Gümüşhane Üniversitesi Fen Bilimleri Dergisi

GUFBD / *GUJS* (2025) 15(2): 516-532 doi: 10.17714/gumusfenbil.1569247 Araştırma Makalesi / *Research Article*

Büyük ölçekli ağlar için polinom zamanlı kritik düğüm tespiti algoritması

A polynomial-time algorithm for critical node detection in large-scale networks

Onur UĞURLU*[1], Yeşim AYGÜL[1]

[1]*İzmir Bakırçay Üniversitesi, Mühendislik ve Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü, İzmir*

• Geliş tarihi / *Received*: 17.10.2024 • Kabul tarihi / *Accepted*: 05.05.2025

Öz

Ağlar, karmaşık sistemlerin modellenmesinde ve analiz edilmesinde kritik bir rol oynar; bu nedenle ağ analizleri hem sosyal hem de teknolojik sistemlerde önemli veriler sağlar. Bu çalışmada, ağlardaki kritik düğümleri tespit etmeyi amaçlayan Kritik Düğüm Problemi (KDP) ele alınmıştır. KDP, bir ağdaki ikili bağlantılılığı minimize eden düğümlerin bulunmasını amaçlayan bir problemdir. Başka bir deyişle, KDP, silinmesiyle ağı benzer boyutlarda çok sayıda segmente bölen düğüm kümesini bulmayı hedefler. Bu problem, sosyal ağlardan biyolojik sistemlere, telekomünikasyon ağlarından kablosuz çok sekmeli ağlara kadar birçok alanda önemli uygulamalara sahiptir. KDP, NP-Zor bir problem olduğundan büyük ölçekli ağlarda optimale yakın etkin çözümlerin bulunması için polinom zamanlı algoritmalara ihtiyaç duyulmaktadır. Bu çalışmanın temel motivasyonu, özellikle büyük ölçekli ağlar üzerinde optimale yakın çözümler bulabilecek polinom zamanlı bir algoritma geliştirmektir. Çalışmada önerilen TrimCut algoritması, ağın dayanıklılığını azaltan kritik düğümleri belirlemek için üç aşamalı bir süreç kullanmaktadır. İlk aşamada, ağdan yüksek dereceli düğümler çıkarılarak ağ seyreltilir, ardından ikinci aşamada Derinlik Öncelikli Arama ağacı ile kesim düğümleri tespit edilir. Son aşamada ise ağdaki kritik düğümler tespit edilir. Önerilen algoritmanın performansı, literatürdeki mevcut algoritmalarla karşılaştırılmıştır. Hesaplamalı deneyler, TrimCut algoritmasının mevcut kütüphane örnekleri üzerindeki hata oranının %6'nın altında olduğunu ve büyük ölçekli ağlarda kaliteli çözümler bulabildiğini göstermektedir.

Anahtar kelimeler: Ağ analizi, Kritik düğümler, Polinom zamanlı algoritmalar

Abstract

Networks play a crucial role in modeling and analyzing complex systems, providing valuable data in both social and technological systems. This study addresses the Critical Node Problem (CNP), which aims to identify critical nodes within networks. CNP seeks to find the nodes whose removal minimizes pairwise connectivity in the network. In other words, CNP aims to identify a set of nodes whose deletion fragments the network into several segments of similar size. This problem has significant applications in various fields, ranging from social networks to biological systems, from telecommunication networks to wireless multi-hop networks. As CNP is an NP-Hard problem, developing polynomial-time algorithms is essential to find efficient solutions for large-scale networks. The primary motivation of this study is to develop a polynomial-time algorithm capable of finding near-optimal solutions, particularly for large-scale networks. The proposed TrimCut algorithm employs a three-stage process to identify critical nodes that reduce network resilience. In the first stage, the network is sparsified by removing high-degree nodes. Then, in the second stage, a Depth-First Search (DFS) tree is used to detect cut nodes. Finally, in the last stage, the critical nodes in the network are identified. The performance of the proposed algorithm has been compared with existing algorithms in the literature. Computational experiments demonstrate that the TrimCut algorithm achieves an error rate below 6% on standard benchmark networks and provides high-quality solutions for large-scale networks.

Keywords*: Network analysis, Critical nodes, Polynomial-time algorithms*

*Onur UĞURLU; onur.ugurlu@bakircay.edu.tr

1. Giriş
1. Introduction

Günümüzde ağlar, toplumsal ve teknolojik altyapının temel unsurlarından biri haline gelmiştir. İnternetten sosyal medya platformlarına, ulaşım sistemlerinden biyolojik yapılara kadar pek çok alanda, sistemler arasındaki ilişkileri temsil etmek için ağ yapıları kullanılır. Matematiksel olarak bu ağlar, düğümler ve bu düğümler arasındaki bağlantıları ifade eden kenarlardan oluşan bir çizge modeli ile temsil edilir. Bu yapılar, toplumsal etkileşim, bilgi akışı ve enerji dağıtımı gibi hayati işlevlerin analizi ve sürdürülebilirliği açısından kritik bir rol oynar. Ağ analizi, bu karmaşık yapıları anlamayı ve optimize etmeyi amaçlar. Ağ analizi; sosyal ağlardaki bilgi yayılımı, biyolojik ağlardaki hücreler arası etkileşimler veya ulaşım ağlarındaki trafiğin düzenlenmesi gibi pek çok farklı alanda uygulanabilir. Özellikle büyük veri çağında, ağ analizleri hem verimliliği artırmak hem de olası riskleri önceden tespit etmek için temel bir araçtır. Bu nedenle, ağların yapısal özelliklerini anlamak hem günlük yaşamda hem de bilimsel çalışmalarda stratejik bir öneme sahiptir.

Ağ analizinde, ağlardaki kritik düğümlerin tespiti en önemli araştırma konularından biridir. Bu kritik düğümler, ağın genel yapısının bozulmasına neden olabilecek zayıf noktaları ifade eder ve bu düğümlerin tespiti, ağların dayanıklılığını artırmak, zayıf noktaları belirlemek ve verimliliği optimize etmek açısından önemli bir rol oynar. Kritik Düğüm Problemi (KDP) (Critical Node Problem) ağın birleştirilmişliğini önemli ölçüde etkileyen düğümleri tespit etmeyi amaçlayan bir optimizasyon problemidir. Bu problemde, bir k tamsayısına eşit ya da daha az sayıda düğüm silinerek kalan çizgedeki ikili bağlantılılık sayısının (en az bir yol ile bağlı olan düğüm çiftlerinin toplam sayısını) enküçüklenmesi hedeflenmektedir (Lalou vd., 2018).

KDP'nin de dahil olduğu Kritik Düğüm Tespiti Problemleri (KDTP), ağların özelliklerini, yapılarını ve işlevlerini analiz etmek ve anlamak için çok önemli bir araçtır. KDTP aslen teorik bir çizge problemi olarak önerilmesine ve bu doğrultuda birçok çalışma olmasına karşın, problem aynı zamanda birçok gerçek hayat probleminin çözümünde de önemli bir rol oynamaktadır. KDTP'nin en doğal uygulama alanı ağların dayanıklılıklarının analiz edilmesidir. Güçsüz bir ağ, rastgele yıkıcı olaylar karşısında savunmasız olabilir. Genel olarak bir ağın zayıflıkları, bozulduğunda ağ performansını önemli ölçüde düşüren düğümler ve kenarları olarak tanımlanır. Bu nedenle KDTP ile kritik düğümlerin bulunması ağdaki temel zayıflıkları keşfetmeye olanak sağlar. Farklı çalışmalar kritik düğümlerin merkez düğümler (Grubesic vd., 2008) gibi literatürde yaygın olarak kullanılan diğer metriklere göre ağ zafiyetini değerlendirme açısından daha uygun olduğunu göstermiştir (Borgatti, 2006; Dinh & Thai, 2011; Dinh vd., 2010; Veremyev vd., 2015).

KDTP'nin yaygın kullanım alanlarından biri de hesaplamalı biyolojidir. Bakteriler ve virüsler gibi biyolojik organizmalar, birbirleriyle etkileşime giren protein kümeleri oluşturarak bir protein ağı meydana getirir. KDTP, proteinlerin düğümler ile, aralarındaki etkileşimin ise kenarlar ile temsil edildiği bir çizge ile modellenebilir. Tüm proteinler arasındaki bağlantıyı koruyan kritik düğümlerin (proteinleri) tespiti birçok biyolojik uygulama için faydalı bilgiler sağlar. Örneğin akılcı ilaç tasarımında bu kritik proteinler ilgili zararlı organizmayı yok etmek ve etkisiz hale getirmek için hedef alınmalıdır (Jhoti & Leach, 2007; Liljefors vd., 2002). Tomaino vd., bir KDTP varyantı ile insan-protein etkileşim ağındaki kritik düğümleri belirleyerek saldırgan kanser hücrelerini yok etmek üzerine çalışmıştır (Tomaino vd., 2012).

KDTP'nin kullanımı için belki de en uygun alan telekomünikasyon ağlarıdır. Telekomünikasyon ağlarında kritik düğümlerin belirlenmesi hem savunma hem de saldırı yaklaşımları için kullanılabilir. Terörist iletişim ağları gibi muhalif ağlarda bireylerin ya da grupların arasındaki iletişimi koparmak için en az saldırı ile muhalif ağdaki iletişimin en aza indirilmesi için kritik düğümlerin tespit edilmesi gerekir (Krebs, 2002; Ovelgonne vd., 2012). Arulselvan vd., Krebs tarafından modellen terörist ağı üzerindeki iletişimi yok etmek için KDTP'yi kullanmışlardır. Ayrıca Commander vd., Kablosuz Ağ Bloke Etme Problemini (Wireless Network Jamming Problem) bir KDTP olarak formüle etmiştir (Commander vd., 2007). Kablosuz Ağ Bloke Etme Problemi, muhalif bir kablosuz iletişim ağını etkisiz hale getirmek için en uygun sayıda sinyal bozucuyu ve konumu belirlemeyi amaçlamaktadır. KDTP'nin askeri alandaki bir diğer uygulaması da tedarik zinciri ağlarıdır. Askeri bir tedarik zincir ağı, taburları düğüm ve aralarındaki bağlantıları kenar olarak düşünülerek bir çizge olarak modellenebilir. Askeri bir tedarik zinciri ağındaki kritik düğümler belirlenip bu düğümlere saldırılarak taburlar arasındaki paylaşım tahrip edilebilir.

