans gösterir, MASC en kötü performansı gösterir.



Şekil 3.29 Algoritmaların çalışma zamanları

3.6. Tartışmalar ve Uygulamalar

Bu bölümde MCUBA ve MASC algoritmalarının uygulamalarını ve genişletilmelerini tartışıyoruz. MCUBA ve MASC önceki bölümlerde anlatıldığı üzere güçlü kenarların seçimi prensibiyle çalışırlar. Ns2'da uygulanan IEEE 802.11 standartlarında hazır gelen RSS değerlerini kenar kalite göstergesi olarak kullandık. Bu gösterge dışındaki diğer ölçütler kullanılabilinir. Lal sadece bir kaç ölçüme bakılarak statik bir TDA üzerinde etkin bir kenar kalite göstergesi hesaplanabileceğini söylemiştir (Lal et al., 2003). Ayrıca Lal harcanan enerjiyle düğümler arasındaki bağlantının kalitesi arasında güçlü bir ilişki olduğunu söylemiştir. Bundan dolayı TDA üzerinde enerji etkin çok zıplamalı (*multi hop*) yönlendirme işlemindeki tasarım parametrelerinden bir tanesi bağlantı seçimidir. MCUBA, DS ve DSTA üzerinde çalışabilecek ve her düğümün çevreden algılama yapıp çıkış düğümüne doğru bilgisini gönderdiği örnek bir çevre izleme uygulaması tanımlayalım. Bu uygulamanın ve TDAnın parametreleri şu şekilde olsun:

- N: Toplam düğüm sayısı
- Δ: Ortalama düğüm derecesi
- T: Uygulamanın çalışacağı zaman
- p: Kümeleme algoritmaları için periyot zamanı
- E: Bir mesaj transferindeki ortalama harcanan enerji
- V: Düğümler tarafından algılanan toplam olaylar
- s: MCUBA'nın *küme_üst_seviyesi*

Olayı algılayan düğümden olay ile ilgili mesajın çıkış düğümüne doğru gitmesinin ortalama $\log_2(N)$ zıplama sürdüğünü varsayalım. Genel olarak TDA uygulamalarında düğümlerin pilleri bitebilir veya düğümlerin çeşitli sebeplerden dolayı çalışması durabilir. Bu yüzden hata toleransını sağlamak için küme

uygulaması periyodik olarak çalıştırılır. MCUBA'nın mesaj karmaşıklığı düğüm başına O(Δ*s*), DS ve DSTA'nın güvenli iletişim için O(Δ)'dir. Bölüm 3.5.2'deki benzetim sonuçlarında DSTA'nın ortalama kenar kalitesi *W*, DS'nin 2.3 *W*, MCUBA'nın 7 *W* olarak bulunmuştur. MCUBA, DS ve DSTA bir mesaj iletim sırasında tekrar gönderim sayılarına α, β, γ diyelim. Lal'in çalışmasından (Lal et al., 2003) yola çıkarak ortalama kenar ağırlıklarına bağlı olarak $\alpha < \beta < \gamma$ diyebiliriz. Duyarga ağında harcanan toplam enerji kümeleme algoritmasının harcadığı enerji ve uygulamanın tekrar gönderimlerle birlikte harcadığı enerjiye eşittir. MCUBA (*M*), DS (*D*), ve DSTA (*K*)'nın toplam enerji tüketimleri şöyledir:

$$M = ((N\Delta s)\ \lfloor T/p \rfloor + \log_2(N)V)\ \alpha\ E \qquad (3.1)$$

$$D = ((N\Delta)\ \lfloor T/p \rfloor + \log_2(N)V)\ \beta\ E \qquad (3.2)$$

$$K = ((N\Delta)\ \lfloor T/p \rfloor + \log_2(N)V)\ \gamma\ E \qquad (3.3)$$

Denklem 3.2 ve denklem 3.3'e bakarak $\beta < \gamma$ olduğundan dolayı *D*'nin *K*'den küçük olduğunu söyleyebiliriz. Bu karşılaştırmayı yaptıktan sonra *M* ve *D*'yi karşılaştıralım. Denklem 3.1 ve denklem 3.2'den *D*-*M*

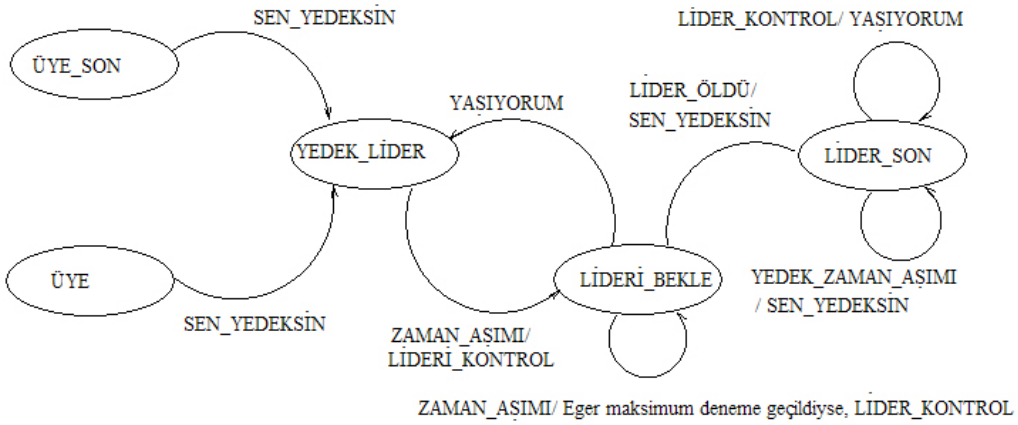
$$D\text{-}M = ((N\Delta)\ \lfloor T/p(\beta - s\alpha) \rfloor + \log_2(N)V(\beta-\alpha)E \qquad (3.4)$$

Denklem 3.4'den yola çıkarsak olay sayısı arttıkça, MCUBA, DS'den daha az enerji harcar. $\beta > s\alpha$ için, değişen düğüm sayıları, düğüm dereceleri ve uygulama zamanları için MCUBA, DS'den daha az enerji harcar. Uygulama tasarımcısı kümeleme algoritmasını seçmeden önce bu kriterleri göz önüne alabilir.

MCUBA ve MASC algoritmaları farklı alt katman duyarga ağı protokolleriyle birlikte çalışabilir. İki algoritma sadece gönder (*send*) ve al (*receive*) temel işlevlerine ihtiyaç duyar. Gönder ve al işlevleri MAC katmanlarındaki (IEEE Std 802.15.4TM-2003, 2003; IEEE 802.11 StandardWorking Group, 2007; Ye et al., 2004; Choi et al., 2009; Polastre et al., 2004) çok önemli bir işlevdir. IEEE 802.15.4'de sinyal-gürültü-oranı ve tespit edilen enerjiyi birlikte kullanan bağlantı kalite göstergesi (*link quality indicator*) bulunur. S-MAC ve TRAMA'da inceleme (*probe*) paketleri ve KMO tekniği kenar kalitesini göstermek için kullanılır (Karl and Willig, 2005). B-MAC (Polastre et al., 2004), M-MAC (Choi et al., 2009) ve WMEWMA (Woo et al., 2003) kenar kalitesini hesaplayabilen diğer alt seviye protokolledir.

Bazı kümeleme yaklaşımlarında, omurga oluşturma algoritmaları kümeleme algoritmalarının üst katmanına konur (Dagdeviren and Erciyes, 2006). Bu yaklaşımlarda önce ağ kümelere ayrılır, daha sonra omurga oluşturulur. MASC algoritması bir omurga oluşturma algoritmasıyla beraber bu yapıda kullanılabilinir. Örneğin, dengeli kümelerle birlikte bir halka omurgası MASC ve omurga oluşturma algoritması (*backbone formation algorithm* (BFA)) (Dagdeviren and Erciyes, 2006)

ile birlikte oluşturulabilinir. Omurga oluşturma için ikinci bir yöntem liderlerin kapsama alanlarını genişletmesidir (Younis and Fahmy, 2004). Kümeleme işlemi bittikten sonra, küme liderleri birbirleriyle haberleşmek için kapsama alanlarını genişletirler. Bu yaklaşım MASC için uygundur. MASC'ın enerji tüketimi MCUBA'da yapıldığı gibi KMO tekniğiyle azaltılabilinir.



Şekil 3.30 Hata toleransı eklentisi için sonlu durum makinesi

Küme liderleri veri işleme, birleştirme ve yönlendirmeden sorumlu oldukları için enerjilerini sıradan düğümlerden daha çabuk harcayabilir. Bundan dolayı kümeleme algoritmaları tarafından lider dönüşümü sağlanabilir. MCUBA ve MASC algoritmaları enerji-etkin küme lideri dönüşümü sağlayacak biçimde genişletebilir. Bu durumda ağ, $G_w(V_w,E_w)$ düğüm ve kenar ağırlıklı yönsüz çizge modeli olarak genişletilebilir. Düğüm ağırlıkları enerjileriyle temsil edilebilir. MASC ve MCUBA'nın lider seçim politikası bu model kullanarak değiştirilebilinir. Küme lideri seçilirken adaylardan enerjisi yüksek düğüm lider olarak seçilir, diğer aday yedek (*backup*) olarak atanır. Bu metoda enerji-etkin küme lideri seçimini sağlar ve algoritmalar üzerinde çok az değişimler uygulanabilir.

Algoritmaların periyodik olarak çalıştırılması hata toleransını sağlayan tek yöntem değildir. Diğer bir metot yedek küme lideri seçimidir (Younis and Fahmy, 2004). MCUBA ve MASC algoritmaları yedek küme liderini yenilemeli seçmeyi sağlayacak biçimde tasarlanabilinir. Şekil 3.30'da gösterilen sonlu durum makinesinde yapılması gereken eklentiler verilmiştir. Bir düğüm *LİDER_SON* durumuna geçtiğinde yedek bir küme lideri seçer ve *SEN_YEDEKSİN* (*YOU_ARE_BACKUP*) mesajı gönderir. Basit olarak liderin komşusu olan ve aynı küme içinde olan enerjisi en yüksek düğüm yedek lider olarak seçilebilir. *ÜYE* veya *ÜYE_SON* durumundaki bir düğüm *YEDEKSİN* mesajını aldığında *YEDEK_LİDER* (*BACKUP_LEADER*) durumuna geçiş yapar. *YEDEK_LİDER* durumundaki düğüm periyodik olarak *LİDERİ_KONTROL* (*POLL_LEADER*) mesajı gönderip *WT_LEADER* durumuna geçer. LİDER_SON durumundaki düğüm *LİDER_KONTROL* mesajı aldığında *YAŞIYORUM* (*HEARTBEAT*) mesajı gönderir. Küme liderinden cevap gelmezse yedek düğüm *kontrol_sayısı* (*poll_number)* kadar

daha *LİDER_KONTROL* mesajı gönderir. Yedek lider, küme liderinin işlevini yitirdiğini anlarsa yeni bir yedek lideri *YEDEKSİN* mesajı gönderip seçer ve kendisi *LİDER_SON* durumuna geçer. Yedek lider düğüm de işlevini yitirebilir. Böylece lider düğüm *LİDER_KONTROL* mesajını almaz. Bu durumda lider düğümde *YEDEK_ZAMAN_AŞIMI* olur, yeni yedek lider seçer ve yeni lidere *YEDEKSİN* mesajını gönderir. Bu yaklaşım küme lideri veya yedek düğümden herhangi biri işlevini yitirdiğinde kullanılabilinir. Eğer ikisi de aynı anda işlevini yitirirse yeni teknikler uygulanmalıdır. Sıradan düğümlerde oluşan hatalardan kurtulmak için genişletilmiş yönlendirme tabloları kullanılabilinir. Bu tablolarda bir düğüm mesajını göndereceği hedef bir düğüm için birden fazla yönlendirici kimliği tutabilir. Bu bilgi mesaj değişimleri sırasında veya KMO tekniği kullanılarak elde edilebilir. Buna karşın yönlendirme tablosundaki büyüme bellek tüketimine yol açabileceğinden dolayı bir eniyileme yöntemi sağlanmalıdır.

4. AĞIRLIKLI BAĞLI HÂKİM KÜME ALGORİTMALARI

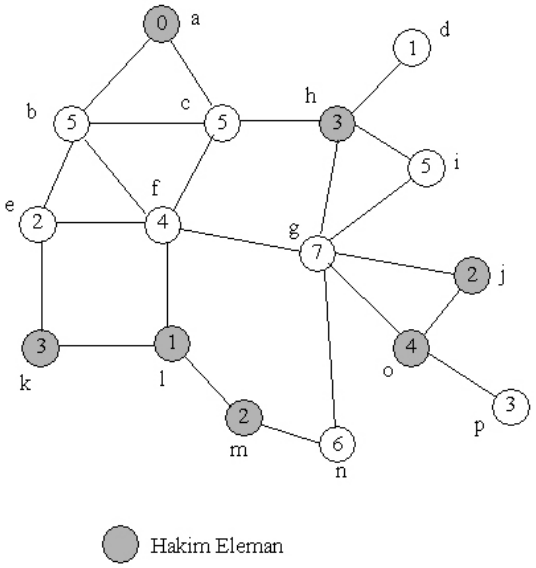
Bu bölümde önerdiğimiz ağırlıklı bağlı hâkim küme algoritmalarının senkron ve yarı asenkron sürümleri, analizleri ve performans değerlendirmeleri verilecektir.

4.1. Ağırlıklı Hâkim Küme Algoritmaları

Bu bölümde senkron ağırlıklı küme örtüsü tabanlı hâkim küme algoritması ve senkron ağırlıklı Steiner ağacı algoritması anlatılacaktır.

4.1.1. Genel Fikir

Bu bölümde önerdiğimiz algoritmalar Şekil 2.6'da verilen Guha'nın algoritmasının ilk basamağında kullan algoritmanın dağıtık versiyonlarıdır. Bölüm 2'de de anlatıldığı üzere Guha'nın algoritması 2 basamaklı olup bağlı HK kurmaktadır. İlk basamakta aç gözlü küme örtüsü tabanlı HK bulur. Bu algoritmanın yaklaşım oranı S en küçük hâkim kümenin ağırlığı olmak üzere ln (S)'dir. Bu algoritma Şekil 2.7'de verilmiştir ve CENTSET (Central weighted set cover based dominating set algorithm) olarak isimlendirilmiştir.



Şekil 4.1 CENTSET'in örnek uygulaması

Şekil 4.1'de CENTSET'in örnek bir uygulaması verilmiştir. Düğümlerin maliyeti içlerine yazılmıştır. İlk olarak a düğümünün ağırlık oranı (a düğümünün maliyeti / (a düğümünün maliyeti + b düğümünün maliyeti + c düğümünün maliyeti)) olarak hesaplanıp, 0 bulunur ve a düğümü hâkim küme içine alınır. Bir sonraki

adımda l düğümünün ağırlık oranı 1/9 olarak hesaplanıp hâkim küme içine alınır. Daha sonra sırasıyla düğüm j, düğüm h, düğüm m, düğüm k ve düğüm o seçilir.

4.1.2. Yayma Tabanlı Senkron Ağırlıklı Hâkim Küme Algoritması

Bu bölümde yayma tabanlı senkron ağırlıklı hâkim küme algoritmasının genel fikri, tanımlanması ve örnek uygulaması verilmiştir.

4.1.2.1. Genel Fikir

Yayma tabanlı senkron ağırlıklı hâkim küme algoritması (Flooding based synchronous weighted dominating set algorithm (FLOODSET)) CENTSET algoritmasının çalışmasına birebir uygun biçimde dağıtık olarak tasarlanmasıyla yapılmıştır. CENTSET'in dağıtık tasarlanması için en kolay uygulanabilecek fikrin FLOODSET olduğunu söyleyebiliriz. FLOODSET senkron rauntlara bölünmüş olup, her rauntta düğümlerin ağırlık oranını yaymasıyla çalışır. Düğümler birbirlerinin ağırlık oranlarını bildikleri için en küçük ağırlık oranını bulabilirler. Bir sonraki bölümde algoritmanın basamakları anlatılmıştır.

4.1.2.2. Tanım

FLOODSET Algoritmasının basamakları Şekil 4.2'de verilmiştir. Her düğüm ilk rauntta komşularına enerji değerini gönderir. Bu algoritmada bir düğümün ağda toplam kaç adet düğüm olduğunu bildiğini varsayıyoruz. Bu bilgi çıkış düğümüne kullanıcı tarafından verildikten sonra, çıkış düğümü tarafından ağa yayılabilir. FLOODSET algoritmasında başlangıçta bir düğüm komşularının hepsinden enerji değerlerini topladığında ağırlık oranı hesaplar ve hesapladığı bu oranı *AĞIRLIK_ORANI(düğüm_kimliği, ağırlık_oranı)* mesajıyla ağa gönderir. Bütün düğümler birbirlerinden *AĞIRLIK_ORANI* mesajını alıp, ağdaki en küçük ağırlığa sahip düğümü bulabilir. En küçük ağırlığa sahip düğüm komşularına *HÂKİM_ELEMAN(düğüm_kimliği)* mesajı gönderir, bu mesajı alan komşuları da *ÖRTÜLDÜM(düğüm_kimliği)* mesajı gönderir ve ağırlık oranlarını günceller. *ÖRTÜLDÜM* mesajını alan düğümler ağırlık oranlarını günceller. Bir sonraki rauntta düğümler benzer işlemleri tekrar ederler. Eğer bir düğümün komşularının tümü hâkim eleman veya örtülmüşse algoritmayı bitirir.

m düğümü için algoritma:
1. Komşuna enerjini gönder.
2. Maliyet=1 / Enerji ve Ağırlık oranı = (Maliyet / Düğümün maliyetiyle birlikte örtülmemiş komşularının maliyeti) denklemiyle hesapla.
3. Tüm ağa *AĞIRLIK_ORANI* mesajıyla hesaplanan ağırlık oranını gönder.

4. Hâkim Küme içerisindeki düğümler dışındaki tüm ağdan *AĞIRLIK_ORANI* mesajını bekle.
5. Eğer ağırlık oranı ağdaki hâkim küme dışındaki tüm düğümlerden küçükse hâkim küme içine gir, *HÂKİM_ELEMAN* mesajı gönder ve algoritmayı sonlandır.
6. Eğer *HÂKİM_ELEMAN* mesajı alırsam *ÖRTÜLDÜM* mesajı gönder. Komşularının tümü hâkim eleman veya örtülmüşse algoritmayı bitir.
7. Eğer *ÖRTÜLDÜM* mesajı alırsan ve komşularının tümü hâkim eleman veya örtülmüşse algoritmayı bitir.
8. Yeni raunt başladığında ikinci basamağa dön.

Şekil 4.2 FLOODSET'in basamakları

FLOODSET algoritmasının tasarımı üzerindeki küçük değişikliklerle, bir düğümün ağdaki bütün düğümleri bilmesi gereksiniminden arınabilir. Şekil 4.3'de değiştirilmiş FLOODSET algoritması verilmiştir. Bu algoritma tek ve çift sayılı rauntlara ayrılmıştır. Tek sayılı rauntlarda düğümler ağırlık oranlarını hesaplayıp ağa yayarlar. Çift sayılı rauntlarda hâkim küme içine giren düğüm komşularına *HÂKİM_ELEMAN* mesajını gönderir, bu mesajı alan düğümler *ÖRTÜLDÜM* mesajını gönderir, böylelikle ağdaki düğümler ağırlık oranlarını tekrardan hesaplar. Şekil 4.3'de verilen değiştirilmiş FLOODSET algoritmasının çalışma süresi, Şekil 4.2'e verilen orijinal algoritmanın çalışma süresinin iki katıdır.

m düğümü için algoritma:
En küçük ağırlık oranı ← ∞
En küçük ağırlık oranına sahip düğümün kimliği ← *m* düğümünün kimliği
Tek Sayılı Rauntlarda:
1. Eğer raunt sayısı 1'se komşuna enerjini gönder.
2. Maliyet=1 / Enerji ve Ağırlık oranı = (Maliyet / Örtülmemiş komşularının maliyeti) denklemiyle hesapla.
3. Tüm ağa *AĞIRLIK_ORANI* mesajıyla hesaplanan ağırlık oranını gönder.
4. *AĞIRLIK_ORANI* mesajı alınca eğer alınan ağırlık oranı < en küçük ağırlıksa: en küçük ağırlık oranı ← alınan ağırlık oranı ve en küçük ağırlık oranına sahip düğümün kimliği ← alınan düğümün kimliği işlemlerini yap.

Çift Sayılı Rauntlarda:
1. Eğer en küçük ağırlık oranına sahip düğümün kimliği = *m*'se hâkim küme içine gir, *HÂKİM_ELEMAN* mesajı gönder ve algoritmayı sonlandır.
2. Eğer *HÂKİM_ELEMAN* mesajı alırsam *ÖRTÜLDÜM* mesajı gönder. Komşularının tümü hâkim eleman veya örtülmüşse algoritmayı bitir.
3. Eğer *ÖRTÜLDÜM* mesajı alırsan ve komşularının tümü hâkim eleman veya örtülmüşse algoritmayı bitir.

Şekil 4.3 Değiştirilmiş FLOODSET'in basamakları

4.1.2.3. Örnek Uygulama

Şekil 4.1'deki ağ üzerinde FLOODSET'in örnek uygulamasını anlatacağız. İlk raundun başında bütün düğümler ağırlık oranların hesaplarlar. A düğümü ağırlık oranını 0/(5+5)=0, b düğümünün ağırlık oranını 5/(5+0+5+4+3)=0.29, c düğümü ağırlık oranını 5/(5+0+5+4+3)=0.29 olarak bulur, diğer düğümler de benzer olarak ağırlık oranını hesaplar. Her düğüm ağa *AĞIRLIK_ORANI* mesajını yayıp, birbirinin ağırlık oranını öğrenir. A düğümü ağdaki tüm düğümlerden ağırlık oranı mesajını aldıktan sonra kendisinin en küçük ağırlık oranına sahip olduğunu hesaplar ve komşularına *HÂKİM_ELEMAN* mesajı gönderir. B düğümü ve c düğümü, a düğümünde *AĞIRLIK_ORANI* mesajını aldıktan sonra *ÖRTÜLDÜM* mesajı gönderir ve a düğümü örtüldüyse kendi ağırlık oranlarını güncellerler. E düğümü, f düğümü ve h düğümü *ÖRTÜLDÜM* mesajını aldıktan sonra b ve c düğümleri örtüldüklerinden dolayı ağırlık oranlarını güncellerler. Aynı zamanda b ve c düğümleri birbirlerinden *ÖRTÜLDÜM* mesajını aldıkları için ağırlık oranlarını tekrar günceller. İkinci raundun başında düğümler ağırlık oranlarını tekrardan ağa yayarlar. Bu rauntta l düğümünün ağırlık oranı 1/(4+2+2+1)=0.11 olduğundan dolayı l düğümü ağırlık oranlarını topladıktan sonra hâkim küme içine girer. Benzer olarak sırasıyla ilerleyen rauntlarda düğüm j, düğüm h, düğüm m, düğüm k ve düğüm o seçilir ve algoritma sonlanır.

4.1.3. Senkron Ağırlıklı Hâkim Küme Algoritması

Bu bölümde senkron ağırlıklı hâkim küme algoritmasının (synchronous weighted dominating set algorithm (SSET)) genel fikri, tanımlaması ve örnek uygulaması anlatılmıştır.

4.1.3.1. Genel Fikir

FLOODSET algoritmasının mesaj karmaşıklığı Bölüm 4.3'de gösterileceği üzere $O(N^3)$'tür. Bu mesaj karmaşıklığını asimptotik olarak düşürebilmek için öncelikle CENSET algoritmasında değişiklikler yaptık ve Şekil 4.4'teki merkezi algoritmayı elde ettik. Bu algoritmayı MODSET (Modified central weighted set cover based dominating set algorithm) olarak isimlendirelim.

1. Tüm düğümler örtüldüyse algoritmayı bitir.
2. En fazla iki zıplama uzaklığında olan komşularından daha küçük ağırlık oranına sahip olan düğümleri HK içine koy.
3. Seçilen düğümleri ve komşularını ört.
4. Seçilen düğümlerin komşularının ağırlıklarını güncelle.
5. Birinci basamağa dön.

Şekil 4.4 MODSET'in basamakları

MODSETle birlikte en fazla iki zıplama uzaklığında olan komşularından daha küçük ağırlık oranına sahip olan düğümleri seçiyoruz. Böylece aynı iterasyon içinde birden fazla düğümü hâkim eleman olarak seçebiliyoruz. MODSET algoritması dağıtık olarak tasarlamak için çok uygundur. Önerdiğimiz senkron dağıtık ağırlıklı küme örtüsü tabanlı HK algoritması (SSET) MODSET'in dağıtık versiyonu olup kendi içinde senkron rauntlara ayrılır. Algoritma başlarken her düğümün komşularını ve ağırlıklarını bildiklerini varsayılmıştır. SSET algoritmasının tanımı bir sonraki bölümde verilmiştir.

4.1.3.2. Tanım

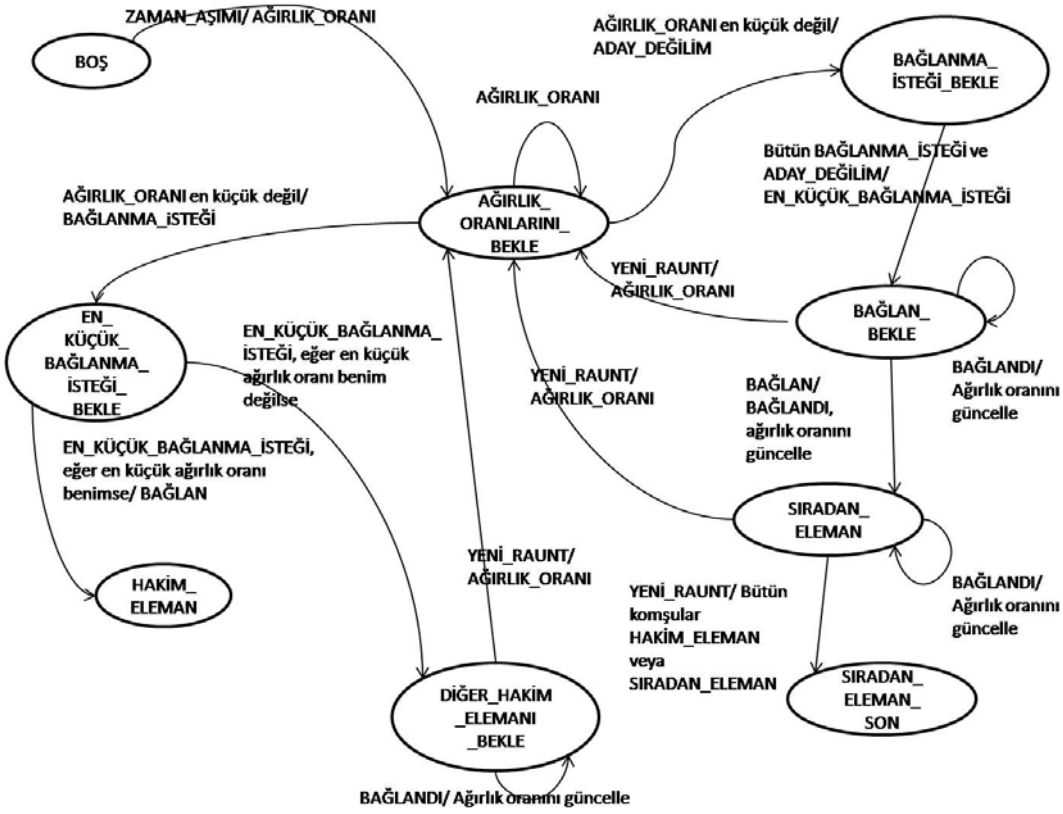
SSET algoritmasının basamakları şu şekildedir:

1. Her raundun başında düğümler ağırlık oranlarını *AĞIRLIK_ORANI (düğüm_kimliği, ağırlık_oranı)* mesajı içinde gönderirler. Ağırlık oranı (*A*) = (Düğümün ağırlığı / örtülmemiş komşularının ağırlık oranı) olarak hesaplanır.

2. Komşuları içinden en küçük ağırlık oranına sahip düğümler komşularına *BAĞLANMA_İSTEĞİ(düğüm_kimliği)* mesajı gönderirler. Eğer bir düğüm komşuları içinde en küçük ağırlığa sahip değilse *ADAY_DEĞİLİM(düğüm_kimliği)* mesajı gönderir. Bir düğüm komşularının hepsinden *BAĞLANMA_İSTEĞİ* veya *ADAY_DEĞİLİM* mesajı topladıktan sonra topladığı *BAĞLANMA_İSTEĞİ* mesajlarının içinde en küçük ağırlık oranını *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ (düğüm_kimliği)* mesajı olarak komşularına iletiyor. Bu mesajın içinde kendi komşularının *A* değerleri vardır. Böylece her düğüm 2 zıplama uzaklıktaki komşularının *A* değerlerini öğrenir.

3. Bir düğüm 2 zıplama uzaklıktaki tüm komşularından daha küçük *A*'ya sahipse o rauntta *HÂKİM_ELEMAN* olur, komşu düğümlerine *BAĞLAN(düğüm_kimliği)* mesajı gönderir ve böylece örtülmemiş komşu düğümler *SIRADAN_ELEMAN* olur. *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN* düğümler örtülmüş (*covered*) düğümlerdir. *SIRADAN_ELEMAN* düğümler *BAĞLAN* mesajını aldıktan sonra örtüldüklerini gösteren *BAĞLANDI(düğüm_kimliği)* mesajını gönderirler.

4. Bir düğümün kendisi ve tüm komşuları örtülmüşse, algoritmayı bitiriyor, değilse *YENİ_RAUNT* kesmesi ile birinci basamağa geçiyor. Algoritmanın sonlu durum makinesi Şekil 4.5'de verilmiştir.



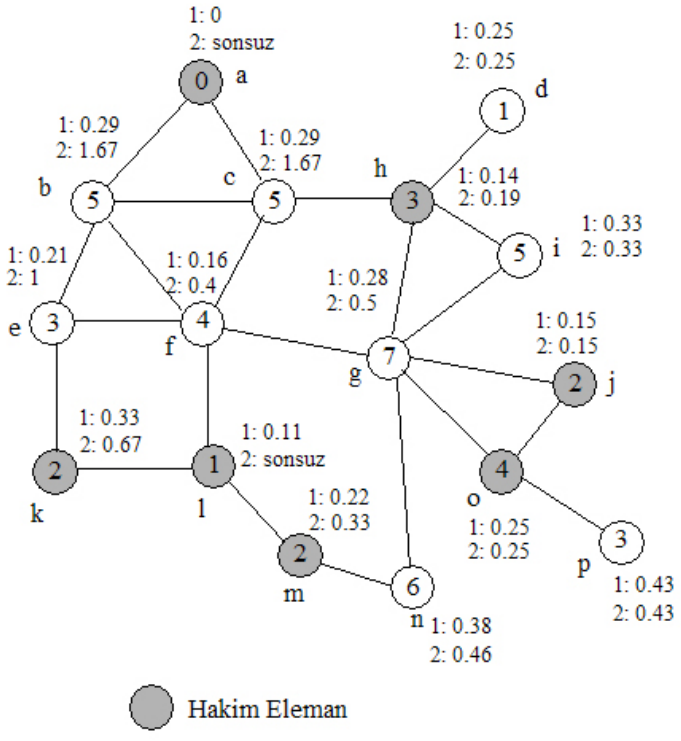
Şekil 4.5 SSET Algoritmanın Sonlu Durum Makinesi

4.1.3.3. Örnek Uygulama

Bu bölümde Şekil 4.6'daki ağ üzerinde SSET algoritmasının örnek uygulamasını yapacağız. Bu şekil üzerinde her düğümün birinci ve ikinci raunttaki ağırlık oranları verilmiştir, "1:" ile gösterilen ilk raundu, "2:" ile gösterilen ikinci raunttaki ağırlık oranını verir. Bu ağırlık oranları (düğümün maliyeti / örteceği komşularının maliyeti) olarak hesaplandı. Örneğin düğüm c'nin ilk raunttaki ağırlık oranı ((c'nin maliyeti)/(c'nin maliyeti+a'nın maliyeti+b'nin maliyeti+f'in maliyeti+h'nin maliyeti) = 5/ (0+5+3+4+5)) 0.29 olarak bulur. İlk rauntta düğüm a, ağırlık oranını 0 bulduktan sonra düğüm b ve düğüm c'ye *AĞIRLIK_ORANI* mesajı olarak bu ağırlık oranını gönderir ve *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçer. Düğüm b ağırlık oranını 0.29 bulur ve komşularına gönderir. Düğüm c'de benzer şekilde ağırlık oranını 0.29 olarak bulduktan sonra bu ağırlık oranını *AĞIRLIK_ORANI* mesajı içinde gönderir. Düğüm a, düğüm b ve düğüm c'den ağırlık oranlarını topladıktan sonra komşuları içinde en küçük ağırlık oranına sahip olduğu için *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumuna geçiş yapar, *BAĞLANMA_İSTEĞİ* mesajını gönderir ve iki zıplama uzaklıktaki komşularının ağırlık oranlarını bekler.

Düğüm b ve düğüm c komşularından *AĞIRLIK_ORANI* mesajlarını topladıktan sonra en küçük ağırlık oranına sahip olmadıkları için *ADAY_DEĞİLİM* mesajını göndererek *BAĞLANMA_İSTEĞİ_BEKLE* durumuna geçip komşularından

ADAY_DEĞİLİM veya *BAĞLANMA_İSTEĞİ* mesajları toplamaya başlarlar. H düğümü de komşuları içinde en küçük ağırlığa sahip olduğu için, içinde c düğümünün de bulunduğu komşularına *BAĞLANMA_İSTEĞİ* mesajı gönderir. B ve c düğümleri komşularından *BAĞLANMA_İSTEĞİ* mesajlarını topladıktan sonra en küçük ağırlıklı düğümün ağırlığını 0 ve kimliğini a düğümü olarak bulup bu bilgiyi *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajı içinde gönderip *BAĞLAN_BEKLE* durumuna geçerler. A düğümü b ve c düğümlerinden *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajını aldıktan sonra komşularına *BAĞLAN* mesajı gönderir ve *HÂKİM_ELEMAN* durumuna geçer. *BAĞLAN* mesajını alan b ve c düğümü komşularına *BAĞLANDI* mesajını gönderir ve böylelikle komşularının ağırlık oranlarını güncellemesini sağlar. Benzer şekilde düğüm l'nin ağırlığı 0.11 olup bütün komşularından küçük olduğu için düğüm l *HÂKİM_ELEMAN* durumuna geçer, *BAĞLAN* mesajı gönderir böylelikle komşuları *SIRADAN_ELEMAN* olur. H düğümü *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajını aldığında düğüm a'nın kendinden daha az ağırlığa sahip olduğunu fark ettiği için *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumundan *DİĞER_HÂKİM_ELEMANI_BEKLE* durumuna geçer. Düğüm h'nin komşusu düğüm l *BAĞLAN* mesajı almadığı için *BAĞLAN_BEKLE* durumunda kalır. Benzer olarak *BAĞLAN* mesajını alamayan ve komşuları içinde en küçük ağırlığa sahip olamayan düğümler *BAĞLAN_BEKLE* durumunda kalır.



Şekil 4.6 SSET algoritması örnek uygulama

2. raunda başladığında *YENİ_RAUNT* kesmesiyle birlikte *DİĞER_HÂKİM_ELEMANI_BEKLE* durumundaki düğüm h, *BAĞLAN_BEKLE*

durumundaki düğüm l ve *SIRADAN_ELEMAN* durumundaki düğüm b ağırlık oranlarını hesaplayıp komşularına gönderir. Bu durumda olan diğer düğümler de aynı şekilde ağırlık oranlarını hesaplayıp komşularına gönderir. Düğüm a ve düğüm l, *HÂKİM_ELEMAN* durumunda olduklarından dolayı ağırlık oranlarını komşularına göndermez. Bu rauntta birinci raunda çok benzer olarak düğüm j ve düğüm m iki zıplama uzaklıktaki komşularından daha küçük ağırlığa sahip olduklarından dolayı hâkim küme içine girer. Şekil 4.6 üzerinde dikkat edilirse bu rauntta düğüm j'nin ağırlık oranı 0.15 ve düğüm m'nin ağırlık oranı 0.33 olup iki zıplama uzaklıktaki komşularından daha küçüktür. 3. rauntta geçmiş iki raunda benzer olarak düğüm h ve düğüm k hâkim küme içine girer. Bu raundun sonunda düğüm d, düğüm h'den başka komşusu olmadığı için *SIRADAN_ELEMAN_SON* durumuna geçer. 4. rauntta düğüm o hâkim küme içine girer. Böylece düğüm a, düğüm l, düğüm h, düğüm m ve düğüm o *HÂKİM_ELEMAN* durumunda algoritmayı tamamlarken, geri kalan düğümler algoritmayı *SIRADAN_ELEMAN_SON* durumunda bitirir.

4.1.4. Yarı Asenkron Ağırlıklı Hâkim Küme Algoritması

Bu bölümde yarı asenkron ağırlıklı hâkim küme algoritmasının (Semi-asynchronous weighted dominating set algorithm (ASYNSET)) genel fikri, tanımı ve örnek uygulamasını göstereceğiz.

4.1.4.1. Genel Fikir

ASYNSET algoritmasının tasarımında SSET algoritması α ve β senkronizerleri kullanılarak senkron sürümden asenkron sürüme dönüştürülmüştür. Senkronizerler yardımıyla senkron algoritmalar asenkron yapılabilir. Bu dönüşümden sonra rauntlar zamanlayıcı tarafından değil, *ÇIKIŞ_DÜĞÜMÜ* tarafından yayılan bir mesajla başlar. ASYNSET'de SSET'de olduğu gibi rauntlar olduğu için buna rağmen zamanlayıcı kullanılmasına gerek kalmadığı algoritma yarı asenkrondur. α ve β senkronizeri ile birlikte tanımlanmış 3 tip senkronizer bulunmaktadır (Lynch, 1996):

1. α senkronizeri: Bir düğüm işlemini bitirdikten sonra komşularına işlemini bitirdiğine dair *TAMAM* mesajı gönderir. Bir düğüm komşularının hepsinden *TAMAM* alırsa yeni işleme geçer.

2. β senkronizeri: Öncelikle ağdaki bütün düğümlerin oluşturduğu kökü belli olan bir ağaç kurulmalıdır. Ağdaki her düğüm ağacın üzerindeki ebeveynini ve çocuklarını bilmelidir. Bir düğüm işlemini bitirdikten sonra, çocuklarından *TAMAM* mesajı bekler, eğer bütün çocuklarından *TAMAM* alırsa ebeveynine *TAMAM* gönderir. Dikkat edilecek olursa ağacın en altındaki yaprak düğümlerin çocuğu olmadığı için işlemlerini bitirdikten sonra *TAMAM* mesajını gönderirler.

3. γ senkronizeri: α senkronizeri ve β senkronizerinin birlikte kullanıldığı senkronizerdir. Bu senkronizerde ağ kümelere bölünmüş her küme içinde kökü belli olan bir ağaç kurulmuştur. Kümelerin içinde β senkronizeri kullanılıp kümeler arasında α senkronizeri kullanılır.

4.1.4.2. Tanım

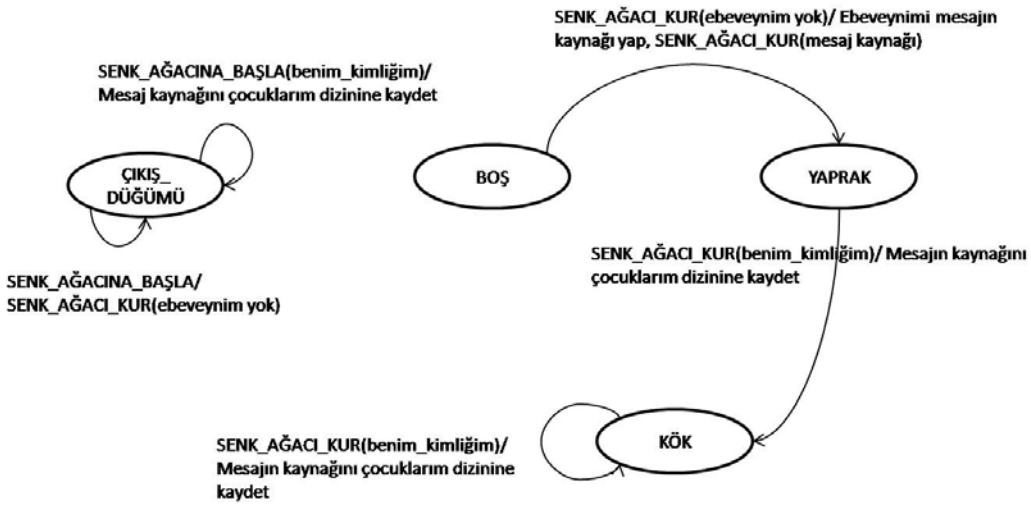
ASYNSET algoritmasının çalışma biçiminin genel olarak anlatıldığı basamaklar Şekil 4.7'de verilmiştir. Bu algoritmada öncelikle her düğüm SSET algoritmasında olduğu gibi 2 zıplama uzaklıktaki komşularının ağırlıklarını öğreniyor. 2 zıplamadaki komşularının ağırlıklarından küçük olanlar SSET algoritmasında olduğu gibi hâkim küme içine girip komşularını örtüyor. Buraya kadar anlatılan iki işlem SSET algoritmasındaki işlemler ile neredeyse birebir aynıdır. Fakat ASYNSET algoritmasında düğümler arasında zaman senkronizasyonu olduğu varsayılmadığından dolayı, düğümlerin yeni raunda başlamaları için rauntları tetikleyen zamanlayıcı dışında bir mekanizma kullanmaları gerekmektedir. Bu mekanizmalar α ve β senkronizerleri sayesinde sağlanabilir. β senkronizeri kullanabilmek için algoritma başlamadan önce düğümler arasında bir ağaç oluşturulduğu varsayılır. Algoritmanın üçüncü basamağında düğümler α senkronizeri ile ağırlıklarını güncelleyip, β senkronizasyon ağacı üzerinden yeni raunda geçmek için senkronizasyon işlemi yaparlar.

1.Her düğüm senkron algoritmada olduğu gibi 2 zıplama uzaklığındaki komşuların ağırlıklarını öğreniyor.

2. 2 zıplamadaki komşularının ağırlıklarından küçük olanlar senkron algoritmada olduğu gibi hâkim küme içine giriyor.

3. Her düğüm β senkronizasyon ağacındaki komşularından *TAMAM* mesajı aldıktan sonra ebeveynine *TAMAM* gönderiyor (Senkronizasyon işlemi). *ÇIKIŞ_DÜĞÜMÜ TAMAM* mesajı alınca bu raunt içinde yeni seçilen bir *HÂKİM_ELEMAN* yoksa *SON* mesajı gönderip algoritmayı bitiriyor, değilse *BAŞLA* mesajı gönderiyor ve sıradan düğümlerle birlikte basamak 1'e geçiyor.

Şekil 4.7 ASYNSET Algoritması basamakları



Şekil 4.8 ASYNSET algoritmasının çalışması için gerekli senkronizasyon ağacının oluşturulması

β senkronizerini kullanabilmek için gerekli senkronizasyon ağacını oluşturabilmek için DSTA algoritması (Erciyes et al. 2008) ebeveynlerin çocuklarını bileceği şekilde basit olarak düzenlenebilir. Şekil 4.8'de düzenlenmiş DSTA algoritmasının sonlu durum makinesi verilmiştir. Şekil 4.8 (a)'da *ÇIKIŞ_DÜĞÜMÜ* sonlu durum makinesi, Şekil 4.8 (b)'de sıradan düğümlerin sonlu durum makinesi verilmiştir. *ÇIKIŞ_DÜĞÜMÜ'*ne kullanıcıdan veya üst katmanlardan gelen *SENK_AĞACINA_BAŞLA* mesajından sonra *ÇIKIŞ_DÜĞÜMÜ* komşularına *SENK_AĞACI_KUR(düğüm_kimliği, ebeveyn_kimliği)* mesajını ağdaki komşularına senkronizasyon ağacını oluşturmaya başlamaları için gönderiyor. Bu mesajı alan düğüm *BOŞ* durumundaysa ebeveynini mesajın kaynağı düğüm olarak kaydedip, *SENK_AĞACI_KUR* mesajını komşularına gönderdikten sonra *YAPRAK* durumuna gönderdikten *YAPRAK* durumuna geçilir. Dikkat edilirse *SENK_AĞACI_KUR* mesajını gönderen düğüm hem kendi ebeveynine haber verirken hem de bu mesaj almayan komşularına ebeveyn olur. *YAPRAK* durumundayken *SENK_AĞACI_KUR* mesajını alan düğümün bir çocuğu olduğu için *KÖK* durumuna geçer ve çocuğunu kaydeder. Bir düğüm bütün komşularından *SENK_AĞACI_KUR* mesajını aldığında senkronizasyon ağacı kurulma işlemi bitmiş olur. Dikkat edilirse bu yaklaşım, bir düğümün senkronizasyon ağacı algoritmasının tamamlandığını tespit etmesini sağlayan bir α senkronizeridir.

ASYNSET algoritmasının tasarımı sırasında SSET algoritmasının sonlu durum makinesi üzerinde değişiklikler yaptık. Bu değişiklikleri yaparken düğümlerin bir raunt başlarken güncel ağırlık oranlarını bilmesini sağlamaya dikkat ettik. Çünkü bir düğüm işlemlerini bitirmeden raundu bitirirse bir sonraki rauntta algoritmanın çalışması yanlış olacaktır. Şekil 4.9'da ASYNSET algoritmasının sonlu durum makinesi verilmiştir. Bu algoritmanın sonlu durum makinesinde SSET algoritmasının sonlu durum makinesine göre yapılan değişimler yana yatık ve kırmızı yazı tipleriyle

ve kalın çizgiler ile belirtilmiştir. ASYNSET algoritmasında gönderilecek mesajlara ait alanlar SSET ile aynıdır, bundan dolayı tekrar belirtilmemiştir. Değişikliklerin listesi aşağıda verilmiştir.

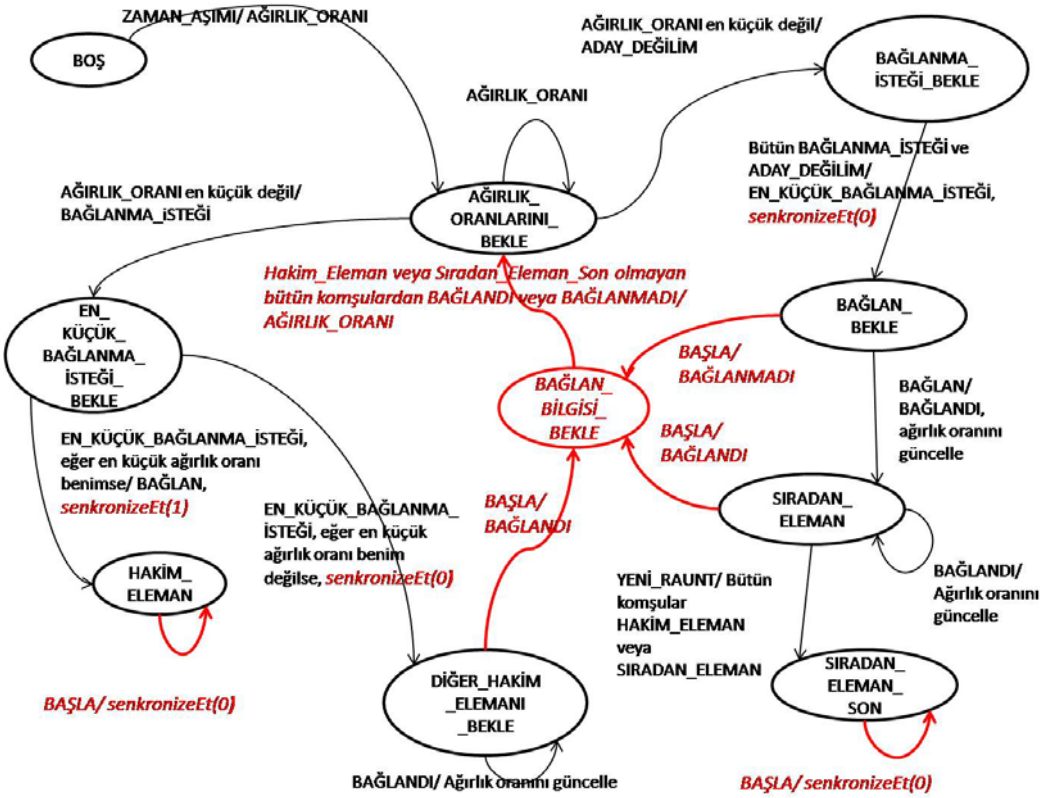
- *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumundan *HÂKİM_ELEMAN* durumuna geçerken düğüm *senkronizeEt(1)* prosedürünü çağırıp β senkronizasyon ağacı üzerinden senkronizasyon işlemini yapar. *SenkronizeEt* prosedüründeki 1 parametresiyle bu raunt içinde yeni bir *HÂKİM_ELEMAN* seçildiğini belirtip, bir sonraki rauntta aynı işlemlerin devam edeceğidir.

- *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumundan *DİĞER_HÂKİM_ELEMANI_BEKLE* durumuna geçerken *senkronizeEt*(0) prosedürüyle senkronizasyon işlemini yapılır. Bu eylemin amacı raundun bittiğini senkronizasyon ağacı üzerinden bildirmektir.

- *HÂKİM_ELEMAN* düğümü *BAŞLA* mesajını aldığında *senkronizeEt*(0) prosedürüyle senkronizasyon işlemini yapar. *senkronizeEt* prosedüründeki 0 parametresinin amacı *HÂKİM_ELEMAN*'ın bu raunt seçilmediğini göstermektir.

- *BAĞLANMA_İSTEĞİ_BEKLE* durumundan *BAĞLAN_BEKLE* durumuna geçerken *senkronizeEt*(0) prosedürüyle senkronizasyon işlemini yapılır. Bu eylemi yapan düğüm bulunduğu raunt sırasında kendi işlemini bitirmiş olabilir, fakat yeni raunt başlamadan önce komşularından herhangi biri *HÂKİM_ELEMAN* olursa ondan gelecek *BAĞLAN* mesajını alır. Bunun sebebi *HÂKİM_ELEMAN* düğümlerinin *BAĞLAN* mesajını senkronizasyon işleminden önce göndermesidir.

- *DİĞER_HÂKİM_ELEMANI_BEKLE* durumundaki düğüm *BAŞLA* mesajını alıp yeni raunda başladığında komşularına *BAĞLANMADI* mesajı gönderiyor ve *BAĞLAN_BİLGİSİ_BEKLE* durumuna geçip *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN_SON* durumundaki komşularının hepsinden *BAĞLANDI* ve *BAĞLANMADI* mesajı bekliyor. Bu işlem ağırlık oranını güncel tutmak için yapılır ve α senkronizerine örnektir.



Şekil 4.9 ASYNSET Algoritması sonlu durum makinesi

- *BAĞLAN_BEKLE* durumundaki düğüm *BAŞLA* mesajını aldığında *BAĞLANMADI* mesajı gönderiyor, *BAĞLAN_BİLGİSİ_BEKLE* durumuna geçip ağırlık oranını güncel tutmak için α senkronizerini kullanıyor.

- *SIRADAN_ELEMAN* durumundaki düğüm *BAŞLA* mesajını aldığında *BAĞLANDI* gönderiyor ve *BAĞLAN_BİLGİSİ_BEKLE* durumuna geçip α senkronizerini kullanıyor.

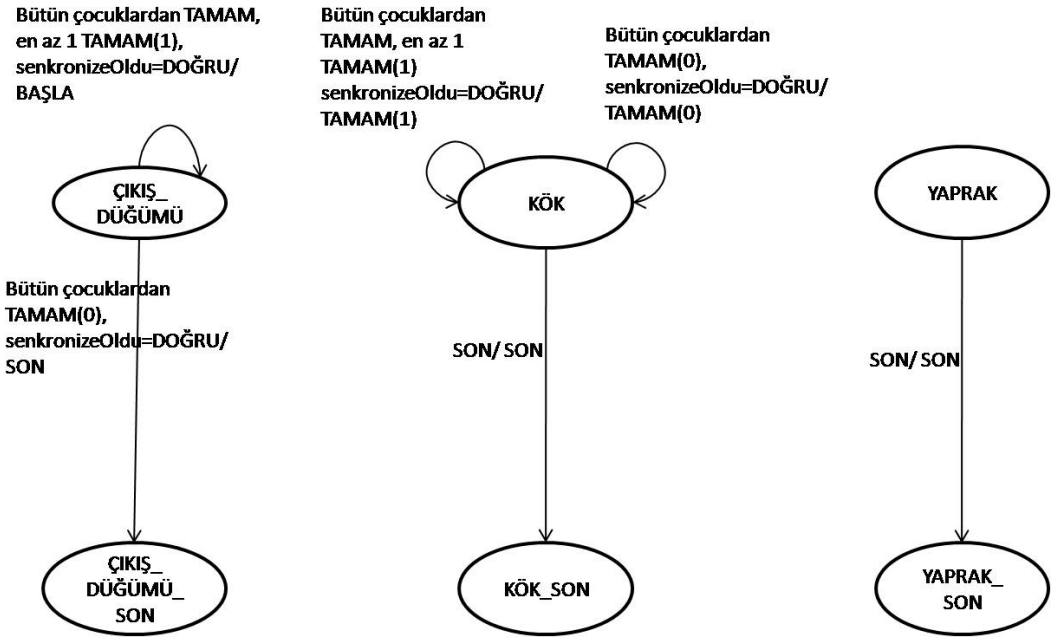
- *BAĞLAN_BİLGİSİ_BEKLE* durumunda olan düğüm *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN_SON* durumunda olan komşularından *BAĞLANDI* veya *BAĞLANMADI* toplayarak α senkronizerini kullanıyor, yeni ağırlık oranını hesaplayarak *AĞIRLIK_ORANI* mesajını gönderiyor.

