

BURSA ULUDAĞ ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2024-2025 EĞİTİM ÖĞRETİM YILI GÜZ DÖNEMİ

ALGORİTMA ANALİZİ RAPORU

***Klasikten İleri Seviyeye***

***Dijkstra, A\**, *ALT ve Contraction Hierarchies* ile Kısa Yol Çalışması**

032290004 BARIŞ IŞIK

032290004@ogr.uludag.edu.tr

032290008 MURAT BERK YETİŞTİRİR

032290008@ogr.uludag.edu.tr

032290037 BUĞRA ÖZGEN

[032290037@ogr.uludag.edu.tr](mailto:032290037@ogr.uludag.edu.tr)

#### **1.0 Giriş**

Kısa yol algoritmaları, özellikle bilgisayar bilimi, mühendislik ve matematik alanlarında, bir graf üzerindeki iki nokta arasındaki en verimli yolu bulmak için kullanılan temel tekniklerden biridir. Navigasyon sistemlerinden lojistik yönetimine, veri ağlarından oyun geliştirmeye kadar birçok alanda bu algoritmalara ihtiyaç duyulmaktadır. Bu bölümde, algoritmaların genel tanımından yola çıkarak en kısa yol probleminin nasıl ortaya çıktığını ve neden bu kadar önemli olduğunu açıklayacağız. Kısa yol problemleri, belirli kısıtlamalar altında bir kaynaktan hedefe minimum maliyetle ulaşmayı hedefleyen bir optimizasyon problemidir. Bu tür problemler, doğru algoritmalar kullanıldığında, sistem verimliliğini artırarak kullanıcı deneyimini olumlu yönde etkileyebilir.

#### **1. 1 Algoritmanın Tanımı**

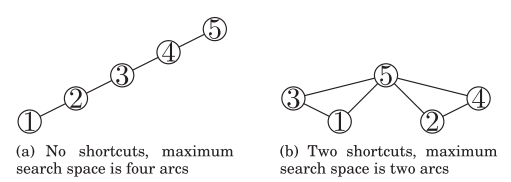
Algoritma, bir problemi çözmek veya belirli bir hedefe ulaşmak amacıyla izlenmesi gereken adımlardan oluşan sistematik bir yöntemdir. Algoritmalar, herhangi bir veriyi belirli işlemlerden geçirerek anlamlı çıktılar elde etmemizi sağlar. Temelde bir dizi sonlu adımdan oluşan algoritmalar, bir sorunu çözmek veya bir işlemi en verimli şekilde gerçekleştirmek için gereklidir. Bilgisayar bilimlerinde ve matematikte algoritmalar, karmaşık problemlerin çözümünü kolaylaştırarak belirli kural ve mantık çerçevesinde sonuç elde etmeyi mümkün kılar. Etkin bir algoritma, işlem süresini kısaltarak ve kaynak kullanımını azaltarak problemin en hızlı şekilde çözülmesini sağlar.

#### **1.2 En Kısa Yol Problemi**

En kısa yol problemi, graf teorisinin en temel problemlerinden biridir ve iki nokta arasında en düşük maliyetli veya en kısa mesafeli güzergahı bulmayı hedefler. Bu problem, hem pozitif ağırlıklı graf yapılarında hem de farklı türdeki grafiklerde karşımıza çıkabilir. Kısa yol problemleri; haritalama, ulaşım ağları ve telekomünikasyon gibi birçok gerçek dünyaya uyarlanmış uygulamada yaygın olarak kullanılmaktadır. Örneğin, bir navigasyon sisteminde, belirli bir başlangıç konumundan hedefe en kısa sürede ulaşmak için bu tür algoritmalar devreye girer. En kısa yol problemi, sadece optimum çözümleri değil, aynı zamanda verimliliği artıracak çözümleri bulmak için de kullanılır.

### **2.0 En Kısa Yol Problemine Genel Bakış**

En kısa yol algoritmaları, belirli bir başlangıç noktasından hedefe en hızlı veya en maliyetsiz şekilde ulaşmak için kullanılır. Klasik algoritmalar arasında Dijkstra, Bellman-Ford ve Floyd-Warshall gibi çözümler yer alır; ancak daha büyük veri setlerinde hızlı çözüm üretebilmek için A\* ve Contraction Hierarchies gibi ileri düzey algoritmalar geliştirilmiştir. Her algoritma, farklı problem türlerine uygun olarak optimize edilmiştir ve özel durumlar için tercih edilir. Günümüzde özellikle büyük ölçekli harita ve ağ yapılarında yüksek verimli algoritmalara ihtiyaç duyulmakta olup, kısa yol algoritmaları bu gereksinimi karşılamaktadır. Her algoritmanın kendine has avantajları ve kullanım alanları bulunur, bu da doğru algoritmayı seçmenin önemini artırır.



Şekil 13: (a) Normal bir grafikte düğümler arasında tüm yolları aramak gerekirken, (b) Kısa yollar (shortcuts) eklenerek arama alanı azaltılır ve en kısa yol hesaplaması daha verimli hale gelir.(13)

#### **2. 1 En Kısa Yol Algoritmaları Neden Önemli**

Kısa yol algoritmaları, modern dünyada birçok uygulamanın temel taşını oluşturur. Ulaşım ağlarından lojistik optimizasyonuna, sosyal ağ analizinden ağ güvenliğine kadar geniş bir yelpazede bu algoritmalar kullanılmaktadır. Bu algoritmaların en önemli özelliği, büyük veri kümelerinde hızlı ve doğru sonuçlar sunarak zaman ve kaynak tasarrufu sağlamasıdır. Örneğin, bir nakliye firmasının rotalarını optimize etmesi veya bir telekomünikasyon ağında veri iletim süresini en aza indirmesi için bu algoritmalar büyük bir rol oynar. Bu algoritmalar olmadan, birçok endüstride maliyetler ve işlem süreleri önemli ölçüde artacaktır. Kısacası, kısa yol algoritmaları, hem kullanıcı deneyimini geliştiren hem de sistemlerin verimliliğini artıran vazgeçilmez araçlardır.

#### **2.2 Algoritma Kategorileri**

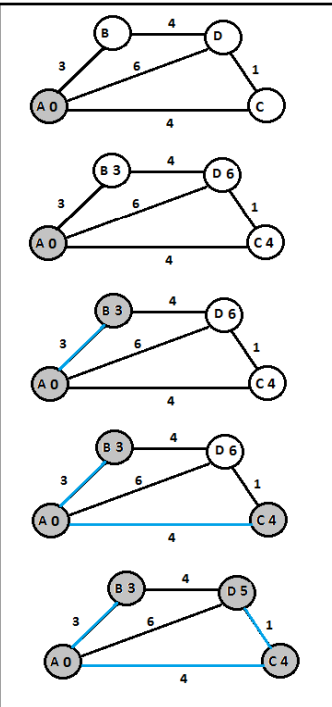
Kısa yol algoritmaları, çeşitli özelliklere ve ihtiyaçlara göre kategorilere ayrılır. Klasik algoritmalar, genellikle belirli kısıtlamaları olmayan, her tür grafikte uygulanabilir yöntemlerdir; Dijkstra ve Bellman-Ford algoritmaları bu kategoriye örnektir. Heuristik tabanlı algoritmalar ise, hedefe yönelik tahminler yaparak aramayı hızlandırır; bu grupta A\* ve ALT algoritmaları öne çıkar. Son olarak, hiyerarşi temelli algoritmalar ise büyük ve karmaşık grafiklerde hiyerarşik yapı kullanarak daha hızlı çözüm sunar, Contraction Hierarchies bu tür algoritmalara örnek olarak verilebilir. Her kategori, belirli türde problemler için farklı avantajlar sunar ve kullanım amacına göre seçilir. Bu kategorilere ayrılan algoritmalar, kullanıcıların ihtiyaçlarına göre en uygun çözümü sunmak üzere optimize edilmiştir.

