SAÜ Fen Bil Der 19. Cilt, 2. Sayı, s. 213-219, 2015

# Tek kaynaktan çıkan maksimum sayıdaki tepe ayrık yolların bulunması probleminin sayımlama tekniği ile etkin çözümü

Murat Erşen Berberler [1*], Zeynep Nihan Berberler[2]

*22.07.2014 Geliş/Received, 26.03.2015 Kabul/Accepted*

**ÖZ**

Tepe ve ayrıt olmak üzere iki türe ayrılan ayrık yolların bulunması problemi ile gerçek zamanlı iletişim, çok geniş ölçekli tümleşim, çizelgeleme, bidon paketleme ve yük dengeleme gibi birçok yöneylem araştırması probleminde alt problem olarak karşılaşılmaktadır. Bu çalışmada uygulama alanlarının bolluğu nedeniyle çok önemli bir yere sahip olan tepe ayrık yolların bulunması probleminin NP_tam karmaşıklık sınıfına ait eniyileme (optimizasyon) versiyonu ele alınacaktır. Ait olduğu problem sınıfının zorluğundan dolayı sezgisel algoritmalar ile yaklaşık çözümler üretilerek üstesinden gelinmeye çalışılan bu probleme tam ve etkin bir çözüm getirebilmek için sayımlama tekniğine dayanan bir algoritma önerilecek ve yöntemin ayrıntılı analizi yapılacaktır.

**Anahtar Kelimeler:** tepe ayrık yollar, maksimum bağımsız küme, sayımlama tekniği

# Finding an efficient solution for the maximum number of single source node disjoint paths by enumeration

**ABSTRACT**

The problem of finding node or edge disjoint paths is encountered as a sub problem in many operations research problems such as real time communication, very large scale integration, scheduling, bin packing and load balancing etc. In this paper, due to the majority of application fields, the optimization version for the problem of finding node disjoint paths related to NP_complete class is handled. An algorithm is proposed based on enumeration technique in order to find an exact and efficient solution for the problem of finding maximum number of single source node disjoint paths that is recently tried to overcome by generating approximate solutions by heuristic algorithms due to the complexity class that it belongs to and the method is analyzed in detail.

**Keywords**: node disjoint paths, maximum independent set, enumeration technique

* Sorumlu Yazar / Corresponding Author
1 Dokuz Eylül Üniversitesi, Fen Fakültesi, Bilgisayar Bilimleri Bölümü, İzmir - murat.berberler@deu.edu.tr
2 Dokuz Eylül Üniversitesi, Fen Fakültesi, Bilgisayar Bilimleri Bölümü, İzmir - zeynep.berberler@deu.edu.tr

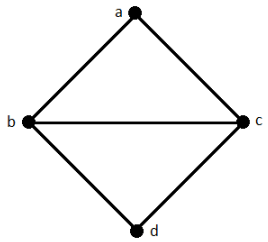
## 1. GİRİŞ (INTRODUCTION)

Ayrık yolların bulunması problemi iletişim ağlarında önemli bir rol oynamaktadır, çünkü iletişimde olan herhangi iki tepe arasında ayrıt veya tepe ortaklığı olmayan yollar etkin bir şekilde bulunabilirse bu tepeler arasındaki veri trafiği alternatif yollar üzerinden paylaştırılarak bant genişliği değeri korunabilir [1]. Eğer ağ üzerinde sadece ayrıt ayrık yolların bulunması problemi ile ilgilenilirse Menger teoremine göre tüm ayrıtların ağırlıkları bir birim alınarak en büyük akış problemini çözen bir algoritma yardımıyla çok terimli (polinom) zamanda sonuca ulaşılabilir [2]. Ancak ayrıt ayrık yollar, ağın yapısına bağlı olarak bazı tepeleri ortak kullanabileceğinden bu tepelerde oluşabilecek bir arıza nedeniyle sistemin önemli bir bölümü etkilenecek ve özellikle bant genişliği gibi iletişim kalitesini arttıracak yaklaşımların etkisi büyük oranda azalacaktır. Her ne kadar ayrıt ayrık yollar çok işlem yükü gerektirmediği için gerçek zamanlı iletişimde tercih edilse de ortak tepe kullanma dezavantajından dolayı statik yapılarda tercih edilmeyip onun yerine tepe ayrık yollar kullanılmaktadır.

