EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

TELSİZ DUYARGA AĞLARI İÇİN ÇİZGE TEORİK TOPOLOJİ KONTROL ALGORİTMALARI

Yasin YİĞİT

Tez Danışmanı: Doç. Dr. Orhan DAĞDEVİREN

Uluslararası Biligisayar Anabilim Dalı

Bilim Dalı Kodu: 619.02.04
Sunuş Tarihi: 15.08.2016

Bornova-İZMİR
2016

Yasin YİĞİT tarafından YÜKSEK LİSANS tezi olarak sunulan “Telsiz Duyarga Ağları İçin Çizge Teorik Topoloji Kontrol Algoritmaları” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 15.08.2016 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	**: Doç. Dr. Orhan DAĞDEVİREN**	**.....**
Raportör Üye	**: Doç. Dr. Muhammed CİNSDİKİCİ**	**.....**
Üye	**: Yrd. Doç. Dr. Hüseyin HIŞIL**	**.....**

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum "Telsiz Duyarga Ağları İçin Çizge Teorik Topoloji Kontrol Algoritmaları" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

.... / .... / 20..

İmzası

Yasin YİĞİT

ÖZET

TELSİZ DUYARGA AĞLARI İÇİN ÇİZGE TEORİK TOPOLOJİ KONTROL ALGORİTMALARI

YİĞİT, Yasin

Yüksek Lisans Tezi, Uluslararası Bilgisayar Anabilim Dalı

Tez Danışmanı: Doç. Dr. Orhan DAĞDEVİREN

Ağustos 2016, 90 sayfa

Telsiz duyarga ağları (TDA) üzerindeki duyargalar yardımıyla çevreden veriler toplayıp bu verileri işleyebilen, birbirleri arasında radyo sinyalleri ile haberleşebilen cihazları içerir. Bu cihazlar pillerinden enerji aldıkları için pillerinin enerjilerinin enerji etkin kullanılması çok önemlidir. TDAda belli patikalar belirleyerek kök düğüme bilgilerin ulaştırılması büyük önem arz etmektedir. Topoloji kontrol algoritmaları ağdaki tepeler arasındaki bağlantıları azaltarak bir omurga oluşturmayı hedefleyen algoritmalardır. Bu bağlamda bir Enine İlk Arama Ağacı (EİAA) oluşturmak ağ üzerindeki mesaj trafiğini azaltabilmektedir.

Tepe örtüsü çizge teorinin önemli problemlerinden biridir. Tepe örtüsü probleminde tepelerden bir küme oluşturarak tepelerin bitişik olduğu bağlantılar örtülür. Tepe örtüsü yardımıyla TDAda güvenli noktalar bulunup bağlantılar izlenir.

Bu tezde, TDAda EİAA ve tepe örtüsünü bütünleşik oluşturmak için yeni algoritmaların tasarımı sunulmuştur. Amaç bu iki yapıyı ayrı ayrı oluşturmak yerine birlikte oluşturarak enerji etkinliği sağlamaktır. Önerilen iki algoritmanın teorik olarak karmaşıklıkları hesaplanmış ve var olan iki algoritma ile mukayese edilerek, aynı şartlar altında alınan benzetim sonuçlarıyla karşılaştırılarak teorik ve pratik olarak irdelenmiştir.

Anahtar kelimeler: Telsiz Duyarga Ağları, Enine İlk Arama Ağacı, Tepe Örtme, Topoloji Kontrol, Dağıtık Algoritmalar

ABSTRACT

GRAPH-THEORETIC TOPOLOGY CONTROL ALGORITHMS FOR WIRELESS SENSOR NETWORKS

YİĞİT, Yasin

MSc in Deparment of International Computer

Supervisor : Assoc. Prof Dr. Orhan DAĞDEVİREN

August 2016, 90 pages

Wireless sensor networks (WSN) consist of devices that are able to collect data from environment by using their onboard sensors, process this data and communicate with each other through radio signals. Since these devices are battery-powered, they should use energy efficiently. Topology control algorithms are algorithms which aim to build a backbone by reducing the number of links in networks. In this context, creating a Breath-First Search Tree (BFST) routing backbone can reduce message traffic.

Vertex cover (VC) is one of the major problems of graph theory. In VC problem, incident edges of vertex are covered by creating a vertex set. On WSN with the help of VC, secure points are found and the edges are monitored.

In this thesis, two algorithms are presented for constructing integrated BFST and VC on WSN. The aim is to ensure the energy efficiency by constructing these two structures together instead of constructing separately. The proposed algorithms are theoretically analyzed and compared with the existing algorithms through simulations.

Keywords: Wireless Sensor Networks, Breath First Search Tree, Vertex Cover, Topology Control, Distributed Algorithm

TEŞEKKÜR

Yüksek lisans eğitimim ve tez çalışmamda bilgi ve deneyimlerini esirgemeyen, verdiği derslerle bakış açımı genişleten, yeni kapılar açmakta bana oldukça yardımı dokunan sayın Doç. Dr. Orhan DAĞDEVİREN'e, Tossim benzetim ortamı hakkında bilgilerini esirgemeyen Sn. Vahid Khalilpour AKRAM'a, yüksek lisans eğitimim boyunca ufkumu genişleten Uluslararası Bilgisayar Enstitüsü öğretim görevlilerine teşekkürlerimi bir borç bilirim.

TÜBİTAK ARDEB 1001 projesi kapsamında 215E115 proje kodlu Dağıtık ve Öz-Kararlı Kapasite Kısıtlı Çizge Teorik Algoritmalar projesi ile bana burs imkanı sağlayan TÜBİTAK'a desteklerinden ötürü teşekkürü bir borç bilirim.

Ayrıca hayatım boyunca benden desteklerini esirgemeyen, beni bugünlere kadar getiren anne ve babama, üniversite hayatım boyunca yanımda olan Derya İNCE'ye en içten sevgi ve teşekkürlerimi bir borç bilirim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

Şekil sayfa

ŞEKİLLER DİZİNİ (devam)

Şekil sayfa

ÇİZELGELER DİZİNİ

Çizelge Sayfa

SİMGELER ve KISALTMALAR

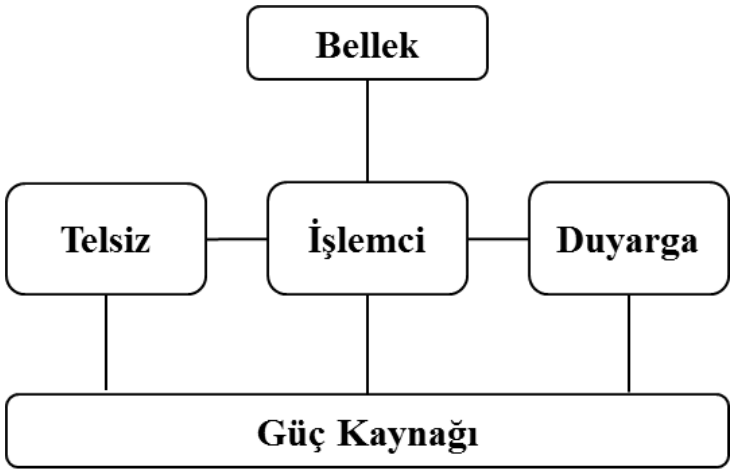
Simgeler	Açıklama
id	düğümünün numarası
Δ	Çizgedeki en büyük derece sayısı
$\in$	Elemanıdır
G	Çizge
E	Kenar
V	Tepe
D	Çizgenin derinliği

Kısaltmalar	Açıklama
TDA	Telsiz Duyarga Ağları
EİAA	Enine İlk Arama Ağacı
inVC	Tepe örtme kümesinde olup olmadığını tutan değişken
VC	Tepe örtme kümesi

1. GİRİŞ

Telsiz Duyarga Ağları (TDA) birden fazla küçük cihazın oluşturduğu ve bu cihazların birbirleriyle veri alışverişinde bulunduğu ağlardır. Telsiz sensör cihazları ortamdan ısı, ışık, nem gibi çeşitli değişkenleri ölçme yetilerine sahip olmakla birlikte, bataryalarının müsaade etmiş olduğu müddetçe veri alışverişinde bulunma, veri saklama, sinyal işleme gibi farklı özelliklere de sahiptirler (Li et al., 2008). Çalışma alanına göre bu cihazların kısıtları da değişmektedir. Cihazlar genellikle kablo ile bir enerji kaynağına bağlı olmayıp üzerinde bulunan piller vasıtası ile çalışmaktadır. Sürekli algılama yapılan bir uygulamada ana problem enerji verimliliği iken, hesaplamaların doğruluğunun ön plana çıktığı bir uygulamada veri kaybının yaşanmaması daha önemli bir problemdir.

TDA bir çok fiziksel parametreyi ölçebildikleri için kullanım alanları da oldukça yaygındır. TDAları, yangın gibi doğal afetlerin, doğal çevrenin ve canlı çeşitliliğinin izlenmesinde, akıllı binalarda ve tesislerde, makina gözetimi ve olası kazaların önlemede, hassas tarımda, tıp ve sağlık alanlarında ve lojistik uygulamalarda yaygın olarak kullanılmaktadır. TDAyı oluşturan duyarga cihazları temel olarak 5 ana bileşenden meydana gelmektedir. Bu bileşenle Şekil 1.1'de verilmiştir.



Şekil 1.1 Telsiz duyarga cihazının donanım mimarisi (Holger et al., 2005).

İşlemci: Kodu çalıştırarak verileri işleyen bileşendir.

Bellek: Programın ve ihtiyaç duyulan verinin saklandığı bileşendir.

Duyarga: Gerçek dünya ile temas eden arayüzdür. Ortamdaki değişkenleri gözlemler.

Telsiz: Kanal üzerinden verilerin gönderilmesini sağlayarak cihazlar arası iletişimi mümkün kılar.

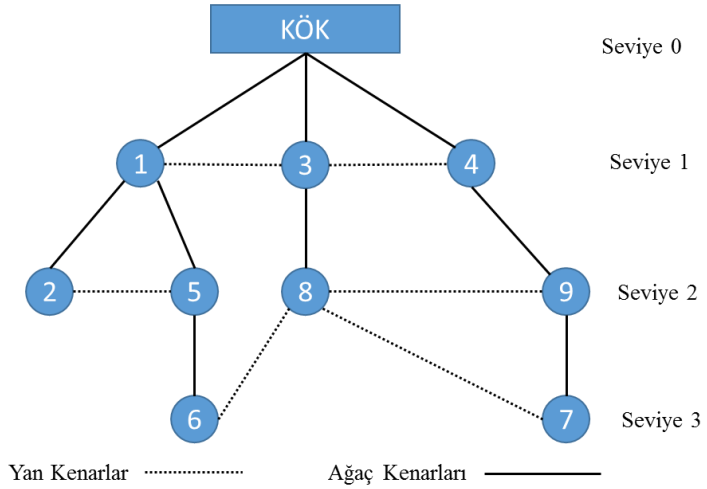
Güç Kaynağı: Genellikle pil olan enerji kısıtlı enerji kaynaklarıdır, bazı durumlarda küçük güneş panelleri de kullanılabilmektedir (Holger et al., 2005).

Telsiz duyarga cihazları küçük boyutlu cihazlar olduğu için, enerji kısıtlı, iletim mesafesinin az olması ve hafıza kısıtları gibi kısıtlamalara maruz kalmaktadır (Gowrishankar et al., 2008). Telsiz duyarga cihazını oluşturan bileşenler mümkün olan en az enerji ihtiyacıyla görevlerini yerine getirmek arasındaki dengeyi sağlamak zorundadır (Holger et al., 2005).

Bu sebeple TDA için geliştirilen algoritmalar da mümkün mertebe enerji verimli ve daha az mesaj transferinin olduğu algoritmalar olmalıdır. Ayrıca belirtmek gerekir ki telsiz duyarga cihazları enerjinin büyük kısmını mesaj transferinde harcarken, hesaplamaya çok daha az enerji harcamaktadır (Holger et al., 2005). Buradan da anlaşılacağı üzere algoritmalar mümkün olduğu kadar ağa çıkmadan cihazda yapılan hesaplamalar yoluyla sonuca ulaşacak tarzda tasarlanmalıdır.

Çevreden toplanan verilerin enerji verimliliğini sağlayacak şekilde çıkış düğümüne gönderilmesini sağlayacak etkin bir ağ topolojisi kurmak TDAdaki en önemli problemlerden biridir (Arapoğlu, 2015). Bu durumlarda daha az yoğun olan bağlı ağlar iletişim için kullanılabilir. Topoloji kontrolü ağın daha az yoğun olan bir alt ağ üzerinden iletişim sağlamasını sağlar. Formal bir tanımla topoloji kontrol, $G(V,E)$ ile tanımlanan bir ağda, herhangi bir $G'(V',E')$ ağı $V' \subset V$ ve $E' \subset E$ şeklinde ifade edilebilir (Erciyes, 2013) .

EİAA topoloji kontrolü için kullanılan önemli yöntemlerden bir tanesidir. EİAA $G(V,E)$ olarak ifade edilen bir çizgede tüm tepelerin içeren ve tepelerin kök düğüme en az zıplama ile ulaşabildiği kapsayan ağaçtır (Erciyes, 2013). EİAA oluştuğunda kök düğüme en uzak olan düğümün uzaklığı bize çizgenin derinliğini vermektedir (Cormen, 2001).

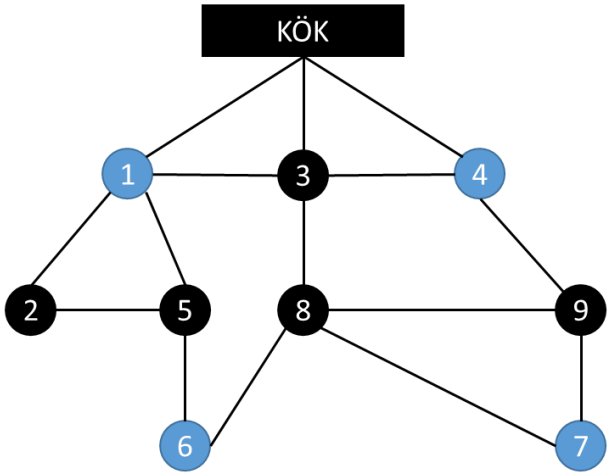


Şekil 1.2 Örnek bir EİAA.

Şekil 1.2'de bir EİAA görülmektedir. EİAA'da kesikli çizgiler yan kenarları ifade ederken, ağacı oluşturan çizgiler düz olarak gösterilmiştir. Örneğin (1,5), (4,9), (5,6) ağaç kenarlarıyken, (3,4), (8,6), (8,9) yan kenarlardır. Her seviyede bulunan düğümler kök düğüme seviyesi kadar uzaklıktadır. 7 numaralı düğüm kök düğüme ulaşmak için 3 kenar geçmek durumundadır ve aynı zamanda bu değer çizgenin derinliğini ifade etmektedir.

EİAA oluşturulurken öncelikle kök düğüm gezilir, daha sonra kök düğümün komşuları ziyaret edilir, bu adımdan sonra çizgede gezilmemiş düğüm kalmayıncaya kadar enine genişleyecek şekilde gezilir. EİAA'nının enine gezilmesinden dolayı ağaçtaki yaprak sayısını yüksek olur. Bu da dolaylı yoldan enerji verimliliğini arttırmaktadır. Ağaçta bulunan ara düğümler mesajları kök düğüme iletmekle görevli oldukları için enerji gereksinimleri daha fazladır. Bu sebeple ne kadar fazla yaprak düğüm olursa mesaj bekleyen ara düğüm sayısı azalır böylece enerji daha verimli kullanılmış olur. Enine arama ağacı hem kök düğüme giden en kısa yolu sağladığı için, hem de oluşturulan ağaçların yaprak sayısı fazlalığı nedeniyle mesaj iletimi için iyi bir kapsayan ağaç türüdür (KeZhong Lu et al., 2005).

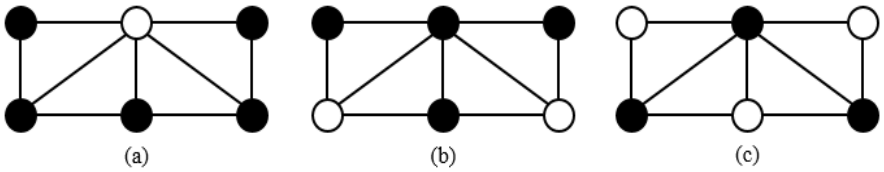
Tepe örtüsü $G(V,E)$ olarak ifade edilen bir yönsüz çizgede $V' \subseteq V$ alt kümesi şeklinde ifade edilen $(u,v) \in E$ kenarlarının uçlarında kalan u veya v tepeleridir (Cormen, 2001).



Şekil 1.3 Tepe örtüsü örneği.

Şekil 1.3'te örnek bir tepe örtüsü görülmektedir. Tepe örtüsü kümesi 6 elemanlı olup, içinde {Kök,2,3,5,8,9} tepeleri bulunmaktadır. Örtülen tepeler sayesinde çizge üzerinde bulunan tüm kenarlar dinlenebilmektedir. Tepe örtüsü probleminin temel amacı tepeleri örterek tüm çizge üzerindeki trafiği izlemektir. Bu sebeple özellikle trafiği izlemek için yerleştirilen kameraların konumunu belirlemede tepe örtüsü probleminden faydalanılmaktadır.

En küçük tepe örtüsü problemini bulmak NP-Tam sınıfında bir problemdir. Aynı şekilde bir çizgenin *k* adet örtülmüş tepesinin olup olmadığına karar vermek de NP-Zor problemdir. Tepe örtüsü problemi Karp tarafından ispatlanan 21 NP-Tam problemden birisidir (Karp, 1972).



Şekil 1.4 Tepe örtüsü örnekleri a)Tepe örtüsü b)Minimal tepe örtüsü c) Minimum tepe örtüsü (Erciyes, 2013).

Şekil 1.4 üzerinde minimum ve minimal tepe örtüsü görülmektedir. Minimum tepe örtüsü çizge için tüm tepe örtüsü kümeleri içerisinde en ideal tepe örtüsü kümesidir. Minimal tepe örtüsü kümesi ise bulunan küme içerisinde bir tepe çıkarılması durumunda tepe örtüsünün bozulacağı kümeleri ifade eder. Minimum tepe örtüsü kümesi minimal tepe örtüsü kümesi olabileceği gibi, bu durumunun tersi geçerli değildir.

Bu tezde, EİAA ve tepe örtüsü problemlerini bütünleşik olarak çözmek amacıyla iki yeni algoritma önerilmiştir. Algoritmaların amacı bu yapıları ayrı ayrı oluşturmak yerine bütünleşik olarak oluşturup daha az bit transferi yaparak enerji tasarruf sağlamaktır. Önerilen iki algoritma Çift Öncelikli Tepe Örtme Algoritması ve Yan Kenar Algoritması olarak isimlendirilmiştir. Bu algoritmaların teorik analizleri yapılmış ve TOSSIM benzetim ortamında kodlanmıştır. Önerilen algoritmaların performanslarını karşılaştırmak amacı ile Açgözlü (Erciyes, 2013) ve KUD (Kavalcı et al., 2014) algoritmaları TOSSIM ortamında kodlanmıştır. Kodlanan algoritmalar alınan ve gönderilen bayt, tepe örtüsü kümeleri, enerji sarfiyatı, zaman gibi parametreler yönünden karşılaştırılmış olup, birbirleri üzerinde olan üstünlükleri irdelenmiştir.