### **3.0 Dijkstra Algoritması**

### E.W. Dijkstra’nın 1958'de geliştirdiği en kısa yol algoritması, lojistik, trafik planlama, ağ optimizasyonu gibi alanlarda kullanılır. Çevrimiçi alışverişle lojistik sektörü hızla büyüyerek ekonomik gelişime katkı sağlar. Bu yüzden teslimat süresini kısaltacak güzergâh planlaması önemlidir. Dijkstra algoritması, graf üzerinde tüm yolları inceleyerek en kısa yolu bulmaya çalışır; fakat birçok gereksiz hesaplama içerir ve verimliliği düşüktür (O(n²))(1)

**3.1 Giriş**

Dijkstra algoritması, bir kaynaktan hedefe en kısa yolu bulurken, her adımda en küçük mesafeye sahip düğümü seçer. Düğüm etiketlerini güncelleyerek, geçiş kenarlarının uzunluklarını dikkate alır. Bu işlem, tüm düğümler işlenene kadar devam eder ve her seferinde en kısa yolun keşfedilmesini sağlar. Algoritma, gereksiz hesaplamaları minimize ederek, büyük ağlarda daha verimli çalışmak için optimize edilebilir. (2)

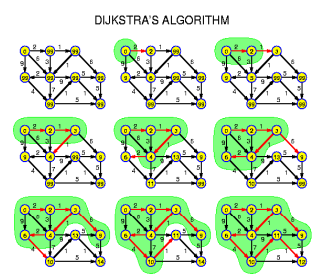
Dijkstra algoritması, bir başlangıç düğümünden grafik üzerindeki diğer tüm düğümlere olan en kısa yolları hesaplamak amacıyla geliştirilmiş olan bir algoritmadır. Yollar arasında en kısa mesafeyi bulmak için maliyeti en düşük düğümü sürekli seçerek ilerleyen bu algoritma, geniş bir uygulama alanına sahiptir. Özellikle, yol haritaları gibi büyük ölçekli grafiklerde kullanılarak, iki nokta arasındaki en kısa rotayı belirleme amacıyla sıklıkla tercih edilmektedir. Algoritma, her bir düğümde maliyet hesaplayarak ilerler ve tüm düğümler ziyaret edilip işlem tamamlandığında başlangıç noktasından diğer tüm düğümlere ulaşan en kısa mesafeleri elde etmiş olur. Dijkstra algoritmasının gücü, belirli bir kaynak noktadan başlayarak doğrusal bir ilerleme ile kısa yolların belirlenmesinde gizlidir. Bu algoritmanın güvenilir ve doğru sonuçlar üretebilmesi sayesinde, özellikle GPS sistemleri gibi rota optimizasyonu gerektiren uygulamalarda geniş çapta kullanılmaktadır.(3)(4)

Şekil 1: Dijkstra Algoritmasının bir uygulaması (1)

**3.2 Algoritmanın Adımları**

Dijkstra algoritmasının çalışması birkaç temel adımdan oluşur. İlk olarak, başlangıç düğümü seçilir ve bu düğümün maliyeti sıfır olarak atanır; diğer tüm düğümlerin maliyeti ise başlangıçta sonsuz kabul edilir, çünkü henüz herhangi bir yol bulunmamıştır. Algoritma, başlangıç düğümünden komşu düğümlere olan maliyetleri hesaplayarak onları bir listeye ekler. Sonrasında, mevcut düğümler arasından en düşük maliyetli düğüm seçilir ve bu düğümden diğer komşu düğümlere olan mesafeler tekrar gözden geçirilir. Her bir adımda, önceki adımların sonuçları dikkate alınarak en kısa yol hesaplamaları güncellenir. (5)

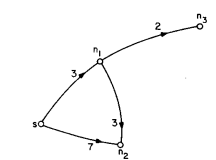
Algoritma, başlangıç düğümünü seçerek başlar ve en kısa yolu bulunan düğümleri içeren boş bir S kümesi oluşturur. Kaynak düğüm 0 ile etiketlenip S kümesine eklenir. Ardından, yeni eklenen düğümden kenarla bağlı olan, S kümesinde olmayan her düğüm etiketlenir. Bu etiketleme işlemi, yeni eklenen düğümün etiketi ile kenar uzunluğunun toplamı şeklindedir. Eğer düğüm zaten etiketliyse, etiket değeri, mevcut etiket ve yeni hesaplanan değerin minimumu olarak güncellenir. S kümesinde olmayan düğümlerden en küçük etikete sahip olan seçilip S kümesine eklenir. Bu süreç, hedef düğüm S kümesine eklenene veya etiketli düğüm kalmayana kadar tekrarlanır. Hedef düğüm etiketlenmişse, bu değer kaynak ile hedef arasındaki mesafedir.(6)



Şekil 2: Adım adım Dijkstra Algoritmasının çalışma mantığı (2)

**3.3 Etiket Algoritması**

Dijkstra etiket algoritması önermektedir. Algoritmada, her düğüm için iki tür etiket tanımlanır: kalıcı etiket ("p-label") ve geçici etiket ("t-label"). Kalıcı etiket, belirli bir adımda en kısa yol bulunmuş olan düğüme atanırken, geçici etiket, o anki en iyi tahmin edilen yolu gösterir. Ayrıca, geçici ve kalıcı etiketlerin bulunduğu düğüm kümeleri tanımlanır.(7)



Şekil 5: Düğümler ve yönlü kenarlar arasındaki ağırlıkları gösteren bir en kısa yol probleminin temsili (5)

**3.4 Zaman Karmaşıklığı**

Dijkstra algoritmasının zaman karmaşıklığı, komşuluk matrisini kullandığında O(V²) olur, burada V, grafikteki düğüm sayısını ifade eder. Eğer bir öncelik kuyruğu veya min-heap kullanılırsa, karmaşıklık O((V + E) log V) seviyesine düşer; burada E, kenar sayısını temsil eder. (8)

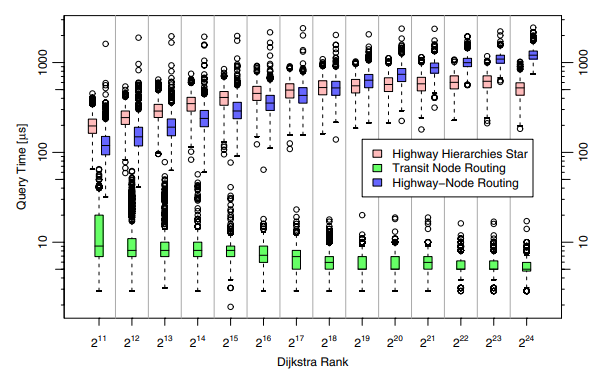
Bu zaman karmaşıklığı, Dijkstra algoritmasını orta büyüklükte grafikler için yeterince hızlı bir seçenek yaparken, daha büyük ölçekli grafiklerde ise optimizasyon ihtiyacını doğurur. Çok büyük grafiklerde, düğüm ve kenar sayısının artmasıyla karmaşıklık da artacağından, işlem süresi oldukça uzayabilir. Yine de algoritmanın akıllı seçim yapısı, gereksiz adımları eleyerek karmaşık grafiklerde dahi doğru ve güvenilir sonuçlar verebilmektedir. Dijkstra algoritması, özellikle ulaşım ağları ve harita sistemlerinde etkin bir yol bulma çözümü olarak öne çıkar.(9)