KDTP'nin en önemli uygulama alanlarından biri de sosyal ağlardır. Sosyal ağlar, insanlar ve aralarındaki etkileşimlerin modellenmesiyle oluşturulduğundan hem bu ağların yapısının hem de üzerlerinde gerçekleşen

olayların incelenmesi önemlidir. Ağdaki toplulukların ve önemli bireylerin tespiti ağın yapısal özelliklerine; bilgi paylaşımı, ürün önerileri ve enfeksiyon yayılımı gibi fenomenler de ağ üzerinde gerçekleşen olaylara örnek olarak verilebilir. KDTP, daha yüksek ikili bağlantılılığın daha hızlı salgına neden olduğu varsayılarak, sosyal bir ağdaki kritik kişileri bulmak ve toplumdaki insanlar arasındaki ikili bağlantıları en aza indirgemek üzere onları aşılamak için kullanılabilir. Sosyal ağlarda kritik düğümleri/kişileri belirlemek, Covid-19 ve benzeri salgınlara karşı geliştirilecek stratejiler için büyük önem taşımaktadır. Geçen yıllarda Nature dergisinde yayımlanan bir çalışmada Block vd. Covid-19 virüsünün yayılımını sosyal ağlar üzerinde modelleyerek insanların sosyalleşme örüntülerinin sonuçlarını incelemiştir (Block vd., 2020).

KDTP ayrıca Kablosuz Çok Sekmeli Ağlar (KÇSA) ile modellenen, nesnelerin interneti, telsiz duyarga ağları ve drone ağları gibi birçok ağ türünde uygulamalara sahiptir. KÇSA'da önemli konulardan biri ağın güvenilirliği ve hata toleransıdır. Güvenilirlik ve hata toleransında önemli etkenlerden biri aktif düğümler arasında sürdürülebilir bağlantının sağlanmasıdır (Akram & Ugurlu, 2023). KÇSA'da bir merkezi iletişim alt yapısı olmadığından dolayı uzak düğümlerin birbirine bağlanması için aradaki diğer düğümlerden yararlanmak gerekmektedir. KÇSA'da kritik düğümlerin tespiti, sonrasında bu düğümlerin yedek düğümler ya da ekstra güç kaynakları ile güçlendirilmesi ağdaki dayanıklılığın minimum maliyet ile giderilmesini sağlar (Dagdeviren vd., 2018). Ayrıca kritik düğümler, ağdaki mevcut iletişim yollarının yükünün artmasına ve paketlerin ulaşım hızının düşmesine sebep olmaktadır. Bu da üst seviye uygulamaların verimliliğini düşürmektedir. Bu sebeple kritik düğümlerin tespiti, ağın yönlendirme performansını düşüren önemli noktaları belirler. Böylelikle bu ağlarda paket yönlendirmelerin (ING Routing) verimliliğini artırabilir. Kritik düğümlerin tespiti, ağın yaşam süresini etkileyen önemli düğümleri de belirleyebilir. Genel olarak, kritik düğümler ağın farklı bölgelerini birbirine bağlayan ara düğümler olduğundan, KÇSA'da paketlerin çoğu kritik düğümlerin üzerinden geçmektedir. Bu durum kritik düğümlerin üzerinde yoğun veri akışına ve bu düğümlerde hızlı enerji tüketimine sebep olur. Dolayısıyla, kritik düğümlerin tespiti ağın yaşama süresinde etkili olan en önemli düğümleri belirleyebilir (Cobanlar vd., 2022).

KDP, NP-Zor bir problem olduğundan problem için özellikle büyük ölçekli ağlarda etkin çözümlerin bulunması için polinom zamanlı algoritmalara ihtiyaç duyulmaktadır. Bununla birlikte, sosyal ağlar ve biyolojik ağlar gibi karmaşık büyük ölçekli sistemlerin analizinde, hızlı ve ölçeklenebilir yöntemlere duyulan ihtiyaç artmıştır. Bu çalışmanın ana motivasyonu, KDP için özellikle sosyal ağlar gibi büyük ölçekli ağlarda etkili bir şekilde çalışabilecek, polinom zamanlı bir algoritma geliştirmektir. Geliştirilen algoritma 3 temel aşamadan oluşmaktadır. Önerilen algoritmanın ilk aşamasında yüksek dereceye sahip düğümler ağdan çıkarılır, ikinci aşamasında ise bir Derinlik Öncelikli Arama ağacı oluşturulur ve kesim düğümleri belirlenir. Son olarak, üçüncü aşamada kritik düğümler tespit edilir.

Çalışmanın geri kalanındaki bölümler ise şu şekilde organize edilmiştir; ikinci bölümde, ele alınan problemin formal tanımı ve çalışma ile ilgili bazı ön bilgiler verilmiştir. Üçüncü bölüm kısa bir literatür incelemesi sunmaktadır. Dördüncü bölüm, önerilen algoritma ve karmaşıklık analizini içermektedir. Beşinci bölümde hesapsal sonuçlar verilmiştir. Son bölüm ise çalışmayla ilgili genel değerlendirmelerden ve gelecekte yapılacak çalışmalar için önerilerden oluşmaktadır.

2. Kritik düğüm problemi
2. Critical node problem

Çeşitli fenomenler ve bu fenomenler arasındaki etkileşimler çizgeler (graf) kullanılarak modellenebilir. Çizge teorisi, bu tür yapıları inceleyen ve bilgisayar bilimlerinde geniş bir uygulama alanına sahip önemli bir çalışma alanıdır. Bir $G = (V, E)$ çizgesi tanımlandığında, V düğüm (nodes/vertices) kümesini, E ise düğümler arasındaki kenarları/ayrıtları (edges/links) temsil eder. Burada $n = |V|$ çizgedeki toplam düğüm sayısını, $m = |E|$ ise çizgedeki kenar sayısını ifade eder. Bir düğümün derecesi (degree), o düğüme bağlı kenarların sayısıdır. Kenarların yalnızca düğümler arasındaki ilişkinin var olup olmadığını belirttiği çizgelere ağırlıksız çizge (unweighted graph) denir. Buna karşılık, kenarların belirli bir sayısal değerle ifade edildiği çizgelere ise ağırlıklı çizge (weighted graph) denir (Diestel, 2012). Bir çizgede, düğümler arasında yalnızca bir kez geçiş yapılan bağlantı kümesine yol (path) denir. Eğer bir çizgede herhangi iki düğüm arasında en az bir yol varsa bu çizgeye bağlı/bağlantılı çizge (connected graph) denir. Başlangıç ve bitiş düğümlerinin aynı olduğu yollara ise çevrim (cycle) adı verilir. Eğer bir çizge çevrim içermiyorsa ve iki düğüm arasında sadece tek bir yol varsa bu çizgeye ağaç (tree) denir. Bir çizgede bağlantılı bileşen (connected component), her iki düğümün de birbirine en az bir yol ile bağlı olduğu en büyük düğüm alt kümesidir. Başka bir deyişle, bağlantılı bileşen,

düğümlerinin birbirine dolaylı ya da doğrudan bağlı olduğu bir düğüm grubudur. Bir çizgede kesim düğümü (articulation point/cut node), çıkarıldığında çizgenin bağlantılı bileşenlerinin sayısını artıran bir düğümdür. Başka bir deyişle, bu düğümün çıkarılması, çizgeyi birden fazla alt bileşene böler. Derinlik Öncelikli Arama (DFS – Depth-First Search) ise çizge arama algoritmalarından biridir. DFS, bir düğümden başlayarak o düğüme bağlı olan tüm düğümleri derinlemesine ziyaret eder ve ardından geriye dönerek diğer dallara geçer. DFS sırasında, çizge üzerinde ziyaret edilen düğümler ve kenarlar bir DFS ağacı (DFS tree) oluşturur. DFS ağacı, bir kök düğümden başlayarak tüm bağlantılı düğümleri keşfetmek için oluşturulan ve çizgenin belirli bir sıralama ile temsil edildiği bir yapıdır. Tablo 1, makale boyunca kullanılan notasyonları özetlemektedir.

Tablo 1. Notasyon listesi
Table 1. *The list of notation*

Notasyon	***Anlamı***
G	Çizge
V	Düğüm kümesi
E	Kenar kümesi
n	Düğüm sayısı
m	Kenar sayısı
k	Kritik düğüm sayısı
Δ	Maksimum düğüm derecesi

Bir $G = (V, E)$ çizgesi ve bir k tamsayısı verildiğinde, KDP, silinmesi ile kalan $G(V \backslash S)$ çizgesinde düğüm ikilileri arasındaki bağlantıyı (pairwise connectivity) en aza indiren ve en fazla k düğümden oluşan bir $S \subseteq V$ kümesi bulmaya çalışır (Arulselvan vd., 2009). KDP'nin NP-Zor bir problem olduğu Arulselvan vd. (2009) tarafından gösterilmiştir. Problemin Veremyev vd. tarafından önerilmiş matematiksel modeli aşağıda verilmiştir (Veremyev vd., 2014a). İlgili modelde n toplam düğüm sayısını, $A = a_{ij}$ bitişiklik matrisini (eğer i ve j düğümleri arasında bir kenar varsa $a_{ij} = 1$, aksi halde $a_{ij} = 0$'dır) temsil eder. Her bir düğüm ikilisi için, eğer i ve j düğümleri arasında bir yol varsa $u_{ij} = 1$, aksi halde $u_{ij} = 0$'dır. Son olarak, eğer i düğümü kritik düğüm olarak silindiyse $v_i = 1$, aksi halde $v_i = 0$'dır.

$$min \sum_{i \in V} u_{ij} \tag{1}$$

k.s.

$$u_{ij} \leq 1 - v_i \qquad \forall i, j \in V,\ i \neq j, \tag{2}$$

$$u_{ij} \leq 1 - v_j \qquad \forall i, j \in V,\ i \neq j, \tag{3}$$

$$u_{ij} \leq \sum_{l \in V} a_{il} u_{lj} \qquad \forall i, j \in V,\ l \neq i, \tag{4}$$

$$u_{ij} \geq \frac{1}{n} \sum_{l \in V} a_{il} u_{lj} - v_i \qquad \forall i, j \in V, \tag{5}$$