Tepe ayrık yolların kullanılması iletişim kalitesini korumayı maksimum seviyede tutmakla birlikte bu yolların bulunması problemi NP-tam karmaşıklık sınıfına aittir [3]. Problemin ait olduğu sınıfın zorluğundan dolayı uygulamada genellikle sezgisel algoritmalar yardımıyla bu probleme yaklaşık çözümler aranmakta [4] olup uygulamada ağ yönlendirme ve güvenli iletişim gibi öyle problemler [5], [6] vardır ki kesin çözüme ihtiyaç duyulduğu için yaklaşık çözümler kabul edilemez. Bunun dışında tepe ayrık yolların bulunması problemi özel ağ türleri üzerinde çözülerek NP-tam karmaşıklık sınıfından P karmaşıklık sınıfına kaydırılmakla [7],[8],[9],[10] birlikte bu çalışmalar uygulamada karşılık bulmamaktadır. Çünkü gerçek dünyada yer alan iletişim ağlarının çok büyük bir bölümü kaotik yapıda olup atıf yapılan çalışmalarda belirtildiği oranda basit bir düzen içermemektedir.

## 2. YAZIN TARAMASI (LITERATURE REVIEW)

Maksimum sayıda tepe ayrık yolların bulunması probleminin tam çözümüne dair bilinen yegâne algoritma bitişiklik matrisinin kuvvetlerinin hesaplanmasına dayanmaktadır. Herhangi bir çizgeye ait $A$ bitişiklik matrisi bir uzunluklu yürüyüşleri vermekte olup, matrisin $k$. kuvveti hesaplandığı takdirde $k$ uzunluklu yürüyüşler elde edilebilir [11]. Yürüyüş rotaları içinden tekrar eden ayrıtlar çıkarıldığında zincirler, tekrar eden tepeler çıkarıldığında da yollar bulunmuş olur. Yöntemin detaylı analizi için Şekil 1'de görülen örnek kullanılacaktır.




Şekil 1. Örnek problem (The sample problem)

Yöntemin ilk adımında 0 ve 1 değerlerini kullanarak bitişik tepeleri gösteren $A$ matrisi tepelerin etiketlerini gösterecek şekilde değişikliğe uğratılır ve benzer şekilde çarpma işleminde değişmeden kullanılacak olan $A'$ baz matrisi oluşturulur. Yöntemin sonraki adımlarında $A$ matrisinin kuvvetleri hesaplanır, ancak burada yapılacak matris çarpımında çarpma ve toplama işlemleri yerine karakter veri yapısına uygun olarak birleştirme(arka arkaya ekleme) ve listeleme(virgül ile ayırma) operatörleri kullanılır. Çarpma işlemine ait yinelemeli formül $A^k = A^{k-1}A'$ şeklinde olup yöntemin örnek problem üzerindeki çıktıları Şekil 2'de görülmektedir.

A =

| | a | b | c | d |
|---|---|---|---|---|
| a | 0 | 1 | 1 | 1 |
| b | 1 | 0 | 1 | 0 |
| c | 1 | 1 | 0 | 1 |
| d | 1 | 0 | 1 | 0 |

A =

| | a | b | c | d |
|---|---|---|---|---|
| a | - | ab | ac | ad |
| b | ba | - | bc | - |
| c | ca | cb | - | cd |
| d | da | - | dc | - |

A' =

| | a | b | c | d |
|---|---|---|---|---|
| a | - | b | c | d |
| b | a | - | c | - |
| c | a | b | - | d |
| d | a | - | c | - |

$A^2$ =

| | a | b | c | d |
|---|---|---|---|---|
| a | - | acb | abc, adc | acd |
| b | bca | - | bac | bad, bcd |
| c | cba, cda | cab | - | cad |
| d | dca | dab, dcb | dac | - |

$A^3$ =

| | a | b | c | d |
|---|---|---|---|---|
| a | - | adcb | - | abcd |
| b | bcda | - | badc | bcad, bacd |
| c | - | cdab | - | cbad |
| d | dcba | dcab, dacb | dabc | - |

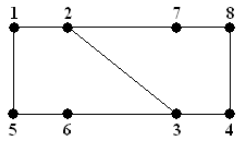
Şekil 2. Yöntemin örnek problem üzerindeki çıktıları (Outputs of the method for the sample problem)