Tez kapsamında, GİRİŞ bölümünde problemin açıklaması ve tez sonucunda beklenenler açıklanmış olup İLGİLİ ÇALIŞMALAR başlığı altında tepe örtüsü problemi için önerilen çizge eşleme ve EİAA bütünleşik algoritmalar örneklerle birlikte açıklanmıştır. ÖNERİLEN ALGORİTMALAR bölümünde Çift Öncelikli Tepe Örtüsü Algoritması ve Yan Kenar Tepe Örtüsü Algoritmasına ait sözde kodlar verilerek örnek çalışmaları detaylı bir şekilde verilmiştir. TEORİK ANALİZ başlığı altında önerilen algoritmaların zaman, bit ve yer karmaşıklıkları teorik olarak analiz edilerek, diğer algoritmalarla karşılaştırılmıştır. Alınan sonuçlar doğrultusunda BENZETİMLERİN DEĞERLENDİRMESİ başlığı altında algoritmaların çıktıları değerlendirilmiş ve SONUÇ kısmında da tezden alınan sonuç ve ilerleyen zamanda yapılacak çalışmalar belirtilmiştir.

2. İLGİLİ ÇALIŞMALAR

Dağıtık tepe örtüsü probleminde iki temel yaklaşım vardır. Bunlar maksimal eşleme ve en yüksek dereceli tepeden başlayarak örtme yapan açgözlü yaklaşımdır. Geliştirilen birçok algoritma bu iki temel üzerine kurulmaktadır. Maksimal eşleme kümesi, aynı tepeyi paylaşmayan kenarların oluşturduğu bir kümedir. Maksimal eşleme ile yapılan tepe örtüsü 2 yakınsama oranına sahiptir çünkü bir kenar iki farklı tepe tarafından gerek olmadığı halde örtülebilir. Açgözlü yaklaşım için belli bir yakınsama oranı olmamasına rağmen pratikte maksimal eşlemeden iyi sonuçlar verdiği gözlemlenmiştir (Kiniwa, 2005).

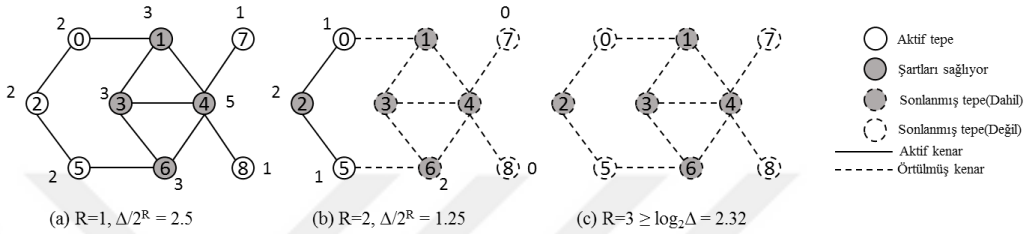
Parnas ve Ron tarafından önerilen Açgözlü tepe örtüsü algoritmasının temel hedefi, çizge üzerinde komşuları arasında en fazla dereceye sahip olan tepeleri bulup, bulunan bu tepeleri tepe örtüsü kümesine dahil etmektir. Tüm tepeler komşuları arasında en büyük dereceli olup olmadıklarına bakıp kendilerini tepe örtüsü kümesine dahil edip, aday kümesinden çıkartırlar. Algoritma sonlanma koşulu sağlanana kadar döngüler vasıtasıyla devam eder. Sonlanma koşulu aşağıda sıralanan maddelere bağlıdır (Parnas ve Ron, 2007).

- Her tepe çizgedeki en büyük dereceyi bilmektedir.
- Her tepe kendi komşularını bilmektedir.
- Her tepe komşularıyla işletişimde olduğunun garantisini mesajlar ve geri bildirim yolu ile garanti etmektedir.
- Raunt sayısını tüm tepeler mesajlar ile öğrenmektedir.

```
/* Tüm tepelerde */
Mesaj tipi: BIRAK
R←0 (Raunt sayacına sıfır atanır)
Raunt başladığında
        R ← R+1
        eğer R > log2(Δ) ise sonlan
        eğer derece > (Δ / (2^R) ise
                Tepe örtme kümesine dahil ol
                Komşularına BIRAK mesajı gönder
Mesaj alındığında:
        eğer BIRAK mesajı alındı ise,
        mesajı gönderen tepeyi komşuluk listesinde sil
```

Şekil 2.1 Dağıtık Açgözlü minimum aday tepe örtüsü algoritması.

Şekil 2.1'de sözde kodu verilen algoritmanın çalışma örneği Şekil 2.2'de görülmektedir. Örnek çizgede en büyük derece yani Δ değeri 5tir. İlk raunt için $R > log_2(\Delta)$ koşulunu sağlayan hiç bir tepe bulunmamaktadır. Bunun yanı sıra $\Delta / (2^R) = 2.5$ olduğu için dereceleri bu değerin üzerinde olan 1, 3, 4 ve 5 numaralı tepeler tepe örtüsü kümesine dahil olurlar. İkinci raunt başladığın $\Delta / (2^R)$ ifadesi 1.25 olacağı için bir önceki rauntta tepe örtüsü kümesine dahil olmayan derecesi bu değerden büyük olan 2 numaralı tepe kümeye dahil olur. Raunt 3'te $R > log_2(\Delta)$ koşulu sağlandığı için algoritma sonlanır.



Şekil 2.2 Açgözlü minimum aday tepe örtüsü algoritması çalışma örneği (İleri et al., 2015).

Dağıtık Açgözlü Minimum Aday Tepe Örtüsü algoritmasının yakınsama oranı $(2.\log_{2(\Delta)}+1)$dır (Parnas ve Ron, 2007). Böylece en kötü durumda algoritmanın yakınsama oranı $O(\log_2(\Delta))$dır.

G(V,E) ile ifade edilen bir çizgede, *M* aynı anda iki tepeye komşu olmayacak şekilde $M \in E$ kümesini bulma işlemi çizge eşlemedir. Açgözlü ve asenkron olarak Hoepman tarafından 2004 yılında önerilen bu çizge eşleme algoritması, yerel olarak en yüksek ağırlıktaki kenarları eşleyerek bu kenarın ucunda kalan tepeleri çizgeden çıkarma mantığına dayalıdır. Bu algoritma küçük bir düzenleme ile tepe örtüsü problemine de uygulanabilir. Şöyle ki, eşleme içinde olan kenarların ucunda kalan tepeleri doğrudan tepe örtüsü kümesi içine alarak aynı çizgede tepe örtüsü problemi de çözülmüş olur. Kenarlara ait ağırlıklar tepe örtüsü problemini çözmek için ihmal edilebilirdir. Bu algoritma da Dağıtık Açgözlü Minimum Aday Tepe Örtüsü algoritmasında olduğu gibi Δ değeri dışındaki tüm kabullenmeleri yaparak sonlanma koşulu sağlanana kadar çalışır.

```
/* Tüm tepelerde */
Mesaj Tipi: ÖNER, BIRAK
aday ← boş, inVC ← doğru, durum ← AKTİF
//Algorima başlar
Tek numaralı rauntlarda
        //aday seç
        komşularının derecesini öğren ve bir komşuluk listesi oluştur
        C ← daha düşük dereceli komşularını seç
        aday ← C kümesinden en büyük dereceli olan komşunu seç
        //Matching
        aday tepeye ÖNER mesajı gönder
        raunt sonlanana kadar
                ÖNER mesajını j'den al
                eğer j = aday // j ve ben birbirimizi aday gösterdiysek
                  eğer raunt = 1
                     eğer (d(ben) > d(j)) | {d(ben) = d(j) & (id(ben) > id(j))}
                       inVC ← doğru

                      değilse
                       inVC ← yanlış
                değilse
                        inVC ← true
                status ← İNAKTİF
Even-numbered round
        eğer durum = İNAKTİF
                tüm komşularıma BIRAK mesajı gönder
                sonlan
        raunt sonlanana kadar
        BIRAK mesajını j'den al
        j tepesini komşuluk listenden çıkar
        eğer hiç bir aktif komşum kalmadıysa
                durum ← İNAKTİF
                sonlan
```

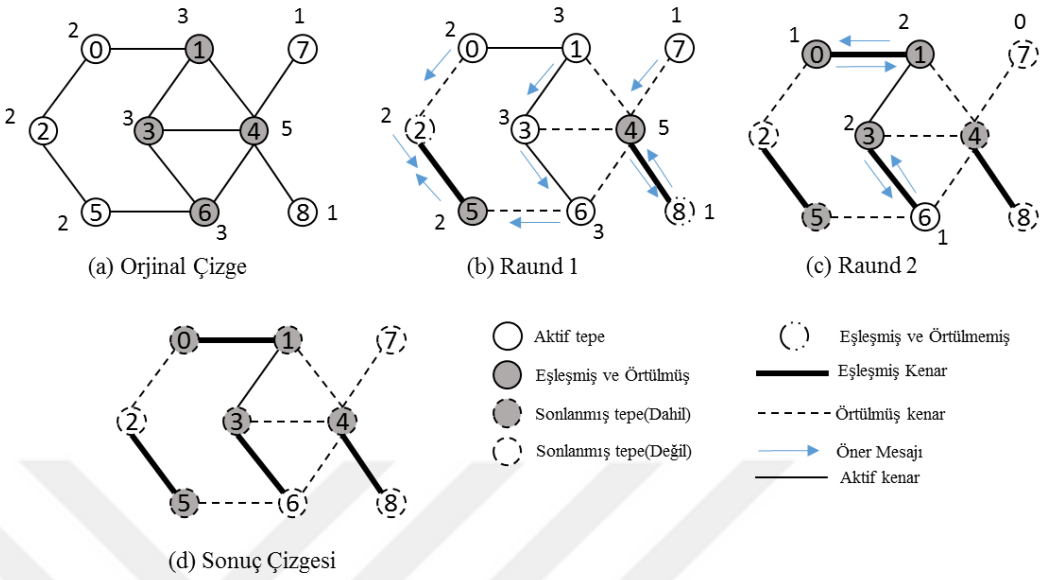
Şekil 2.3 Açgözlü eşleme ile dağıtık tepe örtüsü algoritması.

Açgözlü Eşleme ile Dağıtık Tepe Örtüsü algoritmasının yakınsama oranı, Kavalcı ve arkadaşları tarafından yapılan denemelerde 1 ile 1.6 arasında değişmektedir (Kavalcı et al., 2014). Hoepman tarafından eşleme algoritmasının bir araç olarak kullanıldığı Açgözlü Eşleme ile Dağıtık Tepe Örtüsü algoritmasının yakınsama oranı 2 olarak verilmiştir (Hoepman, 2004).

Algoritmanın sözde kodu ve çalışma örneği, Şekil 2.3 ve

Şekil 2.4'te verilmiştir. Raunt dereceleri ile birlikte gösterilen tepeler birbirlerine *ÖNER* mesajı göndererek eşleşme talebinde bulunurlar. Karşılıklı birbirini öneren (2,5) ve (4,8) tepe çiftleri eşleşerek derecesi büyük olan tepeler

tepe örtüsü kümesine dahil olurlar. Ayrıca eşleşemeyen tepeler derecelerini bir azaltırlar bu sayede diğer



Şekil 2.4 Açgözlü eşleme ile tepe örtüsü algoritmasının çalışma örneği (İleri et al., 2015).

raunda hazırlık yapılmış olur. 2. Raunt başladığında karşılıklı öneri yapan (1,2) ve (3,6) tepe çiftleri eşleşerek derecesi büyük olan tepeler örtülür. Örtülmeyen kenar kalmadığı için algoritma bu aşamada sonlanır. Sonlanmış çizge

Şekil 2.4.e'de görülmektedir.

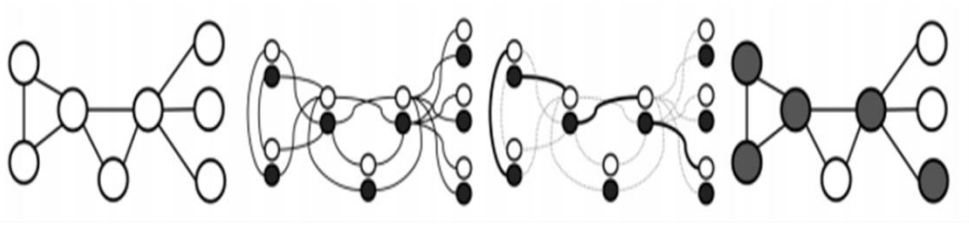
Açgözlü Eşleme algoritmasını kullanarak yapılan tepe örtüsü işleminde algoritmanın karmaşıklığı maksimal eşleme algoritması tarafından belirlenmektedir. Maksimal eşleşmeyi bulma işlemi ikili ve tüm tepelerin rengini bildiği çizgelerde genel çizgelere göre daha kolay olduğu için Hanckowiak ve arkadaşları bu çıkarımdan faydalanarak O(Δ) zaman karmaşıklığında genel çizgelerde maksimal eşleme yapmayı başarmışlardır (Hanckowiak, 1997). Polishchuk ve Suomela (2009) bu yaklaşımı farklı bir tarzda kullanarak O(Δ) zaman karmaşıklığında tepe örtüsü kümesini 3 yakınsama oranı ile çözmüşlerdir. Genel bir çizgeyi ikili bir çizgeye dönüştürdükten sonra eşleme ile tepeler örtülmektedir.

G(V,E) ifade edilen bir genel çizge, $G'(V'_1,V'_2,E)$ şeklinde ifade edilerek,

- Her bir tepe $v \in V$, $v'_1 \in V'_1$ ve $v'_2 \in V'_2$

- Her bir kenar $(u,v) \in E$, $(u'_1,u'_2) \in E'$ ve $(v'_1,v'_2) \in E'$ olacak şekilde ikiye ayrılır.

Diğer bir deyişle çizgedeki tüm tepe ve kenarların birer kopyası alınır. Bu kopyaların biri siyah diğeri beyaz olarak adlandırılmıştır. Her bir siyah kopya bir beyaz ile eşleştirilir.



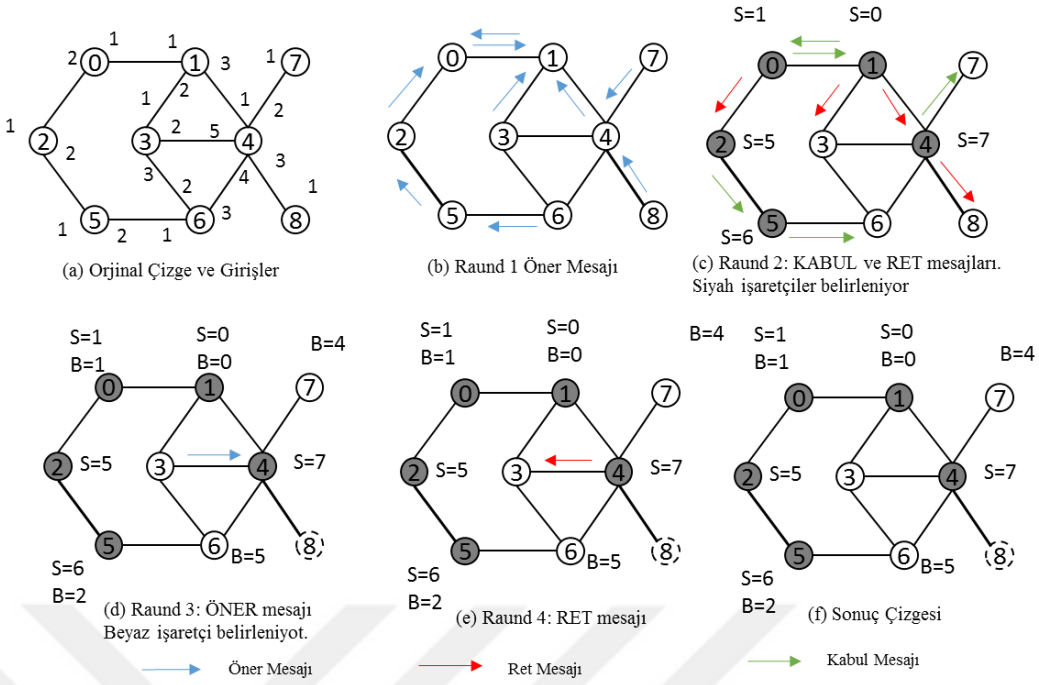
(a) Orjinal çizge G (b) G' ikili çizgesi (c) G' üzerinde maksimal eşleşme (d) Tepe örtme sonucu

Şekil 2.5 Genel bir çizgeden ikili çizge elde edilerek tepe örtüsü örneği (İleri et al., 2015).

```
/* Her bir tepe için*/
Mesaj Tipleri: ÖNER, KABUL, RET
beyaz ←boş, siyah ← boş, i = 0, inVC ←yanlış
Tek numaralı rauntlarda //Beyaz işaretçiler ayarlanır
        eğer (beyaz = null & 1 ≤ i ≤ d ) // d çizgedeki en büyük
derece
                I girişinden mesajı al // beklemeden al
                eğer mesaj KABUL ise
                        white ←i
                        inVC ←true
        eğer (beyaz = null & i ≤ d)
                i = i + 1
                i portuna ÖNER mesajı gönder


Çift numaralı rauntlar //Siyah işaretçiler ayarlanır
        //beklemeden al
        herhangi bir komşudan ÖNER mesajı al ve komşunu P
kümesine ekle
        eğer siyah=null
                p ∈ P en küçük giriş numaralı komşunu seç
                p tepesine KABUL mesajı gönder
                P-p kümesine RET mesajı gönder
                inVC ← doğru
        değilse
                P kümesine RET gönder
```

Şekil 2.6 İkili eşleme ile dağıtık tepe örtüsü algoritması.



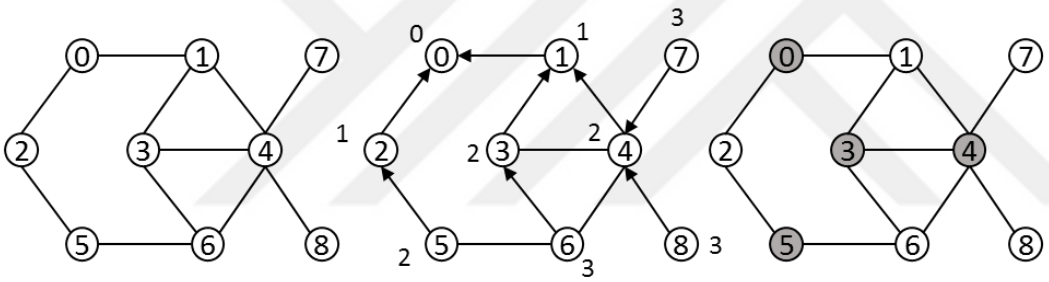
Şekil 2.7 İkili eşleme ile tepe örtüsü algoritmasının çalışma örneği (İleri et al., 2015).