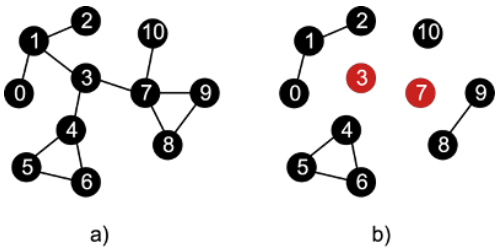
$$u_{ij} \geq a_{ij} - v_i - v_j \qquad \forall i, j \in V,\ i \neq j, \tag{6}$$

$$u_{ij} = u_{ji} \qquad \forall i, j \in V,\ i < j, \tag{7}$$

$$\sum_{i \in V} v_i \leq k, \tag{8}$$

$$v_i, u_{ij} \in \{0,1\} \qquad \forall i \in V. \tag{9}$$

Yukarıdaki modelde amaç fonksiyonu (1), birbirine en az bir yol ile bağlı düğümlerin sayısını minimize eder. i ve j düğümlerinin arasında bir kenarın olmadığı ve bu düğümlerin kritik düğüm olarak seçilmediği varsayıldığında, eğer i ve j arasında bir yol varsa, i düğümüne bağlı bir l düğümü vardır. O halde, l düğümü j düğümüne bir yol ile bağlıdır. u_{ij} ve v_i değişkenleri arasındaki ilişki (2)-(6) kısıtları ile tanımlanmıştır. Kısıt (7) eğer i ve j arasında bir yol varsa, j ve i arasında da bir yol olmasını sağlar. Son olarak, kısıt (8) toplam silinen kritik düğümlerin sayısının k'dan küçük eşit olmasını sağlar.



Şekil 1. Örnek çizge ve $k = 2$ için KDP'nin en kritik düğümleri
Figure 1. *The most critical nodes of the CNP for a sample graph with* $k = 2$

Şekil 1a'da 11 düğümden oluşan örnek bir çizge verilmiştir. Bu örnek çizge üzerinde $k = 2$ için KDP çözülmek istediğinde en iyi çözüme ait S kümesi düğüm 3 ve 7'den oluşmaktadır (Şekil 1b). Bu düğümlerin yokluğunda ağ 4 parçaya bölünür ve kalan ağdaki ikili bağlantılılık değeri 7'ye eşit olur.

3. Literatür incelemesi
3. Literature review

KDTP silinmesiyle kalan çizgede önceden tanımlı bir birleştirilmişlik metriğini minimize eden ya da maksimize eden düğüm kümelerini bulmayı amaçlayan bir problem ailesidir. Literatürde en çok çalışılan birleştirilmişlik metrikleri arasında Bağlantılı Bileşen Sayısı, En Büyük Bağlantılı Bileşen Boyutu ve İkili Bağlantılılık (Pairwise Connectivity) sayısı yer almaktadır. Bu metrikleri ele alan tüm KDTP varyantları, literatürde çokça çalışılan bir problem olan Düğüm-Silme Problemlerinin (Node-Deletion Problems) alt problemleri olarak değerlendirilebilir (Lewis & Yannakis, 1980; Yannakis, 1981). Bu metrikler, KDTP kapsamında ilk olarak 2009 yılında Arulselvan vd. tarafından tanımlanmış ve önerilmiştir (Arulselvan vd., 2009). KDP, basit bir ifadeyle, düğümlerinden bir kısmını silerek ağı benzer boyutlarda çok sayıda parçaya bölmeye çalışır. Dolayısıyla KDP kalan ağdaki hem bağlantılı bileşen sayısını maksimize etmeye hem de en büyük bileşenin boyutunu minimize etmeye çalışan çok amaçlı bir problem olarak görülebilir. KDP kritik düğüm problemleri arasında en çok çalışılan problemdir.

Optimizasyon problemlerinin kesin çözümlerini bulmanın etkili yollarından biri tamsayılı programlamadır. KDP için tamsayılı doğrusal programlama kullanılarak geliştirilen ilk matematiksel formülasyon Arulselvan vd. tarafından önerilmiştir (Arulselvan vd., 2009). Di Summa vd., önerilen ilk formülasyon üzerine çalışmış ve çeşitli iyileştirme önerilerinde bulunmuştur (Di Summa vd., 2012). Bu iki model çok sayıda kısıt kullandığı ($O(n^3)$ kısıt karmaşıklığı) için sadece küçük boyutlu çizgelerde çalışabilmektedir. $O(n^2)$ kısıt karmaşıklığına sahip bir diğer geliştirilmiş matematiksel formülasyon Veremyev vd. tarafından önerilmiştir (Veremyev vd., 2014a). Bu formülasyon 1000 düğüme kadar olan çizgeler için kesin çözümler bulabilmektedir. Bu da araştırmacıların geliştirdikleri sezgisel algoritmaları, en iyi çözümü bilinen örnekler üzerinde test etmelerine imkân sağlamaktadır. Ayrıca KDP için Ventresca ve Aleman, Arulselvan vd. tarafından önerilen formülasyonu baz alarak iki farklı yaklaşım algoritması önermiştir (Ventresca & Aleman, 2014a; Ventresca & Aleman, 2014c; Arulselvan vd., 2009).

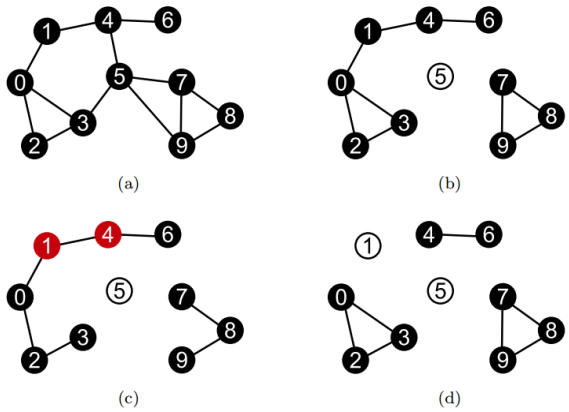
Optimizasyon problemlerinin çözümünde kullanılan bir diğer yaklaşım Metasezgisel Algoritmalardır (Metaheuristic Algorithms). KDP için de literatürde önerilmiş çok sayıda metasezgisel algoritmalar bulunmaktadır. Ventresca (2012) büyük boyutlu örnekler üzerinde Tavlama Benzetimi (Simulated Annealing) ve Popülasyon Tabanlı Artan Öğrenme (Population-Based Incremental Learning) yöntemlerinin KDP üzerinde performansını araştırmıştır (Ventresca, 2012). Aringhieri vd. farklı yerel arama yöntemleri içeren iki farklı metasezgisel algoritma geliştirmiştir (Aringhieri vd., 2016a). Aynı araştırmacılar KDP için bir genetik algoritma yaklaşımı önermiştir (Aringhieri vd., 2016b). Bu çalışmalara ek olarak, Açgözlü Rassallaştırılmış Uyarlamalı bir Arama Yordamı GRASP (Greedy Randomized Adaptive Search Procedure) Purevsuren vd. tarafından önerilmiştir (Purevsuren vd., 2016). Son olarak, Zhou vd. KDP'nin çözümü için bir Memetik Algoritma (Memetic Algorithm) kullanmıştır (Zhou vd., 2019).

KDP, NP-Zor bir problem olduğundan problem için makul sürelerde kabul edilebilir sonuçlar bulmak oldukça önemlidir. Bu amaçla en çok kullanılan yöntemler polinom zamanlı algoritmalardır. Bu kapsamda önerilen ilk algoritma Arulselvan vd. tarafından geliştirilmiştir (Arulselvan vd., 2009). Önerilen algoritmada öncelikle

verilen çizgedeki en büyük bağımsız küme bulunur ve bu kümede olmayan tüm düğümler başlangıç çözümü olarak değerlendirilir. Bu başlangıç çözümü aynı zamanda G çizgesi için bir tepe örtüsü olma özelliğini taşır. Daha sonra çözüm kümesinde düğüm sayısı k'ya eşit olana kadar, çözüm kümesindeki düğümler ilgili kümeden çıkarılır. Önerilen bu yöntemin zaman karmaşıklığı $O(n^2m)$'dir. Bu algoritmanın en büyük dezavantajı en iyi çözümde olması gereken bir v düğümü başlangıç çözümüne giremediğinde, daha sonra çözüme girmesinin bir yolu yoktur. Bu sebeple, bu yöntemin çözümlerini iyileştirmek için çözüm kümesinde olan ve olmayan birer düğümün yer değiştirdiği 2-değişimli (2-exchange) bir yerel arama yöntemi önermiştir. Bu yöntem $O(n^3)$ zaman karmaşıklığı gerektirmektedir. Yerel bilgileri kullanan DFS algoritmasına dayalı daha hızlı algoritmalar Edalatmanesh (2013) tarafından önerilmiştir (Edalatmanesh, 2013). Önerilen yöntemler arasında literatür örnekleri üzerinde en iyi performans gösteren algoritma, DFS tarafından belirlenen sıralamaya dayanan DFSH − Post algoritmasıdır. DFSH − Post algoritmasının zaman karmaşıklığı $O(k(nm + nlogn))$'dir. Ventresca ve Aleman, modifiye edilmiş DFS'ye dayanan etkili bir algoritma önermiş ve sonuçlarını KDP için mevcut kütüphane örnekleri üzerinde raporlamışlardır. Önerilen algoritmanın zaman karmaşıklığı $O(k(n + m))$'dir. Bu çalışmalara ek olarak, Ugurlu, KDP'yi kablosuz çok sekmeli ağlar üzerinde çalışmış ve problem için bir dağıtık algoritma önermiştir (Ugurlu, 2023).