Bu yaklaşım yardımıyla tepe ayrık yolları bulmanın bazı dezavantajları vardır. $n$ adet tepeye sahip bir çizgedeki tüm yolları bulmak için $n-1$ adet matris çarpma işlemi gerekir, ayrıca hızdan bir miktar kazanmak adına tüm matrisler bellekte tutulursa $n\ x\ O(n^2)\ x\ O(n^2) = O(n^5)$ gibi bir bellek karmaşıklığı ile karşılaşılır. Formülde yer alan $n$ değeri saklanması gereken $n$ adet matrisi temsil etmektedir, ilk $O(n^2)$ karmaşıklığı matrislerin herhangi bir hücresindeki karakter veri yapısının kapladığı yer olup ikinci $O(n^2)$ karmaşıklığı ise her bir matrisin $n\ x\ n$ boyutundan gelen maliyettir. Dolayısıyla problem bu yaklaşımla çözülmek istendiğinde $n$ tepe sayısının büyük değerleri için pratikte bellek problemleri çıkacağı açıkça görülmektedir. Öte yandan sadece iterasyon formülünde kullanılacak olan $A^k$, $A^{k-1}$, $A'$ matrislerini bellekte tutarak bir önceki yaklaşıma nazaran daha az olan $3\ x\ O(n^2)\ x\ O(n^2) = O(n^4)$ bellek karmaşıklığı ile bu kez yürütme zamanından büyük ölçüde ödün verilecektir, çünkü her adımda matrislerin içeriklerinin birbirlerine aktarılması gerekecektir. Bu yöntemin diğer dezavantajı da zaman karmaşıklığının yüksek olmasıdır, çünkü $n\ x\ n$ boyutlu iki matrisi çarpmanın maliyetinin $O(n^3)$ olduğu göz önüne alınırsa $n-1$ adet çarpma işlemi yapılacağından $(n-1)\ x\ O(n^3) = O(n^4)$ gibi bir zaman karmaşıklığı sadece çarpmanın maliyetidir. Her bir çarpma işlemi sonucunda matrisin hücreleri içerisinde tekrar eden tepelerin ayıklanması için de $O(n^2)$ 'lik bir işlem maliyeti olduğu göz önüne alınırsa toplam zaman karmaşıklığı $O(n^4)\ x\ O(n^2) = O(n^6)$ olarak bulunur.

Maksimum sayıda tepe ayrık yolların bulunması problemi için önerilen sezgisel algoritmalar en kısa yol problemine çözüm getiren algoritmaları baz almaktadır [4]. Söz konusu iki tepe arasında rota bilgisiyle birlikte bir adet en kısa yol bulunur ve bu yola ait tepe ve ayrıtlar çizgeden atılır, ardından geriye kalan çizge için yeni bir en kısa yol bulunur. Bu işlem iki tepe arasında yol bulunamayıncaya kadar devam eder. Eksik rota bulma potansiyeli yüksek olan bu sezgisel yaklaşımın dezavantajları üçüncü bölümde örnek verilerek ele alınacaktır.

## 3. ÇÖZÜM YAKLAŞIMI (SOLUTION APPROACH)

Sezgisel yöntemlerin düzeltme kullanmazsa genelde yanlış sonuç bulduğu ve önerilecek algoritmanın güçlü yönlerinin ön plana çıkarılabileceği örnek bir problem Şekil 3'de görülmektedir.




Şekil 3. Örnek problem (The sample problem)

Bu örnekte 1 numaralı tepeden diğer tüm tepelere maksimum sayıdaki tepe ayrık yolların bulunması istenseydi sorun yaratan durum büyük bir olasılıkla 1 ve 4 tepeleri arasında oluşacaktı. Çünkü problemi çözmek için önerilen açgözlü sezgisel algoritmalar genellikle en kısa yol problemini çözen algoritmaları baz aldığı için 1-2-3-4 rotası aç gözlü sezgisel tarafından kullanılarak problemin çözümü için toplamda 1 rota verilecekti. Oysa problemin en iyi çözümü 2 olup, bunlar 1-2-7-8-4 ve 1-5-6-3-4 rotalarıdır.

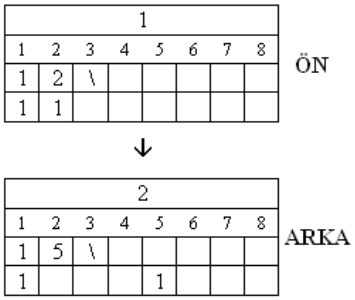
Bu çalışmada önerilecek algoritmanın etkin olabilmesi için tüm veri yapıları bellek karmaşıklığı anlamında alt limitlerde seçilmiştir. Örneğin tepelerin birbirleri arasındaki bağlantı bilgisi düzensiz dizi mantığına dayanan bir veri yapısında tutulmuş olup örnek probleme ait yapı Şekil 4'te verilmiştir.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 1 | 2 | 2 | 5 | |
| 2 | 3 | 1 | 3 | 7 |
| 3 | 3 | 2 | 4 | 6 |
| 4 | 2 | 3 | 8 | |
| 5 | 2 | 1 | 6 | |
| 6 | 2 | 3 | 5 | |
| 7 | 2 | 2 | 8 | |
| 8 | 2 | 4 | 7 | |

Şekil 4. Düzensiz dizi veri yapısı (Jagged array data structure)

Şekilde sol baştaki kolon tepe numaralarını içermekte olup, onun hemen yanında yer alan ve sıfır indisi ile belirtilen kolon ilgili tepenin kaç adet bağlantısı olduğunu belirtmektedir. Sıfır indisli kolon sayesinde klasik bitişiklik matrisi kullanımında karşılaşılan ve çalışma zamanını olumsuz etkileyen bir yaklaşım olan bağlantısı olmadığı halde matris hücrelerinin gereksiz yere dolaşılması durumu burada söz konusu olmamaktadır. 1, 2 ve 3 indisleri ile belirtilen kolonların satırlardaki tepe numaraları ile kesiştiği hücrelerde ise ilgili tepenin bağlantılı olduğu tepeler yer almaktadır.

