Iğdır Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 15(3), 744-754, 2025
Journal of the Institute of Science and Technology, 15(3), 744-754, 2025
ISSN: 2146-0574, eISSN: 2536-4618
Bilgisayar Mühendisliği / *Computer Engineering* **DOI: 10.21597/jist.1569294**
Araştırma Makalesi / *Research Article*
Geliş tarihi / *Received*: 21.10.2024 **Kabul tarihi / *Accepted*: 17.02.2025**

Atıf İçin: Aygül, Y., Uğurlu, O., Akram, V. ve Eliiyi, D.T. (2025). Kentsel Yol Ağlarında Kritik Kenar Tespiti için Etkin Bir Algoritma. *Iğdır Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 15(3), 744-754.

To Cite: Aygül, Y., Ugurlu, O., Akram, V. & Eliiyi, D.T. (2025). An Effective Algorithm for Critical Edge Detection in Urban Road Networks. *Journal of the Institute of Science and Technology*, 15(3), 744-754.

Kentsel Yol Ağlarında Kritik Kenar Tespiti için Etkin Bir Algoritma

Yeşim AYGÜL[1], Onur UĞURLU[1*], Vahid AKRAM[2], Deniz TÜRSEL ELİİYİ[1]

Öne Çıkanlar:
- Ağ analizi
- Kritik kenarlar
- Ağ algoritmaları

Anahtar Kelimeler:
- Trafik ağları
- Ağ analizi
- Kritik kenarlar
- Ağ algoritmaları

ÖZET:

Bu çalışma, trafik ağlarının etkin yönetimi ve optimizasyonu için önemli bir problem olan Kritik Kenar Problemine odaklanmaktadır. Kritik Kenar Problemi, ağdan çıkarılması ile ağın bağlantısallığına en çok zarar verecek kenar kümesini belirlemeyi amaçlayan bir optimizasyon problemidir. Kritik Kenar Problemi trafik yönetimi, acil durum müdahaleleri, altyapı yatırımları ve ağ dayanıklılığını artırma gibi çok sayıda önemli uygulama alanına sahiptir. Bu çalışmanın temel motivasyonu özellikle büyük ölçekli gerçek hayat ağları üzerinde verimli bir şekilde kullanılabilecek yeni bir polinom zamanlı algoritmanın geliştirilmesidir. Geliştirilen algoritmanın efektifliği küçük ölçekli ağlar üzerinde klasik kaba-kuvvet yaklaşımıyla elde edilen optimal sonuçlarla karşılaştırılarak test edilmiştir. Önerilen yöntem, trafik sıkışıklığını azaltma, seyahat sürelerini kısaltma ve çevresel etkileri minimize etme gibi çeşitli kentsel zorluklarla başa çıkma potansiyeli taşımaktadır. Bu araştırma, büyük ölçekli trafik ağlarının anlaşılmasını ve verimli ulaşım sistemlerinin geliştirilmesini teşvik etmek için etkin bir yaklaşım sunmaktadır.

An Effective Algorithm for Critical Edge Detection in Urban Road Networks

Highlights:
- Network analysis
- Critical edges
- Graph algorithms

Keywords:
- Traffic networks
- Network analysis
- Critical edges
- Graph algorithms

ABSTRACT:

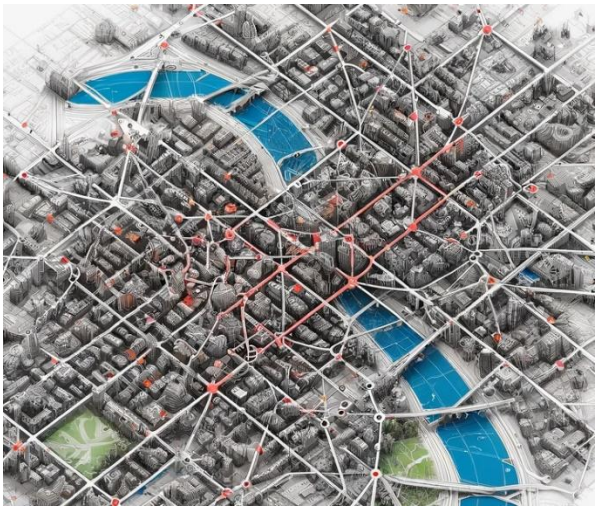
This study focuses on the Critical Edge Problem, which is an important problem for effectively managing and optimizing traffic networks. The Critical Edge Problem is an optimization problem that aims to determine the set of edges whose removal from the network will cause the most damage to the network's connectivity. The Critical Edge Problem has many important application areas, such as traffic management, emergency responses, infrastructure investments, and increasing network resilience. The primary motivation of this study is the development of a new polynomial-time algorithm that can be used efficiently, especially on large-scale real-life networks. The effectiveness of the developed algorithm was tested by comparing it with the optimal results obtained by classical brute-force approaches on small-scale networks, and the theoretical foundations and practical applications of the algorithm were comprehensively examined. The proposed method has the potential to tackle various urban challenges, such as reducing traffic congestion, shortening travel times, and minimizing environmental impacts. This research provides an effective method to promote the understanding of large-scale traffic networks and the development of efficient transportation systems.

[1]Yeşim AYGÜL (**Orcid ID:** 0000-0003-0605-9604), Onur UĞURLU (**Orcid ID:** 0000-0003-2743-5939), Deniz TÜRSEL ELİİYİ (**Orcid ID:** 0000-0001-7693-3980), İzmir Bakırçay Üniversitesi, Mühendislik ve Mimarlık Fakültesi, İzmir, Türkiye
[2]Vahid AKRAM (**Orcid ID:** 0000-0002-4082-6419), Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, İzmir, Türkiye