- *SIRADAN_ELEMAN_SON* düğümü *BAŞLA* mesajını aldığında *senkronizeEt*(0) prosedürüyle senkronizasyon işlemini yapar. *senkronizeEt* prosedüründeki 0 parametresinin amacı *SIRADAN_ELEMAN_SON* durumundaki düğümün bu raunt içinde seçilen yeni bir *HÂKİM_ELEMAN* olmadığını göstermektir.

senkronizeEt prosedürü(durum) Eğer senkronizasyon ağacında *YAPRAK*'sam

```
   Eğer durumum 1 ise (Yeni seçilen HÂKİM_ELEMAN olduğumu gösterir)
        TAMAM(1) gönder.
   Değilse ebeveynime TAMAM(0) gönder
   Koşulu bitir
Değilse
   Eğer bütün çocuklarımdan TAMAM aldıysam
      Eğer bu mesajların hepsi TAMAM(0) ise TAMAM(0) gönder
      Değilse TAMAM(1) gönder
      Koşulu bitir
  Değilse senkronizeOldu ← DOĞRU
  Koşulu bitir
Koşulu bitir
```

Şekil 4.10 ASYNSET algoritması *senkronizeEt* prosedürü



Şekil 4.11 ASYNSET algoritmasının senkronizasyonu için kullanılan sonlu durum makinesi

ASYNSET algoritmasının sonlu durum makinesine eklenen ve yeni raunda geçmeyi sağlayan *senkronizeEt* prosedürü Şekil 4.10'da verilmiştir. Eğer *senkronizeEt* prosedürünü çağıran düğüm senkronizasyon ağacı üzerinde *YAPRAK* durumundaysa ve aynı raunt içinde *HÂKİM_ELEMAN* olarak seçildiyse *TAMAM*(1) mesajını senkronizasyon ağacı üzerinden ebeveynine gönderir ve böylece bir sonraki rauntta aynı işlemlerin devam etmesini sağlar. *YAPRAK* durumundaki düğüm aynı raunt içinde *HÂKİM_ELEMAN* olarak seçilmediyse *TAMAM*(0) mesajını ebeveynine gönderir. Eğer *senkronizeEt* prosedürünü çağıran düğüm senkronizasyon ağacı üzerinde *YAPRAK* durumunda değilse, bütün çocuklarından *TAMAM* mesajı almayı bekler. Bütün çocuklarından *TAMAM* almışsa ve mesajlardan herhangi biri *TAMAM*(1)'se ebeveynine *TAMAM*(1) mesajını iletir, eğer hepsi *TAMAM*(0) 'se ebeveynine *TAMAM*(0) mesajını gönderir. Eğer düğüm bütün çocuklarından *TAMAM*

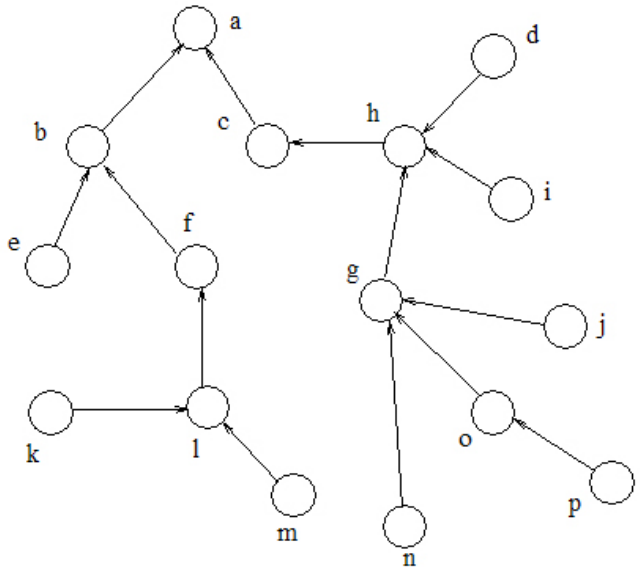
mesajı almamışsa senkronizasyon ağacı üzerinden senkronize olamaz. Bundan dolayı sadece *senkronizeOldu* isimli değişken *DOĞRU* olarak değiştirir ve çocuklarının *TAMAM* mesajı göndermesi bekler.

ASYNSET algoritmasında her düğüm kendi senkronizasyon işlemini bitirdikten sonra *TAMAM* mesajı gönderir. Bu mesajların işlenmesi de ayrıca bir sonlu durum makinesi tasarladık. Şekil 4.11'de bu sonlu durum makinesi verilmiştir. Şekil 4.11 (a)'da *ÇIKIŞ_DÜĞÜMÜ'*nün sonlu durum makinesi, Şekil 4.11 (b)'de *KÖK* düğümlerinin sonlu durum makinesi, Şekil 4.11 (c)'de *YAPRAK* düğümlerinin sonlu durum makinesi verilmiştir. *SenkronizeEt* prosedürüyle birlikte Şekil 4.8'de verilen sonlu durum makineleri ortak olarak *senkronizeOldu* değişkenini kullanırlar. Şekil 4.8 (a)'da *ÇIKIŞ_DÜĞÜMÜ* tüm çocuklarından *TAMAM* mesajı alırsa, *senkronizeOldu* değişkeni *DOĞRU* değerine sahipse ve çocuklarının herhangi birinden *TAMAM*(1) mesajı alırsa *BAŞLA* mesajı gönderip yeni raundu başlatır. Dikkat edilirse *BAŞLA* mesajının işlenmesi Şekil 4.11'deki ASYNSET'in sonlu durum makinesinde gösterilmiştir. Eğer çocuklarının hepsinden *TAMAM*(0) mesajı gelirse *ÇIKIŞ_DÜĞÜMÜ* algoritmayı bitirmek üzere *SON* mesajı gönderir. *KÖK* düğümleri için sonlu durum makinesi Şekil 4.11 (b)'de verilmiştir. Her *KÖK* düğümü de aynen *ÇIKIŞ_DÜĞÜMÜ* gibi çocuklarından *TAMAM* mesajı bekler, *TAMAM* mesajlarını topladıktan sonra bunları ebeveynine *TAMAM* mesajını iletir. *SON* mesajını alan *KÖK* düğümü bu mesajını komşularına sadece bir kez ileterek *KÖK_SON* durumuna geçer. *YAPRAK* düğümlerinin çocuğu olmadığı için *senkronizeEt* prosedürünü çalıştırmaları senkronize olmaları için yeterli olacaktır. Bir *YAPRAK* düğümü *SON* mesajı aldıktan sonra *SON* mesajını komşularına iletip *YAPRAK_SON* durumuna geçer.

4.1.4.3. Örnek Uygulama

Bu bölümde Şekil 4.11'daki ağ üzerinde ASYNSET algoritmasının örnek uygulamasını yapacağız. Algoritmanın ilk raundunda düğümler aynen SSET'de olduğu gibi iki zıplama uzaklıktaki komşularının ağırlıklarını öğrenirler ve sonlu durum makinesi üzerinde aynı durumlar üzerinde geçiş yaparlar. Algoritmanın buraya kadar işleyişi Bölüm 4.1.3.3'te verilen SSET örnek uygulamasıyla neredeyse aynıdır. ASYNSET algoritmasında b ve c düğümleri komşuları içinde düğüm a'yı en küçük ağırlıklı düğüm olarak bulup bu bilgiyi *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajı içinde göndererek, β senkronizasyon ağacı üzerinden senkronize olduktan sonra *BAĞLAN_BEKLE* durumuna geçerler. Bu ağ için kullanılan β senkronizasyon ağacı Şekil 4.12'de gösterilmiştir. Bu ağaç üzerindeki *ÇIKIŞ_DÜĞÜMÜ* düğüm a'dır. A düğümünün çocukları b ve c düğümleridir. B düğümünün ebeveyni a düğümü çocukları e ve f düğümleridir. Düğümler arasındaki diğer ilişkiler Şekil 4.12'deki ağaç üzerinde görülebilir. A düğümü b ve c düğümlerinden

EN_KÜÇÜK_BAĞLANMA_İSTEĞİ mesajını aldıktan sonra komşularına *BAĞLAN* mesajı gönderip, *HÂKİM_ELEMAN* durumuna geçtikten sonra senkronize olur. B ve c düğümü *BAĞLAN* mesajını aldıktan sonra daha önce senkronize olmalarına rağmen *BAĞLAN_BEKLE* durumundan *SIRADAN_ELEMAN* durumuna geçiş yaparlar. Benzer şekilde düğüm l'nin ağırlığı 0,11 olup bütün komşularından küçük olduğu için düğüm l *HÂKİM_ELEMAN* durumuna geçer, komşuları *SIRADAN_ELEMAN* olur ve senkronize olurlar. H düğümü *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajını aldığında düğüm A'nın kendinden daha az ağırlığa sahip olduğunu fark ettiği için *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumundan senkronize olarak *DİĞER_HÂKİM_ELEMANI_BEKLE* durumuna geçer. Bu düğümün c düğümün dışındaki komşuları senkronize olarak *BAĞLAN_BEKLE* durumunda kalır. Raunt 1'in sonunda a düğümüne b ve c düğümünden gelen *TAMAM* mesajları ulaşır. Bu raunt içinde *HÂKİM_ELEMAN* seçimi olduğu için *TAMAM* mesajı *TAMAM*(1) olarak a düğümüne ulaşır, böylelikle bir sonraki raunda başlanabilir.



Şekil 4.12 ASYNSET algoritması senkronizasyon ağacı

2. raunt başladığında a düğümü tarafından *BAŞLA* gönderilerek başlatıldığında düğümler *BAĞLAN_BEKLE*, *SIRADAN_ELEMAN*, *SIRADAN_ELEMAN_SON*, *DİĞER_HÂKİM_ELEMANI_BEKLE* veya *HÂKİM_ELEMAN* durumlarında olabilirler. A düğümü ve l düğümü *HÂKİM_ELEMAN* durumunda oldukları ve bu rauntta yeni bir işlem yapmayacakları için *senkronizeEt* prosedürünü çağırıp senkronize olurlar. B düğümü, c düğümü, f düğümü, k düğümü ve m düğümü *SIRADAN_ELEMAN* durumunda olduklarından bir *HÂKİM_ELEMAN* tarafından örtüldüklerinden dolayı *BAĞLANDI* mesajı gönderip *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçerler. H düğümü ve *DİĞER_HÂKİM_ELEMANI_BEKLE* durumunda olan diğer düğümler herhangi bir *HÂKİM_ELEMAN* tarafından örtülmedikleri için *BAĞLANMADI* mesajı gönderir ve *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçerler. Ayrıca düğüm d ve

BAĞLAN_BEKLE durumundaki diğer düğümler herhangi bir *HÂKİM_ELEMAN* tarafından örtülmedikleri için *BAĞLANMADI* mesajını gönderir ve *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçerler. *BAĞLANDI* ve *BAĞLANMADI* mesajlarının hepsinin gönderimi bittikten sonra düğümler yeni rauntta ağırlık oranlarını hesaplayabilirler. Şekil 4.6'da düğümlerin üzerinde "2:" ile 2.raunttaki ağırlık oranları verilmiştir. Bu rauntta birinci raunda çok benzer şekilde düğüm j ve düğüm m *HÂKİM_ELEMAN* olarak seçilip komşuları olan düğüm g, düğüm o, düğüm n *SIRADAN_ELEMAN* olur. Bu raunt içinde *HÂKİM_ELEMAN* seçildiği için a düğümü senkronizasyon sırasında *TAMAM*(1) mesajını alır. Raunt 3'de raunt 2'ye çok benzer şekilde düğüm h ve düğüm k *HÂKİM_ELEMAN* olarak seçilir. Raunt 4'de sadece o düğümü *HÂKİM_ELEMAN* olur, bu raundun sonunda senkronizasyon yapılırken a düğümüne ulaşan bütün mesajlar *TAMAM*(0) tipindedir. Bundan dolayı ASYNSET algoritması *SON* mesajıyla sonlandırılır.

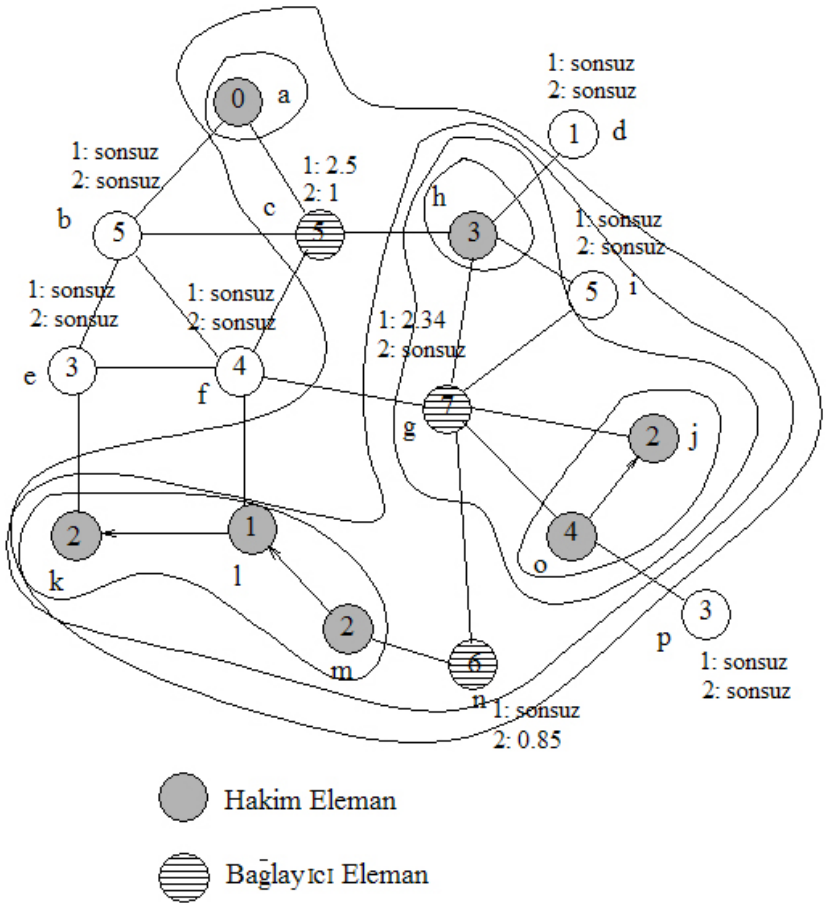
4.2. Ağırlıklı Steiner Ağacı Algoritmaları

Bu bölümde Ağırlıklı Steiner Ağacı algoritmaları açıklanacaktır. Bu algoritmaların genel fikirleri, tanımları ve örnek uygulamaları verilmiştir.

4.2.1. Genel Fikir

Bu bölümde önerdiğimiz algoritmalar Guha'nın ikinci basamakta kullandığı Klein ve Ravi'nin algoritmasının dağıtık sürümleridir. Bu algoritmanın yaklaşım oranı S en küçük bağlayıcıları ağırlığı olmak üzere 2 ln (*S*)'dir. Bu algoritmanın basamakları Şekil 2.8'de verilmiştir. Bu algoritmayı CENTTREE (Central Steiner tree algorithm) olarak isimlendirelim.

Şekil 4.13'de CENTTREE'nin örnek bir uygulaması verilmiştir. Düğümlerin maliyeti içlerine yazılmıştır. Düğümlerin ilk iterasyondaki ağırlık oranları düğümlerin üzerinde "1:" ile ikinci iterasyondaki ağırlık oranları "2:" ile yazılmıştır. İlk iterasyonda g düğümünün ağırlık oranı (g düğümünün maliyeti) / (birleştireceği düğüm sayısı) olarak hesaplanıp 7/3=2.34 bulunur. Benzer bir hesapla c düğümünün ağırlık oranı 2.5 bulunur. Bu iterasyon içinde en küçük ağırlıklı düğüm g olduğu için bağlayıcı olarak seçildikten sonra düğüm h, düğüm g, düğüm o ve düğüm j aynı ağacın içine alınır. Kalan iterasyonlarda sırasıyla düğüm c ve düğüm n seçilir, böylelikle bütün hâkim elemanlar aynı ağacın içine girer.



Şekil 4.13 CENTTREE'nin örnek uygulaması

4.2.2. Yayma Tabanlı Senkron Ağırlıklı Steiner Ağacı Algoritması

Bu bölümde yayma tabanlı senkron ağırlıklı Steiner ağacı algoritmasının genel fikri, tanımlaması ve örnek uygulaması verilecektir.

4.2.2.1. Genel Fikir

Yayma tabanlı senkron ağırlıklı Steiner ağacı algoritması (Flooding based synchronous Steiner tree algorithm (FLOODTREE)) CENTTREE algoritmasının çalışmasına uygun ve dağıtık olarak tasarlanmasıyla yapılmıştır. FLOODTREE algoritması senkron rauntlara bölünmüştür. İlk rauntta hâkim düğümler kendi aralarında ağaç oluştururlar. 2. rauntta aday düğümler birbirlerine ağırlık oranını yayarak en küçük ağırlıklı düğümü bulur. Bir sonraki rauntta ağaçlar bağlanır ve ikinci raunda dönülür. Algoritma bütün ağaçlar bağlanana kadar bu şekilde tekrar eder.

4.2.2.2. Tanım

FLOODTREE algoritmasının basamakları Şekil 4.14'te verilmiştir. Bu algoritmanın birinci raundunda birbirleri arasında patika olan düğümler aynı ağacın içine koymak için hâkim küme içindeki düğümler bilgilerini *HÂKİM_ELEMAN(düğüm_kimliği)* mesajı içinde komşularına gönderir. Bu mesajı alan düğüm bir hâkim elemansa mesajın kaynağını kaydeder ve mesajı iletir. Bu raundun sonunda ağaçlar oluşur, hâkim düğümler hangi ağacın içine dâhil olduğunu ve ağacın içinde kaç eleman olduğunu bilir.

Birinci Rauntta:

1. Her hâkim küme düğümü komşularına *HÂKİM_ELEMAN* mesajını göndersin.
2. *HÂKİM_ELEMAN* mesajını komşusundan alan hâkim eleman bu mesajın kaynağından gelen mesajı sadece bir kez olmak üzere komşularına iletsin. Her hâkim eleman kendinden gelen bütün hâkim eleman bilgilerini kullanarak ağaçtaki toplam düğüm sayısını bulsun. Ağaçtaki en küçük kimlikli düğüm ağacın kimliği olarak seçilsin. Raundun sonunda böylece birbirleri arasında patika olan hâkim elemanlar aynı ağacın içine konur.

İkinci Rauntta:

1. Hâkim Küme içerisindeki düğümler hangi ağaca bağlı olduklarını ve ağaçtaki düğüm sayısını *AĞAÇ_BİLGİSİ* mesajıyla komşularına gönderir.
2. Raundun sonunda hâkim küme içinde olmayan düğümler ağırlık oranını hesaplarlar.

Birinci Raunt Dışındaki Tek Numaralı Rauntlarda:

1. Hâkim küme içinde olmayan düğümler ağırlık oranlarını *AĞIRLIK_ORANI* mesajıyla ağa yayarlar.
2. Raundun sonunda en küçük ağırlık oranına sahip olan düğüm hâkim küme içine girer. Eğer hâkim küme içinde olmayan bütün düğümlerin ağırlık oranları sonsuzsa ve algoritmanın başlangıcındaki hâkim küme elemanları bağlandıysa, bütün düğümler algoritmayı sonlandırır. Eğer algoritmanın başlangıcındaki bütün düğümler bağlanmadıysa adaylar içinden kendisinin bağlı olmadığı başka ağaca bağlı olan bir aday komşusu olan en yüksek enerjili düğümü seç.

İkinci Raunt Dışındaki Çift Numaralı Rauntlarda:

1. Hâkim Küme içerisine yeni giren düğüm *AĞAÇ_BİLGİSİ* mesajını komşularına gönderir. Bu mesajı alan hâkim küme içinde olan düğümler ağaç bilgilerini değiştirip mesajı komşularına iletir. Bu mesajı alan hâkim küme içinde olmayan düğümler bağlı oldukları ağaç bilgilerini değiştirirler.
2. Raundun sonunda hâkim küme içinde olmayan düğümler ağırlık oranını hesaplarlar.

Şekil 4.14 FLOODTREE algoritması basamakları

İkinci rauntta hâkim küme içindeki düğümler ağaç bilgileri *AĞAÇ_BİLGİSİ*(*düğüm_kimliği*, *ağaç_kimliği*, *ağaçtaki_düğüm_sayısı*) mesajıyla komşularına gönderir. Raundun sonunda hâkim küme içinde olmayan düğümler ağırlık oranını hesaplar.

Düğümler, 1 numaralı raunt dışında tek numaralı rauntlarda ağırlık oranlarını *AĞIRLIK_ORANI*(*ağırlık_oranı*) mesajıyla tüm ağa yayarlar. Raundun sonunda hâkim küme içinde olmayan düğümler ağırlık oranlarını hesaplar. Birinci raunt dışındaki tek numaralı rauntlarda ağırlık oranlarını hesaplayan düğümler *AĞIRLIK_ORANI* mesajını ağa yayarlar. Raundun sonunda aday bağlayıcı düğümler birbirlerine birbirlerinin ağırlıklarını öğrenmiş olur, en küçük ağırlık oranına sahip düğüm bağlayıcı olup hâkim küme içine girer. İkinci raunt dışındaki çift numaralı rauntlarda ağaçları bağlanır ve bu ağaçlara komşu olan aday düğümlerin ağırlıkları düzenlenir. Algoritma bu şekilde bütün hâkim elemanlar aynı ağacın içinde olana kadar devam eder.

4.2.2.3. Örnek Uygulama

FLOODTREE'nin örnek uygulaması Şekil 4.13 üzerinde yapılacaktır. Birinci düğüm k, düğüm l, düğüm m, düğüm o, düğüm j, düğüm g, düğüm n ve düğüm c *HÂKİM_ELEMAN* mesajını komşularına gönderir. Hâkim küme içerisindeki düğümler *HÂKİM_ELEMAN* mesajını birbirleri üzerinden yaydıktan sonra düğüm k, düğüm l ve düğüm m aynı ağacın içine, düğüm o ve düğüm j aynı ağacın içine girer, düğüm a ve düğüm h kendi başına bir ağaç oluşturur. Düğüm k, düğüm l ve düğüm m'nin oluşturduğu ağacın kimliği k, düğüm o ve düğüm j'nin oluşturduğu ağacın kimliği j olur. İkinci rauntta hâkim küme içerisindeki düğümler ağaç kimliklerini ve ağaçlarının içindeki eleman sayısını *AĞAÇ_BİLGİSİ* mesajıyla komşularına gönderir. Düğüm g, düğüm o, düğüm j ve düğüm h'den ağaç bilgisi alır ve raundun sonunda 7/3=2.34 olarak ağırlık oranını hesaplar. Düğüm c, düğüm a ve düğüm h'den ağaç bilgisini alır ve raundun sonunda 5/2=2.5 olarak ağırlık oranını hesaplar. Üçüncü rauntta hesaplanan bu ağırlık oranları *AĞIRLIK_ORANI* mesajıyla ağa gönderilir. Raundun sonunda bütün düğümler birbirlerinin ağırlık oranlarını bilir. Düğüm g'nin ağırlık oranı en küçük olduğu için bu raunt sonunda hâkim küme içine girer. Dördüncü rauntta düğüm g komşularına *AĞAÇ_BİLGİSİ* mesajını gönderir, bu mesajı alan düğüm h, düğüm o ve düğüm j bu mesajı iletir ve kendi ağaç bilgilerini günceller. Bu ağaca bağlı olan düğüm c, düğüm d, düğüm i, düğüm n ve düğüm p ağırlık oranlarını günceller. Bir sonraki rauntta üçüncü raunda benzer şekilde ağırlık oranları tekrardan ağa yayılır ve düğüm n hâkim küme içine girer. Bu raunttan sonraki rauntta dördüncü raunda benzer olarak birleşen ağaçların bilgisi paylaşılır. Algoritmada son olarak düğüm c hâkim küme içine girer, bütün hâkim elemanlar aynı ağacın içine alınır ve algoritma 9. rauntta sonlanır.

4.2.3. Senkron Ağırlıklı Steiner Ağacı Algoritması

Bu bölümde senkron ağırlıklı Steiner ağacı Algoritmasının genel fikri, tanımı ve örnek uygulaması gösterilecektir.

4.2.3.1. Genel Fikir

Klein ve Ravi'nin algoritmasını dağıtık yaparken dikkat edilmesi gereken en önemli nokta düğüm ağırlıklarının her iterasyonda azabilmesidir. Bu durum her iterasyonda düğümlerin ağırlıklarının kesinlikle azalmadığı merkezi küme örtüsü tabanlı bağlı hâkim küme örtüsü algoritmasının tam tersidir ve algoritmanın dağıtık olarak tasarlanmasını zorlaştırmaktadır. SSET algoritmasında aynı anda birden fazla düğüm seçilebilirken, yeni tasarlamayı düşündüğümüz dağıtık senkron ağırlıklı Steiner ağacı algoritmalarında maalesef her iterasyonda tek bir düğüm bağlayıcı olarak seçilmelidir. Bu noktadan yola çıkarak tasarladığımız ilk dağıtık algoritma FLOODSET'e benzer olan FLOODTREE algoritmasıdır. Analiz bölümünde de bahsedileceği üzere FLOODTREE algoritmasının mesaj karmaşıklığı üst limitte $O(N^3)$'e eşittir. FLOODTREE algoritmasının mesaj karmaşıklığının asimptotik olarak $O(N^2)$'e düşürebilmek için STREE (Synchronous weighted Steiner tree algorithm) algoritmasını öneriyoruz. STREE algoritmasında iki adet farklı ağaç yapısı kullanarak mesaj karmaşıklığını düşürmeye çalıştık. Bu fikirler şunlardır:

- Birbirleri arasında patika olan hâkim küme elemanlarının oluşturduğu ağacın bir kökü olur. Ağaç bu köke doğru yönlendirilmiş olarak oluşturulur. Bu ağaç üzerinde her düğüm ebeveynini ve çocuklarını bilir. Böylece ağaca komşularından gelen bağlanma istekleri düğüm başına 1 mesaj ile kök düğüme iletilir.

- *ÇIKIŞ_DÜĞÜMÜ* kök olacak şekilde ağ üzerinde yönlendirilmiş bir ağaç kurulur. Bu ağaç üzerinde her düğüm ebeveynini ve çocuklarını bilir. Bu ağaç sayesinde düğümler sadece 1 mesaj göndererek, aday düğümler ağırlık oranlarını *ÇIKIŞ_DÜĞÜMÜ*'ne iletir.

4.2.3.2. Tanım

STREE algoritmasının basamakları Şekil 4.15'de verilmiştir. STREE algoritmasının ilk raundunda hâkim elemanlar kendi aralarında ağaç oluştururlar. Bu basamak FLOODTREE çok benzerdir. FLOODTREE'de ağaç yönlendirilmiş olarak oluşturulmaz, düğümler sadece ağaç bilgilerini öğrenirler. STREE algoritmasında ağaç yönlendirilmiş olarak oluşturulur. Bu basamağı yapmak için hâkim küme içindeki bir m düğümü kendi bilgisini *AĞAÇ_KUR(düğüm_kimliği)* mesajıyla komşularına gönderir. Bu mesajı alan hâkim küme içindeki bir k düğümü

AĞAÇ_KUR mesajını komşularına iletir ve bu iletme süreci mesajı alan düğümler hâkim küme içinde olduğu sürece devam eder. *AĞAÇ_KUR* mesajını alan ve hâkim küme içinde olmayan bir düğüm bu mesajı iletmez. Bu raundun sonunda hâkim küme içindeki bir m düğümü, hâkim küme içinde olup *AĞAÇ_KUR* mesajını aldığı düğümlerle birlikte merkezi bir en küçük kapsayan ağaç algoritması çalıştırıp aynı ağaç içine dâhil olur. Bu ağacın bir kökü vardır ve ağaç köke doğru yönlendirilmiş olarak kurulur. Ağacın kökü ağacın lideri olup *LDR_HELMAN* durumundayken ağaç üzerindeki diğer düğümler *HELMAN* durumundadır. Bu raundun mesaj karmaşıklığı Analiz bölümünde de gösterileceği üzere üst limitte $O(N^2)$'ye eşittir ve hâkim küme elemanları ağaçlarını oluşturmaları için sadece bir kez çalıştırılması yeterlidir.

Raunt 1. *HELMAN* düğümler arasında patika olanları aynı ağaç içine koy. Her ağacı en küçük kimlikli düğüm kök düğüm olacak şekilde yönlendir. Her ağacın kimliği kök düğümün kimliğine eşit olsun.

Raunt 2. *ÇIKIŞ_DÜĞÜMÜ* kök düğüm olduğu ağacı oluştur.

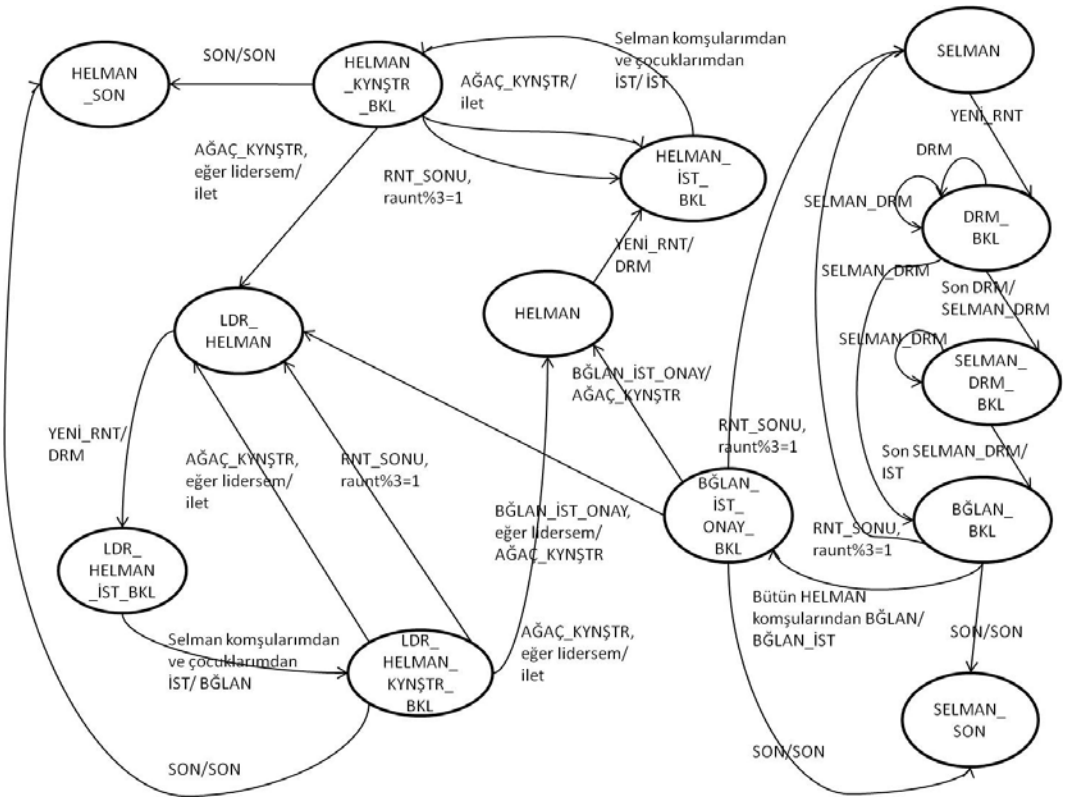
Raunt 3 (Bağlayıcı Seçen İlk Raunt). Şekil 4.6'da gösterilen sonlu durum makinesindeki algoritmayı kullanarak aday bağlayıcı düğümler bütün bağlı oldukları ağaçlardan *BAĞLN* mesajı alırlar.
Raunt 4 (Bağlayıcı Seçen İkinci Raunt). Şekil 4.7'de gösterilen sonlu durum makinesindeki algoritmayı kullanarak aday bağlayıcı düğümlerin ağırlık oranını *ÇIKIŞ_DÜĞÜMÜ*'ne doğru oluşturulmuş ağaç üzerinden gönder. *ÇIKIŞ_DÜĞÜMÜ* raunt sonunda en küçük ağırlıklı düğümü bulacaktır.
Raunt 5 (Bağlayıcı Seçen Üçüncü Raunt). Şekil 4.8'de gösterilen sonlu durum makinesindeki *ÇIKIŞ_DÜĞÜMÜ*'ne *BAĞLN_İST_ONAY* mesajı gönderir ve düğüm bağlayıcı olarak hâkim küme içine girer, Raunt 3'e dönülür. Eğer bağlayıcı düğüm yoksa *ÇIKIŞ_DÜĞÜMÜ SON* mesajı gönderir ve algoritma biter.

Şekil 4.15 STREE algoritması basamakları

STREE algoritmasının ikinci raundunda *ÇIKIŞ_DÜĞÜMÜ*'nün kök olduğu ağaç oluşturulur. Bu ağaç ASYNSET'de verilen Şekil 4.8'deki gibi oluşturulabilir. Bu ağacın amacı bağlayıcı düğümün seçileceği diğer rauntlarda herkesin birbirine mesaj gönderdiği ve mesaj karmaşıklığının $O(N^3)$'e çıktığı FLOODTREE'deki durumu engellemektir. Bu ağaç ASYNSET algoritmasında olduğu gibi β senkronizeri olarak kullanılmıştır. Bu sebeple bu ağaç bir senkronizasyon ağacı olsa da rauntları bu ağaç üzerinden başlatmadığımız için senkronizasyon ağacı olarak isimlendirmiyoruz.

Yukarda anlatılan ilk ve ikinci raunt sadece bir kez uygulanır. Bu rauntlar bittikten sonra bağlayıcı düğümlerin seçildiği üçer rauntluk işlemler başlar. STREE

algoritmasında bağlayıcı seçen ilk raundun sonlu durum makinesi Şekil 4.16'da verilmiştir. Sonlu durum makineleriyle ilgili kısaltmalar hatırlatmak ve sonlu durum makinesinin rahat okunması amacıyla Şekil 4.17'de tekrar verilmiştir.



Şekil 4.16 STREE algoritmasının bağlayıcı seçen ilk raundunun sonlu durum makinesi

HELMAN: HÂKİM_ELEMAN	*SELMAN: SIRADAN_ELEMAN*
BAĞLN: BAĞLAN	*DRM: DURUM*
İST: İSTEK	*RNT: RAUNT*
BKL:BEKLE	*LDR: LİDER*
KYNŞTR: KAYNAŞTIR	

Şekil 4.17 STREE algoritmasının sonlu durum makinelerindeki kısaltmalar

STREE de bağlayıcı seçen ilk raunt şu basamaklardan oluşur:

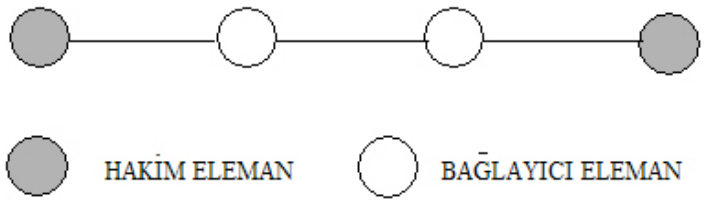
1. *HELMAN* durumundaki düğümler raundun başında ağaç kimliklerini *DRM(düğüm_kimliği,ağaç_kimliği, ağaç_boyutu)* mesajı içinde gönderirler ve *SELMAN* durumundaki düğümlerden *İST(düğüm_kimliği,gerçek_düğüm_kimliği, ağırlık_oranı)* mesajı bekledikleri *HELMAN_İST_BKL* durumuna geçerler. *İST* mesajını içindeki *gerçek_düğüm_kimliği*, *HELMAN* düğümler tarafından *İST* iletilirken kullanılır ve illeri basamaklarda bu işlem gösterilecektir.

2. *SELMAN* durumundaki düğümler raundun başında *DRM_BKL* durumuna geçip *HELMAN* düğümlerden *DRM* mesajı toplayan *SELMAN* düğümler *DRM* mesajlarını toplamayı bitirdikten sonra komşu *SELMAN* düğümlerine komşu oldukları ağaç kimliklerini *SELMAN_DRM(düğüm_kimliği,komşu_ağaç_kimliği)* mesajı içinde gönderirler. *SELMAN_DRM* mesajının gönderilme amacı Şekil

4.18'deki durumda bağlayıcı seçebilmektir. Şekil 4.18'de görüldüğü üzere eğer iki ağaç arasında ikiden fazla zıplama uzaklık varsa ve aralarındaki tek bağlantı noktası *SELMAN* düğümlerse bu düğümlerin bağlayıcı olarak seçilmesi gerekir. *SELMAN_DRM* mesajını toplamayı bitiren düğümler *BĞLAN_BKL* durumuna geçerler.

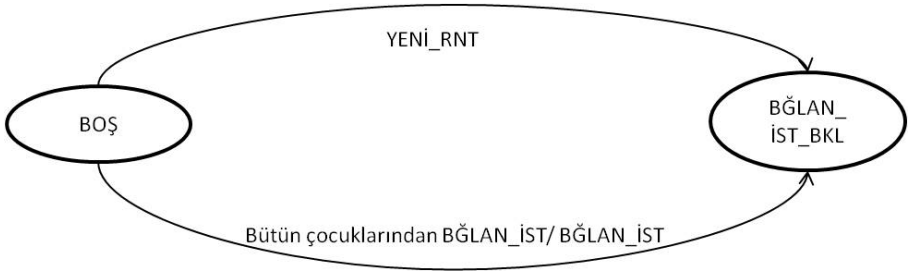
3. Bütün *SELMAN* komşularından *İST* mesajı alan *HELMAN* düğümler ağaç üzerindeki ebeveynlerine bu mesajı iletiyorlar. Ağacın kökündeki bu düğüm ağacın lideri olarak *LDR_HELMAN_İST_BKL* durumundadır ve kendisine gelen en küçük *İST* kaynağına *BĞLAN(ağaç_kimliği,hedef_düğüm_kimliği)* mesajı gönderir.

4. *BĞLAN_BKL* durumundaki *SELMAN* düğümler bağlı oldukları tüm ağaçlardan *BĞLAN* mesajı alırsa bir sonraki raunt için ağırlık oranını (düğümün maliyeti) / (birleştireceği ağaçlar üzerindeki düğüm sayısı) olarak hesaplar. Kenar ağırlıkları hesaba katılmamış ve eşit olarak kabul edilmiştir.

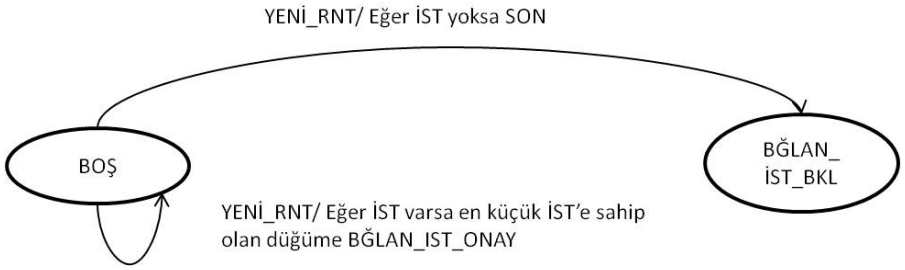


Şekil 4.18 STREE algoritmasında ağaçların birden fazla bağlayıcıyla bağlanması gereken durum

Bağlayıcı seçen ikinci rauntta düğümler *BĞLAN_İST (düğüm_kimliği,gerçek_düğüm_kimliği, ağırlık_oranı)* mesajlarını *ÇIKIŞ_DÜĞÜMÜ*'ne doğru yönlendirilmiş ağaç üzerinden gönderirler. Bu raundun sonlu durum makinesi Şekil 4.19'da verilmiştir. Ağırlık oranı olmayan düğümler, ağırlık oranlarını ∞ yapıp, *BĞLAN_İST* mesajı içinde gönderirler. Şekil 4.19'da görülen durumdaki düğümler de *BĞLAN_İST* mesajının içinde bir "*boolean*" alan ile durumlarını bildirirler. Çocuklarının hepsinden *BĞLAN_İST* mesajı toplayan bir düğüm kendi ağırlık oranıyla birlikte hepsinin içinden en küçük olanını bulup *BĞLAN_İST* mesajıyla ebeveynine gönderir. Bu raundun sonunda *ÇIKIŞ_DÜĞÜMÜ'*nde en küçük ağırlık bulunmuş olur.



Şekil 4.19 STREE algoritmasının bağlayıcı seçen ikinci raundunun sonlu durum makinesi



Şekil 4.20 STREE algoritmasının bağlayıcı seçen üçüncü raundunun sonlu durum makinesi

Bağlayıcının seçiminin tamamlandığı üçüncü rauntta *ÇIKIŞ_DÜĞÜMÜ* kendine ulaşan ağırlık oranları içinden en küçüğünü bulur ve bu ağırlık oranına sahip düğüme *BĞLAN_İST_ONAY(düğüm_kimliği)* mesajı gönderir. *ÇIKIŞD_DÜĞÜMÜ*'ne doğru *BĞLAN_İST* mesajları giderken en küçük ağırlık oranına sahip olan düğüme daha sonra ulaşmak için geriye doğru yollar tutulur. *ÇIKIŞ_DÜĞÜMÜ BĞLAN_İST_ONAY* mesajını bu yoldan gönderir. Bu raundun sonlu durum makinesi Şekil 4.20'de verilmiştir. Ayrıca Şekil 4.16'daki sonlu durum makinesi de bu raunt içinde aktif olur. *BĞLAN_İST_ONAY* mesajını alan *SELMAN* düğüm Şekil 4.16'da gösterilen sonlu durum makinesiyle de gösterildiği üzere *BĞLAN_İST_ONAY_BKL* durumunda eğer yeni oluşturacağı ağaç içinde en küçük kimliğe sahipse *LDR_HELMAN* durumuna, değilse *HELMAN* durumuna geçer. Düğüm yönlendirilmiş ağacı merkezi olarak oluşturduktan sonra *AĞAÇ_KYNŞTR(düğüm_kimliği,ağaç_kimliği)* mesajıyla birleştireceği *LDR_HELMAN_KYNŞTR_BKL* ve *HELMAN_KYNŞTR_BKL* durumundaki düğümlere gönderir. *AĞAÇ_KYNŞTR* mesajını alan bu düğüm *HELMAN* veya *LDR_HELMAN* durumuna geçerler. Böylelikle yeni ağaç bağlayıcı üzerinden kurulmuş olur.

4.2.3.3. Örnek Uygulama

Bu bölümde STREE algoritmasının bir örnek uygulaması verilecektir. Bu örnek uygulama Şekil 4.13 üzerinde yapılacaktır. İçi gri ile dolu düğümler ASYNSET algoritması tamamlandığında bulunan hâkim elemanlardır. STREE algoritmasının ilk raundundan bu düğümler komşularına *AĞAÇ_KUR* mesajını gönderir. Birbirleri arasında patika olan *HELMAN* düğümler bu mesajları iletirler. Bunun sonucunda Şekil 4.13 üzerinde gösterilen yönlendirilmiş ağaçlar oluşur. Bu şekil üzerinde toplam 4 ağaç gösterilmiştir. Bu ağaçların aşağıda listelenmiştir:

- Düğüm k, düğüm l ve düğüm m tarafından düğüm k'ye doğru yönlendirilmiş ağaç.

- Düğüm o ve düğüm j tarafından oluşturulan düğüm j'ye doğru yönlendirilmiş ağaç.

- Düğüm a ve düğüm h'nin kendi başına oluşturduğu ağaçlar.

STREE algoritmasının ikinci raundunda ağdaki bütün düğümler arasında kök düğüme doğru yönlendirilmiş ağaç oluşturulur. Bu ağaç Şekil 4.12'de verilen ASYNSET'e ait olan ağaç gibi oluşturabilir. STREE algoritmasının bir sonraki raundunda ilk bağlayıcı geçilir. *SELMAN* durumunda olan düğüm g, *YENİ_RNT* kesmesinden sonra *DRM_BKL* durumuna geçer ve *HELMAN* düğümlerden *DRM* mesajı bekler. Düğüm h, düğüm o ve düğüm j *HELMAN* durumundayken *YENİ_RNT* kesmesinden sonra ağaçlarının kimliklerini ve ağaçlarının içindeki düğüm sayısını *DRM* mesajı içinde gönderir. Düğüm g'nin komşu olduğu *HELMAN* düğümler düğüm h, düğüm o ve düğüm j olduğu için *HELMAN* komşularının hepsinden *DRM* mesajını toplamış olur ve *SELMAN* komşularını hedef alacak şekilde *SELMAN_DRM* mesajının içinde komşu ağaç kimliklerini gönderir. Düğüm g, *SELMAN* komşuları olan düğüm i, düğüm f ve düğüm n'den *SELMAN_DRM* mesajlarını topladıktan sonra ağırlık oranını hesaplayıp *İST* mesajını *HELMAN* komşularına gönderir. Düğüm g'nin ağırlık oranı (maliyeti) / (birleştireceği *HELMAN* sayısı) formülünden 7 / (2+1) olarak 7/3=2,34 hesaplanır. Benzer şekilde düğüm c ağırlık oranını 5/2'den 2,5 hesaplar. Düğümlerin raunt 1 ve raunt 2'deki ağırlık oranları üstlerine "1:" ve "2:" olarak yazılmıştır. Düğüm g *İST* mesajını gönderdikten sonra *BĞLAN_BKL* durumuna geçer. Benzer şekilde düğüm c *İST* mesajını gönderdikten sonra *BĞLAN_BKL* durumuna geçer. *HELMAN_İST_BKL* durumundaki düğüm o *İST* mesajını düğüm g'den aldıktan sonra başka *SELMAN* komşusu veya *HELMAN* çocuğu olmadığı için *İST* mesajını düğüm j'ye yönlendirir ve *HELMAN_KYNŞTR_BKL* durumuna geçer. Düğüm j, düğüm g'den ve düğüm o'dan *İST* mesajlarını aldıktan sonra başka alacağı *İST* mesajı kalmadığı için en küçük ağırlıklı *İST* mesajı olarak düğüm g'yi bulur ve *BĞLAN* mesajını düğüm o'ya doğru düğüm g'nin de alacağı şekilde gönderir, *LDR_HELMAN_KYNŞTR_BKL* durumuna geçer. Böylelikle düğüm g, düğüm o'nun ağacından *BĞLAN* mesajını almış olur. Düğüm h, düğüm c'den 2.5 ağırlıklı *İST* mesajı, düğüm g'den 2.34 ağırlıklı *İST* mesajını topladıktan sonra *LDR_HELMAN_İST_BKL* durumundan *LDR_HELMAN_KYNŞTR_BKL* durumuna geçerek, düğüm g'yi hedefleyen *BĞLAN* mesajını gönderir, bu mesajı alan düğüm g, çevresindeki bütün *HELMAN* düğümlerden *BĞLAN* mesajını topladığı için *BĞLAN_İST_ONAY_BKL* durumuna geçer. Düğüm c çevresindeki bütün ağaçlardan *BĞLAN* mesajı almadığından dolayı *BĞLAN_BKL* durumunda kalır.

Bağlayıcı seçen ikinci rauntta düğümler ağaç üzerinden *BĞLAN_İST* mesajları gönderirler. Kullanılan ağaç Şekil 4.12'de ASYNSET için verilen ağaçtır. Düğüm g kendi çocukları olan düğüm n, düğüm o ve düğüm j'den *BĞLAN_İST* mesajlarını topladıktan sonra kendinin ağırlığı en küçük olduğu için ağaç üzerinden düğüm h'ye gönderir. Düğüm h çocuklarından *BĞLAN_İST* mesajlarını topladıktan sonra bu

mesajları ebeveyni olan düğüm c'ye daha sonra da *ÇIKIŞ_DÜĞÜMÜ* olan düğüm a'ya iletilir. Benzer şekilde diğer düğümler de buldukları ağırlık oranlarını *BĞLAN_İST* mesajı içinde *ÇIKIŞ_DÜĞÜMÜ'*ne doğru ağaç üzerinden gönderirler. *ÇIKIŞ_DÜĞÜMÜ'*nün bu mesajlar sonucunda bulduğu en küçük ağırlık 2.34 ve bu ağırlığın sahibi g düğümüdür.

Bağlayıcı seçen üçüncü rauntta *ÇIKIŞ_DÜĞÜMÜ* bulduğu en küçük ağırlık g düğümü olduğu için düğüm a→düğüm c→düğüm h→düğüm g patikasını takip eden *BĞLAN_İST_ONAY* mesajını gönderir. Düğüm g bu mesajı aldığında bağlayıcı seçen ilk raundun sonlu durum makinesi üzerinden işler ve *BĞLAN_İST_ONAY_BKL* durumundan *LDR_HELMAN* durumuna geçer ve bağlı olduğu ağaçlara *AĞAÇ_KYNŞTR* mesajını gönderir. Bu mesajı alan düğüm h, düğüm o ve düğüm j *HELMAN* durumuna geçip ağaç bilgilerini ve kimliklerini düzenler. Böylelikle bir bağlayıcı düğüm seçilmiş, ağaçlar yeniden düzenlenmiş ve gereken bilgiler güncelleştirilmiş olur. Bir sonraki rauntta bağlayıcı seçen ilk raunt tekrardan çalıştırılacak ve yeni
Bağlayıcı seçene kadar 3 raunt devam edecektir. Algoritma toplam 11 raunt sürer ve sonunda düğüm g, düğüm n ve düğüm c sırasıyla bağlayıcı olarak seçilir.

4.2.4. Yarı Asenkron Ağırlıklı Steiner Ağacı Algoritması

Bu bölümde yarı asenkron ağırlıklı Steiner Ağacı algoritması (Semi-asynchronous weighted Steiner Tree algorithm (ASYNTREE)) algoritmasının genel fikri, tanımı ve örnek uygulaması verilecektir.

4.2.4.1. Genel Fikir

ASYNTREE algoritması, STREE algoritmasının yarı asenkron olarak çalışan sürümüdür. ASYNTREE algoritmasında düğümler rauntları zamanlayıcıyla değil, *ÇIKIŞ_DÜĞÜMÜ*'nden yayılan *BAŞLA* mesajıyla başlatır. STREE algoritmasının ilk raundunda *HELMAN* düğümler arasında patika olanlar aynı ağaç içine konur. Bu işlemin yapılması için her *HELMAN* düğüm kendi kimliğini içeren bilgiyi komşularına gönderir, komşuları içinde *HELMAN* durumunda olan düğümler bu mesajı bir kez iletirler ve bu işlem tekrarlı bir şekilde devam eder. Dikkat edilirse bu algoritmanın tasarımını çok değiştirmeden asenkron olarak düzenlenmesi zordur. Bir *HELMAN* düğümün kendisiyle arasında *HELMAN* düğümlerden oluşan bir patika olan tüm *HELMAN* düğümlerin bilgisini aldığını ve böylece işlemini tamamlandığını bilmesi gerekir ki bu işlemi her düğümün STREE algoritmasında olduğu gibi yayma tabanlı yapması zor görünmektedir. Bu sebepten dolayı ASYNTREE algoritmasında *HELMAN* düğümler arasındaki ağacı oluşturmak yeni bir algoritma önerdik. Bu algoritmaya DTOR_TREE algoritması olarak isimlendirdik. DTOR_TREE

algoritması bittikten sonra ASYNTREE algoritmasında her düğüm β senkronizeri ile bir sonraki raunda geçmiştir.

STREE algoritmasının 2. raundunda *ÇIKIŞ_DÜĞÜMÜ*'ne doğru *BĞLAN_İST* mesajlarının gitmesi için yeni bir ağaç oluşturulsa da ASYNTREE algoritmasında ASYNSET tarafından oluşturulan senkronizasyon ağacı kullanılabilinir. Eğer ASYNTREE algoritmasından önce bir senkronizasyon ağacı kurulmamışsa, DTOR_TREE algoritmasından önce 1.rauntta senkronizasyon ağacı kurmak gerekir.

STREE algoritmasında bağlayıcı seçilen her rauntta *ÇIKIŞ_DÜĞÜMÜ*'ne doğru oluşturulan ağaç kullanılarak en küçük ağırlıklı bağlayıcı bulunmuş, *ÇIKIŞ_DÜĞÜMÜ* en küçük ağırlıklı düğüme bağlayıcı olduğunu belirten mesajı göndermiş olmasına rağmen bağlayıcı seçen rauntlar arasında bir senkronizasyon mekanizması konulmamış olup, STREE algoritması senkron rauntlara bölünmüştür. ASYNTREE algoritmasında rauntlar kendi içlerinde ve birbirlerine göre asenkrondur. Ayrıca STREE algoritmasında 3 raunda bölünmüş bağlayıcı düğüm seçimi ASYNTREE algoritmasında tek raunt içinde yapılacak şekilde tasarlanmıştır.

4.2.4.2. Tanım

ASYNTREE algoritmasının basamakları Şekil 4.21'de verilmiştir. Raunt 1'de tasarladığımız DTOR_TREE algoritmasıyla *HELMAN* düğümler arasındaki ağaçlar oluşturulur. Bu algoritma çalıştıktan sonra her *HELMAN* düğüm hangi ağaca bağlı olduğunu ve bu ağaç üzerinde kaç adet eleman olduğunu bilir. Bu bilgi bağlayıcı aday düğümlerin ağırlık oranlarını hesaplaması için gereklidir. Bu algoritma biterken her düğüm senkronizasyon ağacı üzerinden senkronize olurlar, böylece *ÇIKIŞ_DÜĞÜMÜ* tarafından bağlayıcı seçen raunt 2'yi başlatır. DTOR_TREE algoritması yerine üzerinde az düzenleme yapılarak GHS algoritması da kullanılabilinir. Raunt 2, ASYNTREE algoritmasının düzenlenmiş sonlu durum makinesi kullanılarak bağlayıcı aday düğümler bulunur ve adaylar arasından en küçüğü *ÇIKIŞ_DÜĞÜMÜ* tarafından seçildikten sonra yeniden aynı raunt bütün *HELMAN* düğümler aynı ağaç üzerinde olana kadar devam eder.

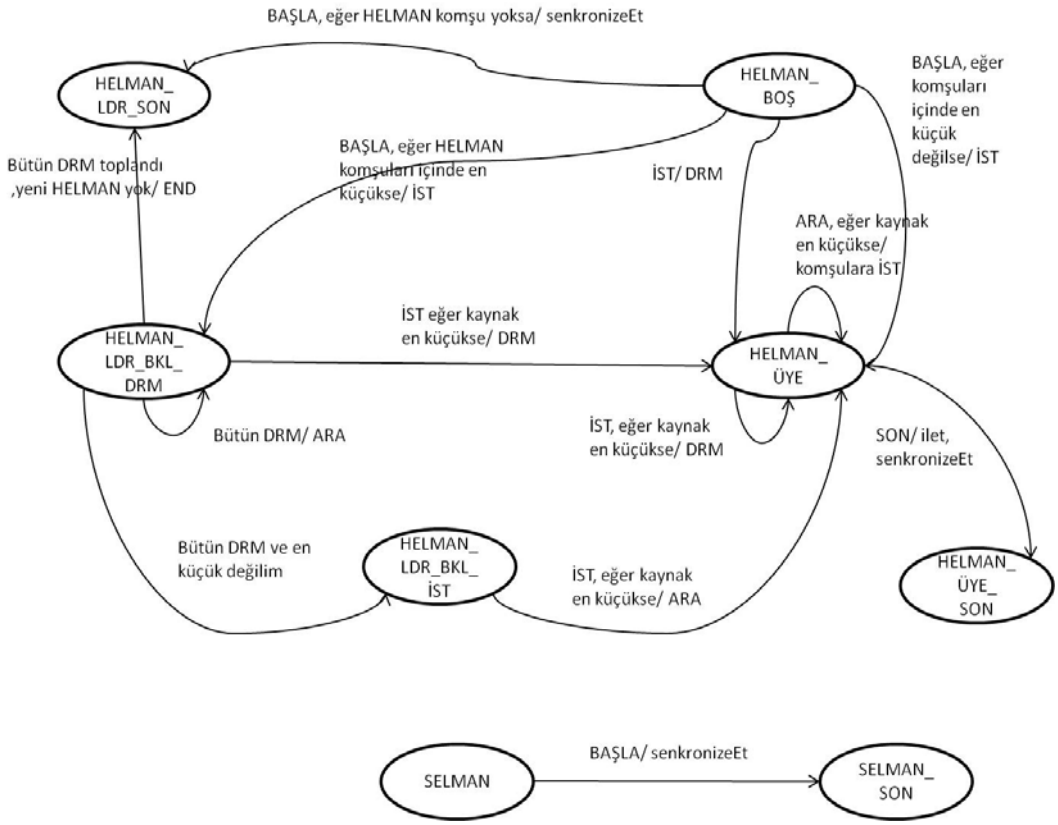
Raunt 1. DTOR_TREE algoritmasıyla veya GHS algoritmasıyla *HELMAN* düğümler arasındaki ağaçlar bulunur. DTOR_TREE algoritmasını bitiren düğüm senkronizasyon ağacı üzerinden senkronize olur.