Genel bir çizge ikili bir çizgeye dönüştürüldükten sonra tepe örtüsü yapabilmek ikili çizgede kopya tepelerden biri eşleme içerisinde ise genel çizgede de tepe örtüsü kümesinden yer almaktadır. Algoritmanın sözde kodu Şekil 2.6'da verilmiştir. Her bir tepenin siyah ve beyaz olmak üzere, karşılıklı olarak birbirlerini göstermek için iki adet işaretçisi bulunmaktadır. Eğer *u* tepesinin siyah işaretçisi *v* tepesini işaret ediyorsa, *v* tepesinin beyaz kopyası ile eşleşmiş anlamına gelmektedir. Ayrıca derecesi *d* olan bir tepe için 1'den *d*'ye kadar olan giriş numaraları mevcuttur. Şekil 2.5'te kabaca tasvir edilen yaklaşımın Şekil 2.7 detaylı adımlarını göstermektedir.

Şekil 2.7.a'da görüldüğü üzere her bir tepenin derece sayısı kadar girişi bulunmaktadır ve bu girişleri 1'den başlayarak numaralandırmışlardır. Tüm tepeler en küçük numaralı girişlerinden birbirlerine Şekil 2.7.b'deki gibi *ÖNER* mesajı atarlar. *ÖNER* mesajları toplandıktan sonra Şekil 2.7.c'de 2. Raunt başında *ÖNER* mesajlarının geldiği en küçük girişe *KABUL* mesajı , diğer girişlere *RET* mesajı gönderilir. Bu aşamada *KABUL* mesajı atan tepeler mesajı attıkları tepeyi siyah işaretçisi olarak ayarlarlar. Şekil 2.7.d'de görüldüğü gibi *KABUL* mesajı alan tüm tepeler mesajın kaynağını beyaz işaretçisi olarak ayarlarlar. Siyah ya da beyaz işaretçisini belirlenmiş olan tepeler tepe örtüsü kümesine dahil olurlar. Böylelikle tepe örtüsü sonucu Şekil 2.7.e'de görülmektedir.

Bu tez kapsamında kodlanan ve ölçümleri alınan, aynı zamanda önerdiğimiz iki algoritmaya örnek teşkil eden EİAA ile Dağıtık Tepe Örtüsü Algoritması Kavalcı ve arkadaşları tarafından önerilmiştir (Kavalcı et al., 2014). Bu algoritmada öncelikle bir EİAA oluşturulup tepe örtüsü işlemi bu ağaç kullanılarak yapılır. EİAA'nın TDA için çok önemli bir yönlendirme ağacı olduğundan bahsetmiştik. Bir TDA üzerinde oluşturulan EİAA üzerinde tüm tepeler komşularına seviye bilgilerini gönderdikten sonra tepeler, tepe örtüsü kümesi içinde olup olmadıklarına karar verirler. KUD algoritmasında ağaç oluşturulduktan sonra, çift seviyeli tepeler doğrudan tepe örtüsü kümesine dahil edilirken, tek seviyeli tepeler tepe örtüsü kümesine girebilmek için kendileri ile aynı seviyede daha küçük bir numaraya sahip komşu tepe ararlar. Algoritmanın sözde kodu Şekil 2.9'da verilmiştir.

Şekil 2.8'de Kavalcı ve arkadaşlar tarafından önerilen Enine İlk Arama Ağacı İle Dağıtık Tepe Örtüsü algoritmasının çalışma örneği görülmektedir.



Şekil 2.8 Enine ilk arama ağacı ile tepe örtüsü algoritmasının örneği (İleri et al., 2015).

Şekil 2.8.a'da orijinal çizge görülmektedir. 0 numaralı tepe algoritmayı başlatan tepedir. 0 numaralı tepe ağacın en üst noktasını oluşturacak şekilde EİAA oluşturulduktan sonra çift seviyeli 0, 3, 4 ve 5 numaralı tepeler doğrudan kapsanmıştır. Diğer tepeler tek seviyeli olmasına rağmen aynı seviyede komşuları bulunmadığı için algoritmanın kalan kısmı çalıştırılmamıştır.

Başlangıçta:
Kök tepe:
Kendi komşuluk listesini oluşturur
"Seviye 0" altyapı mesajını tüm komşularına gönderir
Sayaç T0 mesaj gönderimi için periyodik olarak ayarlanır.
Sayaç T2 ebeveyn ve seviye bilgisini göndermek için tek seferlik kurulur.
Diğer tepeler:
Kendi komşuluk listesini oluşturur
Sayaç T0 mesaj gönderimi için periyodik olarak ayarlanır.
Sayaç T2 ebeveyn ve seviye bilgisini göndermek için tek seferlik kurulur.
T0 tetiklendiğinde:
Eğer kuyrukta bir mesaj varsa ve KABUL mesajı beklenmiyorsa,
Kuyruktan bir mesaj gönder.
T1 sayacını tek seferlik KABUL mesajı beklemek için ayarla.
T1 tetiklendiğinde:
KABUL mesajı gelmediği için mesajını tekrar gönder.
T1 sayacını tek seferlik KABUL mesajı beklemek için ayarla.
Gönderme başarılı olduğunda:
//Mesajın iletildiğinden eminiz
T1 sayacını durdur.
Kabul_Bekleniyor = Yanlış.
T2 tetiklendiğinde:
//Enine İlk Arama Ağacı oluşturuldu
//Tüm tepeler ebeveyn ve seviye bilgilerini biliyor.
Sayaç T3 tepe örtme yapmak için kur
Tüm tepeler birbirine seviye bilgilerini gönderecek.
T3 tetiklendiğinde:
//Tüm tepeler komşuları hakkında gerekli bilgilere sahip
Eğer benim "seviye" çift ise doğrudan tepe örtme kümesi içerisindeyim.
Eğer benim "seviye" tek ise,
Aynı seviyede olduğum benden küçük numaralı bir komşu ara
Eğer bulursan büyük numaralı tepeyi tepe örtme kümesine dahil et.
Bir mesaj alındığında:
Altyapı mesajı ise:
If Seviye > Gönderen_seviye +1 *//daha küçük seviyeli bir ebeveyn*
Seviye = Gönderen_seviye +1
Ebeveyn = Gönderen
Seviye bilgisini tüm komşularına gönder
Seviye mesajı ise:
Gönderen mesajın numarasına göre komşuluk listeni güncelle.

Şekil 2.9 Enine ilk arama ağacı ile dağıtık tepe örtüsü algoritması.

KUD algoritmasının en iyi sonuca yakınsama oranı 1 ile 2 arasında değişmekle birlikte çeşitli deneysel çizgelerde 1.5 değerini geçmediği görülmektedir (Kavalcı et al., 2014).

3. ÖNERİLEN ALGORİTMALAR

Bu tez kapsamında EİAA'nı kullanan iki tepe örtüsü algoritması önerilmiştir. Algoritmalardan ilki Enine İlk Arama ağacı oluştuktan sonra tepelerin seviyelerine bağlı olarak tepe örtüsü bulma işlemi gerçekleştirir. Şöyle ki EİAA oluşturulduktan sonra bir tepe eğer çift seviyeli ise bu tepe kendisini doğrudan kapsar ancak tek seviyeli ise seviyeler arasından Açgözlü Algoritma çalıştırılır. Bu algoritmayı Çift Öncelikli Algoritma olarak isimlendirirken, diğer algoritmamızın ismi Yan Kenar olarak isimlendirilmiştir. Önerdiğimiz ikinci algoritmada ise ağaç oluşturulurken *GERİ* mesajları ile tüm tepelerin yan kenar sayılarını içerecek şekilde gönderilir. Bu sayede kök tepe hangi seviyelerde ne kadar yan kenar olduğunu bilir. Bu sayılara göre yan kenar sayısı fazla olan çift ya da tek sayılı seviyelerdeki tepelerin doğrudan kapsanmasına karar verir. Bunun için bir *SEÇİM* mesajı göndererek tüm tepeleri doğrudan kapsanacak olan seviye hakkında bilgilendirir. *SEÇİM* mesajını alan tepeler eğer seçilmiş seviye içerisindeler ise doğrudan tepe örtüsü kümesine dahil olurlar. Aksi halde seviyeler arasında Açgözlü Algoritma çalıştırılır.

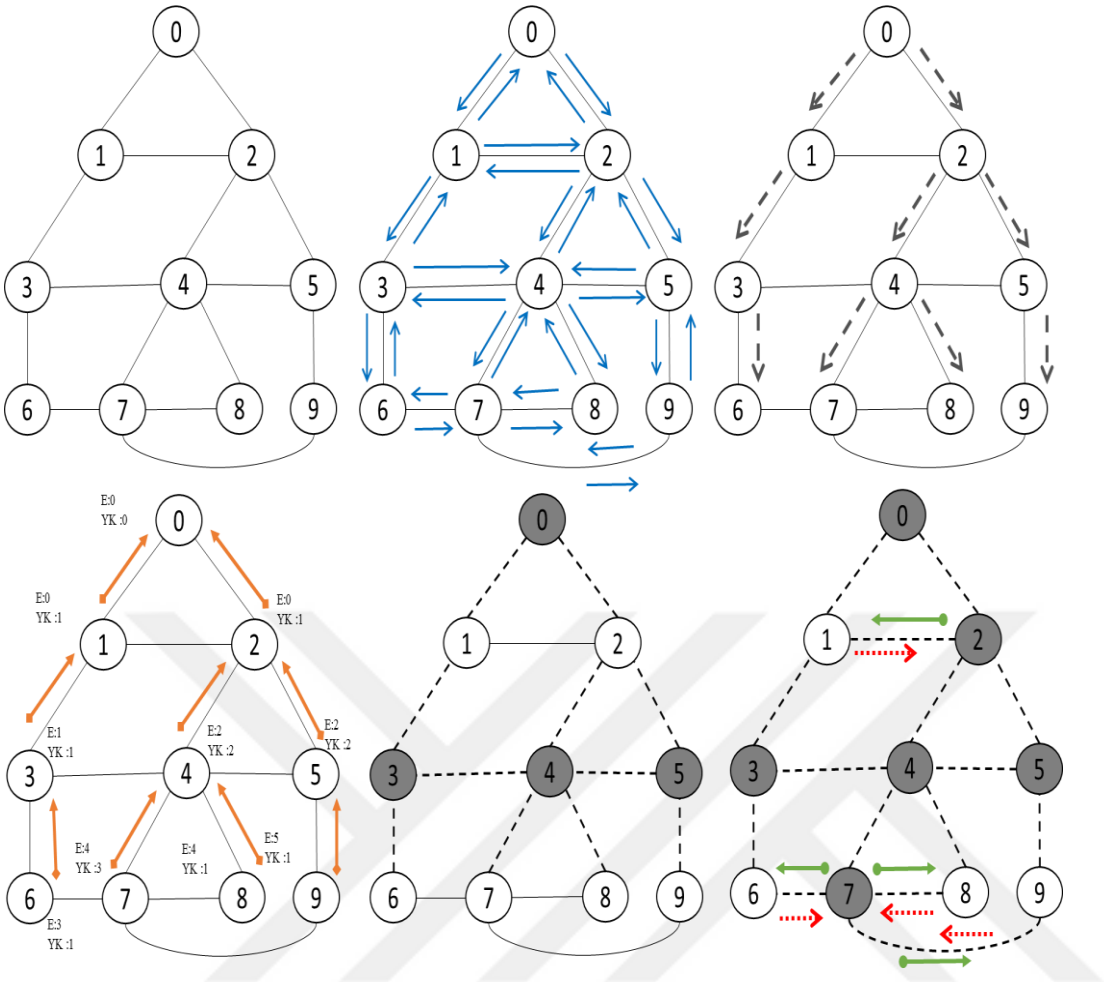
İlk olarak tüm tepeler birbirlerine *MERHABA* mesajı atarak komşuluk listelerini oluştururlar. İlk raunt bitiminde tüm tepeler komşularını biliyor olacaktır. Daha sonra ikinci rauntta kök tepe ilk *İLERİ* mesajını göndererek EİAA'nın oluşumunu başlatmış olur. Kök tepe tarafından gönderilen *İLERİ* mesajını bir tepe aldığında ebeveyn olarak kök tepeyi işaret eder ve seviyelerini günceller. *İLERİ* mesajını alan bir tepe ilerleyen rauntta ileri mesajı gönderir. Telsiz duyarga ağlarında mesajlar tüm ağa gönderildiği için tepeler *İLERİ* mesajı içerisine ebeveyn bilgisini de dahil ederek, ebeveyn tepelerin çocuklar listesini oluşturmaları sağlanır. Eğer bir tepe tüm komşularını sınıflandırdı ve tüm çocuklarından *GERİ* mesajı aldı ise ebeveyn tepeye *GERİ* mesajı atabilir. İlk *GERİ* mesajı atacak olan tepeler ise yaprak tepelerdir. Çünkü yaprak tepeler çocukları olmadığı için yapraklara kadar inen *İLERİ* mesajını aldıktan sonra *GERİ* mesajı atacak olan ilk tepelerdir. *GERİ* mesajı kök tepeye ulaştıktan sonra algoritmalarımızın kullanacağı Enine İlk Arama ağacı oluşmuş olmaktadır.

Çift Öncelikli Tepe Örtüsü Algoritması

ilklendirme:
$e_u \leftarrow \infty$, $Ç_u \leftarrow \emptyset$, $D_u \leftarrow \emptyset$, $GÇ_u \leftarrow \emptyset$ *ileri_gönderildi* $\leftarrow$ *yanlış*, *ileri_alındı* $\leftarrow$ *yanlış*, *geri_gönderildi* $\leftarrow$ *yanlış*, *yan_kenar* $\leftarrow 0$, $seviye_u \leftarrow \infty$, *örtüldü* $\leftarrow$ *yanlış*, *komşu_yan_kenar* $\leftarrow \emptyset$
eğer $id_u = KÖK$ **ise**
  Gönder $İLERİ(0, 0, null)$ Γ_u
yeni bir raunt başladığında:
  eğer *ileri_alındı* = *doğru* & *ileri_gönderildi* = *yanlış* **ise**
    Gönder $İLERİ(id_u, seviye_u, e_u)$
    ileri_gönderildi $\leftarrow$ *doğru*
  değilse eğer $\Gamma_u = Ç_u \cup D_u \cup e_u$ and $GÇ_u = Ç_u$ ve *geri_gönderildi* = *yanlış* **ise**
    Gönder $GERİ(id_u, seviye)$ ebeveyne
    geri_gönderildi $\leftarrow$ *doğru*
  değilse eğer *geri_gönderildi* = *doğru* & *örtüldü* = *yanlış* **ise**
    eğer $seviye\%2 = 0$ **ise** *örtüldü* $\leftarrow$ *doğru*
    değilse eğer $yan\_kenar > max(i : \forall i \in komşu\_yan\_kenar)$ **ise**
      Gönder $KARAR(id_u, yan\_kenar, seviye)$ ve *örtüldü* $\leftarrow$ *doğru*
    değilse Gönder $KARARSIZ(id_u, yan\_kenar, seviye)$
bir tepe $İLERİ(gönderen, seviye, ebeveyn)$ **mesajı aldığında:**
  ileri_alındı $\leftarrow$ *doğru*
  eğer $e_u = \infty$ **ise** $e_u \leftarrow gönderen$ & $seviye_u \leftarrow seviye + 1$
  değilse eğer $ebeveyn = id_u$ **ise** $Ç_u \leftarrow Ç_u \cup gönderen$
  değilse $D_u \leftarrow D_u \cup gönderen$
bir tepe $GERİ(gönderen, ebeveyn, seviye)$ **mesajı aldığında:**
  eğer $ebevyn = id_u$ **ise** $GÇ_u \leftarrow GÇ_u \setminus gönderen$
  değilse eğer $seviye_u = seviye$ **ise** $yan\_kenar \leftarrow yan\_kenar + 1$
bir tepe $KARAR(gönderen, yan\_kenar, seviye)$ **mesajı aldığında:**
  $yan\_kenar \leftarrow yan\_kenar - 1$
  $komşu\_yan\_kenar[gönderen] \leftarrow 0$
bir tepe $KARARSIZ(gönderen, yan\_kenar, seviye)$ **mesajı aldığında:**
  $komşu\_yan\_kenar[gönderen] \leftarrow yan\_kenar$

Şekil 3.1 Çift öncelikli dağıtık tepe örtüsü algoritması.

Çift Öncelikli Tepe Örtüsü algoritmasını Bölüm 3'te verilen fikir ve algoritmasıyla ilgili çalışma örneği Şekil 3.1'te verilmiştir. Öncelikle tüm tepeler birbirlerine *MERHABA* mesajı gönderirler. Daha sonra 0 numaralı kök tepenin *İLERİ* mesajı ile birlikte bu ileri mesajını alan tüm tepeler *İLERİ* mesajı atmaya başlarlar. *İLERİ* mesajlarının hepsinin iletilmesi sonucunda EİAA oluşur. *İLERİ* mesajları yaprak tepelere kadar ulaştıktan sonra algoritmanın *GERİ* safhası başlar. Tüm çocuklarından geri mesajı alan ve tüm komşularını sınıflandıran tüm tepeler *GERİ* mesajı atar. *GERİ* mesajları ile birlikte tüm düğümler sahip oldukları yan



Şekil 3.2 Çift öncelikli algoritmaya ait örnek bir çalışma.

kenar sayılarını ve komşularının yan kenar sayılarını öğrenirler. *GERİ* mesajı 0 numaralı düğüme ulaştığında çift seviyelerde bulunan düğümler doğrudan örtülür. Tek seviyelerde bulunan düğümler ise yan kenar sayılarına bağlı açgözlü yaklaşımla örtülürler. Tek numaralı seviyede bulunan 2 ve 7 numaralı düğümler komşularına göre daha fazla yan kenar sayısına sahip oldukları için kendilerini tepe örtüsü kümesine dahil ederek komşularına *KARAR* mesajı gönderirler. *KARAR* mesajını alan 1, 6, 8 ve 9 numaralı tepeler yan kenar sayısnı bir azaltırlar. Ayrıca 1, 6, 8, 9 numaralı tepeler *KARARSIZ* mesajı atarak komşularına yan kenar sayılarını gönderirler ve bu sayede tüm tepeler komşularının yan kenar sayısını güncelleyerek bir sonraki raunt için hazırlık yapmış olurlar. KUD algoritması tek numaralı seviyelerde düğümleri tepe örtüsü kümesine dahil etmek için, aynı seviyede kendinden daha küçük numaralı bir komşuya sahip olma koşulunu aramaktadır. Bu sebepten dolayı, bu örnek için KUD algoritması 8 ve 9 numaralı tepeleri de tepe örtüsü kümesine dahil edeceği için önerdiğimiz Çift Öncelikli Tepe Örtüsü algoritması daha iyi sonuç vermiştir.