***Sorumlu Yazar/*Corresponding Author**: Onur UĞURLU, e-mail: onur.ugurlu@bakircay.edu.tr

GİRİŞ

Trafik ağları, kentsel planlama ve altyapı yönetiminin en önemli unsurlarından biridir. Bu ağlar, şehirlerin işlevselliği ve sakinlerinin yaşam kalitesi üzerinde önemli bir etkiye sahiptir. Trafik ağlarının analizi, kentsel ve bölgesel planlama disiplinlerinin merkezi konularından biridir. Bu analizler, şehir içi ve şehirlerarası ulaşım sistemlerinin verimliliğini artırmak, trafik yoğunluğunu azaltmak ve sürdürülebilir ulaşım çözümleri geliştirmek için kritik öneme sahiptir (Karimi ve ark., 2021; Minh ve ark., 2022). Trafik ağlarının etkin bir şekilde analiz edilmesi, planlamacılara ve mühendislere, trafik akışını daha iyi anlama ve yönetme imkânı tanır. Bu analizler, çeşitli matematiksel modeller ve simülasyon araçları kullanılarak gerçekleştirilmekte olup, bu modeller yol kullanımını ve araçların yol üzerindeki etkileşimlerini detaylı bir şekilde incelemekte ve değerlendirmektedir. Analiz sonuçları, trafik sıkışıklığını azaltma, seyahat sürelerini kısaltma ve yakıt tüketimini düşürme gibi konularda stratejik kararlar alınmasında temel teşkil etmektedir. Ayrıca, trafik ağlarının optimizasyonu, acil durum yönetimi ve afetlere hazırlık açısından da büyük önem taşımaktadır. Özellikle deprem, sel gibi doğal afetler sırasında trafik ağlarının işlevselliği, acil servislerin olay yerine hızla ulaşabilmesi açısından hayati öneme sahiptir. Bu bağlamda, trafik ağlarında kritik yolların ve geçiş noktalarının belirlenmesi, acil durum araçlarının en etkili şekilde yönlendirilmesini sağlar. Son olarak, trafik ağlarının optimizasyonu karbon ayak izini küçültme çabalarıyla doğrudan ilişkilidir. Etkin trafik yönetimi sayesinde, araçların rölanti süreleri azalır, hızlı ve akıcı trafik akışı sağlanır. Bu durum, araçların karbon salımını azaltarak çevresel sürdürülebilirliğe katkıda bulunur. Dolayısıyla, trafik ağlarının analizi ve optimizasyonu, yalnızca kentsel altyapının sadece işlevselliğini değil, aynı zamanda ekolojik dengesini de destekleyen temel bir unsurdur (Jilani ve ark., 2023). Şekil 1'de bir şehir trafik ağı ve ağdaki kritik yolların (kenarların) temsili bir görseli verilmiştir.



Şekil 1. Şehir trafik ağı ve kritik yolların temsili bir görseli

Ağ analizindeki temel araştırmalardan biri ağdaki kritik elemanların tespit edilmesidir. Ağdan çıkarıldığında ön tanımlı bir metriğe göre ağın bağlantılılığına en çok zarar veren elemanlar (düğüm ya da kenar) kritik eleman olan değerlendirilir (Lalou ve ark., 2018). Kritik Kenar Tespiti Problemleri, bu kenarların kümesini bulmayı amaçlayan optimizasyon problemleri ailesidir. Bu problemlerin efektif şekilde çözülmesi özellikle trafik ağlarında, ana arterlerin veya kilit geçiş noktalarının belirlenmesine olanak tanır. Bu geçiş noktalarının güçlendirilmesi, trafik akışını optimize eder, trafik yoğunluklarının önlenmesinde kritik rol oynar ve acil durum hizmetlerinin etkinliğini artırır. Trafik ağlarında kritik kenar tespiti problemleri aşağıdaki başlıca uygulama alanlarına sahiptir:

- **Trafik Akışının Optimizasyonu:** Kritik kenarlar, trafik yoğunluğunun yönetilmesi ve rotaların etkin bir şekilde planlanması için kullanılabilir. Bu kenarlar üzerindeki trafik akışını izlemek, gerektiğinde alternatif yollar önermek ve trafik sıkışıklığını azaltmak için kritik öneme sahiptir (Tekgöz ve ark., 2022).
- **Acil Durum Yönetimi:** Afet anlarında veya acil durumlarda, kritik kenarlar, yardım ve kurtarma ekiplerinin hızla hareket etmesini sağlayan rotaları belirlenmesine yardımcı olur. Bu kenarların belirlenmesi, acil durum araçlarının en kısa sürede olay yerine ulaşmasını ve etkili bir müdahale yapılmasını sağlar (Eren ve Akdaş, 2023).
- **Altyapı Yatırımları:** Şehir planlamacıları ve mühendisler için kritik kenarların tespiti, altyapı yatırımlarının önceliklendirilmesinde önemli bir rol oynar. Bu bağlantılar, özellikle yüksek trafiğe maruz kalan ve bu nedenle sürekli bakım ve iyileştirmeye ihtiyaç duyulan yolları belirlemede kullanılır (Pana-Micu, 2024).
- **Ağ Dayanıklılığı:** Kritik kenar analizi, trafik ağlarının potansiyel zayıf noktalarını ortaya çıkarabilir. Bu analiz, belirli yolların veya köprülerin hasar görmesi durumunda ağın nasıl tepki vereceğini anlamak için kullanılır. Böylece, ağın dayanıklılığını artıracak şekilde tasarım ve iyileştirmeler yapılabilmektedir (Gauthier ve ark., 2018).