Raunt 2. ASYNTREE algoritmasının düzenlemiş sonlu durum makinesi kullanılarak bağlayıcı düğüm seçilir. Eğer bir bağlayıcı düğüm seçilmişse *ÇIKIŞ_DÜĞÜMÜ*, *AĞAÇ_KYNŞTR* mesajıyla yeniden 2. raundu başlatır, ağırlık oranlarını günceller ve ağaçları birleştirilir. Eğer bir bağlayıcı düğüm

seçilmemişse *ÇIKIŞ DÜĞÜMÜ SON* mesajıyla algoritmayı bitirir.

Şekil 4.21 ASYNTREE algoritmasının basamakları

DTOR_TREE algoritmasıyla birbirleri üzerinden patikalar olan *HELMAN* düğümler aynı ağaç içine konur ve düğümler algoritmayı sonlandırırken çıkış düğümüyle senkronize olurlar, böylece bağlayıcı seçen yeni raunt başlayabilir. DTOR_TREE algoritmasının sonlu durum makinesi Şekil 4.22'de verilmiştir. Algoritma başlarken her düğüm ASYNSET algoritması tarafından bulunan *HELMAN* veya *SELMAN* durumlarında olabilir. Ayrıca ASYNSET algoritması sonlandığında her düğüm komşularının durumunu öğrenir. DTOR_TREE algoritması sadece *HELMAN* düğümler arasında olduğu için algoritma başlarken *SELMAN* düğümler algoritmayı sonlandırmak ve senkronize olmak amacıyla Şekil 4.23'de gösterilen *senkronizeEt* prosedürünü çağırıp *SELMAN_SON* durumuna geçer.

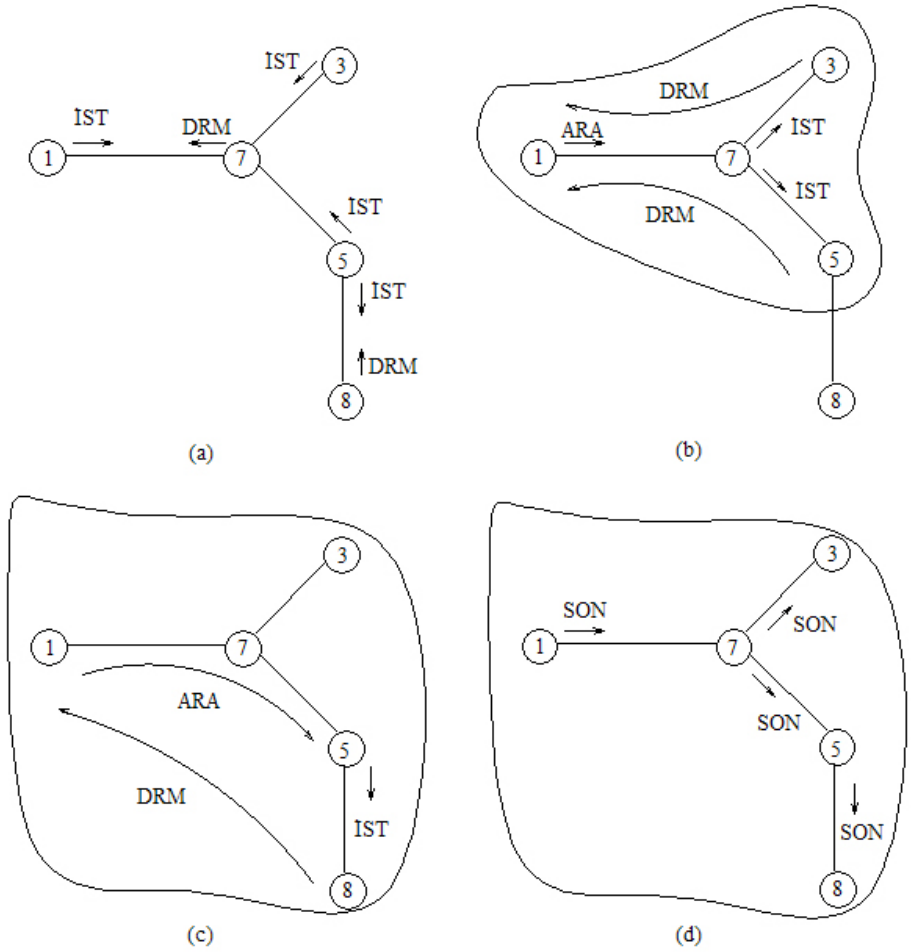


Şekil 4.22 DTOR_TREE algoritması sonlu durum makinesi

senkronizeEt prosedürü()
Eğer senkronizasyon ağacında *YAPRAK* durumundaysam
  Ebeveynime *TAMAM* gönder
Değilse
  senkronizeOldu ←DOĞRU
  Eğer çocuklarımın hepsinden TAMAM mesajı aldıysam
  Ebeveynime *TAMAM* gönder

```
    Koşul bitti
Koşul bitti
```

Şekil 4.23 DTOR_TREE algoritması *senkronizeEt* prosedürü



Şekil 4.24 DTOR_TREE algoritmasının Örnek Uygulaması

HELMAN düğümler DTOR_TREE algoritmasına *HELMAN_BOŞ* durumunda başlarlar. Her *HELMAN* düğüm kendi başına bir ağacı oluşturur ve bu ağacı büyütmek için öncelikle komşu listesini gözden geçirir. Eğer bir *HELMAN* düğümün komşusu yoksa *senkronizeEt* prosedürünü çağırıp *HELMAN_LDR_SON* durumuna geçtikten sonra algoritmayı sonlandırır. Çünkü bu *HELMAN* düğümün bağlanacağı bir ağaç yoktur. Eğer bir *HELMAN* düğüm, *HELMAN* komşuları içinde en küçük kimliğe sahip değilse *HELMAN_ÜYE* durumuna geçip liderlik hakkını kaybeder ve bağlanacağı ağacın liderinden gelecek *İST(düğüm_kimliği)* mesajını bekler. Eğer bir *HELMAN* düğüm, *HELMAN* komşuları içinde en küçük kimliğe sahipse komşularına *İST* mesajı gönderip *LDR_DRM_BKL* durumuna geçiş yapar. *İST* mesajını alan komşu düğümler eğer mesajın kaynağı komşu listesi içinde en küçük kimliğe sahipse komşularının durumlarını *DRM(düğüm_kimliği,komşuların listesi)* mesajı içinde gönderirler. Şekil 4.24 (a)'da bu durum bir örnekle gösterilmiştir. Bu şekildeki bütün düğümler *HELMAN* olup DTOR_TREE algoritmasını çalıştırmaya başlamışlardır. Düğüm 3, düğüm 5 ve düğüm 1 komşuları içinde en küçük kimliğe sahip oldukları için düğüm 7'ye *İST* mesajı gönderiyorlar. Düğüm 5 aynı zamanda düğüm 8'e de *İST*

mesajını gönderiyor. Düğüm 7 komşuları içindeki en küçük kimlikli düğüm 1 olduğu için sadece düğüm 1'e *DRM* mesajını gönderirken, düğüm 8 *DRM* mesajını tek komşusu olan düğüm 5'e gönderiyor. Bu işlemlerin sonunda düğüm 1 ve düğüm 7 aynı ağacın içine giriyorlar.

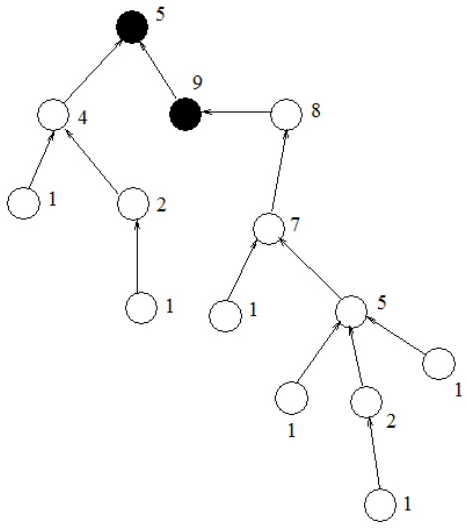
HELMAN_LDR_BKL_DRM durumundaki bir düğüm üyelerinden *DRM* mesajlarını topladıktan sonra *DRM* topladığı düğümlere *ARA(düğüm_kimliği,hedef_düğüm_kimliği)* mesajı gönderiyor. *ARA* mesajını alan *HELMAN_ÜYE* durumundaki düğümler bu mesajı *İST* mesajı olarak komşularına iletiyorlar. *İST* mesajını alan *HELMAN_ÜYE* veya *LDR_DRM_BKL* durumundaki düğümler *DRM* mesajıyla cevap veriyorlar. Şekil 4.24 (b)'de bu durum bir örnekle gösterilmiştir. Düğüm 1, düğüm 7'ye *ARA* mesajı göndermiş, düğüm 7'de bu mesajı düğüm 3 ve düğüm 5'e *İST* olarak iletmiştir. *İST* mesajını alan düğüm 3 ve düğüm 5 bu mesaja *DRM* ile cevap verir. *DRM* mesajı düğüm 7 üzerinden geçerek düğüm 1'e iletilir, düğüm 5 ve düğüm 3, düğüm 1'in ağacı içine dâhil olur. Düğüm 7, bu düğümleri ağacı içine aldıktan sonra benzer bir şekilde düğüm 5'e *ARA* mesajı gönderir ve benzer olarak düğüm 8'i ağacı içine alır. Bu basamak Şekil 4.24 (c)'de bu durum gösterilmiştir. *HELMAN_LDR_BKL_DRM* durumundaki bir düğüm ağacına ekleyebileceği başka bir *HELMAN* bulamadıklarında *SON* mesajını gönderip *senkronizeEt* olurlar. *SON* mesajının içinde ağacın içinde kaç adet düğüm olduğu bilgisi ve ağacın kimliği bulunur. Ağacın kimliği liderin kimliğiyle aynıdır. Böylelikle her *HELMAN* düğüm hangi ağaca bağlı olduğunu ve bu ağaç üzerinde kaç düğüm olduğunu bilir. Bu durum Şekil 4.24 (d)'de verilmiştir. Düğüm 1'den gönderilen *SON* mesajı diğer düğümlere yayılmış ve ağaç üzerinde algoritmanın çalışması tamamlanmıştır.

DTOR_TREE algoritmasının diğer rauntlarla senkronize olmasını sağlamak amacıyla tasarlanan *senkronizeEt* prosedürü Şekil 4.25'de ve *TAMAM* mesajlarının işlenmesi için gerekli prosedür Şekil 4.26'da verilmiştir. *SenkronizeEt* prosedürü ASYNSET için verilen *senkronizeEt* prosedürünün basitleştirilmişidir. Senkronizasyon ağacında *YAPRAK* durumunda olan bir düğüm ebeveynine *TAMAM* mesajını gönderir, eğer *YAPRAK* durumunda değilse çocuklarından *TAMAM* almışsa *TAMAM* mesajını gönderir. Bir düğümün çocuklarından *TAMAM* mesajı aldığı zaman çalıştırdığı prosedür ASYNSET algoritmasında *TAMAM* mesajlarının işlenmesi için kullanılan sonlu durum makinesine çok benzerdir. Bu prosedüre göre *ÇIKIŞ_DÜĞÜMÜ* çocuklarından *TAMAM* mesajı almışsa ve *senkronizeEt* prosedürünü işletmişse *BAŞLA* mesajını göndererek ASYNTREE algoritmasını başlatır. Eğer bu prosedürü işleten düğüm *ÇIKIŞ_DÜĞÜMÜ* değilse, çocuklarından *TAMAM* mesajlarını topladıktan ve *senkronizeEt* prosedürünü işlettikten sonra *TAMAM* mesajını ebeveynine iletir.

TAMAM alındığında
Eğer çocuklardan bütün *TAMAM* mesajları alındıysa ve
senkronizeOldu=DOĞRU'ysa
    Eğer *ÇIKIŞ_DÜĞÜMÜ*'ysem
        Çocuklara *BAŞLA* mesajı gönder
    Değilse ebeveyne *TAMAM* mesajı gönder
    Koşul bitti
Koşul bitti

Şekil 4.25 DTOR_TREE algoritmasında *TAMAM* mesajlarının işlenmesi

DTOR_TREE algoritmasının tasarımı ve kodlanması kolay olsa da mesaj karmaşıklığı Analiz Bölümünde ispatlanacağı üzere $|HK_T|$ hâkim elemanların kendi aralarında oluşturduğu en büyük ağacın eleman sayısı olmak üzere $O(|HK_T|^2 \log_2(|HK_T|))$'e eşit olur. Böylelikle asenkron algoritmanın genel olarak mesaj karmaşıklığı $O(N^2)$'den $O(N^2\log_2(N))$'ye çıkar. Bu durumdan kurtulmak için DTOR_TREE algoritmasının yerine GHS algoritmasını değişiklikler ile birlikte kullanabiliriz. GHS Algoritması üzerinde yapmamız gereken değişikliklerin listesi şöyledir:



Şekil 4.26 GHS algoritmasında *REPORT* mesajları kullanılarak ağaçta kaç adet düğümün olduğunun bulunması

- GHS algoritmasında *REPORT* mesajları ağacın altındaki düğümlerden çekirdek düğümlere doğru çıkarken ağacın altında toplam kaç düğümden *REPORT* toplandığını tutan bir değişken koyarsak çekirdek düğümleri ağaç üzerindeki kaç adet düğüm olduğunu bulabilir. Şekil 4.26'da bu içinde sayılarla örnek bir uygulama gösterilmiştir. İçi koyu ile gösterilen düğümler ağaçtaki çekirdek düğümlerdir. Okların yönleri *REPORT* mesajının gönderileceği yönlerdir. Yaprak düğümler sadece kendileri gönderdikleri için *REPORT*(1) olarak mesajı gönderirler. 2 yaprak çocuğu olan ve bir yaprak

çocuğu olan düğüme ebeveyni olan düğümün çocuğu olan bir düğümün toplamda *REPORT*(5) mesajı göndermesi gerekir.

- GHS algoritmasını sonlandıran çekirdek düğümler (*core nodes*), dal (*branch*) kenarlar üzerinden *SON* mesajı gönderirlerse her düğüm hangi ağaca bağlı olduğunu ve ağaçta kaç adet düğüm olduğunu keşfedebilir. Çekirdek düğümler içinde kimliği küçük olan düğüm lider seçilirse tek bir lider seçilmiş olur. GHS algoritmasında her düğüm ebeveynini *in-branch* değişkeniyle bilir, ayıca *SON* mesajı yayılırken bu bilgi tekrardan sağlanabilir.

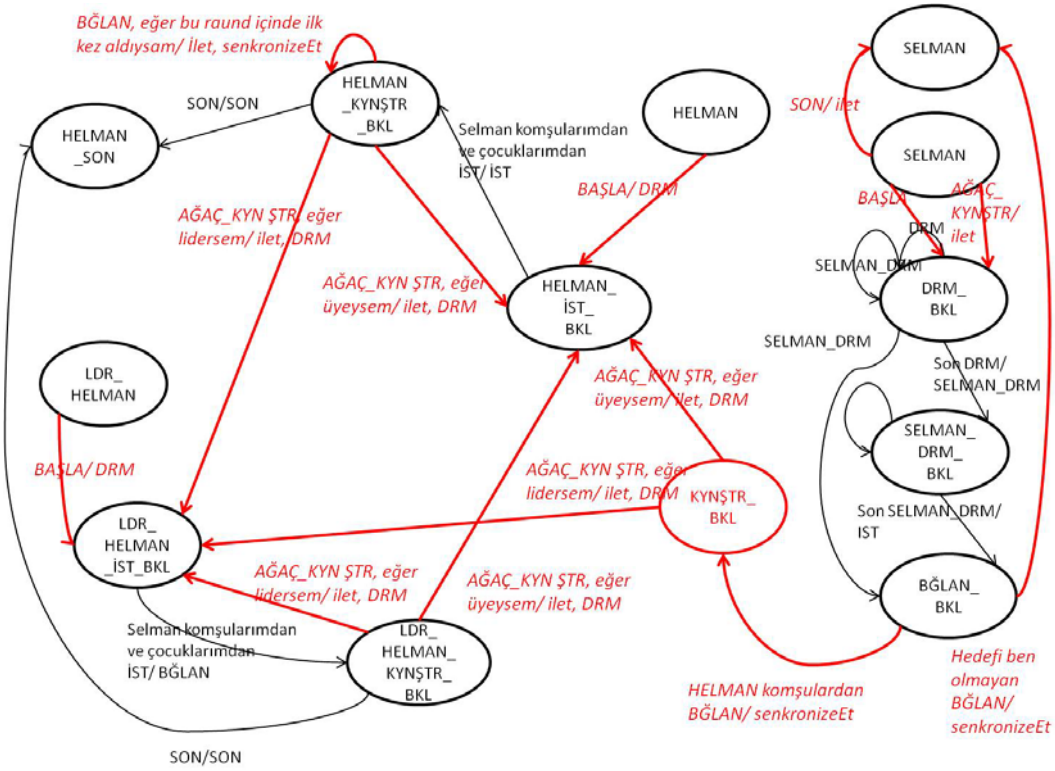
- GHS algoritmasında düğümler algoritmayı sonlandırırken kendi tasarladığımız *senkronizeEt* prosedürünü çağırırsak düğümleri senkronize edebiliriz.

GHS Algoritmasının mesaj karmaşıklığı $O(\log_2(N)\ N)$'dir. Biz her düğümün sadece 2 adet fazladan mesaj (*TAMAM* ve *SON*) mesajı göndermesini önerdiğimiz için mesaj karmaşıklığı sabit kalır. Böylelikle asenkron algoritmanın analiz bölümünde gösterileceği üzere mesaj karmaşıklığı $O(N^2)$ olur.

ASYNTREE algoritmasının STREE algoritması üzerinden düzenlemiş sonlu durum makinesi Şekil 4.27'de verilmiştir. Bu sonlu durum makinesinde STREE'ye göre yapılan değişiklikler kırmızı ve yana yatık yazı tipiyle yazılmıştır. Yeni düzenlenen sonlu durum makinesinde *ÇIKIŞ_DÜĞÜMÜ* yeni raundu başlatmak, ağırlıklarını güncellemek ve ağaçları birleştirmek için *AĞAÇ_KYNŞTR(yeni_ağaç_kimliği, ağaçtaki_düğüm_sayısı, bağlanmış ağaçların listesi)* mesajını ağa yayar. *AĞAÇ_KYNŞTR* mesajının içindeki alanlar STREE'ye göre değiştiği için tekrardan yazılmıştır. Buna uygun olarak *BĞLAN_İST*(*düğüm_kimliği*, *gerçek_düğüm_kimliği*, *ağaçtaki_düğüm_sayısı, bağlanmış ağaçların listesi*) mesajının alanları da güncellenmiştir. Bu mesajlar dışındaki mesajların alanları aynıdır.

ÇIKIŞ_DÜĞÜMÜ bağlayıcı düğümü en küçük ağırlıklı aday olarak seçer ve ağacı bu bağlayıcı olarak bağlarken bu ağaç üzerindeki en küçük kimlikli düğümü lider olarak, diğer düğümleri ağaç üyesi olarak seçer ve bu bilgiyi mesajın içinde gönderir. ASYNTREE algoritmasındaki değişiklikleri şu şekilde listeleyebiliriz:

- *HELMAN_İST_BKL* durumundan *HELMAN_KYNŞTR_BKL* geçiş yaparken *senkronizeEt* prosedürüyle senkronizasyon ağacı üzerinden senkronize olurlar. *HELMAN_KYNŞTR_BKL* durumundaki bir düğüm bir sonraki rauntta *ÇIKIŞ_DÜĞÜMÜ'*nden yayılacak *AĞAÇ_KYNŞTR* mesajını bekler.



Şekil 4.27 ASYNTREE algoritmasının sonlu durum makinesi

- *LDR_HELMAN_KYNŞTR_BKL, LDR_HELMAN_KYNŞTR_BKL* veya *BĞLAN_İST_ONAY_BKL* durumundaki bir düğüm *AĞAÇ_KYNŞTR* mesajını aldığında eğer kendisi yeni ağaç üzerinde üyeyse *DRM* mesajı göndererek *HELMAN_İST_BKL* durumuna geçiyor. *AĞAÇ_KYNŞTR* mesajını aldığında eğer kendisi yeni ağaç üzerinde liderse *DRM* mesajı göndererek *LDR_HELMAN_İST_BKL* durumuna geçiyor.

- *BĞLAN_BKL* durumundaki bir düğüm kendisine gönderilmeyen bir *BĞLAN* mesajını alırsa kendisinin bu raunt içinde aday olmadığını tespit eder ve *senkronizeEt* prosedürünü çağırır.

- *BĞLAN_BKL* durumundayken çevresindeki tüm komşularından *BĞLAN* mesajı aldığında *BĞLAN_İST_ONAY_BKL* durumuna geçiş yapmadan *senkronizeEt* prosedürünü çağırıyor.

- *SELMAN* durumundaki bir düğüm *AĞAÇ_KYNŞTR* mesajını aldığında bu mesajı komşularına göndererek *DRM_BKL* durumuna geçiyor.

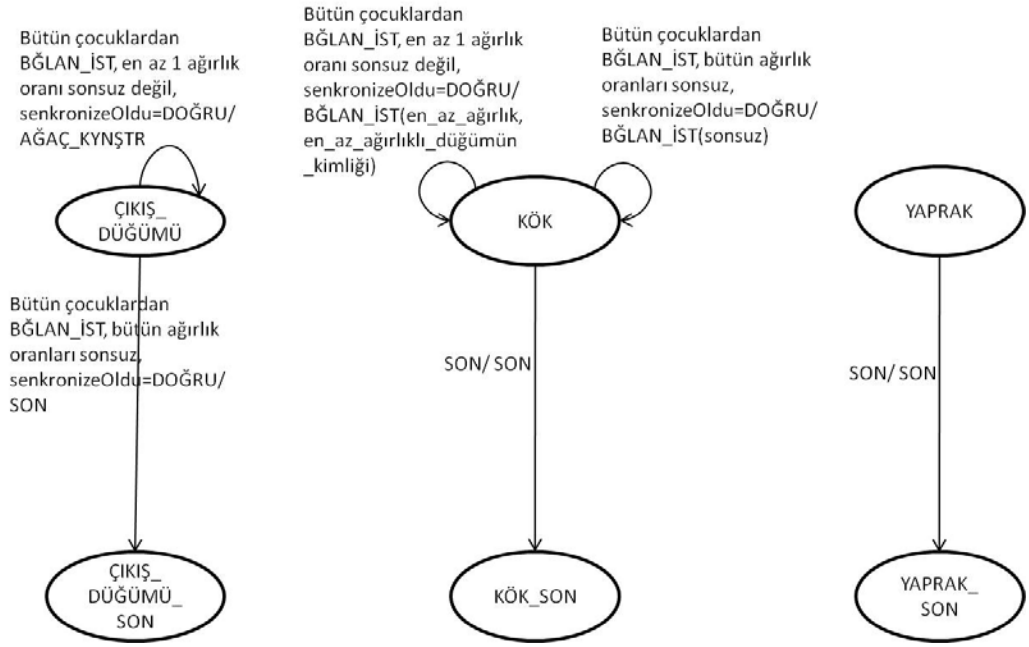
- Bir düğüm *LDR_HELMAN_İST_BKL* durumundan *LDR_HELMAN_KYNŞTR_BKL* durumuna geçerken *BĞLAN* mesajını çocuklarına gönderir. *BĞLAN* mesajını alan *HELMAN_KYNŞTR_BKL* durumundaki düğümler bu mesajı komşularına birer kez iletirler.

```
senkronizeEt prosedürü()
Eğer senkronizasyon ağacında YAPRAK durumundaysam
  Eğer BĞLAN_İST_ONAY_BKL durumundaysam // Aday bağlayıcıyım
     Ebeveyne BĞLAN_İST(ağırlık_oranı, düğüm_kimliği) gönder
  Değilse ebeveyne BĞLAN_İST(∞, düğüm_kimliği) mesajı gönder // Aday
bağlayıcı değilim
   Koşul bitti
Değilse
   Eğer çocuklarımdan BĞLAN_İST aldıysam
      Ebeveyne BĞLAN_İST(en_küçük_ağırlık_oranı,
karşılık_gelen_düğüm_kimliği) gönder
   Değilse senkronizeOldu ←DOĞRU
   Koşul bitti
Koşul bitti
```

Şekil 4.28 ASYNTREE algoritmasının *senkronizeEt* prosedürü

ASYNTREE algoritmasının *senkronizeEt* prosedürü Şekil 4.28'de verilmiştir. Bu prosedür ASYNSET'de verilen prosedüre çok benzemektedir. Eğer bir düğüm β senkronizasyon ağacı üzerinde *YAPRAK* durumundaysa ve sonlu durum makinesini işletirken *BĞLAN_İST_ONAY_BKL* durumundaysa ebeveynine kendi kimliğiyle birlikte *BĞLAN_İST*(*düğüm_kimliği*, *gerçek_düğüm_kimliği*, *ağaçtaki_düğüm_sayısı, bağlanmış ağaçların listesi*) mesajını gönderir. Bu mesajın içindeki *düğüm_kimliği* gönderen düğüm kimliği olarak kullanırken *gerçek_düğüm_kimliği* ebeveynler tarafından en küçük kimlikli düğümün bilgisini göndermek için kullanılır. Eğer *YAPRAK* düğüm *BĞLAN_İST_ONAY_BKL* durumunda değilse bu düğümün aday bağlayıcı olmadığını gösterir, bundan dolayı bu düğüm komşusuna ağırlık oranı olarak ∞ gönderir. Eğer *YAPRAK* durumunda olmayan bir düğüm bütün çocuklarından *BĞLAN_İST* mesajı toplarsa bu mesajlar içinde en küçük ağırlık oranına sahip olanını kimliğiyle birlikte ebeveynine gönderir.

ASYNTREE algoritmasında ASYNSET algoritmasında olduğu gibi her düğüm kendi senkronizasyon işlemini bitirdikten sonra *BĞLAN_İST* mesajı gönderir. Bu mesajların işlenmesi için gerekli sonlu durum makinesi ASYNSET algoritması için verilen sonlu durum makinesiyle çok benzerdir. Şekil 4.29 bu sonlu durum makinesi verilmiştir. *ÇIKIŞ_DÜĞÜMÜ* yeni bir raundu başlatmak için içinde bağlayıcı düğümün ve bağlanan ağaçların, lider ve üye düğümlerle birlikte bilgisini bulunduran *AĞAÇ_KYNŞTR* mesajını gönderir. *ÇIKIŞ_DÜĞÜMÜ* bütün komşularından ∞ ağırlık aldığında *SON* mesajını gönderir ve algoritma sonlanır.



Şekil 4.29 ASYNTREE algoritmasının senkronizasyonu için kullanılan sonlu durum makinesi

4.2.4.3. Örnek Uygulama

Bu bölümde ASYNTREE algoritmasının bir örnek uygulaması verilecektir. Bu örnek uygulama Şekil 4.13'de gösterilmiştir. ASYNSET algoritması tamamlandıktan sonra içi gri ile dolu düğümler hâkim küme içine girerler. Bu düğümler ASYNTREE algoritmasıyla birleştirilip bağlı hâkim küme oluşur.

ASYNTREE algoritmasının ilk raundunda DTOR_TREE veya düzenlenmiş GHS algoritması kullanılarak *HELMAN* düğümler ağaç yapısına konur. Bu ağacın oluşması önceki bölümlerde örneğini anlattığımız DTOR_TREE ile birlikte aynı şekilde yapılabilir. Ağaçlara ait olan kenarlar Şekil 4.13'da gösterilmiştir. Toplamda 4 adet ağacımız bulunmaktadır. İlk ağaç a düğümünden oluşur, ağaçtaki düğüm sayısı 1 ve ağacın kimliği a'dır. İkinci ağacımız sadece h düğümünden oluşur, ağaçtaki düğüm sayısı 1 ve ağacın kimliği h'dır. Üçüncü ağacımız j ve o düğümlerinden oluşur, ağacın kimliği j, düğüm sayısı 2 ve ağacın kimliği j'dir. Dördüncü ağaç k, l ve m düğümlerinden oluşur, ağaçtaki eleman sayısı 3 ve ağacın kimliği l'dir.

İlk rauntta ağaçlar oluştuktan sonra ikinci rauntta ilk bağlayıcı düğüm seçilmeye başlar. *SELMAN* durumunda olan düğüm g, *ÇIKIŞ_DÜĞÜMÜ*'nden yayılan *BAŞLA* mesajını aldıktan sonra *DRM_BKL* durumuna geçer ve *HELMAN* düğümlerden *DRM* mesajı bekler. Düğüm h, düğüm o ve düğüm j *HELMAN* durumundayken *BAŞLA* mesajını aldıktan sonra ağaçlarının kimliklerini ve ağaçlarının içindeki düğüm sayısını *DRM* mesajı içinde gönderirler. *DRM* mesajlarının paylaşılması ve *İST* mesajlarının gönderilmesi STREE ile aynıdır. Düğüm g ve düğüm o'nun *İST*

mesajları düğüm j'ye ulaşır. Düğüm j, düğüm g'den ve düğüm o'dan *İST* mesajlarını aldıktan sonra başka alacağı *İST* mesajı kalmadığı için en küçük ağırlıklı *İST* mesajı olarak düğüm g'yi bulur ve *BĞLAN* mesajını düğüm o'ya doğru düğüm g'nin de alacağı şekilde gönderir, *LDR_HELMAN_KYNŞTR_BKL* durumuna geçer ve Şekil 4.12'de gösterilen ağaç üzerinden senkronize olur. Böylelikle düğüm g, düğüm o'nun ağacından *BĞLAN* mesajını almış olur. Eş zamanlı olarak düğüm o *HELMAN_İST_BKL* durumundayken *BĞLAN* mesajını aldıktan sonra bu mesajı iletir, bu raunt içindeki işlemini bitirmiş olur ve senkronize olur. Düğüm h, düğüm c'den 2.5 ağırlıklı *İST* mesajı, düğüm g'den 2,34 ağırlıklı *İST* mesajını topladıktan sonra *LDR_HELMAN_İST_BKL* durumundan *LDR_HELMAN_KYNŞTR_BKL* durumuna geçerek, düğüm g'yi hedefleyen *BĞLAN* mesajını gönderir, bu mesajı alan düğüm g, çevresindeki bütün *HELMAN* düğümlerden *BĞLAN* mesajını topladığı için *KYNŞTR_BKL* durumuna geçip senkronize olur ve senkronizasyon ağacı üzerinden göndereceği *BĞLAN_İST* mesajının içine senkronizasyon ağacı üzerinde çocuklarından gelen *BĞLAN_İST* mesajlarının içinde daha düşük bir ağırlık olmadığı için kendi ağırlığını, kimliğini ve birleştireceğini ağaçların kimliğini koyup gönderir. Düğüm c, düğüm g'nin hedeflendiği *BĞLAN* mesajını aldıktan sonra kendisinin aday olmadığını görür, senkronize olur, *SELMAN* durumuna geçip bir sonraki raundu bekler. Bütün düğümler senkronize olduktan sonra *ÇIKIŞ_DÜĞÜMÜ'*ne senkronizasyon ağacı üzerindeki çocuklarından *BĞLAN_İST* mesajı ulaşır. *ÇIKIŞ_DÜĞÜMÜ'*nün bu mesajlar sonucunda bulduğu en küçük ağırlık 2.34 ve bu ağırlığın sahibi g düğümüdür. *ÇIKIŞ_DÜĞÜMÜ* yeni raundu başlatmak ve bu ağaçları birleştirmek için *AĞAÇ_KYNŞTR* mesajının içinde bu ağaçların kimliğini ve g düğümünün kimliğini gönderir. *AĞAÇ_KYNŞTR* mesajını alan h, g, o ve j düğümleri aynı ağacın içine dâhil olur ve yeni raunda başlarlar. Diğer düğümler için *AĞAÇ_KYNŞTR* mesajı sadece yeni raunda başlamayı gösterir.

2. rauntta ilk raunda çok benzer olarak *HELMAN* düğümler *DRM* mesajlarını gönderir ve *SELMAN* düğümler bu mesajları topladıktan sonra ağırlık oranlarını bulup *İST* mesajları içinde gönderir. Bu rauntta düğüm n'nin ağırlık oranı en düşük olduğu için düğüm n bağlayıcı olarak seçilir ve düğüm k, düğüm l, düğüm m, düğüm o, düğüm j, düğüm g ,düğüm h, düğüm n aynı ağaç içine dâhil olur. 3.rauntta düğüm c bağlayıcı olarak seçilir, bütün *HELMAN* düğümler aynı ağacın içine dâhil olur, düğümler raundun sonunda senkronize olduktan sonra *ÇIKIŞ_DÜĞÜMÜ SON* mesajını gönderir ve algoritma sonlanır.

4.3. Analiz

Bu bölümde FLOODSET, SSET, ASYNSET, FLOODTREE, STREE ve ASYNTREE algoritmalarının doğruluk ve yaklaşım oranı analizi, mesaj karmaşıklıkları, zaman karmaşıklıkları ve uzay karmaşıklıkları verilecektir.

4.3.1. Doğruluk ve Yaklaşım Oranı Analizi

Teorem 4.1. FLOODSET algoritması CENSET ile aynı düğümleri hâkim küme içine alır.

İspat. FLOODSET algoritmasında her düğüm raundun başında ağırlık oranlarını bütün ağa yayar. Böylece bütün düğümler birbirlerinin ağırlık oranlarını bilirler. Raundun sonunda hâkim küme içine giren düğüm *HÂKİM_ELEMAN* mesajıyla komşularını örter. Komşuları da *ÖRTÜLDÜM* mesajıyla kendi komşularına haber verip ağırlıklarını güncellemesini sağlar. Böylece her rauntta her düğüm birbirlerinin ağırlık oranını bilir ve ağırlık oranları sürekli güncellenir. Sonuç olarak FLOODSET algoritması, CENTSET ile aynı düğümleri hâkim küme içine alır. ■

Gözlem 4.1. SSET ve ASYNSET algoritmaları sonlandıktan sonra *HÂKİM_ELEMAN* durumundaki düğümler hâkim eleman, *SIRADAN_ELEMAN_SON* durumundaki düğümler en az bir hâkim elemana bir zıplama uzaklıkta bağlı olan ve en az bir hâkim elemana bağlı elemanlardır.

Gözlem 4.2. CENTSET algoritması işletildikçe düğümlerin ağırlık oranları azalmaz.

Ön Teorem 4.1. Bir n_v düğümünün ağırlık oranı w(v) eğer 2 zıplama uzaklığındaki tüm düğümlerden küçükse CENTSET tarafından seçilir.

İspat. Aksinin doğru olduğunu varsayalım. Bu durumda n_v'nin 1 zıplama uzaklığındaki komşularından en az bir tanesinin *HÂKİM_ELEMAN* olması gerekir. Bu düğüme n_x diyelim, $w(x) < w(v)$ olması gerekir. Hâlbuki w(v), 2 zıplama uzaklıktaki düğümler içerisinde en küçüğüdür. w(v)'nin artabilmesi için n_v nin 1 zıplama uzaklıktaki komşularının en az bir tanesinin örtülmesi gerekmektedir fakat n_v düğümünün ağırlık oranı 2 zıplama uzaklığındaki tüm komşularından küçük olduğu için bu durumun gerçekleşmesi mümkün değildir. w(x)'in azalabilmesi Gözlem 4.2'e göre mümkün değildir. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Ön Teorem 4.2. MODSET ile CENTSET aynı düğümleri hâkim eleman olarak seçer.

İspat. Ön Teorem 4.1'e göre bir n_v düğümünün ağırlık oranı 2 zıplama uzaklığındaki tüm düğümlerden küçükse CENTSET tarafından hâkim eleman olarak seçilir. Bu özellikten yola çıkarak her iterasyonda birbirlerine en az 3 zıplama

uzaklıkta olan ve ağırlık oranı 2 zıplama uzaklığındaki düğümlerden küçük olan her düğüm küme örtüsü içine aynı anda alınır ve komşularının ağırlıkları güncellenir. Böylece CENTSET ve MODSET aynı düğümleri seçerler. ■

Ön Teorem 4.3. SSET algoritması her rauntta 2 zıplama uzaklığında en küçük ağırlık oranına sahip olan düğümleri hâkim eleman olarak seçer.

İspat. Tersinin doğru olduğunu düşünelim. Böylece SSET algoritmasında her rauntta 2 zıplama uzaklığında en küçük ağırlık oranına sahip düğüm hâkim eleman seçilmez veya seçilen hâkim eleman 2 zıplama uzaklığında en küçük ağırlık oranına sahip düğüm değildir. SSET'in sonlu durum makinesine göre bir raundun başında *BOŞ*, *BAĞLAN_BEKLE*, *SIRADAN_ELEMAN* ve *DİĞER_HÂKİM_ELEMANI_BEKLE* durumundaki düğümler ağırlık oranlarını hesaplayıp *AĞIRLIK_ORANI* mesajını komşularına gönderir ve *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçerler. *AĞIRLIK_ORANLARINI_BEKLE* durumundaki bir düğüm eğer komşularından tüm *AĞIRLIK_ORANI* mesajlarını topladıysa ve tüm komşularından küçük ağırlık oranına sahipse *BAĞLANMA_İSTEĞİ* mesajı göndererek *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumuna geçer. Birbirlerine komşu olan iki düğümün aynı anda *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumuna geçmesi mümkün değildir. Komşusundan daha büyük ağırlık oranına sahip düğüm *ADAY_DEĞİLİM* mesajı göndererek *BAĞLANMA_İSTEĞİ_BEKLE* durumuna geçer. *BAĞLANMA_İSTEĞİ_BEKLE* durumundaki düğüm bütün komşularından *ADAY_DEĞİLİM* veya *BAĞLANMA_İSTEĞİ* mesajlarını toparladıktan sonra en küçük ağırlığa sahip komşusunun kimliğini *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* içine gönderiyor. Bir düğümün *HÂKİM_ELEMAN* olması için *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumundayken *HÂKİM_ELEMAN* olmayan tüm komşularından kendi kimliğini içeren *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajı alması gereklidir ki bu düğüm 2 zıplama uzaklığındaki komşularından en küçük ağırlık oranına sahip olanıdır. *HÂKİM_ELEMAN* seçilen düğüm o raunt içinde 2 zıplama uzaklığında en küçük ağırlık oranına sahip olanıdır, bunun dışında bir düğüm *HÂKİM_ELEMAN* olarak seçilemez. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Ön Teorem 4.4. SSET algoritmasında bir raunt içinde hâkim eleman seçimi yaptıktan sonra düğüm ağırlıklarını günceller.

İspat. Tersinin doğru olduğunu düşünelim. Bu durumda hâkim eleman seçimi yapıldıktan sonra hâkim elemana bağlı olan düğümlerin ve bu düğümlerin komşularının ağırlıklarını güncellenmemesi gerekir. Böylece aşağıda listelediğimiz durumlardan herhangi birinin doğru olması gerekir:

1. Hâkim eleman seçimi doğru yapılmaz: Bu önermenin doğru olmadığı Ön Teorem 4.3'te gösterilmiştir.

2. Hâkim eleman seçimi yapıldıktan sonra komşular ağırlıklarını güncellemez: SSET'in sonlu durum makinesine göre bir düğüm *HÂKİM_ELEMAN* olarak seçildikten sonra *BAĞLAN* mesajı gönderir. *BAĞLAN* mesajını alan düğümler *BAĞLAN_BEKLE* 'ten başka bir durumda olamaz. Bu önerme de doğru değildir.

3. Hâkim elemanın komşularının komşuları ağırlıklarını güncellemez: SSET'in sonlu durum makinesine göre bir düğüm *HÂKİM_ELEMAN* tarafından gönderilen *BAĞLAN* mesajını aldıktan sonra tüm komşularına *BAĞLANDI* mesajı gönderir. *BAĞLANDI* mesajını alan *SIRADAN_ELEMAN*, *DİĞER_HÂKİM_ELEMANI_BEKLE* ve *BAĞLAN_BEKLE* durumundaki düğümler ağırlıklarını günceller. Düğümlerin *BOŞ*, *AĞIRLIK_ORANLARINI_BEKLE*, *BAĞLANMA_İSTEĞİ_BEKLE* ve *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumlarında kalmaları mümkün değildir çünkü her düğüm mutlaka *AĞIRLIK_ORANI* mesajını gönderir, *BAĞLANMA_İSTEĞİ* veya *ADAY_DEĞİLİM*'den herhangi birini gönderir ve eğer komşuları içinde en küçük ağırlığa sahip değilse *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajını da gönderir. *SIRADAN_ELEMAN_SON* ve *HÂKİM_ELEMAN* durumundaki düğümlerin *BAĞLAN* mesajı ile ağırlık oranlarını güncellemesi gerekmez. Bu önerme de doğru değildir.

Varsayımımız ile çelişiyoruz. Teoremimiz doğrudur. ■

Teorem 4.2. SSET algoritması, CENSET ile aynı düğümleri *HÂKİM_ELEMAN* olarak seçer.

İspat. Ön Teorem 4.2'den yola çıkarak, SSET algoritmasının MODSET ile aynı düğümleri hâkim eleman olarak seçtiğini gösterirsek, teoremimizi ispatlayabiliriz. SSET algoritmasının MODSET ile aynı düğümleri seçmediğini varsayalım. Böylece aşağıda listelediğimiz durumlardan en az bir tanesinin doğru olması gerekir:

1. SSET algoritması, 2 zıplama uzaklığında en küçük ağırlık oranına sahip herhangi bir düğümü hâkim eleman olarak seçmez. Bu önermenin doğru olmadığını Ön Teorem 4.3'de gösterdik.

2. SSET algoritması hâkim eleman seçimi yaptıktan sonra düğüm ağırlıklarını güncellemez. Bu önermenin doğru olmadığını Ön Teorem 4.4'de gösterdik.

Yukarıdaki önermeler doğru olmadığı için varsayımız ile çelişiyoruz, bundan dolayı teoremimiz doğrudur. ■

Sonuç Teoremi 4.1. SSET algoritmasının en küçük ağırlıklı bağlı hâkim kümenin ağırlığına yaklaşım oranı ln (*S*)' dir.

İspat. Teorem 4.1'e göre SSET algoritması, CENTSET ile aynı düğümleri hâkim eleman olarak seçer, bu sebeple iki algoritmanın yaklaşım oranı aynıdır. ■

Ön Teorem 4.5. SSET algoritması sonlandığında bir düğüm hâkim küme içinde veya hâkim küme içindeki bir elemana bağlanmış olur.

İspat. Ön Teorem 4.4 doğruysa Gözlem 4.1'e göre SSET algoritması sonlandığında *HÂKİM_ELEMAN* veya *SIRADAN_ELEMAN_SON* durumlarında olabilir. Tersinin doğru olduğunu varsayalım. Bu durumda algoritma sonlandığında *BOŞ*, *AĞIRLIK_ORANLARINI_BEKLE*, *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE*, *DİGER_HÂKİM_ELEMANI_BEKLE*, *BAĞLANMA_İSTEĞİ_BEKLE* veya *SIRADAN_ELEMAN* durumlarının herhangi birinde olabilir. Bu durumları sırayla aşağıda inceliyoruz:

- *BOŞ*: Bir düğüm *ZAMAN_AŞIMI* olduktan sonra *AĞIRLIK_ORANI* mesajı gönderip *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçiş yapar. *BOŞ* durumundayken *AĞIRLIK_ORANI* mesajını alan düğüm *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçiş yapar. Düğüm *BOŞ* durumunda kalmaz.

- *AĞIRLIK_ORANLARINI_BEKLE*: Düğüm *HÂKİM_ELEMAN* veya *SIRADAN_ELEMAN_SON* durumu dışındaki tüm komşularından *AĞIRLIK_ORANI* mesajını aldıktan sonra bu durumdan geçiş yapar. Belirtilen bu durumlar dışındaki her düğüm bir raundun başında *AĞIRLIK_ORANI* mesajı gönderir. Böylelikle herhangi bir düğüm bu konumda kalmaz.

- *BAĞLANMA_İSTEĞİ_BEKLE*: Komşu düğümler *ADAY_DEĞİLİM* veya *BAĞLANMA_İSTEĞİ* mesajı göndereceğinden dolayı düğüm bu konumda kalmaz, *BAĞLAN_BEKLE* durumuna geçiş yapar.

- *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE*: Komşu düğümler *ADAY_DEĞİLİM* veya *BAĞLANMA_İSTEĞİ* mesajı göndereceğinden dolayı düğüm bu konumda kalmaz, *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajlarının hepsinin içinde kendi kimliği varsa *HÂKİM_ELEMAN* veya *DİĞER_HÂKİM_ELEMANI_BEKLE* durumuna geçer.

- *DİGER_HÂKİM_ELEMANI_BEKLE*: Bir düğüm her raundun başında bu durumdan *AĞIRLIK_ORANLARINI_BEKLE* 'ye geçiş yapar.

- *BAĞLAN_BEKLE*: Bir düğüm her raundun başında bu durumdan *AĞIRLIK_ORANLARINI_BEKLE*'ye geçiş yapar veya *BAĞLAN* mesajı alarak *SIRADAN_ELEMAN* olur.
- *SIRADAN_ELEMAN*: *DOMINATEE* durumundaki bir düğümün eğer komşularından en az bir tanesi *HÂKİM_ELEMAN* olmuşsa ve diğer komşuları da *HÂKİM_ELEMAN* veya bir hâkim elemana bağlandıysa, *SIRADAN_ELEMAN* durumundaki düğüm *SIRADAN_ELEMAN_SON* durumuna geçiş yapar.

Yukarda görüldüğü üzere *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN_SON* dışındaki düğümler bu durumlarda kalamaz. *HÂKİM_ELEMAN* olacak düğümler sırayla *AĞIRLIK_ORANLARINI_BEKLE*, *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* ve *HÂKİM_ELEMAN* durumlarına geçerler. *SIRADAN_ELEMAN_SON* olacak düğümler *AĞIRLIK_ORANLARINI_BEKLE*, *BAĞLANMA_İSTEĞİ_BEKLE*, *BAĞLAN_BEKLE*, *SIRADAN_ELEMAN*, *SIRADAN_ELEMAN_SON* durumlarına geçerler. *HÂKİM_ELEMAN* veya *SIRADAN_ELEMAN_SON* dışında kalan düğümler için algoritma yeni rauntta tekrardan başlar. *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN_SON* durumlarından da diğer durumlara geçiş SSET'in sonlu durum makinesine göre yoktur. Varsayımız ile çelişiyoruz, bundan dolayı önermemiz doğrudur. ■

Teorem 4.3. SSET algoritması, kilitlenme ve açlık içermez.

İspat. Tersinin doğru olduğunu farz edelim. SSET algoritmasını çalıştıran bir düğüm algoritma sonlandığında *SIRADAN_ELEMAN_SON* veya *HÂKİM_ELEMAN* dışında bir durumda başka bir düğümden mesaj bekliyor olması gereklidir, fakat Ön Teorem 4.5'e göre bu durum mümkün değildir, varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Ön Teorem 4.6. ASYNSET algoritması her rauntta 2 zıplama uzaklığında en küçük ağırlık oranına sahip olan düğümleri hâkim eleman olarak seçer.

İspat. Bu teoremin ispatı Ön Teorem 4.3 ile aynıdır çünkü ASYNSET'in sonlu durum makinesi üzerinde *HÂKİM_ELEMAN* seçen geçişleri SSET ile aynıdır.■

Ön Teorem 4.7. ASYNSET algoritması her rauntta *HÂKİM_ELEMAN* düğümler seçildikten ve komşularını bu durumdan haberdar ettikten sonra yeni raunt başlar.

İspat. Ön Teorem 4.6'nın ispatında verildiği üzere ASYNSET algoritmasında iki zıplama uzağındaki düğümlerin bilgisini *AĞIRLIK_ORANI* ve *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajlarıyla paylaşan düğüm senkronize olarak *BAĞLAN_BEKLE* durumuna geçer. Bu düğüm senkronize olmasına rağmen komşularından biri *HÂKİM_ELEMAN* olursa *BAĞLAN* mesajını gönderdikten sonra senkronize olur. Mesajların alınmasını gönderme sırasına göre olduğunu kabul edilirse, raunt bitmeden *BAĞLAN_BEKLE* durumundaki düğüm *BAĞLAN* mesajını alıp haberdar olur.

Bir sonraki raundun başlaması için *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* durumundan *HÂKİM_ELEMAN* durumuna geçen düğümler, *BAĞLANMA_İSTEĞİ_BEKLE* durumundan *BAĞLAN_BEKLE* durumuna geçen düğümler, raundun başında *HÂKİM_ELEMAN* veya *SIRADAN_ELEMAN_SON* durumunda olan düğümler işlerini bitirdikleri için *senkronizeEt* prosedürünü çağırırlar. Bu durumların dışındaki düğümlerin hepsi ara durumdadırlar. ■

Ön Teorem 4.8. ASYNSET algoritmasında *senkronizeEt* prosedürüyle ve *TAMAM* mesajlarının işlenmesiyle senkronizasyon sağlanır.

İspat. Tersinin doğru olduğunu kabul edelim. Böylece aşağıdaki durumlardan herhangi birinin doğru olması gerekir:

- *SenkronizeEt* prosedürünü çağıran bir düğümün çocuklarından *TAMAM* mesajlarını beklemeden senkronize olabildiğini düşünelim. Hâlbuki ASYNSET algoritmasının *senkronizeEt* prosedürünü incelediğimizde sadece bu ağaç üzerinde *YAPRAK* durumundaki düğümlerin çocuklarından *TAMAM* mesajı almadan *TAMAM* gönderdiklerini görüyoruz. *YAPRAK* durumundaki düğümlerin çocuğu olmadığı için bu önermemiz doğru olamaz.

- Düğümlerin *TAMAM* mesajlarını işlerken *senkronizeEt* prosedürünü beklemeden senkronize olduklarını düşünelim. *TAMAM* mesajlarının

işlendiği sonlu durum makinesinde *senkronizeEt* prosedürünü çağırıp, *senkronizeOldu* değişkeni DOĞRU değerine sahip olmadan ebeveynine *TAMAM* mesajı iletemez, *ÇIKIŞ_DÜĞÜMÜ* yeni raundu başlatamaz. Bu önermemiz de doğru olamaz.

- Düğümlerin ASYNSET algoritmasının sonlu durum makinesinde ara durumlardayken *senkronizeEt* prosedürünü çağırdığını düşünelim. Bu önermenin de doğru olmadığını Ön Teorem 4.7'nin ispatında görebiliriz.

Yukarda sıraladığımız durumların hiçbirinin olma ihtimali yoktur, algoritmamız senkronizasyon işlemini doğru yapar. Varsayımımız ile çelişiyoruz. Teoremimiz doğrudur. ■

Ön Teorem 4.9. ASYNSET algoritması bir raunt içinde hâkim eleman seçimi yaptıktan sonra ağırlık oranlarını bir sonraki rauntta günceller.

İspat. Bir düğümün Ön Teorem 4.6'a göre bir raunt sonunda olabileceği durumlar *HÂKİM_ELEMAN*, *DİGER_HÂKİM_ELEMANI_BEKLE*, *SIRADAN_ELEMAN*, *SIRADAN_ELEMAN_SON* ve *BAĞLAN_BEKLE*'dir. ASYNSET algoritmasının sonlu durum makinesine göre *DİĞER_HÂKİM_ELEMANI_BEKLE* ve *BAĞLAN_BEKLE* durumundaki düğümler raunt başlarken *BAĞLANMADI* mesajı gönderirler, *SIRADAN_ELEMAN* durumundaki düğümlerse *BAĞLANDI* mesajı gönderip komşularının ağrılık oranlarını güncellemesini sağlarlar. *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN_SON* durumlarındaki düğümlerin komşularına ağırlıklarını güncellemesi için herhangi bir mesaj göndermesine gerek yoktur. ■

Teorem 4.4. ASYNSET algoritması, CENTSET ile aynı düğümleri *HÂKİM_ELEMAN* olarak seçer.

İspat. ASYNSET algoritmasının küme örtüsü tabanlı hâkim küme algoritmasıyla aynı düğümleri seçmediğini varsayalım. Böylece aşağıda listelediğimiz durumlardan en az bir tanesinin doğru olması gerekir:

1. ASYNSET algoritması, 2 zıplama uzaklığında en küçük ağırlık oranına sahip herhangi bir düğümü hâkim eleman olarak seçmez veya seçilen *HÂKİM_ELEMAN* düğümler komşularını bu durumdan haberdar etmez. Bu önermenin doğru olmadığını Ön Teorem 4.6 ve Ön Teorem 4.7'de gösterdik.

2. ASYNSET algoritmasında düğümler raundu bitirirken senkronizasyonu bozan en az bir düğüm vardır. Bu önermenin doğru olmadığını Ön Teorem 4.8'de gösterdik.

3. ASYNSET algoritmasında düğüm ağırlıkları güncellenmez. Bu önermenin doğru olmadığını Ön Teorem 4.9'de gösterdik.

Yukarıdaki önermeler doğru olmadığı için varsayımız ile çelişiyoruz, bundan dolayı teoremimiz doğrudur. ■

Sonuç Teorem 4.2. ASYNSET algoritmasının en küçük ağırlıklı bağlı hâkim kümenin ağırlığına yaklaşım oranı ln (*S*)'dir.

İspat. Teorem 4.1'e göre ASYNSET algoritmasıyla küme örtüsü tabanlı hâkim küme algoritmasıyla aynı düğümleri seçtiğinden dolayı yaklaşım oranı ln (*S*)'dir. ■

Ön Teorem 4.10. ASYNSET algoritması sonlandığında bir düğüm hâkim küme içinde veya hâkim küme içindeki bir elemana bağlanmış olur.

İspat. Bu teoremin ispatı SSET'e benzerdir. Ön Teorem 4.10 doğruysa Gözlem 4.2'e göre ASYNSET algoritması sonlandığında *HÂKİM_ELEMAN* veya *SIRADAN_ELEMAN_SON* durumlarında olabilir. Tersinin doğru olduğunu varsayalım. Bu durumda algoritma sonlandığında *BOŞ*, *AĞIRLIK_ORANLARINI_BEKLE*, *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE*, *BAĞLAN_BEKLE*, *DİGER_HÂKİM_ELEMANI_BEKLE*, *BAĞLANMA_İSTEĞİ_BEKLE,* *BAĞLAN_BİLGİSİ_BEKLE* veya *SIRADAN_ELEMAN* durumlarının herhangi birinde olabilir. Bu durumları aşağıda inceleyelim:

- *BOŞ*: Bir düğüm *ZAMAN_AŞIMI* olduktan sonra *AĞIRLIK_ORANI* mesajı gönderip *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçiş yapar. *BOŞ* durumundayken *AĞIRLIK_ORANI* mesajını alan düğüm *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçiş yapar. Düğüm *BOŞ* durumunda kalmaz.

- *AĞIRLIK_ORANLARINI_BEKLE*: Düğüm *HÂKİM_ELEMAN* veya *SIRADAN_ELEMAN_SON* durumu dışındaki tüm komşularından *AĞIRLIK_ORANI* mesajını aldıktan sonra bu durumdan geçiş yapar. Belirtilen bu durumlar dışındaki her düğüm bir raundun başında *AĞIRLIK_ORANI* mesajı gönderir. Böylelikle herhangi bir düğüm bu konumda kalmaz.

- *BAĞLANMA_İSTEĞİ_BEKLE*: Komşu düğümler *ADAY_DEĞİLİM* veya *BAĞLANMA_İSTEĞİ* mesajı göndereceğinden dolayı düğüm bu konumda kalmaz, *BAĞLAN_BEKLE* durumuna geçiş yapar.