4. Önerilen algoritma
4. Proposed algorithm

KDP üzerine gerçekleştirilen çalışmalar ve elde edilen kapsamlı sonuçlar incelendiğinde, çizgedeki en yüksek dereceye sahip düğümlerin genellikle en iyi çözüm kümelerinde yer aldığı, ancak kritik düğümler söz konusu olduğunda sadece derece kısıtının dikkate alınmasının çözüm kalitesini düşürdüğü gözlemlenmiştir. Bu sonuçlar ışığında, çizgedeki en yüksek dereceye sahip bir çekirdek kümenin çizgeden çıkarılmasının KDP için kaliteli çözümler üretebileceği fikrine ulaşılmıştır. Bu düşünceye dayanarak KDP için geliştirilen polinom zamanlı algoritmada (TrimCut), öncelikle yüksek bağlantıya sahip $k/2$ adet düğüm ağdan çıkarılır. Ağın bu şekilde seyreltilmesi, kalan ağdaki kritik düğümlerin çok daha etkin bir şekilde tespitine olanak sağlamaktadır. Ardından, kalan ağda DFS ağacı oluşturulur ve ağdaki kesim düğümleri tespit edilir. Sonrasında tespit edilen her bir kesim düğümünün KDP amaç fonksiyonuna sağlayacağı azalma miktarı hesaplanır ve bu fonksiyonda en çok azalmayı sağlayan düğüm ağdan çıkarılır. Seyreltilmiş ağdaki bu işlem $k/2$ kere tekrarlanarak, toplamda k adet kritik düğüm belirlenir.



Şekil 2. TrimCut algoritmasının temel adımları
Figure 2. *The basic steps of the TrimCut algorithm*

Şekil 2'de TrimCut algoritmasının temel adımları görselleştirilmiştir. Şekil 2a'daki örnek çizge üzerinde $k = 2$ için KDP çözülmek istenildiğinde, TrimCut öncelikle ağdaki en yüksek dereceye sahip bir $(2/2 = 1)$ düğümü kritik düğüm olarak belirler ve bu düğümü (düğüm 5) ağdan çıkarır (Şekil 2b). Daha sonra kalan seyreltilmiş ağda DFS ağaçları oluşturulur (kalan ağda birden fazla bağlantılı bileşen olduğu için TrimCut birden fazla DFS ağacı içeren bir orman oluşturmaktadır). Sonraki adımda DFS ağaçlarındaki kesim düğümleri (kırmızı düğümler) tespit edilir (Şekil 2c) ve bu düğümler arasından KDP'nin amaç fonksiyonunu en çok azaltan düğüm, kritik düğüm (düğüm 1) olarak belirlenerek ağdan çıkarılır. Büyük ve özellikle seyrek olmayan ağlarda kesim düğümleri bulunmayabilir. Bu durumda TrimCut algoritması DFS ağacında en fazla çocuğa sahip düğümü kritik düğüm olarak belirler. DFS ağacında en fazla çocuğa sahip düğüm, genellikle ağın bağlantılılığını en çok etkileyen düğüm olarak değerlendirilebilir. Örnek çizge için TrimCut algoritmasının

bulduğu kritik düğümler ve kalan ağın yapısı Şekil 2d'de görselleştirilmiştir. Şekil 2'deki örnek üzerinde de görüldüğü gibi TrimCut algoritması birbirini takip eden üç temel aşamadan oluşmaktadır. İlk aşamada ağdaki en yüksek bağlantıya sahip düğüm kritik düğüm olarak tespit edilir ve ağdan çıkarılır. Bu işlem $k/2$ kere tekrarlanarak ağ seyreltilir. Daha sonra ikinci aşamada seyreltilmiş ağda DFS ağacı/ağaçları ile kesim düğümleri belirlenir. Son aşamada ise oluşturulmuş ağaçtaki/ağaçlardaki en kritik düğüm belirlenir. Görüldüğü üzere ikinci ve üçüncü aşama birbirini takip eden bir süreçtir ve bu süreç $k/2$ kere tekrarlanarak ağdaki en kritik k adet düğüm tespit edilir.

Şekil 3'te TrimCut'ın ilk aşamasının sözde kodu verilmiştir. İlgili algoritmada öncelikle her bir düğümün derece (komşu/bağlantı) sayısı hesaplanır (Algoritma 1 – Adım 1-3). Daha sonrasında ağdaki en yüksek dereceye sahip düğüm tespit edilir ve bu düğüm kritik düğüm olarak kaydedilir (Algoritma 1 – Adım 5-11). Sonraki adımda belirlenen kritik düğümün tüm komşularının dereceleri 1 eksiltilir (Algoritma 1 – Adım 12-14). Son olarak belirlenen kritik düğüm ağdan çıkarılır ve silinen düğüm sayısı güncellenir (Algoritma 1 – Adım 15-16). Bu işlemler silinen düğüm sayısı, $k/2$'ye eşit olana kadar tekrarlanır (Algoritma 1 – Adım 5-17).

Algoritma 1: En Yüksek Bağlantıya Sahip Düğümlerin Tespiti Algoritması

Girdi: $\boldsymbol{G(V,E)}$, $\boldsymbol{k}$
Başlangıçta: (deg[]) ← 0, *silinen_dugum_sayısı* ← 0
1: **For** $\boldsymbol{G}$'deki her bir $\boldsymbol{v_i}$ düğümü için **do**
2: | deg[$\boldsymbol{v_i}$] = $\boldsymbol{v_i}$'nin komşu sayısı
3: **end for**
4: **while** (*silinen_dugum_sayısı* < $\boldsymbol{k}$/2) **do**
5: | max = 0
6: | **for** $\boldsymbol{G}$'deki her bir $\boldsymbol{v_i}$ düğümü için **do**
7: | | **if** (deg[$\boldsymbol{v_i}$] > max) **then**
8: | | max = deg[$\boldsymbol{v_i}$]
9: | | *kritik_dugum* = $\boldsymbol{v_i}$
10: | | **end if**
11: | **end for**
12: | **for** *kritik_dugum*'un her bir v_j komşusu için **do**
13: | | deg[v_j] = deg[v_j] – 1
14: | **end for**
15: | $\boldsymbol{G}$ = $\boldsymbol{G}$ \ {*kritik_dugum*}
16: | *silinen_dugum_sayısı* = *silinen_dugum_sayısı* + 1
17: **end while**

Şekil 3. TrimCut algoritması sözde kodu 1: En yüksek bağlantıya sahip düğümlerin tespiti.
Figure 3. *TrimCut algorithm Pseudocode 1: Detection of nodes with highest connections*

Şekil 4'te TrimCut algoritmasının 2. aşamasının sözde kodu verilmiştir. Algoritma 2, DFS ağaçlarının oluşumunu ve kesim düğümlerinin tespitini içermektedir. TrimCut algoritması DFS ağacı oluşturmak için yığın veri yapısını kullanmaktadır. Ağdaki en yüksek dereceye sahip $k/2$ adet kritik düğüm ağdan çıkarıldıktan sonra, boş bir yığın (S) ile başlanarak DFS ağacı oluşturulur. Öncelikle ağda henüz ziyaret edilmemiş bir v_i düğümü S'e eklenir ve ilgili düğümün $r[v_i]$, $b[v_i]$ ve $root[v_i]$ değerleri güncellenerek düğüm ziyaret edildi olarak işaretlenir (Algoritma 2 - Adım 2-3). Burada $r[v_i]$, v_i düğümünün DFS ağacındaki ziyaret sırasıdır. Yani, kök düğümün sırası 0'dır ve bir v_i düğümü t adet düğümden sonra ziyaret edildiyse sırası $r[v_i] = t$ olacaktır. $b[v_i]$ ise v_i düğümünün kendisi ya da çocuklarının komşuları (orijinal ağdaki) arasındaki en küçük ziyaret sırasıdır. Eğer bir v_i düğümünün $b[v_i]$ değeri, v_i düğümünün ebeveyninin (v_{parent_i}) $r[v_i]$ değerine eşit ise bu durumda, v_i düğümü ebeveyninden daha düşük bir ziyaret sırasına sahip bir düğüme ebeveyni üzerinden geçmeden ulaşamaz, dolayısıyla v_{parent_i} bir kesim düğümüdür. $root[v_i]$ değeri de, eğer v_i düğümü bir DFS ağacının kök düğümü ise 1, aksi halde 0'dır. Eğer bir düğüm kök düğümse ve birden fazla çocuğu varsa bu düğüm de bir kesim düğümüdür. DFS ağacının kökü için bir düğüm belirlenip ilgili değerleri güncellendikten sonra, S yığınından sıradaki düğüm çekilir (v_j) ve bu düğümün henüz ziyaret edilmemiş bir komşusu olup olmadığı kontrol edilir (Algoritma 2 - Adım 5-7). Eğer böyle bir komşu düğüm var ise (v_k), bu düğüm S yığınına eklenir ve $parent[v_k]$, $r[v_k]$ değerleri güncellenir (Algoritma 2 – Adım 8). $parent[v_k]$,

v_k düğümünün DFS ağacındaki ebeveynidir. Sonrasında $ch[v_j][]$ matrisi güncellenir ve v_k düğümü ziyaret edildi olarak işaretlenir (Algoritma 2 – Adım 9-10). $ch[v][]$ matrisi, v düğümünün çocuk bilgilerini saklamaktadır ($ch[v][0]$, v düğümünün çocuk sayısını, $ch[v][t]$ ise v düğümünün t. çocuğunu temsil etmektedir). Sonraki adımda v_k düğümünün komşularının $b[]$ değerleri güncellenir (Algoritma 2 – Adım 11-15). Eğer S yığınından çekilen v_j düğümünün ziyaret edilmemiş bir komşusu yoksa v_j düğümünün $b[v_j]$ değeri ve sonrasında $parent[v_j]$'nin $ns[parent[v_j]]$ ve $stree[parent[v_j]][]$ değerleri güncellenir ve v_j düğümü S yığınından çıkarılır (Algoritma 2 – Adım 17-24).