Yan Kenar Tepe Örtüsü Algoritması

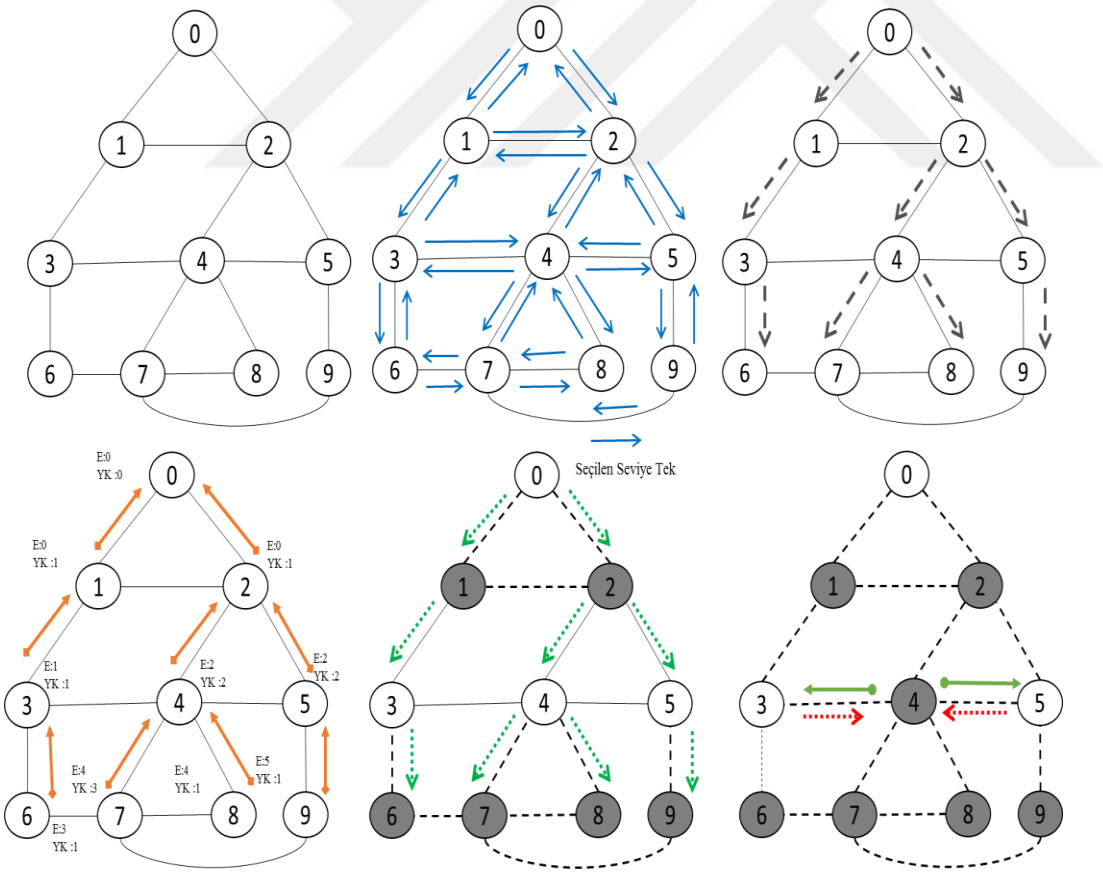
ilklendirme:
$e_u \leftarrow \infty$, $Ç_u \leftarrow \emptyset$, $D_u \leftarrow \emptyset$, $GÇ_u \leftarrow \emptyset$ *ileri_gönderildi* $\leftarrow$ *yanlış*, *ileri_alındı* $\leftarrow$ *yanlış*, *geri_gönderildi* $\leftarrow$ *yanlış*, *yan_kenar* $\leftarrow 0$, $seviye_u \leftarrow \infty$, *örtüldü* $\leftarrow$ *yanlış*, *komşu_yan_kenar* $\leftarrow \emptyset$, $seçim_u \leftarrow 0$, $çyk_u \leftarrow \emptyset$, $tyk_u \leftarrow \emptyset$, *seçim_alındı* $\leftarrow$ *yanlış*
eğer $id_u = KÖK$ **ise**
    Gönder $İLERİ(0, 0, null)$ Γ_u
yeni bir raunt başladığında:
    eğer *ileri_alındı* = *doğru* & *ileri_gönderildi* = *yanlış* **ise**
        Gönder $İLERİ(id_u, seviye_u, e_u)$
        ileri_gönderildi $\leftarrow$ *doğru*
    değilse eğer $\Gamma_u = Ç_u \cup D_u \cup e_u$ and $GÇ_u = Ç_u$ ve *geri_gönderildi* = *yanlış* **ise**
        Gönder $GERİ(id_u, seviye, çyk_u, tyk_u)$ ebeveyne
        eğer $id_u == KÖK$ **ise** $seçim \leftarrow max\{çyk_u, tyk_u\}$
            Gönder $SEÇİM(seçim)$
        geri_gönderildi $\leftarrow$ *doğru*
    değilse eğer *seçim_alındı* $\leftarrow$ *doğru* **ise**
        Gönder $SEÇİM(seçim)$
    değilse eğer *geri_gönderildi* = *doğru* & *örtüldü* = *yanlış* **ise**
        eğer $seviye\%seçim_u = 0$ **ise** *örtüldü* $\leftarrow$ *doğru*
        değilse eğer $yan\_kenar > max(i : \forall i \in komşu\_yan\_kenar)$ **ise**
            Gönder $KARAR(id_u, yan\_kenar, seviye)$ ve *örtüldü* $\leftarrow$ *doğru*
        değilse Gönder $KARARSIZ(id_u, yan\_kenar, seviye)$
bir tepe $İLERİ(gönderen, seviye, ebeveyn)$ **mesajı aldığında:**
    ileri_alındı $\leftarrow$ *doğru*
    eğer $e_u = \infty$ **ise** $e_u \leftarrow gönderen$ & $seviye_u \leftarrow seviye + 1$
    değilse eğer $ebeveyn = id_u$ **ise** $Ç_u \leftarrow Ç_u \cup gönderen$
    değilse $D_u \leftarrow D_u \cup gönderen$
bir tepe $GERİ(gönderen, ebeveyn, seviye, çyk, tyk)$ **mesajı aldığında:**
    eğer $ebevyn = id_u$ **ise** $GÇ_u \leftarrow GÇ_u \setminus gönderen$
    değilse eğer $seviye_u = seviye$ **ise** $yan\_kenar \leftarrow yan\_kenar + 1$
    eğer $seviye_u = seviye$ **ise** $çyk_u \leftarrow çyk + yan\_kenar$ ve $tyk_u \leftarrow tyk$
    değilse$tyk_u \leftarrow tyk + yan\_kenar$ ve $çyk_u \leftarrow çyk$
bir tepe $SEÇİM(seçim)$ **mesajı aldığında:**
    seçim_alındı $\leftarrow$ *doğru* ve $seçim_u \leftarrow seçim$
bir tepe $KARAR(gönderen, yan\_kenar, seviye)$ **mesajı aldığında:**
    $yan\_kenar \leftarrow yan\_kenar - 1$
    $komşu\_yan\_kenar[gönderen] \leftarrow 0$
bir tepe $KARARSIZ(gönderen, yan\_kenar, seviye)$ **mesajı aldığında:**
    $komşu\_yan\_kenar[gönderen] \leftarrow yan\_kenar$

2

Şekil 3.3 Yan kenar dağıtık tepe örtüsü algoritması.

Şekil 3.4'te Yan Kenar Algoritmasının aynı çizge üzerinde çalışması gösterilmektedir. EİAA'nın oluşturulması adımına kadar aynı olan Yan Kenar Algoritması, GERİ mesajlarının içeriği açısından Çift Öncelikli Algoritmadan farklılık arz eder. Şöyle ki Yan Kenar Algoritmasında *GERİ* mesajlarının içerisine tepeler yan kenar sayılarını da eklerler bu bilgi her bir seviye yukarı çıktığında toplanarak kök tepeye ulaşmış olur.

Nihayetinde *GERİ* mesajı 0 numaralı kök tepeye ulaştığında kök tepede çift ve tek seviyelerdeki toplam yan kenar sayıları mevcuttur. Bu bilgiler doğrultusunda yan kenar sayısı fazla olan seviye kök tepe tarafından seçilerek ağaç üzerinde tüm tepelere bildirilir. *SEÇİM* mesajını alan bir tepe seçilmiş olan bir seviyede ise doğrudan tepe örtüsü kümesine dahil olur. Bundan sonraki adımlar klasik açgözlü yaklaşımla çözümlenir. 1, 2, 6, 7, 8, 9 numaralı tepeler doğrudan tepe örtüsü kümesine dahil oldukları için sonlanır. Çift seviyeli tepeler olan 0, 3, 4 ve 5 numaralı tepeler arasında Açgözlü Algoritma çalıştırılır. 0 numaralı tepe için bu işlemi yapmaya gerek yoktur. Tek seviyeler doğrudan kapsandığı için 0 numaralı tepeye ait kenarlar bir alt seviyedeki tepeler tarafından kapsanmıştır. 3, 4, 5 numaralı tepelerden yan kenar sayısı 2 olan 4 numaralı tepe aynı seviyedeki komşularına *KARAR* mesajı atar ve tepe örtüsü kümesine dahil olur. *KARAR* mesajını alan 3 ve 5 numaralı tepeler de yan kenar sayılarını 1 azaltarak 0 yaparlar ve algoritma bu tepeler için de sonlanmış olur. Ayrıca *KARAR* komşuları arasında daha az yan kenara sahip olan tepeler de komşularına *KARARSIZ* mesajı atarlar.



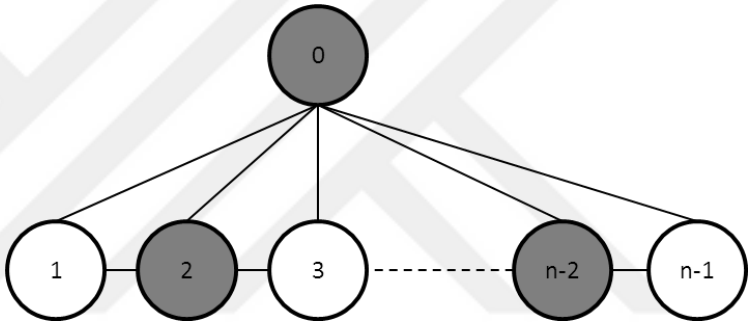
Şekil 3.4 Yan kenar algoritmasına ait çalışma örneği.

4. TEORİK ANALİZ

Teorik analiz yapılırken *MERHABA* mesajlarının önceden paylaşıldığı varsayılmıştır.

Kuram 1. Çift Öncelikli Algoritmanın zaman karmaşıklığı *O(n)*dir.

İspat. En kötü durumda kök tepeye bağlı *n*-1 adet tepe olur ve bu tepeler birbirileri ile zincir oluşturur. EİAA'nın oluşması *O(D)* zaman karmaşıklığına sahiptir. *n*-1 adet tepe içeren bir zincirde Açgözlü tepe örtmesi yapmanın zaman karmaşıklığı O(*n*)dir. Bu durumda Çift Öncelikli Tepe Örtme Algoritmasının zaman karmaşıklığı *O(D)+O(n)=O(n)*dir. Şekil 4.1'de bu duruma örnek teşkil eden bir çizge görülmektedir. □



Şekil 4.1 Çift öncelikli algoritma için en kötü durumun sağlandığı bir çizge.

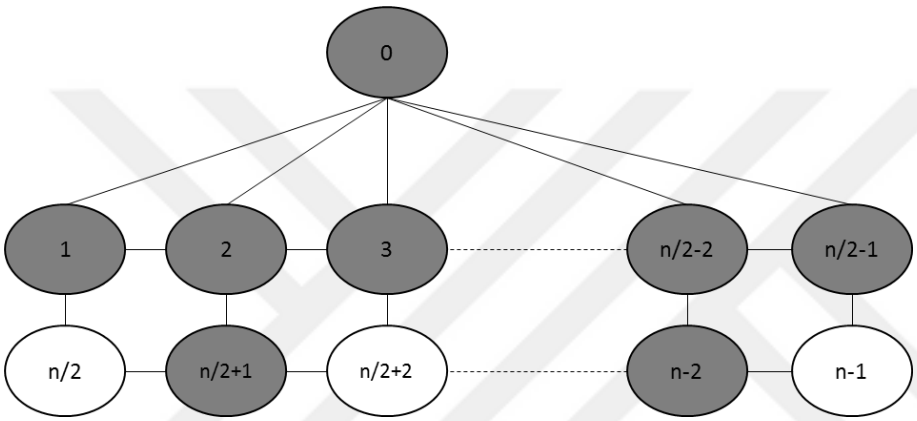
Kuram 2. Çift Öncelikli Algoritmanın bit karmaşıklığı $O(n^2(\log_2(n)))$dir.

İspat. Tüm tepeler kesinlikle birer adet *İLERİ* ve *GERİ* mesajları atacaklardır. Yalnızca kök tepe *GERİ* mesajı atmayacaktır. Bu durumda EİAA'nın oluşması için 2*n*-1 adet mesaj gönderilmesi gerekmektedir. Ağaç oluşturulduktan sonra en kötü durumun Şekildeki gibi olduğunu düşünürsek her bir rauntta 1 adet *KARAR* ve *n*-2 adet *KARARSIZ* mesajı atılacaktır. Toplamda *O(n)* raunt sürecek olan bu algoritma çalışma süreci boyunca *O(n*n-1+2n+1)* adet mesaj atılacaktır. Bu ifadeyi sadeleştirecek olursak algoritma boyunca gönderilen mesaj sayısı asimptotik olarak $O(n^2)$dir. Her bir mesaj $\log_2(n)$ ile ifade edilebileceği için toplam bit karmaşıklığı $O(n^2\log_2(n))$ olarak bulunur. □

Kuram 3. Çift Öncelikli Algoritmanın uzay karmaşıklığı $O(\Delta\log_2(n))$dir.

İspat. Tüm tepelerde komşular, çocuklar, diğer bilgilerini tutmak için kümeler tanımlanmıştır. Ayrıca ebeveyn, yan kenar, derece bilgisini tutmak için bir tam sayı değişkenler yer almaktadır. Bu bağlamda komşuları, çocukları ve diğerlerini tutmak için $\Delta log_2(n)$lik toplam 3 adet kümeye ihtiyaç vardır. Öte yandan ebeveyn, derece, yan kenar bilgilerini de tutmak için $log_2(n)$lik 3 adet değişken gerekmektedir. Tüm bu yer gereksinimleri toplanacak olursa, uzay karmaşıklığı $O(3\Delta log_2(n)+ 3log_2(n))$ ile ifade edilir. Sadeleştirecek olursak genel bir ifadeyle uzay karmaşıklığı $O(\Delta log_2(n))$dir. □

Kuram 4. Yan Kenar Algoritmasının zaman karmaşıklığı $O(n)$dir.



Şekil 4.2 Yan kenar algoritması için en kötü durumun sağlandığı bir çizge.

İspat. EİAA'nın oluşması $O(D)$ zaman karmaşıklığına sahiptir. Bir zincir üzerinde Açgözlü tepe örtüsü yapma işlemi $O(n)$ zaman karmaşıklığına sahiptir. Bu durumda Yan Kenar Algoritmasının zaman karmaşıklığı $O(D)+O(n)=O(n)$dir. Şekil 4.2'de en kötü durumu gösteren bir çizge görülmektedir. □

Kuram 5. Yan Kenar Algoritmasının bit karmaşıklığı $O(n^2(log_2(n)))$dir.

İspat. Tüm tepeler kesinlikle birer adet *İLERİ* ve *GERİ* mesajları atacaklardır. Yalnızca kök tepe *GERİ* mesajı atmayacaktır. Yaprak tepeler haricinde kalan tüm tepelerde *SEÇİM* mesajı atacaklardır. Bu durumda EİAA'nın oluşması $O(n)$ mesaj karmaşıklığına sahiptir. Ağaç oluşturulduktan sonra en kötü durumun Şekil 4.2'deki gibi olduğunu düşünürsek her bir rauntta 1 adet *KARAR* ve n-2 adet *KARARSIZ* mesajı atılacaktır. Toplamda $O(n)$ raunt sürecek olan bu algoritma çalışma süreci boyunca $O(n*n-1+3n)$ adet mesaj atılacaktır. Bu ifadeyi sadeleştirecek olursak algoritma boyunca gönderilen mesaj sayısı asimptotik

olarak $O(n^2)$dir. Her bir mesaj $log_2(n)$ ile ifade edilebileceği için toplam bit karmaşıklığı $O(n^2 log_2(n))$ olarak bulunur. □

Kuram 6. Yan Kenar Algoritmasının uzay karmaşıklığı $O(\Delta log_2(n))$dir.

İspat. Tüm tepelerde komşular, çocuklar, diğer bilgilerini tutmak için kümeler tanımlanmıştır. Ayrıca ebeveyn, yan kenar, derece bilgisini tutmak için bir tam sayı değişkenler yer almaktadır. Bu bağlamda komşuları, çocukları ve diğerlerini tutmak için $\Delta log_2(n)$lik toplam 3 adet kümeye ihtiyaç vardır. Öte yandan ebeveyn, derece, yan kenar bilgilerini de tutmak için $log_2(n)$lik 3 adet değişken gerekmektedir. Tüm bu yer gereksinimleri toplanacak olursa, uzay karmaşıklığı $O(3\Delta log_2(n)+ 3log_2(n))$ ile ifade edilir. Sadeleştirecek olursak genel bir ifadeyle uzay karmaşıklığı $O(\Delta log_2(n))$dir. □

Bu bölümde teorik analizi yaparken kabul edilen varsayımları Açgözlü ve KUD algoritması için de kabul edersek, KUD algoritmasının dışında tüm algoritmaların zaman, bit ve yer karmaşıklığı eşit olacaktır. KUD algoritmasının zaman karmaşıklığı $O(D)$ bit karmaşıklığı $O(n^3 log_2(n))$ olarak ifade edilmiştir. (Kavalcı et al., 2014)

5. BENZETİMLERİN DEĞERLENDİRİLMESİ

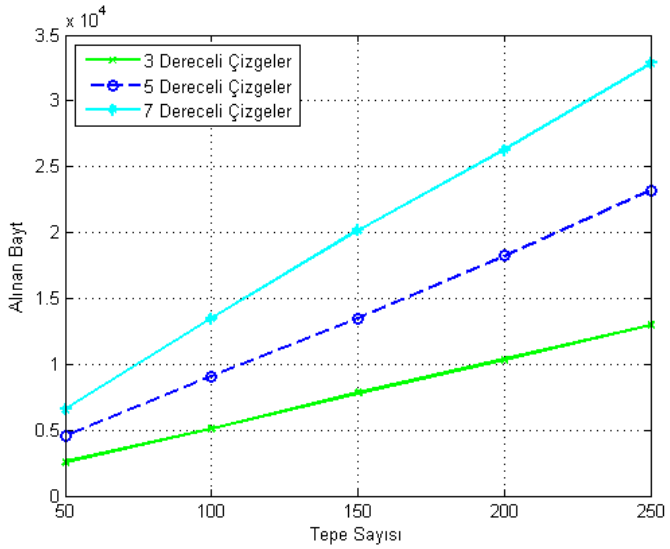
EİAA ve Açgözlü Tepe Örtüsü Algoritması, KUD, Çift Öncelikle Tepe Örtüsü ve Yan Kenar Tepe Örtüsü algoritmaları rastgele üretilen birim disk çizge modellerinde 3, 5 ve 7 dereceli çizgelerde 10 farklı deneme alınarak 50 tepeden 250 tepeye kadar 50'şer artan çizgeler üzerinde test edilmiştir. Test sonucunda elde edilen veriler, gönderilen mesaj sayısı, alınan mesaj sayısı, gönderilen bayt sayısı, alınan bayt sayısı, tepe örtüsü kümesinin eleman sayısı, zaman ve enerji sarfiyatıdır. Elde edilen sonuçlar 10 denemenin ortalaması alınarak bulunmuştur. Çizelge 5.1'de benzetim ortamına ait parametreler görülmektedir.

Çizelge 5.1 Benzetime ait parametreler.