Literatürde kritik kenar problemleri incelendiğinde, bir kenarın kritikliği için farklı metriklerin kullanıldığı görülmektedir (ağdaki toplam en kısa yollara getirilen maksimum artış, ağ akışındaki maksimum azalma vb.). Bu problemler literatürde aynı zamanda "Ağ Engelleme Problemleri" (Walteros ve Pardalos, 2012) olarak bilinmekte ve birçok araştırmacının ilgisini çekmektedir. Rashmi ve ark., (2020), Louvain algoritmasını kullanarak trafik tıkanıklığını tespit etmeyi ve alternatif güzergâhlar önermeyi hedeflemiştir. Bu çalışmada araştırmacılar, ABD yol ağlarında Neo4j araçlarıyla topluluk yapılarını analiz etmişlerdir. Yuan ve ark., (2023) tarafından kentsel yol ağlarındaki kritik bağlantıların belirlenmesi için Coğrafi Bilgi Sistemi tabanlı bir yöntem önerilmektedir. Çalışma, yol altyapısının dağılımı, yol yoğunluğu ve ağ erişilebilirliği gibi faktörleri değerlendirerek kritik bağlantıları tespit etmektedir. Zhao ve ark., (2023) tarafından, trafik akışı ve yol ağı yapısını dikkate alarak kritik tıkanıklık yaşanan yolların belirlenmesi için yeni bir model önerilmiştir. Önerilen model, yol segmentlerinin özelliklerini öğrenmek için Markov Zinciri tabanlı bir rastgele yürüyüş ve Skip-gram modelini birleştirmektedir. Saito ve ark., (2024) coğrafi yol ağlarını analiz ederek, tahliye durumlarında kritik sokakların belirlenmesi için yeni bir yöntem sunmuştur. Önerilen yöntem, yol ağlarını topolojik bir temsil kullanarak modellemekte ve yüksek önem taşıyan sokakları tespit etmektedir. Kritik kenar problemleri için detaylı bir literatür incelemesi Smith ve ark. (2013) ve Smith ve Song (2019) incelenebilir.

Dinh ve ark. (2011) tarafından gerçekleştirilen çalışmada, ikili bağlantılılığa en büyük zararı veren kenarlar kritik kenar olarak değerlendirilmiştir (Kritik Kenar Problemi - (KKP)). Literatürde bu problem, özellikle bilgisayar bilimleri, elektrik mühendisliği ve sosyal bilimler dahil olmak üzere çeşitli disiplinlerde incelenmiştir (Yu ve ark., 2018). Optimizasyon problemlerinin çözümleri temelde ikiye ayrılabilir: Kesin yöntemler ve yaklaşık yöntemler. Kaba kuvvet ve doğrusal programlama gibi kesin yöntemler ele alınan problemin optimal (en iyi) çözümünü bulmayı garanti eder, ancak bu yöntemler büyük boyutlu gerçek hayat problemleri için çok uzun hesaplama süreleri gerektirmektedir. Bu sebeple araştırmacılar büyük boyutlu örnekler için makul sürelerde optimale yakın çözümler bulmayı hedefleyen yaklaşık yöntemleri tercih etmektedir. Bu kapsamda en çok kullanılan yöntemlerden biri polinom zamanlı algoritmalardır. KKP, NP-Zor bir problem olduğundan (Bazgan ve ark., 2013), problemin boyutu arttıkça, problem için makul sürelerde optimal çözümlerin bulunması neredeyse imkansızdır. Bu bağlamda, bu çalışmanın temel motivasyonu büyük boyutlu trafik ağları

üzerinde KKP için hesaplama verimliliği açısından etkin bir algoritma geliştirmektir. Geliştirilen algoritmanın efektifliği optimal sonucu bilinen küçük ağlar üzerinde gösterilmiştir.

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, önerilen algoritma ve karmaşıklık analizini içermektedir. Dördüncü bölüm hesapsal sonuçları sunmaktadır. Son bölümde çalışmayla ilgili genel değerlendirmelere ve gelecekte yapılacak çalışmalar için önerilere yer verilmiştir.

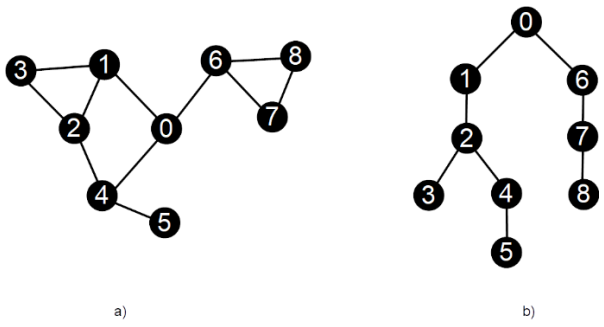
MATERYAL VE METOT

Hedef Problemin Formal Tanımı ve Ön Bilgiler

Belirli fenomenler ve bu fenomenler arasındaki ilişkilerden oluşan birçok yapı çizgeler ile modellenebilmektedir. Çizge kuramı bu yapıların incelendiği bilgisayar bilimlerindeki en önemli çalışma alanlarından biridir. Günlük hayatta karşılaşılan pek çok problem çizgeler üzerinde optimizasyon problemleri olarak görülebilir. $G(V,E)$ bir çizge olmak üzere, V kümesi çizgedeki düğümleri, E kümesi ise çizgedeki kenarları temsil etmektedir. Ayrıca $n=|V|$ çizgedeki düğüm sayısını, $m=|E|$ ise çizgedeki kenar sayısını ifade etmektedir. Bir çizgede yoğunluk, çizgenin sahip olduğu kenarların sayısının, maksimum olası kenar sayısına oranıdır ve çizgenin ne kadar sıkı bağlantılı olduğunu gösterir. Bir çizgenin bir düğümünün derecesi, o düğüme bağlı kenarların sayısıdır. Δ ise bir çizgenin maksimum derecesi (maksimum komşu sayısı) olarak tanımlanır. Bir başka deyişle, bir çizgenin herhangi bir düğümünün sahip olduğu en fazla kenar sayısını ifade eder. Çizgedeki kenarlar sadece düğümler arasındaki bir ilişkinin olup olmadığını gösteriyorsa (0/1) bu çizgelere ağırlıksız çizge denilmektedir. Öte yandan çizgedeki kenarlar nümerik değerlere sahipse bu çizgelere ağırlıklı çizge adı verilir (Diestel, 2012). KKP doğası gereği için kenarların belli ağırlıklara sahip bir problem olduğu açıktır (trafik ağları düşünüldüğünde bu ağırlık yolların uzunluğunu temsil edecektir).