- *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE*: Komşu düğümler *ADAY_DEĞİLİM* veya *BAĞLANMA_İSTEĞİ* mesajı göndereceğinden dolayı düğüm bu konumda kalmaz, *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajlarının hepsinin içinde kendi kimliği varsa *HÂKİM_ELEMAN* veya *DİĞER_HÂKİM_ELEMANI_BEKLE* durumuna geçer.

- *DİGER_HÂKİM_ELEMANI_BEKLE*: Bir düğüm her raundun başında bu durumdan *BAŞLA* mesajını alarak *AĞIRLIK_ORANLARINI_BEKLE* 'ye geçiş yapar.

- *BAĞLAN_BEKLE*: Bir düğüm her raundun başında *BAŞLA* mesajını alarak bu durumdan *AĞIRLIK_ORANLARINI_BEKLE*'ye geçiş yapar veya *BAĞLAN* mesajı alarak *SIRADAN_ELEMAN* olur.

- *SIRADAN_ELEMAN*: *SIRADAN_ELEMAN* durumumdaki düğüm her raundun başında *BAŞLA* mesajını alarak *BAĞLAN_BİLGİSİ_BEKLE* durumuna geçiş yapar. *SIRADAN_ELEMAN* durumundaki bir düğümün eğer komşularından en az bir tanesi *HÂKİM_ELEMAN* olmuşsa ve diğer komşuları da *HÂKİM_ELEMAN* veya bir hâkim elemana bağlanmışsa, *SIRADAN_ELEMAN* durumundaki düğüm *SIRADAN_ELEMAN_SON* durumuna geçiş yapar.

- *BAĞLAN_BİLGİSİ_BEKLE*: Bir düğüm *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN_SON* durumlarındaki düğümlerden *BAĞLANDI* veya *BAĞLANMADI* mesajlarını topladıktan sonra *AĞIRLIK_ORANLARINI_BEKLE* durumuna geçiş yapar. Düğüm bu durumda kalmaz.

Yukarda görüldüğü üzere *HÂKİM_ELEMAN* ve *SIRADAN_ELEMAN_SON* dışındaki düğümler bu durumlarda kalamaz. *HÂKİM_ELEMAN* olacak düğümler sırayla *AĞIRLIK_ORANLARINI_BEKLE*, *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ_BEKLE* ve *HÂKİM_ELEMAN* durumlarına geçerler. *SIRADAN_ELEMAN_SON* olacak düğümler *BAĞLAN_BİLGİSİ_BEKLE*, *AĞIRLIK_ORANLARINI_BEKLE*, *BAĞLANMA_İSTEĞİ_BEKLE*, *BAĞLAN_BEKLE*, *SIRADAN_ELEMAN*, *SIRADAN_ELEMAN_SON* durumlarına geçeler. *HÂKİM_ELEMAN* veya *SIRADAN_ELEMAN_SON* dışında kalan düğümler için algoritma yeni rauntta tekrardan başlar. ■

Teorem 4.5. ASYNSET algoritması kilitlenme ve açlık içermez.

İspat. Bu teoremin ispatı SSET ile benzerdir. Tersinin doğru olduğunu farz edelim. ASYNSET algoritmasını çalıştıran bir düğüm algoritma sonlandığında *SIRADAN_ELEMAN_SON* veya *HÂKİM_ELEMAN* dışında bir durumda başka bir düğümden mesaj bekliyor olması gereklidir, fakat Ön Teorem 4.10'a göre bu durum mümkün değildir, varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Teorem 4.6. FLOODTREE algoritması CENTREE algoritmasıyla aynı düğümleri bağlayıcı olarak seçer.

İspat. FLOODTREE Algoritmasının CENTREE ile aynı düğümleri seçmediğini varsayalım. Bu durumda aşağıdaki önermelerden en az birinin doğru olması gerekir:

- Algoritmanın başında ağaçlar oluşmaz. Bu önerme doğru değildir çünkü birinci rauntta ağaçlar oluşturulur.

- Yeni oluşan ağaç bilgisi aday düğümlere ulaşmaz. Bu önerme doğru değildir çünkü ikinci rauntta ağaç bilgileri aday düğümlere ulaşır.

- Düğümler birbirlerinin ağırlık oranını öğrenemez. Bu önerme doğru değildir çünkü birinci raunt dışındaki tek numaralı rauntlarda düğümler birbirlerinin ağırlık oranlarını öğrenir.

- Bağlayıcı düğüm seçildikten sonra ağaçlar birleşmez, ağaç bilgileri güncellenmez. Bu önerme de doğru değildir ikinci raunt dışındaki çift numaralı rauntlarda ağaçlar bağlayıcı üzerinden birleştirilir ve ağırlık oranları güncellenir.

FLOODTREE algoritması, CENTREE algoritmasıyla aynı düğümleri bağlayıcı olarak seçer. Varsayımımız ile çelişiyoruz. Teoremimiz doğrudur. ■

Ön Teorem 4.11. STREE Algoritmasının ilk raundunun sonunda hâkim küme düğümlerinden birbirleri arasında patika olanlar aynı ağacın içine dâhil olurlar.

İspat. Tersinin doğru olduğunu varsayalım. Bu durumda hâkim küme düğümlerinden birbirleri arasında patika olanlar aynı ağaca dâhil olmaz. Hâlbuki hâkim küme içindeki her düğüm *AĞAÇ_KUR* mesajını gönderip, kendine gelen *AĞAÇ_KUR* mesajını iletir. Raundun sonunda kimliği en küçük düğüm ağacın

kimliği olacak şekilde yönlü en küçük kapsayan ağaç algoritması çalıştırır. Varsayımımızın doğru olabilmesi için düğümlerin arasında patika olması gerekir ki bu durumda *AĞAÇ_KUR* mesajı iletilir, varsayımımız ile çelişiyoruz, teorem doğrudur. ■

Ön Teorem 4.12. STREE Algoritmasının ikinci raundunun sonunda *ÇIKIŞ_DÜĞÜMÜ* kök düğüm olacak şekilde ağaç oluşur.

İspat. Tersinin doğru olduğunu varsayalım. Bu durumda herhangi bir düğümün ebeveyninin veya çocuklarını bilmiyor olması gerekir, hâlbuki her düğüm en az bir kez *SENK_AĞACI_KUR* mesajını gönderir. *SENK_AĞACI_KUR* mesajını ilk kez alan bir düğüm ebeveynini bulmuş olur, daha önceden almış bir düğüm eğer ebeveyni kendiyse çocuğunu öğrenir. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Ön Teorem 4.13. STREE'nin bağlayıcı seçen ilk raundunda *HELMAN* düğümler ağaç kimliklerini ve ağaçların içinde kaç eleman olduğunu gönderirler.

İspat. STREE'nin bağlayıcı seçen ilk raundunun sonlu durum makinesinde görüldüğü üzere ilk rauntta ağacının kimliğini ve ağacının içindeki eleman sayısını bilen *LDR_HELMAN* durumundaki düğümde *YENİ_RNT* kesmesi olduktan sonra ağaç bilgilerini *DRM* mesajı içinde gönderir ve *LDR_HELMAN_İST_BKL* durumuna geçeri. Benzer olarak ilk rauntta ağaç bilgilerini bilen *HELMAN* durumundaki düğümde *YENİ_RNT* kesmesi olduktan sonra ağaç bilgilerini *DRM* mesajı içinde gönderir ve *HELMAN_İST_BKL* durumuna geçer.■

Ön Teorem 4.14. STREE'nin bağlayıcı seçen ilk raundunda *SELMAN* düğümler *HELMAN* düğümlere bağlanma isteklerini ağırlık oranlarıyla birlikte gönderirler.

İspat. Ön Teorem 4.13'e göre *HELMAN* düğümlerin *DRM* mesajlarını gönderir. STREE'nin bağlayıcı seçen ilk raundunun sonlu durum makinesinde görüldüğü üzere *SELMAN* düğümler *DRM* mesajlarını topladıktan sonra *İST* göndermeden önce *SELMAN_DRM* mesajlarını toplar. *DRM* ve *SELMAN_DRM* mesajlarını toplayan *SELMAN* düğümler bağlanma isteklerini *İST* mesajı içinde ağırlık oranlarıyla göndererek *BĞLAN_BKL* durumuna geçerler. ■

Ön Teorem 4.15. STREE'nin bağlayıcı seçen ilk raundunda her ağaç çevresindeki en küçük ağırlıklı *SELMAN* düğümü bulur.

İspat. STREE'nin bağlayıcı seçen ilk raundunun sonlu durum makinesinde görüldüğü üzere *DRM* mesajlarını gönderen *HELMAN* düğümler *HELMAN_İST_BKL* veya *LDR_HELMAN_İST_BKL* durumuna geçerler. Bu durumlardayken ağaçlarındaki tüm çocuklarından ve tüm *SELMAN* komşularından *İST* alan *HELMAN* düğüm ağaç üzerinden ebeveynlerine en küçük ağırlığı *İST* mesajı içinde iletir ve *HELMAN_KYNŞTR_BEKLE* durumuna geçer. *LDR_HELMAN_İST_BKL* durumundaki düğüm gerekli *İST* mesajlarını topladıktan sonra en küçük ağırlıklı *SELMAN*'a *BĞLAN* mesajını ağacın kökünden yapraklarına doğru iletip *LDR_HELMAN_KYNŞTR_BKL* durumuna geçer. Böylece her ağaç çevresindeki en küçük ağırlıklı *SELMAN* düğümü bulmuş olur. ■

Ön Teorem 4.16. STREE'nin bağlayıcı seçen ilk rauntta bir *SELMAN* düğüm bağlı olduğu ağaçların hepsinde en az ağırlık oranına sahipse *BĞLAN* mesajı alır.

İspat. *SELMAN* düğümler *HELMAN* düğümlerden cevap bekledikleri sırada Ön Teorem 4.14'de gösterildiği üzere *BĞLAN_BKL* durumundadırlar. *BĞLAN_BKL* durumundaki bir düğüm *BĞLAN* mesajını mutlaka alır, çünkü Ön Teorem 4.15'de gösterildiği üzere *HELMAN* ağacının kökünden yapraklarına doğru *BĞLAN* mesajı iletilir ve bu iletim sırasında *SELMAN* düğümler bu ağacın en az bir elemanına komşu olduğu için *BĞLAN* mesajını alır. STREE'nin bağlayıcı seçen sonlu durum makinesinde görüldüğü üzere *BĞLAN* mesajının içinde kendi kimliği varsa bu mesajı bağlanma onayı olarak alıp *BĞLAN_BKL* durumundan *KYNŞTR_BKL* durumuna geçer, aksi durumda *BĞLAN_BKL* durumundan *SELMAN* durumuna geçer. ■

Ön Teorem 4.17. STREE'nin bağlayıcı seçen ilk raundunda bir *SELMAN* düğümün eğer bağlanma isteğinde bulunduğu ağaçların üzerinde ağırlık oranı en küçükse raundun sonunda çevresindeki tüm ağaçlardan onaylanır ve aday bağlayıcı düğüm olur.

İspat. Tersinin doğru olduğunu varsayalım. Bu durumda aşağıdaki önermelerden en az birinin doğru olması gerekmektedir:

- *HELMAN* düğümler algoritma başında hangi ağaca bağlı olduklarını ve ağaçta kaç adet *HELMAN* olduğunu yanlış bilebilir. Bu önermenin doğru olmadığı Ön Teorem 4.11'de gösterildi.

- *HELMAN* düğümler ağaç kimliklerini ve ağaç üzerinde kaç adet düğüm olduğunu göndermeyebilir. Bu önermenin yanlış olduğu Ön Teorem 4.13'de verilmiştir.

- *SELMAN* düğümler ağaçlara bağlanma isteği göndermez. Bu önermenin yanlış olduğu Ön Teorem 4.14'de verilmiştir.

- Ağaçlar kendilerine bağlı *HELMAN* düğümler içinde en küçük ağırlık oranına sahip olanı bulamaz. Bu önermenin yanlış olduğu Ön Teorem 4.15'de verilmiştir.

- *SELMAN* düğüm çevresindeki ağaçlar içinde en küçük ağırlığa sahipse bağlanma onayı alamaz. Bu önermenin yanlış olduğu Ön Teorem 4.16'da verilmiştir.

Yukarıda listeciğimiz önermelerin hepsi yanlıştır. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Ön Teorem 4.18. STREE Algoritmasında bağlayıcı seçen ikinci raundun sonunda *ÇIKIŞ_DÜĞÜMÜ'*ne doğru oluşturulan ağaç üzerinden ağırlık oranları iletilir.

İspat. Ön Teorem 4.12'de ağacın doğru kurulduğu gösterilmiştir. Her düğüm kendindeki ve çocuklarından gelen ağırlık oranını ebeveynine iletir. ■

Ön Teorem 4.19. STREE Algoritmasında bağlayıcı seçen üçüncü raundun sonunda bağlayıcı seçilir veya düğümler algoritmayı sonlandırır.

İspat. Ön Teorem 4.18'e göre ağırlık oranları ağaç üzerinden iletilir. *ÇIKIŞ_DÜĞÜMÜ* en küçük ağırlık oranına sahip düğüme *BĞLAN_İST_ONAY* mesajı gönderir. *BĞLAN_İST_ONAY* mesajını alan aday bağlayıcı düğüm bağlayıcı seçen ilk raundun sonlu durum makinesine gere *BĞLAN_İST_ONAY_BKL* durumunda olduğundan *HELMAN* veya *LDR_HELMAN* durumuna geçiş yapar ve *AĞAÇ_KYNŞTR* mesajını gönderir. *AĞAÇ_KYNŞTR* mesajını alan ve bir ağacın elemanı olan düğüm *LDR_HELMAN_KYNŞTR_BKL* veya *HELMAN_KYNŞTR_BKL* durumunda olabilir. Bu mesajı alan *HELMAN* düğüm komşularına iletir. Eğer aday düğüm yoksa *ÇIKIŞ_DÜĞÜMÜ* tüm düğümlere *SON* mesajı gönderir ve düğümler algoritmayı bitirir. ■

Teorem 4.7. STREE Algoritması, CENTREE algoritmasıyla kenar ağırlıkları eşit alındığında aynı düğümleri bağlayıcı olarak seçer.

İspat. İlk raundun doğruluğu Ön Teorem 4.11'de, ikinci raundun doğruluğu Ön Teorem 4.12'de, bağlayıcı seçen ilk raundun doğruluğu Ön Teorem 4.17, bağlayıcı seçen ikinci raundun doğruluğu Ön Teorem 4.18 ve bağlayıcı seçen üçüncü raundun doğruluğu Ön Teorem 4.19'de verilmiştir. Bu rauntlar sırayla çalıştığında en küçük

ağırlığa sahip düğüm bağlayıcı olarak seçilir. STREE algoritmasına göre hâkim düğümler aynı ağaç üzerinden olmadığı sürece bağlayıcı seçen ilk raunt (Raunt 3)'e dönülür ve böylece işlem tüm bağlayıcılar seçilene kadar algoritma devam eder. ■

Sonuç Teoremi 4.3. STREE Algoritmasının yaklaşım oranı 2 ln (*S*)' dir.

İspat. Teorem 4.7'ye göre STREE, CENTREE'yle aynı düğümleri bağlayıcı olarak seçer. CENTREE algoritması 2 ln (*S*) yaklaşımlı olduğu için STREE aynı yaklaşıma oranına sahiptir. ■

Gözlem 4.3. DTOR_TREE algoritması sonlandıktan sonra *HELMAN_LDR_SON* durumundaki düğüm ağacın lideri, *HELMAN_ÜYE_SON* durumundaki düğüm ağacın üyesidir.

Ön Teorem 4.20. DTOR_TREE algoritmasını çalıştıran bir *HELMAN* düğümün *HELMAN* komşusu yoksa tek başına ağaç oluşturur.

İspat. DTOR_TREE algoritmasının sonlu durum makinesinde görüldüğü üzere her *HELMAN* düğüm algoritmaya *HELMAN_BOŞ* durumunda başlar, *HELMAN* komşusu yoksa *HELMAN_LDR_SON* durumuna geçer ve senkronize olur. ■

Ön Teorem 4.21. DTOR_TREE algoritmasında her *HELMAN* düğüm kendinden küçük kimlikli *HELMAN* düğüm bulana kadar ilk iterasyonda *İST* diğer iterasyonlarda *ARA* göndermeye devam eder, bulamazsa sonlanır ve lider olur.

İspat. DTOR_TREE algoritmasının sonlu durum makinesinden görülebileceği üzere eğer *HELMAN* düğüm komşuları içinde en küçük kimliğe sahip değilse *HELMAN_BOŞ* durumundan *HELMAN_ÜYE* durumuna geçiş yapar ve aramayı bırakır. Eğer komşuları içinde en küçük kimliğe sahipse ilk rauntta *HELMAN_BOŞ* durumundan *HELMAN_LDR_BKL_DRM* durumuna *İST* mesajı göndererek geçer ve daha sonraki iterasyonlarda *HELMAN_LDR_BKL_DRM* durumu üzerinde dönmeye devam ederek *ARA* mesajı gönderir. Bulamazsa *HELMAN_LDR_BKL_DRM* durumundan *HELMAN_LDR_SON* durumuna gelerek algoritmanın işlenişini sonlandırır. ■

Ön Teorem 4.22. DTOR_TREE algoritmasında her *HELMAN* düğüm bulduğu en küçük kimlikli *HELMAN*'dan eşit veya küçük kimlikli *HELMAN* düğümden aldığı *İST* mesajına *DRM* ile cevap verir, *ARA* mesajını *İST* olarak komşularına iletir.

İspat. DTOR_TREE algoritmasının sonlu durum makinesinden görüleceği üzere *İST* alan düğüm *HELMAN_ÜYE*, *HELMAN_DRM_BKL*, veya *HELMAN_BOŞ*

durumlarında olabilir. Bu durumlardan *HELMAN_ÜYE*'ye geçiş yapar. Bir düğümün *ARA* alabilmesi için öncelikle *İST* alması gereklidir, *İST* aldıktan sonra da *HELMAN_ÜYE* olur. *HELMAN_ÜYE* durumundaki düğüm *ARA* mesajını *İST* olarak komşularına iletir. ■

Ön Teorem 4.23. DTOR_TREE algoritmasında düğümler senkronize olduktan sonra mutlaka *HELMAN_LDR_SON* veya *HELMAN_ÜYE_SON* durumuna geçer.

İspat. DTOR_TREE algoritmasının sonlu durum makinesinden görüleceği üzere sonlu durum makinesinde senkronize olmayı sağlayan 3 geçiş vardır. Bu geçişler *LDR_DRM_BKL* durumundan *HELMAN_LDR_SON* durumuna geçiş, *HELMAN_BOŞ* durumundan *HELMAN_LDR_SON* durumuna geçiş ve *HELMAN_ÜYE* durumundan *HELMAN_ÜYE_SON* durumuna geçiştir. ■

Teorem 4.8. DTOR_TREE algoritması tamamlandığında birbirleri arasında patika olan *HELMAN* düğümler arasında öyle bir ağaç oluşur ki; bu ağacın tek bir lideri vardır, her düğüm bu ağacın kimliğini, ağaç üzerinde kaç adet düğüm olduğunu ve ağaç üzerindeki ebeveynini bilir.

İspat. Tersinin doğru olduğunu varsayalım. Bu durumda aşağıdaki önermelerden en az birinin doğru olması gerekir:

- Lider *HELMAN* düğümler kendilerinden küçük kimlikli *HELMAN* düğüm buluna kadar aramaya devam etmezler ve ağaç oluştuktan sonra bir lider düğüm bulunamaz. Bu önermenin doğru olmadığı Ön Teorem 4.20 ve Ön Teorem 4.21'de gösterilmiştir.

- Üye *HELMAN* düğümler lider *HELMAN* düğümler tarafından gelen bütün aramalara kayıtsız kalabilirler. Bu önermenin doğru olmadığı Ön Teorem 4.22'de gösterilmiştir.

- Senkronizasyon işlemi hatalı olabilir. Bu önermenin doğru olmadığı Ön Teorem 4.23'da gösterilmiştir.

- Düğümler ağacın kimliğini, üzerinde kaç düğüm olduğunu ebeveynlerini ve çocuklarını doğru bulmayabilir. Bu önermenin doğru olmadığı 1. durumda gösterilen tek lider olmasıyla ve DTOR_TREE algoritmasının sonlu durum makinesinde lider düğümün *SON* mesajını üyelerine yaymasıyla gösterilmiştir.

Böylece algoritma sonlandığında her ağaçta *HELMAN_LDR_SON* durumunda bir düğüm olup, ağaçtaki diğer düğümler *HELMAN_ÜYE_SON* durumundaki üyelerdir. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur. ■

Sonuç Teoremi 4.4. DTOR_TREE algoritması, kilitlenme ve açlık içermez.

İspat. Tersinin doğru olduğunu varsayalım. DTOR_TREE algoritmasını çalıştıran bir düğüm algoritma sonlandığında *HELMAN_LDR_SON* veya *HELMAN_ÜYE_SON* dışında bir durumda başka bir düğümden mesaj bekliyor olması gereklidir, fakat Teorem 4.8'e göre her düğüm bu durumlardan birinde algoritmayı sonlandırır. ■

Teorem 4.9. Düzenlenmiş GHS algoritmasının sonunda arasında patika olan *HELMAN* düğümler arasında öyle bir ağaç oluşur ki; bu ağacın tek bir lideri vardır, her düğüm bu ağacın kimliğini, ağaç üzerinde kaç adet düğüm olduğunu ve ağaç üzerindeki ebeveynini bilir.

İspat. Düzenlenmiş GHS algoritmasında çekirdek düğümler ağaçta kaç adet düğüm olduğunu *REPORT* mesajı içinde öğrenirler ve diğer düğümlere *SON* mesajı içinde gönderirler. GHS algoritmasında her düğüm ebeveynini *in-branch* değişkeniyle bilir, ayrıca *SON* mesajı yayılırken bu bilgi tekrardan sağlanabilir. Ağacın lideri ve kimliği çekirdek düğümler içindeki en küçük düğümün kimliğidir. ■

Ön Teorem 4.24. ASYNTREE'nin bağlayıcı seçen algoritmasında *HELMAN* düğümler her rauntta ağaç kimliklerini ve ağaçların içinde kaç eleman olduğunu gönderirler.

İspat. ASYNTREE'nin bağlayıcı seçen sonlu durum makinesinde görüldüğü üzere ilk rauntta ağacının kimliğini ve ağacının içindeki eleman sayısını bilen *LDR_HELMAN* durumundaki düğüm *BAŞLA* mesajını alarak ağaç bilgilerini *DRM* mesajı içinde gönderir ve *LDR_HELMAN_İST_BKL* durumuna geçeri. Benzer olarak ilk rauntta ağaç bilgilerini bilen *HELMAN* durumundaki düğüm *BAŞLA* mesajını alarak ağaç bilgilerini *DRM* mesajı içinde gönderir ve *HELMAN_İST_BKL* durumuna geçer. Diğer rauntlarda *LDR_HELMAN_KYNŞTR_BKL* durumundaki düğüm *AĞAÇ_KYNŞTR* mesajı içinde ağaç bilgilerini alarak *LDR_HELMAN_İST_BKL* durumuna geçtikten sonra ağaç bilgilerini *DRM* mesajı içinde gönderir. Diğer rauntlarda *HELMAN_KYNŞTR_BKL* durumundaki düğüm *AĞAÇ_KYNŞTR* mesajı içinde ağaç bilgilerini alarak *HELMAN_İST_BKL* durumuna geçtikten sonra ağaç bilgilerini *DRM* mesajı içinde gönderir. Benzer olarak diğer rauntlarda *KYNŞTR_BKL* durumunda olan yeni seçilmiş *HELMAN* düğüm

AĞAÇ_KYNŞTR mesajı içinde ağaç bilgilerini aldıktan sonra *DRM* mesajıyla bu bilgileri gönderir. Diğer rauntlara *HELMAN* düğümlerin *HELMAN_KYNŞTR_BKL*, *LDR_HELMAN_KYNŞTR_BKL* veya *KYNŞTR_BKL* durumlarından birinde başlamaları Ön Teorem 4.26 ve Ön Teorem 4.28'de ispatlanmıştır. ■

Ön Teorem 4.25. ASYNTREE'nin bağlayıcı seçen algoritmasında *SELMAN* düğümler *HELMAN* düğümlere bağlanma isteklerini ağırlık oranlarıyla birlikte gönderirler.

İspat. Bu teoremin ispatı ASYNTREE'nin sonlu durum makinesi üzerinden gidilerek Ön Teorem 4.14 ile aynıdır.■

Ön Teorem 4.26. ASYNTREE'nin bağlayıcı seçen algoritmasında her ağaç çevresindeki en küçük ağırlıklı *SELMAN* düğümü bulur.

İspat. Bu teoremin ispatı ASYNTREE'nin sonlu durum makinesi üzerinden gidilerek Ön Teorem 4.15 ile aynıdır. ■

Ön Teorem 4.27. ASYNTREE'nin bağlayıcı seçen algoritmasında her *SELMAN* düğüm bağlanma isteğinde bulunduğu ağaçtan olumlu veya olumsuz cevap alır.

İspat. *SELMAN* düğümler *HELMAN* düğümlerden cevap bekledikleri sırada Ön Teorem 4.25'de gösterildiği üzere *BĞLAN_BKL* durumundadırlar. *BĞLAN_BKL* durumundaki bir düğüm *BĞLAN* mesajını mutlaka alır, çünkü Ön Teorem 4.26'da gösterildiği üzere *HELMAN* ağacının kökünden yapraklarına doğru *BĞLAN* mesajı iletilir ve bu iletim sırasında *SELMAN* düğümler bu ağacın en az bir elemanına komşu olduğu için *BĞLAN* mesajını alır. ASYNTREE'nin bağlayıcı seçen sonlu durum makinesinde görüldüğü üzere *BĞLAN* mesajının içinde kendi kimliği varsa bu mesajı bağlanma onayı olarak alıp *BĞLAN_BKL* durumundan *KYNŞTR_BKL* durumuna geçer, aksi durumda *BĞLAN_BKL* durumundan *SELMAN* durumuna geçer. ■

Ön Teorem 4.28. ASYNTREE'nin bağlayıcı seçen algoritmasında düğümler işlemlerini bitirmeden önce senkronize olmazlar.

İspat. *HELMAN* düğümler senkronize olmadan önce *BĞLAN* mesajını göndermesi, *SELMAN* düğümler de senkronize olmadan önce topladıkları *BĞLAN* mesajlarına göre aday olup olmadığını belirlemesi gerekir. ASYNTREE'nin bağlayıcı seçen sonlu durum makinesinde görüldüğü üzere senkronize olunan durumlar şunlardır:

- *LDR_HELMAN_İST_BKL* durumundaki *HELMAN* düğüm gerekli bütün *İST* mesajları topladıktan sonra *BĞLAN* mesajını gönderip, *LDR_HELMAN_KYNŞTR_BKL* durumuna geçer ve senkronize olur.

- *HELMAN_İST_BKL* durumundaki *HELMAN* düğüm *BĞLAN* mesajını ilettikten sonra senkronize olur.

- *BĞLAN_BKL* durumundaki *SELMAN* düğüm hedefi kendisi olmayan bir *BĞLAN* mesajı alırsa *SELMAN* durumuna geçer ve senkronize olur.

- *BĞLAN_BKL* durumundaki bir düğüm bütün ağaçlarından hedefi kendisi olan bir *BĞLAN* mesajı topladığında *KYNŞTR_BKL* durumuna geçer ve senkronize olur.

Bu durumların hepsi senkronizasyon kuralına uygundur. ■

Ön Teorem 4.29. ASYNTREE'nin bağlayıcı seçen algoritmasında her rauntta *ÇIKIŞ_DÜĞÜMÜ* en küçük ağırlık oranına sahip olan aday bağlayıcı düğümü bulur. Bağlayıcı aday yoksa algoritma sonlanır.

İspat. Ön Teorem 4.28'e göre bağlayıcı aday düğümler ağırlık oranlarını bulduktan sonra senkronizasyon ağacı üzerinden bu bilgiyi gönderirler. ASYNTREE algoritmasının senkronizasyon mekanizmasına göre senkronizasyon ağacı üzerindeki *YAPRAK* düğümler buldukları ağırlık oranını *BĞLAN_İST* mesajı içinde ebeveynlerine iletirler. *YAPRAK* ve *ÇIKIŞ_DÜĞÜMÜ* olmayan düğümlerse *BĞLAN_İST* mesajlarını toplayıp *senkronizeEt* prosedürünü çağırdıktan sonra en küçük ağırlık oranını ve orana sahip düğümün kimliğini ebeveynlerine iletirler, bu iletim *ÇIKIŞ_DÜĞÜMÜ*'ne kadar devam eder. Böylelikle *ÇIKIŞ_DÜĞÜMÜ* en küçük ağırlık oranını bulur. Eğer aday bağlayıcı düğüm varsa *ÇIKIŞ_DÜĞÜMÜ AĞAÇ_KYNŞTR* mesajı içinde birleşecek ağaçların kimliği, yeni ağacın kimliği, ağaçtaki toplam düğüm sayısını gönderir. Eğer bağlayıcı aday düğüm yoksa *ÇIKIŞ_DÜĞÜMÜ SON* mesajını gönderir, *HELMAN* düğümler *HELMAN_KYNŞTR_BKL* veya *LDR_HELMAN_KYNŞTR_BKL* durumlarında olabileceklerinden dolayı *SON* mesajını alarak *HELMAN_SON* durumuna geçerler. *SELMAN* düğümler sadece *SELMAN* durumunda olabileceklerinden dolayı *SON* mesajını alıp *SELMAN_SON* durumuna geçerler. ■

Teorem 4.10. ASYNTREE'nin bağlayıcı seçen algoritmasında her rauntta en küçük ağırlıklı düğüm bağlayıcı olarak seçilir ve bütün *HELMAN* düğümler aynı ağacın içine alındığında algoritma sonlanır.

İspat. Tersinin doğru olduğunu varsayalım. Bu durumda aşağıdaki önermelerden en az birinin doğru olması gerekmektedir:

- *HELMAN* düğümler algoritma başında hangi ağaca bağlı olduklarını ve ağaçta kaç adet *HELMAN* olduğunu yanlış bilebilir. Bu önermenin doğru olmadığı Teorem 4.8'de DTOR_TREE algoritmasının doğruluk ispatında ve Teorem 4.9'da düzenlemiş GHS algoritmasının doğruluk ispatında verilmiştir.

- *HELMAN* düğümler ağaç kimliklerini ve ağaç üzerinde kaç adet düğüm olduğunu göndermeyebilir. Bu önermenin yanlış olduğu Ön Teorem 4.24'de verilmiştir.

- *SELMAN* düğümler ağaçlara bağlanma isteği göndermez. Bu önermenin yanlış olduğu Ön Teorem 4.25'de verilmiştir.

- Ağaçlar kendilerine bağlı *HELMAN* düğümler içinde en küçük ağırlık oranına sahip olanı bulamaz. Bu önermenin yanlış olduğu Ön Teorem 4.26'da verilmiştir.

- *SELMAN* düğümler bağlayıcı aday olup olmadıklarını bulamazlar. Bu önermenin yanlış olduğu Ön Teorem 4.27'de verilmiştir.

- Düğümler işlemlerini bitirmeden önce senkronize olurlar. Bu önermemin yanlış olduğu Ön Teorem 4.28'de verilmiştir.

- *ÇIKIŞ_DÜĞÜMÜ* en küçük ağırlıklı bağlayıcıyı bulamaz ve algoritmayı sonlandıramaz. Bu önermenin yanlış olduğu Ön Teorem 4.29'da verilmiştir.

Yukarıda listelediğimiz önermelerin hepsi yanlıştır. Varsayımımız ile çelişiyoruz, teoremimiz doğrudur, algoritmamız doğru çalışır. ■

4.3.2. Mesaj Karmaşıklığı

Ön Teorem 4.30. FLOODSET algoritmasında bir hâkim eleman seçildiği r. raunda en çok $(N\text{-}r\text{+}1)^2 + \Delta + 1$, en az $(N\text{-}r\text{+}1)^2 + 2$ mesaj gönderimi yapılır.

İspat. FLOODSET algoritması her rauntta bir hâkim eleman seçer, seçilen hâkim eleman bir sonraki rauntta mesaj gönderimi yapmaz. Böylece r. rauntta $N\text{-}r\text{+}1$ kadar düğüm birbirine *AĞIRLIK_ORANI* gönderir. Seçilen hâkim eleman

komşularına *HÂKİM_ELEMAN* mesajı gönderir, eğer komşularının hepsi örtülmemişse Δ kadar *ÖRTÜLDÜM* mesajı gönderilir. En iyi durumda en az bir komşu örtüleceği için bir adet *ÖRTÜLDÜM* mesajı gönderilir. Böylelikle *r*. rauntta en çok $(N\text{-}r+1)^2 + \Delta + 1$, en az $(N\text{-}r+1)^2 + 2$ mesaj gönderimi yapılır. ■

Teorem 4.11. FLOODSET algoritmasının mesaj karmaşıklığı *N* düğüm için HK hâkim küme olmak üzere $\Theta(N^2 |HK|)$'e, alt limitte $\Omega(N^2)$'e, üst limitte $O(N^3)$'e eşittir.

İspat. Ön Teorem 4.30'a göre algoritmanın *r*. rauntta en çok $(N\text{-}r+1)^2 + \Delta + 1$, en az $(N\text{-}r+1)^2 + 2$ mesaj gönderimi yapılır. FLOODSET algoritması her rauntta bir hâkim eleman seçer, $r = |HK|$ olmak üzere iyi durumda toplam olarak $\sum_{1}^{r=|HK|}((N-r+1)^2+2) \in \Omega(N^2|HK|)$, kötü durumda toplam olarak $\sum_{1}^{r=|HK|}((N-r+1)^2+\Delta+1) \in O(N^2|HK|)$ olduğundan dolayı FLOODSET algoritmasının mesaj karmaşıklığı $r=|HK|$ için $\Theta(N^2|HK|)$'e eşittir. FLOODSET algoritması en kötü durumda $r=N$ için $\sum_{1}^{r=N}((N-r+1)^2+\Delta+1) \in O(N^3)$'e eşittir. En iyi durumda $|HK|=1$ için $\Omega(N^2)$'e eşittir. ■

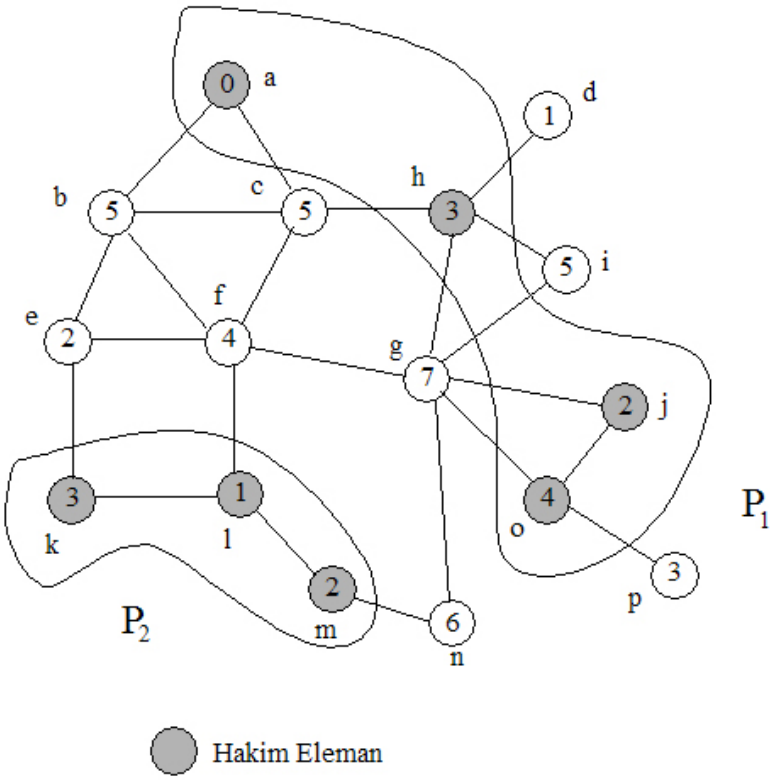
Tanım 4.1. P kümesi: SSET algoritması çalıştıktan sonra üretilen bir n_v hâkim elemanından itibaren başlayarak en fazla iki zıplamada bir hâkim eleman olacak şekilde ulaşılabilen tüm hâkim elemanların oluşturduğu küme olarak tanımlıyoruz. P_M, P kümeleri içinde en çok elemanı olan kümedir. Örneğin Şekil 4.30'de üzerinde SSET çalışan ağda üretilen P kümeleri, $P_1 = \{a, h, o, j\}$ ve $P_2 = \{ k, l , m\}$'dir. En çok elemanı olan küme P_1 olduğu için $P_1=P_M$.

Ön Teorem 4.31. SSET algoritmasının en çok süreceği raunt sayısı $r = |P_M|$.

İspat. Tanım 4.1'e göre $v_i \in P_i$ ve $v_j \in P_j$ olduğunda v_i ve v_j arasında en az 3 zıplama uzaklık vardır. Bundan dolayı en kötü durumda herhangi bir rauntta P kümelerinin her birinden eşzamanlı olarak birer adet hâkim eleman seçilebilir. En çok elemanlı P kümesi bitene kadar algoritma devam edebilir, bu yüzden $r= |P_M|$'dir. ■

Ön Teorem 4.32. SSET algoritmasındaki bir raundun mesaj karmaşıklığı $\Theta(N)$'e eşittir.

İspat. SSET'in sonlu duruma makinesi göre her düğüm *AĞIRLIK_ORANI* mesajını örtülmediyse gönderir. *ADAY_DEĞİLİM* veya *BAĞLANMA_İSTEĞİ* mesajlarından sadece bir tanesi gönderilir. *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* veya *BAĞLAN* mesajlarından da sadece bir tanesi gönderilir. *BAĞLANDI* mesajı da *BAĞLAN* mesajını alan düğümler tarafından gönderilir. Başka mesaj gönderimi yoktur. Böylece her düğüm bir raunt içinde maksimum 4, mesaj gönderimi yapar, N düğüm için mesaj karmaşıklığı $\Theta(N)$'e eşittir. ■



Şekil 4.30 P kümeleri

Teorem 4.12. SSET algoritmasının mesaj karmaşıklığı N düğüm için $\Theta(N\,|P_M|)$'e, üst limitte $O(N^2)$'ye, alt limitte $\Omega(N)$'e eşittir.

İspat. Ön Teorem 1.6'e göre algoritma $r=|P_M|$ raunt çalışacaktır. Ön Teorem 4.32'ya göre de her raundun mesaj karmaşıklığı $\Theta(N)$'dir. Böylece algoritmanın karmaşıklığı $\Theta(N\,|P_M|)$'e eşit olur. Alt sınırda $|P_M|=1$ olabilir ve alt limit $\Omega(N)$'e eşit olur. Üst limitte $|P_M|=N$ olabilir böylece $O(N^2)$'ye eşit olur. ■

Ön Teorem 4.33. ASYNSET algoritmasının en çok süreceği raunt sayısı $r = |P_M|$.

İspat. ASYNSET Algoritmasının çalışması SSET ile aynıdır, bundan dolayı en çok süreceği raunt süreceği raunt sayısı $r=|P_M|$. ■

Ön Teorem 4.34. ASYNSET algoritmasındaki bir raundun mesaj karmaşıklığı $\Theta(N)$'e eşittir.

İspat. Her düğüm raunda başlamak için *BAŞLA* mesajını iletir. ASYNSET'in sonlu duruma makinesi göre bir düğüm raundun başında *BAĞLANMADI* veya *BAĞLANDI* gönderebilir. Daha sonra ağırlık oranları hesaplandıktan sonra *AĞIRLIK_ORANI* mesajını gönderir. *ADAY_DEĞİLİM* veya *BAĞLANMA_İSTEĞİ* mesajlarından sadece bir tanesi gönderilir. *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* veya *BAĞLAN* mesajlarından da sadece bir tanesi gönderilir. Son olarak her düğüm raundun sonunda senkronize olmak için bir adet *TAMAM* mesajını ebeveynine gönderir. Başka mesaj gönderimi yoktur. Böylece her düğüm bir raunt içinde maksimum 6 mesaj gönderimi yapar, N düğüm için mesaj karmaşıklığı $\Theta(N)$'e eşittir. ■

Teorem 4.13. ASYNSET algoritmasının mesaj karmaşıklığı N düğüm için $\Theta(N|P_M|)$'e, üst limitte $O(N^2)$'ye, alt limitte $\Omega(N)$'e eşittir.

İspat. İspat Teorem 4.12 ile aynıdır, Ön Teorem 4.33 ve Ön Teorem 4.34'den gösterilir. ■

Ön Teorem 4.35. FLOODTREE Algoritmasının ilk raundunun mesaj karmaşıklığı N_d aralarında birbirleri üzerinden patika olan ve HK düğümlerinden oluşan ayrık kümeler olduğunda $\Theta(\sum|N_d{}^2| + |HK| - \sum|N_d|)$ olmak üzere alt sınırda $\Omega(|HK|)$'ya üst sınırda $O(N^2)$'ye eşittir.

İspat. FLOODTREE Algoritmasında ilk rauntta HK içindeki bir düğüm komşusuna *HÂKİM_ELEMAN* mesajı gönderir ve bu mesaj HK içindeki bir düğüm tarafından alınmışsa iletilmeye devam eder. Bu durumda birbirlerine kendi aralarında mesajları iletebilen HK düğümlerinden oluşan her bir N_d kümesinin mesaj karmaşıklığı $N_d{}^2$ 'ye eşittir ve toplamları $\sum|N_d{}^2|$ dir. Geriye kalan HK - $\sum|N_d|$ kadar düğüm sadece 1 tane *HÂKİM_ELEMAN* mesajı gönderir, bu durumda mesaj karmaşıklığı toplam olarak $\Theta(\sum|N_d{}^2| + |HK| - \sum|N_d|)$ ye eşit olur. Alt sınırda HK düğümlerinin hiçbiri birbirine komşu değilse mesaj karmaşıklığı bütün $|N_d| = 1$ olacağı için $\Omega(|HK|)$ ya eşittir. Üst sınırda $|N_d| = |HK| = N$ için $O(N^2)$ olur. ■

Ön Teorem 4.36. FLOODTREE Algoritmasının ikinci raundunun mesaj karmaşıklığı HK hâkim küme olduğunda $\Theta(N|HK|)$ olmak üzere alt sınırda $\Omega(N)$'ye üst sınırda $O(N^2)$'ye eşittir.

İspat. FLOODTREE Algoritmasında ikinci raundunda HK elemanları ağa hâkim eleman mesajını yayar. Böylece ağda toplam $N|HK| \in \Theta(N|HK|)$ kadar

mesaj gönderir. Alt sınırda |HK|=1 için $\Omega(N)$'ye üst sınırda |HK|=N $O(N^2)$'ye eşittir. ■

Ön Teorem 4.37. FLOODTREE algoritmasının bağlayıcı seçen birinci raundun dışındaki tek numaralı raundun algoritmanın sonundaki toplam mesaj karmaşıklığı N düğüm için B bağlayıcı sayısı olmak üzere $\Theta(N^2 B)$'e, alt limitte $\Omega(N^2)$'e, üst limitte $O(N^3)$'e eşittir.

İspat. Algoritmanın r. rauntta $(N\text{-}r+1)$ kadar düğüm ağa *AĞIRLIK_ORANI* yayar. FLOODTREE algoritması her rauntta bir hâkim eleman seçer, $r = B$ olmak üzere iyi durumda toplam olarak $\sum_{1}^{r=B}((N-r+1)^2) \in \Omega(BN^2)$, kötü durumda toplam olarak $\sum_{1}^{r=B}((N-r+1)^2) \in O(N^2B)$ olduğundan dolayı FLOODTREE algoritmasının mesaj karmaşıklığı $r=B$ için $\Theta(N^2B)$'e eşittir. FLOODTREE algoritması en kötü durumda $r=N$ için $\sum_{1}^{r=N}(N-r+1)^2 \in O(N^3)$'e eşittir. En iyi durumda B=1 için $\Omega(N^2)$'e eşittir. ■

Ön Teorem 4.38. FLOODTREE algoritmasının ikinci raunt dışındaki tek numaralı raundunun mesaj karmaşıklığı N düğüm için T birleşen ağaçlardaki toplam düğüm sayısı olmak üzere $\Theta(T)$'e, üst limitte $O(N)$'e eşittir.

İspat. FLOODTREE algoritmasının ikinci raunt dışındaki tek numaralı raundunda bağlayıcı seçilen düğüm *AĞAÇ_BİLGİSİ* mesajını komşularına gönderir. Bu mesajı alan ağaç üzerindeki düğümler bu mesajı bir kez iletir. Bundan dolayı mesaj karmaşıklığı $\Theta(T)$'e eşittir. Üst limitte mesaj karmaşıklığı $T=N$ için $O(N)$'e eşittir. ■

Teorem 4.14. FLOODTREE Algoritmasının mesaj karmaşıklığı B bağlayıcı sayısı olmak üzere $\Theta(N^2 B)$'ye üst sınırda $O(N^3)$'e eşittir.

İspat. FLOODTREE Algoritmasının mesaj karmaşıklığı (mk):

İlk raundun mk + ikinci raundun mk+ (tek numaralı rauntların mk + çift numaralı rauntların mk)

$\Theta(\sum|N_d{}^2|+|HK|-\sum|N_d|)+\Theta(N|HK|)+\Theta(N^2B)+\Theta(BT) \in \Theta(N^2B)$

Üst sınırda $B=N$ için $O(N^3)$'e eşittir. ■

Ön Teorem 4.39. STREE Algoritmasının ilk raundunun mesaj karmaşıklığı N_d aralarında birbirleri üzerinden patika olan ve HK düğümlerinden oluşan ayrık kümeler olduğunda $\Theta(\sum| N_d{}^2 | + |HK|- \sum| N_d |)$ olmak üzere alt sınırda $\Omega(|HK|)$'ya üst sınırda $O(N^2)$'ye eşittir.

İspat. Ön Teorem 4.35'te FLOODTREE için ispat edilmiştir. ■

Ön Teorem 4.40. STREE Algoritmasının ikinci raundunun mesaj karmaşıklığı $\Theta(N)$'dir.

İspat. STREE Algoritmasının ikinci raundunda *ÇIKIŞ_DÜĞÜMÜ* diğer düğümlere *SENK_AĞACI_KUR* mesajını yayarlar. Her düğüm bir tane *SENK_AĞACI_KUR* mesajını gönderir, bundan dolayı mesaj karmaşıklığı $\Theta(N)$'dir. ■

Ön Teorem 4.41. STREE Algoritmasının bağlayıcı seçtiği ilk raundunun mesaj karmaşıklığı $\Theta(N)$'dir.

İspat. STREE Algoritmasının bağlayıcı seçtiği ilk rauntta *SELMAN* düğümler *SELMAN_DRM* ve *İST* mesajı gönderirken, *HELMAN* düğümler *DRM*, *BĞLAN* ve *İST* mesajı gönderirler. *BĞLAN* mesajı da her zaman gönderilmez. Bir düğüm tarafından en çok gönderilen mesaj sayısı 3'tür. Bundan dolayı mesaj karmaşıklığı $\Theta(N)$'e eşittir. ■

Ön Teorem 4.42. STREE Algoritmasının bağlayıcı seçtiği ikinci rauntta mesaj karmaşıklığı $\Theta(N)$'dir.

İspat. Bu rauntta her düğüm *ÇIKIŞ_DÜĞÜMÜ*'ne doğru ağırlık oranlarını içeren tek bir *BĞLAN_İST* mesajını iletir. Bundan dolayı mesaj karmaşıklığı $\Theta(N)$'dir. ■

Ön Teorem 4.43. STREE Algoritmasının bağlayıcı seçtiği üçüncü rauntta mesaj karmaşıklığı $\Theta(N)$'dir.

İspat. Bu rauntta *ÇIKIŞ_DÜĞÜMÜ* en küçük ağırlıklı düğüme *BĞLAN_İST_ONAY* mesajını gönderir. Bu gönderim işlemi en fazla N-1 zıplamada gerçekleşebilir. Eğer ÇIKIŞ_DÜĞÜMÜ algoritmayı bitirecekse tüm düğümlere *SON* mesajını yayar. Bu işlem için de N mesaj gönderimi gerekir. Sonuç olarak bu raundun mesaj karmaşıklığı $\Theta(N)$'dir. ■

Teorem 4.15. STREE Algoritmasının mesaj karmaşıklığı Θ($\sum| N_d{}^2 |$+ |HK| N)'e, üst sınırda O(N^2)'ye eşittir.

İspat. STREE Algoritmasının mesaj karmaşıklığı (mk):

İlk raundun mk + İkinci raundun mk + |HK| (Bağlayıcı seçen ilk raundun mk + Bağlayıcı seçen ikinci raundun mk + Bağlayıcı seçen üçüncü raundun mk)

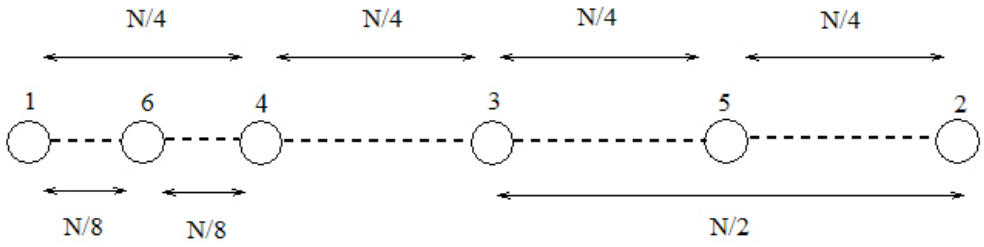
Ön Teorem 4.39, Ön Teorem 4.40, Ön Teorem 4.41, Ön Teorem 4.42, Ön Teorem 4.42'a göre denklem şu şekilde doldurabilir:

Θ ($\sum| N_d{}^2 |$ + |HK|- $\sum| N_d |$) + Θ(N) + |HK| (Θ(N) + Θ(N) + Θ(N)) Є

Θ($\sum| N_d{}^2 |$+ |HK| N)'e eşit olur. Üst limitte |HK|=N için O(N^2) olur. ■

Ön Teorem 4.44. DTOR_TREE algoritmasının mesaj karmaşıklığı N_d aralarında birbirleri üzerinden patika olan ve HK düğümlerinden oluşan ayrık kümelerken üst sınırda O($N_d{}^2 \log_2(N_d)$) kadar mesaj harcar.

İspat. Şekil 4.31'de DTOR_TREE algoritması için en kötü topoloji verilmiştir. Bu topolojide en büyük kimlikli düğüm *İST* ve *ARA* mesajları göndererek aramaya başlar, son olarak en küçük kimlikli düğüm aramayı bitirmiştir. Bu durumda toplam mesaj sayısı:



Şekil 4.31 DTOR_TREE algoritması mesaj karmaşıklığı en kötü durum

Düğüm 1: İlk iterasyonda *İST* mesajı gönderir ve *DRM* cevabı alır. Bu iterasyonda 2 mesaj gönderimi yapılır. İkinci iterasyonda düğüm 1, komşusuna *ARA* mesajını gönderir, komşusu *ARA* mesajını *İST* olarak iletir, *İST* mesajını alan düğüm *DRM* mesajını iki zıplamada iletir. İkinci iterasyonda 4 mesaj gönderimi yapılır. Böylece düğüm 1 toplam (N_d)(N_d - 1) mesaj gönderimi yapar.

Düğüm 2: Düğüm 2, düğüm 1'i buluna kadar ağacını genişletir. Toplamda (N_d -1)(N_d - 2) mesaj gönderimi yapılır.

Düğüm 3: Düğüm 3, düğüm 2 ve düğüm 1'i buluna kadar ağacını genişletir, çünkü bu düğümleri aynı iterasyon içinde bulur. Toplamda (N_d -2) (N_d - 3) mesaj gönderimi yapılır.

Düğüm 4: Düğüm 1 ve düğüm 3'u bulana kadar ağacını genişletir. Toplamda (N_d /2-2)(N_d /2 -3) mesaj gönderimi yapılır.

Düğüm 5: Düğüm 2 ve düğüm 3'u bulana kadar ağacını genişletir. Toplamda (N_d / 2-2)(N_d /2 -3) mesaj gönderimi yapılır.

Düğüm 6: Düğüm 1 ve düğüm 4'ü bulana kadar ağacını genişletir. Toplamda (N_d / 4-2)(N_d /4 -3) mesaj gönderimi yapılır.

Bütün düğümlerin gönderdiği mesajlar bu şekilde toplanırsa toplam $T \in O(N_d{}^2 \log_2(N_d))$ kadar mesaj alır. ■

Ön Teorem 4.45. Düzenlenmiş GHS algoritmasının mesaj karmaşıklığı $O(N_d \log_2(N_d))$'dır.

İspat. GHS Algoritmasının mesaj karmaşıklığı $O(N_d \log_2(N_d))$'dir. Düzenlenmiş GHS algoritmasındaki düğümler sadece bir adet *SON* ve *TAMAM* mesajları gönderdikleri için algoritmanın mesaj karmaşıklığı değişmez. ■

Ön Teorem 4.46. ASYNTREE algoritmasının bağlayıcı seçen raundunun mesaj karmaşıklığı $\Theta(N)$'dir.

İspat. ASYNTREE'nin bağlayıcı seçen algoritmasında *HELMAN* düğümler *DRM*, *İST*, *BĞLAN* ve *BĞLAN_İST* mesajlarını gönderirler. *SELMAN* düğümler *DRM*, *SELMAN_DRM*, *İST* ve *BĞLAN_İST* mesajlarını gönderirler. Raunt başlarken her düğüm 1 kere *AĞAÇ_KYNŞTR* mesajını gönderir. Bir düğüm bir raunt içinde en fazla 5 mesaj gönderir, bundan dolayı mesaj karmaşıklığı $\Theta(N)$'dir. ■

Teorem 4.16. ASYNTREE Algoritmasının mesaj karmaşıklığı DTOR_TREE algoritması kullanıldığında üst sınırda $O(N^2 \log_2(N))$, düzenlenmiş GHS algoritması kullanıldığında B bağlayıcı sayısı olmak üzere $\Theta(NB)$, alt sınırda $\Omega(N \log_2(N))$, üst sınırda $O(N^2)$'dir.

İspat. ASYNTREE Algoritmasının mesaj karmaşıklığı (mk):

İlk raundun mk + $|B|$ (Bağlayıcı seçen raundun mk)

İlk rauntta DTOR_TREE kullanılırsa Ön Teorem 4.44'e göre üst sınırda $N_d=N$ için $O(N^2\log_2(N))$ mesaj harcanır. Bağlayıcı seçen raundun mesaj karmaşıklığı Ön Teorem 4.46'ya göre $\Theta(N)$ olduğu için toplam mesaj karmaşıklığı üst sınırda $B=N$ için $O(N^2\log_2(N))+ O(N^2) \in O(N^2\log_2(N))$ mesaj harcanır.

Eğer ilk rauntta düzenlenmiş GHS kullanılırsa $N_d=N$ ve $B=N$ için Ön Teorem 4.45'e göre $O(N \log_2(N)) + O(N^2) \in O(N^2)$ mesaj harcanır. Alt limitte $B \leq log_2(N)$ *için* $\Omega(Nlog_2(N))$ kadar mesaj gereklidir. ■

4.3.3. Zaman Karmaşıklığı

Ön Teorem 4.47. FLOODSET algoritmasında D ağın çapı olmak üzere bir raundun zaman karmaşıklığı $\Theta(D)$'ye eşittir.