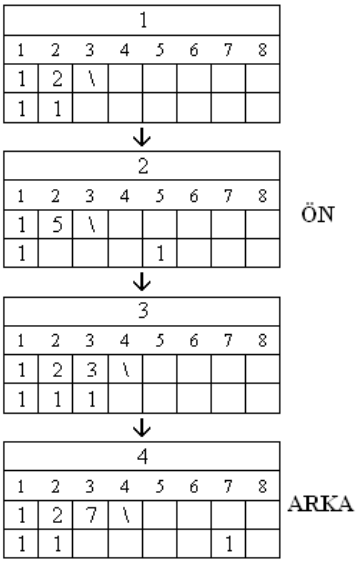
Algoritmanın ilk yaptığı işlem, başlangıç tepesine ait satıra konumlanarak bu tepeye bitişik tepeleri indis sırasına göre kuyruğa eklemektir. Örnek problem için başlangıç tepesi olarak 1 numaralı tepe seçilirse algoritma ilk adımda Şekil 5'te verilen kuyruk yapısını oluşturacaktır.




Şekil 5. Algoritmanın ilk adımı çalıştıktan hemen sonra kuyruğun durumu (The status of the queue immediately after the execution of the first step of the algorithm)

Kuyruğun her bir elemanının içeriğinde; kuyrukta yer aldığı sıra numarası (1. satır), karakter dizisi (3. satır) ve tepelerin kullanılma durumunu gösteren tepe numaraları vektörü (4. satır) öğeleri yer almaktadır. Şekilde görülen 2. satır sembolik olup indisleri belirtmektedir ve gerçek yapıda yer almamaktadır. Kuyruğun 1 numaralı elemanına ÖN ve 2 numaraları elemanına ARKA işaretçi atamaları yapılarak algoritmanın ilk adımı tamamlanır.

Algoritmanın yoğun bir şekilde yineleme yaptığı ve kuyruğun tamamlanması işlemini gerçekleştirdiği ikinci adımı, ÖN işaretçisi ARKA işaretçisinden farklı olduğu sürece tekrarlanır. İlk etapta ÖN işaretçisinin temsil ettiği elemana ait karakter dizisinin son üyesine bitişik olan tepeler kuyruğa eklenir ve eğer bitişik tepelerden herhangi birisi daha önceden kullanılmış bir tepe ise problemin tanımı gereği (tepe ayrık) göz ardı edilir. Örneğin birinci düğümü gösteren ÖN işaretçisinin gösterdiği karakter dizisinin son elemanı olan 2 tepesine bitişik olan tepeler 1, 3 ve 7 sırasıyla Şekil 4'te verilen veri yapısından çekilir. Ardından tepe numaraları vektöründe 1 tepesi kullanımda olduğu için sadece 3 ve 7 tepeleri kuyruğa eklenerek ÖN işaretçisi için ikinci düğüm ve ARKA işaretçisi için de dördüncü düğüm güncellemeleri yapılır. Kuyruğa yapılan bu eklemeler ve güncellemelerden sonraki durum Şekil 6'da görülmektedir.