Bir çizgede, birbirine bağlı bir dizi düğümden oluşan ve her bir düğümün yalnızca bir kez ziyaret edildiği ardışık bağlantılar kümesine yol denir. Başlangıç ve bitiş noktası aynı olan yollara ise çevre denir. Eğer bir çizge çevre içermiyorsa ve her iki düğüm arasında yalnızca tek bir yol bulunuyorsa bu tür çizgelere ağaç denir. Derinlik Öncelikli Arama (DFS), bir çizge ya da ağacın düğümlerini derinlemesine keşfetmek için kullanılan bir arama algoritmasıdır. Bu algoritma, başlangıç düğümünden başlar ve her bir düğümün tüm komşularını ziyaret etmeden önce derinlere inerek bir yol boyunca ilerler. Bir düğümden geriye dönme ihtiyacı doğduğunda, önceki dallara geri dönülerek keşfedilmeyen diğer düğümlere geçilir. DFS, özellikle geri izleme teknikleri gerektiren problemlerin çözümünde kullanılan bir arama algoritmasıdır. DFS ağacı ise, bir çizgenin DFS ile gezilmesi sırasında oluşturulan ve başlangıç düğümünden itibaren ulaşılan düğümleri ve kullanılan kenarları içeren bir ağaç yapısıdır. DFS ağacı, çizgenin hangi düğümlerinin hangi sırayla ziyaret edildiğini gösterir (Diestel, 2012). Şekil 2.a'da örnek bir çizge, Şekil 2.b'de ise bu örnek çizge üzerinde 0 düğümünü kök kabul ederek oluşturulmuş DFS ağacı verilmektedir.

Ağ birleştirilmişliği ağların sağlamlığı hakkında bilgi veren önemli bir ölçüttür. Ağ analizinde bir ağın birleştirilmişliği için önemli olan elemanların alt kümelerini (düğüm ya da kenar) tespit etme sorunu özellikle son yıllarda araştırmacılar tarafından sıklıkla çalışılan önemli bir problem haline gelmiştir (Lalou ve ark., 2018).



Şekil 2. a) Örnek bir çizge ve b) bu çizge üzerindeki DFS ağacı

Bir ağın bağlılığı için önemli elemanların bulunması, ağın zedelenebilirliği, dayanıklılığı ve atak toleransı gibi birçok temel yapısal özelliğinin incelenmesinde büyük rol oynamaktadır. Bu önemli elemanlar, literatürde "Kritik Elemanlar" olarak bilinmektedir. Kritik elemanların tespit edilmesindeki temel amaç yokluğunda ağda olumsuz etki oluşturacak grupları belirlemek ve bu grubun korunmasını ya da güçlendirilmesini sağlamaktır. Problem iç güvenlik stratejileri (Grubesic ve ark., 2008; Houck ve ark., 2004), ulaşım (Jenelius ve ark., 2006) ve enerji (Salmeron ve ark., 2004) gibi pek çok sektörde uygulamalara sahiptir.

Ağırlıklı bir çizgede iki düğüm arasındaki bağlantı bir tamsayı ile (iki düğüm arasındaki mesafe) gösterilir. Ağırlıklı bir $G=(V, E)$ çizgesi ve bir k tamsayısı verildiğinde, KKP (Walteros ve Pardalos, 2012), silinmesi ile elde edilen $G(V, E\backslash B)$ çizgesindeki düğüm ikilileri arasındaki en kısa yolların toplamını enbüyükleyen ve uzunluklarını toplamı k değerini geçmeyen bir $B \subseteq E$ kümesini bulmaya çalışır.

KKP problemi aşağıdaki gibi formülize edilebilir.

Girdi: $G=(V, E)$ çizgesi ve bir k tamsayısı

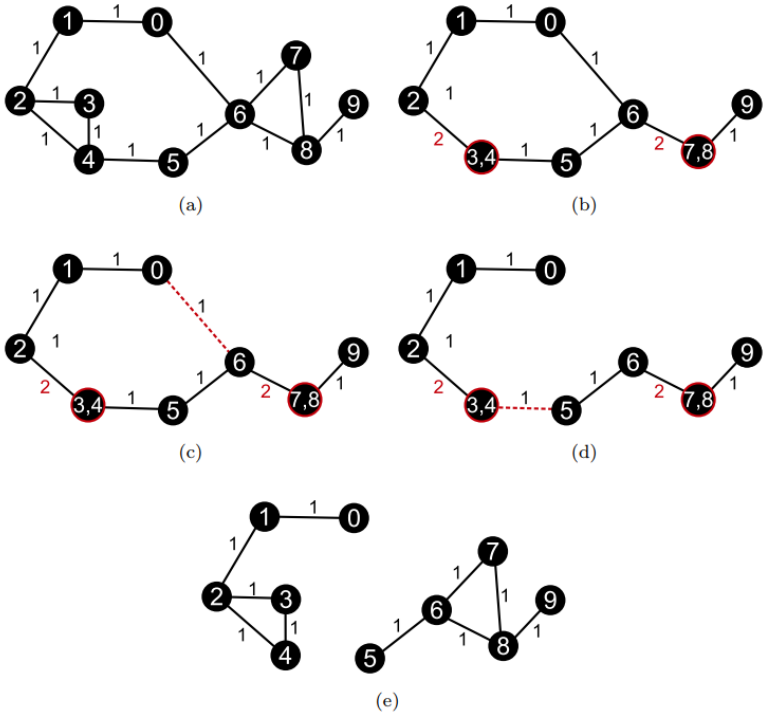
Çıktı: $|B| \leq k$ olacak şekilde $max \sum_{i,j \in V} l_{ij}$

Önerilen formülasyonda l_{ij} parametresi, i ve j düğümleri arasındaki en kısa yolun uzunluğunu temsil eder. Aralarında bağlantı olmayan iki düğümün aralarındaki en kısa yol büyük bir sabit alındığında ikili bağlantının minimize edilmesi, ağdaki en kısa yolların toplamının maksimize edilmesine denk bir amaç fonksiyonu olmaktadır.