Tepe Dağılımı	Rastgele
Kök tepe pozisyonu	Çizge içerisinde rastgele bir pozisyon
Tepe sayıları	50-250
Ortam Erişim Katmanı	Zaman Bölmeli Ortam Erişimi (TDMA)
İletim gücü	3 dBm
İletim mesafesi	50 m
Tepelerin dereceleri	3, 5 ve 7

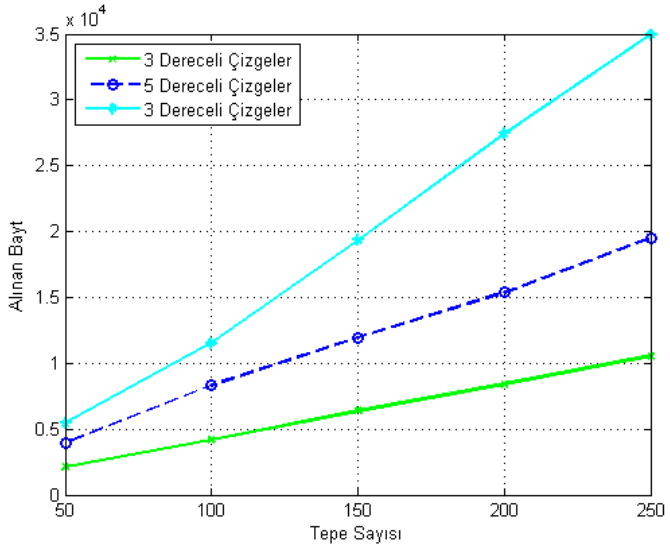
Açgözlü Algoritma çalışmadan önce, senkron EİAA oluşturulmaktadır. Bu nedenle ölçümlerde iki farklı kodun çıktıları toplanarak sonuç elde edilmiştir. KUD, Çift Öncelikli ve Yan Kenar Algoritmalarında ise öncelikle bir EİAA oluşturup bütünleşik olarak tepe örtüsü işlemi yapılmaktadır. EİAA senkron olarak oluşturulmuştur. Algoritma ilk adımda her tepenin komşularına *MERHABA* mesajı atmasıyla başlamaktadır. Bu adımdan sonra komşularını öğrenen tepeler senkron EİAA oluştururlar.

Tüm kodlar aynı şartlar altında ve aynı çizgeler üzerinde çalıştırılmıştır. Her türden 10 farklı çizge üzerinde alınan ölçümler sonucunda ortalamalar alınarak karşılaştırmalar yapılmıştır. Bu bölümde alınan bayt, gönderilen bayt, zaman, enerji ve tepe örtüsü kümesi bakımından algoritmaların sonuçları karşılaştırılmıştır.



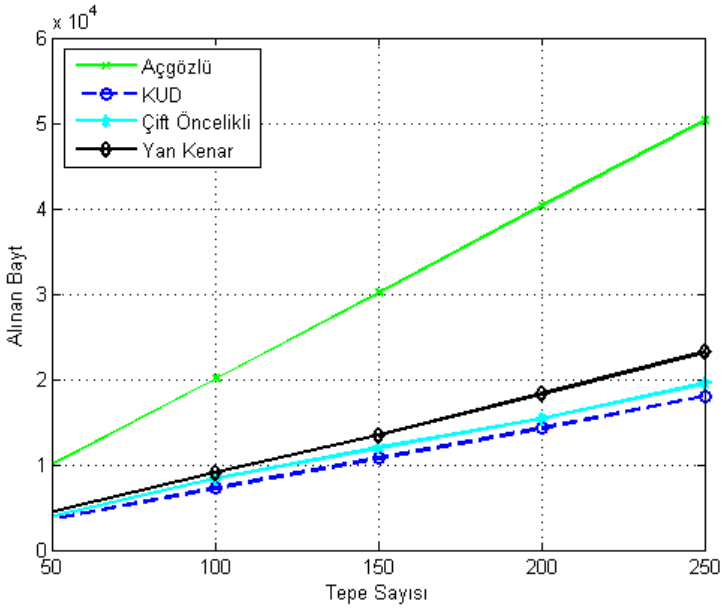
Şekil 5.1 Yan kenar algoritmasında artan tepe sayısına ve dereceye bağlı alınan bayt.

Şekil 5.1'de Yan Kenar Algoritmasının dereceye bağlı alınan bayt değerleri görülmektedir. Artan düğüm sayısı ve dereceyle birlikte alınan baytın da doğrusal olarak arttığı görülmektedir. Çizgenin ortalama derece sayısı arttıkça gönderilen bir mesajı alacak olan tepe sayısı artacağı için alınan bayt sayısında artış gözlenmektedir. Aynı şekilde çizgedeki tepe sayısı arttıkça gönderilen mesajlar artacağı için bu vasıtayla alınan mesaj sayısında da artış olacaktır. Böylece alınan bayt sayında tepe sayısına bağlı olarak artış görülmektedir.



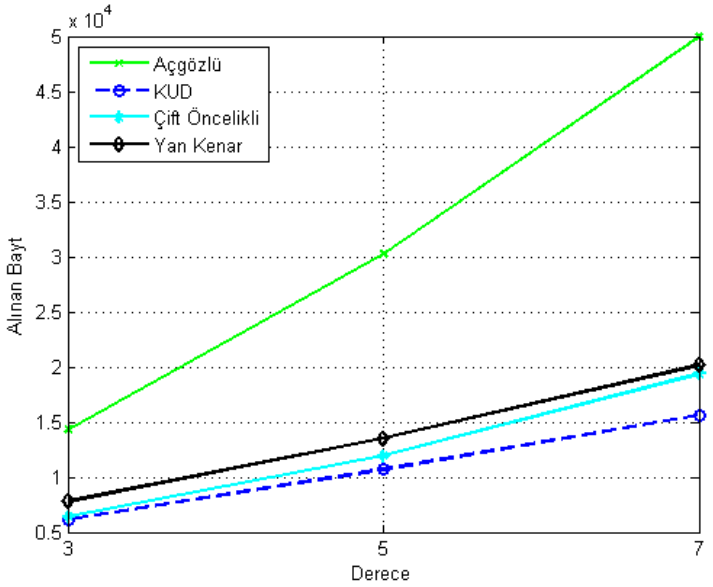
Şekil 5.2 Çift öncelikli algoritmada artan tepe sayısına ve dereceye bağlı alınan bayt.

Şekil 5.2'de Çift Öncelikli Algoritmada alınan bayt değerleri görülmektedir. Yan Kenar Algoritmasında olduğu gibi artan dereceye ve tepe sayısına bağlı olarak alınan bayt değerlerinde artış görülmektedir. Yan Kenar Algoritmasında Çift Öncelikli Algoritmaya nazaran artan yoğunluk ve tepe sayısına bağlı olarak alınan bayt sayısında bir miktar azalma görülmektedir. Bunun temel sebebi, Yan Kenar Algoritması doğrudan kapsanacak olan seviyeleri yan kenar sayılarına göre belirlediği için bazı çizgelerde tek numaralı seviyelerde daha fazla yan kenar olduğundan ve doğrudan bu seviyeleri kapsayarak kalan seviyelerdeki mesaj trafiğini azalttığı için Çift Öncelikli Algoritmalardan daha iyi sonuç vermektedir.



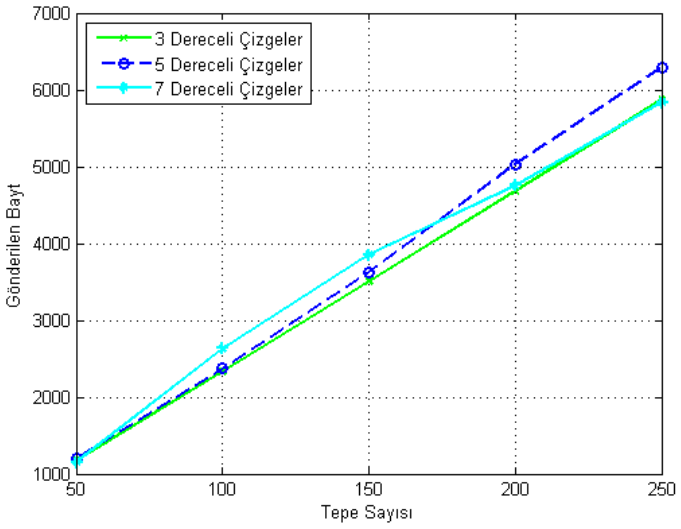
Şekil 5.3 5 dereceli çizgelerde algoritmalara ait alınan bayt değerleri.

EİAA'nın oluşturulup bu ağaçtan bağımsız olarak Açgözlü tepe örtüsü algoritması çalıştırılan çizgelerde derece ve tepe sayısına bağlı olarak alınan bayt değerleri görülmektedir. Şekil 5.3'te görüldüğü gibi Açgözlü Algoritmanın alınan bayt değerleri nerdeyse önerdiğimiz iki algoritmanın 2 katına kadar fazla alınan bayt değeri bulunmaktadır. Önerilen algoritmalarda 7 derecelik 250 tepelik çizgelerde elde edilen bayt değeri, Açgözlü Algoritmada 3 derecelik 150 tepeli çizgelerde alınamamaktadır.



Şekil 5.4 150 tepeli çizgelerde dereceye bağlı olarak alınan bayt.

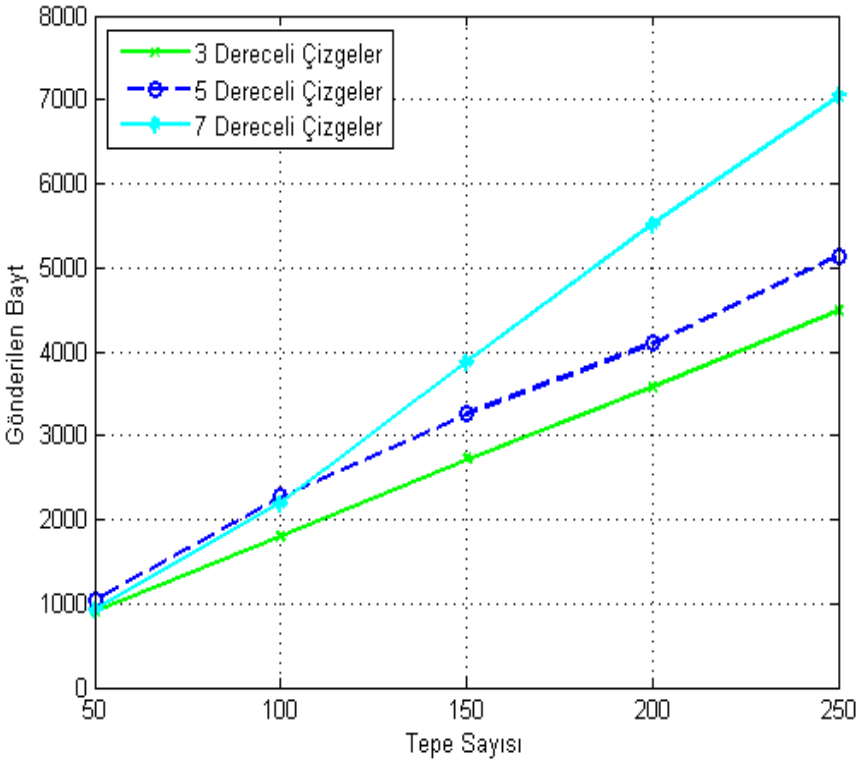
150 tepe bulunan çizgelerde artan dereceye göre alınan bayt değerleri Şekil 5.4'teki grafikte görülmektedir. KUD Algoritması, Çift Öncelikli Algoritma ve Yan Kenar Algoritması birbirlerine yakın sonuçlar verirken Açgözlü Algoritma daha öncede belirtildiği gibi 3 kata kadar fazla alınan bayt değerine sahiptir.



Şekil 5.5 Yan kenar algoritmasında artan derece ve tepe sayısına bağlı gönderilen bayt.

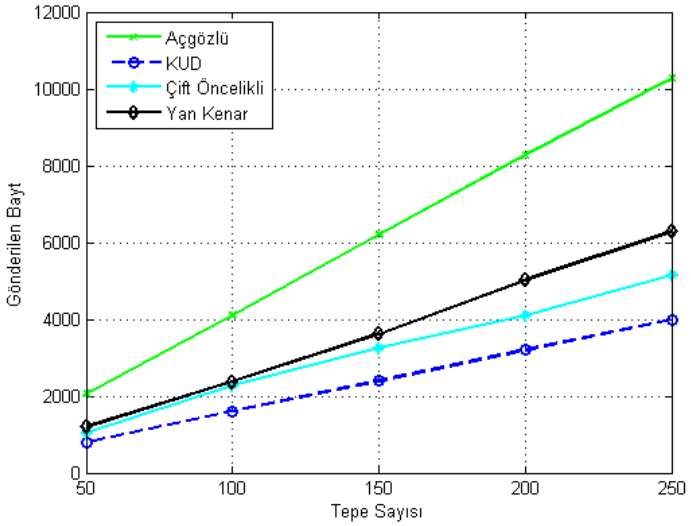
Yan Kenar Algoritmasında artan derece ve tepe sayısına bağlı gönderilen bayt değerlerini gösteren grafik Şekil 5.4'te verilmiştir. Gönderilen bayt sayısında artışta etkili olan faktörün tepe sayısı olduğu grafikten görülebilmektedir.

Gönderilen bayt değerinde derece sayısının alınan bayt değerindeki kadar etkili olmadığı görülmektedir. Yan Kenar Algoritması *MERHABA* mesajı ile birlikte bir tepenin en az 3 mesaj atacağını garanti eder bu mesajlar, *MERHABA*, *İLERİ*, *GERİ* mesajlarıdır. Bunun dışında gönderilen mesajlar düğümlerin konumlarına ve seviyelerine bağlı olarak değişmektedir. Bu sebeple tüm derecelerde yakın sayılarda mesaj gönderilmektedir. Tepe sayısı arttığında 7 dereceli çizgelerde 3 dereceli çizgelerde elde edilen bayt sayısı elde edildiği görülmektedir. Bu durumun da temel sebebini önceden de bahsedildiği gibi, tek seviyelerin doğrudan kapsanması olarak açıklayabiliriz.



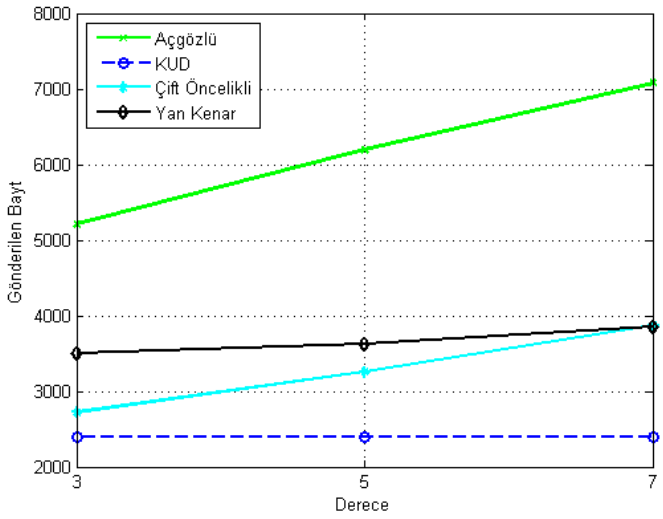
Şekil 5.6 Çift öncelikli algoritmada artan derece ve tepe sayısına bağlı gönderilen bayt.

Şekil 5.6'da Çift Öncelikli Algoritmada artan derece ve tepe sayısına bağlı olarak gönderilen bayt değerleri görülmektedir. Çift Öncelikli Algoritmada tepe sayısının gönderilen bayta olan etkisi fazlayken, derecenin şu an ki 3, 5 ve 7 derecelik çizgeler için tepe sayısına göre etkisi oldukça azdır. Ayrıca Yan Kenar Algoritmasında mesaj paketleri daha çok parametre içermesine rağmen birbirine yakın sonuçlar vermişlerdir. Hatta 7 dereceli çizgelerde Yan Kenar Algoritmasının daha iyi sonuçlar verdiği görülmektedir.



Şekil 5.7 5 dereceli çizgelerde tepe sayısına bağlı gönderilen bayt.

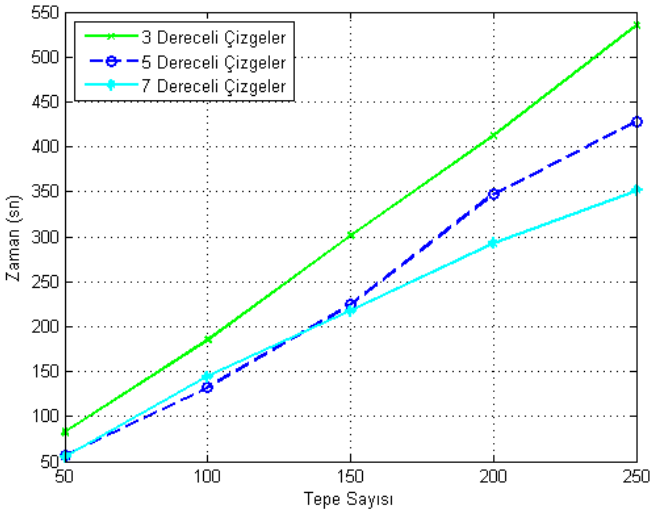
Şekil 5.7'de algoritmaların gönderilen bayt değerleri açısından karşılaştırılmaları görülmektedir. Gönderilen bayt sayısında etken faktörün tepe sayısı olduğu görülmektedir. Önerdiğimiz iki algoritmaya göre Açgözlü Algoritmanın mesaj boyutu daha küçük olmasına rağmen daha kötü sonuçlar verdiği görülmektedir.



Şekil 5.8 150 tepeli çizgelerde dereceye bağlı olarak gönderilen bayt.

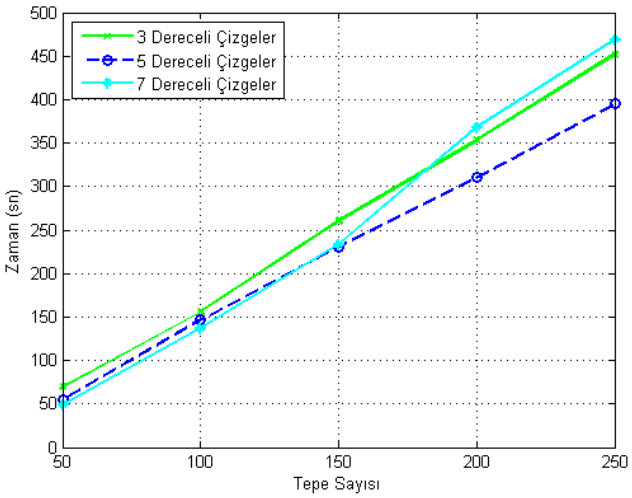
150 tepe bulunan çizgelerde dereceye bağlı olarak gönderilen bayt değerleri Şekil 5.8'de verilmiştir. Söylendiği gibi özellikle KUD Algoritması ve önerdiğimiz iki algoritmada derece sayısının gönderilen bayta fazlaca bir etkisi

bulunmamaktadır. Ayrıca gönderilen bayt değerlerinde KUD en iyi sonucu verirken, önerdiğimiz iki algoritma, Çift öncelikli ve Yan Kenar Algoritmaları birbirlerine yakın sonuçlar vermekle birlikte KUD algoritmasının biraz üzerinde sonuçlar elde edilmiştir. Açgözlü Algoritma ise gönderilen bayt değerlerinde de diğer algoritmalardan başarı olarak çok geride kalmıştır.