İspat. Her düğüm birbirine *AĞIRLIK_ORANI* mesajını eş zamanlı göndereceği için zaman karmaşıklığı $\Theta(D)$'ye eşit olur. ■

Teorem 4.17. FLOODSET algoritmasının zaman karmaşıklığı $\Theta(D\ |HK|)$, üst sınırdaysa $O(N\ D)$'dir.

İspat. FLOODSET Algoritmasında Ön Teorem 4.47'ye göre bir raundun zaman karmaşıklığı $\Theta(D)$'ye eşittir. Bu durumda $|HK|$ kadar rauntta toplam $\Theta(D\ |HK|)$ kadar zaman harcanır. Üst limitte $|HK|=N$ için $O(N\ D)$'e eşit olur. ■

Ön Teorem 4.48. SSET algoritmasındaki bir raundun zaman karmaşıklığı $\Theta(\Delta)$ 'e eşittir.

İspat. Bir raunt içindeki işlemler aşağıda listelenmiştir:

- Düğümler birbirleriyle *AĞIRLIK_ORANI* mesajını paylaşır. Bu işlem $\Theta(\Delta)$ zaman alır.
- *ADAY_DEĞİLİM* ve *BAĞLANMA_İSTEĞİ* mesajlarının gönderilmesi $\Theta(\Delta)$ zaman alır.
- *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* mesajının gönderilmesi $\Theta(\Delta)$ zaman alır.
- *BAĞLAN* mesajının gönderilmesi eş zamanlı olduğundan $\Theta(1)$ zaman alır.
- *BAĞLANDI* mesajının gönderilmesi $\Theta(\Delta)$ zaman alır.

Bu işlemler için gerekli $\Theta(4\Delta+1) \in \Theta(\Delta)$'dir. ■

Teorem 4.18. SSET algoritmasının zaman karmaşıklığı, *N* düğüm için Θ(Δ $|P_M|$)'e alt limitte Ω(Δ)'e ye üst limitte O(*N*Δ)'e eşittir.

İspat. Ön Teorem 4.31'e göre algoritma $r=|P_M|$ raunt çalışacaktır. Ön Teorem 4.48'e göre her raundun zaman karmaşıklığı Θ(Δ)'e eşittir. Böylece algoritma Θ(Δ $|P_M|$) zaman alır. Alt sınırda $|P_M|=1$ olduğunda algoritma Ω(Δ) süre alır. Üst limitte $|P_M|=N$ için O(*N*Δ) süre algoritma çalışır. ■

Ön Teorem 4.49. ASYNSET algoritmasındaki bir raundun zaman karmaşıklığı Θ(*D*+*Δ*) 'e eşittir.

İspat. Bir raundun içinde *ÇIKIŞ_DÜĞÜMÜ BAŞLA* mesajı göndererek raundu başlatır. Bu işlem Θ(*D*) kadar vakit alır. Bu işlemden sonra düğümler *BAĞLANMADI* veya *BAĞLANDI* mesajlarını göndererek ağırlıklarını güncellerler. Bu işlem her düğümün aynı anda bir mesaj aldığını ve bu mesajı ağırlık oranını bulmak için kullandığını düşünürsek Θ(*Δ*) zaman alır. Düğümler, daha sonra iki zıplama uzaklıktaki komşularının ağırlıklarını öğrenirler. Bu işlemin ilk basamağında her düğüm kendi ağırlık oranını *AĞIRLIK_ORANI* mesajını içinde gönderir ve komşuları içinde en küçük ağırlığı bulur, bu işlem Θ(*Δ*) vakit alır. Düğümler bu mesajları topladıktan sonra en küçük ağırlığı *BAĞLANMA_İSTEĞİ* veya *ADAY_DEĞİLİM* mesajlarını gönderiler, bu işlemler eş zamanlı olarak Θ(*Δ*) zaman alır. Aday olmayan düğümler *EN_KÜÇÜK_BAĞLANMA_İSTEĞİ* içinde en küçük adayın kimliğini göndermeleri Θ(*Δ*) zaman alır. Böylelikle *HÂKİM_ELEMAN* düğümler seçilip komşularına *BAĞLAN* mesajı gönderirler, bu işlem yapılırken HÂKİM_ELEMAN düğümlerin çevresinde mesaj gönderen başka komşuları olmadığı için Θ(1) kadar vakit alır. En son işlemde düğümler senkronize olurlar, bu işlem Θ(*D*) kadar vakit alır. Toplam harcanan zaman bu işlemlerin toplam zamanına eşittir. Böylece algoritma Θ(2*D*+4Δ+*1*) Є Θ(*D*+Δ) kadar zaman alır. ■

Teorem 4.19. ASYNSET algoritmasının zaman karmaşıklığı, *N* düğüm için Θ((*D*+Δ) $|P_M|$)'e alt limitte Ω(*D*+Δ)'e üst limitte O(*N*(*D*+Δ))'e eşittir.

İspat. Ön Teorem 4.33'a göre algoritma $r=|P_M|$ raunt çalışacaktır. Ön Teorem 4.49'a göre her raundun zaman karmaşıklığı Θ(*D*+Δ)'e eşittir. Böylece algoritma Θ((*D*+Δ) $|P_M|$) zaman alır. Alt sınırda $|P_M|=1$ olduğunda algoritma Ω(*D*+Δ) süre alır. Üst limitte $|P_M|=N$ için O(*N*(*D*+Δ)) süre algoritma çalışır. ■

Ön Teorem 4.50. FLOODTREE algoritmasının ilk raundunun zaman karmaşıklığı *D* ağın çapı olmak üzere Θ ($|N_{dmax}|$) alt sınırda Ω(1), üst sınırda O(*D*)'dir.

İspat. |HK| içindeki her düğümün birbiri üzerinden *HÂKİM_ELEMAN* mesajını eş zamanlı olarak gönderdiği için N_d kümelerinin içinden en çok elemana sahip olan N_{dmax} kümesi kadar zamana ihtiyaç vardır, sonuçta O($|N_{dmax}|$) kadar algoritma zaman alır. Alt sınırda N_{dmax} = 1 için algoritma Ω(1) zaman alır. Üst sınırda N_{dmax}= D olabileceğinden dolayı O(D) zaman gereklidir. ■

Ön Teorem 4.51. FLOODTREE algoritmasının ikinci raundunun zaman karmaşıklığı en kötü durumda O(D)'ye eşit olur.

İspat. Hâkim küme içindeki düğümlerin ağa *AĞAÇ_BİLGİSİ* mesajını yayar, bu işlem en çok ağın çapı kadar vakit alır. ■

Ön Teorem 4.52. FLOODTREE algoritmasının birinci raunt dışındaki tek numaralı rauntların zaman karmaşıklığı en kötü durumda O(D)'ye eşit olur.

İspat. Hâkim küme içinde olmayan düğümlerin ağa *AĞIRLIK_ORANI* mesajını yayar, bu işlem en çok ağın çapı kadar vakit alır. ■

Ön Teorem 4.53. FLOODTREE Algoritmasının ikinci raunt dışındaki çift numaralı rauntların zaman karmaşıklığı en kötü durumda O(D)'ye eşit olur.

İspat. Yeni birleşen ağaca ait olan düğümlerin ağa *AĞAÇ_BİLGİSİ* mesajını yayar, bu işlem en çok ağın çapı kadar vakit alır. ■

Teorem 4.20. FLOODTREE algoritmasının zaman karmaşıklığı, B bağlayıcı düğüm sayısı olmak üzere Θ(BD)'e alt limitte Ω(D)'e ye üst limitte O(ND)'e eşittir.

İspat. ASYNTREE Algoritmasının zaman karmaşıklığı (zk):

İlk raundun zk + İkinci raundun zk + $|B|$ (tek numaralı raundun zk + çift numaralı raundun zk)

Ön Teorem 4.50, Ön Teorem 4.51, Ön Teorem 4.52, Ön Teorem 4.53 kullanılarak:

$O(D)+O(D)+|B|(O(D)+O(D)) \in \Theta(BD)$

Üst limitte B=N için O(ND)'e eşittir.■

Ön Teorem 4.54. STREE Algoritmasının ilk raundunun zaman karmaşıklığı D ağın çapı olmak üzere Θ ($|N_{dmax}|$) alt sınırda Ω(1), üst sınırda O(D)'dir.

İspat. Ön Teorem 4.50 ile aynıdır. ■

Ön Teorem 4.55. STREE Algoritmasında ikinci raundun zaman karmaşıklığı Θ(*D*)'dir.

İspat. ÇIKIŞ_DÜĞÜMÜ ağa *AĞAÇ_KUR* mesajını yayar, mesaj eş zamanlı olarak ağa yayılırken, ağın çapı kadar vakit alır. ■

Ön Teorem 4.56. STREE Algoritmasında bağlayıcı seçen ilk raundun zaman karmaşıklığı Θ($|N_{dmax}|$+Δ), alt sınırda Ω(Δ) , üst sınırda O(*N*)'dir.

İspat. Yapılan işlemlerin listesi aşağıdadır:

- *SELMAN*lar tarafından *DRM* mesajlarının gönderilmesi için Θ(Δ) zaman gereklidir.
- *SELMAN*lar tarafından *SELMAN_DRM* mesajlarının gönderilmesi için Θ(Δ) zaman gereklidir.
- *SELMAN*lar tarafından *İST* mesajlarının gönderilmesi için Θ(Δ) zaman gereklidir.
- *İST* mesajlarının *HELMAN*lar üzerinden iletilip *LDR_HELMAN* durumuna gitmesi için Θ($|N_{dmax}|$) zaman gereklidir.
- *BĞLAN* mesajının *LDR_HELMAN* tarafından aday düğüme gitmesi için Θ($|N_{dmax}|$) zaman gereklidir.

Bu işlemler toplamda Θ(2$|N_{dmax}|$+3Δ) Є Θ($|N_{dmax}|$+Δ) kadar zaman gerektirir. ■

Ön Teorem 4.57. STREE Algoritmasında bağlayıcı seçen ikinci raundun zaman karmaşıklığı *D* ağın çapı olmak üzere Θ(*D*)'ye eşittir.

İspat. Bağlayıcı seçen ikinci rauntta ağacın en altından başlayarak en üstüne doğru *BĞLAN_İST* mesajları iletilir. Bu durumda geçen zaman senkronizasyon ağacının derinliği veya çizgenin çapı kadar olabilir. ■

Ön Teorem 4.58. STREE Algoritmasında bağlayıcı seçen üçüncü raundun zaman karmaşıklığı *D* ağın çapı olmak üzere Θ(*D*)'ye eşittir.

İspat. Yeni seçilen bağlayıcı düğüm en fazla *D*-1 zıplama uzaklıkta olabilir. Böylece üst limitteki zaman karmaşası O(*D*)'e eşittir. ■

Teorem 4.21. STREE Algoritmasının zaman karmaşıklığı Θ(*NB*) 'e, üst limitte O(N^2)'ye eşittir.

İspat. STREE Algoritmasının zaman karmaşıklığı (zk) aşağıdaki denklem ile gösterilir:

İlk raunt zk + İkinci raunt zk + |HK| (Üçüncü raunt + Dördüncü raunt + Beşinci raunt)

Ön Teorem 4.54, Ön Teorem 4.55, Ön Teorem 4.56, Ön Teorem 4.57, Ön Teorem 4.58'e göre değerlerini yerine koyarsak:

O(*D*) + O(*D*) + *B*(O(*D*)+O(*N*)+O(*D*)) Є Θ(*NB*)

Üst limitte *B*=*N* için O(N^2) zaman alır.■

Ön Teorem 4.59. DTOR_TREE algoritmasının zaman karmaşıklığı N_d aralarında birbirleri üzerinden patika olan ve HK düğümlerinden oluşan ayrık kümelerken üst sınırda O($N_d{}^2$+*D*) kadar zaman sürer.

İspat. DTOR_TREE algoritmasında Şekil 4.31'de gösterilen en kötü durumda düğüm 1'in diğer düğümlere ulaşması (N_d)(N_d - 1) mesaj gönderimi sonrası olur ve O($N_d{}^2$) kadar zaman alır. DTOR_TREE algoritmasının senkronize olması için gereken süre *D*'dir, böylece toplam en kötü durumunda O($N_d{}^2$+*D*) kadar vakit gerekir. ■

Ön Teorem 4.60. Düzenlenmiş GHS algoritmasının zaman karmaşıklığı O(N_d $\log_2(N_d)$ + *D*)'dır.

İspat. GHS Algoritmasının zaman karmaşıklığı O(N_d $\log_2(N_d)$)'dir. Düzenlenmiş GHS algoritmasında *SON* mesajının gitmesi ve senkronizasyon işlemi için O(*D*) vakit alır, bu algoritmanın toplam zaman karmaşıklığı O(N_d $\log_2(N_d)$ + *D*)'ye eşittir. ■

Ön Teorem 4.61. ASYNTREE algoritmasının bağlayıcı seçen raundunun zaman karmaşıklığı O(N_d+*D*+*Δ*)'dir.

İspat. ASYNTREE algoritmasının bağlayıcı seçen raunttaki ilk işlem *ÇIKIŞ_DÜĞÜMÜ'*nden gelen *BAŞLA* mesajının yayılmasıdır ve bu işlem Θ(*D*) vakit alır. Daha sonra *SELMAN* düğümler *HELMAN* düğümlerden *DRM* mesajları toplarlar, bu işlem Θ(Δ) vakit alır. *HELMAN*'lar *DRM* mesajlarını topladıktan sonra *İST* mesajı gönderirler, bu işlem de Θ(Δ) zaman alır. Daha sonra *İST* mesajları

LDR_HELMAN'a kadar iletilir, bu işlem $\Theta(N_d)$ vakit alır. *İST* mesajlarından sonra *BĞLAN* mesajının ters yönde iletilmesi de $\Theta(N_d)$ vakit alır. Düğümler son olarak yeni raundu başlatmak için senkronize olurlar. Bu işlem $\Theta(D)$ vakit alır. Toplamda süre işlem $T \in O(N_d+D+\Delta)$'dır. ■

Teorem 4.22. ASYNTREE Algoritmasının zaman karmaşıklığı DTOR_TREE algoritması kullanıldığında üst sınırda $O(N^2)$, düzenlenmiş GHS algoritması kullanıldığında B bağlayıcı sayısı olmak üzere $\Theta(BD+B\Delta+ (N_d+B)\log_2(N_d) + N_dB)$, üst sınırda $O(N^2)$'dir.

İspat. STREE Algoritmasının zaman karmaşıklığı (zk):

İlk raundun zk + $|B|$ (Bağlayıcı seçen raundun zk)

İlk rauntta DTOR_TREE kullanılırsa Ön Teorem 4.59'a göre üst sınırda $N_d=N$ için $O(N^2)$ sürede olur. Bağlayıcı seçen raundun zaman karmaşıklığı Ön Teorem 4.60'a göre $O(N_d+D+\Delta)$ olduğu için zaman karmaşıklığı üst sınırda $B=N$, $N_d=N$, için $O(N^2)$ zaman sürer. Eğer ilk rauntta düzenlenmiş GHS kullanılırsa Ön Teorem 60'e göre toplamda $\Theta(B (N_d+D+\Delta) + N_d \log_2(N_d)) = \Theta(BD+B\Delta+(N_d+B)\log_2(N_d)+BN_d)$, üst sınırda $N_d=B=N$ için $O(N^2)$'dir. ■

4.3.4. Uzay Karmaşıklığı

Uzay karmaşıklığı yapılırken farklı veri tiplerinin sabit ve birbirleriyle yakın miktarda bellek alanları tuttukları farz edilmiştir.

Teorem 4.23. FLOODSET algoritmasında en büyük boyutlu mesajın uzay karmaşıklığı $\Theta(2)$'e eşittir.

İspat. FLOODSET algoritmasında *AĞIRLIK_ORANI*(*düğüm_kimliği*, *ağırlık_oranı*) mesajında 2 alan olduğundan dolayı $\Theta(2)$'e eşittir. ■

Teorem 4.24. FLOODSET algoritmasında bir düğümün kullanması gereken belleğin uzay karmaşıklığı $\Theta(4)$'e eşittir.

İspat. FLOODSET algoritmasında her düğüm düğüm_kimliği, ağırlık_oranı, en_küçük_ağırlık_oranı, durum_bilgisi değişkenlerini tutar. Bundan dolayı kullanılacak belleğin uzay karmaşıklığı $\Theta(4)$'e eşittir. ■

Teorem 4.25. SSET ve ASYNSET algoritmasında en büyük boyutlu mesajın uzay karmaşıklığı $\Theta(2)$'e eşittir.

İspat. SSET ve ASYNSET algoritmasında en büyük boyutlu mesaj *AĞIRLIK_ORANI*(düğüm_kimliği, ağırlık_oranı) olduğundan dolayı $\Theta(2)$'e eşittir. ■

Teorem 4.26. SSET, ASYNSET, STREE, GHS algoritmalarında bir düğümün kullanması gereken belleğin uzay karmaşıklığı O(Δ)'ya eşittir.

İspat. SSET, ASYNSET, STREE, GHS algoritmalarında bir düğümün komşularının listesini tutması gerekir. Bundan dolayı Δ'lık bir dizi tutar. Diğer değişkenler sabit sayıda olduğundan dolayı düğüm tutması gereken alan O(Δ)'ya eşittir. ■

Teorem 4.27. STREE algoritmasında en büyük boyutlu mesajın uzay karmaşıklığı Θ(3)'e eşittir.

İspat. STREE algoritmasında en büyük boyutlu mesaj *İST*(düğüm_kimliği,gerçek_düğüm_kimliği, ağırlık_oranı) olduğundan dolayı Θ(3)'e eşittir. ■

Teorem 4.28. DTOR_TREE algoritmasında en büyük boyutlu mesajın uzay karmaşıklığı O(Δ)'ya eşittir.

İspat. DTOR_TREE algoritmasında en büyük boyutlu mesaj *DRM*(*düğüm_kimliği*,*komşuların listesi*) olduğundan dolayı O(Δ)'e eşittir. ■

Teorem 4.29. DTOR_TREE algoritmalarında bir düğümün kullanması gereken belleğin uzay karmaşıklığı HK_T kümedeki düğüm sayısı olmak üzere $O(HK_T)$'e eşittir.

İspat. DTOR_TREE algoritmasında bir düğüm sabit sayıda değişken kullanır ve bunun yanında lider düğüm HK_T kadar bir liste tutar. Bundan dolayı uzay karmaşıklığı $O(HK_T)$'e eşittir. ■

Teorem 4.30. ASYNTREE algoritmasında en büyük boyutlu mesajın uzay karmaşıklığı HK_T kümedeki düğüm sayısı olmak üzere $O(HK_T)$'e eşittir.

İspat. ASYNTREE algoritmasında en büyük boyutlu mesaj *BĞLAN_İST*(düğüm_kimliği, gerçek_düğüm_kimliği, ağaçtaki_düğüm_sayısı, bağlanmış ağaçların listesi) olduğundan dolayı $O(HK_T)$'a eşittir. ■

Teorem 4.31. Düzenlenmiş GHS algoritması kullanılan ASYNTREE algoritmasında bir düğümün kullanması gereken belleğin uzay karmaşıklığı O(Δ)'e eşittir.

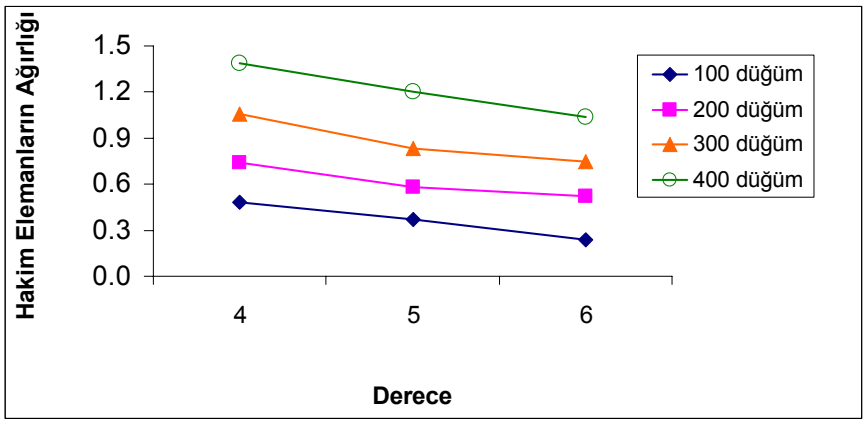
İspat. GHS algoritmasında bir düğümün kullandığı belleğin uzay karmaşasının O(Δ)'e eşit olduğu Teorem 4.26'da gösterilmiştir. ASYNTREE algoritmasında sabit sayıda değişken kullanılır, bunun yanında komşuluk listesi tutulduğu için O(Δ)'e eşit olur. Sonuç olarak GHS kullanılan ASYNTREE algoritmasının uzay karmaşıklığı O(Δ)'e eşit olur. ■

4.4. Performans Değerlendirmesi

FLOODSET, SSET, ASYNSET, FLOODTREE, STREE ve ASYNTREE algoritmalarını ns2 versiyon 2.31 benzetim ortamında uyguladık. Rastgele eşit yoğunlukta yayılan 100 düğümden 400 düğüme kadar düğüm içeren ağlar ürettik. IEEE 802.11 radyo katmanı kullanılırken, Wang'ın çalışmasında da yaptığı gibi çarpışmaların olmadığı ideal bir MAC katmanı kullanıldı (Wang et al., 2006). İletim gücü 0.660 mW olarak, alım gücü 0.395mW olarak ayarlanırken kapsama alanı 250m'ye sabitlendi. Ortalama düğüm derecesini 4, 5 ve 6 olarak ayarlamak için düğümler eşleme bölümünde Şekil 3.13'de gösterilen yüzey alanları üzerine yerleştirildi. Düğümlere rastgele enerjiler verildi ve enerji aralıkları 0-50J, 0-100J, 0-150J olarak değiştirilerek testler yapıldı. Algoritmalarımızı literatürdeki çizge teorik kümeleme algoritmalarıyla karşılaştırmak için Wang'ın, Bao'nun ve Chaterjee'nin MIS (Maksimal Bağımsız Küme) tabanlı algoritmaları da ns2 üzerinde uygulandı. Wang'ın algoritmasının ikinci safhasında GHS algoritması uygulandı. Hâkim düğümlerin ağırlığı, bağlayıcı düğümlerin ağırlığı, hâkim düğümlerin sayısı, bağlayıcı düğümlerin sayısı, tüketilen enerji miktarları ve gönderilen mesaj gönderimleri ve çalışma zamanları ölçülmüştür.

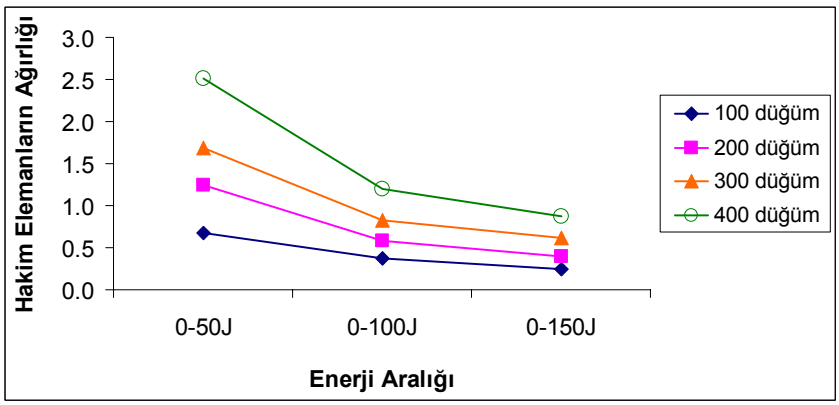
4.4.1. Hâkim Düğümlerin Ağırlığı

Ağırlıklı HK algoritmalarının öncelikli hedefi az maliyetli düğümleri içine almaktır. Böylece olabildiğince küçük maliyette bir HK kurulmuştur. Yüksek enerjili bir düğümün maliyeti düşük olduğunu düşünürsek, küçük maliyette bir HK oluşturmak aynı zamanda yüksek enerjili düğümlerden bir HK oluşturmamızı sağlar. Böylece hâkim kümemizdeki düğümlerin enerjileri daha uzun süre dayanacak ve enerji yönünden daha güvenli bir HK elde etmiş olacağız. Aksi belirtilmedikçe bundan sonraki testlerde ortalama düğüm derecesi 5, düğüm sayısı 300 ve düğümlerin enerjileri 0-100J arasındadır.

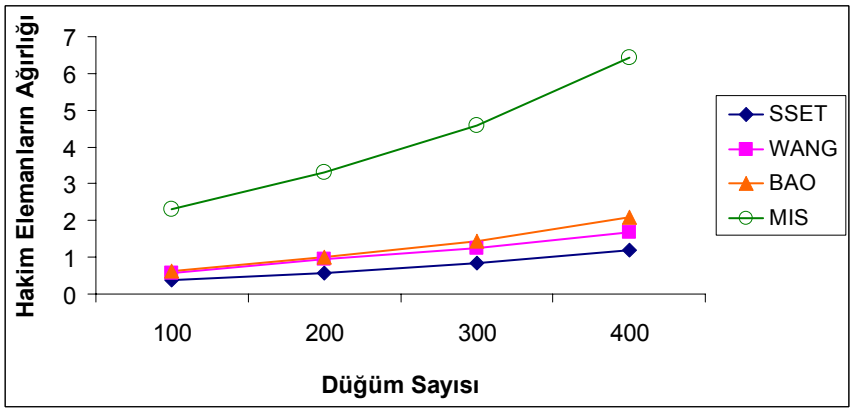


Şekil 4.32 SSET Algoritmasında Düğümlerin Ortalama Derecelerine Bağlı Üretilen Hâkim Elemanların Ağırlığı

Şekil 4.32'de düğümlerin ortalama derecelerine bağlı olarak SSET tarafından üretilen hâkim elemanların ağırlığı verilmiştir. Ortalama düğüm derecesi arttıkça SSET daha yüksek enerjili düğümleri seçer ve böylece HK içindeki düğümlerin toplam maliyeti azalır. Ayrıca Şekil 4.32'de görüldüğü üzere düğüm sayısı arttıkça hâkim kümenin maliyeti düzenli bir şekilde artar. Şekil 4.33'de enerji aralıklarına bağlı olarak hâkim kümenin maliyet değişimi gösterilmiştir. 0-50J aralığında yapılan testlerde düğümler 0 ile 50 J arasında rastgele enerji atanmıştır. Enerjinin üst limiti arttırıldıkça düğümlerin yüksek enerjili olma ihtimali artmaktadır. Şekil 4.33'de görüldüğü üzere SSET düğümlerin enerjileri arttıkça yüksek enerjili düğümleri seçer ve hâkim kümenin maliyetini düşürür.

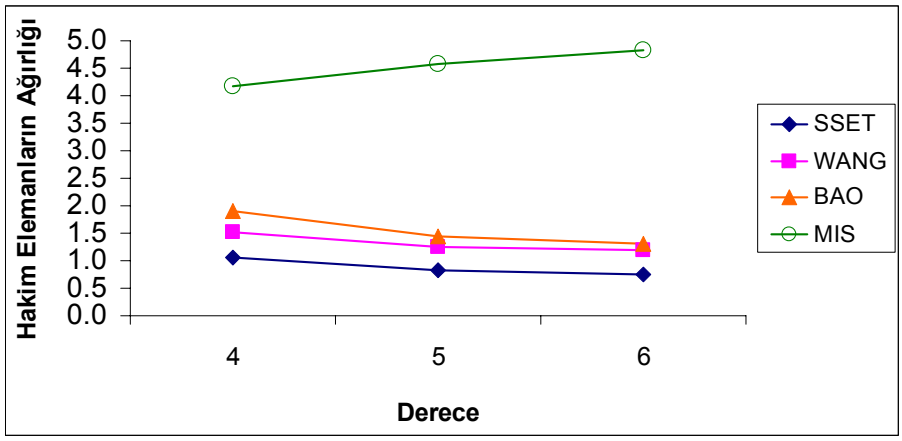


Şekil 4.33 SSET Algoritmasında Enerji Aralıklarına Bağlı Algoritmaların Ürettiği Hâkim Elemanların Ağırlığı

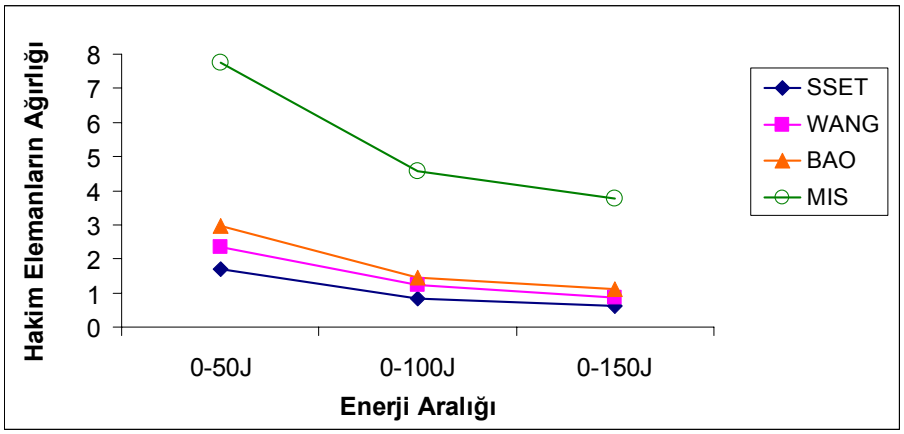


Şekil 4.34 Düğüm Sayısına Bağlı Algoritmaların Ürettiği Hâkim Elemanların Ağırlığı

Şekil 4.34'de değişen düğüm sayısına bağlı olarak algoritmaların ürettiği HK elemanlarının ağırlıkları verilmiştir. SSET'in seçtiği hâkim kümenin maliyeti diğer algoritmalardan daha küçük çıkmıştır. MIS algoritması en kötü performansı gösterirken diğer algoritmalardan çok daha maliyetli bir HK seçmiştir. Wang'ın algoritması da iyi performans gösterirken Bao'nun algoritması Wang'ın algoritmasına yakın bir performans göstermiştir. Şekil 4.35'de değişen düğüm derecelerine bağlı olarak algoritmaların ürettiği HK elemanlarının ağırlıkları verilmiştir. Şekil 4.34'da gördüğümüz performans sıralaması değişmemiş ve en iyi performansı SSET göstermiştir. Enerji aralıklarını değiştirdiğimiz zaman da algoritmaların arasındaki performans sıralamasının değişmediği Şekil 4.36'da görülmektedir.



Şekil 4.35 Düğüm Derecesine Bağlı Algoritmaların Ürettiği Hâkim Elemanların Ağırlığı

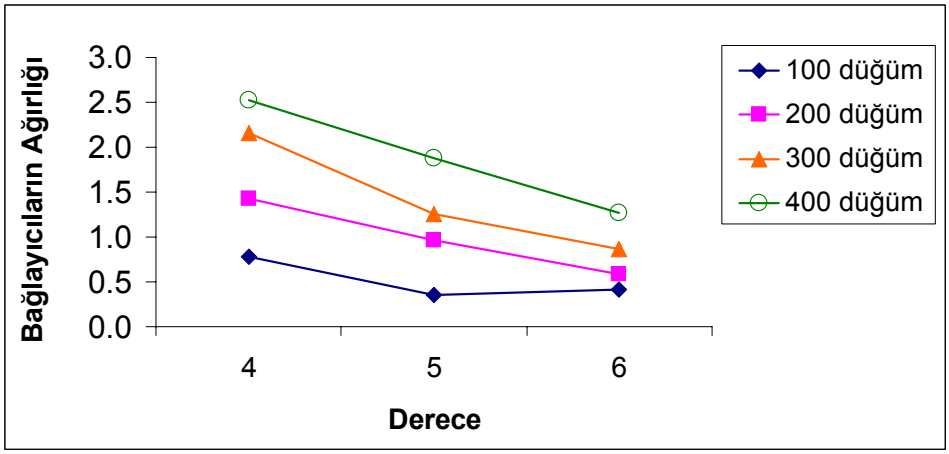


Şekil 4.36 Enerji Aralıklarına Bağlı Algoritmaların Ürettiği Hâkim Elemanların Ağırlığı

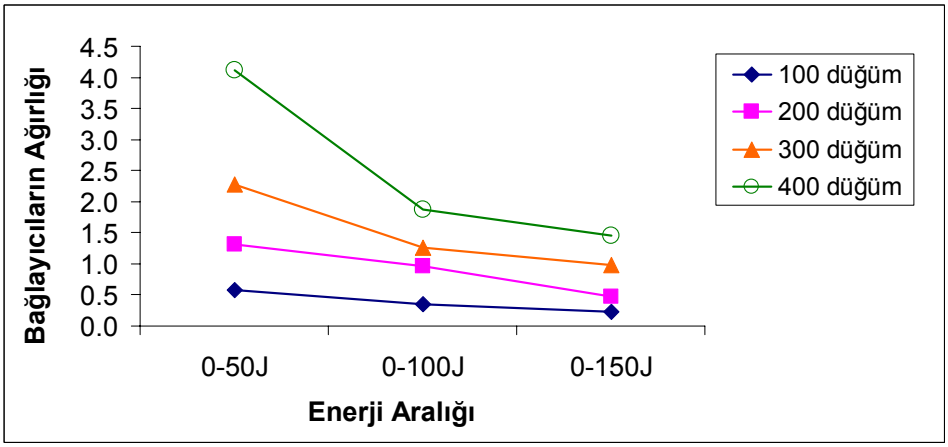
4.4.2. Bağlayıcı Düğümlerin Ağırlığı

Şekil 4.37 ve Şekil 4.38'te STREE algoritmasının değişen düğüm derecelerine ve enerji aralıklarına göre bağlayıcıların aralıkları görülmektedir. Bu noktada tekrardan hatırlatırsak STREE ve ASYNTREE algoritmalarının seçtiği bağlayıcı düğümler tamamen aynıdır, STREE senkron çalışırken, ASYNTREE asenkron çalışır. Düğüm dereceleri ve enerji aralıkları arttıkça değişen düğüm sayılarına

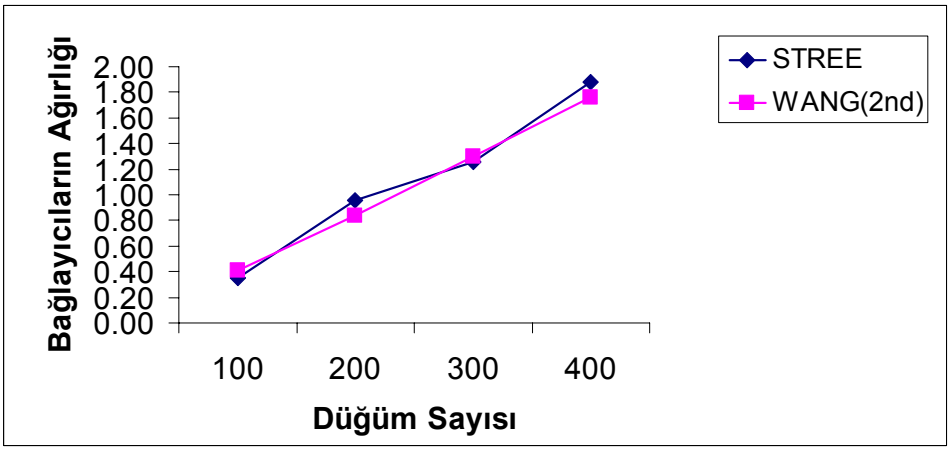
rağmen bağlayıcı ağırlıkları azalmaktadır. Düğüm derecesi arttıkça maliyeti az olan düğümlerle maliyeti çok olan düğümler arasında daha fazla bağlantı olmasını algoritmamız etkin kullanmıştır. Ayrıca enerji aralıkları arttıkça maliyeti az olan düğüm sayısı artmasını da algoritmamız etkin kullanmış, daha az maliyetli düğümleri seçmiştir.



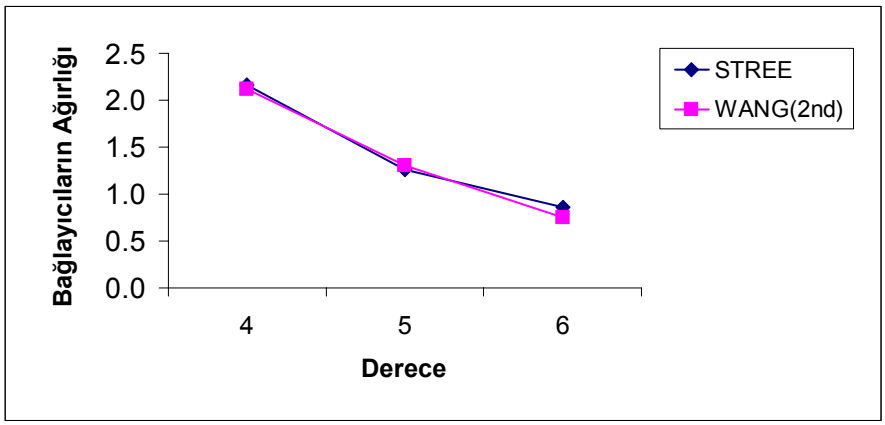
Şekil 4.37 STREE algoritmasının değişen derecelere göre bağlayıcı ağırlıkları



Şekil 4.38 STREE algoritmasının değişen enerji aralıklarına göre bağlayıcı ağırlıkları

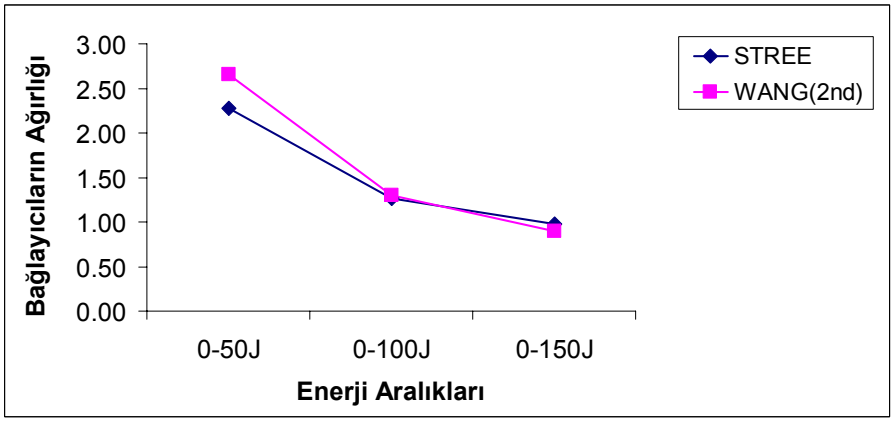


Şekil 4.39 STREE algoritması ve WANG(2nd)'ın bağlayıcı ağırlıklarının düğüm sayısına göre karşılaştırılması



Şekil 4.40 STREE algoritması ve WANG(2nd)'ın bağlayıcı ağırlıklarının düğüm derecelerine göre karşılaştırılması

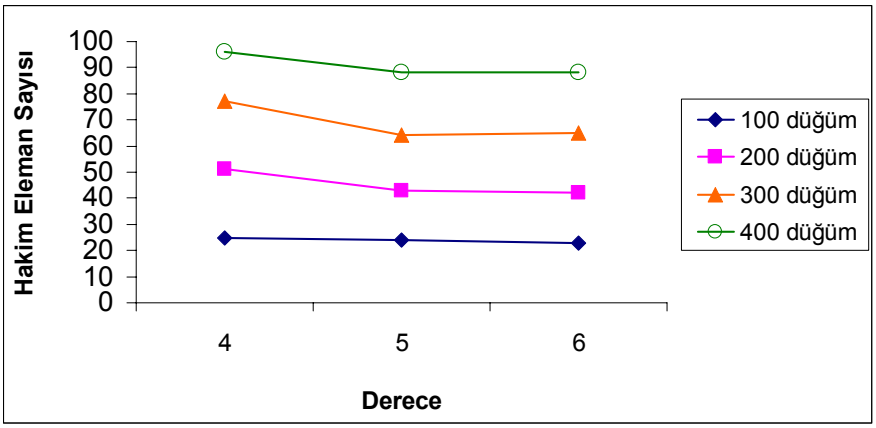
Şekil 4.39, Şekil 4.40, ve Şekil 4.41 STREE ve WANG'ın ikinci safhasının seçtiği bağlayıcıların ağırlıklarının karşılaştırılması görülmektedir. Değişen düğüm sayılarına, düğüm derecelerine ve enerji aralıklarına rağmen STREE ve WANG'ın algoritmaları yaklaşık sonuçlar vermektedir. İki algoritmanın bulduğu bağlayıcıların ağırlıkları yaklaşıktır.



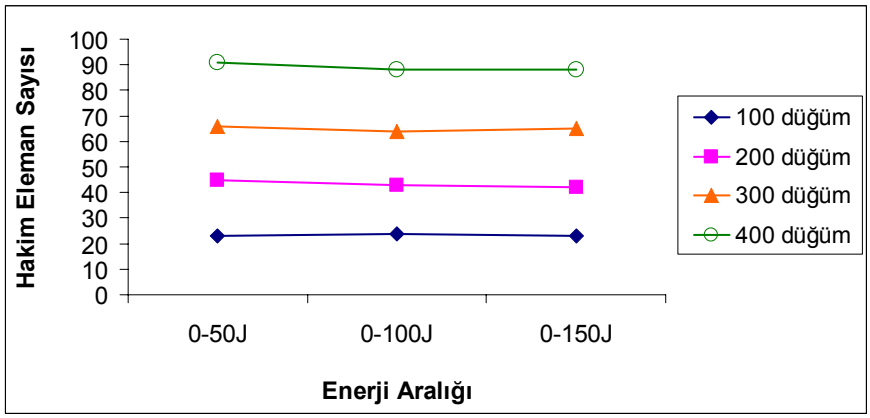
Şekil 4.41 STREE algoritması ve WANG(2nd)'ın bağlayıcı ağırlıklarının enerji aralıklarına göre karşılaştırılması

4.4.3. Hâkim Düğümlerin Sayısı

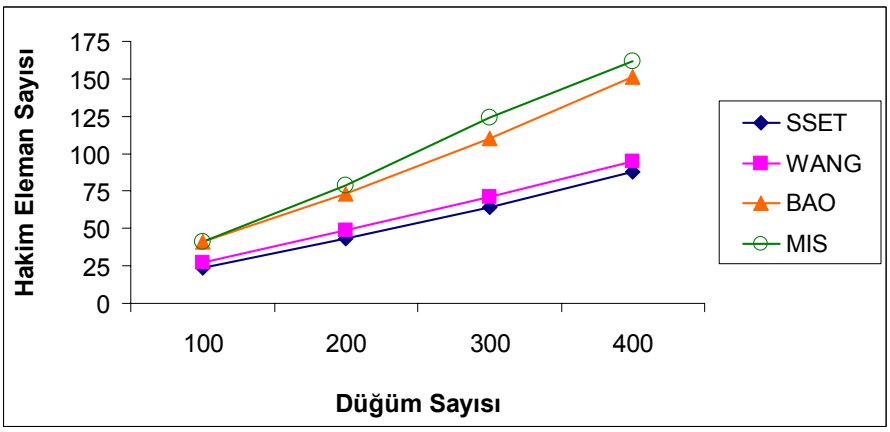
HK içindeki eleman sayısının olabildiğince düşük olması genel olarak HK algoritmalarının önemli hedeflerindendir. Ağdaki düğümlerden olabildiğince en azını HK içine almak gerekir. Şekil 4.42 ve Şekil 4.43'de SSET algoritmasının ortalama düğüm derecesine ve düğüm enerjisine bağlı olarak ürettiği HK içindeki eleman sayısı verilmiştir. Ortalama olarak SSET, toplam düğüm sayısının en fazla dörtte biri kadar hâkim eleman üretir. SSET'in ürettiği HK içindeki eleman sayısı, ortalama düğüm derecesi arttıkça düzenli ve küçük bir eğimle azalır. Bunun sebebi düğümler arasındaki bağlantı arttıkça HK içine alınan bir düğümün daha çok sayıda sıradan düğüme bağlanmasıdır. Enerji aralığı arttıkça SSET'in ürettiği HK içindeki eleman sayısı kararlı değerler gösterir.



Şekil 4.42 SSET'de ortalama düğüm derecesine bağlı hâkim eleman sayısı



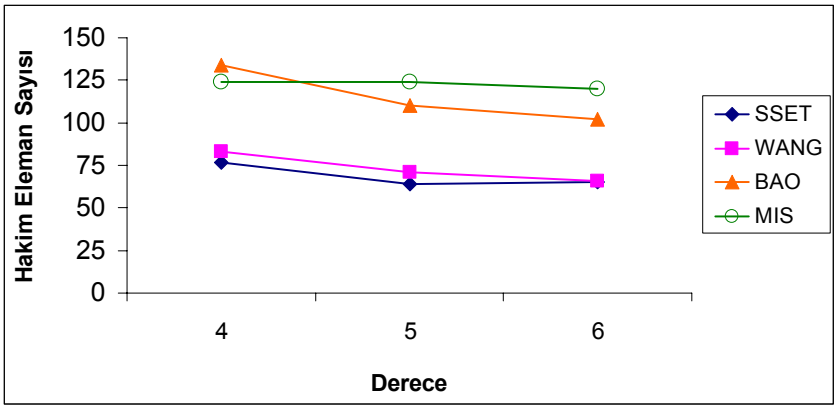
Şekil 4.43 SSET'de düğümlerin enerji değişimlerine bağlı hâkim eleman sayısı



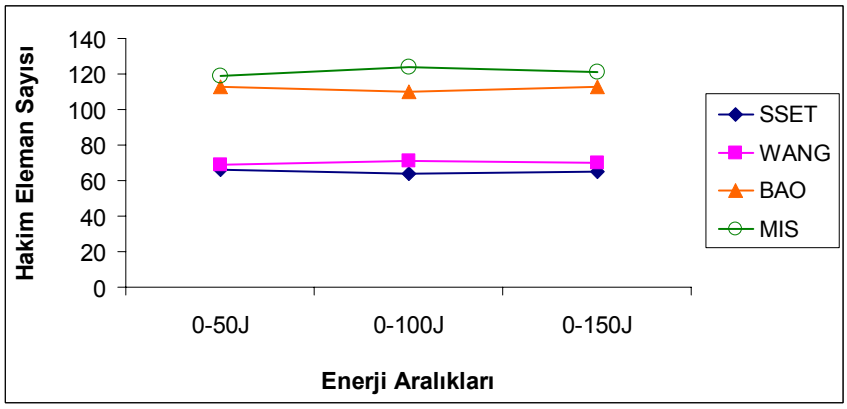
Şekil 4.44 Düğüm sayısına bağlı algoritmaların ürettiği hâkim eleman sayısı

Şekil 4.44, Şekil 4.45 ve Şekil 4.46'te algoritmaların seçtiği hâkim eleman sayılarının karşılaştırılmaları gösterilmiştir. Şekil 4.44'de değişen düğüm sayılarına rağmen SSET diğer algoritmalardan daha az sayıda hâkim eleman ürettiği görülmektedir. Wang'ın algoritması, SSET'e yakın sayıda eleman üretmiştir, MIS algoritması en kötü performansı gösterirken BAO'nun algoritması MIS'e yakın sayıda eleman üretmiştir. Şekil 4.45'de değişen düğüm derecelerine rağmen algoritmaların performans sıralamaları aynı olup MIS dışındaki algoritmalar artan düğüm dereceleriyle birlikte daha az sayıda hâkim eleman seçerler. Şekil 4.46'da algoritmaların enerji aralıkları değişimlerine bağlı olarak ürettikleri hâkim

elemanların sayısı görülmektedir. SSET algoritması bu şartlarda da en iyi performansı gösterirken diğer algoritmaların sıralaması aynı kalmış, algoritmaların performansını enerji aralıkları etkilememiştir.



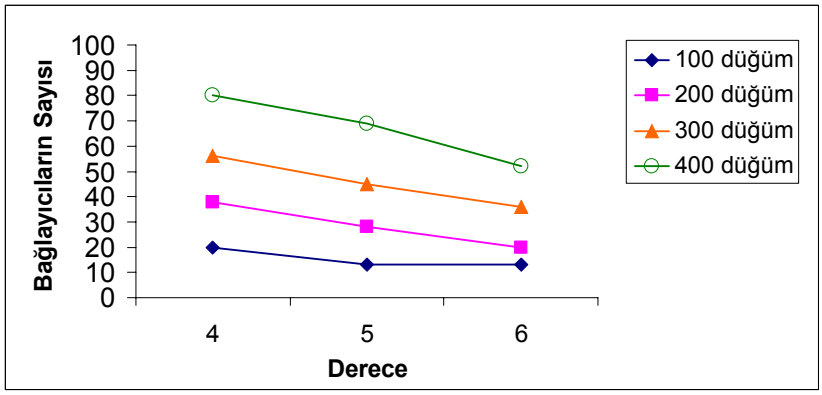
Şekil 4.45 Dereceye bağlı algoritmaların ürettiği hâkim eleman Sayısı



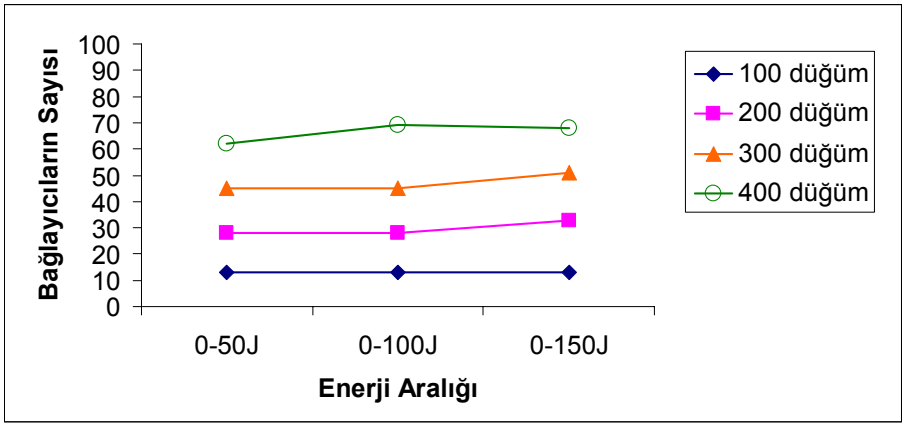
Şekil 4.46 Düğümlerin enerji değişimlerine bağlı algoritmaların ürettiği hâkim eleman sayısı

4.4.4. Bağlayıcı Düğümlerin Sayısı

Bağlayıcı düğümlerin ağırlıkları gibi bağlayıcı düğümlerin sayısı da ağırlıklı bağlı hâkim küme algoritmalarını incelerken önemli bir parametredir. Şekil 4.47 ve Şekil 4.48'te STREE algoritmasının değişen düğüm derecelerine ve enerji aralıklarına göre bağlayıcıların sayıları görülmektedir.

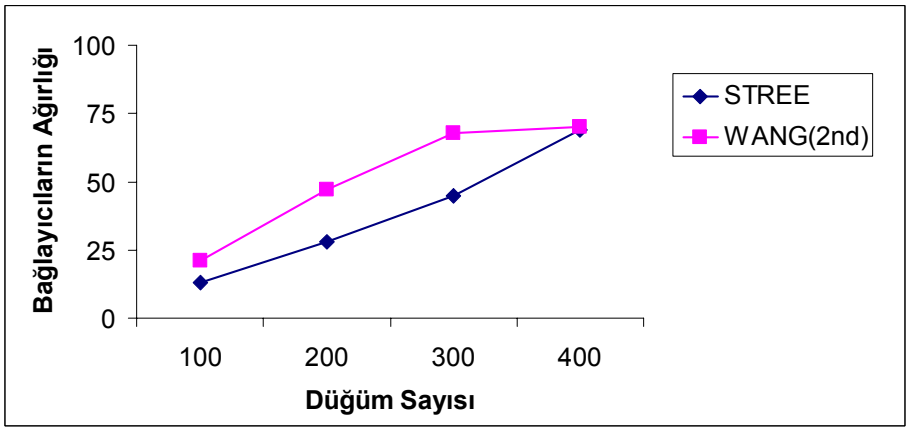


Şekil 4.47 STREE algoritmasının değişen derecelere göre bağlayıcı sayıları

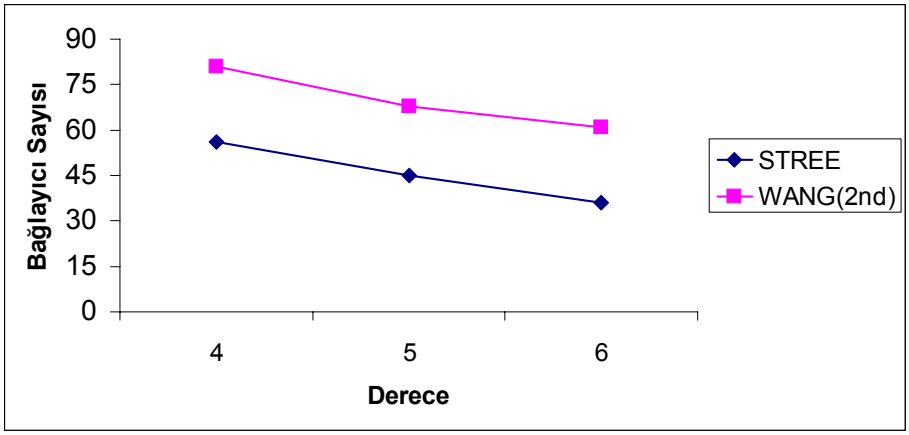


Şekil 4.48 STREE algoritmasının değişen enerji aralıklarına göre bağlayıcı sayıları

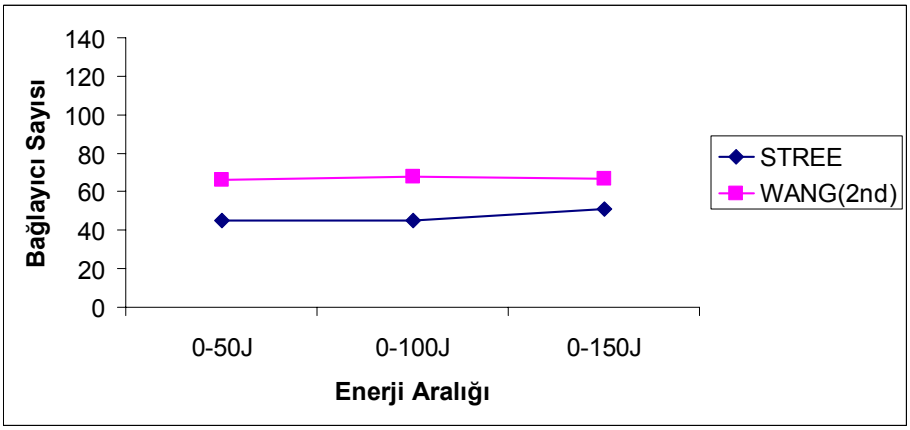
Artan düğüm dereceleriyle birlikte düğümlerin sayısı düzenli olarak azalmıştır. Bunun sebebi seçilen bir bağlayıcıyla birlikte daha fazla düğümün örtülebilmesidir. Artan enerji aralıklarıyla birlikte benzer sayıda bağlayıcı elde edilmiş, önceki bölümde anlatıldığı üzere bağlayıcıların ağırlığı azalmıştır. Şekil 4.49, Şekil 4.50 ve Şekil 4.51'de STREE ve WANG'ın ikinci safhasının karşılaştırılması verilmiştir. Artan düğüm sayıları, değişen düğüm derecesi ve enerji aralıklarına rağmen STREE algoritması WANG'ın algoritmasına göre daha az sayıda bağlayıcı seçmiştir. Önceki bölümde gösterildiği üzere STREE ve WANG'ın algoritmaları yaklaşık ağırlıkta bağlayıcı bulmalarına rağmen STREE algoritması daha az sayıda hâkim eleman bulmuştur.