Önerilen Algoritma

Herhangi bir ağaçta bir kenarın ağaçtan çıkarılması ağı 2 parçaya ayırır (Ağaçlarda herhangi iki düğüm arasında sadece tek bir yol olduğundan, ağaçtaki herhangi bir kenarın ağaçtan çıkarılması ağacın bağlantısının kopmasına sebep olacaktır.). KKP için geliştirilen algoritmanın (Tree_Cut) temel fikri ağı mümkün olduğunca en az bilgi kaybı ile bir ağaç yapısına çevirmek ve sonrasında ağaçtaki en kritik kenarları tespit etmektir. Dolayısıyla, Tree_Cut algoritması 2 temel aşamadan oluşmaktadır. İlk aşamada ilgili ağ, ağaç yapısına dönüştürülür. Herhangi bir ağı bir ağaca çevirmek için ağdaki çevrelerin silinmesi gerekmektedir. Bu doğrultuda, geliştirilen algoritmada literatürde kullanılan yöntemlerden düğüm birleştirme (Nagamochi ve ark., 1994; Stoer ve Wagner, 1997; Brinkmeier, 2007) yöntemi kullanılmıştır. Bu yöntemde ele alınan iki düğüm (diyelim ki düğüm u ve düğüm v) birleştirilirken u ve v tek bir düğüm olarak ele alınır ve u ve v arasındaki kenar silinir (yöntemin detaylı açıklaması Şekil 3'te verilmiştir). Herhangi bir ağda bir çevrenin bulunması için ilgili ağ boyutu (ağdaki düğüm sayısı) 2'den fazla olan ya bir alt tam çizge ya da bir alt çevre çizge içermelidir. Boyutu 3'ten büyük olan herhangi bir tam çizge 3 düğümden oluşan bir tam çizge içermek zorunda olduğu için Tree_Cut algoritması ilk olarak çizgedeki 3 düğümden oluşan tam çizgeleri tespit edip, ilgili alt çizgenin iki düğümünü birleştirir. Bu işlem ağda bir alt tam çizge kalmaya kadar devam eder.

Bu işlem sonrası ağda bir çevrenin bulunabilmesi için boyutu 3'ten büyük olan bir çevre çizgenin (3 düğümden oluşan çevre çizgeler aynı zaman 3 düğümden oluşan bir tam çizge olduğu için bir önceki adımda dikkate alınmıştır) bulunması gerekir. Bu çevre çizgelerin tespiti için Tree_Cut algoritması bir DFS ağacı oluşturur ve ağaç oluşturma sürecinde çevre oluşturan kenarlar tespit edilerek, KKP'in çözüm kümesine eklenir. Ağdaki tüm çevreler silindikten sonra aşama 2'de, kalan ağaçta (ve sonraki iterasyonlardaki ağaçlarda) KKP'in amaç fonksiyonuna göre en çok kazanç getiren kenar tespit edilerek ağdan çıkarılır.



Şekil 3. Tree_Cut algoritmasının temel adımları

Tree_Cut algoritması hem ağırlıklı hem de ağırlıksız çizgeler üzerinde efektif bir şekilde çalışmaktadır. Anlatım kolaylığı açısından algoritmanın temel çalışma prensipleri ağırlıksız bir ağ üzerinde anlatılmıştır. Şekil 3'te Tree_Cut algoritmasının temel adımları görselleştirilmiştir. Şekil 3.a' daki örnek çizge üzerinde k=2 için KKP çözülmek istenildiğinde, Tree_Cut algoritması öncelikle ağı ağaç yapısına çevirir. Bu doğrultuda ilk olarak 3 düğümden oluşan tam çizgeler tespit edilir. Şekil 3.a'daki çizge incelendiğinde düğüm 2, 3 ve 4'ten oluşan bir tam çizge görülmektedir. Burada tam çizgedeki iki düğüm seçilerek (düğüm 3 ve 4) birleştirilir ve aralarındaki kenar çizgeden çıkarılır.

Düğüm 3 ve 4 birleştirilirken bu iki düğümün tüm bağlantıları bu yeni düğüme eklenir ve eğer birleştirilen düğümlerin ortak komşuları varsa bu komşu ile olan kenarın değeri 1 artırılır (Şekil 3.b) (aynı durum düğüm 6, 7 ve 8 içinde geçerlidir). Düğüm 3,4 ve düğüm 7,8 birleştirildikten sonra ağda herhangi bir 3 düğümlü tam çizgenin kalmadığı görülmektedir (Şekil 3.b). Sonraki adım bir DFS ağacı ile ağdaki çevre tespit edilir (düğüm 0-1-2-3,4-5-6-0), ve çevre oluşumunu sağlayan kenar, kritik kenar (0-6) olarak işaretlenir (Şekil 3.c). Ağın ağaç yapısının dönüşümü tamamlandıktan sonra oluşturulan DFS ağacında KKP amaç fonksiyonuna en çok azalma sağlayan ve aynı zamanda değeri en düşük olan kenar seçilerek kritik kenar olarak (3,4-5) işaretlenir. Örnek çizge için Tree_Cut algoritmasının bulduğu kritik kenarlar ve kalan ağın yapısı Şekil 3.e'de görselleştirilmiştir.

Algorithm 1: TreeCut- Aşama 1

Girdi: $G(V;E)$, k
Başlangıçta: (parent[], S) ← {}
1: **while** (G'de 3 düğümlü tam çizge olduğu sürece) **do**
2: **for** G'deki her düğüm v_i için **do**
3: **for** v_i'nin her komşusu v_j için **do**
4: **for** v_j'nin her komşusu v_k için **do**
5: **if** (v_i, v_k'nin komşusuysa) **then**
6: $merge(v_i, v_k)$
7: **end if**
8: **end for**
9: **end for**
10: **end for**
11: **end while**
12: v_i= G'deki herhangi bir düğüm
13: push(v_i, S)
14: v_i'yi ziyaret edildi olarak işaretle
15: **while** (S boş değilse) **do**
16: v_j=peek(S)
17: v_k= v_j'nin ziyaret edilmemiş bir komşusu
18: **if** (v_k mevcutsa) **then**
19: push(v_k, S)
20: parent[v_k]=v_j
21: v_k'yi ziyaret edildi olarak işaretle
22: **if** (v_k, ziyaret edilmiş bir komşu v_z'ye sahipse ve (v_j!=v_z)) **then**
23: $e(v_k, v_z)$'yi kritik kenar olarak işaretle
24: $G = G \setminus \{e(v_k, v_z)\}$
25: **end if**
26: **else**
27: pop(S)
28: **end if**
29: **end while**