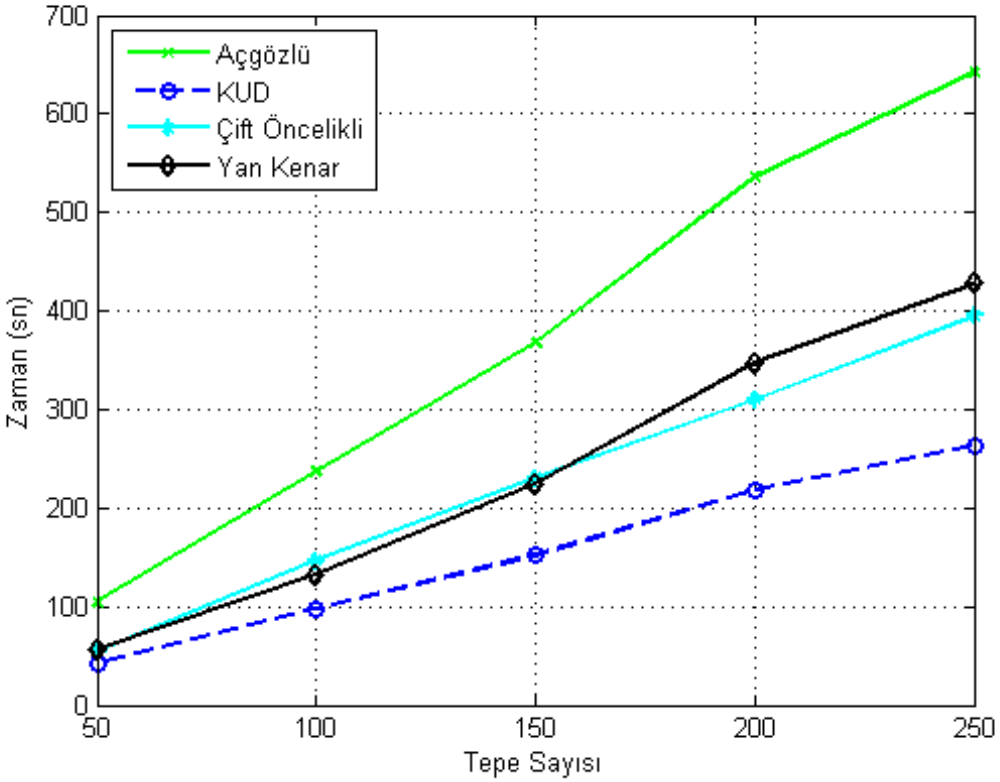
Şekil 5. 9 Yan kenar algoritmasında zaman ölçümleri.

Yan Kenar Algoritmasında zaman ölçümleri Şekil 5.9'da gösterilmiştir. Algoritmanın çalışma süresi tepe sayısına bağlı olarak doğrusal artış göstermiştir. Ayrıca 7 dereceli çizgelerde 3 ve 5 dereceli çizgelere göre daha iyi sonuçlar verdiği görülmektedir.



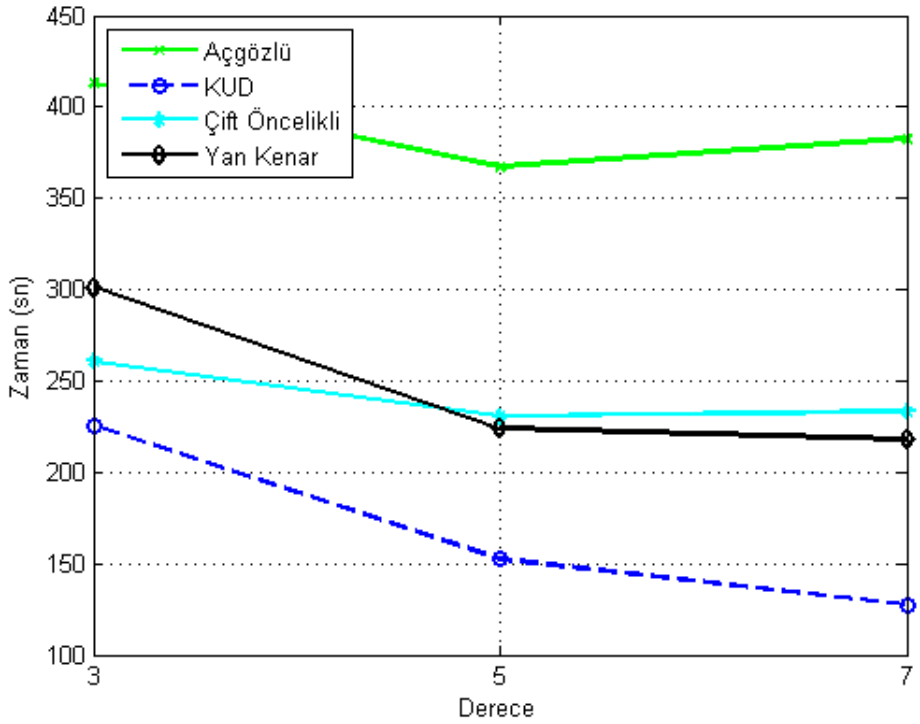
Şekil 5.10 Çift öncelikli algoritmasında zaman ölçümleri.

Şekil 5.10'da Çift Öncelikli Algoritmanın değişen tepe sayısı ve derece ortalamasına bağlı olarak çalışma zamanları görülmektedir. 7 dereceli çizgelerde en uzun çalışma süresi elde edilirken algoritma 5 dereceli çizgelerde daha çabuk sonuçlar vermektedir. Çift Öncelikli Algoritma çalışma süresi olarak Yan Kenar Algoritmasından daha iyi sonuçlar vermiştir. Bunun sebebi, Çift Öncelikli Algoritmada enini ilk arama ağacı oluşturulduktan sonra tepelerin kapsama işlemi hemen başlamaktadır ancak Yan Kenar Algoritmasında hangi seviyelerin kapsanacağı bilgisi öncelikle ağaç üzerinde tüm tepelere bildirildiği için burada bir gecikme yaşanmaktadır.



Şekil 5.11 5 dereceli çizgelerde artan tepe sayısına bağlı zaman ölçümleri.

Şekil 5.11'de algoritmaların çalışma süreleri verilmiştir. Çalışma süresinde etkili olan faktör tepe sayısıdır. 5 dereceli çizgelerde önerdiğimiz iki algoritma KUD Algoritmasıyla yakın sonuçlar vermiştir ve derece sayısı arttıkça önerdiğimiz algoritmaların sonuçları daha da iyiye gitmektedir.



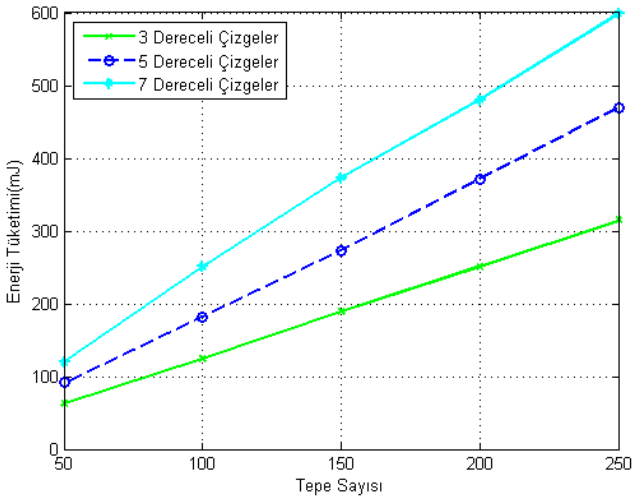
Şekil 5.12 150 tepeli çizgelerde dereceye bağlı olarak çalışma süreleri.

Şekil 5.12'de 150 tepeli 3, 5 ve 7 dereceli çizgelerde algoritmaların çalışma süreleri görülmektedir. Görüldüğü gibi en kısa sürede sonlanan algoritma KUD algoritması olurken, önerdiğimiz algoritmalar yakın sonuçlar vererek KUD algoritmasını takip etmektedir. Açgözlü Algoritma ise çalışma süresi olarak en kötü sonucu veren algoritma olmuştur. Ayrıca Şekil 5.12'den de görülebileceği gibi derece sayısı arttıkça algoritmaların çalışma sürelerinde bir azalama olmaktadır.

Uygulamalarda kullanılan Iris düğümler 3,3 volta sahiptir. (http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf, Erişim Tarihi: 25.05.2016) Iris düğümlerin veri gönderirken ve alırken ne kadar akım harcadığı dikkate alınarak sistemde tüketilen enerji

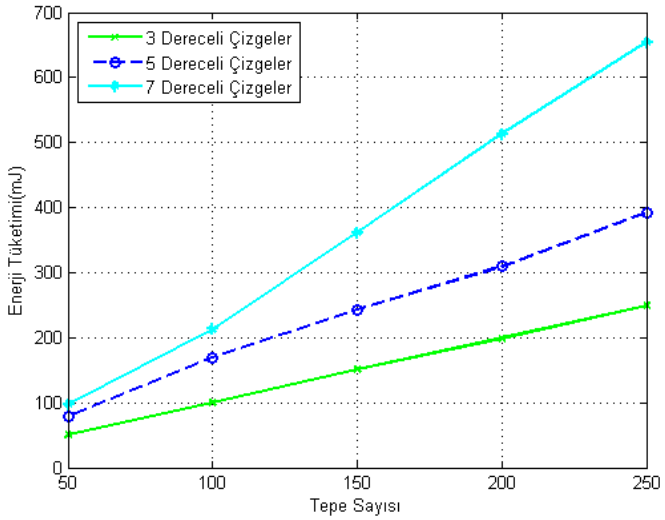
$$\frac{(Gönderilen\ Bayt\ Sayısı * 0{,}2) + (Alınan\ Bayt\ Sayısı * 0{,}13)}{30} * 3{,}3\ mJ$$

formülü ile hesaplanmıştır.



Şekil 5.13 Yan kenar algoritmasında enerji tüketimi.

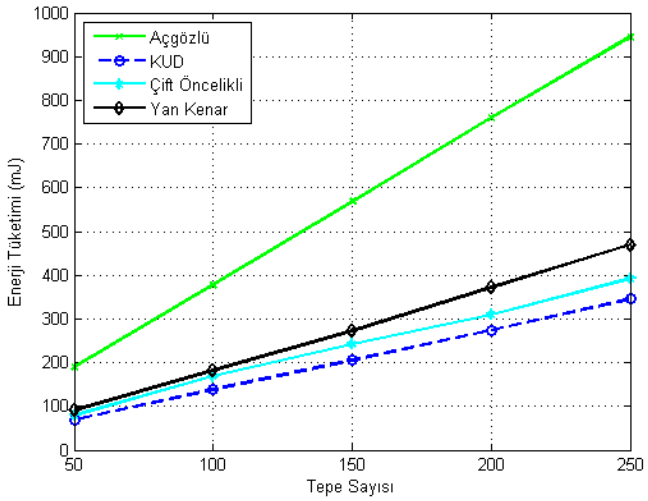
Şekil 5.13'te Yan Kenar Algoritmasının enerji tüketimi verilmiştir. Formülden de anlaşılacağı gibi enerji tüketiminde gönderilen baytın katkısı daha fazladır. Ancak alınan bayt sayısı gönderilen bayt sayısında daima fazla olduğu için enerji tüketiminde alınan baytlar daha baskındır. Bu vesile ile Şekil 1'de de bahsedildiği gibi dereceye bağlı olarak alınan bayt arttığı için enerji tüketiminde de tepe sayısı ve derece birlikte rol oynamaktadır.



Şekil 5.14 Çift öncelikli algoritmada enerji tüketimi.

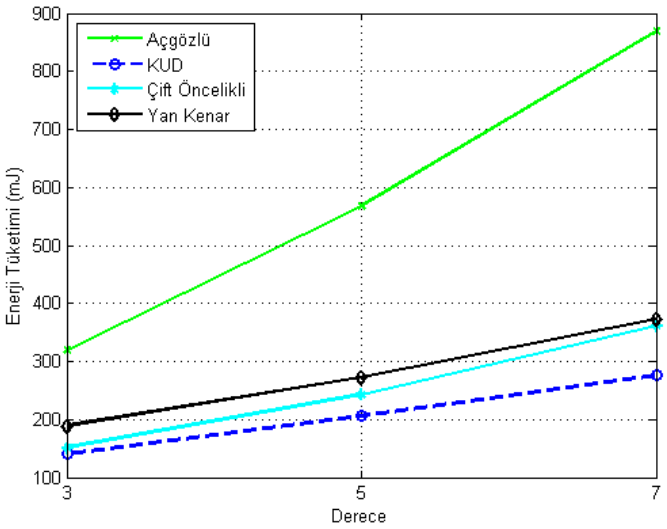
Çift Öncelikli Algoritmada enerji tüketimi Şekil 5.14'te verilmiştir. 3 ve 5 dereceli çizgelerde Yan Kenar Algoritmasında iyi sonuçlar veren Çift Öncelikli

Algoritma, 7 dereceli çizgelerde alınan ve gönderilen bayt değerlerinin Yan Kenar Algoritmasından fazla olması sebebiyle daha fazla enerji sarf etmektedir.



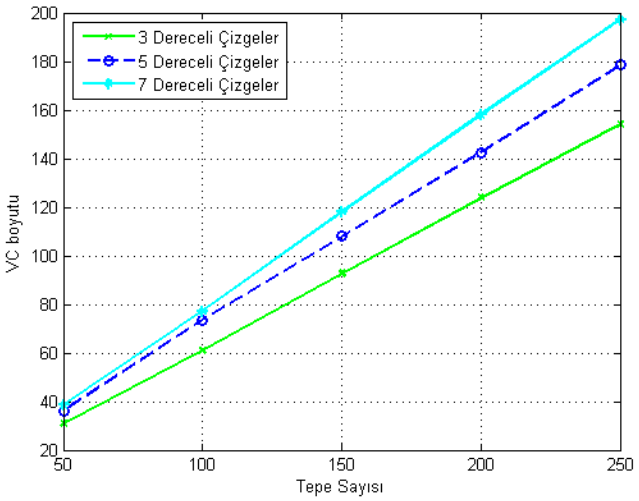
Şekil 5.15 5 dereceli çizgelerde algoritmaların enerji tüketimi.

Şekil 5.15'te görüldüğü üzere, Açgözlü Algoritmada da alınan bayt sayısına etki eden derece sayısı ve gönderilen bayt sayısına etki eden tepe sayısı sebebiyle, artan bayt değerleri sebebiyle enerji tüketimine her iki faktör de etki etmektedir. Açgözlü Algoritma önerilen iki algoritmadan yer yer 3 kat fazla enerji ile çizgelerde tepe örtüsü işlemi yapmaktadır.



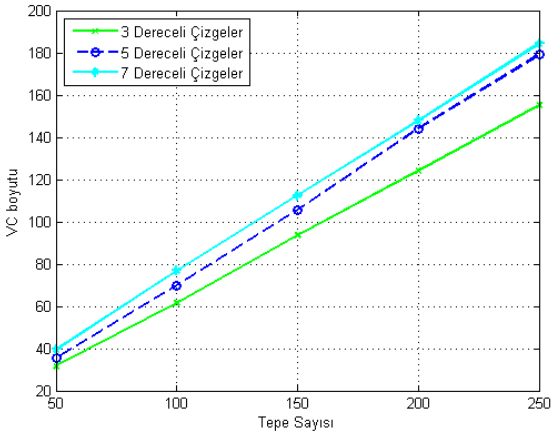
Şekil 5.16 150 tepeli çizgelerde dereceye bağlı olarak enerji tüketimi.

Şekil 5.16'da 150 tepeli çizgelerde dereceye bağlı enerji tüketim grafiği verilmiştir. KUD algoritması diğer ölçümlerde olduğu gibi bu ölçümde de tüm algoritmalara göre daha iyi sonuç vermiştir. Önerdiğimiz iki algoritma Çift öncelikli ve Yan Kenar Algoritmaları KUD'a yakın sonuçlar vermiştir. Açgözlü Algoritma ise en kötü sonucu veren algoritma olmuştur. Ayrıca derece artışı KUD, Çift öncelikli ve Yan Kenar Algoritmalarında, Açgözlü Algoritmaya gösterdiği kadar büyük etki göstermemiştir.



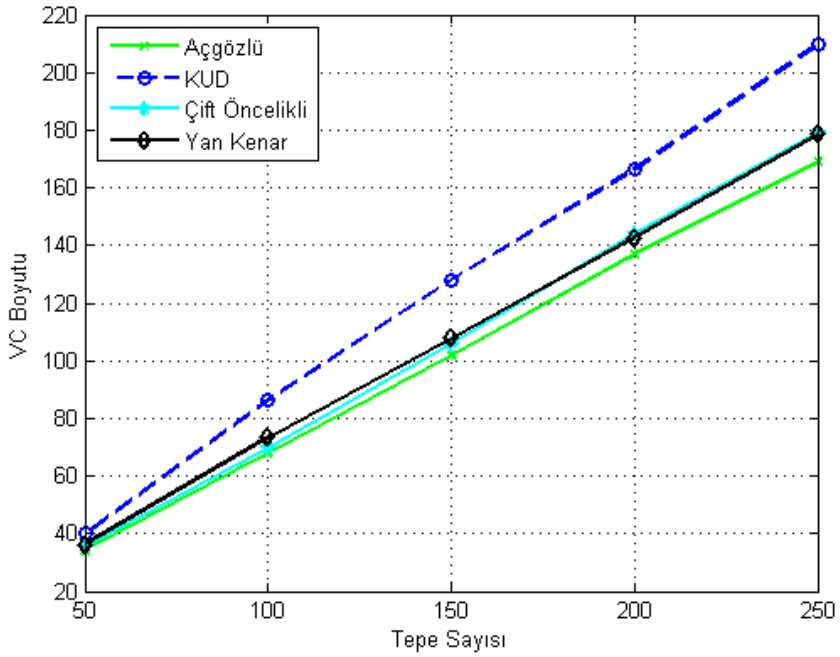
Şekil 5.17 Yan kenar algoritmasında tepe örtüsü kümesi.

Yan Kenar Algoritmasında Tepe Örtüsü Kümesinin eleman sayısı Şekil 5.17'deki grafikte verilmiştir. 250 tepeli 7 dereceli bir çizgede ortalama 200 elemanlık tepe örtüsü kümesi bulan algoritmada, derece sayısı arttıkça aynı tepe sayısına ait çizgelerde tepe örtüsü kümesinin de artığı görülmektedir.



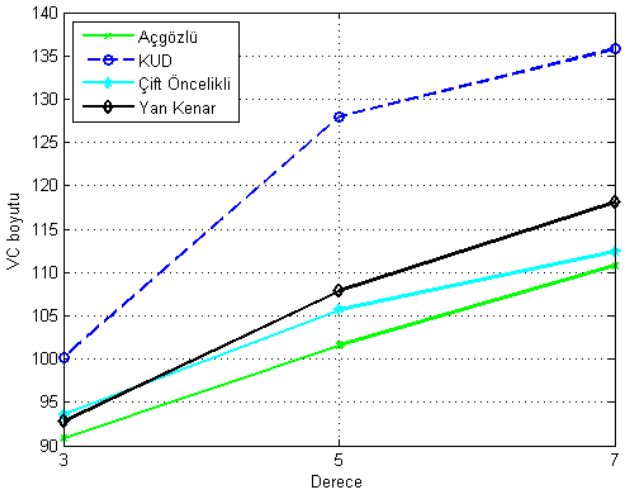
Şekil 5.18 Çift öncelikli algoritmada tepe örtüsü kümesi.