**3.5 Verimlilik Karmaşıklığı**

Dijkstra algoritması, küçük ve orta ölçekli grafiklerde oldukça etkilidir; fakat büyük ve karmaşık grafiklerde yavaşlayabilir. A\* ve ALT gibi algoritmalarla karşılaştırıldığında, Dijkstra algoritmasının doğrudan en kısa mesafeyi hesaplamak konusunda başarılı olduğu söylenebilir. Ancak bu algoritma, büyük grafiklerde hızını koruyamaz. Bu yüzden, büyük veri setleri üzerinde çalışırken performansı artırmak için bazı iyileştirmelere ihtiyaç duyulmaktadır. Dijkstra algoritması, büyük grafiklerde zaman kaybını önlemek amacıyla bazı geliştirici algoritmalarla birlikte kullanılabilir. Bu algoritmanın doğrudan en kısa mesafe hesaplamadaki güvenilirliği sayesinde, birçok yol bulma probleminin çözümünde tercih edilir. Özellikle daha basit yol bulma ihtiyaçlarında Dijkstra algoritması hızlı, doğru ve etkin bir çözüm sunar.(a\*8)

**3.6 Örnekler**

Dijkstra algoritmasının zaman karmaşıklığının iyileştirilmesi, daha hızlı çözüm süreçleri ve daha verimli hesaplamalar anlamına gelir, özellikle büyük ve karmaşık ağlarda önemli faydalar sağlar. Fibonacci yığınları gibi gelişmiş veri yapıları kullanıldığında, algoritmanın işlem süresi önemli ölçüde azalır, bu da gerçek zamanlı uygulamalarda daha iyi performans sağlar. Ancak, çok büyük veri tabanlarında veya yoğun bağlantılara sahip grafikte, algoritmanın verimliliği düşebilir, bu da uygulamaların kaynak tüketimini arttırabilir ve çözüm sürelerini uzatabilir.(8)



Şekil 8: Çeşitli hızlandırma tekniklerinin Dijkstra sıralamasına karşı sorgu performansı. (8)

**4.0 A\* Algoritması**

Nils Nilsson ve ekibinin 1968 yılında yayımladığı bu çalışma, A\* algoritmasının temelini atan önemli bir araştırmadır. A\* algoritması, bir başlangıç ve hedef noktası arasında en kısa ve en düşük maliyetli yolu bulmak için geliştirilmiştir. Algoritmanın en önemli özelliği, toplam maliyeti iki bileşen ile hesaplamasıdır: “g(n)” başlangıçtan mevcut düğüme kadar olan gerçek maliyeti, “h(n)” ise mevcut düğümden hedefe olan tahmini maliyeti temsil eder. Bu iki maliyetin toplamı, her adımda en iyi yolun seçilmesine olanak tanır.(a\*1)

**4.1 Giriş**

A\* algoritması, Dijkstra algoritmasından türetilmiş olup, hedefe ulaşma süresini kısaltmak için bir "heuristic" (sezgisel) fonksiyon kullanır. Bu fonksiyon, kaynak düğümden hedef düğüme olan tahmini maliyeti hesaplar ve algoritmanın belirli bir yöne doğru ilerlemesini sağlar. A\* algoritmasının asıl gücü, her adımda hem düğüme ulaşma maliyetini hem de tahmini hedef maliyetini değerlendirerek en verimli yolu seçmesidir. Yol ağları gibi büyük grafiklerde hedef odaklı bir arama yapması gerektiğinde A\* algoritması büyük avantaj sağlar. Özellikle ulaşım ve oyun geliştirme gibi alanlarda, en kısa yolu en hızlı şekilde bulmak amacıyla sıkça kullanılan bir algoritmadır. Bu yöntemle, büyük grafiklerde bile gereksiz düğümler göz ardı edilerek hedefe ulaşılabilir.(a\*8)

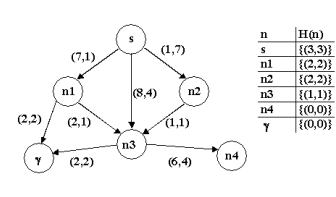
A\* algoritmasının büyük ağlarda daha verimli çalışmasını sağlamak için bölgesel heuristic fonksiyonlar ve çok seviyeli grafik yapıları kullanılır. Bu yöntemler, algoritmanın hem hızını hem de bellek kullanımını optimize ederek büyük ölçekli uygulamalarda performansı artırır. A\* algoritmasının özellikle büyük veri kümelerinde ve karmaşık grafiklerde daha etkili bir şekilde uygulanabilmesi için önemli öneriler sunar.(a\*2)

**4.2 Sezgisel Fonksiyon**

Heuristic fonksiyonun tutarlı veya alt-değerleme yapması gerektiği üzerinde durulur; bu sayede algoritma optimal çözümler üretebilir. Tutarlı bir heuristic fonksiyon, arama sürecinde her düğümün doğru bir şekilde değerlendirildiğinden emin olur, bu da algoritmanın verimliliğini artırır. A\* algoritması, uygun bir heuristic ile gereksiz düğüm ziyaretlerini en aza indirir ve bu sayede robotik, yapay zeka ve rota planlama gibi alanlarda etkin bir şekilde kullanılabilir.(a\*1)

Büyük ölçekli grafiklerdeki yol bulma problemlerinde algoritmanın performansını artırmak amacıyla çeşitli iyileştirmeler sunulmaktadır. Özellikle, heuristic fonksiyonların nasıl optimize edilebileceği ve bidirectional search (çift yönlü arama) gibi tekniklerin A\* ile entegre edilerek arama sürecinin hızlandırılabileceği üzerinde durulur. Çift yönlü arama ile hem başlangıç hem de hedef noktalarından aynı anda arama yapılarak işlem süresi kısaltılır.(a\*2)

Hedef yönelimli aramanın ardındaki sezgi, en kısa yolların hedefin genel yönüne doğru ilerlemesi gerektiğidir. A\* araması , (u,v) kenarının ağırlığını w(u,v)−π(u)+π(v)olarak değiştirerek bunu başarır; burada π(v), d(v,t) için bir alt sınırdır. Bu değişikliğin hedefe doğru yönelen kenarları kısalttığına dikkat edin. Eklenen ve çıkarılan düğüm potansiyelleri π(v) herhangi bir yol boyunca birbirini götürdüğünden, bu kenar ağırlığı değişikliği en kısa yolları korur. Ayrıca, tüm kenar ağırlıkları pozitif kaldığı sürece Dijkstra algoritması hâlâ kullanılabilir. Yol haritalarında A\* algoritmasını rota planlaması için kullanmanın klasik yöntemi, d(v,t) ’yi v ve t arasındaki Öklid mesafesine ve ağdaki en hızlı yolun ortalama hızına göre tahmin etmektir. Bu çok muhafazakâr bir tahmin olduğu için, en hızlı rotayı bulmada hızlanma oldukça düşüktür. (a\*7)



Şekil 7: Örnek grafik ve sezgisel fonksiyon. (7)

**4.3.0 Örnek Girişi**

A\* algoritmasının pratik kullanımları arasında yol haritalarında iki nokta arasındaki en kısa mesafeyi bulma işlemi öne çıkar. Örneğin, bir şehir haritasında, başlangıç ve hedef noktaları arasındaki en kısa rotayı bulmak için A\* algoritması kullanıldığında, algoritma her adımdan önce hedefe olan mesafeyi tahmin eder ve o yönde ilerlemeyi tercih eder. Böylece, daha hızlı ve optimize edilmiş bir yol arama süreci sağlanmış olur. Bu özellik, gereksiz düğümleri göz ardı etmeyi mümkün kılar ve işlemin yalnızca hedef odaklı gerçekleşmesini sağlar. A\* algoritmasının bu yeteneği, gerçek dünya uygulamalarında sıkça tercih edilmesinin en büyük nedenlerinden biridir. (a\*10)