Şekil 4.49 STREE algoritması ve WANG(2nd)'ın bağlayıcı sayılarının düğüm sayılarına göre karşılaştırılması



Şekil 4.50 STREE algoritması ve WANG(2nd)'ın bağlayıcı sayılarının düğüm derecelerine göre karşılaştırılması

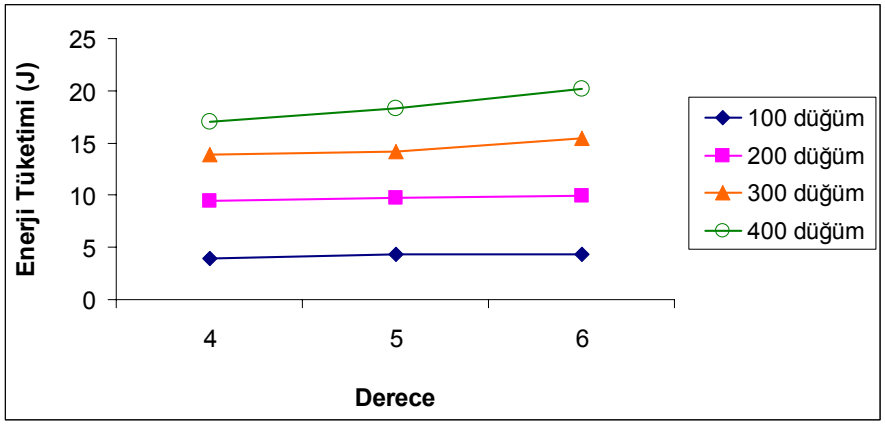


Şekil 4.51 STREE algoritması ve WANG(2nd)'ın bağlayıcı sayılarının enerji aralıklarına göre karşılaştırılması

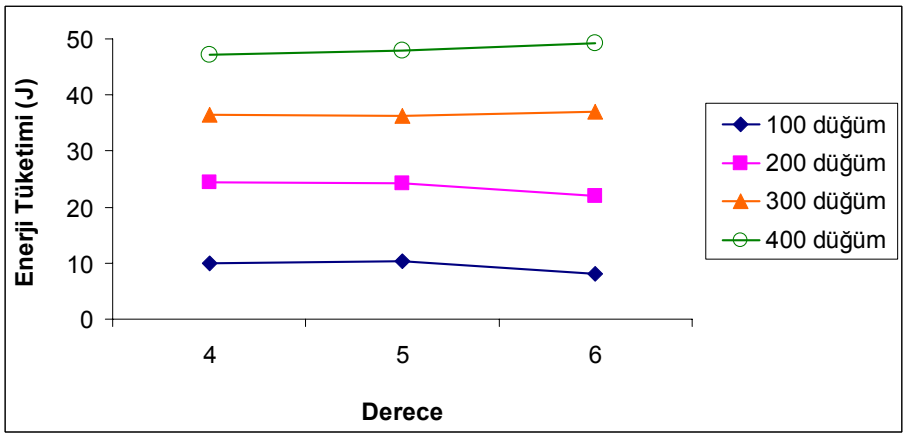
4.4.5. Tüketilen Enerji Miktarları ve Mesaj Gönderimleri

Bu bölümde kodlanan bütün algoritmalarının tükettiği enerji miktarları ve gönderilen toplam mesaj sayılırı ölçülmüştür. Şekil 4.52'de SSET'in Şekil 4.53'de ASYNSET algoritmasının değişen düğüm derecelerine enerji tüketimleri ölçülmüştür. Bu şekillerde görüldüğü üzere düğüm sayısı arttıkça harcanan enerji düzenli olarak artmış, değişen düğüm derecelerine rağmen enerji tüketimi yaklaşık kalmıştır. Şekil 4.54'de ağırlıklı hâkim küme algoritmalarının enerji harcamaları verilmiştir. En az enerji harcayan algoritma MIS algoritmasıdır, daha sonra sırayla BAO, WANG, SSET, ASYNSET ve FLOODSET algoritmalarıdır. SSET algoritması WANG'ın algoritmasına göre yaklaşık 10 kat fazla enerji harcamış olmasının yanında FLOODSET algoritmasından yaklaşık 100 kat daha az enerji harcamıştır. ASYNSET algoritmasının SSET algoritmasından daha fazla enerji harcamasının sebebi her raunt senkronizasyon işlemi yapılması ve her raunt fazladan *BAĞLANMADI* mesajlarının gönderilmesidir. MIS, BAO, WANG, SSET algoritmalarının enerji harcamalarına dikkat edilirse algoritmalara daha fazla kural eklenip hâkim kümenin ağırlığı azaltıldıkça ve algoritmalar daha kaliteli sonuçlar

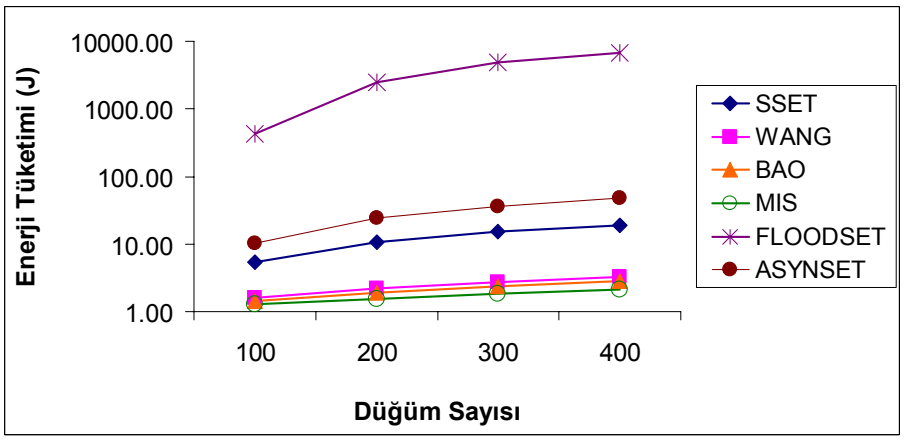
buldukça daha fazla enerji harcarlar. Şekil 4.55 ASYNSET algoritmasının mesaj gönderimi verilmiştir. Bu sonuçlar Şekil 4.52'de verilen enerji tüketim sonuçlarına çok benzerdir. Yine benzer olarak Şekil 4.56'da verilen algoritmaların gönderilen mesaj sayıları Şekil 4.53'de verilen algoritmaların enerji harcamaları sonuçlarına benzerdir.



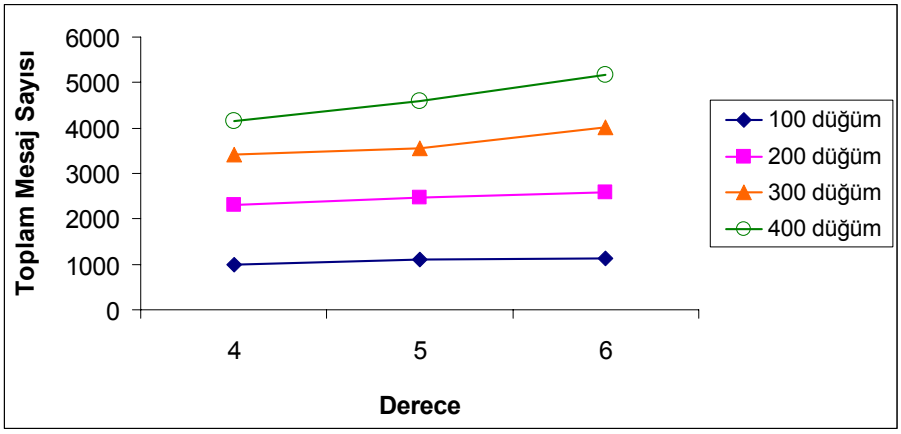
Şekil 4.52 SSET algoritmasının değişen derecelerine göre enerji tüketimi



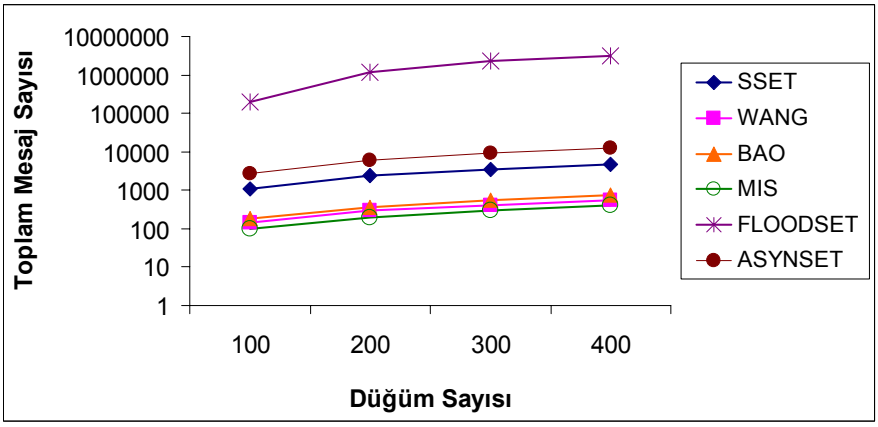
Şekil 4.53 ASYNSET algoritmasının değişen derecelere göre enerji tüketimi



Şekil 4.54 Hâkim küme algoritmalarının değişen düğüm sayılarına göre enerji tüketimleri

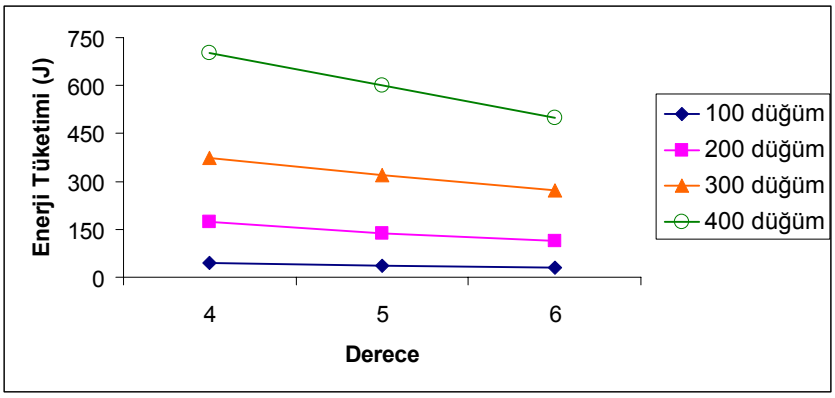


Şekil 4.55 ASYNSET algoritmasının değişen düğüm derecesine göre mesaj gönderim sayıları

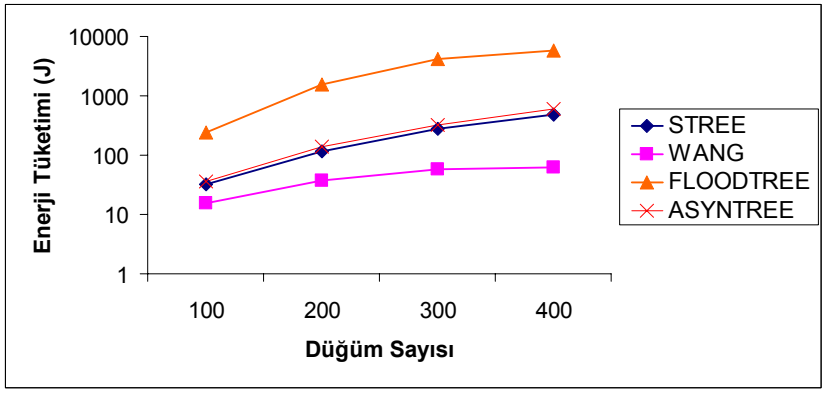


Şekil 4.56 Hâkim küme algoritmalarının değişen düğüm sayılarına göre mesaj gönderim sayıları

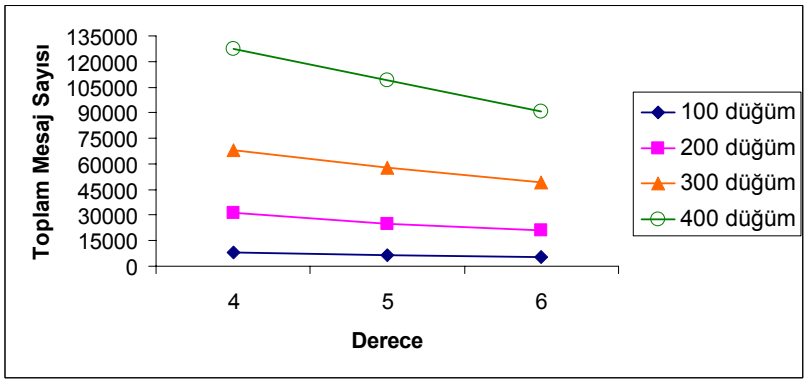
ASYNTREE algoritmasının değişen düğüm sayıları ve değişen düğüm derecelerine göre enerji harcamaları Şekil 4.57'de görülmektedir. ASYNTREE algoritmasında düğüm dereceleri arttıkça bağlayıcı sayıları azalır, bağlayıcı sayısı azaldığı için raunt sayısı da azalır ve dolayısıyla mesaj sayısı azalır, bundan dolayı enerji tüketimleri azalır. Düğüm sayısı artıkça dikkat edileceği üzere enerji tüketiminin $O(N^2)$ mesaj karmaşıklığına uygun olarak artığını görüyoruz. Şekil 4.58'de ASYNTREE, FLOODTREE, WANG ve STREE algoritmalarının enerji tüketimlerini görmekteyiz. WANG'ın algoritması en az enerjiyi harcarken sırayla STREE, ASYNTREE ve FLOODTREE algoritmalarının olduğunu görmekteyiz. Şekil 4.59'da ASYNTREE algoritmasının mesaj harcaması verilmiştir. Bu algoritmanın harcadığı mesaj sayısı, Şekil 4.57'de verilen enerji tüketimine benzerdir. Şekil 4.60'da verilen algoritmaların mesaj gönderimleri Şekil 4.58'de verilen algoritmaların enerji tüketimleriyle benzerdir.



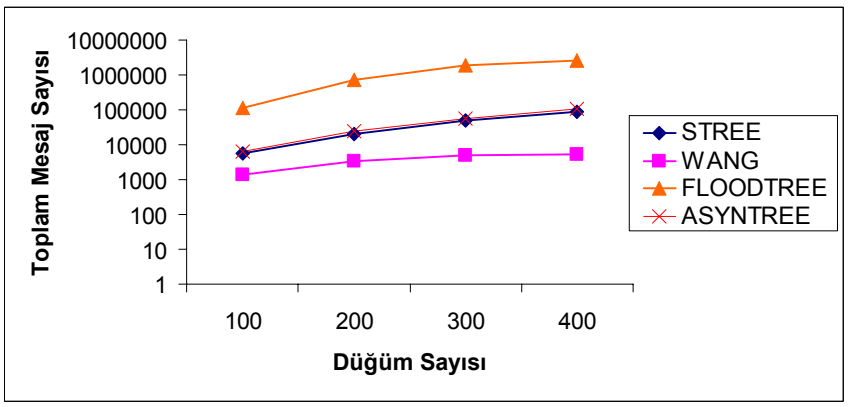
Şekil 4.57 ASYNTREE algoritmasının değişen derecelere göre enerji tüketimi



Şekil 4.58 Steiner ağacı algoritmalarının değişen düğüm sayılarına göre enerji tüketimleri

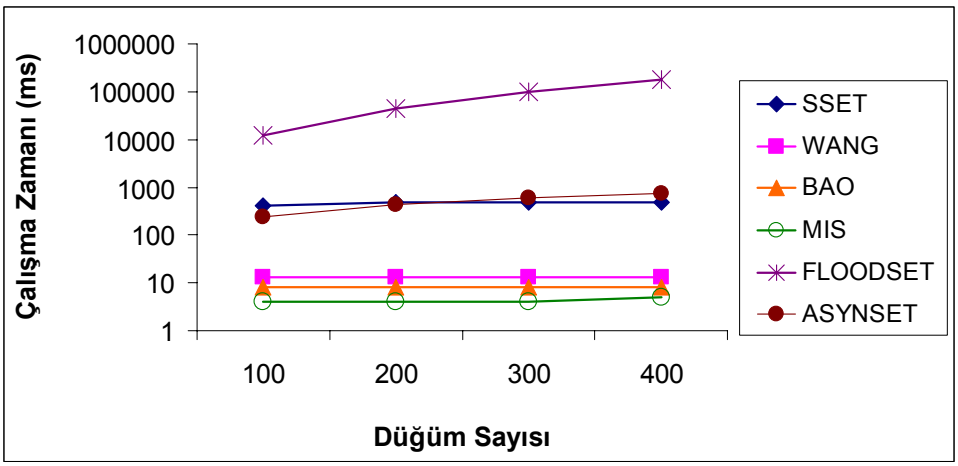


Şekil 4.59 ASYNTREE algoritmasının değişen düğüm derecesine göre mesaj gönderim sayısı

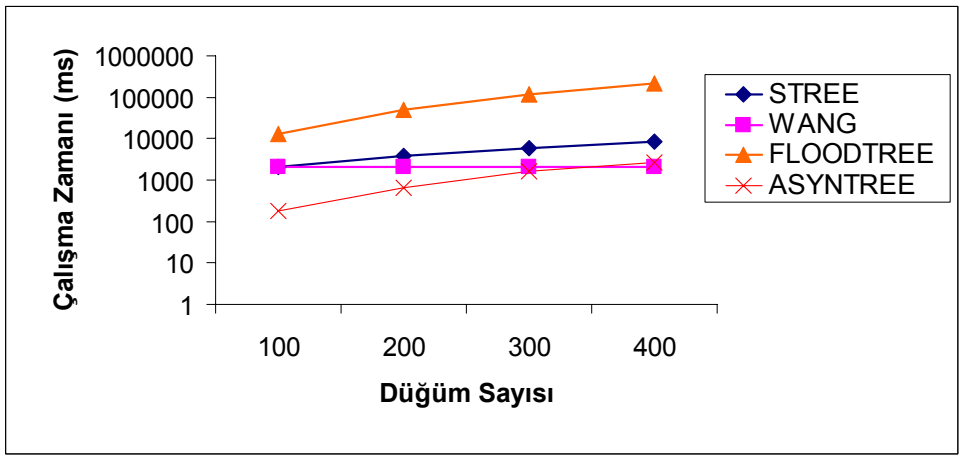


Şekil 4.60 Steiner ağacı algoritmalarının değişen düğüm sayılarına göre mesaj gönderim sayıları

4.4.6. Çalışma Zamanları



Şekil 4.61 Hâkim küme algoritmalarının değişen düğüm sayılarına göre çalışma zamanları



Şekil 4.62 Hâkim küme algoritmalarının değişen düğüm sayılarına göre çalışma zamanları

Bu bölümde benzetimini yaptığımız bütün algoritmaların süre kullanımları verilecektir. Şekil 4.61'de sırayla MIS, BAO, WANG, SSET-ASYNSET ve FLOODSET algoritmalarının süre kullandığını görmekteyiz. MIS, BAO, WANG ve SSET algoritmalarına bakılacak olursa enerji tüketimlerinde yaptığımız yorumun çok benzerine yapabiliriz; algoritmalara ek kurallar eklendikçe ve algoritmalar daha kaliteli sonuçlar buldukça kullandıkları süre artar. SSET ve ASYNSET'in süre kullanımlarının yaklaşık aynıdır. SSET algoritmasında düğümler işlemlerini bitirdikten sonra yeni raunda geçene kadar boş süre harcasalar da ASYNSET algoritmasında bu süre senkronizasyon için kullanılmıştır. FLOODSET algoritmasının bu algoritmalara göre çok daha fazla süre harcadığını görmekteyiz. Şekil 4.62'de ağırlıklı Steiner ağacı algoritmalarının süre kullanımları verilmiştir. FLOODTREE algoritması en yüksek zamanı kullanırken, ASYNTREE algoritması en iyi performansı gösterip WANG'dan ve STREE'den daha az süre harcamıştır.

4.5. Tartışmalar ve Uygulamalar

Performans değerlendirmesi bölümündeki benzetim sonuçlarından görüldüğü üzere önerdiğimiz algoritmalar, toplamda WANG'ın algoritmasından daha az ağırlıklı ve daha az sayıda elemanlı omurgalar bulurken, daha çok enerji harcamıştır. Bu bölümde tasarladığımız senkron ve asenkron algoritmaların ve WANG'ın algoritmasının üzerine bir uygulama tanımlayıp bu uygulamanın sonuçlarına göre enerji maliyeti analizi (*break-even analysis*) göstereceğiz. Bu uygulamayı tanımlamadan önce omurganın ve kümelerin nasıl oluştuğunu anlatalım. Analizimizi yapabilmemiz için 3 tip ağırlıklı bağlı hâkim küme omurgası oluşturmamız gereklidir. Bu omurgalar sırasıyla

1. Senkron algoritmalar ile oluşturulan omurga: Bu omurganın oluşturulması için SSET ve STREE algoritmaları sırayla çalıştırılır.

2. Asenkron algoritmalar ile oluşturulan omurga: Bu omurganın oluşturulması için ASYNSET ve ASYNTREE algoritmaları sırayla çalıştırılır. Bu algoritmaların oluşturacağı omurga senkron algoritmaların oluşturacağı omurga ile aynıdır, fakat tüketilen enerji miktarı daha fazladır.

3. WANG'ın algoritması ile oluşturulan omurga: Bu omurganın oluşturulması için WANG'ın algoritmasının iki safhası arka arkaya çalıştırılır.

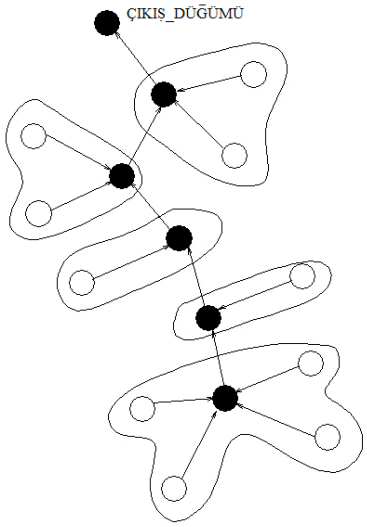
Algoritmalar çalıştırılıp omurgalar oluşturulduktan sonra omurga *ÇIKIŞ_DÜĞÜMÜ'*ne doğru yönlendirilir ve kümeler oluşur. Bunun sebebi sıradan düğümlerden toplanan verilerin birleştirilerek omurga düğümleri tarafından birleştirilerek *ÇIKIŞ_DÜĞÜMÜ'*ne doğru iletilmesi için bir patikanın tanımlanmasıdır. Bu işlem Şekil 4.63'de verilmiştir. Bu işlemin yapılması için *ÇIKIŞ_DÜĞÜMÜ'*nden sıradan düğümlere doğru *PATİKA_KUR* mesajı gönderilir. Bu mesajı alan düğüm bir omurga düğümüyse ve mesajı ilk kez aldıysa, mesajın kaynağını kendi ebeveyni olarak seçer ve bu mesajı komşularına iletir. Bu mesajı alan düğüm sıradan bir düğümse, mesajı ilk kez aldıysa ve mesajın kaynağı bir omurga düğümüyse bu mesajın kaynağını lideri olarak kaydeder ve *BAĞLANDI* mesajının içinde liderin kimliğini koyarak komşularına iletir.

```
PATİKA_KUR mesajını aldıktan sonra
Eğer PATİKA_KUR'u daha önceden almadıysam
  Eğer mesajın kaynağı HAKİM ELEMAN 'sa
    Eğer SIRADAN ELEMAN 'sam
      küme_lideri ← mesajın kaynağı
```

```
        BAĞLANDI mesajını ilet
      Değilse
        omurga_üzerindeki_ebeveynim ← mesajın kaynağı
        PATİKA_KUR gönder
      Koşulu bitir
    Koşulu bitir
Koşulu Bitir

BAĞLANDI mesajını aldıktan sonra mesajın kaynağını çocuklarım dizinine ekle
```

Şekil 4.63 *ÇIKIŞ_DÜĞÜMÜ*'ne doğru patikanın ve kümelerin oluşması için çalışan algoritmalar



Şekil 4.64 SSET ve STREE 'in birlikte oluşturdukları örnek bir omurga ve kümeler

Sıradan bir düğümünden *BAĞLANDI* mesajını alan omurga düğümü eğer içinde lider kimliği olarak kendi kimliğini görürse küme elemanlar içine kaydediyor. Dikkat edilirse her düğüm sadece bir mesaj göndererek kümeleri ve patikaları oluşturuyor. Böylece Şekil 4.64'daki küme ve omurga mimarisi ortaya çıkar. Aslında önerdiğimiz algoritmalar tarafından *ÇIKIŞ_DÜĞÜMÜ*'ne doğru patikalar Steiner ağacı yapılırken tanımlanmış olsa da kümelerin oluşması için bu aşamanın yapılması gerekir.

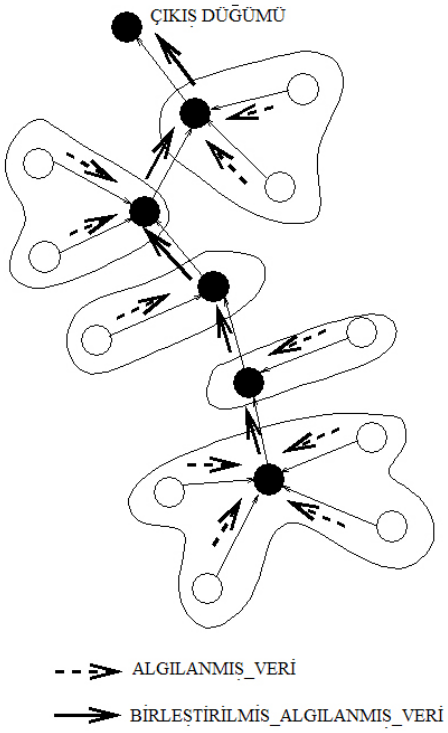
```
Uygulama başladığında veya zamanlayıcı dolduğunda
Eğer sıradan düğümsen
   ALGILANMIŞ_VERİ mesajını küme_lideri'ne gönder
   Zamanlayıcıyı çalıştır
Değilse
   Çocuklardan ALGILANMIŞ_VERİ mesajını bekle
   Eğer ÇIKIŞ_DÜĞÜMÜ'ysem BİRLEŞTİRİLMİŞ_ALGILANMIŞ_VERİ
mesajını omurga üzerindeki ebeveynine gönder
   Koşulu bitir
Koşulu bitir
```

ALGILANMIŞ_VERİ mesajı alındığında
Veriyi kaydet
Bütün çocuklardan *ALGILANMIŞ_VERİ* mesajı toplandığında veriyi birleştir

BİRLEŞTİRİLMİŞ_ALGILANMIŞ_VERİ mesajı alındığında
Mesajı omurga üzerindeki ebeveynine gönder

Şekil 4.65 Sıradan düğümlerin ve liderlerin mesajları göndermek için çalıştırdığı algoritma

Kümeler, omurga ve patikalar oluşturulduktan sonra düğümler uygulamayı çalıştırmaya başlıyorlar. Bu uygulamada her düğüm periyodik olarak çevreden bir algı yaparlar ve bu veriyi *ALGILANMIŞ_VERİ* mesajının içinde küme liderlerine gönderiyorlar. Küme liderleri bütün küme üyelerinden *ALGILANMIŞ_VERİ* mesajını topladıktan sonra bu veriyi omurga üzerinden *BİRLEŞTİRİLMİŞ_ALGILANMIŞ_VERİ* mesajı olarak omurga üzerinden iletiyor. Uygulamanın kodları Şekil 4.65'de verilmiştir. Şekil 4.66'de örnek bir çalışma gösterilmiştir. Önerdiğimiz uygulama bir akıllı tarım uygulamasına örnek olarak gösterilebilir. Bu uygulamada düğümler topraktan periyodik olarak aldıkları ölçümleri önce küme liderlerine gönderebilirler, küme liderleri de bu mesajları omurga üzerinden iletebilir.

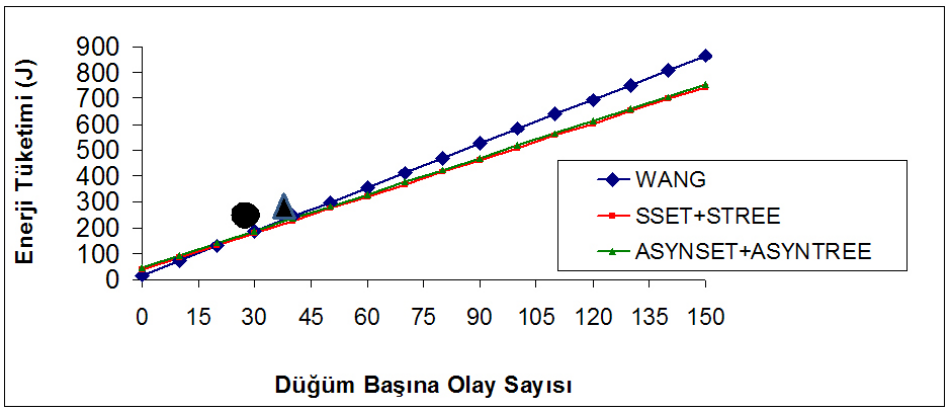


Şekil 4.66 Sıradan düğümlerin ve liderlerin mesajlarını göndermesi

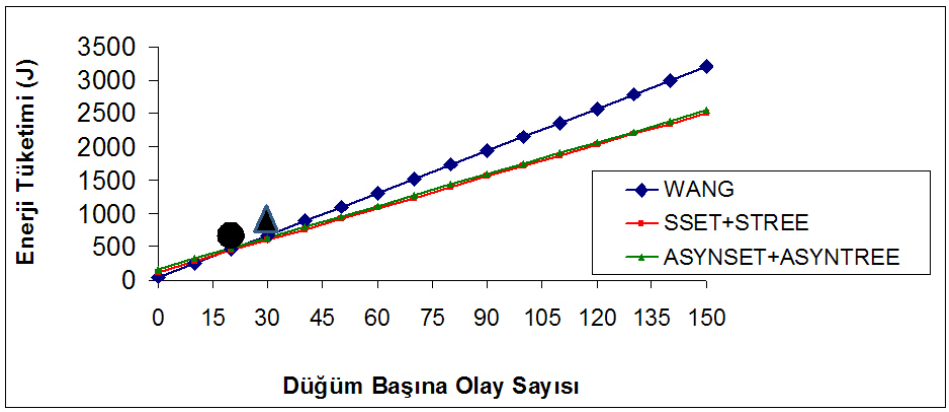
Performans değerlendirmesini yaparken değişen düğüm sayılarına göre, düğüm derecelerine göre ve değişen enerji aralıklarına göre ölçtük. Ölçümler yapılırken omurgaların oluşturulması için tüketilen enerji miktarını sistemin

kurulması için gereken maliyet olarak kabul ettik. Bir periyot içinde her düğüm uygulamayı çalıştırır ve uygulamanın düğümlere getirdiği enerji maliyetini toplam tüketilen enerjiye ekledik. Böylece toplam enerji maliyeti sistemin kurulması için gerekli maliyet ve her periyotta düğümlerin harcadığı toplam enerji olarak hesaplandı.

Değişen düğüm sayılarına göre performans sonuçları: Şekil 4.67 ve Şekil 4.68, Şekil 4.69'de değişen düğüm sayılarına göre uygulamanın bu omurgalar üzerinde kullandığı enerji miktarları verilmiştir. Şekillerin üzerinde siyah içi dolu daire ile gösterilen nokta WANG'ın enerjisinin senkron omurgaların enerji tüketimini yakaladığı noktayken siyah üçgenle gösterilen nokta WANG'ın enerjisinin asenkron omurgaların enerji tüketimini yakaladığı noktadır. Şekil 4.67'de 100 düğümlük ve ortalama 5 derecelik bir ağ üzerinde omurgaların enerji harcaması verilmiştir. Yaklaşık düğüm başına 30. olayda (event) WANG ile oluşturulan omurganın harcadığı toplam enerjinin SSET ve STREE ile oluşturulan omurganın toplam harcadığı enerjiyi geçtiğini gördük. Benzer bir şekilde yaklaşık 40. olaydan sonra WANG'ın algoritmasının ASYNSET ve ASYNTREE'den daha fazla enerji tükettiğini görmekteyiz.

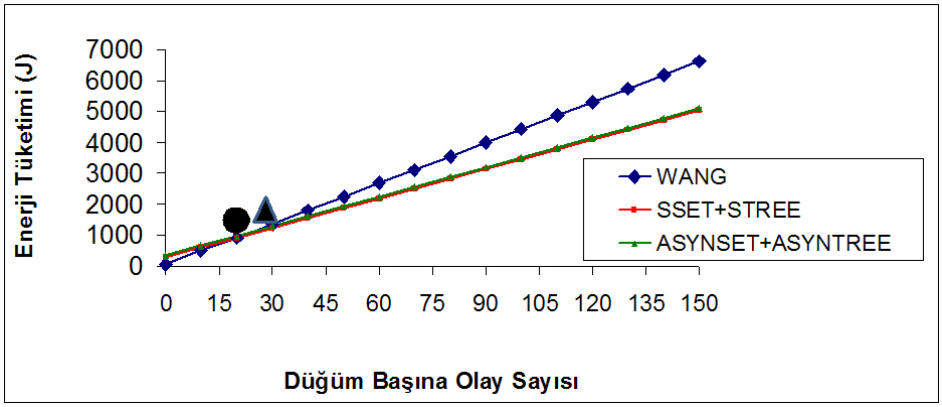


Şekil 4.67 100 düğümlük, ortalama derecesi 5 ve enerji aralığı 0-100 olan ağ üzerindeki enerji ölçümleri



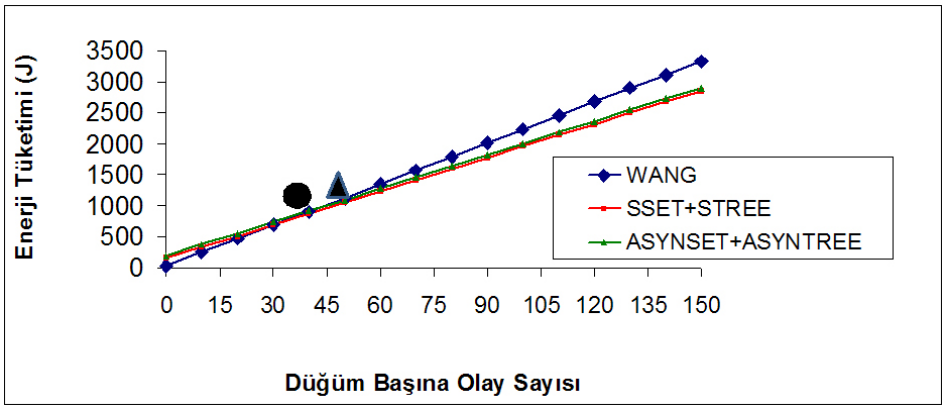
Şekil 4.68 200 düğümlük, ortalama derecesi 5 ve enerji aralığı 0-100 olan ağ üzerindeki enerji ölçümleri

Şekil 4.68'de 200 düğümlük ve ortalama 5 derecelik bir ağ üzerinde omurgaların enerji harcaması verilmiştir. Bu grafik üzerinde WANG'ın algoritmasının enerji harcaması yaklaşık 20. olaydan sonra SSET ve STREE'yi, 30. olaydan sonra da ASYNSET ve ASYNTREE'yi geçer. Şekil 4.69'de 300 düğümlük ve 5 derecelik bir ağ üzerinde omurgaların enerji harcamaları verilmiştir. Bu grafik üzerinde de WANG'ın algoritmasının enerji harcaması yaklaşık 20. olaydan sonra SSET ve STREE'yi, 30. olaydan sonra ASYNSET ve ASYNTREE'yi geçer. Bu grafikleri inceledikten sonra önerdiğimiz algoritmaların örnek uygulama üzerinde daha az enerji harcadığını ve düğüm sayısı arttıkça WANG'ın algoritmasının enerji tüketiminin önerdiğimiz omurgaların enerji tüketimlerini daha az olay tespitinden sonra yakaladığını görmekteyiz. Böylece düğüm sayısı arttıkça önerdiğimiz omurga oluşturma algoritmalarının WANG'ın algoritmasına oranla daha avantajlı olduğunu söyleyebiliriz.



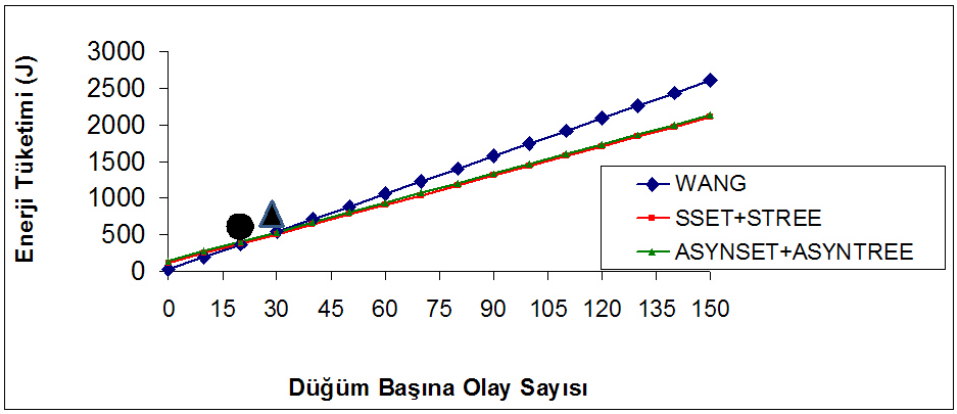
Şekil 4.69 300 düğümlük, ortalama derecesi 5 ve enerji aralığı 0-100 olan ağ üzerindeki enerji ölçümleri

Değişen düğüm derecelerine göre performans sonuçları: Şekil 4.68, Şekil 4.70 ve Şekil 4.71'da değişen düğüm derecelerine göre omurganın, uygulamayla birlikte toplam kullandığı enerji miktarları verilmiştir.



Şekil 4.70 200 düğümlük, ortalama derecesi 4 ve enerji aralığı 0-100 olan ağ üzerindeki enerji ölçümleri

Şekil 4.70'de 200 düğümlük ve ortalama derecesi 4 olan bir ağ üzerinde omurgaların uygulamayla birlikte toplam enerji tüketimleri gösterilmiştir. Bu ağ üzerinde WANG'ın algoritmasının enerji tüketimi SSET ve STREE algoritmalarını yaklaşık 40. olaydan sonra, ASYNSET ve ASYNTREE algoritmaları 50. olaydan sonra geçer. Şekil 4.68'da önceki bölümde de anlatılan 200 düğümlük ve ortalama derecesi 5 olan ağ üzerindeki enerji tüketimi verilmiştir. Bu ağ üzerinde WANG 'ın algoritması SSET ve STREE'yi yaklaşık 20. olaydan sonra, ASYNSET ve ASYNTREE'yi de yaklaşık 30. olaydan sonra yakalar. Şekil 4.71'da 200 düğümlük ve ortalama düğüm derecesi 6 olan ağ üzerinde algoritmaların enerji tüketimleri gösterilmiştir. Bu ağ üzerinde WANG'ın algoritması yaklaşık 25. olaydan sonra SSET ve STREE'yi geçmiş, 30. olaydan sonra da ASYNSET ve ASYNTREE'yi geçmiştir. Bu grafikleri inceledikten sonra önerdiğimiz algoritmaların örnek uygulama üzerinde düğüm derecesi arttıkça WANG'ın algoritmasının enerji tüketimine göre daha avantajlı olduğunu söyleyebiliriz.

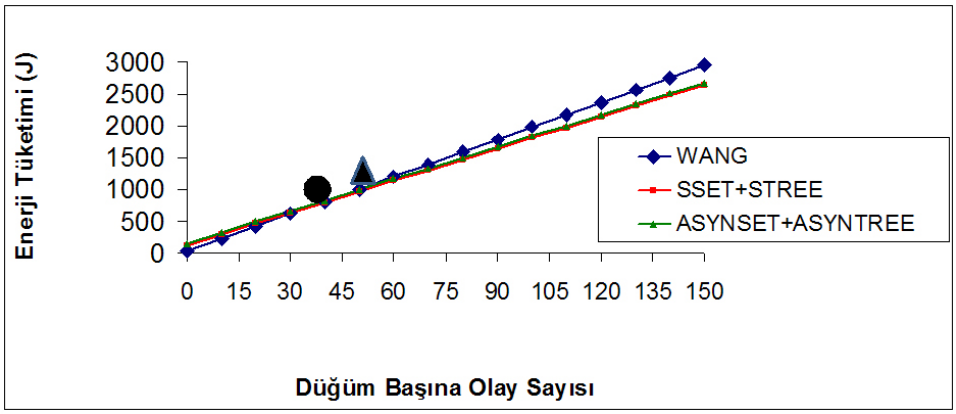


Şekil 4.71 200 düğümlük, ortalama derecesi 6 ve enerji aralığı 0-100 olan ağ üzerindeki enerji ölçümleri

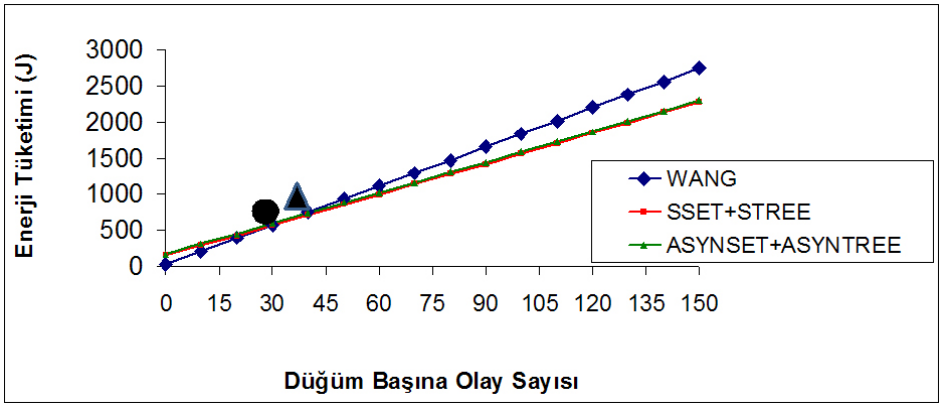
Değişen enerji aralıklarına göre performans sonuçları: Şekil 4.68, Şekil 4.72 ve Şekil 4.73'de değişen düğüm enerji aralıklarına göre enerji tüketimleri verilmiştir. Şekil 4.72'da 200 düğümlük ve ortalama derecesi 5 olan bir ağ üzerinde düğüm enerji aralıkları 0-50J olduğunda enerji tüketimleri gösterilmiştir. WANG'ın algoritmasının enerji tüketimi SSET ve STREE'yi yaklaşık 40. olaydan, ASYNSET ve ASYNTREE'yi 50. olaydan sonra geçer.

Şekil 4.68'da daha önceki iki alt bölümde değindiğimiz üzere düğüm enerji aralıkları 0-100J olduğunda WANG 'ın algoritması SSET ve STREE'yi yaklaşık 20. olaydan sonra, ASYNSET ve ASYNTREE'yi de yaklaşık 30. olaydan sonra yakalar. Şekil 4.73'de 200 düğümlük ve ortalama derecesi 5 olan ve enerji aralıkları 0-150J bir ağ üzerindeki enerji tüketimleri verilmiştir. Bu ağ üzerinde WANG 'ın algoritması SSET ve STREE'yi yaklaşık 30. olaydan sonra, ASYNSET ve ASYNTREE'yi de yaklaşık 40. olaydan sonra yakalar. Sonuç olarak enerji

aralıklarını 0-50J'den arttıkça önerdiğimiz algoritmalar daha kaliteli omurgalar üreteceklerinden dolayı WANG'ın algoritmasının enerji tüketiminin daha az olay tespitinden sonra önerdiğimiz algoritmaları geçebileceğini söyleyebiliriz. Genel sonuç olarak önerdiğimiz algoritmalar WANG'ın algoritmasından daha maliyetli olmasına rağmen üzerinde örnek bir TDA uygulaması için daha az enerji harcayacağını ve uygulama çalıştıkça maliyeti yüksek olan algoritmalarımızın maliyetinin WANG'ın algoritmasının maliyetinden daha düşük olacağını gösterdik. Artan düğüm sayılarına, düğüm derecelerine ve enerji aralıklarına karşın algoritmamızın daha kaliteli omurga oluşturduğundan dolayı üzerinde çalışan uygulamanın daha az enerji harcadığını ve maliyetin daha çabuk karşılandığını gösterdik.



Şekil 4.72 200 düğümlük, ortalama derecesi 5 ve enerji aralığı 0-50 olan ağ üzerindeki enerji ölçümleri



Şekil 4.73 200 düğümlük, ortalama derecesi 5 ve enerji aralığı 0-150 olan ağ üzerindeki enerji ölçümleri

5. ETMEN TABANLI KÜMELEME VE KONUMLANDIRMA ÇERÇEVESİ

Bu bölümde önerdiğimiz etmen tabanlı konumlandırma, kümeleme ve omurga oluşturma algoritmaları verilecektir. Önerilen algoritmaları anlatmadan önce algoritmaların üzerinde çalıştığı ağ modelini göstereceğiz.

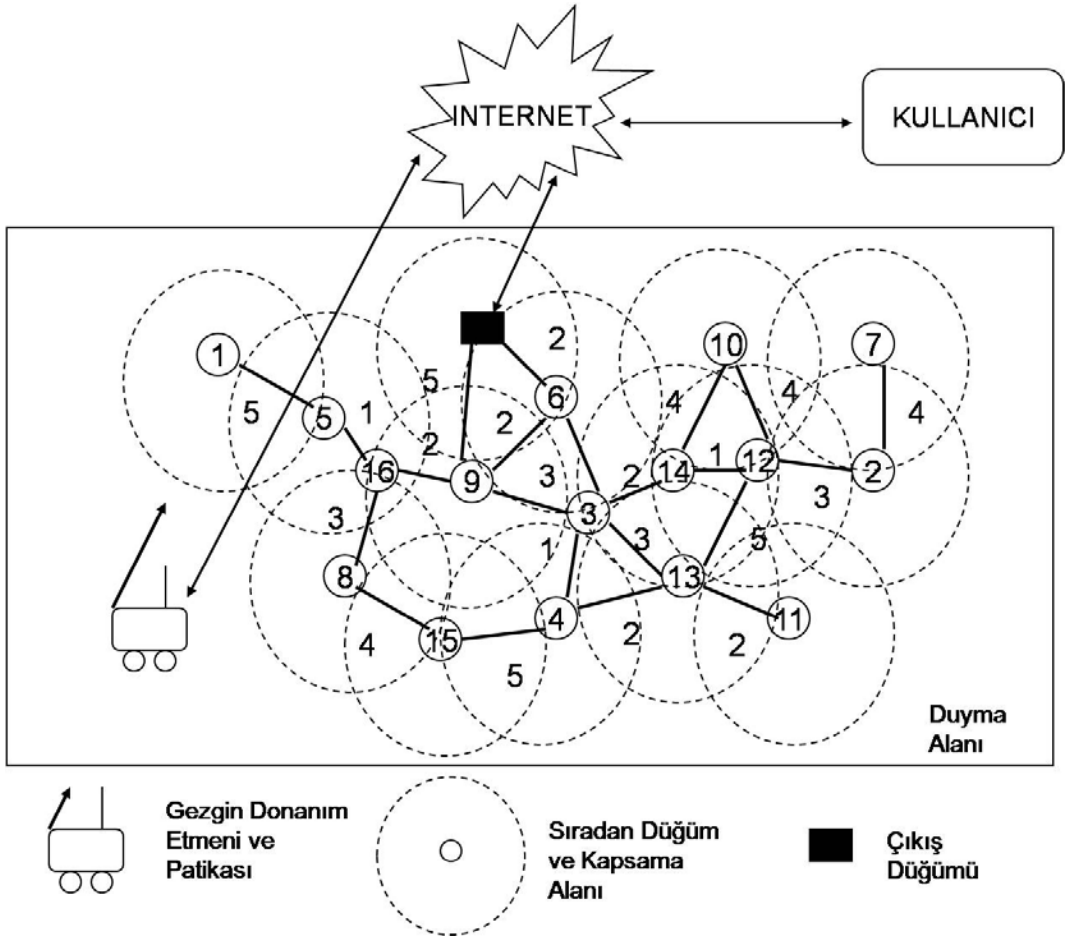
5.1. Ağ Modeli

Melez (*hybrid*) ağ modelimiz Şekil 5.1'de verilen modelden genişletilmiştir ve Şekil 5.1'de gösterilmiştir. Bu ağ modelinde 3 tip alet (*device*) bulunmaktadır:

1. Sıradan duyarga düğümü (*Ordinary sensor device*)

2. Çıkış düğümü (*Sink sensor device*)

3. Gezgin donanım etmeni (*Mobile agent device*)



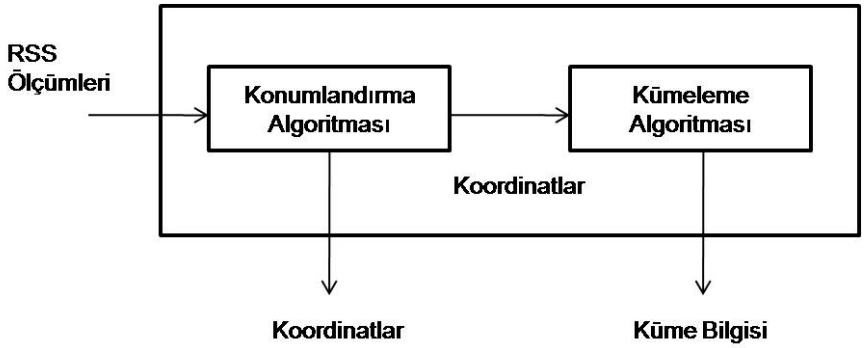
Şekil 5.1 Genişletilmiş Ağ Modeli

Bu modelin varsayımları Bölüm 1'de verilen varsayımlardan 6. madde dışındaki maddelerin ve aşağıdaki varsayım listesinin eklenmesiyle elde edilir:

1. Etmen gezgin olup, ağın haritasına sahiptir. Etmen düğümü duyarga düğümleri arasından bir patika tanımlayabilir.
2. Etmen düğümünün GPS aleti vardır.
3. Etmen düğümü Internet veya uydu üzerinden kullanıcıyla haberleşebilir. Etmen düğümünün konumlandırma ve kümeleme yapabilmek için yeteri kadar enerjisi, belleği, radyosu ve işlemcisi vardır.
4. Etmen düğümü duyarga düğümlerinden veri toplamaktan sorumlu değildir; fakat Internet üzerinden kullanıcı tarafından ayarlanabilir.
5. Düğümlerin komşularını veya konumlarını bilmelerine gerek yoktur.

5.2. Genel Fikir

TDA için bir konumlandırma yaklaşımı ve iki kümeleme algoritmasını içeren bir çerçeve öneriyoruz. Önerdiğimiz çerçeve literatürdeki etmen çalışmalarından (Tong et al., 2003; Lotfinezhad and Liang, 2005; Pathirana and Bulusu, 2005) farklı olarak kümeleme ve konumlandırma işlemlerinin ikisini birden düğüm başına sabit mesaj karmaşıklığıyla bitirir. Bunun yanında, literatürdeki dağıtık konumlandırma algoritmalarında (Alhmiedat and Yang, 2007; Doherty et al., 2001; Shang et al., 2003; Niculescu et al., 2003; Nagpal et al., 2003) ve kümeleme algoritmalarında (Banerjee and Khuller, 2000; Erciyes et al., 2008; Wu and Li, 1999; Dai and Wu, 2004; Nanuvala, 2006; Cokuslu et al., 2006), düğümlerin komşularını veya konumlarını bilmeleri gerekir. Bu işlemler büyük ölçekli TDAlarda enerji ve ekonomik açılardan çok maliyetli olabilir. Diğer taraftan önerdiğimiz çerçevede, düğümler konumlarını veya komşularını bilmeleri gerekmez. Bunun yerine donanım etmeni duyarga alanında gezinirken düğümlerden aldığı mesajlara göre önerdiğimiz çerçeveyi merkezi olarak çalıştırarak konumlandırma ve kümeleme işlemlerini bitirmekle sorumludur. Sıradan düğümler etmenden gelen sadece 3 tane mesajı cevaplamaktan sorumludur, bundan dolayı mesaj karmaşıklığımız düğüm başına sabittir. Donanım etmeni bir robot veya hareket edebilen, algoritma çalıştırabilen ve kablosuz haberleşme yapabilen herhangi bir cihaz olabilir. Aynı zamanda donanım etmeninde GPS alıcısı vardır. Donanım etmenindeki kablosuz haberleşme standartlarıyla sıradan düğümlerin kablosuz haberleşme standartları aynı olması gerekir. Şekil 5.2 donanım etmeni tarafından işletilen konumlandırma ve kümeleme işlemlerini göstermektedir.



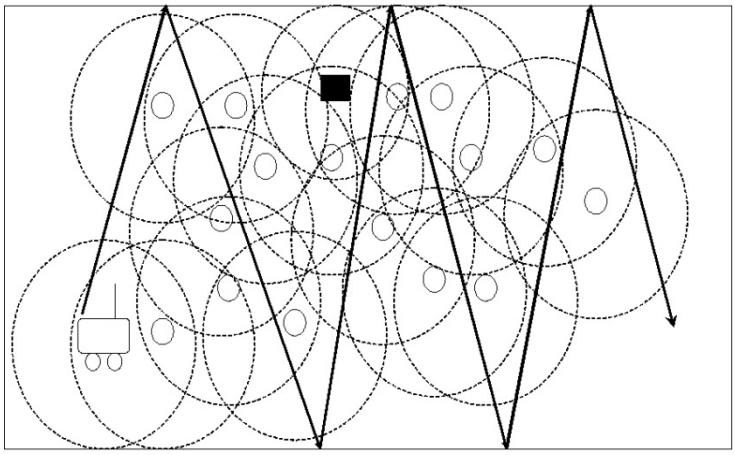
Şekil 5.2 Çerçevemiz içindeki konumlandırma ve kümeleme işlemleri

Çerçevemizin konumlandırma tekniği RSS bilgilerine dayanır. Konumlandırma algoritması RSS değerlerini girdi olarak alıp düğümlerin koordinatını çıktı olarak verir. Koordinatlar bir kenar ağırlıklı çizge modeline çevrilip önerdiğimiz çizge teorik kümeleme yaklaşımları tarafından çalıştırılır. Çizgenin kenarları düğümler arasındaki uzaklıklardır. İki düğüm arasındaki uzaklık sinyalin kapsama alanından büyükse kenar ağırlıklı çizgede bu iki düğüm bağlı olmaz. Önerdiğimiz iki kümeleme algoritması bağlı hâkim küme üretirken ikinci algoritma en küçük kapsayan ağaç üretir. Etmen konumlandırma ve kümeleme işlemini bitirdikten sonra ağa bilgileri gönderir. İlerleyen bölümlerde konumlandırma ve kümeleme algoritmaları detaylı olarak anlatılacaktır.