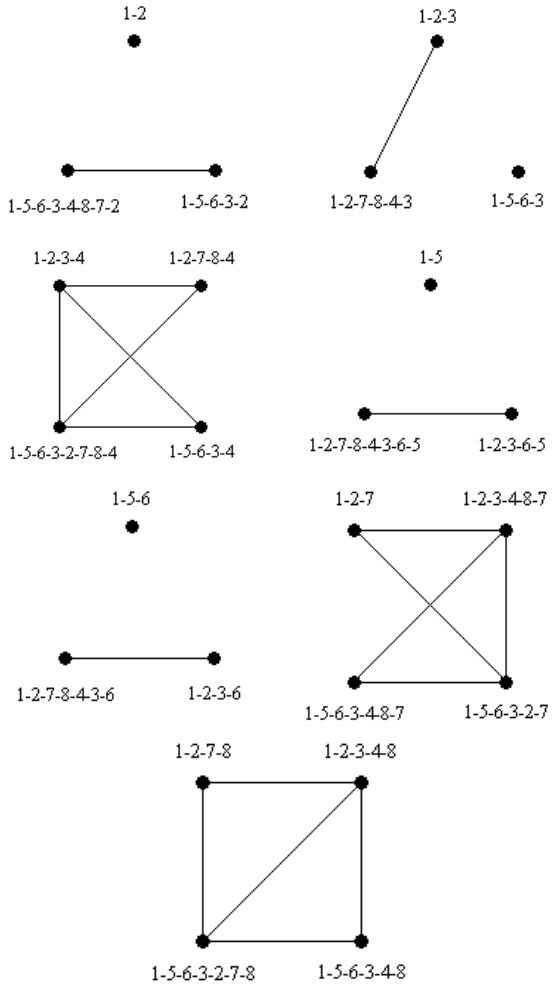

Şekil 6. Algoritmanın ikinci adımı çalıştıktan sonra kuyruğun durumu (The status of the queue after the execution of the second step of the algorithm)

Algoritmada kuyruğun oluşturulması adımı ÖN ve ARKA işaretçileri eşitlendiğinde sonlanır. Ele alınan örnek problem için toplam 24 adımda tamamlanan kuyruk Şekil 7 'de görüldüğü gibi olacaktır.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | \ | | | | | |
| 2 | 1 | 5 | \ | | | | | |
| 3 | 1 | 2 | 3 | \ | | | | |
| 4 | 1 | 2 | 7 | \ | | | | |
| 5 | 1 | 5 | 6 | \ | | | | |
| 6 | 1 | 2 | 3 | 4 | \ | | | |
| 7 | 1 | 2 | 3 | 6 | \ | | | |
| 8 | 1 | 2 | 7 | 8 | \ | | | |
| 9 | 1 | 5 | 6 | 3 | \ | | | |
| 10 | 1 | 2 | 3 | 4 | 8 | \ | | |
| 11 | 1 | 2 | 3 | 6 | 5 | \ | | |
| 12 | 1 | 2 | 7 | 8 | 4 | \ | | |
| 13 | 1 | 5 | 6 | 3 | 2 | \ | | |
| 14 | 1 | 5 | 6 | 3 | 4 | \ | | |
| 15 | 1 | 2 | 3 | 4 | 8 | 7 | \ | |
| 16 | 1 | 2 | 7 | 8 | 4 | 3 | \ | |
| 17 | 1 | 5 | 6 | 3 | 2 | 7 | \ | |
| 18 | 1 | 5 | 6 | 3 | 4 | 8 | \ | |
| 19 | 1 | 2 | 7 | 8 | 4 | 3 | 6 | \ |
| 20 | 1 | 5 | 6 | 3 | 2 | 7 | 8 | \ |
| 21 | 1 | 5 | 6 | 3 | 4 | 8 | 7 | \ |
| 22 | 1 | 2 | 7 | 8 | 4 | 3 | 6 | 5 | \ |
| 23 | 1 | 5 | 6 | 3 | 2 | 7 | 8 | 4 | \ |
| 24 | 1 | 5 | 6 | 3 | 4 | 8 | 7 | 2 | \ |

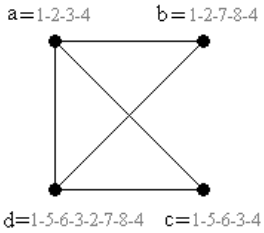
Şekil 7. Örnek problem için tamamlanmış kuyruk veri yapısı (Completed queue data structure for the sample problem)

Kuyruk tamamlandığında başlangıç tepesinden diğer tüm tepelere tepe ve ayrıt ortaklığı kısıtı olmaksızın tüm alternatif rotalar elde edilmiş olur. Bu noktada problemin tanımı gereği tekrar eden tepelerin ayıklanarak en çok sayıda alternatifin oluşturulması işleminin yapılması gerekmektedir. Algoritmanın ikinci ana bölümü bu işlemi yapmak üzere tasarlanmış olup sırasıyla her bitiş tepesi için o tepeye ait alternatif yollar kullanılarak bitişiklik matrisi oluşturulur. Bitişiklik matrisi oluşturulurken başlangıç ve bitiş tepesi haricinde tepe ortaklığı olan yollar bir ayrıtla bağlanır, bu durum Şekil 8'de görülmektedir.