Şekil 4. Tree_Cut Algoritmasının Sözde Kodu – Aşama 1

Şekil 3'teki örnek üzerinde de görüldüğü gibi Tree_Cut algoritması birbirini takip eden iki temel aşamadan oluşmaktadır. İlk aşamada verilen ağ ağaç yapısına dönüştürülür, ikinci aşamada ise oluşturulan ağaçta kritik kenarlar tespit edilir. Şekil 4'te Tree_Cut algoritmasının ilk aşamasının sözde kodu verilmiştir. İlgili algoritmada ilk olarak 3 düğümlü tam çizgeler tespit edilmeye çalışılır. Bu doğrultuda her bir v_i düğümü için v_i düğümünün tüm komşuları (v_j) kontrol edilir. Eğer v_i ve v_j düğümlerine komşu bir v_k düğümü varsa bu durumda v_i ve v_k düğümleri birleştirilir (Algoritma 1 – Adım 2-10). Bu işlem ağda 3 düğümlü bir tam çizge kalmayana kadar devam eder (Algoritma 1 – Adım 1-11). Daha sonra ağdaki 4 veya daha fazla düğüm içeren çevre çizgeler araştırılır. Bu doğrultuda boş bir yığın (S) ile başlanarak DFS ağacı oluşturulur. Bunun için ağda henüz ziyaret edilmemiş bir v_i düğümü S'e eklenir ve ilgili düğüm ziyaret edildi olarak işaretlenir (Algoritma 1 - Adım 12-14). 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 1 - Adım 16-17). Eğer böyle bir komşu düğüm var ise (v_k), bu düğüm S yığınına eklenir, *parent[*v_k*]* değeri güncellenir ve ziyaret edildi olarak işaretlenir (Algoritma 1 - Adım 19-21). Eğer v_k düğümünün daha önce ağaca eklenmiş bir komşusu v_z var ise bu durumda *e(*v_k*,* v_z*)* kenarı kritik kenar olarak işaretlenir ve ağdan çıkarılır (Algoritma 1 - Adım 22-25). Eğer S yığınından çekilen v_j düğümünün ziyaret edilmemiş bir komşusu yoksa v_j düğümü S yığınından

çıkarılır (Algoritma 1 – Adım 26-28). Bu adımlar sonrası ilgili ağ herhangi bir çevre içermez, böylelikle Tree_Cut algoritmasının ilk aşaması tamamlanmış olur.

Algorithm 2: TreeCut- Aşama 2

Girdi: $G(V;E)$, k
Başlangıçta: (parent[], S) ← {}, ns[] ← 0
1: **while** (ziyaret edilmemiş bir düğüm v_i mevcutsa) **do**
2: push(v_i, S)
3: v_i'yi ziyaret edildi olarak işaretle
4: **while** (S boş değilse) **do**
5: v_j=peek(S)
6: v_k= v_j'nin ziyaret edilmemiş bir komşusu
7: **if** (v_k mevcutsa) **then**
8: push(v_k, S)
9: parent[v_k]=v_j
10: v_k'yi ziyaret edildi olarak işaretle
11: **else**
12: ns[parent[v_j]]=ns[parent[v_j]]+ns[v_j]+1
13: pop(S)
14: **end if**
15: **end while**
16: **end while**
17: **for** G'deki her kenar $e(v_i, v_j)$ için **do**
18: **if** (parent[v_i]==v_j) **then**
19: pr=v_j
20: **else**
21: pr=v_i
22: **end if**
23: p=v_i
24: **while** (parent[p] != 0) **do**
25: p=parent[p]
26: **end while**
27: deger1=ns[p]-ns[pr]
28: deger2=ns[pr]
29: deger3= ns[p]*(ns[p]-1)/2
30: deger4= deger1*(deger1-1)/2
31: value4= deger4+deger2*(deger2-1)/2
32: fark=deger3-deger4
33: skor= $e(v_i, v_j)$/fark
34: **if** (min>skor) **then**
35: min=skor
36: $critic\_e1 = v_i$
37: $critic\_e2 = v_j$
38: **end if**
39: **end for**
40: $G = G \setminus \{e(critic\_e1, critic\_e2)\}$

Şekil 5. Tree_Cut Algoritmasının Sözde Kodu – Aşama 2

Şekil 5'te Tree_Cut algoritmasını ikinci aşamasının sözde kodu verilmiştir. Algoritmaya boş bir yığın (S) ile başlanarak DFS ağacı oluşturulur. Öncelikle ağda henüz ziyaret edilmemiş bir v_i düğümü S'e eklenir ve ilgili düğüm ziyaret edildi olarak işaretlenir (Algoritma 2 - Adım 2-3). 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 5-6). Eğer böyle bir komşu düğüm var ise (v_k), bu düğüm S yığınına eklenir, *parent[*v_k*]* değeri güncellenir ve ziyaret edildi olarak işaretlenir (Algoritma 2 - Adım 8-10). Eğer S

yığınından çekilen v_j düğümünün ziyaret edilmemiş bir komşusu yoksa *parent*[v_j]'nin *ns*[*parent*[v_j]] değeri güncellenir ve v_j düğümü *S* yığınından çıkarılır. (Algoritma 2 – Adım 12-13). (Burada *ns*[*v*] değeri *v* düğümünü kök kabul eden alt ağaçtaki düğümlerin (*v*'nin torunları+1) sayısıdır.) Bu işlemler *S* yığını boş olana kadar devam eder (Algoritma 2 – Adım 4-15). Eğer *S* yığını boş ise ve *G* çizgesinde ziyaret edilmemiş bir düğüm var ise bu düğüm *S* yığınına eklenir ve yukarıda açıklanan bu işlemler tekrarlanır (Algoritma 2 – Adım 1-16).