5.3. Konumlandırma Algoritması

TDAında merkezi ve dağıtık yaklaşımlarda çok sayıda mesaj değişimi olabilmekte veya hata oranları çok büyük çıkabilmektedir. Hata oranını ve mesaj değişimi düşürmek için ağda gezinen ve bilgi toplayan bir donanım etmeninin konumlandırma işlemini yapmasını önerdik. Şekil 5.3'de donanım etmeninin ağ üzerindeki hareketi görülmektedir. Konumlandırma algoritmamızda donanım etmenimiz Şekil 5.3'de gösterilen patika üzerinde giderken sabit aralıklarla durup *İSTEK(X,Y) (REQUEST)* mesajı gönderir. *İSTEK(X,Y)* mesajını alan bir düğüm, mesajın içinden gelen X ve Y herhangi biri kayıtlı değilse bu konumları kaydedip *CEVAP(düğüm_kimliği)* (*REPLY*) mesajıyla bu mesaja karşılık verir. *CEVAP* mesajları gönderilirken etmen ile düğüm arasında TDMA (*Time Division Multiple Access*) yapılır. Etmen *İSTEK*(*X,Y*) mesajını gönderdikten sonra mesajı alan düğümler zamanlayıcıyı *düğüm_kimliği**(*mesaj iletim süresi=Yaklaşık 10ms*) zamanına kurarak bu zamanlayıcı dolduğunda *CEVAP* mesajını gönderir. Düğümün aynı konumlardan gelen *İSTEK* mesajına cevap vermemesinin sebebi etmenin trilaterasyon ile konumlandırma yapması ve trilaterasyon işlemi için birbirlerinden farklı konumların gerekliliğidir. Etmen en büyük *düğüm kimliği**(*mesaj iletim süresi*) kadar bekleyip bütün *CEVAP* mesajlarını toplar. Etmen *CEVAP* mesajını aldığı zaman *CEVAP* mesajının RSS'ini ve o an bulunduğu konumu kaydeder. Kaydedilen

bu bilgiler etmenin sıradan düğümlerin konumunu hesaplamasında kullanılır. Etmen Şekil 5.3'de gösterilen zigzag patikasında gezinirken her düğümden 3 adet *CEVAP* mesajı alır ve bu mesajların RSS'ini ve o an bulunduğu konumları kaydeder.



Şekil 5.3 Etmenin zigzag patikası üzerindeki hareketi

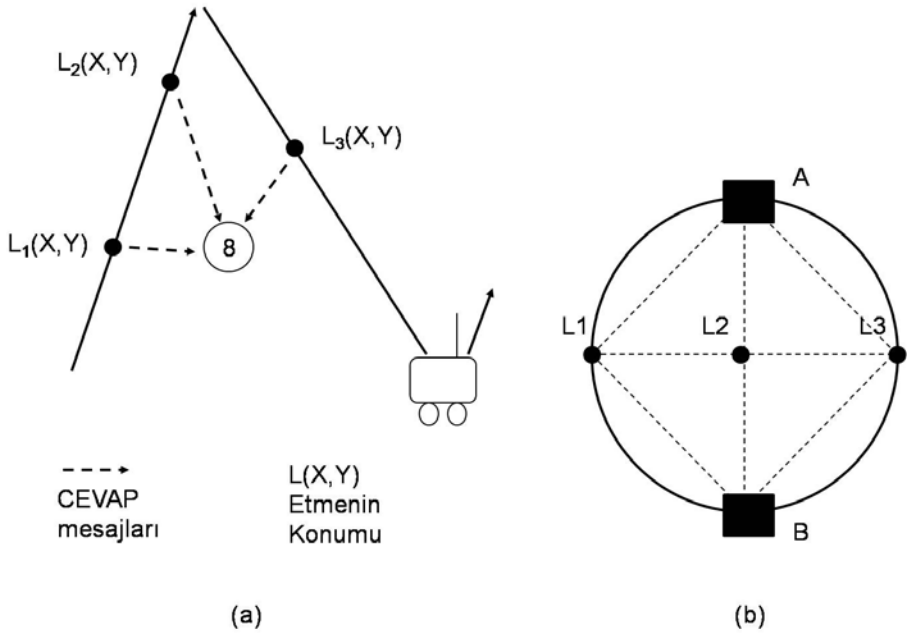
Etmen konumlandırma yapmak için trilaterasyon tekniğini kullanır. Bu teknikte, bir düğümün konumunun bulunabilmesi için üç uzaklık ölçümü ve bunlara karşılık gelen üç konum bilgisi gereklidir. Etmen tarafından kaydedilen X ve Y koordinatlarından herhangi bir koordinata ait 3 ölçümüm birbirine eşit olmaması önemlidir. Bundan dolayı etmen zigzag patikası üzerinden hareket eder. Zigzag patikasının seçilmesinin diğer sebebi etmenin duyarga alanı üzerindeki bütün düğümlere ulaşabilmesidir. Ağdaki bütün düğümleri gezebilmeyi ve düğümlerden doğrusal olmayan koordinat bilgilerini toplamayı sağlayan zigzag dışındaki herhangi bir patikayı etmen kullanabilir.

Şekil 5.4 (a)'da etmenin bir düğümü konumlandırma yönteminin çalışması görülüyor. Düğüm 8'in konumunu bulmak için etmen 3 adet *CEVAP* mesajını 3 farklı koordinat (L1(x,y), L2(x,y), L3(x,y)) üzerindeyken alır. Daha sonra etmen *CEVAP* mesajlarının RSS değerlerini uzaklık değerine çevirir. Uzaklık değerleri ve konum bilgilerini kullanarak etmen trilaterasyon tekniğiyle düğümün konumunu bulur. Trilaterasyon için gerekli denklemler Denklem 5.1, denklem 5.2, denklem 5.3'de verilmiştir. Bu denklemdeki etmenin konum bilgileri (X_{L1}, Y_{L1}), (X_{L2}, Y_{L2}), and (X_{L3}, Y_{L3}) olarak verilmiştir, RSS değerlerinden çıkarılan uzaklık bilgileri d_{rss1}, d_{rss2}, d_{rss3} değişkenleriyle temsil edilmiştir. Üç denklemi kullanarak düğümün konumu ($X_{düğüm}$) hesaplanabilir. Şekil 5.4 (b)'de X koordinatları eşit olduğunda konumlandırmaki problem görülmektedir; A ve B koordinatları L1, L2 ve L3 koordinatlarına aynı uzaklığa sahiptir.

$$(X_{düğüm} - X_{L1})^2 + (Y_{düğüm} - Y_{L1})^2 = d^2_{rss1} \qquad (5.1)$$

$$(X_{düğüm} - X_{L2})^2 + (Y_{düğüm} - Y_{L2})^2 = d^2_{rss2} \qquad (5.2)$$

$$(X_{düğüm} - X_{L3})^2 + (Y_{düğüm} - Y_{L3})^2 = d^2_{rss3} \quad (5.3)$$



Şekil 5.4 Konumlandırma Yöntemi

Etmen düğümlerin konum ve küme bilgilerini bulduktan sonra ağı gezdiği patikayı ters yönden bir daha giderek düğümlere konum ve küme bilgilerini *BİLGİ(düğüm_kimlik listesi, düğüm_konum listesi, kümeleme bilgileri)* mesajından gönderir. Etmen ağı ters yönde gezerken sadece mesajın iletim süresini sağlayacak kadar durma yapar, bundan dolayı ikinci turda daha az zaman harcar. Düğümler bu turda sadece mesaj alımı yaparlar. *BİLGİ* mesajını alan düğüm için konumlandırma ve kümeleme işlemi bitmiş olur.

5.4. Kümeleme Algoritmaları

Bu bölümde etmen tarafından çalıştırılan kümeleme algoritmaları verilecektir. İki algoritma önerilmiştir. İlk algoritma en küçük kapsayan ağaç omurgasıyla birlikte kümeleri oluşturur, ikinci algoritma bağlı hâkim küme omurgasıyla birlikte kümeleri oluşturur.

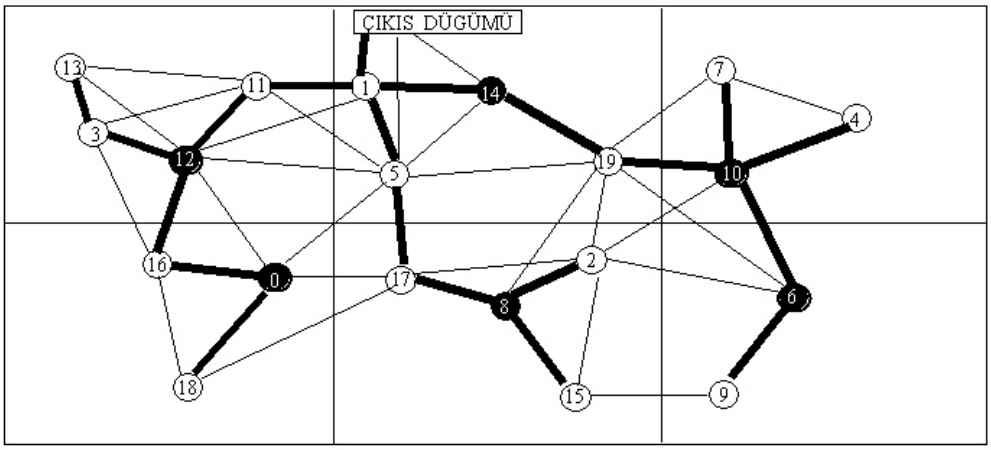
5.4.1. En Küçük Kapsayan Ağaç Algoritması

Literatürdeki kapsayan ağaç algoritmalarında dengeli kümelemenin öncelikli hedef olarak seçilmediği görülmüştür (Erciyes et al., 2008). Önerdiğimiz algoritma (etmen tabanlı en küçük kapsayan ağaç algoritması (agent based minimum spanning tree algorithm (AST))) lider düğümler arasında çıkış düğümüne doğru yönlenmiş en küçük kapsayan ağaç oluştururken kümelerin elemanlarını dengeli olarak oluşturur.

Bu işlemlerin yapılabilmesi için öncelikle alan *X* * *Y*'lik ızgaralara (*grid*) bölünür. *X* ve *Y* değeri kullanıcı tarafından önceden belirlenen değerlerdir. Her ızgara bir kümedir, bundan dolayı *X* * *Y* toplam küme sayısını verir. *N* adet düğüm duyarga alanına tamamen düzgün olarak dağıtılmışsa her küme içindeki beklenen düğüm sayısı *N*/(*X***Y*)'dir. Düğümler kümelenmek için ızgaralara bölündükten sonra, küme içindeki bağlantıyı sağlamak için aynı küme içindeki düğümler arasında en küçük kapsayan ağaç oluşturulur. En çok sayıda komşusu olan düğüm küme lideri olarak seçilir. Kümelerarası iletişim kurmak için küme liderleri arasında en küçük kapsayan ağaç omurgası oluşturuldu. En küçük kapsayan ağacı oluşturmak için Kruskal'ın algoritması (Kruskal, 1956) kullandık, Prim (Prim, 1957) veya Borouvka'nın algoritmaları (Jungnickel, 2005) da kullanılabilinir. Algoritmanın bütün basamakları Şekil 5.5'te verilmiştir. AST algoritmasını örnek bir ağ üzerinde çalıştırılması Şekil 5.6'da verilmiştir. Ağ 6 kümeye bölünmüştür, küme liderleri siyah ile gösterilmiş, en küçük kapsayan ağaç kenarları kalın çizgilerle gösterilmiştir. Şekil 5.5'de küme içi ve dışı seçilen kenarlar aynı olduğu için hepsi birlikte kalın çizgilerle gösterilmiştir.

AST Algoritması
Başlangıçta: Kruskal_En_Küçük_Kapsayan_Ağaç(çizge) algoritması en küçük kapsayan ağacı Kruskal'ın algoritmasıyla hesaplar.
Girdi: *X_Parça_Numarası*, *Y_Parça_Numarası*
Düğümlerin hesaplanan koordinatlarından ve kapsama alanından $G_w = (V, E)$ ağırlıklandırılmış çizgesini bul
T_w= Kruskal_En_Küçük_Kapsayan_Ağaç(G_w)
G_w 'yi *X_Parça_Numarası* x *Y_Parça_Numarası* kadar ızgaralara (Grid) böl
Her G_i ızgarası için
    T_i= Kruskal_En_Küçük_Kapsayan_Ağaç(G_i)
    Her G_i için en çok bağlantısı olan düğümü lider seç (ch_i), eğer birden fazla varsa
    En büyük kimlik olanı al
    T_i deki düğümleri ch_i'ye doğru yönlendir
döngüyü bitir
Çıktı: T_w omurga, T_i küme içi ağaçlar, ch_i küme liderleri

Şekil 5.5 AST algoritması



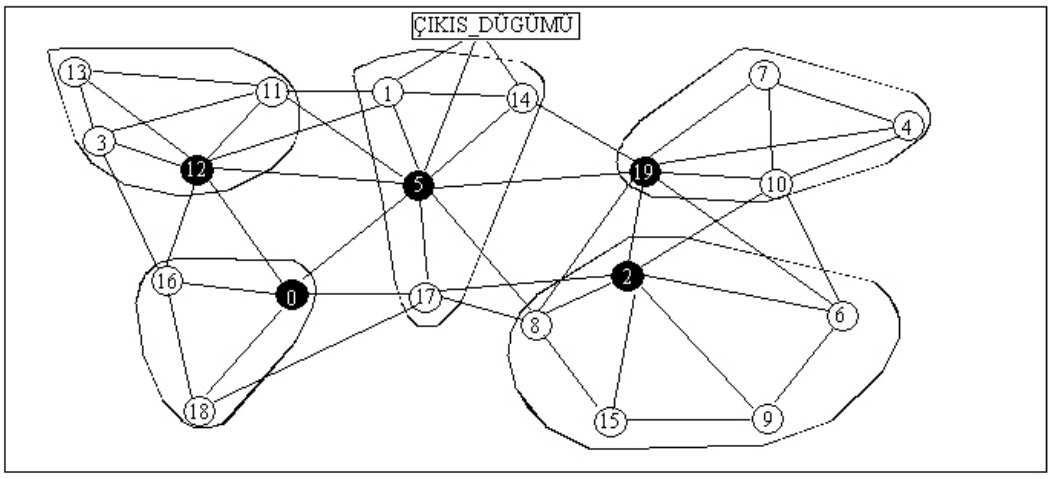
Şekil 5.6 AST algoritmasının örnek bir ağ üzerinde çalıştırılması

5.4.2. Bağlı Hâkim Küme Algoritması

BHK algoritmanın en önemli hedefi küme içindeki eleman sayısını olabildiğince azaltmaktır. Bu hedefin doğrultusunda Guha'nın algoritmasını (Guha and Khuller, 1998) kullanarak yeni bir algoritma (etmen tabanlı bağlı hâkim küme algoritması (*agent based dominating set algorithm*) (ADS)) öneriyoruz. Önerdiğimiz algoritmada koordinatlardan çizgeyi oluşturduktan sonra her sıradan düğümü en yakınındaki küme liderine bağlıyoruz. Eğer birden fazla lider adayı varsa kimliği en büyük lidere bağlıyoruz. ADS algoritması Şekil 5.7'de, örnek uygulaması Şekil 5.8'de verilmiştir. Örnek uygulamada küme liderleri siyah ile boyanmış, kümelerin sınırları çizilmiştir.

ADS Algoritması
Başlangıç: Guha_BHK(çizge) Guha'nın algoritmasıyla BHK bulur
Düğümlerin hesaplanan koordinatlarından ve kapsama alanından $G_w = (V, E)$ ağırlıklandırılmış çizgesini bul
S = Guha_CDS(G_w)
Her sıradan düğümü S içinde en yakın olduğu hâkim elemana bağla
Çıktı: S omurgası

Şekil 5.7 ADS algoritması



Şekil 5.8 ADS algoritmasının örnek bir ağ üzerinde çalıştırılması

5.5. Analiz

Bu bölümde konumlandırma analizi ve kümeleme analizi verilmiştir.

5.5.1. Konumlandırma Çalışma Zamanı Analizi

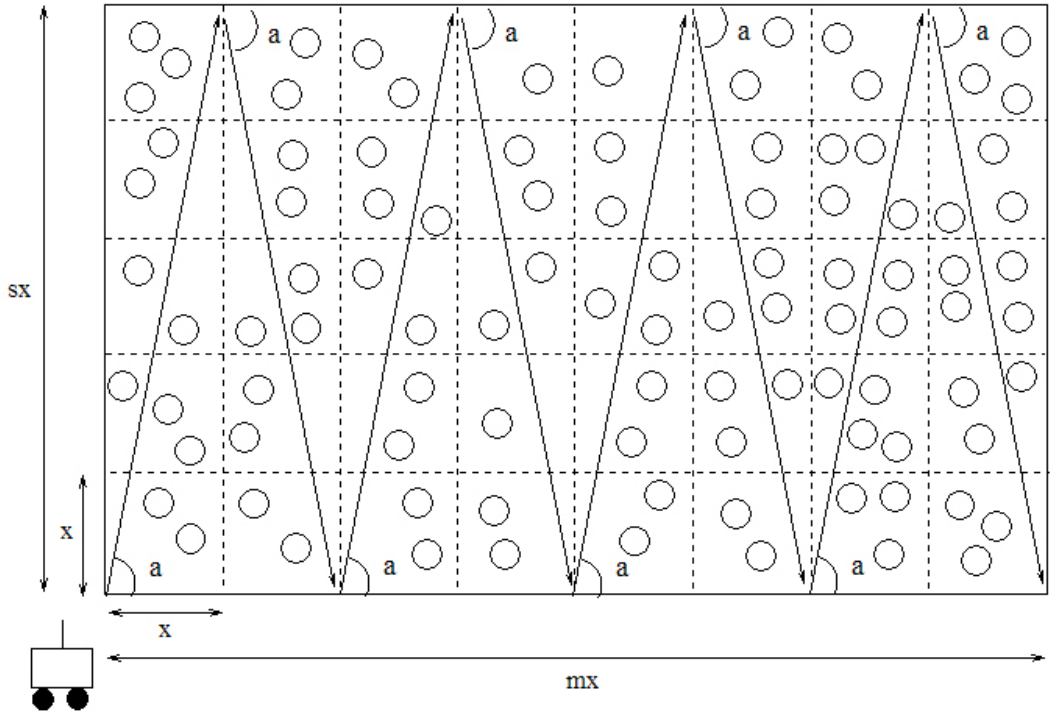
Önerdiğimiz tekniğin konumlandırma analizini yapabilmek için öncelikle etmenin örnek bir alan üzerindeki hareketini ve bu örnek alanla ilgili ölçüleri Şekil 5.9'da veriyoruz. Şekil 5.9'da duyarga alanı üzerine duyarga düğümleri rastgele yerleştirilmiştir. Duyarga alanın toplam büyüklüğü $sx*mx$ m^2 kadar olup, bu alan, bir

kenarı x m olan karelere bölünmüştür. Etmenin bu alan üzerindeki hareketin açısı a'dır.

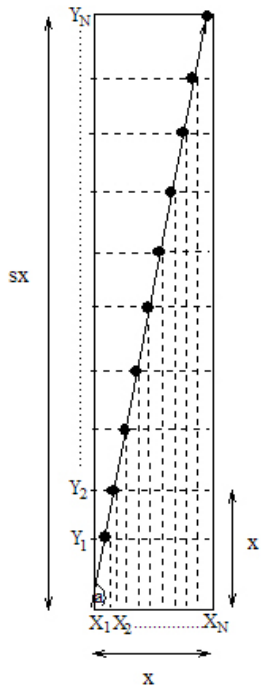
Gözlem 5.1. Etmenin düğümleri trilaterasyon ile konumlandırması için X ve Y koordinatlarından herhangi bir koordinata ait 3 ölçümün birbirine eşit olmaması gereklidir.

Teorem 5.1. Şekil 5.9'da gösterilen genel durumda, tan $a = s$ için, düğümlerin ve etmenin kapsama alanı T olduğunda ve $x\sqrt{2} < T$ eşitsizliğini sağladığında etmen zigzag hareketi boyunca her $\frac{x\sqrt{1+s^2}}{2s}$ kadar yol aldığında bir durup düğümlerden toplam 3'er adet ölçüm alırsa bütün düğümleri trilaterasyon ile konumlandırır.

İspat. $x\sqrt{2} < T$ olduğu durumda Şekil 5.10'da görüldüğü üzere etmen bir kare içindeyken o kare içindeki bütün düğümlerden mesaj alıp gönderebilir. Etmen bir kare içindeyken 3 farklı konumda ölçüm alırsa bu aldığı ölçümlerin X ve Y koordinatları zigzag hareketinden dolayı Şekil 5.10'da gösterildiği üzere $Y_0 \neq Y_1 \neq \ldots \neq Y_N$ ve $X_0 \neq X_1 \neq \ldots \neq X_N$ olmak üzere farklı olacaktır.

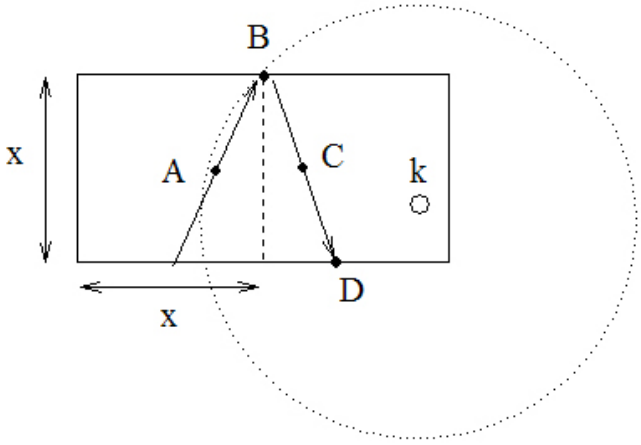


Şekil 5.9 Etmenin duyarga alanı üzerindeki hareketi ve alanın ölçüleri



Şekil 5.10 Etmenin *s* adet kareyi çapraz olarak geçmesi

Şekil 5.10'da gösterilen *s* adet kareyi içeren dikdörtgen üzerinde etmenin toplam aldığı yol Pisagor bağıntısından $\sqrt{x^2 + x^2 s^2} = x\sqrt{1+s^2}$ 'e eşittir. Yukarıda anlatıldığı üzere etmenin 3 farklı konumdayken ölçüm alabilmesi için düzenli olarak her $\frac{x\sqrt{1+s^2}}{2s}$ kadar yol aldıktan sonra ölçüm alması yeterli olacaktır.



Şekil 5.11 Etmenin aynı koordinatlara sahip olduğunda İSTEK gönderme durumu

Şekil 5.10'da anlatılan durumun dışında Şekil 5.11'de gösterilen durum da olabilir. Şekil 5.10'da etmenin iki kare üzerindeki hareketi ve $Node_k$ düğümüyle birlikte kapsama alanı verilmiştir. $Node_k$ düğümü etmenden A, B ve C konumlarındayken *İSTEK* mesajı alabilir, bu durumda A ve C konumlarının Y koordinatları birbirine eşittir. $Node_k$ düğümü C koordinatlarına gelen mesaja *CEVAP*

göndermeyeceğinden dolayı bir sonraki koordinatlardan (D)'den gelen mesaja *CEVAP* gönderir.

Böylelikle etmen düğümden ölçüm alırken A, B ve D koordinatlarında bulunur ve bu koordinatların X,Y değerlerinin hepsi farklıdır. Şekil 5.9'da gösterilen etmenin genel hareketi incelenirse Şekil 5.10 ve Şekil 5.11 üzerinde anlatılan durumlar dışında başka bir durum olamayacağı görülür. Teoremimiz doğrudur. ■

Teorem 5.2. Şekil 5.9'da gösterilen genel durumda v ortalama hızıyla giden etmenin düğümleri kümelemesi ve konumlandırması için geçen süre, t bir mesajın iletim süresi olduğunda $\frac{2xm\sqrt{1+s^2}}{v}$ +($m2s$+1)Nt'e eşittir.

İspat. Etmenin Şekil 5.10'daki bir dikdörtgenin köşegenini geçmek için $\frac{x\sqrt{1+s^2}}{v}$ süre harcar. Bu köşegenlerden m adet olduğu için toplam süre $\frac{xm\sqrt{1+s^2}}{v}$ 'e eşit olur. Etmenin hareketine başladığı noktadaki durmasını saymazsak, etmen her dikdörtgen içinde 2s kadar durma yapar. Toplam durma işlemi böylece $m2s$ olur. Etmenin hareketine başladığı noktayı eklersek etmen $m2s$+1 kadar durma yapar ve her durmada TDMA yapıldığından dolayı Nt vakit harcanır. Ayrıca etmenin geri dönüşünü de eklersek ve geri gönderken durmadığı varsayarsak $\frac{xm\sqrt{1+s^2}}{v}$ kadar daha süre harcanır. Sonuç olarak etmen için toplam geçen süre $\frac{2xm\sqrt{1+s^2}}{v}$ +($m2s$+1)Nt'e eşit olur. ■

5.5.2. Kümeleme Analizi, Mesaj ve Uzay Karmaşıklığı

Bu bölümde kümeleme algoritmalarının doğruluk analizi ve mesaj karmaşıklığı verilmiştir. Bu bölümde küme kalitesiyle ilgili teoremlerin hepsinde konumlandırma hatasının 0'a yaklaştığı kabul edilmiştir. Uzay karmaşıklığı yapılırken farklı veri tiplerinin sabit ve birbirleriyle yakın miktarda bellek alanları tuttukları farz edilmiştir.

Teorem 5.3. Çerçevemizin mesaj karmaşıklığı düğüm başına O(3)'dür.

İspat. Her düğüm sadece etmenden gelen 3 *İSTEK* mesajını trilaterasyon tekniğinde 3 konum ve uzaklık bilgisi gerektiğinden dolayı cevaplar. Bundan dolayı düğüm başına mesaj karmaşıklığı O(3)'tür. ■

Sonuç Teoremi 5.2. Çerçevemizin mesaj karmaşıklığı O(N)'dir.

İspat. Her düğüm sadece 3 mesaj gönderir, toplam gönderilen mesaj sayısı 3*N* olduğunda mesaj karmaşıklığı O(*N*)'dir. ■

Teorem 5.4. ADS algoritması, bağlı bir hâkim küme kurarak, düğümleri küme üyesi ve küme lideri olarak sınıflandırır ve böylece ağı kümelere böler.

İspat. ADS algoritması Guha'nın BHK algoritmasını kullanır. Guha'nın BHK algoritmasının doğruluğu (Guha and Khuller, 1998)'dan görülebilir. Daha sonra Şekil 5.7'de verilen ADS algoritmasının 4. satırı her düğümü en yakın küme liderine birleştirir. ■

Teorem 5.5. ADS algoritması en kötü durumda H harmonik fonksiyon, *D* de en az elemanlı bağlı hâkim küme olmak üzere 2(1+H(*Δ*)) *D* kadar küme üretir.

İspat. ADS algoritması Guha'nın BHK algoritmasını kullanır. Bundan dolayı teorik olarak üreteceği küme sayısı Guha'nın küme sayısı ile eşdeğerdir. İspatı (Guha and Khuller, 1996)'de bulunabilir. ■

Teorem 5.6. AST algoritması, omurgayı ve kümeleri en küçük kapsayan ağaç topolojisinde oluşturup, düğümleri küme üyesi ve küme lideri olarak sınıflandırır, bu şekilde ağı kümelere böler.

İspat. AST algoritması duyarga ağını coğrafi olarak eşit alanlara bölüp, aynı alandaki düğümleri aynı kümenin içine koyar. Algoritma Kruskal'ın en küçük kapsayan ağaç algoritmasını kullanır ve bu algoritmanın doğruluğu (Kruskal, 1956)'da verilmiştir. Her kümenin lideri de en küçük kapsayan ağaç içindeki en çok bağlı düğümdür. Birden fazla varsa, kimliği büyük olan düğüm küme lideri olur. Aynı alan içinde birden fazla lider düğüm bulunmaz. ■

Teorem 5.7. AST algoritması, *N* adet düğümün düzgün olarak dağıldığı bir ağı *M* adet kümeye bölerse her kümede beklenen eleman sayısı *N*/*M* dir.

İspat. AST algoritması duyarga ağını coğrafi olarak eşit alanlara bölüp, aynı alandaki düğümleri aynı kümenin içine koyar. Düğümler alana eşit olarak yayılmışsa düzgün dağılıştan (*uniform distribution*) her küme içindeki beklenen düğüm değeri *N*/*M* dir. ■

Teorem 5.8. Çerçevemizde düğümlerin gönderdiği en büyük boyutlu mesajın uzay karmaşıklığı Θ(1)'e eşittir.

İspat. Çerçevemizde düğümler sadece *CEVAP*(*düğüm_kimliği*) mesajını gönderirler, bundan dolayı Θ(1)'e eşittir. ■

Teorem 5.9. Çerçevemizde bir düğümün kullanması gereken belleğin uzay karmaşıklığı Θ(11)'e eşittir.

İspat. Bir düğümün kullanması gerekli olan değişkenler şunlardır:

1. Düğümün kimliği
2. Düğümün X koordinatı
3. Düğümün Y koordinatı
4. Küme bilgisi: Ebeveyn bilgisi
5. Durum bilgisi: Hâkim küme içinde olup olmadığı
6. Düğümlerin cevapladığı 3'er adet X ve Y koordinatları (Toplam 6 alan)

■

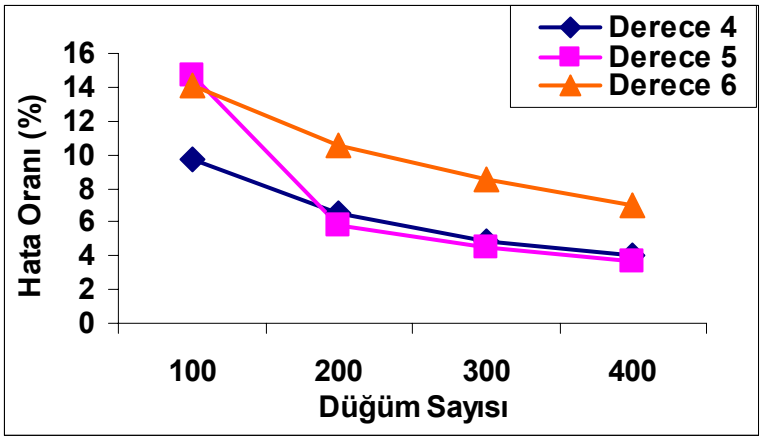
5.6. Performans Değerlendirmesi

Önerdiğimiz çerçeveyi ns2 benzetim ortamında uyguladık. 100 düğümden 400 düğüme kadar ağlar oluşturup Bölüm 3.5'te verilen düzen uygulanmıştır. Etmen dışındaki bütün düğümler hareketsizdir. Benzetimlerimizde $x\sqrt{2} <$ 250 m olacak şekilde seçilmesi gerektiğinden x < 176.8 m için *x* değeri 140 m alınmıştır. Etmenin hızı 15 m/s olarak alınmıştır. Analiz bölümünde etmenin gerekli konumları toplaması için alması gereken periyodik yol verilmiştir. Benzetimler yapılırken etmen her dikdörtgen içinde 4 adet ölçüm alırsa trilaterasyon için yeterli olduğu tespit edilmiştir. Çerçevemizin konumlandırma hatası, kümeleme kalitesi ve enerji tüketimleri verilmiştir. Algoritmamızın kümeleme performansını karşılaştırmak için DSTA ve DS algoritmaları, konumlandırma performansını karşılaştırmak için yenilemeli üçlü laterasyon (*iterative trilateration* (IT)) algoritması uygulanmıştır.

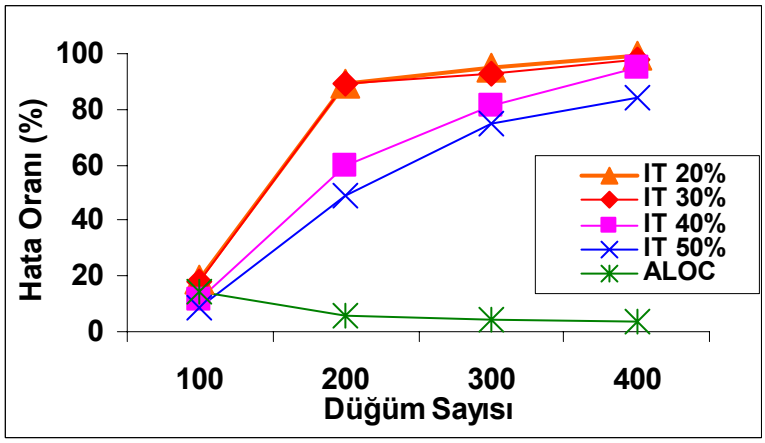
5.6.1. Konumlandırma Performansı

Öncelikle konumlandırma performansını değişen düğüm derecelerine göre ölçtük. Şekil 5.12'de önerdiğimiz algoritmanın düğüm derecelerine bağlı olarak hata oranları verilmiştir. Ortalama konumlandırma hatasını bütün düğümlerin ortalamaları olarak hesapladık. Şekil 5.12'de görüldüğü düğüm derecesi arttıkça hata oranında az bir artış, düğüm sayısı arttıkça hata oranının azaldığı gözlemlenmiştir. Bu sonuçlara göre konumlandırma yaklaşımımız ölçeklenebilirdir. IT algoritmasının ve bizim önerdiğimiz konumlandırma algoritmasının (*agent based localization* (ALOC)) hata oranları karşılaştırdık. IT algoritmasındaki referans düğüm sayısını % 20 'den % 50'ye kadar değiştirerek konumlandırma hatalarını ölçtük. Şekil 5.13'de görüldüğü

üzere referans düğüm yüzdesi arttıkça IT'nin hata oranı düşse de, ALOC algoritmasının hata oranı IT'den her durumda çok daha küçüktür.



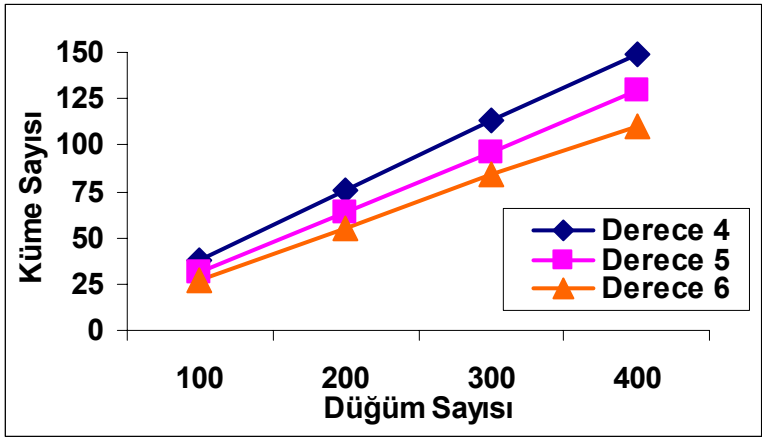
Şekil 5.12 Değişen düğüm derecelerine göre konumlandırma hata oranları



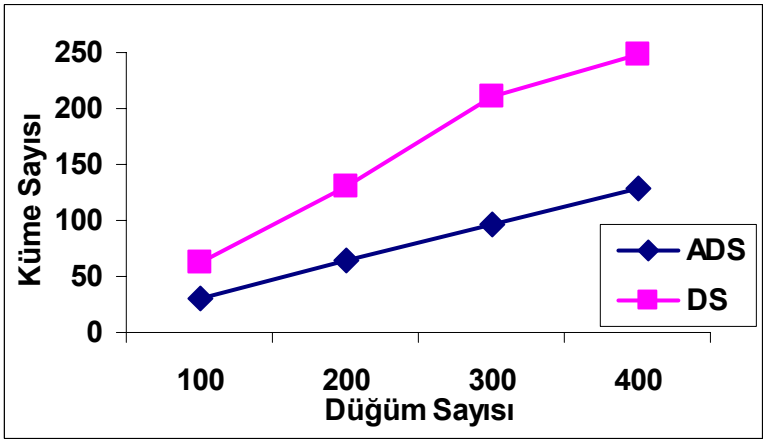
Şekil 5.13 Algoritmaların konumlandırma hata oranları

5.6.2. Kümeleme Performansı

Bu bölümde önerdiğimiz kümeleme algoritmalarının performansları karşılaştırılmalı olarak verilecektir. Kümeleme kalitesini ölçmek için daha önce Bölüm 3.5.3'de yaptığımız gibi kümelerin sayısı ve varyasyon katsayısı ölçütleri kullanılmıştır. Şekil 5.14'de görüldüğü üzere ortalama düğüm derecesi arttıkça ADS tarafından üretilen küme sayısı azalır. Bunun sebebi ortalama düğüm sayısı arttıkça bir küme liderinin birden fazla sıradan düğümü örtebilmesidir. Şekil 5.15'de görüldüğü üzere ADS algoritmasının ürettiği küme sayısı DS algoritmasının yaklaşık yarısı kadardır. Ağ büyüdükçe, ADS algoritması DS'den çok daha iyi sonuçlar üretir.

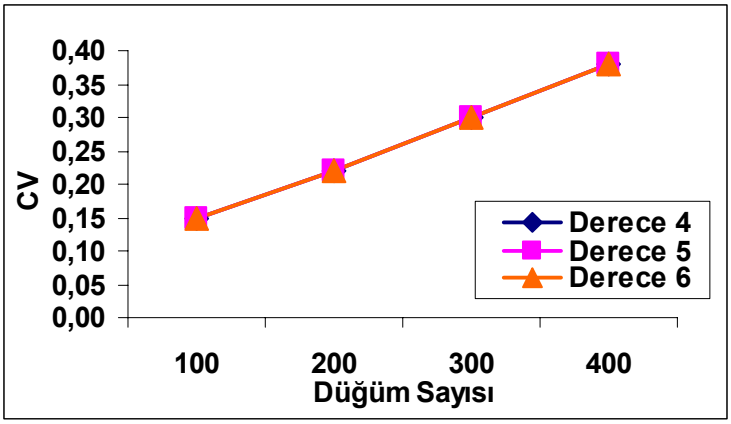


Şekil 5.14 Değişen düğüm derecesine göre ADS'nin küme sayıları

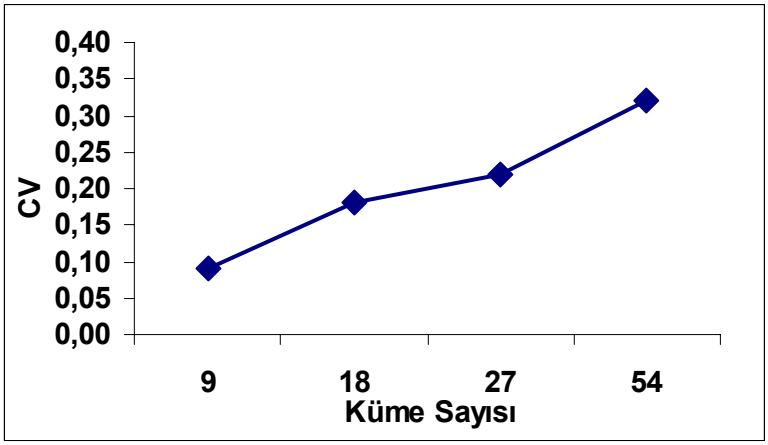


Şekil 5.15 Algoritmalar tarafından üretilen küme sayıları

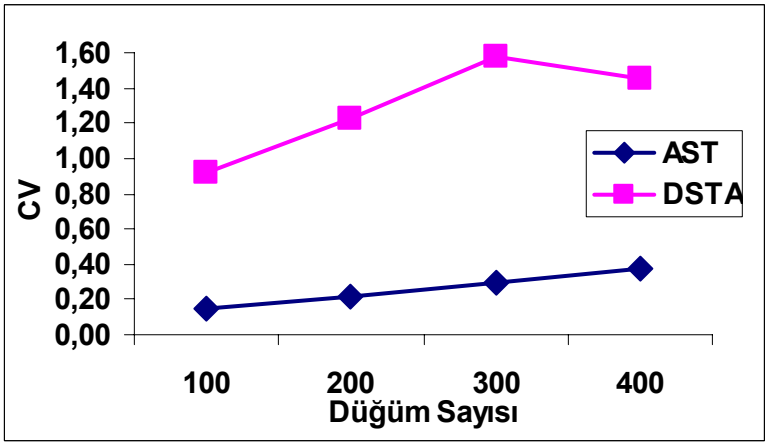
AST algoritmasının küme dengesini ölçmek için CV değerlerini ölçtük. Şekil 5.16'da görüldüğü üzere AST tarafından oluşturulan kümelerin dengesi değişen ortalama düğüm derecelerine karşın kararlıdır. Küme sayısını arttırdığımızda AST tarafından üretilen kümelerin CV değerleri arttığı Şekil 5.17'de görülmektedir. Bundan dolayı AST'nin küme sayısının az olduğu durumlarda alanı daha etkin bölüp, daha dengeli kümeler oluşturduğunu görüyoruz. AST ve DSTA algoritmalarının karşılaştırılmaları Şekil 5.18'de verilmiştir. AST'nin, DSTA'dan çok daha dengeli kümeler oluşturduğu ve düğüm sayısı arttıkça CV değerlerinin daha az eğimle arttığı şekilde görülmektedir. Şekil 5.19'da konumlandırma hatasından kaynaklanan AST algoritmasındaki bağlantı hataları verilmiştir. Birbirlerinin kapsama alanı içinde olmayan *A* ve *B* düğümleri, konumlandırma işlemindeki hatadan dolayı kümeleme sırasında aralarında bağlantı olabilir. Ortalama olarak düğüm sayısına bağlı olarak hata oranı %9'a eşittir. Bu bağlantı hatalarının giderilmesi için bir ilerleyen bölümlerde bir yöntem önerilecektir.



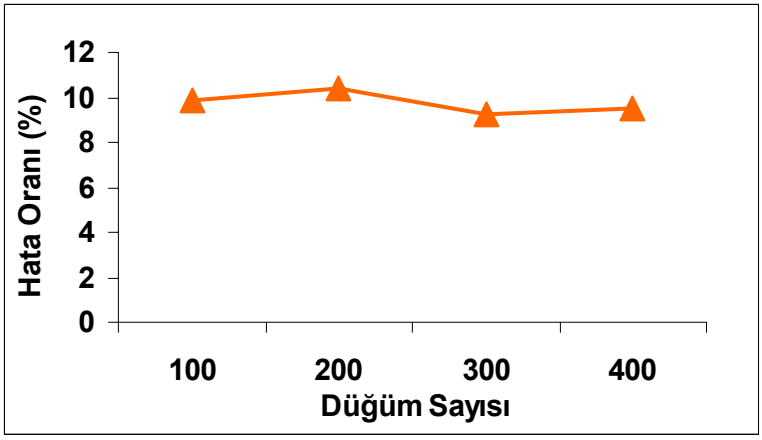
Şekil 5.16 Değişen düğüm derecesine göre AST'nin CV değerleri



Şekil 5.17 Değişen küme sayılarına göre CV değerleri



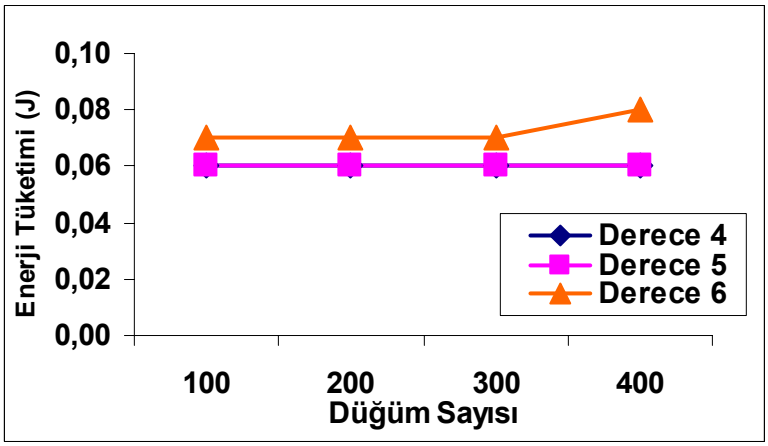
Şekil 5.18 Algoritmaların CV değerleri



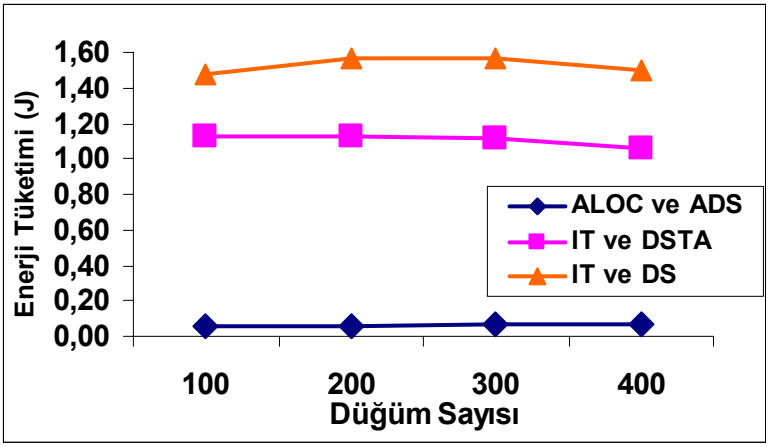
Şekil 5.19 Konumlandırma hatasından kaynaklanan AST'deki bağ hataları

5.6.3. Enerji Tüketimleri ve Zaman Ölçümleri

Algoritmaların enerji tüketimlerini ölçtük. Şekil 5.20'de görüldüğü üzere ortalama düğüm derecesi arttıkça düğüm başına harcanan enerji az bir artış gösterir. Şekil 5.21'de bizim çerçevemizin ve dağıtık algoritmaların enerji tüketimleri verilmiştir. IT ve DSTA algoritmaları dağıtık konumlandırma ve kümeleme işlemini için birlikte yapmak için uygulanmıştır. IT ve DS algoritmaları aynı sebepten dolayı uygulandı ve toplam enerji tüketimi ölçüldü. ALOC ve ADS algoritmaları bizim önerdiğimiz çerçevenin içindeki konumlandırma ve kümeleme algoritmalarıdır. ADS ve AST algoritmaları aynı enerjiyi tükettiği için sadece ADS algoritmasını şekil içinde verdik. Şekil 5.21'de görüldüğü üzere önerdiğimiz çerçevenin enerji tüketimi dağıtık yaklaşımlara göre çok daha azdır.

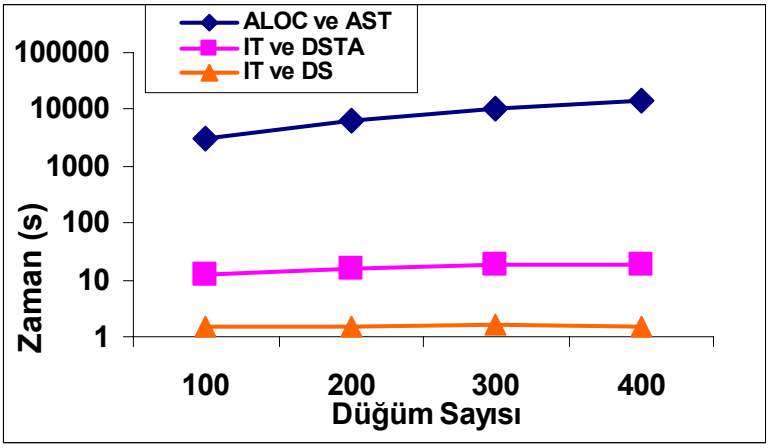


Şekil 5.20 Değişen düğüm derecesine göre çerçevemizin enerji tüketimleri



Şekil 5.21 Algoritmaların enerji tüketimleri

Şekil 5.22'de algoritmaların süre kullanımları verilmiştir. Önerdiğimiz yöntem daha kaliteli konumlandırma, kümelemeyi daha az enerji tüketimiyle birlikte yapmasına karşın diğer yöntemlerden çok daha fazla enerji harcar. Sonuç olarak önerdiğimiz çerçevenin konumlandırma ve kümeleme performansları dağıtık yaklaşımlara çok daha üstün olup, enerji tüketimleri çok daha azdır. Çerçevemizin performansı literatürdeki algoritmalara göre çok daha iyi olmasına rağmen, donanım etmenin ekonomik maliyeti çok önemli bir dezavantajdır. Önümüzdeki bölümde ekonomik maliyetleri hesap ederek donanım etmeninin uygun olabileceği uygulama tipleri verilecektir. Ayrıca önerdiğimiz yöntemin üzerine hata toleransını sağlayan ve çalışma zamanını düşüren geliştirmeler verilecektir.



Şekil 5.22 Algoritmaların zaman tüketimleri

5.7. Tartışmalar ve Uygulamalar

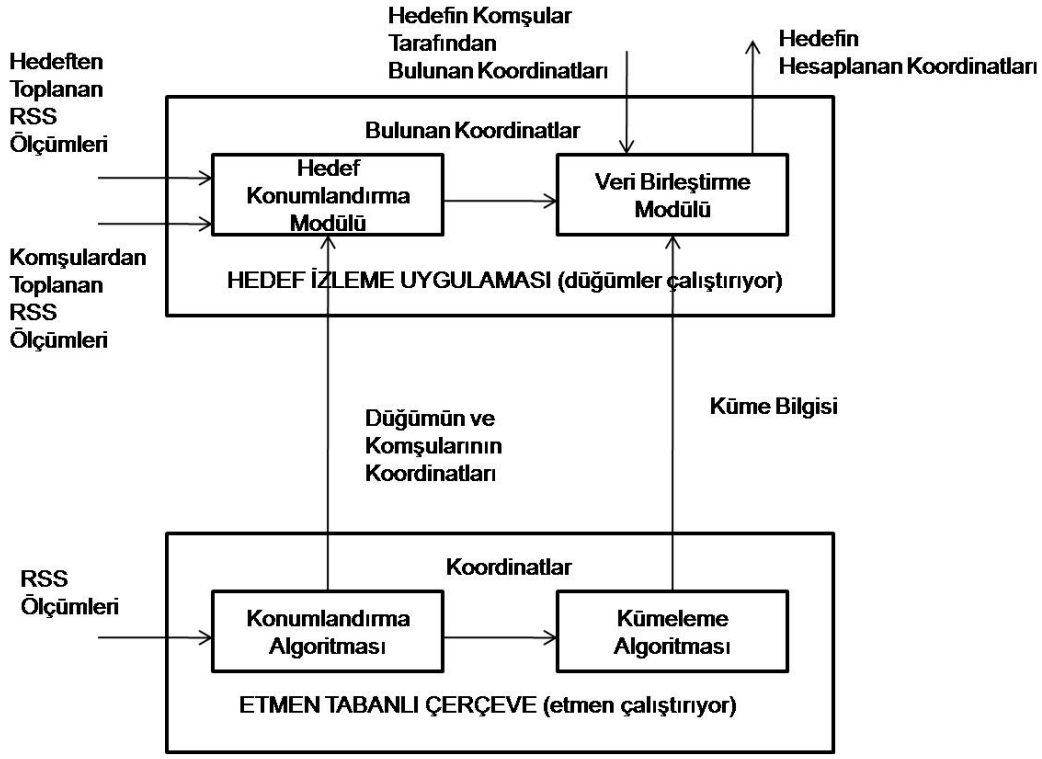
5.7.1. Örnek Uygulamalar

Bu bölümde çerçevemizin kullanılabilineceği konumlandırma ve kümelemenin gerektiği uygun TDA uygulamaları verilecektir. Hedef izleme duyarga ağları için önemli bir uygulama türüdür (Alaybeyoglu et al., 2009). Hedef izleme

uygulamalarında hareketli bir hedefin konum bilgileri duyarga düğümleri tarafından bulunur ve bulunan konum bilgileri çıkış düğümüne doğru yönlendirilir. Bundan dolayı hedef izleme uygulaması çalışmadan önce sıradan düğümlerin konumlandırması işlemi yapılmalıdır. Konumlandırma işlemi zor olabileceğinden dolayı bazı uygulamalarda düğümlerin konumlarını bildikleri varsayılır (Alaybeyoglu et al., 2009).

Düğümlerin kümelenmesi işlemi hedef izleme uygulamalarında ölçeklenebilirlik sağlar. Küme tabanlı uygulamaların en önemli avantajları enerji tüketiminin azalması ve ağ yaşamının uzatılmasıdır. Ayrıca kümeleme sayesinde hedef izleme uygulamalarında veri birleştirme ve yönlendirme kolaylaşır (Alaybeyoglu et al., 2009). Önerdiğimiz çerçeve konumlandırma ve kümelemeyi birlikte sağladığından dolayı Şekil 5.23'de gösterilen dağıtık hedef izleme uygulamasına entegre edilebilir. Genel olarak bir dağıtık hedef izleme uygulaması iki modülü içerir: hedef konumlandırma ve veri birleştirme. Hedef konumlandırma modülü girdi olarak üzerinde çalıştığı düğümün ve komşularının koordinatlarını, hedeften toplanan RSS değerlerini ve komşularının topladığı RSS değerlerini alır. Bu bilgilerin hepsi bizim önerdiğimiz çerçeve tarafından sağlanabilir. Veri birleştirme modülü girdi olarak hedefin düğüm ve komşuları tarafından hesaplanmış konumlarını ve bizim çerçevemiz tarafından sağlanan kümeleme bilgilerini alır. Veri birleştirme modülü konum bilgilerini birleştirerek çıkış düğümüne doğru yönlendirme yapar. Bizim çerçevemiz kapsayan ağaç ve BHK omurgaları üretir ve bu iki yapı da hedef izleme uygulamaları için uygundur (Sharp et al., 2005; Kamath et al., 2007; Zemin and Hai, 2008).