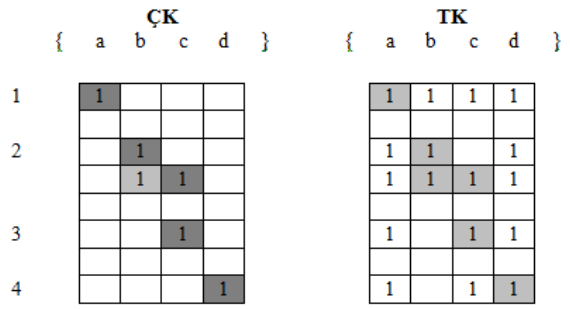

Şekil 8. 1 numaraları tepeden çıkan tüm yollar (All paths starting from node number 1)

Bu aşamadan sonra maksimum sayıdaki tepe ayrık yolların bulunması probleminin çözümü için NP-tam karmaşıklık sınıfına ait olan "Maksimum Bağımsız Küme Problemi" nin çözülmesi gerekmektedir. Maksimum bağımsız küme probleminin tam çözümünde detayları [12]'de belirtilen ve küme veri yapısını temel alan alt program kullanılacaktır. Bu alt programın çalışmasını kısaca açıklamak için Şekil 8'de yer alan 4 numaralı tepeye ait küçük problem ele alınabilir. Gösterim kolaylığı açısından 1-2-3-4 yolu a harfi, 1-2-7-8-4 yolu b harfi, 1-5-6-3-4 yolu c harfi ve 1-5-6-3-2-7-8-4 yolu da d harfi ile temsil edilsin, bu durum Şekil 9'da görülmektedir. Alt programda kullanılacak her bir tepeye ait küme tipindeki veri yapıları da ak={a,b,c,d}, bk={a,b,d}, ck={a,c,d} ve dk={a,b,c,d} olup ilgili tepenin bitişik olduğu tepeler kümenin elemanı seçilerek oluşturulmuştur.




Şekil 9. 4 numaralı tepe ile biten yollar (Paths ending with node 4)

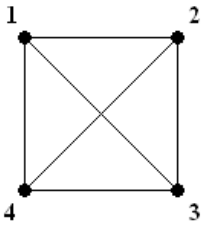
Sayımlama işlemi sırasında, çözüme giren tepeleri içeren "Çözüm Kümesi (ÇK)" ve çizgede yer alan tüm tepelerin kullanılıp kullanılmadığının kontrolü için de "Tepeler Kümesi (TK)" ele alınan temel veri yapılarıdır. Her adım başlarken ÇK ve TK boş küme olarak belirlenecek ve tüm tepeler TK kümesinde yer alana kadar yineleme devam edecektir. Alfabetik olarak öncelik a harfinde olduğundan a tepesini içeren {a} kümesi ÇK={} kümesiyle, bu tepeye ait olan ve komşulukları içeren ak kümesi de TK={} kümesi ile birleşim işlemine tâbi tutulur ve ÇK={a}, TK={a,b,c,d} sonuçları elde edilir. TK kümesi tüm tepeleri içerdiği için a tepesine ait olan yineleme tamamlanmış olur. Algoritmanın ilk adımı olduğu için mevcut ÇK en iyi sonuç olarak atanır ve b tepesinin kullanıldığı ikinci adıma geçilir. Bu adımda b tepesini içeren {b} kümesi ÇK={} kümesi ile ve bu tepeye ait olan bk kümesi de TK={} kümesi ile birleşim işlemine tabii tutularak ÇK={b}, TK={a,b,d} sonuçları elde edilir. Bu aşamada TK kümesi tüm tepeleri içermediğinden, yineleme TK içinde yer almayan ve sıradaki ilk eleman olan c tepesi ile devam ederek ÇK={b,c} ve TK={a,b,c,d} kümeleri elde edilir. Bu adımda bulunan ÇK kümesinin eleman sayısı rekor olarak tutulan bir önceki adımdakinden daha fazla olduğu için rekor 2 olarak güncellenir. Algoritma bu şekilde c ve d tepeleri için de çalışır ve değeri 2 olan rekordan daha büyük bir sayı bulunamadığından sonuçta ÇK={b,c} ve |ÇK|=2 elde edilir. b harfi ile 1-2-7-8-4 yolu ve c harfi ile de 1-5-6-3-4 yolu temsil edildiğinden 1 tepesinden çıkan ve 4 tepesinde son bulan tepe ayrık en çok iki yol olduğu ve bunların da 1-2-7-8-4 ve 1-5-6-3-4 oldukları elde edilmiş olur. Alt programın 4 numaralı tepe ile biten yollar için adım adım çalıştırılmasını gösteren Şekil 10 aşağıda görülmektedir.




Şekil 10. 4 numaralı tepeye ait örnek için algoritmanın adım adım çalıştırılması (The step by step execution of the algorithm for the sample of node 4)

## 4. ALGORİTMANIN ANALİZİ VE HESAPLAMA DENEMELERİ (ANALYSIS OF THE ALGORITHM AND COMPUTATIONAL EXPERIMENTS)

Algoritmanın analizini yapabilmek için Şekil 11'de görülen 4 tepeli tam çizge ele alınacaktır.