G çizgesindeki tüm düğümler ziyaret edildikten sonra kritik kenar tespiti işlemine geçilir. Bunun için öncelikle ağdaki bir kenarın $e(v_i, v_j)$ uç düğümlerinden diğerinin ebeveyni olan düğüm (*pr*) seçilir (Algoritma 2 – Adım 18-23). Daha sonra bu düğümlerin bulunduğu ağacın kökü (*p*) bulunur (Algoritma 2 – Adım 24-26). Sonraki adımda *e(vi, vj)*'nin ağaçtan çıkması ile oluşan 2 bileşenin boyutları hesaplanır ve bu değerler *deger1* ve *deger2*'de saklanır (Algoritma 2 – Adım 27-28). *Deger3*'e, $e(v_i, v_j)$'in ağaçtan çıkmadığı durumundaki KKP amaç fonksiyonu değeri ve *deger4*'e, $e(v_i, v_j)$'in ağaçtan çıktığı durumdaki KKP amaç fonksiyonu değeri atanır (Algoritma 2 – Adım 29-31). Sonrasında *deger3- deger4* farkı alınarak $e(v_i, v_j)$ kenarının KKP amaç fonksiyonuna getireceği kazanç (*dif*) hesaplanır (Algoritma 2 – Adım 32). Daha sonra, $e(v_i, v_j)$ kenarının değeri *dif* değerine bölünerek $e(v_i, v_j)$ kenarının *score*'u hesaplanır (Algoritma 2 – Adım 33). *score* değeri ile hem bir kenarın amaç fonksiyonuna etkisi hem de silinecek kenar sayısına etkisi dikkate alınmış olur. *score* değeri en düşük olan kenar, kritik kenar olarak seçilir (Algoritma 2 – Adım 34-38). Bu işlemler tüm kenarlar için tekrarlanır (Algoritma 2 – Adım 17-39). Son olarak, seçilmiş olan kritik kenar ağdan çıkarılır (Algoritma 2 – Adım 40) ve bu işlemler silinen kenarların toplam değeri *k*'dan küçük olduğu sürece devam eder.

Karmaşıklık analizi

Teorem 1: Tree_Cut algoritmasının zaman karmaşıklığı $O(n^2\Delta^2)$'dir.

İspat: Algoritmanın ilk aşamasında ağdaki çevreler tespit edilip, ağ bir ağaca dönüştürülmektedir. Bu doğrultuda ilk olarak ağdaki 3 düğümlü tam çizgeler tespit edilmektedir. Ağdaki 3 düğümlü bir tam çizgenin tespiti işlemi $O(n\Delta^2)$ zaman gerektirmektedir. Her bir birleştirme işlemi sonrasında ağdaki düğüm sayısı 1 azalacağı için ağda en fazla *n-1* adet 3 düğümlü tam çizge tespit edilebilir. Dolayısıyla, ağdaki tüm 3 düğümlü tam çizgelerin tespiti işlemi toplamda $O(n^2\Delta^2)$ zaman gerektirmektedir. Sonraki adım 4 ya da daha fazla düğüm içeren çevrelerin tespiti için DFS ağacı kullanılmaktadır ve bu işlemde $O(n + m)$ zaman gerektirir. Dolayısıyla algoritmanın ilk aşamasının toplam zaman karmaşıklığı $O(n^2\Delta^2 + n + m) = O(n^2\Delta^2)$. Algoritmanın ikinci aşamasında öncelikle bir DFS ağacı (ağaçları) oluşturulur ve sonrasında kritik kenar tespiti işlemi yapılır. Dolayısıyla ikinci aşama $O(n + m)$ zaman gerektirmektedir. İkinci aşama *k* kere tekrarlanabileceği için toplamda $O(k(n + m)) = O(km) = O(kn^2)$ zaman gerektirmektedir. Algoritmanın tüm aşamaları dikkate alındığında Tree_Cut algoritmasının zaman karmaşıklığı $O(n^2\Delta^2) + O(kn^2) = O(n^2\Delta^2)$'dir.

BULGULAR VE TARTIŞMA

Hesapsal Sonuçlar

Bilindiği kadarı ile KKP için literatürde bir örnek kütüphane bulunmamaktadır. Bu sebeple KKP için geliştirilen Tree_Cut algoritmasının performans analizi için küçük boyutlu rastgele çizgeler üretilmiş ve bu örneklerin optimal değerleri kaba-kuvvet yöntemi ile bulunmuştur. Oluşturulan kütüphanede düğüm sayıları 10, 15 ve 20 arasında değişmektedir. Her bir düğüm sayısı için çizgenin yoğunluk (ağdaki mevcut kenar sayısının ağda olabilecek tüm maksimum kenar sayısına oranı) değeri 0.2, 0.21, 0.22, 0.23, 0.24 ve 0.25 (ilgili yoğunluk oranları hem ağların bağlantılılığı hem de gerçek hayat örnekleri dikkate alınarak belirlenmiştir) olmak üzere 6 farklı örnek üretilmiştir. Dolayısıyla,

oluşturulan kütüphane toplamda 18 örnek içermektedir. Üretilen örneklerin boyutları dikkate alınarak k değeri ağdaki düğüm sayısının %10'u olarak belirlenmiştir. Geliştirilen algoritmanın hata oranları eşitlik 1'deki formül ile hesaplanmıştır.

$$Hata\ Oranı = \left(\frac{bulunan\ \text{çözüm} - eniyi\ \text{çözüm}}{eniyi\ \text{çözümü}}\right) * 100 \quad (1)$$

Çizelge 1. KKP optimal değerleri ve Tree_Cut algoritmasının sonuçları

n	*d*	*Optimal*	*Tree_Cut*	*Hata Oranı*
10	0.2	20	20	0
10	0.21	20	20	0
10	0.22	20	20	0
10	0.23	20	21	0.05
10	0.24	20	21	0.05
10	0.25	14	14	0
15	0.2	43	43	0
15	0.21	49	51	0.041
15	0.22	49	51	0.041
15	0.23	67	67	0
15	0.24	78	79	0.013
15	0.25	78	78	0
20	0.2	136	136	0
20	0.21	153	153	0
20	0.22	136	136	0
20	0.23	122	122	0
20	0.24	136	136	0
20	0.25	153	153	0

Çizelge 1'de oluşturulan kütüphane örnekleri üzerinde sırasıyla ağdaki düğüm sayısının %10'luk k değerleri için KKP'nin optimal amaç fonksiyonu değerleri ve Tree_Cut algoritmasının sonuçları verilmiştir. İlgili çizelgede verilen örnek çizge üzerinde tespit edilen kritik kenarlar çıkarıldıktan sonra bağlantılı kalan (aralarında bir yol bulunan) düğüm ikililerinin sayısı verilmiştir. Bu değer ne kadar küçük ise, çizgede birbirleri ile bağlantısı bulunmayan düğümlerin sayısı o kadar fazladır.