Şekil 5.18'de Çift Öncelikli Algoritmada tepe örtüsü kümesinin elaman sayıları görülmektedir. Dereceye bağlı olarak kümenin eleman sayısında artış görülmektedir. Çift Öncelikli Algoritmanın Yan Kenar Algoritmasına yakın sonuçlar verdiğini Şekil 17 ve Şekil 18'i karşılaştırarak görebilmekteyiz. Ancak Yan Kenar Algoritması tek numaralı seviyeleri kapsadığı özellikle 200 üzeri tepe sayısına sahip 7 derecelik çizgelerde daha fazla tepe örttüğü görülmektedir. Bunun sebebi ise yan kenar sayısı fazla olan tek numaralı seviyeleri kapsayarak fazladan tepe örtmesi yapmasıdır. Mesaj ve enerjiden tasarruf ederken bir yandan Örtülen Tepe sayısında artış elde edildiği için bu algoritmalar arasında seçim yapılırken çizgelerin tepe sayısı ve derecesi iyi ele alınmalıdır.



Şekil 5.19 5 dereceli çizgeler için tepe örtüsü kümesi.

Şekil 5.19'da tüm algoritmalar için tepe örtüsü kümesi grafiği verilmiştir. Açgözlü Algoritma Çift Öncelikli Algoritmadan çok az iyi sonuçlar vermesine rağmen, özellikle gönderilen bayt ve enerji tüketiminde daha kötü sonuçlar verdiği için böyle bir durumla karşı karşıya kalındığında Çift Öncelikli Algoritmanın seçilmesinde fayda vardır.



Şekil 5.20 150 tepeli çizgelerde dereceye bağlı olarak tepe örtüsü kümesi.

Şekil 5.20'de 150 tepeli çizgelerde dereceye bağlı olarak tepe örtüsü kümesi verilmiştir. KUD algoritması performans ve zaman olarak iyi sonuçlar vermiş olmasına rağmen tepe örtüsü kümesinde diğer algoritmalara göre daha kötü sonuçlar vermiştir. Çift öncelikli ve Açgözlü Algoritma birbirine yakın sonuçlar verirken, Yan Kenar Algoritması 3 dereceli çizgelerde Çift Öncelikli Algoritmadan daha iyi sonuç vermiştir.

Algoritmalar arasında seçim yaparken öncelikle hangi parametrede hassasiyet göstereceğimizi belirlememiz gerekmektedir. Büyük topolojilerde mümkün olduğu kadar az tepe örtüsü istiyorsak Çift Öncelikli Algoritmayı seçmemiz gerekirken, daha az mesaj ve enerjiyle tepeleri örtmek istiyorsak Yan Kenar Algoritmasına öncelik vermemiz gerekmektedir. Ayrıca düğüm başına kullanılan enerji miktarlarında en iyi sonucu KUD algoritması verirken en kötü sonucu Açgözlü algoritma vermiştir. Ancak KUD algoritmasının tepe örtüsü kümesinin boyutunun büyüklüğü de yadsınamaz bir gerçektir. Algoritmaların düğüm başına tükettikleri enerji, Açgözlü algoritma 3.8 mJ, Yan kenar algoritması 1.8 mJ, Çift öncelikli algoritma 1.6 ve KUD 1.4 mJ şeklinde sıralanmaktadır.

6. SONUÇ

Bu yüksek lisans tezi kapsamında iki farklı tepe örtüsü algoritması önerilmiş olup bu algoritmaların teorik analizleri ve TOSSIM benzetim ortamında pratik sonuçları alınarak performans analizleri yapılmıştır. Bütünleşik tepe örtüsü algoritmaları ile bir ağaç elde edilmiş olup bu ağaç üzerinden tepe örtüsü işlemi yapmak ayrı ayrı çalıştırılan EİAA ve tepe örtüsü algoritmalarının toplamından daha iyi sonuçlar vermiştir. Hali hazırda TDA için bir yönlendirme ağacı tüm topolojiler ve uygulamalar için kurulduğu için bu ağaç üzerine tepe örtüsü yaparak çizgedeki tepelerin örtülmemiş olanlarının radyo alıcısını kapatarak önemli ölçüde enerji tasarrufu sağlanabilmektedir. Yalnızca tepe örtüsü kümesine dahil olan tepeler komşularından mesaj bekleyeceklerdir.

Önerilen iki algoritma da birbirlerine yakın sonuçlar vermekle birlikte, EİAA ve Açgözlü Algoritmanın toplamından enerji, alınan bayt, gönderilen bayt, zaman olarak daha iyi sonuçlar vermişlerdir. Ayrıca önerdiğimiz algoritmalarında temelini oluşturan KUD algoritmasına da yakın sonuçlar verdiği gibi tepe örtüsü kümesinde daha başarılı olmuştur. Tepe örtüsü kümesinin boyutunda en iyi sonucu Açgözlü Algoritma verirken, KUD algoritması alınan tüm sonuçlarda daha fazla tepe örterek diğer algoritmalara göre daha kötü sonuçlar vermiştir. Yan Kenar Algoritması tek numaralı seviyelerde daha fazla yan kenar bulunması halinde oldukça iyi sonuçlar vermiş olup zaman zaman KUD algoritmasına yakın enerji ve bayt karmaşıklığı sonuçları vermiştir.

İleriye dönük olarak tepe örtüsü problemine ait öz-kararlı ve kapasite kısıtlı algoritmalar üzerinde bu kodların geliştirilmesi ve gerçek cihazlar üzerinde denenmesi ve bu tezin uluslararası bir dergide yayınlanması düşünülmektedir.

KAYNAKLAR DİZİNİ

Akram, V. and Dagdeviren, O., 2013, Breadth-First Search-Based Single-Phase Algorithms for Bridge Detection in Wireless Sensor Networks. *Sensors*, *13*(7), 8786–8813.

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E., 2002, Wireless sensor networks: a survey. *Computer networks*, *38*(4), 393–422.

Arapoğlu Ö., 2015, Telsiz Duyarga Ağları İçin Hakim Küme Algoritmaları, Yüksek Lisans Tezi, Ege Üniversitesi, Fen Bilimleri Enstitüsü, İzmir

Cormen, T. H., and Cormen, T. H. (Ed.)., 2001, *Introduction to algorithms* (2nd ed). Cambridge, Mass: MIT Press.

Erciyes, K., 2013, *Distributed Graph Algorithms for Computer Networks*. London: Springer London. 324 p.

Ergen, S. C., and Varaiya, P., 2010, TDMA scheduling algorithms for wireless sensor networks. *Wireless Networks*, *16*(4), 985–997. http://doi.org/10.1007/s11276-009-0183-0

Hoepman, J.-H. 2004, Simple distributed weighted matchings, *arXiv preprint cs/0410047*

http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf, Erişim Tarihi: 25.05.2016

Ileri, C. U., Ural, C. A., Dagdeviren, O., and V, Kavalci., 2015, On Vertex Cover Problems in Distributed System, *Advanced Method for Complex Network Analysis*, (ss. 1-29) IGI-Global.

Karl, H., and Willig, A., 2005, *Protocols and architectures for wireless sensor networks*. Hoboken, NJ: Wiley.

KAYNAKLAR DİZİNİ (devam)

Karp, R. M., 1972, Reducibility among Combinatorial Problems. Içinde R. E. Miller, J. W. Thatcher, & J. D. Bohlinger (Ed.), *Complexity of Computer Computations* (ss. 85–103). Springer US.

Kavalci, V., Ural, A., and Dagdeviren, O., 2014, Distributed Vertex Cover Algorithms for Wireless Sensor Networks. *International journal of Computer Networks & Communications*, *6*(1), 95–110.

Kiniwa, J., 2005**,** Approximation of self-stabilizing vertex cover less than 2. (p. 171–182). Springer.

Li, Y., Thai, M. T., and Wu, W. (Ed.)., 2008, *Wireless Sensor Networks and Applications*. Boston, MA: Springer US.

Lu, K., Huang, L., Wan, Y., and Xu, H., 2005, Energy-efficient data gathering in large wireless sensor networks. (s. 5)

Mahajan, S., and Malhotra, J., 2011, Energy Efficient Path Determination in Wireless Sensor Network Using BFS Approach. *Wireless Sensor Network*, *3*(11), 351–356.

Panconesi, A., Karoński, M., and Hańćkowiak, M., 1997, *On the distributed complexity of computing maximal matchings*. Computer Science Department.

Parnas, M., and Ron, D., 2007, Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, *381*(1–3), 183–196.

Polishchuk, V., and Suomela, J., 2009, A simple local 3-approximation algorithm for vertex cover. *Information Processing Letters*, *109*(12), 642

Turau, V., 2006, Computing bridges, articulations, and 2-connected components in wireless sensor networks. *Algorithmic Aspects of Wireless Sensor Networks* (ss. 164–175). Springer.

ÖZGEÇMİŞ

1990 yılında İzmir'de dünyaya gelen Yasin YİĞİT, Bornova Cem Bakioğlu Anadolu lisesinden mezun olduğu 2009 yılında Ege Üniversitesi Ziraat Fakültesi Tarım makinaları bölümünde lisans eğitimine başlamıştır. 2013 yılında lisans eğitiminin hemen ardında Uluslararası Bilgisayar Enstitüsünde yüksek lisans eğitimine başlamıştır. Tezsiz yüksek lisans öğrencisi olarak girdiği enstitüye tezli yüksek lisans öğrencisi olarak devam etmektedir.

Çalışma alanları duyarga ağları, çizge teorisi ve dağıtık algoritmalardır.

EKLER

EK 1 Çift Öncelikli Algoritmanın Kodları
EK 2 Yan Kenar Algoritmasının Kodları

EK 1 ÇİFT ÖNCELİKLİ ALGORİTMANIN KODLARI

EvenFirstApp.nc

```
#define AM_RADIO 6
configuration EvenFirstApp {}
implementation
{
  components MainC,EvenFirst as App;
  components LedsC,RandomC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  components new TimerMilliC() as TRound;
  components ActiveMessageC;
  App.Boot -> MainC.Boot;
  App.AMSend -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.Packet -> AMSenderC;
  App.TRound->TRound;
  App.TFinish->TFinish;

  App.HReceive -> AMReceiverC;
  App.FReceive -> AMReceiverC;
  App.BReceive -> AMReceiverC;
  App.DReceive -> AMReceiverC;
  App.Random->RandomC;
  App.Leds -> LedsC;
}
```

EvenFirst.nc

```
                #include <Timer.h>
#include <stdio.h>
#define SIM_TIME sim_time()/10000000000.0
#define SINK 0

#define SINK 0

#define NODECOUNT 250
#define TIMESLOT 100
```

```
#define DECIDE 1
#define UNDEC 2

int send_byte = 0;
int recv_byte = 0;
int send_count = 0;
int recv_count = 0;
int covered_node = 0;

bool bfs_const = FALSE;


typedef nx_struct Hpacket
{
nx_uint16_t source;
}Hpacket;

typedef nx_struct Fpacket
{
nx_uint16_t source;
nx_uint16_t parent;
nx_uint16_t level;
}Fpacket;

typedef nx_struct Bpacket
{
nx_uint16_t source;
nx_uint16_t dest;
nx_uint16_t cross;
nx_uint16_t level;
}Bpacket;

typedef nx_struct Dpacket
{
nx_uint16_t source;
nx_uint16_t cross;
nx_uint16_t level;
nx_uint8_t type;
```

```
}Dpacket;

module EvenFirst
{
uses
{
interface Leds;
interface Random;
interface Receive as HReceive;
interface Receive as FReceive;
interface Receive as BReceive;
interface Receive as DReceive;

interface SplitControl as AMControl;
interface Packet;
interface Timer<TMilli> as TRound;

interface Boot;
interface AMSend;

}
}

implementation
{

int counter = 0;

int my_parent = 1000;
int my_level = 1000;
int degree=0;
int my_cross=0;
int round =  0;
bool radioBusy=FALSE,
packetReady=FALSE,
forward_recvd = FALSE,
forward_send = FALSE,
terminate=FALSE,
```

```
covered=FALSE,
decide_recvd=FALSE,
finish_forward=FALSE;

bool hello_finish = FALSE;
bool first_forward_sended = FALSE;
bool covered_phase=FALSE;
int neighs[NODECOUNT] = {NULL};
int children[NODECOUNT] = {NULL};
int backward_children[NODECOUNT] = {NULL};
int others[NODECOUNT] = {NULL};
int neighcrossedge[NODECOUNT] = {NULL};
int neighs_layer[NODECOUNT] = {NULL};



message_t message;


bool compare(int *array1, int *array2) //backward children ve children arraylerini
karşılaştırmak için fonksiyon
{
int i;
for( i=0;i<NODECOUNT; i++)
{

if(array1[i] != array2[i])
{

return FALSE;
}
}

return TRUE;
}
```

```
bool all_neighs_response(int *neighs, int *childs , int *others, int parent) //tüm komşularımı tanımladım mı ?
{
int i;
for (i= 0; i < NODECOUNT; i++)
{
if(i==parent)
continue;

if( (childs[i] | others[i] )  != neighs[i])
{
return FALSE;
}
}
return TRUE;
}


bool  imax(int *neighs, int degree, int id) /*komşularım içerisinde en büyük derecelimiyim,
                                                    eşitlikte büyük idli nod seçilir*/
{
int i;
int neighs_id;
int max=neighs[0];

for (i = 0; i < NODECOUNT; i++)
{
if(neighs[i]>=max)
{
max = neighs[i];
neighs_id = i;
}
}

if(degree>max)
{
```

```
return TRUE;
}
if(degree<max)
{
return FALSE;
}
if(degree==max)
{
if(id>neighs_id)
{
return TRUE;
}
else
{
return FALSE;
}
}
}



event void Boot.booted()
{
call AMControl.start();
}

event void AMControl.startDone(error_t error)
{
if (error != SUCCESS)
{
call AMControl.start();
}
else
{

call TRound.startOneShot(((TOS_NODE_ID)*TIMESLOT));



}
```

```
}

event void TRound.fired() //her nod komşularına hello mesajı atıyor bu sayede herkes komşularını bilmiş oluyor.
{
counter++;
dbg_clear("OUT","%4.3f | %-3d %d \n",SIM_TIME,TOS_NODE_ID,counter);

call TRound.startOneShot(NODECOUNT*TIMESLOT);


if(counter==1)
{
//paketi hazırla
Hpacket* packet;
packet = (Hpacket*)call Packet.getPayload(&message, sizeof(Hpacket));
packet->source=TOS_NODE_ID;

packetReady=TRUE;
if(radioBusy==FALSE && packetReady==TRUE)
{
	if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Hpacket)) == SUCCESS)
	{
		dbg_clear("OUT","%4.3f | %-3d Sent HEEEEEEELLO %d \n",SIM_TIME,TOS_NODE_ID,sizeof(Hpacket));
		radioBusy=TRUE;
		packetReady=FALSE;
		send_count++;
		send_byte += sizeof(Hpacket);
	}
}

hello_finish = TRUE;
}
else if((hello_finish == TRUE) && (finish_forward==FALSE) )
{
```

```
if((TOS_NODE_ID==0) && (first_forward_sended == FALSE) && counter ==
2)
{
	Fpacket* packet;
	my_level = 0;
	my_parent = 0;

	forward_send = TRUE;
	packet = (Fpacket*)call Packet.getPayload(&message, sizeof(Fpacket));
	packet->source=TOS_NODE_ID;

	packet->level = my_level;
	packet->parent = my_parent;
	packetReady=TRUE;

	if(radioBusy==FALSE && packetReady==TRUE)
	{
		if (call AMSend.send(AM_BROADCAST_ADDR,&message,
sizeof(Fpacket)) == SUCCESS)
		{
			dbg_clear("OUT","%4.3f | %-3d Sent FORWARD size of
%d and \n",SIM_TIME,TOS_NODE_ID,sizeof(Fpacket));
			radioBusy=TRUE;
			packetReady=FALSE;
			send_count++;
			send_byte += sizeof(Fpacket);
			finish_forward = TRUE;
		}
	}
	first_forward_sended = TRUE;

}
else if( (my_level==counter-1))
{
	Fpacket* packet;
	packet = (Fpacket*)call Packet.getPayload(&message, sizeof(Fpacket));
	packet->source=TOS_NODE_ID;
	packet->level= my_level;
```

```
	packet->parent = my_parent;
	packetReady=TRUE;

	if(radioBusy==FALSE && packetReady==TRUE)
	{
		if (call AMSend.send(AM_BROADCAST_ADDR,&message,
sizeof(Fpacket)) == SUCCESS)
		{
			dbg_clear("OUT","%4.3f | %-3d Sent  FORWARD size of
%d  \n",SIM_TIME,TOS_NODE_ID,sizeof(Fpacket));
			radioBusy=TRUE;
			packetReady=FALSE;
			send_count++;
			send_byte += sizeof(Fpacket);
			finish_forward = TRUE;
		}
	}

}
}
else if(all_neighs_response(neighs,children,others,my_parent) &&
compare(children,backward_children) && covered_phase==FALSE)
{
Bpacket* packet;

packet = (Bpacket*)call Packet.getPayload(&message, sizeof(Bpacket));
//parentıma backward atıyorum
packet->source=TOS_NODE_ID;
packet->level = my_level;
packet->dest=my_parent;
packet->cross=my_cross;
packetReady=TRUE;
if(radioBusy==FALSE && packetReady==TRUE)
{
	if (call AMSend.send(AM_BROADCAST_ADDR,&message,
sizeof(Bpacket)) == SUCCESS)
	{
```

```
                dbg_clear("OUT","%4.3f | %-3d Sent  Packet ---BACKWARD---
size of %d  \n",SIM_TIME,TOS_NODE_ID,sizeof(Bpacket));
                radioBusy=TRUE;
                packetReady=FALSE;
                send_count++;
                send_byte += sizeof(Bpacket);
                if(TOS_NODE_ID==SINK)
                {
                        bfs_const = TRUE;
                }
        }
}
terminate = TRUE;
covered_phase = TRUE;
}
else if(covered==FALSE && bfs_const == TRUE)
{
int i,j,k,l;
Dpacket* packet;
if(my_level%2 == 0 && covered == FALSE)
{
        covered = TRUE;
        covered_node++;
        my_cross = 0;
}
else
{
        if(my_cross>                    0                    &&
imax(neighcrossedge,my_cross,TOS_NODE_ID)==TRUE  &&  covered  ==
FALSE)
        {
                covered = TRUE;
                covered_node++;
                my_cross = 0;
                packet  =  (Dpacket*)call  Packet.getPayload(&message,
sizeof(Dpacket));
                packet->type = DECIDE;
                packet->source=TOS_NODE_ID;
```

```
                packet->level = my_level;
                packet->cross=my_cross;
                packetReady=TRUE;
                if(radioBusy==FALSE && packetReady==TRUE)
                        {
                                if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Dpacket)) == SUCCESS)
                                        {
                                                dbg_clear("OUT","%4.3f | %-3d Sent Packet -DECICE- size of %d \n",SIM_TIME,TOS_NODE_ID,sizeof(Dpacket));
                                                radioBusy=TRUE;
                                                packetReady=FALSE;
                                                send_count++;
                                                send_byte += sizeof(Dpacket);
                                        }
                        }
        }
        else
        {
                if(decide_recvd==TRUE && covered == FALSE)
                {
                        packet = (Dpacket*)call Packet.getPayload(&message, sizeof(Dpacket));
                        packet->type = UNDEC;
                        packet->source=TOS_NODE_ID;
                        packet->level = my_level;
                        packet->cross=my_cross;
                        packetReady=TRUE;
                        if(radioBusy==FALSE && packetReady==TRUE)
                        {
                                if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Dpacket)) == SUCCESS)
                                {
                                        dbg_clear("OUT","%4.3f | %-3d Sent Packet -UNDEC- size of %d  \n",SIM_TIME,TOS_NODE_ID,sizeof(Dpacket));
```

```
                                        radioBusy=TRUE;
                                        packetReady=FALSE;
                                        send_count++;
                                        send_byte += sizeof(Dpacket);
                                }
                        }
                }
        }

}
decide_recvd = FALSE;
if(covered==TRUE || my_cross ==0)
{
        call TRound.stop();
}
}
}

event message_t* HReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{
if(len==sizeof(Hpacket))
{ //hello paketi alırsam
Hpacket* payload = (Hpacket*)pkt;
dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of
%d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
neighs[payload->source] = 1; //gönderen nodu komşuluk listeme ekliyorum
degree++;
recv_count++;
recv_byte += sizeof(Hpacket);
}

return bufPtr;
}
event message_t* FReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{
if(len == sizeof(Fpacket))//forward mesajı alırsam
{
Fpacket* payload = (Fpacket*) pkt;
```

```
dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
forward_recvd = TRUE;
recv_count++;
recv_byte += sizeof(Fpacket);
forward_recvd = TRUE;

if(my_parent==1000) //parentımı set etmediysem
{
my_parent = payload->source; //mesajı gönderen nodu parentım yapıyorum
my_level = (payload->level)+1; //levelimi arttırıyorum
}
else if(payload->parent == TOS_NODE_ID) //mesajı gönderen nod parentında beni gösteriyorsa
{
children[payload->source] = 1; //child listeme ekliyorum
}
else //hiç biri değilse others listeme ekliyorum
{
others[payload->source] = 1;
}
if(payload->level == my_level)
{
my_cross++; //eğer kendi layerımdan forward alırsam cross edge sayımı arttırıyorum
}
}
return bufPtr;
}

event message_t* BReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{

if(len == sizeof(Bpacket)) //backward mesajı aldığımda
{
Bpacket* payload = (Bpacket*) pkt;
if(payload->dest==TOS_NODE_ID)
{
```

```
dbg_clear("OUT","%4.3f | %-3d Received ---BACKWARD--- Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
backward_children[payload->source] = 1; //mesaj bana gönderildiyse gönderen nodu backward_children listeme ekliyorum
recv_count++;
recv_byte += sizeof(Bpacket);
}
if(payload->level==my_level)
{
dbg_clear("OUT","%4.3f | %-3d Received ---BACKWARD--- Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
neighcrossedge[payload->source] = payload->cross;
neighs_layer[payload->source] = 1;
recv_count++;
recv_byte += sizeof(Bpacket);
}
}
return bufPtr;
}

event message_t* DReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{
Dpacket* payload = (Dpacket*) pkt;
if (len == sizeof(Dpacket) && payload->type == DECIDE)
{
if(payload->level == my_level && my_cross > 0)
{
dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
recv_count++;
recv_byte += sizeof(Dpacket);
decide_recvd = TRUE;
my_cross--;
neighcrossedge[payload->source] = 0;
}
}

if(len == sizeof(Dpacket) && payload->type == UNDEC)
```

```
{
if (payload->level == my_level && my_cross > 0)
{
dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
recv_count++;
recv_byte += sizeof(Dpacket);
neighcrossedge[payload->source] = payload->cross;
}
}
return bufPtr;

}

event void AMControl.stopDone(error_t error) {}
event void AMSend.sendDone(message_t* bufPtr, error_t error)
{
radioBusy=FALSE;

}

}
```