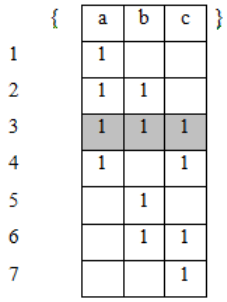



Şekil 11. 4 tepeli tam çizge (Complete graph with 4 nodes)

Bu örnek için kuyruğun oluşturulması tamamlandığında 1-2, 1-3, 1-4, 1-2-3, 1-2-4, 1-3-2, 1-3-4, 1-4-2, 1-4-3, 1-2-3-4, 1-2-4-3, 1-3-2-4, 1-3-4-2, 1-4-2-3, 1-4-3-2 olmak üzere toplamda 15 adet rota elde edilmektedir. Bu rotalar, 1 numaralı başlangıç tepesi ilk pozisyonda sabit kalmak koşuluyla 2, 3 ve 4 uzunluklu 1-X, 1-X-X, 1-X-X-X kalıpları kullanılarak olası tüm permütasyonların oluşturulmasıyla bulunmuştur. Sayısal olarak ta 1-X:1*3, 1-X-X:1*3*2, 1-X-X-X:1*3*2*1 değerleri hesaplanıp toplanırsa $P(3,1) + P(3,2) + P(3,3) = 3 + 6 + 6 = 15$ (1) 'deki formülün toplam rota sayısını verdiği açıkça görülmektedir.

$$\sum_{k=1}^{n-1} \frac{(n-1)!}{(n-k)!} \qquad (1)$$

Algoritmanın diğer önemli kısmını oluşturan ve maksimum bağımsız küme probleminin çözdürüldüğü alt program olan ikinci bölümünde ise 3 tepeli ve etiketleri {a,b,c} olan boş bir çizge üzerinde algoritma çalıştırıldığında Şekil 12'de görülen tablo elde edilir. Ele alınan çizge boş olduğu için hiçbir bağlantı içermeyeceğinden komşuluk bilgisini tutan ak={a}, bk={b} ve ck={c} küme veri yapıları sadece kendi etiketlerini içerecektir. Tablonun sol kolonunda görüleceği üzere boş küme hariç tüm alt kümelerin ele alındığı 7 adet durumun yineleme adımları bulunmaktadır ki bunun anlamı algoritmanın $2^n - 1$ formülü ile elde edilen 7 farklı alt kümeyi taradığını göstermektedir.




Şekil 12. 3 tepeli boş çizgeye ait veri yapısı (Data structure for empty graph with 3 nodes)

Tüm uygun çözümlerin oluşturulması ve bu çözümler içinden maksimum sayıda yolları verenlerin seçilmesi kısımlarından oluşan algoritma rastgele oluşturulmuş örnekler üzerinde çalıştırılmış ve sonuçlar Tablo 1'de paylaşılmıştır. Tablonun solunda yer alan $n$ kolonu problemin boyutunu yani ele alınan çizgedeki tepe sayısını, %20 ile %90 arasında değer alan kolonlar ise çizgedeki ayrıt yoğunluğunu göstermektedir. A kolonu bu çalışmada önerilen algoritmaya, B kolonu ise yazın taraması kısmında bahsedilen yönteme ait olup milisaniye cinsinden CPU sürelerini içermektedir.

Tablo 1. Hesaplama denemelerinin sonuçları (Results of the computational experiments)

| | **20%** | | **30%** | |
|---|---|---|---|---|
| **n** | **A** | **B** | **A** | **B** |
| 20 | 20 | 30 | 40 | 60 |
| 30 | 160 | 280 | 260 | 440 |
| 40 | 810 | 1310 | 1010 | 1980 |
| 50 | 2610 | 4210 | 3160 | 6180 |
| 60 | 6110 | 10260 | 8520 | 16240 |
| 70 | 15140 | 23220 | 20040 | 35520 |
| 80 | 29790 | 46890 | 40050 | 70370 |
| 90 | 52740 | 85800 | 66070 | 129520 |

| | **40%** | | **50%** | |
|---|---|---|---|---|
| **n** | **A** | **B** | **A** | **B** |
| 20 | 60 | 80 | 70 | 90 |
| 30 | 390 | 590 | 510 | 740 |
| 40 | 1810 | 2570 | 2230 | 3120 |
| 50 | 5780 | 8030 | 6950 | 10150 |
| 60 | 15840 | 22310 | 21060 | 29010 |
| 70 | 32230 | 47210 | 40030 | 59820 |
| 80 | 66210 | 94130 | 81570 | 118050 |
| 90 | 116890 | 174250 | 164450 | 219890 |