Duyarga ağının yaşam süresini uzatmak uygulamalar için çok önemlidir çünkü düğümler birçok sebepten dolayı işlevini yitirebilir. Hata toleransı etmenin ağı periyodik olarak gezinmesiyle sağlanabilir. Ayrıca etmen sıradan düğümler için pil şarj edici olarak kullanılabilir (Yao et al., 2005). Çerçevemizin bu avantajları olmasına rağmen etmenin ekonomik maliyeti önemlidir. Bu maliyeti hesaplamak ve çerçevemizin kullanılabilir olduğu göstermek için örnek bir uygulamada dağıtık yaklaşımla birlikte karşılaştırarak analiz ettik. TDAda çalışacak bir hedef izleme uygulaması geliştirdiğimiz varsayalım. Uygulamamız düğümlerin konumlandırma ve kümeleme bilgilerine ihtiyaç duyacaktır. Bu işlemleri gerçekleştirmek için iki yöntemi seçebiliriz: bizim çerçevemiz veya dağıtık yaklaşım. Hata toleransını sağlama için, seçilen yaklaşımın periyodik çalışması sağlanmalıdır. Uygulama ve ağ parametreleri aşağıdaki gibi verilmiştir:



Şekil 5.23 Çerçevemizin hedef izleme uygulamasıyla entegrasyonu

- N: Düğüm Sayısı
- Δ: Ortalama düğüm derecesi
- T: Hedef izleme için gerekli çalışma zamanı
- p: Periyot zamanı
- E_N: Bütün düğümlerdeki toplam enerji
- E_M: Bir mesaj transferindeki harcanan ortalama enerji
- C_N: Bütün düğümlerin pillerin değiştirilme maliyeti
- C_A: Etmenin maliyeti
- C_G: Bir GPS alıcısı maliyeti
- G: GPS alıcılı düğümlerin yüzdesi (referans düğümlerin yüzdesi)

Bu parametreleri tanımladıktan sonra gezgin donanım etmeninin ekonomik olarak karşılanabilirliğiyle ilgili analizi dağıtık yaklaşımla kıyaslayarak yapabiliriz. Düğümlerin enerji tüketimin yaklaşık aynı olduğunu ve enerji tüketimlerinin mesaj değişimlerinden kaynaklandığını (Heinzelmen et al., 2000)'da olduğu gibi varsayalım. Bunların yanında düğümlerin pilleri tükenince aynı anda hepsinin birden pillerinin değiştirildiğini kabul edelim. İnsanların girmesinin zor olduğu ortamlara kurulan duyarga düğümleri işlevlerini yitirdiklerinde pillerin değişimi söz konusu değildir, bunun yerine yeni düğümler ağa eklenir. Sonuç olarak iki durumda da

ortaya yeni bir maliyet çıkmaktadır. Maliyet analizimiz GPS alıcılarının ve pillerin maliyeti üzerinden yapacağız. Düğümler hareketsizken dağıtık yaklaşımın maliyeti (C_D) diyelim:

$$C_D = konumlandırma + kümeleme + GPS \quad (5.4)$$

Düğümler hareketsiz oldukları için periyodik konumlandırmaya ihtiyaç yoktur. Diğer taraftan küme liderleri işlevlerini yitirebileceği ve küme üyelerinden kopabileceği için dağıtık kümeleme periyodik olarak yapılmalıdır. Dağıtık kümeleme ve konumlandırma işlemleri (Erciyes et al., 2008, Xiao and Ouksel, 2006) için toplamda güvenli iletişim de hesaba katılırsa ΔN mesaj değişimi gerekebilir. Bunun sebebi güvenli iletişim için alt seviye *ONAY* mesajlarının gerekliliğidir. Aynı zamanda BFS-DSTA gibi protokoller için ΔN mesaj değişimi gereklidir. Dağıtık yaklaşımın toplam maliyeti:

$$C_D = \lfloor \left(\frac{\Delta N E_M}{E_N} \right) \rfloor C_N + \lfloor \left(\frac{\Delta N E_M T}{E_N p} \right) \rfloor C_N + GNC_G \quad (5.5)$$

$$C_D = \lfloor \left(\frac{\Delta N E_M}{E_N} (1 + \frac{T}{p}) \right) \rfloor C_N + GNC_G \quad (5.6)$$

Bizim çerçevemizin çalışması için toplam $3N$ mesaj gerekir. Bundan dolayı bizim çerçevemizin toplam maliyeti (C_F):

$$C_F = konumlandırma\_ve\_kümeleme + etmen \quad (5.7)$$

$$C_F = \lfloor \frac{3NE_m T}{E_N p} \rfloor C_N + C_A \quad (5.8)$$

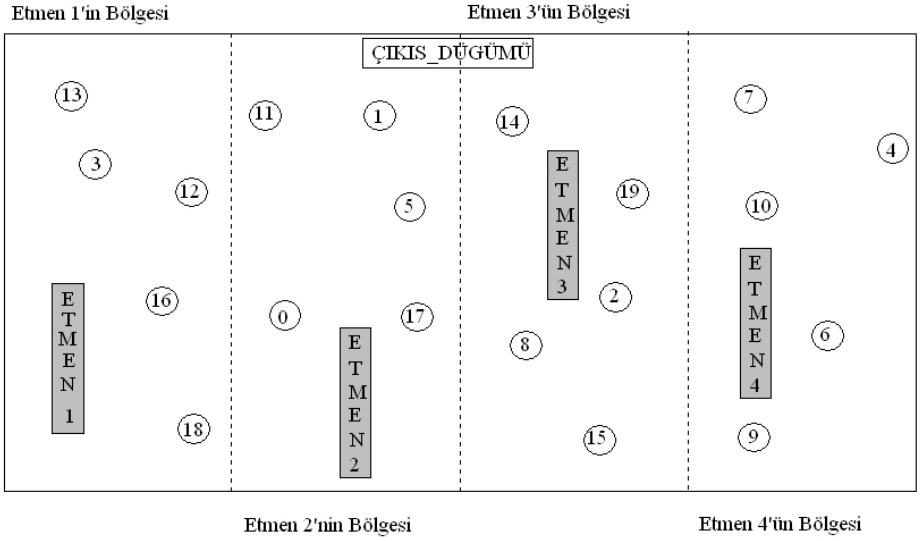
$C_F < C_D$ için denklem 5.6 ve 5.8'den etmenin maliyetini çıkarabiliriz:

$$C_A < \lfloor \left(\left(\frac{(\Delta - 3)T}{p} + \Delta \right) \frac{NE_M}{E_N} \right) \rfloor C_N + GNC_G \quad (5.9)$$

Denklem 5.9'dan yola çıkarak, $\Delta > 3$ için çerçevemizin özellikle ortalama derecesi yüksek olan duyarga ağlarında ekonomik olarak uygulanabilir olduğunu görmekteyiz. GPS alıcı düğüm sayısı arttıkça çerçevemizin dağıtık yaklaşıma göre daha az maliyetli olduğunu Denklem 5.9'dan görmekteyiz. Ayrıca uygulamanın çalışma zamanı arttıkça bizim çerçevemiz dağıtık yaklaşıma göre daha az maliyetlidir.

5.7.2. Hata Toleransı ve Çalışma Zamanının Düşürülmesi

Etmenin işlevini yitirebilmesi bizim çerçevemizdeki en önemli sorunlardan biridir. Bu problemin üstesinden gelebilmek için aynı ağ üzerinde birden fazla etmen ağda aynı anda kullanılabilinir. Bu yaklaşım Şekil 5.24'de verilmiştir. Duyarga alanı bölümlere ayrılmış ve her etmen kendi ayrılmış alanından sorumludur. Etmenler birbirleriyle haberleşip ağdan topladıkları bilgileri paylaşabilirler ve önerdiğimiz çerçeveyi yarı dağıtık olarak uygulayabilirler. Eğer etmenlerden herhangi biri işlevini yitirirse, komşu etmenlerden biri işlevini yitirenin görevini alabilir. Dikkat edilirse sisteme yeni etmen eklenmesi sayesinde her etmenin sorumlu olduğu alan küçüleceğinden dolayı etmenlerin ağı konumlandırma ve kümeleme süresi etmen sayısıyla birlikte azalmaktadır. Örneğin ağa yeni bir etmen daha eklendiğinde eski etmenin sorumlu olduğu alan yarıya düşeceği için çalışma zamanı yaklaşık yarıya düşebilir. Sisteme yeni bir etmen eklendiğinde etmenin maliyeti tüm maliyete eklenir; fakat düğümlerin iletişim maliyeti değişmez.

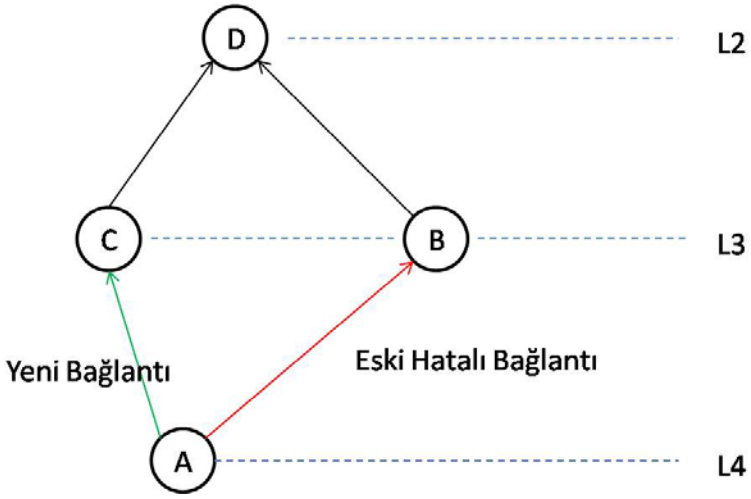


Şekil 5.24 Çoklu etmen yaklaşımı

5.7.3. Kümeleme Hatalarının Düzeltilmesi

Bu bölümde çerçevemizde oluşabilecek konumlandırma hatasından kaynaklanan kümelemede seçilen bağlantı hatalarını düzeltmek için yöntemler önerilecektir. Bu önereceğimiz yöntemler aynı zamanda hata toleransını sağlamak için de kullanılabilinir. AST algoritmasında bağlantısının kopuk olduğunu fark eden bir düğüm komşularına *KEŞİF* mesajı gönderir. Bağlantılarında sorun olmayan (Ebeveynleriyle bağlantı kurabilen) düğümler *KEŞİF* mesajını alınca ağaç üzerindeki seviyelerini *DURUM(düğüm_kimliği, ağaç_seviyesi)* mesajı içinde gönderir. Dikkat edilirse bu durumun sağlanabilmesi için AST algoritmasına seviye bilgisinin eklenmesi gerekir. AST algoritması etmen tarafından işletildiği için seviye bilgisinin eklenmesi sorun oluşturmayacaktır. Seviye bilgisi *ÇIKIŞ_DÜĞÜMÜ'*nün seviyesi 0

olarak kabul edilip eklenir. *DURUM* mesajını alan düğüm yeni ebeveynini bulmak için *DURUM* mesajları içinde kendinden daha küçük veya aynı seviyedeki düğümler içinde RSS'i en yüksek olanını seçer. RSS değeri uzaklıkla birlikte azaldığından dolayı düğümler bağlantıları tamir ederken en küçük kapsayan ağaca olabildiğince yakınlaşmaya çalışır. Şekil 5.25'te örnek bir durum verilmiştir. 4. Seviyedeki düğüm A ile 3. seviyedeki düğüm B arasındaki bağlantı sorunlu olduğu için düğüm A kendisine ebeveyn olarak 3. seviyedeki düğüm C'yi seçmiştir. Bağlantı düzeltmelerinden sonra yeni oluşan ağaç en küçük kapsayan ağaç olmayabilir, hatalı bağlantı sayısı kadar algoritmaya etkide bulunabilir.



Şekil 5.25 AST algoritmasında bağlantı hatalarının düzeltilmesi

ADS algoritmasında oluşan hataları düzeltmek için AST algoritmasında yapılan işleme benzer bir işlem yapılır. ADS algoritmasında bağlı hâkim küme oluşturulduktan sonra etmen merkezi olarak *ÇIKIŞ_DÜĞÜMÜ'*ne doğru hâkim küme içindeki düğümler üzerinden her düğümün seviyesini bildiği bir ağaç oluşturur. Eğer bir düğümün bağlantısı sorunluysa komşularına *KEŞİF* mesajı gönderir. *KEŞİF* mesajını alan düğümler içinde bağlantısı sorunlu olmayan düğümler kimliklerini, ağaç_seviyelerini ve durum bilgilerini (hâkim küme içinde veya değil) *DURUM(düğüm_kimliği, ağaç_seviyesi, düğüm_durumu)* mesajı içinde gönderir. Bir düğüm *DURUM* bilgilerini toparladıktan sonra hâkim küme içinde olan komşuları içinde en küçük ağaç seviyesine sahip olan düğüme bağlanır. Eğer hâkim küme içinde başka komşusu yoksa en küçük ağaç_seviyesine sahip komşusuna *HÂKİM_KÜMEYE_KATIL* mesajı göndererek bağlantısını düzeltir. *HÂKİM_KÜMEYE_KATIL* mesajını alan düğüm hâkim küme içine dâhil olur, böylece hâkim küme korunmuş olur. Bağlantı düzeltmelerinden sonra bağlı hâkim kümenin yaklaşım oranı Guha'nın algoritmasından fazla olabilir, hatalı bağlantı sayısı kadar algoritmaya etkide bulunabilir.

6. SONUÇLAR

Bu tezde TDA üzerinde çalışan çizge teorik küme ve omurga tabanlı iletişim mimarileri verilmiştir. İlk olarak TDA'nın çizge teorik modellenmesi gösterilmiş ve bu model üzerinde kümeleme ve omurga oluşturma problemlerinin tanımları ve bu altyapıların oluşturulma amaçları verilmiştir. Literatürdeki çizge teorik kümeleme ve omurga oluşturma algoritmaları anlatılmıştır. Literatürdeki TDA üzerinde çizge teorik kümeleme ve omurga oluşturma algoritmaları genel olarak kapsayan ağaç tabanlı ve hâkim küme algoritmaları olarak anlatılmıştır. Bu algoritmalar incelendikten sonra çizgeyi kenar ağırlıklı veya düğüm ağırlıklı olarak inceleyen algoritma sayısının çok az olduğu görülmüştür. Ayrıca bu tezde kümelemeye ve omurga oluşturmaya basamak olarak kullanılan çizge eşleme ve konumlandırmayla ilgili literatür özeti verilmiştir. Bu iki işleminin daha önce bir algoritma tarafından kümeleme veya omurga oluşturma için basamak olarak kullanılmadığı görülmüştür.

Bu tezin ilk katkısı literatürdeki çizge teorik algoritmalardan farklı olarak sinyal kalitesi yüksek kanalları kümeleme ve omurga oluşturma işlemi sırasında seçen algoritmaların önerilmesidir. Bu hedefi gerçekleştirmek için literatürdeki algoritmalardan farklı olarak ağırlıklı eşleme tekniği kümeleme ve omurga oluşturma sırasında kullanılmıştır. Bu amaçla öncelikle Hoepman'ın algoritması TDAya uyarlanmıştır. Bu uyarlama sonucunda eşleme kümesi en büyük eşlemeye çok yakın çıkarken, algoritmanın mesaj gönderim sayıları ve zaman ölçümleri ölçeklenebilir ve TDA için uygun değerler olarak ölçülmüştür. Bu noktadan yola çıkarak MASC algoritması önerilmiştir. MASC algoritması Hoepman'ın algoritmasının genişletilmiş versiyonu olup bu algoritma kümeleme yapmak üzere düzenlenmiştir. Dikkat edilirse Hoepman'ın algoritması sadece çizge eşleme yaparken MASC algoritması çizge eşleme tekniğini kullanarak kümeleme işlemini yapmayı hedef alır. MASC algoritması senkron rauntlara bölünmüş olup her rauntta Hoepman'ın algoritması çalıştırılmıştır. Bu algoritmanın tasarımı sonlu durum makinesiyle gösterilmiş ve sonlu durum makinesi üzerindeki durumlar açıklanmıştır. MASC algoritmasının çalışması TDA üzerinde örnek bir uygulamayla gösterilmiş ve algoritma 4 raunt çalıştırılmıştır.

MASC algoritması çizge eşleme tekniğini kümeleme için kullanan ilk yaklaşım olduğundan dolayı önemlidir. Fakat bu algoritmanın senkron olarak çalışması, omurga oluşturmaması ve literatürdeki diğer çizge teorik algoritmalardan daha fazla enerji ve mesaj tüketmesi bu algoritmanın önemli dezavantajları olarak gösterilmiştir. Bu noktalardan yola çıkarak MCUBA algoritması önerilmiştir. MCUBA algoritmasında asenkron çalışmayı, omurga oluşturmayı ve enerji tüketimini azaltmayı sağlayacak geliştirmeler önerilmiştir. Asenkron çalışmayı sağlamak için *İSTEK* mesajları sonradan kullanmak üzere kaydedilmiş ve

küme_üst_seviyesi parametresi eklenmiştir. Omurga oluşturma işlemi kümelemeye benzer olarak yapılmış, bu işlem yapılırken *ÇIKIŞ_DÜĞÜMÜ* sürekli lider olmuş ve kümeleri ağır kenarlar üzerinden birleştirmiştir. Enerji tüketimindeki ve mesaj gönderimindeki azalmalar KMO tekniği kullanılarak yapılmış ve bu teknikle birlikte kümeleme ve omurga işlemleri sırasında mesaj gönderimleri ve buna bağlı olarak tüketilen enerji azalmıştır. Örnek bir uygulama üzerinde MCUBA algoritmasının kümeleme ve omurga oluşturma işlemleri gösterilmiştir.

MASC ve MCUBA algoritmalarının doğruluk analizi, kümeleme kalitesi analizi, seçilen kenar kalitesi analizi, mesaj ve zaman karmaşıklığı analizi ve enerji tüketimindeki eksilmelerin analizi verilmiştir. Algoritmaların doğruluk analizi sonlu durum makinesi üzerinden gidilerek yapılmış, algoritmalarda kilitlenme ve açlık olmadığı gösterilmiş ve algoritmalar sonlandığında kümelerin ve omurgaların ağırlıklı çizge eşleme tekniğiyle doğru olarak yapıldığı ispatlanmıştır. Kümeleme kalitesi analizi yapılırken kümelerin ulaşabileceği seviyeler ispatlanmıştır. R raunt çalışan MASC algoritmasında bir kümenin maksimum seviyesinin 2^R olduğu, *s=küme_üst_seviyesi* olan MCUBA algoritmasında bir kümenin ulaşabileceği en büyük seviyenin $2s$-2 olduğu ispatlanmıştır. Seçilen kenarların kalitesinin yaklaşım oranının r raunda çalışan algoritma için 1/4 olduğu ispatlanmıştır. MASC algoritmasının mesaj karmaşıklığının alt sınırda $\Omega(rN)$, üst sınırda $O(\Delta 2^r N)$ olduğu, MCUBA algoritmasının mesaj karmaşıklığının alt sınırda $\Omega(N)$, üst sınırda $O(\Delta sN)$ olduğu ispatlanmıştır. MASC algoritmasının zaman karmaşıklığı $O(\Delta 2^r r)$ *olduğu,* MCUBA'nın zaman karmaşıklığı alt sınırda $\Omega(\log_2(N))$, üst sınırda $O(s \Delta \log_2(s))$ olduğu ispatlanmıştır. MASC'ın ve MCUBA'nın uzay karmaşıklığının $O(N)$ olduğu ispatlanmıştır. Kulak misafiri olma tekniği MCUBA'da *küme_üst_seviyesi* ≥ 8 olduğunda ortalama bir kümeleme oturumunda % 40'dan fazla enerji tasarrufu sağladığı ispatlanmıştır.

MCUBA ve MASC algoritmaları ns2 benzetim ortamında uygulanmıştır. Bu algoritmaların seçilen kenar ağırlıkları, kümeleme kalitesi, enerji tüketimleri ve çalışma zamanları ölçülmüştür. Bu algoritmalar kapsayan ağaç algoritması olan DSTA ve hâkim küme algoritması olan DS algoritmasıyla karşılaştırılmıştır. Ayrıca ağırlıklı eşleme tekniğinin, ağırlıksız eşleme tekniğine göre kenar seçmedeki etkinliğini göstermek için MOD-MASC algoritması önerilmiştir. Algoritmalar seçilen kenar ağırlıklarına göre MCUBA, MASC, MOD-MASC, DS ve DSTA olarak sıralanmıştır. Dikkat edilecek olursa eşleme tabanlı algoritmalar, kapsayan ağaç ve hâkim küme algoritmalardan daha etkin kenarlar seçtiği ölçülmüştür ve ağırlıklı eşleme tekniğinin ağırlıksız eşlemeye göre de üstünlüğü benzetim sonuçlarından elde edilmiştir. Algoritmaların kümeleme kalitesi küme sayısı ve CV değerleriyle ölçülmüştür. MASC ve MCUBA algoritmalarının DSTA ve DS algoritmalarına oranla daha dengeli ve kontrollü kümeler oluşturduğu ölçümler sonucunda elde

edilmiştir. MASC algoritması karşılaştırılan algoritmalara oranla çok enerji harcasa da MCUBA algoritmasının enerji harcamasının literatürdeki algoritmalara yakın olduğu ölçülmüştür. MASC algoritmasının çalışma zamanını artıran rauntların sonunda kalan boş geçen süreler MCUBA algoritmasında bitirilmiş böylece MCUBA algoritmasının çalışma zamanı literatürdeki algoritmalara yakın ölçülmüştür.

MCUBA ve MASC algoritmalarının üzerinde çalışabileceği örnek bir uygulama tanımlanmış ve uygulamaya ait olan mesaj gönderimi DS ve DSTA ile karşılaştırılarak verilmiştir. Önerilen algoritmaların etkin kenar seçmesinin önemi uygulamaya ve algoritmalara ait parametreler birlikte kurulan denklemlerle gösterilmiştir. Ayrıca MCUBA ve MASC algoritmalarında enerji etkin küme lideri seçiminin yapılması için bir yöntem önerilmiştir. Son olarak önerilen algoritmalara eklenebilecek bir hata tolerans geliştirilmesi verilmiş, böylelikle düğümlerin göçmesi sonrasında da algoritmanın çalışması sağlanmıştır.

Bu tezin ikinci katkısı literatürdeki çok az çalışılmış bir konu olan ağırlıklı bağlı hâkim küme problemi üzerinde çalışıp literatürdeki algoritmalardan daha az ağırlıklı hâkim küme bulan algoritmalar tasarlamaktır. Ağırlıklı bağlı hâkim küme problemi ağ üzerinde en az ağırlığa sahip düğümlerden bağlı hâkim küme oluşturmak olduğundan ve düğümün ağırlığı düğümün enerjisiyle ters orantılı olarak hesaplanabileceğinden dolayı enerji etkin bir omurga olarak ağırlıklı bağlı hâkim küme TDA için çok uygun görülmüştür. Bu noktadan yola çıkarak iki basamaklı olan ve 3 ln (S) yaklaşımlı Guha'nın merkezi ağırlıklı bağlı hâkim küme algoritması dağıtık yapılmıştır. Guha'nın algoritmasının birinci basamağı olan ln (S) yaklaşımlı merkezi ağırlıklı küme örtüsü tabanlı hâkim küme algoritmasının (CENTSET) dağıtık üç versiyonu tasarlanmıştır: FLOODSET, SSET ve ASYNSET algoritmaları. İkinci basamak olan 2 ln (S) yaklaşımlı Klein ve Ravi'nin merkezi Steiner ağacı algoritmasının (CENTTREE) dağıtık üç versiyonu tasarlanmıştır: FLOODTREE, STREE ve ASYNTREE algoritmaları.

FLOODSET algoritması senkron rauntlara bölünmüş ve her rauntta aday düğümler ağırlık oranlarını ağa yaymıştır. Düğümler böylece her rauntta birbirlerinin ağırlık oranlarını öğrenmişler ve en küçük ağırlık oranına sahip düğüm hâkim içine girmiştir. Bu algoritma CENTSET ile aynı düğümleri hâkim küme içine alsa da mesaj karmaşıklığı $O(N^3)$ olarak ispatlandığı için TDA'da çok enerji harcayabileceği gösterilmiştir. Bu dezavantajı gidermek için öncelikle CENTSET algoritmasının üzerinde düzenlemeler yapılmış ve dağıtık yapılmaya çok uygun olan MODSET algoritması tasarlanmıştır. MODSET algoritmasında CENTSET algoritmasından farklı olarak her iterasyonda birden fazla düğüm hâkim küme içine alınmıştır. Bu işlemin sağlanabilmesi için bir düğümün bir iterasyon içinde iki zıplama uzaklığındaki komşularından daha küçükse CENTSET tarafından bulunan hâkim

küme içine gireceği ispatlanmıştır. Böylelikle MODSET'in dağıtık sürümü olan SSET algoritması tasarlanmıştır. SSET algoritması da FLOODSET gibi senkron rauntlara bölünmüş ve her rauntta düğümler iki zıplama uzaklığındaki komşularının ağırlığını öğrenmişler ve her düğüm eğer öğrendiği ağırlık oranlarından daha küçükse hâkim küme içine girmiştir. SSET algoritmasının mesaj karmaşıklığı $O(N^2)$'ye düşürülmüştür. SSET algoritmasında örnek bir uygulama üzerinde gösterildiği üzere raunt sayısı 7 raunttan 4 raunda düşürülmüş, böylece kullanılan süre de azaltılmıştır.

SSET algoritması FLOODSET'le aynı hâkim kümeyi bulsa da tasarımındaki geliştirilmelerden dolayı daha etkin olduğu gösterilmiştir. Fakat SSET algoritması senkron rauntlara bölünmüş olup rauntların tetiklenmesi için zamanlayıcı kullanılması gerekmektedir. Tasarım bir kademe daha geliştirilmiş ve algoritma zamanlayıcı kısıdından kurtarılıp ASYNSET algoritması tasarlanmıştır. ASYNSET algoritmasında β senkronizeri kullanılmış ve *ÇIKIŞ_DÜĞÜMÜ* her raundu *BAŞLA* mesajı göndererek başlatmıştır. Algoritma senkronizeler yardımıyla asenkron olarak tasarlandığı için yarı asenkron olarak isimlendirilmiştir. SSET algoritmasının sonlu durum makinesi üzerinde değişiklikler yapılarak ASYNSET algoritmasında senkronizasyona uygun hale getirilmiştir. ASYNSET algoritmasında SSET algoritmasından fazladan her rauntta senkronizasyon mesajları ve ağın çapı kadar bir süre maliyetleri eklenmiştir. Bunlara karşılık mesaj karmaşıklığı üst sınırda değişmemiştir.

FLOODSET, SSET ve ASYNSET algoritmalarıyla hâkim küme bulunduktan sonra FLOODTREE, STREE ve ASYNTREE algoritmalarıyla birlikte ağırlıklı Steiner ağacı kurularak hâkim küme elemanları birleştirilmiştir. FLOODTREE algoritmasında ağaçlar bilgilerini ağa göndermişler, bu bilgileri toplayan aday bağlayıcı düğümler ağırlık oranını hesaplayıp ağa yaymıştır. Böylece her aday bağlayıcı düğüm birbirinin ağırlık oranını bildiği için Klein ve Ravi'nin algoritmasında olduğu gibi en küçük ağırlıklı düğüm bulunabilmiştir. Fakat üst sınırda bu algoritmanın mesaj karmaşıklığı $O(N^3)$ olarak ispatlandığı için enerji etkin TDA'da sorunlu olabilir. Bundan dolayı STREE algoritması tasarlanmıştır. STREE algoritmasının mesaj karmaşıklığı düşürebilmek için iki tür ağaç önerilmiştir: *HELMAN* düğümlerin kendi aralarında oluşturduğu ağaç ve *ÇIKIŞ_DÜĞÜMÜ*'ne doğru oluşturulan ağaç. Bu ağaçların kullanılmasıyla mesaj karmaşıklığı $O(N^2)$'ye düşürülmüştür. STREE algoritması SSET algoritması gibi senkron rauntlara bölünmüş olup zamanlayıcı yardımıyla çalışır şekilde tasarlanmıştır. STREE algoritması rauntların *ÇIKIŞ_DÜĞÜMÜ* tarafından yayılan bir *BAŞLA* mesajıyla çalışmasını sağlayacak şekilde ASYNTREE algoritması tasarlanmıştır. ASYNTREE algoritmasında *ÇIKIŞ_DÜĞÜMÜ*'ne doğru yönlendirilen ağaç β senkronizeri olarak

kullanılmıştır. Bunun yanında bazı mesajların içerikleri değiştirilmiş asenkron çalışmayı sağlayacak şekilde uygun hale getirilmiştir.

Önerilen ağırlıklı bağlı hâkim küme algoritmalarının doğruluğu, mesaj karmaşıklıkları, zaman karmaşıklıkları ve uzay karmaşıklıkları analiz edilmiştir. Algoritmaların doğruluk analizi sonlu durum makinesi üzerinden yapılmıştır ve doğruluklarını gösterirken CENTSET ve CENTTREE ile aynı düğümleri seçtiği ispatlanmıştır. Ayrıca doğruluk analizinde algoritmalarda kilitlenme ve açlık olmadığı ispatlanmıştır. FLOODSET algoritmasının mesaj karmaşıklığının $\Theta(N^2 |HK|)$'e, alt limitte $\Omega(N^2)$ üst limitte $O(N^3)$'e eşit olduğu, SSET ve ASYNSET algoritmalarının mesaj karmaşıklığının $\Theta(N |P_M|)$'e, üst limitte $O(N^2)$'ye, alt limitte $\Omega(N)$'e eşit olduğu, FLOODTREE algoritmasının mesaj karmaşıklığının B bağlayıcı sayısı olmak üzere $\Theta(N^2 B)$'ye üst sınırda $O(N^3)$'e eşit olduğu, STREE algoritmasının mesaj karmaşıklığının $\Theta(\sum |N_d|^2 + |HK| N)$'e, üst sınırda $O(N^2)$'e eşit olduğu, ASYNTREE algoritmasının mesaj karmaşıklığının GHS algoritması kullanıldığında B bağlayıcı sayısı olmak üzere $\Theta(NB)$, alt sınırda $\Omega(N)$, üst sınırda $O(N^2)$ olduğu ispatlanmıştır. Ayrıca algoritmaların zaman karmaşıklıkları için alt ve üst sınırlarda bulunmuştur. FLOODSET algoritmasının zaman karmaşıklığının $\Theta(D|HK|)$, üst sınırda $O(DN)$ olduğu, SSET algoritmasının zaman karmaşıklığının $\Theta(\Delta|P_M|)$'e alt limitte $\Omega(\Delta)$'e ye üst limitte $O(N\Delta)$'e eşit olduğunu, ASYNSET algoritmasının zaman karmaşıklığının $\Theta((D+\Delta)| P_M |)$'e alt limitte $\Omega(D+\Delta)$'e üst limitte $O(N(D+\Delta))$'e eşit olduğunu, STREE algoritmasının zaman karmaşıklığının $\Theta(NB)$ 'e, üst limitte $O(N^2)$'ye eşit olduğu, ASYNTREE algoritmasının zaman karmaşıklığının düzenlenmiş GHS algoritması kullanıldığında B bağlayıcı sayısı olmak üzere $\Theta(BD+B\Delta+(N_d+B)\log_2(N_d)+N_dB)$'e, üst sınırda $O(N^2)$ olduğu ispatlanmıştır. Ayrıca algoritmaların uzay karmaşıklıkları analiz edilmiştir. Algoritmalarda en kötü durumda kullanılacağı bellek ve mesajların en büyük boyutu $O(HK_T)$ veya $O(\Delta)$ 'e eşit olduğu ispatlanmıştır.

Önerilen algoritmaların ns2 ortamında benzetim yapılmıştır. Hâkim düğümlerin ağırlığı, bağlayıcı düğümlerin ağırlığı, hâkim düğümlerin sayısı, bağlayıcı düğümlerin sayısı, tüketilen enerji miktarları, mesaj gönderimleri ve çalışma zamanları ölçülmüştür. Önerilen algoritmalar literatürdeki algoritmalar ile karşılaştırılmıştır. Önerdiğimiz hâkim küme algoritmaları literatürdeki algoritmalardan daha az ağırlıklı hâkim küme bulmuş ve daha az sayıda hâkim düğümü seçtiği ölçülmüştür. Önerdiğimiz Steiner ağacı algoritmaları Wang'ın algoritmasıyla yaklaşık aynı ağırlıkta bağlayıcılar seçerken daha az sayıda bağlayıcı seçtiği ölçülmüştür. Sonuç olarak önerdiğimiz algoritmalar literatürdeki algoritmalardan daha kaliteli omurga ürettiği benzetim sonuçlarından sonra elde edilmiştir. Buna rağmen önerdiğimiz algoritmaların enerji tüketimleri, mesaj gönderimleri ve zaman kullanımları literatürdeki algoritmalardan daha fazla olduğu

ölçülmüştür. TDA üzerinde enerji tüketimi çok önemli olduğu için bir uygulama geliştirilmiş ve uygulamanın farklı omurga tipleri üzerinde enerji ölçümü yapılmıştır. Benzetim sonuçlarına göre ortalama 30. olaydan sonra Wang'ın algoritmasının senkron algoritmaları yakaladığı, 40. olaydan sonra asenkron algoritmaları yakaladığı ve bundan sonra sürekli farkın arttığı ölçülmüştür. Bu sayede önerdiğimiz algoritmalar omurga oluşturmak için daha fazla enerji harcamasına karşın, uygulama çalışırken daha az enerji harcanmasını sağladığı ölçülmüştür.

Bu tezin üçüncü katkısı TDA için merkezi kümeleme ve omurga oluşturma probleminin çözümü için etmen tabanlı bir çerçevedir. Bu çerçevenin genel fikri bir donanım etmeninin ağda gezinip düğümlerden mesaj toplaması, bu mesajların RSS değerlerini kullanarak konumlandırma yapması ve konumlandırma bilgisini kullanarak kümelemenin yapılmasıdır. Etmen, sıradan düğümlerden çok daha güçlü donanıma sahip olduğu için enerji, bellek ve işlemci gücü harcayan işlemler etmen tarafına kaydırılmıştır. Konumlandırma işleminde (ALOC) trilaterasyon kullanılmıştır. Trilaterasyon tekniğine uygun olarak etmenin zigzag patika üzerinde hareket etmesi önerilmiştir. Kümeleme için iki algoritma önerilmiştir: AST ve ADS. AST algoritmasında dengeli kümeler ve en küçük kapsayan ağaç omurgası oluşturan bir algoritma önerilirken, ADS algoritmasında Guha'nın bağlı hâkim küme algoritmasının uyarlanmış sürümünün kullanılması önerilmiştir.

Etmen tabanlı konumlandırma ve kümeleme çerçevesinin doğruluğu, mesaj ve uzay karmaşıklığı ve çalışma süresi analiz edilmiştir. Konumlandırmanın doğruluğunu analiz ederken etmenin alan üzerindeki hareketi parametrik olarak gösterilmiş ve trilaterasyonun sağlanması için etmenin periyodik olarak ölçüm alma süresi ispatlanmıştır. Kümeleme algoritmalarında Kruskal'ın en küçük kapsayan ağaç algoritması ve Guha'nın bağlı hâkim küme algoritması kullanıldığı için algoritmaların doğrulukları bu algoritmalar üzerinden ispat edilmiştir. Önerilen en küçük kapsayan ağaç tabanlı algoritmanın dengeli kümeleme yaptığı analiz edilmiştir.

Önerilen çerçeve ns2 benzetim ortamı üzerinde uygulandı. Çerçevenin konumlandırma performansı, kümeleme performansı, tüketilen enerji miktarı ve çalışma süresi ölçülmüştür. Çerçevemizin konumlandırma performansının literatürdeki önemli yaklaşımlardan biri olan IT yönteminden daha çok iyi olduğu ölçülmüştür. Ayrıca çerçevemizin kümeleme performansının literatürdeki algoritmalardan daha iyi olduğu görülmüştür. Bunların yanında ölçümler sonucunda enerji tüketiminin çok daha az olduğu gözlemlenmiştir. Çerçevemiz konumlandırma ve kümelemeyi daha az enerjiyle daha kaliteli olarak yaparken literatürdeki algoritmalardan daha fazla zaman harcamıştır. Bunun yanında konumlandırma hatalarından dolayı kümeleme sonucunda oluşan bağlarda %8'lik bir hata ölçülmüştür. Ayrıca etmenin ekonomik maliyetinin önemli bir dezavantaj olduğu

bildirilmiştir. Bu dezavantajları düşünerek örnek bir TDA uygulamasının parametreleri tanımlanmış ve etmenin maliyetinin karşılanabileceği noktalar gösterilmiştir. Çalışma zamanını düşürmek ve hata toleransını sağlamak amacıyla TDAnın alanlara bölünmesi ve yeni etmenler eklenmesi yöntemi önerilmiştir. Son olarak konumlandırma hatasından kaynaklanan bağların tamir edilmesi ve hata toleransının sağlanması için bir yöntem önerilmiştir.

Gelecek çalışma olarak eşleme tabanlı kümeleme ve omurga oluşturma algoritmasının yaklaşım oranını arttıracak bir algoritma üzerinde çalışabilinir. Bunun yanında aynı yaklaşım oranını sağlayan fakat daha az kaynak kullanan bir algoritma üzerinde çalışabilinir. RSS dışında başka bağ göstergeleri seçilip algoritmaların performansları ölçülebilinir. IEEE 802.15.4 gibi TDAya özgü standartlar üzerinde benzetimler yapılabilinir. Benzer olarak ağırlıklı bağlı hâkim küme algoritmalarının yaklaşım oranını düşürecek veya kaynak kullanımı azaltacak algoritmalar tasarlanabilinir. Düğümlerin maliyetleri için sadece düğümlerin enerjileri değil, başka kıstaslar da kullanılıp farklı uygulamalara uygun omurga tipleri oluşturulabilinir. Benzetimler IEEE 802.11 veya IEEE 802.15.4 gibi standartları üzerinde yapılıp, daha gerçekçi bir benzetim yapılabilinir. Önemli bir çalışma alanı da hem kenar ağırlıklarına hem de düğüm ağırlıklarına yaklaşım oranı olan algoritmalar tasarlamak olabilir. Etmen tabanlı çerçevenin konumlandırma ve kümeleme kalitesini arttırmak üzere geliştirmeler yapılabilinir. Konumlandırma için Kalman filtresi kullanılabilinir. Merkezi ağırlıklı bağlı hâkim küme algoritmaları tasarlanıp bu çerçeve üzerinde kullanılabilinir. Merkezi kümelemede hata toleransını sağlamak için *k*-bağlı *m*-hâkim küme (Wu and Li, 2008) algoritmaları önerilebilinir. Ayrıca donanım etmeni ek servisler verecek şekilde genişletilebilinir, kaynak kısıtlı duyarga düğümlerinde dağıtık olarak kodlamak için çok uygun olmayan servisler çerçevemiz sayesinde TDA ortamına uygulanabilir.

KAYNAKLAR DİZİNİ

Akyildiz, I. F. , Su, W. , Sankarasubramaniam, Y. and Cayirci, E., 2002, A Survey on Sensor Networks. IEEE Communications Magazine, 40(8):102-114pp.

Alaybeyoglu, A., Dagdeviren, O., Erciyes K. and Kantarci, A., 2009, Performance Evaluation of Cluster-based Target Tracking Protocols for Wireless Sensor Networks, In Proceedings of the 24th International Symposium on Computer and Information Sciences (ISCIS).

Alhmiedat T. A. and Yang S. H., 2007, A Survey: Localization and Tracking Mobile Targets through Wireless Sensors Network , PGNet.

Alzoubi, K., Wan, P.-J. and Frieder, **O.**, 2002, New Distributed Algorithm for Connected Dominating Set in Wireless Ad Hoc Networks, In Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS).

Awerbuch, B., 1987, Optimal Distributed Algorithms for Minimum Weight Spanning Tree, Counting, Leader Election, and Related Problems, In Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC).

Avis, D., 1983, A Survey of Heuristics for the Weighted Matching Problem, Networks, 3:475–493pp.

Banerjee, S. and Khuller, S., 2001, A Clustering Scheme for Hierarchical Routing in Wireless Networks. Technical Report CS-TR-4103, University of Maryland, Annapolis, MD, USA.

Bao, L. and Garcia-Luna-Aceves, J. J., 2003, Topology Management in Ad hoc Networks, In Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, ACM Press, 129–140pp.

Bharghavan V. and Das, B., 1997, Routing in Ad Hoc Networks Using Minimum Connected Dominating Sets, In Proceedings of the International Conference on Communications'97, 376-380pp.

Biagioni E. and Bridges, K., 2002, The Application of Remote Sensor Technology to Assist the Recovery of Rare and Endangered Species, In Special issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications, 16:3.

Boukerche, A., 2005, Handbook of Algorithms for Wireless Networking and Mobile Computing, Chapman & Hall/CRC.

Chatterjee, M., Das, S. K. and Turgut, D., 2002, WCA: A Weighted Clustering Algorithm for Mobile Ad hoc Networks, Journal of Cluster Computing Special Issue on Mobile Ad hoc Networks, 5:193-204pp.

Chattopadhyay, S., Higham, L. and Seyffarth, K., 2002, Dynamic and Self-Stabilizing Distributed Matching. In Proceedings of PODC'02, 290–297pp.

Chen, Y. P. and Liestman, A.L., 2002, Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks, In Proceedings of MOBICOM'02, 165-172pp.

Choi, L., Lee, S.H. and Choi, H., 2009, M-MAC: Mobility-Based Link Management Protocol for Mobile Sensor Networks, In Proceedings of STFSSD'09, 210–214pp.

Chvatal, V., 1979**,** A Greedy Heuristic for the Set-Covering Problem,Mathematics of Operations Research, 4(3):233–235pp.

Clark , B.N., Colbourn, C.J. and Johnson, D.S., 1990, Unit disk graphs, Discrete Math., 86(1–3):165–177pp.

Cokuslu, D., Erciyes, K. and Dagdeviren, O., 2006, A Dominating Set Based Clustering Algorithm for Mobile Ad Hoc Networks, In Proceedings of ICCSA'06, 571–578pp.

Czygrinow, A. and Hanckowiak, M., 2006, Distributed Algorithms for Weighted Problems in Sparse Graph, J. Discrete Algorithms, 4(4):588–607pp.

Dagdeviren, O. and Erciyes, K., 2006, A Distributed Backbone Formation Algorithm for Mobile Ad Hoc Networks, In Proceedings of ISPA'06, 219–230pp.

Dagdeviren, O. and Erciyes, K., 2010, Graph Matching-Based Distributed Clustering and Backbone Formation Algorithms for Sensor Networks, Oxford The Computer Journal, http://comjnl.oxfordjournals.org /cgi/content/abstract/bxq004v1, (in press).

Dagdeviren, O., Korkmaz, I., Tekbacak, F. and Erciyes K., 2010, A Survey of Agent Technologies for Sensor Networks, IETE Technical Review, (in press).

Doherty, L., Pister, K. S. J. and Ghaoui L. E., 2001, Convex Position Estimation in Wireless Sensor Networks. In Proceedings of IEEE INFOCOM, 3:1655-1663pp.

Dai, F., and Wu, J., 2004, An Extended Localized Algorithm for Connected Dominating Set Formation in Ad Hoc Wireless Networks, IEEE Transactions On Parallel and Distributed Systems, 15(10), 908-920pp.

Drake, D. and Hougardy, S., 2003, Improved Linear Time Approximation Algorithms for Weighted Matchings, In Proceedings of APPROX'03, 14–23pp.

Erciyes, K., Ozsoyeller, D. and Dagdeviren, O., 2008, Distributed Algorithms to Form Cluster Based Spanning Trees in Wireless Sensor Networks, In Proceedings of ICCS'08, 519–528pp.

Fall K. and Varadhan. K., 2010, The Ns Manual. http://www.isi.edu/nsnam/ns/doc/index.html, (Son erişim:22.06.2010)

Feo, T., Goldschimdt, O. and Khellaf, M., 1992, One-half approximation algorithms for the k-partition problem**,** Operations Research, 40:170-173pp.

Gabow, H., 1990, Data Structures for Weighted Matching and Nearest Common Ancestors with Linking, In Proceedings of SODA'90, 434–443pp.

Gallagher, R.G., Humblet, P.A. and Spira, P.M., 1983, A Distributed Algorithm for Minimum-weight Spanning Trees, ACM TOPLAS, 5(1): 66-77pp.

Ganeriwal S., Kumar, R. and Srivastava, M. B., 2003, Timing-sync Protocol for Sensor Networks, In Proceedings of SenSys '03, 138-149pp.

Garay, J., Kutten, S. and Peleg, D., 1998, A Sub-Linear Time Distributed Algorithm for Minimum-Weight Spanning Trees, SIAM J. COMPUT, 659-668pp.

Grimaldi, R. P., 1999, Discrete and Combinatorial Mathematics, An Applied Introduction, Addison Wesley Longman.

Guha, S. and Khuller, S., 1998, Approximation Algorithms for Connected Dominating Sets ,Algorithmica Vol. 20, 374-387pp.

Heinzelman, W. R., Chandrakasan, A. and Balakrishnan, H., 2000, Energy Efficient Communication Protocol for Wireless Microsensor Networks, In Proceedings of the 33rd IEEE Hawaii International Conference on System Sciences, 1-10pp.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. and Pister, K., 2000**,** System Architecture Directions for Networked Sensors, Proceedings of the ACM ninth international conference on Architectural support for programming, 93-104pp.

Hoepman, J. H., 2004, Simple Distributed Weighted Matchings. Nijmeng Institute for Computing and Information Sciences, Radbound University, Nijmegen, The Netherlands, cs.DC/0410047.

Hsu, S.-H, Hsu, C.-C., Lin, S.-S. and Lin, F.-C, 2004, A Multi-Channel MAC Protocol Using Maximal Matching for Ad Hoc, In Proceedings of ICDCSW'04, 505–510pp.

http://www.alertsystems.org, 2010, (Son erişim: 22.07.2010).

IEEE 802.11 StandardWorking Group, 2007, Draft standard for information technology—telecommunications and information exchange between systems—LAN/MAN specific requirements Part 11: wirelessLANmedium access control(MAC) and physical layer (PHY) specifications. IEEE P802.11-REVma/D9.0, IEEE, Washington.

IEEE Std 802.15.4TM-2003, 2003, IEEE standard for information technology—telecommunications and information exchange between systems—local and metropolitan area networks—specific requirements—Part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs). IEEE,Washington.

Jungnickel, **D.,** 2005, Graphs, Networks and Algorithms, 3. edition, ISBN: 978-3-540-72779-8.

Kamath, S., Meisner, E. and Isler, V., 2007, Triangulation Based Multi Target Tracking with Mobile Sensor Networks, In Proceedings of IEEE International Conference on Robotics and Automation, 3283-3288pp.

Karl, H. and Willig A., 2005, Protocols and Architectures for Wireless Sensor Networks, John Wiley & Sons.

Karaata, M. and Saleh, K., 2000, A Distributed Self-stabilizing Algorithm for Finding Maximal Matching. Comput. Syst. Sci. Eng., 3:175–180pp.

Klein, P., and Ravi, P., 1995, A nearly best-possible approximation algorithm for node-weighted Steiner trees, Journal of Algorithms, 19(1):104-115pp.

Kruskal, J. B., 1956, On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In Proceedings of the American Mathematical Society, 7(1):48–50pp.

Kuhn, F., Moscibroda, T. and Wattenhofer, R., 2004, Unit Disk Graph Approximation, In Proceedings of DIALM-POMC'04, 17–23pp.

Lal, D., Manjeshwar, A., Herrmann, F., Uysal-Biyikoglu, E. and Keshavarzian, A., 2003, Measurement and Characterization of Link Quality Metrics in Energy Constrained Wireless Sensor Networks, In Proceedings of GLOBECOM'03, 1:446–452pp.

Langendoen, K. and Reijers, N., 2003, Distributed Localization in Wireless Sensor Networks: A Quantative Comparison, Elsevier Computer Networks, 43:499-518pp.

Lee, H., Dong, H., and Aghajan, **H.,** 2006, Robot-assisted Localization Techniques for Wireless Image Sensor Networks, In Proceedings of IEEE

Conf. on Sensor, Mesh, and Ad Hoc Communications and Networks (SECON).

Lotfinezhad M. and Liang, B., 2005, Energy Efficient Clustering in Sensor Networks with Mobile Agents, In Proceedings of the IEEE Wireless Communications and Networking Conference, 1872-1877pp.

Lotker, Z., Patt-Shamir, B. and Pettie, S., 2008, Improved Distributed Approximate Matching, In Proceedings of SPAA'08, 129–136pp.

Lynch, N., 1996, Distributed Algorithms, Morgan Kaufmann Publishers Inc.

Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D. and Anderson, J.., 2002, Wireless Sensor Networks for Habitat Monitoring. In Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA.

Misra R. and Mandal C., 2010, Minimum Connected Dominating Set Using a Collaborative Cover Heuristic for Ad Hoc Sensor Networks, IEEE Transactions on Paralel and Distributed Systems, 21(3):292-302pp.

Nagpal, R. Shrobe, H. and Bachrach, **J.,** 2003**,** Organizing a Global Coordinate System from Local Information on An Ad Hoc Sensor Network, Proceedings of the Second International Workshop on Information Processing in Sensor Networks, 333-348pp.

Nanuvala, N., 2006, An Enhanced Algorithm to Find Dominating Set Nodes in Ad Hoc Wireless Networks, Master of Science Thesis in the College of Arts and Science, Georgia State University.

Ni , C., Liu , H., Bourgeois, A. G. and Pan, Y., 2005, An enhanced approach to determine connected dominating sets for routing in mobile ad hoc networks, International Journal of Mobile Communications, 3(3):287-302pp.

Niculescu, D. and Nath, B., 2001, Ad-hoc positioning system, In Proceedings of IEEE GlobeCom, San Antonio (USA).

Ning X., 2005**,** A Survey of Sensor Network Applications, IEEE Communications Magazine, 5(5): 774-788pp.

Nieberg, T., 2008, A Local Approximation Algorithm for Maximum Weight Matching, In Proceeding of CTW'08, 44–47pp.

Pathirana, P. N., Bulusu, N., Savkin, A. V. and Jha, S., 2005, Node Localization Using Mobile Robots in Delay-Tolerant Sensor Networks, IEEE Transactions on Mobile Computing, 4(3):285-96pp.

Pettie S. and Sanders, P., 2004, A Simple Linear Time 2/3-Є Approximation for Maximum Weight Matching. Inform. Process. Lett., 91(6):271–276pp.

Peleg D. and Rubinovich V., 2000, A Near Tight Lower Bound on the Time Complexity of Distributed Minimum Spanning Tree Construction, SIAM Journal of Computing.

Polastre, J., Hill, J. and Culler, D., 2004, Versatile Low Power Media Access for Wireless Sensor Networks, In Proceedings of SENSYS'04, 95–107pp.

Preis, R., 1999, Linear Time 1/2-approximation Algorithm for Maximum Weighted Matching in General Graphs, ACM Trans. Algorithms, 1:107–122pp.

Prim, R. C., 1957, Shortest Connection Networks and Some Generalizations, Bell System Technical Journal, 36:1389–1401pp.

Rajendran, V., Obraczka, K. and Garcia-Luna-Aceves, J.J., 2003, Energy-Efficient Collision-Free Medium Access Control for Wireless Sensor Networks, In Proceedings of SENSYS'03, 181–192pp.

Rao, S., 2008, Distributed Systems: An Algorithmic Approach. CRC Press.

Savarese, C., Langendoen, K. and Rabaey, **J.,** 2002, Robust positioning algorithms for distributed ad-hoc wireless sensor networks, In Proceedings of USENIX Technical Annual Conference, 317–328pp.

Savvides, A., Park, H. and Srivastava, M., 2002, The Bits and Flops of the N-hop Multilateration Primitive for Node Localization Problems, In Proceedings of First ACM International Workshop on Wireless Sensor Networks and Application (WSNA), 112–121pp.

Schneider, J. and Wattenhofer, R., 2008, A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs, In Proceedings of 27th ACM Symposium on Principles of Distributed Computing (PODC).

Shang, Y., Fromherz M. P. J., Ruml W. and Zhang, Y., 2003, Localization from Mere Connectivity, In Proceedings of International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), 201-212pp.

Sharp, C., Schaffert, S., Woo, A., Sastry, N., Karlof, C., Sastry, S. and Culler, D., 2005, Design and Implementation of a Sensor Network System for Vehicle Tracking and Autonomous Interception, In Proceedings of the Second European Workshop on Wireless Sensor Networks, 93-107pp.

Schwiebert, L., Gupta, S. K. S. and Weinmann, J., 2001**,** Research Challenges in Wireless Networks of Biomedical Sensors. In Proceedings of Mobile Computing and Networking, 151-165pp.

Song, J.-H., Hong, F. and Guo, Y., 2005, A Distributed Monitoring Mechanism for Mobile Ad Hoc Networks, In Proceedings of ISPAN '05,236-240pp.

Stojmenovic, I., Seddigh, M. and Zunic, J., 2002, Dominating Sets and Neighbor Elimination-based Broadcasting Algorithms in Wireless Networks, IEEE Trans. on Parallel and Dist. Sys., 14-25pp.

Srivastava, M. B., Muntz, R. R. and Potkonjak, M., 2001**,** Smart Kindergarten: Sensorbased Wireless Networks for Smart Developmental Problem-solving Enviroments, In Proceedings of Mobile Computing and Networking, 132-138pp.

Thai, M. T., Wang, F., Liu, D., Zhu, S. and Du, D.-Z., 2007, Connected Dominating Sets in Wireless Networks with Different Transmission Ranges, IEEE Transactions on Mobile Computing, 6(7):721-730pp.

Tong, L., Zhao, Q. and Adireddy, S., 2003**,** Sensor Networks with Mobile Agents, In Proceedings of Military Communications International Symposium, 688-693pp.

Uehara, R. and Chen, Z., 2000, Parallel Approximation Algorithms for Maximum Weighted Matching in General Graphs. Inform. Process. Lett., 76:13–17pp.

Vazirani, V., 2001, Approximation Algorithms, Springer-Verlag, Berlin.

Wang, Y., Wang, W. and Li, X.-Y., 2006, Efficient Distributed Low-Cost Backbone Formation for Wireless Networks. IEEE Trans. Parallel Distrib. Syst., 17(7): 681-693pp.

Wattenhofer, M. and Wattenhofer, R., 2004, Distributed Weighted Matching, In Proceedings of DISC'04, Amsterdam, 335–348pp.

West, D., 2001, Introduction to Graph Theory, Second edition, Prentice Hall, 2001.

Woo, A., Tong, T. and Culler, D., 2003, Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks, In Proceedings of SENSYS'03, 14–27pp.

Wu, J., and Li, H., 1999, A Dominating-set-based Routing Scheme in Ad hoc Wireless Networks. Telecommun. Syst., 3:63–84pp.

Wu, Y. and Li, Y., 2008, Construction Algorithms for k-Connected m-Dominating Sets in Wireless Sensor Networks. In Proceedings of the 9th ACM international Symposium on Mobile Ad Hoc Networking and Computing.

Xiao L. and Ouksel, A. M., 2006, Scalable Self-Configuring Integration of Localization and Indexing in Wireless Ad-Hoc Sensor Networks, In Proceedings of the 7th International Conference on Mobile Data Management.

Yao, W., Li, M., and Wu, N. Y., 2005, Inductive Charging with Multiple Charger Nodes in Wireless Sensor Networks, In Proceedings of the International Workshop on Sensor Networks, 262-270pp.

Ye,W., Heidemann, J. and Estrin, D., 2004, Networking, Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks. IEEE/ACM Trans., 12(3):493–506pp.

Younis, O. and Fahmy, S., 2004, Distributed Clustering in Ad-hoc Sensor Networks: A Hybrid, Energy-Efficient Approach, In Proceedings of INFOCOM'04, 629–640pp.

Youssef, A., Younis, M.F., Youssef, M. and Agrawala, A., 2007, Establishing Overlapped Multihop Clusters in Wireless Sensor Networks, International Journal of Sensor Networks, 2:108–117pp.

Yuanyuan, Z., Xiaohua, J., and Yanxiang, **H.**, 2007, A Distributed Algorithm for Constructing Energy-balanced Connected Dominating Set in Wireless Sensor Networks, International Journal of Sensor Networks, 2:68-76pp.

Zemin, W. and Hai, W., 2008, Target Tracking Based on Connected Dominating Set in WSN, In Proceedings of the 4th International Conference on Wireless Communications, 1-4pp.

ÖZGEÇMİŞ

Türkiye Cumhuriyeti vatandaşı olan Orhan DAĞDEVİREN, 10.10.1982 tarihinde İzmir'de dünyaya gelmiştir. Lisans eğitimini 2004 yılında İzmir Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği'nde tamamlamıştır. Yüksek lisans eğitimini 2006 yılında İzmir Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği Bölümü'nde tamamladıktan sonra, 2006 yılından itibaren doktora eğitimine Ege Üniversitesi Uluslararası Bilgisayar Enstitüsü'nde devam etmektedir. İyi derecede İngilizce bilgisine sahiptir.

2004 yılından itibaren İzmir Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği Bölümü'nde Araştırma Görevlisi olarak görev yapmaktadır. Araştırma alanları dağıtık algoritma ve sistem tasarımı, entegrasyonu ve analizi, kablosuz haberleşme ve ağ protokolü tasarımı, kablosuz duyarga ağları ve gezgin tasarsız ağlar üzerindeki senkronizasyon problemleridir. Bugüne kadar bu konularda çeşitli bilimsel dergi ve kitap serilerinde yayınlanan ve uluslararası yayın indekslerince taranan 10'dan fazla makalesi ve çeşitli ulusal ve uluslararası bilimsel konferanslarda sunulan 10'dan fazla bildirisi bulunmaktadır.