**4.3.1 Örnek - “Dinamik Ağırlandırma”**

A\* algoritmasında “Dynamic weighting” yani dinamik ağırlıklandırma adı verilen bir teknik söz konusudur. Bu teknik, robotik sistemlerdeki engeller ve çevre koşulları sürekli değiştiğinden, A\* algoritmasının hedefe olan tahmini maliyetini (heuristic) esnek hale getiriyor. Bu sayede, robotlar hareket halindeyken karşılarına engeller çıktığında veya çevre koşulları değiştiğinde, yol bulma işlemi daha hızlı ve etkin bir şekilde güncelleniyor. Dinamik ağırlıklandırma ile heuristic fonksiyon, çevredeki değişikliklere göre optimize ediliyor ve bu da robotun hem enerji tasarrufu yapmasına hem de görevleri daha hızlı tamamlamasına yardımcı oluyor. Bu yaklaşım, A\* algoritmasının geleneksel uygulamalarında bulunmayan, robotik sistemler için özellikle geliştirilmiş yenilikçi bir çözüm sunuyor. Sonuç olarak, makale A\* algoritmasını, robotik yol bulma problemlerinde dinamik koşullara uyarlayarak hız ve performans artışı sağlayacak şekilde geliştiriyor.(a\*3)

**4.3.2 Örnek - NLP**

NLP, dillerin yapısını, anlamını ve kurallarını inceleyen bir alan ve bu tür karmaşık verilerde en iyi çözümü bulmak genellikle zorlayıcı olabiliyor. İşte A\* algoritması burada devreye giriyor. NLP uygulamalarında, bir cümledeki her sözcüğün doğru sırada ve doğru yapıda olup olmadığını belirlemek, yani en uygun geçiş yollarını bulmak, dilin yapısal kurallarına bağlı kalmayı gerektiriyor. A\* algoritması, bu tür analizlerde geçerli maliyeti ve hedefe olan tahmini maliyeti dikkate alarak en uygun sonucu bulmaya çalışıyor.(a\*4)

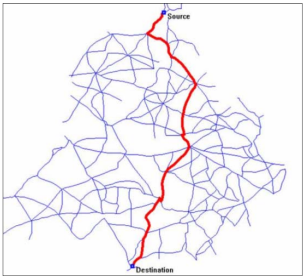
Doğru bir heuristic fonksiyon seçildiğinde, A\* algoritması sözcüklerin veya cümle yapılarına göre hızlı ve doğru bir şekilde analiz yapabiliyor. Örneğin, sözdizimsel analizde dil kurallarına dayalı bir tahminsel maliyet hesaplaması yapmak, algoritmanın doğru cümle yapılarını bulmasına yardımcı oluyor. Bu tür bir heuristic, dilin kurallarına uygun en iyi yapıyı bulmada rehberlik sağlıyor. A\* algoritması, doğru heuristic fonksiyonlarla çalıştığında NLP alanında karmaşık problemlere çözüm bulmada oldukça etkili bir araç olarak öne çıkıyor ve bu sayede dil işleme süreçlerinde hız ve doğruluk kazanımı sağlıyor.(a\*4)

**4.3.2 Örnek- Trafik**

Şehir içi trafiği yönlendirme, karmaşık ve sürekli değişen dinamiklerle dolu bir alandır; yol tıkanıklıkları, hava durumu değişiklikleri veya trafik kazaları gibi faktörler rota hesaplamalarını sürekli güncellemeyi gerektirir. A\* algoritması, bu tür durumlarda trafiği daha etkili yönetmek ve sürücülere en hızlı rotayı sağlamak için tercih edilmektedir. (a\*5)

Şehirdeki trafiğin yoğun olduğu saatlerde daha doğru tahminler yapmak için heuristic fonksiyonlar üzerine eklenen ağırlıklandırma yöntemlerinden bahsediliyor. Böylece algoritma, yol tıkanıklıklarını veya acil durumları göz önünde bulundurarak sürücülere en iyi yolu seçme imkanı sunuyor. Bu sayede, algoritma trafik yoğunluğunu daha etkili bir şekilde yönetmekte ve sürücüler için optimum rotalar belirlemektedir.(a\*5)

A\* algoritması, otoyol veya anayol gibi büyük yol ağlarında da etkin bir şekilde kullanılabilir. Örneğin, otoyol üzerinde bir şehirden diğerine giderken, A\* algoritması önemli ve kısa rotaları önceleyerek gereksiz yolları elemeye çalışır. Bu durumda, otoyollara öncelik verilir ve ana yollar üzerinden hedefe ulaşmak için en kısa yol hesaplanır. Böylelikle, algoritma karmaşık yol ağlarında dahi hızlı sonuçlar üretebilir. A\* algoritmasının özellikle büyük grafiklerde kısa sürede sonuç verebilmesi, yol ağları gibi büyük veri kümelerinde etkili bir yol bulma stratejisi olarak kullanılmasını sağlamaktadır.(a\*9)



Şekil 4: Acil Müdahale Sisteminde En Kısa Yol (4)

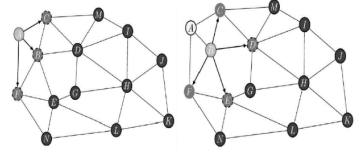
**4.3.2 Örnek- Dinamik grafikler**

Dinamik grafikler, sürekli değişen veri ve bağlantılara sahip ağlardır; bu durum özellikle gerçek dünyada, örneğin ulaşım ağları veya iletişim ağlarında görülür. A\* algoritmasının dinamik grafiklerde kullanılabileceğini ve geleneksel A\* algoritmasının bu tür uygulamalarda nasıl iyileştirilebileceğini hesaplanmıştır. Dinamik grafiklerde rota planlarken, A\* algoritmasının belirli düğümleri yeniden değerlendirme gereksinimi artmaktadır çünkü grafik sürekli değişmektedir.(a\*6)

A\* algoritmasının dinamik ortamlara daha uygun hale getirilmesi için “incremental search” (artımsal arama) gibi teknikler kullanılır. Bu teknik, daha önce hesaplanmış düğümleri tekrar tekrar değerlendirerek işlem süresini azaltmakta ve gereksiz maliyetlerden kaçınmaktadır. Ayrıca, sürekli güncellenen veri setlerinde daha verimli arama yapabilmek için algoritmanın hangi düğümleri dikkate alması gerektiği üzerinde de durulmaktadır. Böylece, A\* algoritması dinamik değişimlere anında uyum sağlayarak daha doğru ve güncel rotalar önerebilmektedir. (a\*6)

**4.3.2 Örnek- Robotik**

A\* algoritması, robotik uygulamalarda yaygın olarak tercih ediliyor çünkü bir başlangıç noktasından hedefe en kısa yolu bulmada oldukça etkili. Ancak, özellikle dinamik çevrelerde yani sürekli değişen koşullara sahip ortamlarda algoritmanın bazı sınırları var. Bu makale, A\* algoritmasının verimliliğini artırmak amacıyla dinamik ortamlar için özel geliştirmeler öneriyor. Örneğin, geleneksel A\* algoritması her yeni durumda tüm düğümleri yeniden değerlendirmeye yönelirken, bu çalışma sadece belirli düğümlerin tekrar gözden geçirilmesini sağlayan bir teknik öneriyor. Bu da, zamandan tasarruf sağlayarak robotik sistemlerin çevreye daha hızlı tepki vermesini sağlıyor.(a\*3)