Algoritma 2: DFS Ağacı Oluşturulması ve Kesim Düğüm Tespiti Algoritması

Girdi: $\boldsymbol{G(V,E)}$, $\boldsymbol{k}$
Başlangıçta: ($\boldsymbol{ch}[][]$, $\boldsymbol{stree}[][]$)← 0, ($\boldsymbol{root}[]$, $\boldsymbol{parent}[]$, $\boldsymbol{r}[]$, $\boldsymbol{b}[]$, $\boldsymbol{ns}[]$) ← 0, $S \leftarrow \{\}$
while (ziyaret edilmemiş bir düğüm $\boldsymbol{v_i}$ mevcutsa) **do**
  push($\boldsymbol{v_i}$, $\boldsymbol{S}$); $\boldsymbol{r[v_i] = 1}$; $\boldsymbol{b[v_i] = 1}$; $\boldsymbol{root[v_i] = 1}$
  $\boldsymbol{v_i}$'yi ziyaret edildi olarak işaretle
  while ($\boldsymbol{S}$ boş değilse) **do**
    $\boldsymbol{v_j = peek(S)}$
    $\boldsymbol{v_k = v_j}$'nin ziyaret edilmemiş bir komşusu
    if ($\boldsymbol{v_k}$ mevcutsa) **then**
      push($\boldsymbol{v_k}$, S); $\boldsymbol{parent[v_k] = v_j}$; $\boldsymbol{r[v_k] = r[v_j] + 1}$
      $\boldsymbol{ch[v_j][0] + +}$; $\boldsymbol{ch[v_j][ch[v_j][0]] = v_k}$
      $\boldsymbol{v_k}$'yı ziyaret edildi olarak işaretle
      for $\boldsymbol{v_k}$'nın her bir $\boldsymbol{v_z}$ komşusu için **do**
        if ($\boldsymbol{b[v_z] > b[v_k]}$) **then**
          $\boldsymbol{b[v_z] = b[v_k]}$
        end if
      end for
    else
      if ($\boldsymbol{b[v_j] < b[parent[v_j]]}$) **then**
        $\boldsymbol{b[parent[v_j]] = b[v_j]}$
      end if
      $\boldsymbol{ns[parent[v_j]] = ns[parent[v_j]] + ns[v_j] + 1}$
      $\boldsymbol{stree[parent[v_j]][0] + +}$
      $\boldsymbol{stree[parent[v_j]][stree[parent[v_j]][0]] = ns[v_j] + 1}$
      pop($\boldsymbol{S}$)
    end if
  end while
end while
for $\boldsymbol{G}$'deki her bir $\boldsymbol{v_i}$ düğümü için **do**
  if ($\boldsymbol{root[v_i] == 0}$) **then**
    for $\boldsymbol{v_i}$'nin her bir $\boldsymbol{v_j}$ komşusu için **do**
      if ($\boldsymbol{b[v_j] == r[v_i]}$) **then**
        $\boldsymbol{v_i}$'yi kesim düğümü olarak işaretle
      end if
    end for
  else
    if ($\boldsymbol{ch[v_i] > 1}$) **then**
      $\boldsymbol{v_i}$'yi kesim düğümü olarak işaretle
    end if
  end if
end for

Şekil 4. TrimCut algoritması sözde kodu 2 : DFS ağacının oluşturulması ve kesim düğümlerinin tespiti
Figure 4. *TrimCut algorithm Pseudocode 2: Establishing the DFS tree and detection of cut nodes*

$ns[v]$ değeri v düğümünü kök kabul eden alt ağaçtaki düğümlerin (v'nin torunları+1) sayısıdır. $stree[v][]$ ise v düğümünün çocuklarını kök kabul eden alt ağaçlardaki düğümlerin sayılarını saklar ($stree[v][0]$, v düğümünün çocuk sayısını, $stree[v][t]$ ise v düğümünün t. çocuğunu kök kabul eden alt ağaçtaki düğüm sayını temsil etmektedir). Bu işlemler S yığını boş olana kadar devam eder (Algoritma 2 – Adım 4-25). Eğer S yığını boş ise ve G çizgesinde ziyaret edilmemiş bir düğüm var ise bu düğüm S yığınına eklenir ve yukarıda açıklanan işlemler tekrarlanır (Algoritma 2 – Adım 1-26). G çizgesindeki tüm düğümler ziyaret edildikten sonra kesim düğümlerinin tespiti işlemine geçilir. Eğer bir v_i düğümü herhangi bir DFS ağacının kök düğümü değilse ve bu düğümün bir (v_j) komşusunun $b[v_j]$ değeri $r[v_i]$ değerine eşitse v_i düğümü, kesim düğümü olarak işaretlenir (Algoritma 2 – Adım 28-33). Eğer bir v_i düğümü bir DFS ağacının kök düğümü ise ve bu düğümün birden çok çocuğu varsa v_i düğümü kesim düğümü olarak işaretlenir (Algoritma 2 – Adım 35-37). Bu işlemler çizgedeki tüm düğümler için tekrarlanır (Algoritma 2 – Adım 27-39).

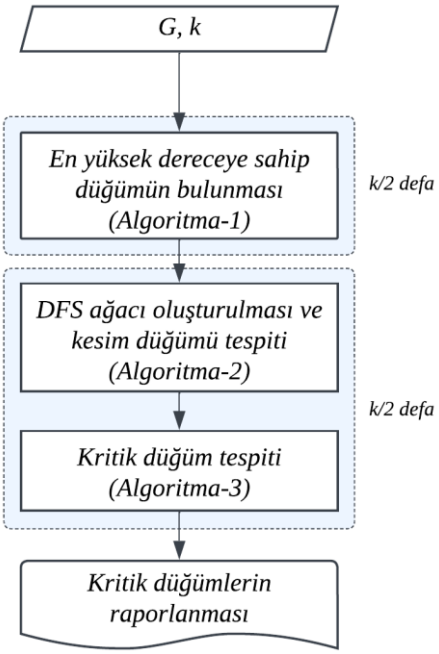
Algoritma 3: Kritik Düğüm Tespiti Algoritması

Girdi: $\boldsymbol{G(V,E)}$, $\boldsymbol{k}$, $\boldsymbol{parent}[]$, $\boldsymbol{root}[]$, $\boldsymbol{ch}[][]$, $\boldsymbol{stree}[][]$
Başlangıçta: max ← 0
if (herhangi bir kesim düğümü mevcutsa) **then**
  for her bir kesim düğümü $\boldsymbol{v_i}$ için **do**
    $\boldsymbol{p = v_i}$
    while ($\boldsymbol{parent[p] != 0}$) **do**
      $\boldsymbol{p = parent[p]}$
    end while
    for $\boldsymbol{v_i}$'nin her bir $\boldsymbol{v_j}$ komşusu için **do**
      if ($\boldsymbol{b[ch[v_i][v_j]] == r[v_i]}$) **then**
        value1 = *value1* + $\boldsymbol{(stree[v_i][v_j] * (stree[v_i][v_j] - 1)) / 2}$
        value2 = *value2* + $\boldsymbol{stree[v_i][v_j]}$
      end if
    end for
    value3 = $\boldsymbol{ns[p]}$ − *value2*
    value1 = *value1* + $\boldsymbol{(}$*value3* $\boldsymbol{* (}$*value3* $\boldsymbol{- 1)) / 2}$
    value4 = $\boldsymbol{(ns[p] * (ns[p] - 1)) / 2}$
    fark = *value4* − *value1*
    if (*fark* > max) **then**
      max = *fark*
      kritik_dugum = $\boldsymbol{v_i}$
    end if
  end for
else
  for her bir kesim düğümü olmayan düğüm $\boldsymbol{v_i}$ için **do**
    if ($\boldsymbol{ch[v_i][0]}$ > max) **then**
      max = $\boldsymbol{ch[v_i][0]}$
      kritik_dugum = $\boldsymbol{v_i}$
    end if
  end for
end if
$\boldsymbol{G = G \setminus \{}$*kritik_dugum*$\boldsymbol{\}}$

Şekil 5. TrimCut algoritması sözde kodu 3: Kritik Düğüm Tespiti
Figure 5. *TrimCut algorithm Pseudocode 3: Detection of critical node*

Şekil 5'te TrimCut algoritmasının 3. aşamasının sözde kodu verilmiştir. Algoritma 3 tespit edilen kesim düğümleri arasından, kritik düğümün belirlenme işlemlerini içermektedir. Eğer ağda kesim düğümü tespit edildiyse bu kesim düğümlerinin ağdan çıkarıldığında KDP'nin amaç fonksiyonuna getireceği azalma hesaplanır (Algoritma 3 – Adım 1-22). Bunun için öncelikle v_i düğümünün bağlı olduğu DFS ağacının kökü bulunur (Algoritma 2 – Adım 3-6). Sonrasında v_i düğümünün ağaçtan çıkarılması ile ağaçtan kopacak v_i düğümünün çocuklarını içeren bağlantılı bileşenlerin toplam boyutu (*value2*) ve bu bileşenlerin toplam KDP amaç fonksiyonu değeri (*value3*) hesaplanır (Algoritma 3 – Adım 7-12). Daha sonra ağaçtan kopan bağlantılı

bileşenlerin toplam boyutu ağacın boyutundan çıkarılarak kalan ağacın boyutu hesaplanır (*value3*) ve bu ağacın KDP amaç fonksiyonu değeri *value1*'e eklenir (Algoritma 3 – Adım 13-14). Dolayısıyla *value1*, v_i düğümünün ağaçtan çıkarıldığı durumdaki KDP amaç fonksiyonu değerine eşit olur. Sonraki adımda hiçbir düğümün ağaçtan çıkarılmadığı durumdaki *value4* (KDP amaç fonksiyonu) ve *value1* farkı ile düğümünün KDP amaç fonksiyonuna getireceği kazanç (*fark*) hesaplanır (Algoritma 3 – Adım 15-16). En büyük fark değerine sahip v_i düğümü kritik düğüm olarak seçilir. Bu işlemler tüm kesim düğümleri için tekrarlanır (Algoritma 3 – Adım 2-21). Eğer ağda kesim düğümü tespit edilmediyse DFS ağaçlarındaki en fazla çocuğu olan düğüm kritik düğüm olarak seçilir (Algoritma 3 – Adım 23-29). Son olarak seçilmiş olan kritik düğüm ağdan çıkarılır (Algoritma 3 – Adım 30). Algoritma 2 ve Algoritma 3 $k/2$ kere tekrarlanarak ağdaki en kritik k adet düğüm tespit edilir.