| | 60% | | 70% | |
|---|---|---|---|---|
| **n** | **A** | **B** | **A** | **B** |
| 20 | 80 | 110 | 90 | 120 |
| 30 | 710 | 910 | 830 | 1050 |
| 40 | 2820 | 3760 | 3250 | 4310 |
| 50 | 9450 | 12200 | 10720 | 14050 |
| 60 | 27850 | 36160 | 32360 | 42110 |
| 70 | 48670 | 61490 | 56880 | 74060 |
| 80 | 101320 | 132270 | 121660 | 156110 |
| 90 | 201910 | 265010 | 234310 | 309970 |

| | 80% | | 90% | |
|---|---|---|---|---|
| **n** | **A** | **B** | **A** | **B** |
| 20 | 110 | 140 | 130 | 150 |
| 30 | 970 | 1210 | 1160 | 1370 |
| 40 | 3940 | 4900 | 4650 | 5540 |
| 50 | 13170 | 16100 | 15080 | 17900 |
| 60 | 39580 | 48200 | 46340 | 54160 |
| 70 | 71240 | 86990 | 84910 | 98010 |
| 80 | 147850 | 180230 | 172340 | 204840 |
| 90 | 291640 | 355130 | 340060 | 401230 |

## 5. SONUÇ (CONCLUSION)

İki tepe arasında yer alan maksimum sayıdaki tepe ayrık yolların bulunması problemi NP-tam karmaşıklık sınıfındandır. Bu çalışmada problem bir adım öteye götürülerek bir tepeden diğer tüm tepelere maksimum sayıdaki tepe ayrık yolların bulunması şekline dönüştürülmüş ve sayımlama yöntemiyle probleme tam ve etkin bir çözüm getirebilmek için yeni bir algoritma önerilmiştir. Elde edilen sonuçlar önerilen algoritmanın bellek ve zaman karmaşıklığı ölçütleri bakımından yazın taraması bölümünde bahsedilen yöntemden daha etkin olduğunu göstermekte olup hesaplama denemelerinde kullanılan kaynak kodlar http://kisi.deu.edu.tr/murat.berberler/midp/ adresinde yer almaktadır.

## KAYNAKLAR (REFERENCES)

[1] Z. Xie, Z. Chen, H. Leng ve J. Zhang, «Finding Arc and Vertex-Disjoint Paths in Networks,» *Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, Chengdu, 2009.

[2] W. Kocay ve D. Kreher, Graphs, Algorithms, and Optimization, Florida: Chapman&Hall/CRC, 2005.

[3] K. R., Complexity of Computer Computations, New York: Plenum Press, 1972.

[4] M. Dahshan, «Maximum-Bandwidth Node-Disjoint Paths,» *(IJACSA) International Journal of Advanced Computer Science and Applications,* cilt 3, no. 3, pp. 48-56, 2012.

[5] D. Dolev, C. Dwork, O. Waarts ve M. Yung, «Perfectly secure message transmission,» *The Journal of the ACM (JACM),* cilt 40, no. 1, pp. 17-47, 1993.

[6] S. DaeHo ve M. Thottethodi, «Disjoint-path routing: Efficient communication for streaming applications Parallel & Distributed Processing,» *IEEE International Symposium*, Rome, 2009.

[7] L. Lipták, E. Cheng, J. Kim ve S. Kim, «One-to-many node-disjoint paths of hyper-star networks,» *Discrete Applied Mathematics,* cilt 16, no. 13-14, pp. 2006-2014, 2012.

[8] C. Lai, «Optimal Construction of All Shortest Node-Disjoint Paths in Hypercubes with Applications,» *IEEE Transactions on Parallel and Distributed Systems,* cilt 23, no. 6, pp. 1129-1134, 2012.

[9] Y. Xiang ve I. Stewart, «One-to-many node-disjoint paths in (n,k)-star graphs,» *Discrete Applied Mathematics,* cilt 158, no. 1, pp. 62-70, 2010.

[10] K. Kaneko ve Y. Suzuki, «Node-to-Set Disjoint Paths Problem in Pancake Graphs,» *IEICE TRANSACTIONS on Information and Systems,* Cilt E-86D, no. 9, pp. 1628-1633, 2003.

[11] J. Bondy ve U. Murty, Graph Theory with Applications, North Holland: Elsevier Science, 1982.

[12] E. Nasibov, M. Berberler ve C. Atılgan, «An Efficient Algorithm for Exact Solution of Maximum Independent Set Problem in Dense Graphs,» *3rd International Symposium on Computing in Science and Engineering*, Aydın, 2013.