Şekil 3: Köşeler arasındaki koşu yolunu gösterir, burada V köşeleri, E ise kenarı temsil eder. (3)

**5.0 ALT (A\* with Landmarks and Triangle Inequality)**

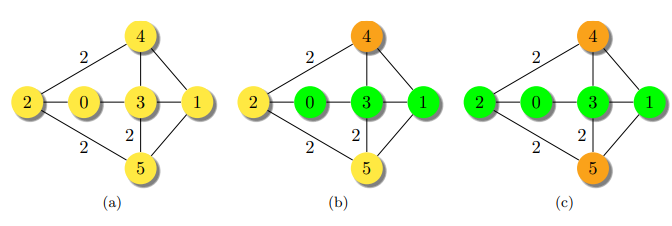
A\* algoritmasından türetilmiş olan ve "ALT algoritması" olarak bilinen bir yöntemi tanıtır ve harici bellekte saklanan büyük veri setleri üzerinde nasıl daha verimli bir yol bulma işlemi yapılabileceğini gösterir. ALT algoritması, A\* algoritmasında olduğu gibi başlangıç noktasından hedefe olan mesafeyi tahmin etmek için “landmark” (işaret noktaları) kullanır. Bu işaret noktaları, iki nokta arasındaki yolun tahmininde yol gösterici olur ve algoritmanın gereksiz düğümleri incelemeden, daha hızlı bir şekilde çözüme ulaşmasına olanak tanır.(alt1)

ALT algoritmasının en önemli avantajı, büyük yol ağlarında veya karmaşık grafiklerde daha hızlı çözüm üretebilmesidir. Algoritma, grafik üzerinde belirli stratejik noktalarda konumlanmış işaret noktaları yardımıyla iki düğüm arasındaki mesafeyi tahmin eder. Bu işaret noktaları, algoritmanın hedefe giden en kısa yolu seçmesinde yönlendirici bir rol oynar. Büyük yol ağlarında, ALT algoritması gereksiz düğümleri eleyerek yol bulma sürecini hızlandırır ve özellikle yol ağı haritaları gibi uygulamalarda oldukça etkilidir.(alt4)

**5.1 Giriş**

Algoritmanın harici belleğe erişim sayısını azaltmak için bazı optimizasyon yöntemleri önemlidir. Bu optimizasyonlarla, yalnızca gerekli verinin belleğe yüklenmesi sağlanarak daha verimli bir yol bulma süreci elde ediliyor. ALT algoritmasının bu özelliği, haritalama ve rota planlaması gibi büyük veri setlerine sahip uygulamalarda ciddi hız ve maliyet avantajları sağlıyor. Bu makale, büyük ve yoğun grafiklerde ALT algoritmasının harici bellekle nasıl etkili çalışabileceğine dair önemli bir rehber niteliğinde.(alt1)

ALT algoritması, üç ana bileşenden oluşur: A\* algoritması, işaret noktaları ve üçgen eşitsizliği ilkesi. Bu yöntemde, belirlenen işaret noktaları yardımıyla iki düğüm arasındaki mesafe hızlıca tahmin edilir ve algoritmanın hedefe en kısa yoldan ulaşması sağlanır. Her işaret noktası, mesafeyi tahmin etmede yol gösterici bir rol üstlenir ve algoritmanın işlemlerini azaltarak daha verimli çalışmasını sağlar. İşaret noktaları, özellikle yol haritalarında mesafeyi tahmin etme sürecinde oldukça kullanışlıdır ve büyük ağlarda hızlı sonuçlar üretmeye olanak tanır. Bu yöntem, ALT algoritmasının büyük veri kümeleri üzerinde dahi hızlı sonuç verebilmesini sağlar.(alt5)



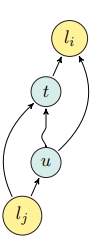
Şekil 9: Solda, karşıt örneğin oluşturabileceği grafik gösterilmektedir. Aksi belirtilmediği takdirde tüm kenar uzunlukları 1'dir. Ortadaki grafikte, 4. dönüm noktasını kullanan 0-1 sorgusunun ALT arama alanı gösterilmektedir. Sağda, aynı arama alanı, dönüm noktaları kullanılarak gösterilmektedir (9)

**5.2 Otoyol Hiyerarşileri**

Büyük ölçekli grafiklerde hızlı yol bulma için ALT algoritmasını "Highway Hierarchies" (Otoyol Hiyerarşileri) adı verilen bir yöntemle birleştiriyor. ALT algoritması, A\* algoritmasına dayansa da büyük ağlarda (örneğin ülke çapındaki haritalar gibi) performansını artırmak için işaret noktalarını ve üçgen eşitsizliği özelliklerini kullanarak optimize edilmiştir. "Highway Hierarchies" yaklaşımı ise, ağdaki daha uzun mesafelerde önemli olan “ana yolları” belirleyerek algoritmanın yalnızca kritik düğümleri incelemesini sağlar. Bu, yol bulma sürecinde zaman kazandırır ve karmaşıklığı azaltır.(alt2)

Otoyol Hiyerarşileri (Highway Hierarchies, HH), Contraction Hierarchies’den önce geliştirilmiş, büyük grafiklerde (özellikle yol ağlarında) en kısa yol araması için kullanılan hiyerarşik bir yaklaşımdır. HH’nin ardındaki fikir, uzun mesafeli yolları kapsayan "otoyollar" veya kısayollar oluşturmaktır. Bu yöntem, arama alanını basitleştirmek için kenar azaltma ve düğüm filtreleme tekniklerini kullanarak hedefe doğru gitmesi muhtemel yollar üzerine odaklanır. HH, büyük grafiklerde belirli "ana" kenarların veya "otoyolların" seyahate hakim olduğu fikrine dayanarak tasarlanmıştır.(alt4)

ALT algoritmasının otoyol hiyerarşileriyle birleştiğinde daha verimli hale gelir. Örneğin, grafik ağın belirli kısımlarını (örneğin otoyollar ve ana yollar) öne çıkararak, algoritma önemsiz düğümleri göz ardı eder. Böylece, ALT algoritması daha dar bir yol seçeneği üzerinde çalışarak işlem süresini kısaltır ve daha hızlı sonuçlar sunar. Bu çalışma, özellikle büyük ölçekli ulaşım ağları gibi karmaşık sistemlerde yol bulma sorunlarının çözümünde ALT algoritmasının etkinliğini artıracak hiyerarşik bir yapı sağlıyor.(alt2)

**5.3 İşaretleyiciler ve Ön İşleme**

ALT algoritmasının yol ağlarında daha hızlı rota planlaması için kullanılabilir. ALT algoritması, "A\*", "Landmark" ve "Triangle Inequality" (Üçgen Eşitsizliği) ilkelerini bir araya getirerek çalışır. Buradaki ana fikir, belirli stratejik noktalarda yer alan “işaret noktalarını” (landmarks) kullanarak, iki nokta arasındaki mesafeyi daha hızlı hesaplayabilmektir. İşaret noktaları, bir başlangıç noktasından hedefe olan mesafeyi önceden tahmin etmek için kullanılır ve bu sayede gereksiz düğümler değerlendirme dışında tutularak işlem süresi kısalır.(alt3)

Şekil 12: ALT için üçgen eşitsizlikleri. (12)

Özellikle büyük yol ağlarında bu yöntemin hızlandırıcı etkileri vardır.Örneğin, ALT algoritması üçgen eşitsizliği prensibini kullanarak işaret noktaları aracılığıyla yol mesafesini tahmin eder ve büyük yol ağlarında doğru tahminler yaparak süreci hızlandırır. Algoritmanın verimliliğini artırmak için işaret noktalarının en uygun şekilde seçilmesi de önemlidir.Bu seçim, algoritmanın gereksiz hesaplamalardan kaçınmasını sağlarken, iki nokta arasındaki en kısa yolu hızlıca bulmaya yardımcı olur. (alt3)