Çizelge 1 incelendiğinde Tree_Cut algoritmasının 13 örnekte optimal çözümü bulduğu görülmektedir. Tüm örnekler dikkate alındığında Tree_Cut algoritmasının ortalama hata oranı 0.011'dir.

SONUÇ

Bu çalışmada, kentsel trafik ağları üzerinde kritik kenarların belirlenmesi için yeni bir polinom zamanlı algoritma önerilmiştir. Önerilen algoritma, ilgili ağı ağaç yapısına dönüştürerek ve kenarları bağlantı üzerindeki etkilerine göre stratejik olarak seçerek, ölçeklenebilirlik ve hesaplama maliyeti ile ilgili sınırlamalarını aşmaktadır. Hesapsal sonuçlar, önerilen algoritmanın küçük boyutlu ağlar üzerinde minimal hata ile neredeyse optimal çözümler elde etme yeteneğini göstermektedir. Dolayısıyla Tree_Cut algoritması trafik yoğunluğunu azaltma, acil durum müdahale sürelerini kısaltma ve şehir planlamasında stratejik kararlar alınmasını gibi önemli alanlarda kullanılabilir. Bu araştırmanın bulguları, trafik ağı analiz metodolojilerinin ilerlemesine katkıda bulunarak, daha sağlam ve dayanıklı ulaşım sistemlerinin yolunu açmaktadır. Gelecek araştırmalar Tree_Cut algoritmasının farklı türde ağlara uygulanabilirliğinin incelenmesi gibi konuları içerebilir.

TEŞEKKÜR

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

Çıkar Çatışması

Makale yazarları aralarında herhangi bir çıkar çatışması olmadığını beyan ederler.

Yazar Katkısı

Yazarlar makaleye eşit oranda katkı sağlamış olduklarını beyan eder.

KAYNAKLAR

Bazgan, C., Toubaline, S., & Vanderpooten, D. (2013). Critical edges for the assignment problem: Complexity and exact resolution. *Operations Research Letters*, *41*(6), 685-689.

Brinkmeier, M. (2007). A simple and fast min-cut algorithm. *Theory of Computing Systems*, *41*(2), 369-380.

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

Dinh, T.N., & Thai, M.T. (2011). Precise structural vulnerability assessment via mathematical programming. In Military Communications Conference, 2011-MILCOM 2011, IEEE (pp. 1351–1356).

Eren, T., & Akdaş, E. (2023). Afet ve acil durum yönetiminde arama kurtarma ekiplerinin oluşturulması. Afet ve risk dergisi, 6(3), 1060-1073.

Gauthier, P., Furno, A., & El Faouzi, N. E. (2018). Road network resilience: how to identify critical links subject to day-to-day disruptions. Transportation research record, 2672(1), 54-65.

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.

Houck, D.J., Kim, E., O'Reilly, G.P., Picklesimer, D.D., & Uzunalioglu, H. (2004). A network survivability model for critical national infrastructures. *Bell Labs Technical Journal, 8*(4), 153–172.

Jenelius, E., Petersen, T., & Mattsson, L.G. (2006). Importance and exposure in road network vulnerability analysis. *Transportation Research Part A: Policy and Practice, 40*(7), 537–560.

Jilani, U., Asif, M., Zia, M. Y. I., Rashid, M., Shams, S., & Otero, P. (2023). A systematic review on urban road traffic congestion. Wireless Personal Communications, 1-29.

Karimi, H., Ghadirifaraz, B., Shetab Boushehri, S. N., Hosseininasab, S. M., & Rafiei, N. (2021). Reducing traffic congestion and increasing sustainability in special urban areas through one-way traffic reconfiguration. Transportation, 1-24.

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

Minh, Q. T., Le Hoang, H. N., & Nhat, M. N. (2022). Effective traffic routing for urban transportation capacity and safety enhancement. IATSS Research, 46(4), 574-585.

Nagamochi, H., Ono, T., & Ibaraki, T. (1994). Implementing an efficient minimum capacity cut algorithm. *Mathematical Programming, 67*, 325–341.

Pana-Micu, F. (2024). Graph theory algorithms in optimizing urban infrastructure in smart cities. In Proceedings of the Central and Eastern European eDem and eGov Days 2024 (pp. 125-129).

Rashmi, R., Champawat, S., Varun Teja, G., & Lavanya, K. (2020). Analysis of road networks using the louvian community detection algorithm. In Soft Computing for Problem Solving: SocProS 2018, Volume 2 (pp. 749-757). Springer Singapore.

Salmeron, J., Wood, K.R., & Baldick, R. (2004). Analysis of electric grid security under terrorist threat. *IEEE Transactions on Power Systems, 19*(2), 905–912.

Smith, J.C., Prince, M., & Geunes, J. (2013). Modern network interdiction problems and algorithms. In Handbook of combinatorial optimization (pp. 1949-1987).

Smith, J.C., & Song, Y. (2019). A survey of network interdiction models and algorithms. *European Journal of Operational Research, 283*(3), 797-811.

Stoer, M., & Wagner, F. (1997). A simple min-cut algorithm. *Journal of the ACM, 44*(4), 585–591.

Tekgöz, H., Kolukısa, N., & Karabatak, M. (2022). Trafik Işıklarının Optimum Planlaması için Çizge Tabanlı Çözüm Önerisi. Fırat Üniversitesi Mühendislik Bilimleri Dergisi, 34(1), 15-23.

Walteros, J.L., & Pardalos, P.M. (2012). Selected topics in critical element detection. In Applications of mathematics and informatics in military science (pp. 9-26).

Yu, E. Y., Chen, D. B., & Zhao, J. Y. (2018). Identifying critical edges in complex networks. *Scientific reports*, *8*(1), 14469.