Şekil 6. Önerilen algoritmanın akış şeması
Figure 6. *The flowchart of the proposed algorithm*

Şekil 6'da TrimCut algoritmasının akış şeması verilmiştir. TrimCut algoritmasında ilk aşamada ağdaki en yüksek bağlantıya sahip düğüm kritik düğüm olarak tespit edilip, ağdan çıkarılır ve bu işlem $k/2$ kere tekrarlanarak ağ seyreltilir. Daha sonra ikinci aşamada seyreltilmiş ağda DFS ağacı/ağaçları ile kesim düğümleri belirlenir ve son aşamada ise oluşturulan ağaçtaki/ağaçlardaki en kritik düğüm belirlenir. İkinci ve üçüncü aşama birbirini takip eden bir birleşik bir süreçtir ve bu süreç $k/2$ kere tekrarlanarak ağdaki en kritik k adet düğüm tespit edilir.

4.1. Karmaşıklık analizi
4.1. Complexity analysis

TrimCut algoritmasının zaman karmaşıklığı aşağıdaki şekildedir.
Teorem 1: TrimCut algoritmasının zaman karmaşıklığı $O(kn^2)$'dir.
İspat: Algoritmanın ilk aşamasında ağdaki en yüksek dereceye sahip düğüm bulunmakta ve bu düğüm ağdan çıkarıldıktan sonra düğümün tüm komşularının dereceleri güncellenmektedir. Önerilen algoritmada her bir düğümün komşularının saklandığı bir veri yapısı kullanılmıştır. Dolayısıyla, seçilen bir düğümün komşularının taranması için $O(\Delta)$ zaman gerekmektedir. En yüksek dereceye sahip düğümün tespiti için ise $O(n)$ zaman gerekmektedir. Dolayısıyla, en yüksek dereceye sahip düğümün tespit edilmesi ve komşu düğümlerinin derecelerinin güncellenmesi için $O(n + \Delta)$ zaman gerekmektedir. Bu adım $k/2$ kere tekrarlanacağından algoritmanın ilk aşaması toplamda $O((k/2)\, x\, (n + \Delta)) = O(kn)$ zaman gerektirmektedir. Algoritmanın ikinci aşamasında DFS ağacı oluşturulur ve kesim düğümleri belirlenir. Bu işlem $O(n + m)$ (DFS ağacının oluşturulması) zaman gerektirir. Algoritmanın son aşamasında ise tüm düğümler taranarak ağdan çıkarılacak düğüm belirlenir. Bu yöntem $O(n^2)$ zaman gerektirir. DFS ağacı oluşturma ve kritik düğümlerin tespiti süreci

$(O(n + m) + O(n^2))$ $k/2$ kere tekrarlanacağından toplamda $O((k/2)\, x\, (n + m + n^2)) = O(kn^2)$ zaman gerekmektedir. Dolayısıyla, algoritmanın tüm aşamaları dikkate alındığında TrimCut algoritmasının zaman karmaşıklığı $O(kn) + O(kn^2) = O(kn^2)$'dir.

5. Hesapsal sonuçlar
5. Computational results

Önerilen algoritmanın performans analizi için 2 farklı veri seti kullanılmıştır. İlk veri seti küçük ölçekli rastgele çizgelerden, ikinci veri seti ise KDP için literatürde kullanılan çizgelerden oluşmaktadır. TrimCut algoritmasının örnekler üzerindeki hata oranları eşitlik 10'daki formül ile hesaplanmıştır.

$$Hata\ Oranı = \left|\frac{bulunan\_çözüm - optimal\_çözüm}{optimal\_çözüm}\right| * 100 \quad (10)$$

İlgili formülde *optimal_çözüm* KDP'nin optimal çözümünün amaç fonksiyonun değeridir. *Bulunan_çözüm* ise TrimCut algoritması tarafından tespit edilen çözümün amaç fonksiyonunun değeridir. Önerilen algoritma C dilinde kodlanmış ve hesapsal sonuçlar Intel Xeon 6 (CPU 3.3 GHz) işlemci ve 32 RAM'e özelliklere sahip bir bilgisayar üzerinde alınmıştır. Önerilen algoritmanın C dili ile yazılmış kaynak koduna https://github.com/Yesim7/TrimCut-Algorithm adresinden erişilebilir.

5.1. Küçük ölçekli çizgeler üzerindeki sonuçlar
5.1. Results on small-scale graphs

Küçük boyutlu çizgelerin KDP için optimal sonuçları doğrusal programlama ile (Veremyev vd. tarafından önerilen formülizasyon kullanılarak) bulunmuştur (Veremyev vd., 2014b). Oluşturulan veri setinde düğüm sayıları 30, 40 ve 50 arasında değişmektedir. Her bir düğüm sayısı için çizgenin yoğunluk (ağdaki mevcut ayrıt sayısının ağda olabilecek tüm maksimum ayrıt sayısına oranı) değeri 0.1, 0.15, 0.2, 0.25 ve 0.3 (ilgili yoğunluk oranları hem ağların bağlantılılığı hem de gerçek hayat örnekleri dikkate alınarak belirlenmiştir) olmak üzere 5 farklı örnek üretilmiştir.

Tablo 2. $k = 0.05n$ ve $k = 0.1n$ için KDP optimal değerleri ve TrimCut algoritmasının tarafından bulunan çözümlerin KDP amaç fonksiyonu değerleri
Table 2. *The CNP optimal values and the objective function values of the solutions found by the TrimCut algorithm for* $k = 0.05n$ ve $k = 0.1n$

n	***d***	***Optimal***		***TrimCut***		***Hata Oranı***	
		k=0.05n	***k=0.1n***	***k=0.05n***	***k=0.1n***	***k=0.05n***	***k=0.1n***
30	0.1	207	180	207	180	0	0
	0.15	325	300	325	300	0	0
	0.2	351	325	351	325	0	0
	0.25	351	325	351	325	0	0
	0.3	351	325	351	325	0	0
40	0.1	631	529	666	529	0.0555	0
	0.15	666	561	666	561	0	0
	0.2	666	561	666	561	0	0
	0.25	666	595	666	595	0	0
	0.3	666	595	666	595	0	0
50	0.1	990	861	990	903	0	0.049
	0.15	1035	903	1035	946	0	0.048
	0.2	1035	946	1035	946	0	0
	0.25	1035	946	1035	946	0	0
	0.3	1035	946	1035	946	0	0

Tablo 2'de oluşturulan orta boyutlu veri seti üzerinde sırasıyla düğüm sayısının %5 ($k = 2$) ve %10'luk ($k = 3$) k değerleri için KDP'nin optimal amaç fonksiyonu değerleri ve TrimCut algoritmasının sonuçları verilmiştir. Tablo 2 incelendiğinde TrimCut algoritmasının sadece 1 örnekte optimal çözümü bulamadığı görülmektedir. %5'lik k değeri için TrimCut algoritmasının ortalama hata oranı 0.004'tür. Tablo 2

incelendiğinde TrimCut algoritmasının sadece 2 örnekte optimal çözümü bulamadığı görülmektedir. %10'luk k değeri için TrimCut algoritmasının ortalama hata oranı 0.006'dır.

5.2. Kütüphane örnekleri üzerindeki sonuçlar
5.2. Results on library samples

Tablo 3'te optimal sonuçları bilinen literatürdeki kütüphane örnekleri üzerinde TrimCut algoritmasının sonuçları verilmiştir. İlgili kütüphane, her biri 4 adet örnek içeren 4 farklı çizge ailesinden oluşmaktadır. Barabasi–Albert (BA) çizgeleri ölçekten bağımsız (scale-free) ağlardır. Forest-Fire (FF) çizgeleri ise bir yangının bir ormanda nasıl yayıldığını taklit ederek, BA çizgeleri gibi ölçekten bağımsız bir yapıya sahip ancak daha yoğun bir yapıdadır. Erdos–Renyi (ER) rastgele çizgelerdir. Son olarak, Watts–Strogatz (WS) küçük-dünya (small world) yapısını taklit edecek şekilde tasarlanmış, daha yoğun bir yapıya sahip çizgelerdir ve KDP için çözülmesi en zor çizge ailesi olarak bilinmektedir. Bu çizgelerden hiçbiri gerçek bir ağın tam bir kopyası olarak düşünülmez. Ancak, gerçek ağlar genellikle her bir ağ türüne ait özelliklerin bir karışımını sergiler (Aringhieri vd., 2016). TrimCut algoritmasının literatürdeki kütüphane örnekleri üzerindeki sonuçları, ilgili örnekler üzerinde sonuçlarını raporlayan mevcut polinom zamanlı algoritmalar (*DFSH-Post* (Edalatmanesh, 2013) *ve GreedyVA* (Ventresca & Aleman, 2015)) ile karşılaştırılmıştır.

Tablo 3. Optimal sonuçları bilinen literatürdeki kütüphane örnekleri üzerinde algoritmaların KDP amaç fonksiyonu değerlerinin karşılaştırılması
Table 3. *Comparison of the CNP objective function values of the algorithms on benchmark instances in the literature with known optimal results*