**6.0 Contraction Hierarchies**

Daraltma Hiyerarşileri (CH) algoritması, büyük yol ağı grafiklerinde en kısa yolları hesaplamak için benzersiz bir yaklaşımdır. Bu çalışma, ilk olarak 2008 yılında Geisberger, Sanders, Schultes ve Delling tarafından sunulmuş olup, o zamandan beri diğer gelişmiş rota planlama algoritmalarına ilham kaynağı olmuştur.(ch2)

CH algoritması, grafikte uzun mesafeli sorguların önemsiz düğümleri atlaması için kısayollar ekler. Geisberger ve ekibi tarafından geliştirilen bu algoritma, düğümleri "önem" sırasına göre az önemliden çok önemliye doğru daraltarak her daraltma sırasında belirli komşular arasında kısayollar oluşturur. Sorgulama aşamasında ise başlangıç ve hedef düğümlerden iki yönlü bir arama yapılır, ancak sadece daha yüksek öneme sahip düğümlere giden kenarlar ziyaret edilir.(ch3)

Küçültme sürecinin amacı, ağdaki düşük öncelikli yolları kaldırarak yalnızca ana rotaları bırakmaktır. Bu sayede, CH algoritması daha hızlı ve verimli sonuçlar üretebilir. Büyük yol ağlarında, gereksiz düğümleri ve yolları eleyerek en kısa yolu bulmak daha kolay hale gelir. CH algoritması, yol ağları gibi büyük veri setleri üzerinde hızlı bir çözüm sağlaması sayesinde çokça tercih edilmektedir.(ch9)

**6.1 Giriş**

En kısa yol problemleri uzun süredir araştırılmaktadır, ancak Dijkstra algoritması büyük grafikler için verimli değildir. Büyük, gerçek dünya grafiklerinin (örneğin kıtaların yol ağları) işlenmesi, daha gelişmiş algoritmalar gerektirir. Bu bağlamda, hiyerarşik yönlendirme için basit bir yaklaşım sunuyoruz. Ağırlıklı, yönlendirilmiş bir grafik G = (V, E) üzerinde düğümler, önem sırasına göre sıralanır (u<v v düğümü u'dan daha önemlidir). Düğümler bu sıraya göre daraltılarak bir hiyerarşi oluşturulur. Bir düğüm u, ağdan çıkarılarak kalan üst grafikte en kısa yollar korunacak şekilde daraltılır; bu, ⟨v,u,w⟩ yollarını ⟨v,w⟩ kısayol kenarı ile değiştirerek sağlanır. Kısayollar, yalnızca ⟨v,u,w⟩ yolunun v'den w'ye tek en kısa yol olması durumunda eklenir. Bu işlemle elde edilen yapı, daraltma hiyerarşisi (CH) olarak adlandırılır.(ch1)

**6.2 Algoritmanın Adımları**

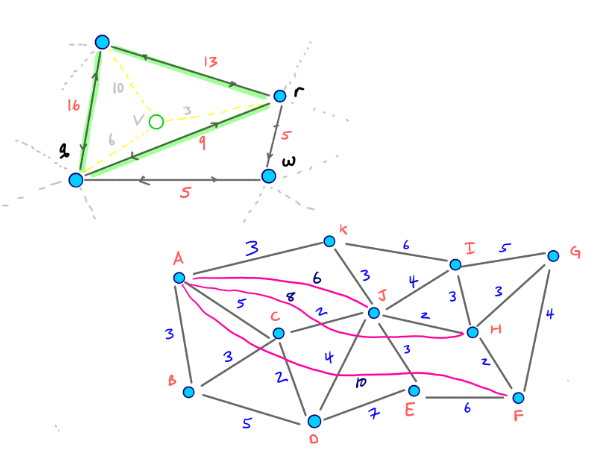
CH algoritması, adım adım ilerleyerek düğümleri küçültme süreciyle çalışır. İlk olarak, ağdaki düşük öneme sahip düğümler belirlenir ve kaldırılır. Kalan düğümler arasındaki bağlantılar yeniden düzenlenir ve yalnızca en önemli rotalar bırakılır. Bu işlem her adımda tekrarlandıkça, grafik giderek daha sade hale gelir ve algoritmanın çalışması hızlanır. CH algoritması, böylelikle büyük ölçekli grafiklerde dahi hızlı yol bulma performansı gösterir.(ch10)

**6.3 Ön İşlemler ve Düğüm Daralması**

Contraction Hierarchies (CH) algoritmasında, en kısa yolu hızlı bir şekilde bulabilmek için "preprocessing" yani ön işleme aşaması kritik bir adımdır. Bu aşama, algoritmanın çalışmadan önce ağı önceden düzenleyerek verimliliği artırmasını sağlar. Preprocessing sürecinde düğümler belirli bir sıralamayla "contraction" yani küçültme işlemine tabi tutulur. Düğüm küçültme işlemi, düşük öncelikli düğümlerin ağı en az bozacak şekilde kaldırılmasını içerir ve her düğümün kaldırılması sırasında gerekli bağlantılar diğer düğümlerle yeniden düzenlenir. Böylece, sadece en önemli yollar ve yüksek bağlantılar korunur, grafik daha sade hale gelir.(ch11)

Düğüm sıralaması, esasen bir daraltma işlemi simülasyonu olduğundan, zaman bağımlı duruma uyarlanması temel olarak kolaydır. Ancak, daraltma için öncelik kuyruğunda kullanılan maliyet fonksiyonuna yeni terimler eklemek anlamlıdır. Şu anki basit maliyet fonksiyonumuz üç terim içerir: Daha önce açıklanan kenar farkı, zaman bağımlı kenar ağırlık fonksiyonlarını tanımlamak için gereken segment sayısındaki değişikliği ölçen benzer bir segment farkı ve sorgu zamanı ile daraltmanın düzgünlüğünü etkileyen bir terim olan q. Başlangıçta her düğümde q=0 olur. Bir düğüm v daraltıldığında, v'nin komşusu u, q(u):=max⁡(q(u),q(v)+segments(u,v)) hesaplar. Burada segments(u,v), zaman bağımlı kenar ağırlığındaki segment sayısını ifade eder. Düğüm sıralaması, bir düğümün birden fazla kez daraltılmasını simüle edebileceğinden, daha önce yapılmış tüm arama sonuçlarını hatırlayan bir önbellek kullanılır.(ch4)

Doğru bir prosedür elde etmek için herhangi bir otoyol-düğüm kümesini seçebiliriz. Ancak, bu seçim ön işleme ve sorgu performansı üzerinde büyük bir etkiye sahiptir. Kabaca söylemek gerekirse, birçok en kısa yol üzerinde bulunan bir düğüm, yüksek seviyeli düğüm kümesine dahil edilmelidir. İlk uygulamada, otoyol düğüm kümesi V olarak, G grafiğinin otoyol hiyerarşisinin belirli bir seviye çekirdek düğümler kümesini kullanıyoruz. Başka bir deyişle, otoyol hiyerarşilerinin yapım prosedürünün düğümlerin önemini belirlemesine izin veriyoruz.(ch7)



Şekil 11: Contraction Hierarchies (CH) algoritmasına benzer şekilde, graf üzerinde en kısa yol bulmayı hızlandırmak için kullanılan bir yöntemi gösterir. (11)