EK 2 YAN KENAR ALGORİTMASININ KODLARI

CrossEdgeApp.nc

```
#define AM_RADIO 6
configuration CrossEdgeApp {}
implementation
{
  components MainC,EvenFirst as App;
  components LedsC,RandomC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  components new TimerMilliC() as TRound;
  components ActiveMessageC;

  App.Boot -> MainC.Boot;
  App.AMSend -> AMSenderC;
  App.AMControl -> ActiveMessageC;
  App.Packet -> AMSenderC;
  App.TRound->TRound;

  App.HReceive -> AMReceiverC;
  App.FReceive -> AMReceiverC;
  App.BReceive -> AMReceiverC;
  App.DReceive -> AMReceiverC;

  App.Random->RandomC;
  App.Leds -> LedsC;
}
```

CrossEdge.nc

```
        #include <Timer.h>
#include <stdio.h>
#define SIM_TIME sim_time()/10000000000.0
#define SINK 0

#define SINK 0
```

```
#define NODECOUNT 250
#define TIMESLOT 100
#define DECIDE 1
#define UNDEC 2

int send_byte = 0;
int recv_byte = 0;
int send_count = 0;
int recv_count = 0;
int covered_node = 0;
int odd_cross =0;
int even_cross = 0;
bool bfs_const = FALSE;
double alg_time ;
typedef nx_struct Hpacket
{
nx_uint16_t source;
}Hpacket;

typedef nx_struct Fpacket
{
nx_uint16_t source;
nx_uint16_t parent;
nx_uint16_t level;
}Fpacket;

typedef nx_struct Bpacket
{
nx_uint16_t source;
nx_uint16_t dest;
nx_uint16_t cross;
nx_uint16_t odd_level_cross_edge;
nx_uint16_t even_level_cross_edge;
nx_uint16_t level;
}Bpacket;

typedef nx_struct Dpacket
{
```

```
nx_uint16_t source;
nx_uint16_t cross;
nx_uint16_t level;
nx_uint8_t type;
}Dpacket;

typedef nx_struct Choosenpacket
{
nx_uint16_t source;
nx_uint8_t choose;
}Choosenpacket;

module CrossEdge
{
uses
{
interface Leds;
interface Random;
interface Receive as HReceive;
interface Receive as FReceive;
interface Receive as BReceive;
interface Receive as DReceive;
interface Receive as ChoosenReceive;
interface SplitControl as AMControl;
interface Packet;
interface Timer<TMilli> as TRound;
interface Boot;
interface AMSend;

}
}

implementation
{

int counter = 0;
int my_parent = 1000;
int my_level = 1000;
```

```
int degree=0;
int my_cross=0;
int round =  0;
bool radioBusy=FALSE,
packetReady=FALSE,
forward_recvd = FALSE,
forward_send = FALSE,
begin_backward = FALSE,
covered=FALSE,
decide_recvd=FALSE,
finish_forward=FALSE;
bool hello_finish = FALSE;
bool first_forward_sended = FALSE;
bool covered_phase=FALSE;
bool has_a_child = FALSE;

bool choice_recvd= FALSE;
bool choice_sended = FALSE;
int neighs[NODECOUNT] = {NULL};
int children[NODECOUNT] = {NULL};
int backward_children[NODECOUNT] = {NULL};
int others[NODECOUNT] = {NULL};
int neighcrossedge[NODECOUNT] = {NULL};
int neighs_layer[NODECOUNT] = {NULL};
int choice = 0;



message_t message;


bool compare(int *array1, int *array2) //backward children ve children arraylerini
karşılaştırmak için fonksiyon
{
int i;
for( i=0;i<NODECOUNT; i++)
{
if(array1[i] != array2[i])
```

```
{
return FALSE;
}
}
return TRUE;
}

bool all_neighs_response(int *neighs, int *childs , int *others, int parent) //tüm komşularımı tanımladım mı ?
{
int i;
for (i= 0; i < NODECOUNT; i++)
{
if(i==parent)
continue;
if( (childs[i] | others[i] )  != neighs[i])
{
return FALSE;
}

}
return TRUE;
}


bool  imax(int *neighs, int degree, int id) /*komşularım içerisinde en büyük derecelimiyim,
eşitlikte büyük idli nod seçilir*/
{
int i;
int neighs_id;
int max=neighs[0];

for (i = 0; i < NODECOUNT; i++)
{
if(neighs[i]>=max)
{
max = neighs[i];
```

```
neighs_id = i;
}
}
if(degree>max)
{
return TRUE;
}
if(degree<max)
{
return FALSE;
}
if(degree==max)
{
if(id>neighs_id)
{
return TRUE;
}
else
{
return FALSE;
}
}



}

event void Boot.booted()
{
call AMControl.start();
}

event void AMControl.startDone(error_t error)
{
if (error != SUCCESS)
{
call AMControl.start();
}
else
```

```
{
call TRound.startOneShot(((TOS_NODE_ID)*TIMESLOT));
}
}
event void TRound.fired() //her nod komşularına hello mesajı atıyor bu sayede herkes komşularını bilmiş oluyor.
{
counter++;
dbg_clear("OUT","%4.3f | %-3d %d \n",SIM_TIME,TOS_NODE_ID,counter);

call TRound.startOneShot(NODECOUNT*TIMESLOT);
if(counter==1)
{
//paketi hazırla
Hpacket* packet;
packet = (Hpacket*)call Packet.getPayload(&message, sizeof(Hpacket));
packet->source=TOS_NODE_ID;
packetReady=TRUE;
if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Hpacket)) == SUCCESS)
{
//dbg_clear("OUT","%4.3f | %-3d Sent HELLO %d \n",SIM_TIME,TOS_NODE_ID,sizeof(Hpacket));
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Hpacket);
}
}

hello_finish = TRUE;
}
else if((hello_finish == TRUE) && (begin_backward ==FALSE) && (finish_forward==FALSE) )
{
```

```
if((TOS_NODE_ID==0) && (first_forward_sended == FALSE) && counter == 2)
{
Fpacket* packet;
my_level = 0;
my_parent = 0;
forward_send = TRUE;
packet = (Fpacket*)call Packet.getPayload(&message, sizeof(Fpacket));
packet->source=TOS_NODE_ID;
packet->level = my_level;
packet->parent = my_parent;
packetReady=TRUE;

if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Fpacket)) == SUCCESS)
{
dbg_clear("OUT","%4.3f | %-3d Sent FORWARD size of %d and \n",SIM_TIME,TOS_NODE_ID,sizeof(Fpacket));
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Fpacket);
finish_forward = TRUE;
}
}
first_forward_sended = TRUE;
call TRound.startOneShot(((counter+1)*(TOS_NODE_ID+1)*TIMESLOT)+(TOS_NODE_ID+1)*TIMESLOT);
}
else if( (my_level==counter-1))
{
Fpacket* packet;
packet = (Fpacket*)call Packet.getPayload(&message, sizeof(Fpacket));
packet->source=TOS_NODE_ID;
packet->level= my_level;
```

```
packet->parent = my_parent;
packetReady=TRUE;

if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Fpacket))
== SUCCESS)
{

dbg_clear("OUT","%4.3f | %-3d Sentttttt FORWARD size of %d
\n",SIM_TIME,TOS_NODE_ID,sizeof(Fpacket));
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Fpacket);
finish_forward = TRUE;

}
}

}
}
else if(all_neighs_response(neighs,children,others,my_parent) &&
compare(children,backward_children) && covered_phase==FALSE)
{
if(TOS_NODE_ID!=0)
{
Bpacket* packet;
packet = (Bpacket*)call Packet.getPayload(&message, sizeof(Bpacket));
//parentıma backward atıyorum
packet->source=TOS_NODE_ID;
packet->level = my_level;
packet->dest=my_parent;
packet->cross=my_cross;

if(my_level %2 == 0)
{
packet->even_level_cross_edge = even_cross+my_cross;
```

```
packet->odd_level_cross_edge = odd_cross;
}
else
{
packet->even_level_cross_edge = even_cross;
packet->odd_level_cross_edge = odd_cross+my_cross;
}
packetReady=TRUE;
if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Bpacket))
== SUCCESS)
{
dbg_clear("OUT","%4.3f | %-3d Sent  Packet ---BACKWARD--- size of %d
\n",SIM_TIME,TOS_NODE_ID,sizeof(Bpacket));
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Bpacket);
}
}
}
else if(TOS_NODE_ID==0)
{
Choosenpacket* packet;
packet = (Choosenpacket*)call Packet.getPayload(&message,
sizeof(Choosenpacket));
packet->source = TOS_NODE_ID;
call TFinish.startOneShot(4000000);
if(even_cross>=odd_cross)
{
packet->choose = 0;
covered=TRUE;
}
else
{
packet->choose = 1;
}
```

```
packetReady=TRUE;

if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message,
sizeof(Choosenpacket)) == SUCCESS)
{
dbg_clear("OUT","%4.3f | %-3d Sent Packet CHOICE size of %d
\n",SIM_TIME,TOS_NODE_ID,sizeof(Hpacket));
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Choosenpacket);
choice_sended = TRUE;

}
}
}

covered_phase = TRUE;
}

else if(choice_recvd ==TRUE && has_a_child == TRUE && choice_sended ==
FALSE)
{
Choosenpacket* packet;
packet = (Choosenpacket*)call Packet.getPayload(&message,
sizeof(Choosenpacket));
packet->source = TOS_NODE_ID;
packet->choose = choice;
packetReady=TRUE;
if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message,
sizeof(Choosenpacket)) == SUCCESS)
{
dbg_clear("OUT","%4.3f | %-3d Sent Packet CHOICE %d size of %d
\n",SIM_TIME,TOS_NODE_ID,choice,sizeof(Hpacket));
```

```
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Choosenpacket);
choice_sended = TRUE;
}
}
}
else if(covered==FALSE && choice_recvd ==TRUE)
{
int i,j,k,l;
Dpacket* packet;

if(my_level%2 == choice && covered == FALSE)
{
covered = TRUE;
covered_node++;
my_cross = 0;
}
else
{
if( my_cross> 0 && imax(neighcrossedge,my_cross,TOS_NODE_ID)==TRUE
&& covered == FALSE)
{
covered = TRUE;
covered_node++;
my_cross = 0;
packet = (Dpacket*)call Packet.getPayload(&message, sizeof(Dpacket));
packet->type = DECIDE;
packet->source=TOS_NODE_ID;
packet->level = my_level;
packet->cross=my_cross;
packetReady=TRUE;
if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Dpacket))
== SUCCESS)
{
```

```
dbg_clear("OUT","%4.3f | %-3d Sent Packet -DECICE- size of %d
\n",SIM_TIME,TOS_NODE_ID,sizeof(Dpacket));
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Dpacket);
}
}
}
else
{
if(decide_recvd==TRUE && covered == FALSE)
{
packet = (Dpacket*)call Packet.getPayload(&message, sizeof(Dpacket));
packet->type = UNDEC;
packet->source=TOS_NODE_ID;
packet->level = my_level;
packet->cross=my_cross;
packetReady=TRUE;
if(radioBusy==FALSE && packetReady==TRUE)
{
if (call AMSend.send(AM_BROADCAST_ADDR,&message, sizeof(Dpacket))
== SUCCESS)
{
dbg_clear("OUT","%4.3f | %-3d Sent Packet -UNDEC- size of %d
\n",SIM_TIME,TOS_NODE_ID,sizeof(Dpacket));
radioBusy=TRUE;
packetReady=FALSE;
send_count++;
send_byte += sizeof(Dpacket);
}
}
}
}


}

decide_recvd = FALSE;
```

```
if(covered==TRUE || my_cross ==0)
{
call TRound.stop();
}
}



}
event message_t* HReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{
if(len==sizeof(Hpacket))
{ //hello paketi alırsam
Hpacket* payload = (Hpacket*)pkt;

dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of
%d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);



neighs[payload->source] = 1; //gönderen nodu komşuluk listeme ekliyorum
degree++;
recv_count++;
recv_byte += sizeof(Hpacket);
}

return bufPtr;

}

event message_t* FReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{
if(len == sizeof(Fpacket))//forward mesajı alırsam
{
Fpacket* payload = (Fpacket*) pkt;
dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of
%d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
forward_recvd = TRUE;
recv_count++;
```

```
recv_byte += sizeof(Fpacket);
forward_recvd = TRUE;

if(my_parent==1000) parentımı set etmediysem
{
my_parent = payload->source; //mesajı gönderen nodu parentım yapıyorum
my_level = (payload->level)+1; //levelimi arttırıyorum
}
else if(payload->parent == TOS_NODE_ID) //mesajı gönderen nod parentında beni gösteriyorsa
{
children[payload->source] = 1; //child listeme ekliyorum
has_a_child = TRUE;
}
else //hiç biri değilse others listeme ekliyorum
{
others[payload->source] = 1;
}

if(payload->level == my_level)
{
my_cross++; //eğer kendi layerımdan forward alırsam cross edge sayımı arttırıyorum
}
}

return bufPtr;

}

event message_t* BReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{

if(len == sizeof(Bpacket)) //backward mesajı aldığımda
{
Bpacket* payload = (Bpacket*) pkt;

if(payload->dest==TOS_NODE_ID)
```

```
{
dbg_clear("OUT","%4.3f | %-3d Received ---BACKWARD--- Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
backward_children[payload->source] = 1; //mesaj bana gönderildiyse gönderen nodu backward_children listeme ekliyorum
recv_count++;
recv_byte += sizeof(Bpacket);
odd_cross += payload->odd_level_cross_edge;
even_cross += payload->even_level_cross_edge;
}
if(payload->level==my_level)
{
dbg_clear("OUT","%4.3f | %-3d Received ---BACKWARD--- Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
neighcrossedge[payload->source] = payload->cross;
recv_count++;
recv_byte += sizeof(Bpacket);
}
}

return bufPtr;
}

event message_t* DReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{
Dpacket* payload = (Dpacket*) pkt;
if (len == sizeof(Dpacket) && payload->type == DECIDE)
{
if(payload->level == my_level && my_cross > 0)
{
dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
recv_count++;
recv_byte += sizeof(Dpacket);
decide_recvd = TRUE;
my_cross--;
neighcrossedge[payload->source] = 0;
}
```

```
}

if(len == sizeof(Dpacket) && payload->type == UNDEC)
{
if (payload->level == my_level && my_cross > 0)
{
dbg_clear("OUT","%4.3f | %-3d Received Packet from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
recv_count++;
recv_byte += sizeof(Dpacket);
neighcrossedge[payload->source] = payload->cross;
}
}

return bufPtr;
}
event message_t* ChoosenReceive.receive(message_t* bufPtr,void* pkt, uint8_t len)
{
Choosenpacket* payload = (Choosenpacket*) pkt;

if(len == sizeof(Choosenpacket) && payload->source == my_parent)
{
dbg_clear("OUT","%4.3f | %-3d Received Packet CHOOOOSEOSOEOSAOEOAES from %d size of %d\n",SIM_TIME,TOS_NODE_ID, payload->source, len);
choice_recvd =TRUE;
recv_count++;
recv_byte += sizeof(Choosenpacket);
choice = payload->choose;
}

return bufPtr;
}
event void AMControl.stopDone(error_t error) {}
event void AMSend.sendDone(message_t* bufPtr, error_t error)
{
radioBusy=FALSE; } }
```