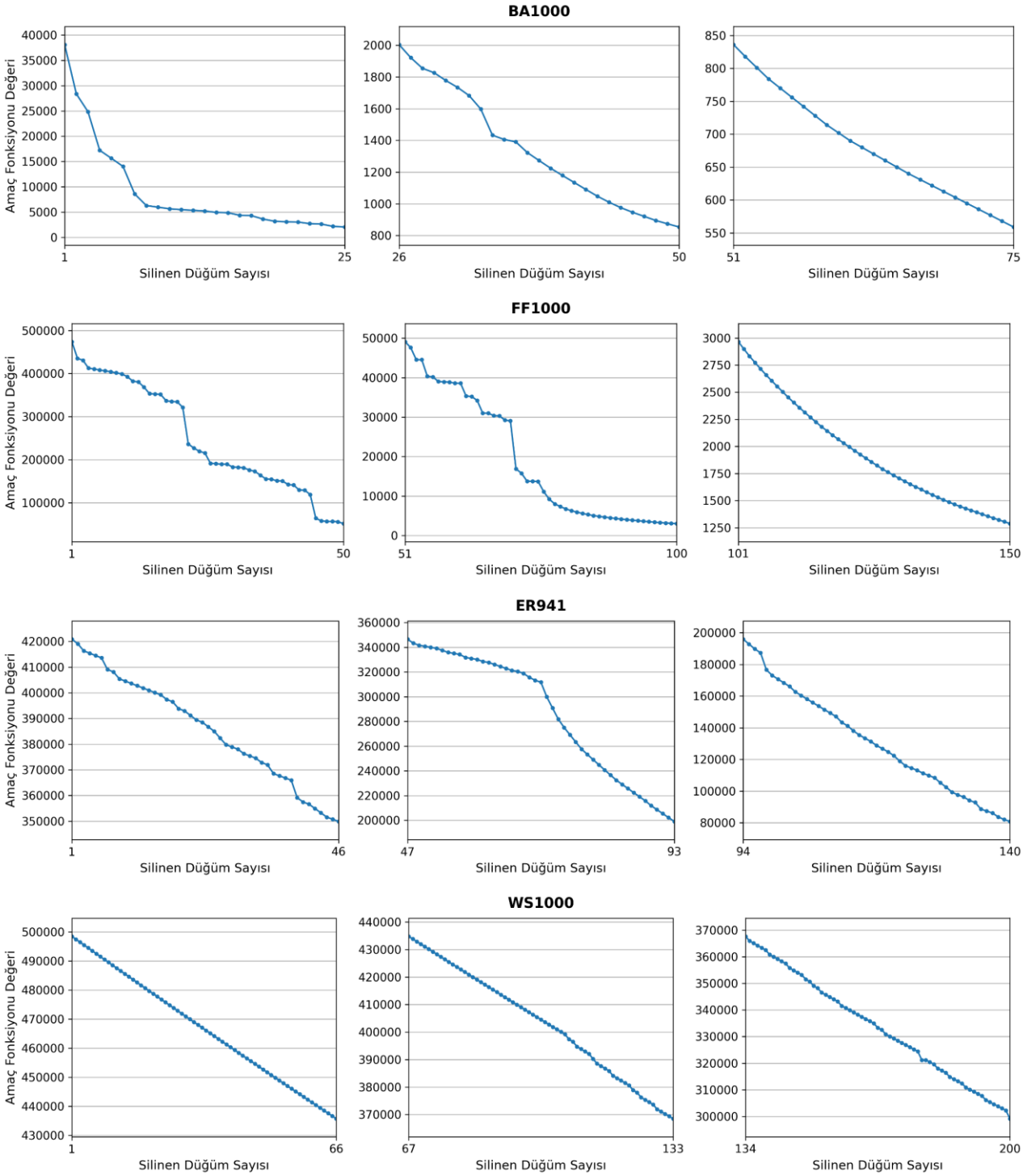
Instance	*n*	*k*	*Optimal*	*DFSH-Post*	*Hata Oranı*	*GreedyVA*	*Hata Oranı*	*TrimCut*	*Hata Oranı*
BA	500	50	195	203	0.041	199	0.021	199	0.021
BA	1000	75	558	580	0.039	559	0.002	559	0.002
BA	2500	100	3704	4292	0.159	3726	0.006	3722	0.005
BA	5000	150	10196	12273	0.204	10216	0.002	10199	0.001
FF	250	50	194	302	0.557	217	0.119	199	0.026
FF	500	110	257	344	0.339	293	0.140	269	0.047
FF	1000	150	1260	1880	0.492	1414	0.122	1288	0.022
FF	2000	200	4545	7432	0.635	5002	0.101	4793	0.055

Tablo 3 incelendiğinde BA çizge ailesi için tüm algoritmaların kaliteli sonuçlar ürettiği görülmektedir. Ancak FF çizge ailesi için DFSH-Post ve GreedyVA algoritmalarının çözümlerinin optimal çözümden uzaklaştığı görülmektedir. Buna karşın geliştirilen TrimCut algoritması FF çizge ailesi için de optimal çözüme yakın çözümler bulmaktadır. BA çizge ailesi için DFSH-Post, GreedyVA ve TrimCut algoritmalarının ortalama hata oranları sırasıyla 0.111, 0.008 ve 0.007'dir. FF çizge ailesi içinse DFSH-Post, GreedyVA ve TrimCut algoritmalarının ortalama hata oranları sırasıyla 0.506, 0.12 ve 0.037'dir. Tablo 3'teki sonuçlar incelendiğinde TrimCut algoritmasının optimal değeri bilinen kütüphane örnekler üzerindeki hata oranının %6'yı geçmediği görülmektedir.

Tablo 4. Optimal sonuçları bilinmeyen literatürdeki kütüphane örnekleri üzerinde algoritmaların KDP amaç fonksiyonu değerlerinin karşılaştırılması
Table 4. *Comparison of the CNP objective function values of the algorithms on benchmark instances in the literature with unknown optimal results*

Instance	*n*	*k*	*DFSH-Post*	*GreedyVA*	*TrimCut*
ER	235	50	1144	3011	362
ER	466	80	19952	28994	16213
ER	941	140	114166	116135	80834
ER	2344	200	1606656	1395584	1434375
WS	250	70	16110	16110	14878
WS	500	125	55163	69751	46589
WS	1000	200	319600	319600	299154
WS	1500	265	653015	761995	366231

Tablo 4'te optimal sonuçları bilinmeyen kütüphane örnekleri üzerinde DFSH-Post, GreedyVA ve TrimCut algoritmalarının sonuçları verilmiştir. İlgili tablo incelendiğinde TrimCut algoritmasının tüm örnekler için rakiplerinden çok daha kaliteli çözümler ürettiği görülmektedir. ER çizge ailesi için DFSH-Post, GreedyVA ve TrimCut algoritmalarının sonuçlarının ortalamaları sırasıyla 435478.75, 385931 ve 382946'dır. WS çizge ailesi içinse DFSH-Post, GreedyVA ve TrimCut algoritmalarının sonuçlarının ortalamaları sırasıyla 260972, 291864 ve 181713'tür.



Şekil 7. Önerilen algoritmanın amaç fonksiyonu değeri – silinen düğüm sayısı grafikleri.
Figure 7. *The objective function values vs the number of deleted nodes in the proposed algorithm.*

Şekil 7 her bir çizge ailesinin bir örneği üzerinde TrimCut algoritmasının silinen düğüm sayısına göre elde ettiği KDP amaç fonksiyonu değerlerini göstermektedir. BA1000 ve FF1000 ağlarından az sayıda düğümün çıkarılmasıyla amaç fonksiyon değerinde hızlı bir düşüş gözlemlenmektedir. Bu durum söz konusu ağların

nispeten daha az dirençli yapıya sahip olduğunu göstermektedir. ER941 ağında ise amaç fonksiyon değerindeki düşüşün kademeli olduğu görülürken, WS1000 ağında bu azalmanın, silinen düğüm sayısıyla doğrusal bir ilişki sergilediği söylenebilir. Bu sonuçlar farklı ağ tiplerinin topolojik özelliklerinin, ağların düğüm kayıplarına karşı dayanıklılıklarını etkilediğini göstermektedir.

Tüm sonuçlar incelendiğinde GreedyVA ve TrimCut algoritmalarının hata oranlarının DFSH-Post'tan daha iyi olduğu görülmektedir. Bununla birlikte, GreedyVA ve TrimCut algoritmalarının zaman karmaşıklıkları da DFSH-Post algoritmasının zaman karmaşıklığından daha düşüktür. Bu sebeple GreedyVA ve TrimCut algoritmalarının çalışma süreleri kıyaslanmış ve raporlanmıştır. Tablo 5 kütüphane örnekleri üzerinde GreedyVA ve TrimCut algoritmasının çalışma zamanlarını kıyaslamaktadır (GreedyVA algoritmasının kaynak kodu çalışmanın yazarlarından temin edilmiştir). Tablodaki sonuçlar incelendiğinde GreedyVA'nın TrimCut algoritmasına göre ortalama 1.3 kat daha fazla çalışma zamanına ihtiyaç duyduğu görülmektedir.

Çalışmada karşılaştırılan algoritmalar incelendiğinde, DFSH-Post'un zaman karmaşıklığı $O(nm + nlogn)$, GreedyVA'nın zaman karmaşıklığı $O(k(nm + nlogn))$ ve TrimCut algoritmasının zaman karmaşıklığı $O(k(nm + nlogn))$'dir. Bu 3 algoritma da temel DFS yöntemini kullanmakta, ancak düğümlerin kritikliklerinin belirlenmesi için farklı seçim mekanizmaları kullanmaktadır. DFSH-Post algoritmasında tek bir DFS ağacı oluşturulup, önerilen skorlama mekanizmaları ile k adet kritik düğüm tespit edilmektedir. GreedyVA algoritmasında ise DFS ağacı k kere oluşturulup, her bir ağaçtaki en yüksek skora sahip kesim düğümü ya da sıradan bir düğüm kritik düğüm olarak seçilir. GreedyVA algoritmasının en büyük dezavantajlardan biri kesim düğümü olmayan çizgelerde çözüm kalitesinin düşmesidir. Bu eksikliği gidermek adına önerilen algoritmada ilk olarak en yüksek dereceye sahip $k/2$ düğüm kritik düğüm olarak belirlenir ve çizgeden çıkarılır. Daha sonra seyreltilmiş çizgede kesim düğümleri önceliklendirilerek kritik düğümler araştırılır. Bu süreçte $k/2$ kere DFS ağacı oluşturulup, her bir ağaçtan/ormandan bir kritik düğüm seçilir. Dolayısıyla, GreedyVA ve TrimCut algoritmalarının iterasyon sayıları eşit olmakla birlikte, GreedyVA TrimCut algoritmasına göre $k/2$ ekstra DFS ağacı oluşturmaktadır. Bu da algoritmanın işlem maliyetini yükseltmektedir. Bu yaklaşım farkı, Tablo 5'te verilen değerlerin sebebini oluşturmaktadır.