**6.4 Otoyol-Düğüm Rotasyonu, HNR**

Otoyol-düğüm yönlendirmesi (Highway-Node Routing, HNR) bağlamında ele almak için bazı kavramları benimsememiz gerekiyor, daha doğrusu kanonik en kısa yollar, kapsama yolları ve örtüşme grafikleri kavramlarını. HNR, hiyerarşik yönlendirme için genel bir yaklaşımdır. Düğüm kümesinde bir hiyerarşi tanımlamak için başka bir tekniğe dayanır. Contraction Hierarchies (CH), her düğümün kendine ait bir seviyeye sahip olduğu uç bir durumdur. HNR’nin sorgu algoritmasını kullandığımız için, CH’lerin HNR’nin bir uygulaması olduğunu kanıtlayabilirsek, sorgu algoritması için doğruluk ispatını da benimseyebiliriz. (ch6)

**6.5 Sınırlı Yerel Arama**

Yerel aramalar, özellikle uzun mesafeli kenarlarla (örneğin feribot bağlantıları) zaman alıcı olabilir. Contraction Hierarchy (CH) oluşturmak için her kenarda bu aramalar en az iki kez yapılır: birincisi kenar farkları ve öncelik terimlerini hesaplamak, ikincisi ise düğüm küçültmesi içindir. İlk arama, kısayollar eklendikçe güncellenmesi gerektiği için saklanmaz. Yerel aramaları sınırlamak, algoritmayı geçersiz kılmaz çünkü sadece bir alt küme tanık yolu bulunur, ancak ek kısayollar grafiği yoğunlaştırır ve sonraki küçültmeleri daha zaman alıcı yapar. Bu nedenle, yerel arama sınırı dikkatle seçilmeli ve belirli sayıda düğüm küçültüldükten sonra değiştirilmelidir. (ch8)

**6.6 Adımlar ve Örnekler**

Contraction Hierarchies (Küçültme Hiyerarşisi) yönteminin temel bir kısıtlaması, ön işlem sürecinin metriğe bağımlı olmasıdır—yani, kenar ağırlıkları (metrik olarak da adlandırılır) bilinmelidir. Kullanıcı tercihleri veya trafik yoğunluğu gibi nedenlerle metrike yapılan önemli değişiklikler, pahalı yeniden hesaplamalar gerektirebilir. Bu nedenle çok seviyeli örtüşme (multilevel-overlay) MLD tekniklerini genişleten bir Özelleştirilebilir Rota Planlama (CRP) yaklaşımı önerildi. Bu yaklaşım üç aşamada çalışır. İlk ve maliyetli aşamada, yalnızca ağın topolojik yapısını kullanan, metrikten bağımsız yardımcı veriler hesaplanır. İkinci ve çok daha az maliyetli bir aşamada ise bu yardımcı veriler, belirli bir metriğe göre özelleştirilir ve böylece üçüncü aşamada hızlı sorgular yapılması sağlanır. (ch5)

**6.6.1 Örnek - Ülke Haritası**

CH algoritması, "highway dimension" kavramı çerçevesinde yol ağlarında verimli ve hızlı çözüm bulma amacıyla kullanılıyor. Özellikle geniş bir yol ağı düşünülerek önemli düğümlerin belirlenmesi üzerinden CH algoritmasının nasıl çalıştığı açıklanıyor. Bu örnekte, büyük ölçekli bir yol ağı olan bir ülke haritası ele alınıyor. (ch12)

**Aşamalar:**

1. **Ağ Yapısının İncelenmesi**: CH algoritması, öncelikle yol ağını analiz ederek düğümleri (yani kavşak noktaları) ve kenarları (yolları) tanımlar. Büyük bir yol ağı olan ülke haritasında, her yerel yol ve bağlantı yerine yalnızca ana yollar ve otobanlar gibi yüksek geçiş yoğunluğuna sahip olan yollar üzerinde yoğunlaşır.
2. **Highway Dimension Belirleme**: "Highway dimension" kavramı, sadece belirli yolların ve düğümlerin önemli olduğunu vurgular. Bu boyutlandırma sayesinde, algoritma sadece ana yolları ve bağlantıları işleyerek daha hızlı sonuçlar elde eder.
3. **Önemli Düğümleri İşleme**: Ana yollar üzerindeki düğümler belirlenir ve düşük öncelikli düğümler çıkarılarak yalnızca önemli bağlantılar analiz edilir. Bu sayede, algoritma tüm ağı analiz etmek yerine yalnızca ana arterlere odaklanarak hız kazanır.
4. **Sonuç**: Örneğin bir şehirler arası seyahat yapılacaksa, CH algoritması yalnızca ana yollar üzerinden çalıştığı için çok daha hızlı bir şekilde sonuç verir. Abraham ve arkadaşları bu yöntemin büyük yol ağlarında CH algoritmasının verimliliğini artırdığını gösterir.

**6.6.2 Örnek - Yol Ağları**

CH algoritmasının dinamik yol ağlarında kullanımına dair zorluklar ele alınıyor. Özellikle, sürekli güncellenen yol ağlarında CH algoritmasının performansında yaşanan düşüşlere dair bir örnek verilmiştir.

**Aşamalar:**

1. **Yol Ağı Analizi**: CH algoritması, öncelikle mevcut yol ağını analiz ederek hangi düğümlerin öncelikli olduğunu belirler. Ancak, dinamik bir şehirde yol ağı sürekli olarak değişmektedir (yeni yollar açılabilir veya bazı yollar kapanabilir).
2. **Değişimlerin İzlenmesi**: Şehir içindeki yol ağı dinamik bir yapıya sahiptir ve düzenli olarak güncellenir. CH algoritması, her yeni yol açıldığında veya mevcut bir yol kapandığında tüm ağı baştan işlemelidir.
3. **Yeniden Ön İşleme**: Her değişiklik sonrası, CH algoritması tüm düğümleri tekrar işlemek zorunda kalır. Bu yeniden ön işleme süreci zaman alıcıdır ve algoritmanın performansını düşürür.
4. **Sonuç**: Örneğin bir şehrin merkezi ile uzak bir nokta arasındaki yolun kısa sürede bulunması gerektiğinde, sürekli güncellenen bir ağda bu yeniden işleme süreci CH algoritmasını yavaşlatır. Bauer ve ekibi, bu örnekle birlikte CH algoritmasının statik ağlarda verimli olduğunu, ancak dinamik yapılarda zorluklarla karşılaştığını gösteriyor.

Bu örnek, CH algoritmasının sabit yapılarda başarılı olduğunu, fakat dinamik ağlarda performansının sınırlı kalabileceğini anlatıyor. (ch13)

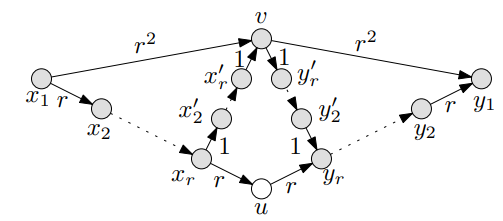
**6.6.3 Örnek - Almanya Yol Ağı**

CH algoritmasının geniş grafiklerde nasıl çalıştığı bir Almanya yol ağı örneği ile ele alınmış. Bu örnek, CH algoritmasının büyük ölçekli grafiklerdeki ön işleme sürecini anlamamızı sağlıyor.