Tablo 5. Algoritmaların çalışma sürelerinin (saniye) karşılaştırılması
Table 5. *Comparing the run times (in seconds) of the algorithms*

Instance	***n***	***k***	***GreedyVA***	***TrimCut***
BA	500	50	0.089	0.069
BA	1000	75	0.531	0.401
BA	2500	100	4.182	3.135
BA	5000	150	24.243	18.329
FF	250	50	0.022	0.016
FF	500	110	0.174	0.138
FF	1000	150	0.936	0.723
FF	2000	200	4.789	3.702
ER	235	50	0.019	0.014
ER	466	80	0.115	0.084
ER	941	140	0.623	0.606
ER	2344	200	7.150	5.341
WS	250	70	0.027	0.021
WS	500	125	0.179	0.141
WS	1000	200	1.094	0.884
WS	1500	265	3.275	2.603

6. Sonuç
6. Conclusion

Bu çalışmada, KDP için büyük ölçekli ağlarda optimale yakın çözümler sunmak amacıyla TrimCut algoritması geliştirilmiştir. KDP'nin ağ dayanıklılığı ve zayıflıklarının tespiti gibi önemli uygulama alanlarına sahip olduğu göz önüne alındığında, önerilen algoritma sosyal ağlar, biyolojik ağlar, telekomünikasyon ağları ve kablosuz çok sekmeli ağlar gibi farklı ağ yapıları üzerinde uygulanabilir. Algoritmanın temel yaklaşımı, öncelikle yüksek dereceli düğümlerin çıkartırılarak ağın seyreltilmesi ve ardından DFS ile kesim düğümlerini

önceliklendirilerek kritik düğümlerin tespit edilmesi üzerine kuruludur. Gerçekleştirilen deneyler, TrimCut algoritmasının literatürdeki diğer polinom zamanlı algoritmalarla karşılaştırıldığında daha iyi performans sergilediğini ve hata oranının %6'nın altında olduğunu göstermiştir. Özellikle büyük ölçekli ağlarda, algoritmanın hem hesaplama süresi açısından verimli olduğu hem de çözüm kalitesi bakımından tatmin edici sonuçlar verdiği tespit edilmiştir. Bu çalışmada elde edilen sonuçlar, ağ güvenilirliği ve zayıf noktaların tespit edilmesi konularında yeni yaklaşımlar sunarken, gelecekteki çalışmalar için de potansiyel araştırma alanları sunmaktadır. Özellikle, farklı ağ yapılarında zayıflıkların tespit edilmesi ve algoritmanın daha da geliştirilmesi, ileride yapılacak çalışmalar için önemli katkılar sağlayabilir.

Teşekkür
Acknowledgement

Bu çalışma TÜBİTAK 3501 programı tarafından (121F092) desteklenmiştir. Y. Aygül, BİDEB 2211-A programı kapsamında verdiği burs desteğinden dolayı TÜBİTAK'a teşekkür eder.

Yazar katkısı
Author contribution

Yazarlardan Onur UĞURLU makale yazımının düzenlenmesinde, teorik çerçeve oluşturma ve yazım aşamasında katkı sağlamıştır. Yeşim AYGÜL literatür taraması, algoritmanın veri seti üzerinde test edilmesinde, ilgili tablo ve şekillerin düzenlenmesinde yer almıştır.

Etik beyanı
Declaration of ethical code

Bu makalenin yazarları, bu çalışmada kullanılan materyal ve yöntemlerin etik kurul izni ve / veya yasal-özel izin gerektirmediğini beyan etmektedir.

Çıkar çatışması beyanı
Conflicts of interest

Yazarlar herhangi bir çıkar çatışması olmadığını beyan eder.

Kaynaklar
References

Akram, V. K., & Ugurlu, O. (2023). Detecting the most vital articulation points in wireless multi-hop networks. IEEE/ACM Transactions on Networking. https://doi.org/10.1109/TNET.2023.3308142

Aringhieri, R., Grosso, A., Hosteins, P., & Scatamacchia, R. (2016). Local search metaheuristics for the critical node problem. Networks, 67(3), 209–221. https://doi.org/10.1002/net.21671

Aringhieri, R., Grosso, A., Hosteins, P., & Scatamacchia, R. (2016). A general evolutionary framework for different classes of critical node problems. Engineering Applications of Artificial Intelligence, 55, 128–145. https://doi.org/10.1016/j.engappai.2016.06.010

Arulselvan, A., Commander, C. W., Elefteriadou, L., & Pardalos, P. M. (2009). Detecting critical nodes in sparse graphs. Computers & Operations Research, 36(7), 2193–2200. https://doi.org/10.1016/j.cor.2008.08.016

Block, P., Hoffman, M., Raabe, I. J., Dowd, J. B., Rahal, C., Kashyap, R., & Mills, M. C. (2020). Social network-based distancing strategies to flatten the COVID-19 curve in a post-lockdown world. Nature Human Behaviour, 4(6), 588–596. https://www.nature.com/articles/s41562-020-0898-6

Borgatti, S. P. (2006). Identifying sets of key players in a social network. Computational and Mathematical Organization Theory, 12(1), 21–34. https://doi.org/10.1177/0160017607308679

Cobanlar, M., Yildiz, H. U., Akram, V. K., Dagdeviren, O., & Tavli, B. (2022). On the tradeoff between network lifetime and k-connectivity-based reliability in UWSNs. IEEE Internet of Things Journal, 9(23), 24444-24452.

Commander, C. W., Pardalos, P. M., Ryabchenko, V., Uryasev, S., & Zrazhevsky, G. (2007). The wireless network jamming problem. Journal of Combinatorial Optimization, 14(4), 481–498. https://link.springer.com/article/10.1007/s10878-007-9071-7

Dagdeviren, O., Akram, V. K., & Tavli, B. (2018). Design and evaluation of algorithms for energy efficient and complete determination of critical nodes for wireless sensor network reliability. IEEE Transactions on Reliability, 68(1), 280-290.

Di Summa, M., Grosso, A., & Locatelli, M. (2012). Branch and cut algorithms for detecting critical nodes in undirected graphs. Computational Optimization and Applications, 53(3), 649–680. https://link.springer.com/article/10.1007/s10589-012-9458-y

Diestel, R. (2012). Graph theory (4th ed., Vol. 173). Graduate Texts in Mathematics, Springer.

Dinh, T. N., & Thai, M. T. (2011). Precise structural vulnerability assessment via mathematical programming. In Military Communications Conference, IEEE, 1351–1356. https://doi.org/10.1109/MILCOM.2011.6127492

Dinh, T. N., Xuan, Y., Thai, M. T., Park, E., & Znati, T. (2010). On approximation of new optimization methods for assessing network vulnerability. In INFOCOM Proceedings IEEE, 1–9. https://doi.org/10.1109/INFCOM.2010.5462098

Edalatmanesh, M. (2013). Heuristics for the critical node detection problem in large complex networks. Ph.D. dissertation, Faculty of Mathematics and Science, Brock University, St. Catharines, Ontario. http://hdl.handle.net/10464/4984

Grubesic, T. H., Matisziw, T. C., Murray, A. T., & Snediker, D. (2008). Comparative approaches for assessing network vulnerability. International Regional Science Review, 31(1), 88–112. https://doi.org/10.1177/0160017607308679

Jhoti, H., & Leach, A. R. (2007). Structure-based drug discovery (1st ed.). Springer. https://link.springer.com/book/10.1007/1-4020-4407-0

Krebs, V. (2002). Uncloaking terrorist networks. First Monday, 7(4), 1–7. https://doi.org/10.5210/fm.v7i4.941

Lalou, M., Tahraoui, M. A., & Kheddouci, H. (2018). The critical node detection problem in networks: A survey. Computer Science Review, 28, 92-117.

Lewis, J. M., & Yannakakis, M. (1980). The node-deletion problem for hereditary properties is NP-complete. Journal of Computer and System Sciences, 20(2), 219–230. https://doi.org/10.1016/0022-0000(80)90060-4

Liljefors, T., Krogsgaard-Larsen, P., & Madsen, U. (2002). Textbook of drug design and discovery (3rd ed.). CRC Press. https://doi.org/10.1201/b12381

Ovelgonne, M., Kang, C., Sawant, A., & Subrahmanian, V. (2012). Covertness centrality in networks. In Advances in Social Networks Analysis and Mining, ASONAM, IEEE, 863–870. https://doi.org/10.1109/ASONAM.2012.156

Purevsuren, D., Cui, G., Win, N. N. H., & Wang, X. (2016). Heuristic algorithm for identifying critical nodes in graphs. Advanced Computer Science: International Journal, 5(3), 1–4. https://doi.org/10.1016/j.cor.2019.02.006

Tomaino, V., Arulselvan, A., Veltri, P., & Pardalos, P. M. (2012). Studying connectivity properties in human protein–protein interaction network in cancer pathway. In Data Mining for Biomarker Discovery, Springer, 187–197. https://link.springer.com/chapter/10.1007/978-1-4614-2107-8_10

Ugurlu, O. (2023). ADA-PC: An asynchronous distributed algorithm for minimizing pairwise connectivity in wireless multi-hop networks. Computer Networks, 227, 109703. https://doi.org/10.1016/j.comnet.2023.109703

Ventresca, M. (2012). Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. Computers & Operations Research, 39(11), 2763–2775. https://doi.org/10.1016/j.cor.2012.02.008

Ventresca, M., & Aleman, D. (2014). A derandomized approximation algorithm for the critical node detection problem. Computers & Operations Research, 43, 261–270. https://doi.org/10.1016/j.cor.2013.09.012

Ventresca, M., & Aleman, D. (2015). Efficiently identifying critical nodes in large complex networks. Computational Social Networks, 2(6), 1–12. https://computationalsocialnetworks.springeropen.com/articles/10.1186/s40649-015-0010-y

Veremyev, A., Boginski, V., & Pasiliao, E. L. (2014a). Exact identification of critical nodes in sparse networks via new compact formulations. Optimization Letters, 8(4), 1245–1259. https://link.springer.com/article/10.1007/s11590-013-0666-x

Veremyev, A., Prokopyev, O. A., & Pasiliao, E. L. (2014b). An integer programming framework for critical elements detection in graphs. *Journal of Combinatorial Optimization*, *28*, 233-273. https://link.springer.com/article/10.1007/s10878-014-9730-4

Veremyev, A., Prokopyev, O. A., & Pasiliao, E. L. (2015). Critical nodes for distance-based connectivity and related problems in graphs. Networks, 66(3), 170–195. https://doi.org/10.1002/net.21622

Yannakakis, M. (1981). Node-deletion problems on bipartite graphs. SIAM Journal on Computing, 10(2), 310–327. https://doi.org/10.1137/0210022

Zhou, Y., Hao, J. K., & Glover, F. (2019). Memetic search for identifying critical nodes in sparse graphs. IEEE Transactions on Cybernetics, 49(10), 3699–3712. https://doi.org/10.1109/TCYB.2018.2848116