**Aşamalar:**

1. **Yol Ağı Oluşturma**: CH algoritması, Almanya genelindeki yol ağını analiz eder. Her şehir arasındaki bağlantıları, ana yolları ve otobanları içeren büyük bir grafik oluşturur.
2. **Öncelikli Düğümleri Belirleme**: CH algoritması, her düğümün önemini değerlendirir. Örneğin, şehirlerin içindeki küçük yollar düşük öncelikli olarak işaretlenir ve çıkarılırken, şehirler arasındaki ana yollar ve otobanlar yüksek öncelikli düğümler olarak seçilir.
3. **Ön İşleme Süreci**: Düşük öncelikli düğümler çıkarıldıktan sonra, geriye kalan önemli düğümler arasındaki bağlantılar daha basit bir yapı oluşturur. CH algoritması, bu basitleştirilmiş grafik üzerinde çalışarak çok daha hızlı sonuçlar elde eder.
4. **Sonuç**: Örneğin Almanya’nın büyük şehirleri arasındaki en kısa yollar hızlıca hesaplanabilir. Bu örnek, CH algoritmasının yalnızca gerekli düğümleri analiz ederek büyük grafiklerde hız kazandığını ve işlem süresini kısalttığını göstermektedir.

Bu örnek, CH algoritmasının ön işleme sürecinde büyük grafiklerde nasıl hız kazandığını ayrıntılı bir şekilde açıklar. (ch14)



Şekil 10: V düğümünün daralmasından sonra kenar farkı ve U düğümünün sayısı değişir. (10)

**6.6.4 Örnek - Reach Based Routing**

CH algoritması yol ağı üzerindeki “reach-based” routing tekniği ile örneklenmiştir. CH algoritmasının geniş yol ağlarında belirli rotalar üzerinde nasıl optimize edildiği anlatılmaktadır.

**Aşamalar:**

1. **Yüksek Öncelikli Yolların Seçimi**: CH algoritması, özellikle geniş yol ağlarında sık kullanılan ana yolları ve otobanları yüksek öncelikli yollar olarak seçer. Örneğin, büyük bir şehirde ara sokaklar veya mahalle içi yollar gibi düşük öncelikli yollar işlem dışı bırakılır.
2. **Ön İşleme ve Küçültme**: Ana yollar ve otobanlar üzerindeki düğümler ve bağlantılar işlenir. Düşük öncelikli yollar ve bağlantılar çıkarılarak grafik küçültülür, yalnızca şehirler arası veya yüksek öneme sahip düğümler arasında kalan bağlantılar korunur.
3. **Güzergah Seçimi**: Örneğin, bir şehirler arası seyahat planlanırken algoritma yalnızca yüksek öncelikli yollar üzerinden bağlantılar kurar. Bu, güzergahı sadeleştirerek daha hızlı bir işlem yapılmasını sağlar.
4. **Sonuç**: Örneğin bir şehirden diğerine yapılacak yolculuklarda algoritma, yalnızca ana yollar üzerinden bir çözüm sunarak süreci hızlandırır. Bu örnek, CH algoritmasının yüksek öneme sahip yollar üzerinden işlem yaparak geniş grafiklerde nasıl zaman kazandırdığını gösterir.

Bu örnek, CH algoritmasının geniş grafiklerde etkili sonuçlar vermek için düşük öncelikli düğümleri nasıl dışladığını anlatmaktadır.

**7.0 Karşılaştırmalı Analiz**

### **1. Dijkstra Algoritması**

* **Verimlilik**: Yoğun grafikler için uygun olsa da büyük grafiklerde yavaş kalabilir.
* **Zaman Karmaşıklığı**: Adjacency matrix ile O(V^2) veya öncelikli kuyruk ve adjacency list kullanımıyla O(E+Vlog⁡V) (V = düğüm sayısı, E = kenar sayısı).

### **2. *A\* Algoritması***

* **Verimlilik**: Büyük, ızgara tabanlı haritalar veya yol ağlarında uygun bir sezgi ile oldukça verimlidir.
* **Zaman Karmaşıklığı**: En kötü durumda Dijkstra ile benzer, ancak uygun bir sezgi ile genellikle çok daha hızlıdır.

### **3. Contraction Hierarchies (CH)**

* **Verimlilik**: Özellikle büyük, seyrek yol ağlarında hızlı sorgulama süreleri sağlar.
* **Zaman Karmaşıklığı**: Ön işleme karmaşıktır (O(Vlog⁡V)O(V \log V)O(VlogV)), ancak sorgulama süresi çok düşüktür.

### **4. *ALT (A\*, Landmarklar ve Üçgen Eşitsizliği) Algoritması***

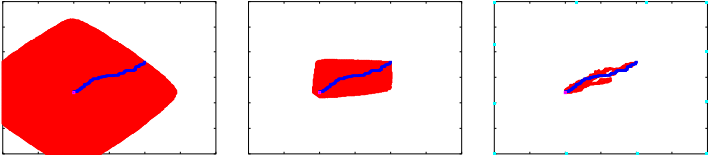
* **Verimlilik**: Özellikle yol ağlarında, düzensiz mesafeler varsa iyi performans gösterir ve genellikle A\*’dan daha hızlıdır.
* **Zaman Karmaşıklığı**: Ön işleme süresi landmark sayısına ve grafiğin boyutuna bağlıdır; sorgular hızlıdır.

**7.1 Algoritmaların Karşılaştırılması**

**Dijkstra Algoritması:** Dijkstra algoritması, her bir düğümden kaynak düğüme olan mesafeyi hesaplayarak tüm düğümlere en kısa yolu bulmayı amaçlayan bir yöntemdir. Algoritmanın temel özelliği, her düğümde olası tüm yolları değerlendirerek ilerlemesidir. Bu nedenle, bir kaynak düğümden tüm düğümlere en kısa yolları bulmak için güvenilir ve doğru sonuçlar sağlar. Ancak hedef odaklı bir stratejiye sahip olmadığı için, özellikle çok geniş grafiklerde yavaşlayabilir ve performansı düşebilir. Dijkstra algoritması, veritabanı veya ağ yönetimi gibi uygulamalarda çok kullanışlıdır çünkü tüm düğümlere olan en kısa yolların hesaplanmasına izin verir. Fakat daha dinamik ve hızlı sonuçlar gerektiren durumlarda, özellikle sadece iki nokta arasındaki en kısa yolu bulmak gerektiğinde, A\* ve ALT gibi algoritmalarla yarışmakta zorlanır.

**A\* Algoritması:** A algoritması, Dijkstra algoritmasına bir hedef odaklılık ekleyen geliştirilmiş bir versiyondur. A\* algoritması, düğümlerin hedefe olan uzaklıklarını tahmin eden bir “heuristic” fonksiyonunu kullanarak gereksiz düğümleri atlamasını sağlar. Bu sayede, hedefe odaklanarak çalışır ve yalnızca gerekli düğümler üzerinde işlem yapar. Özellikle, iki nokta arasında hızlı bir şekilde en kısa yol hesaplaması gereken durumlarda oldukça etkilidir. Ancak A\* algoritmasının başarısı büyük ölçüde kullanılan heuristic fonksiyonuna bağlıdır; uygun bir heuristic fonksiyonu seçilmezse, algoritmanın performansı düşebilir veya yanlış sonuçlar elde edilebilir. A\* algoritması çoğunlukla yol bulma, robotik navigasyon ve oyun geliştirme alanlarında tercih edilir çünkü hızla hedef düğüme ulaşmasını sağlar. Bu algoritmanın performansı, özellikle yol ağları gibi karmaşık grafiklerde büyük ölçüde hız kazandırır.

**ALT Algoritması:** ALT algoritması, A\* algoritmasından daha hızlı sonuçlar elde etmek için işaret noktaları (landmark) ve üçgen eşitsizliği kavramlarını kullanarak geliştirilen bir versiyondur. İşaret noktaları sayesinde algoritma, kaynak ve hedef düğümler arasında daha hızlı ve daha verimli bir yol bulur. ALT algoritması, büyük ölçekli grafiklerde, özellikle yol ağları gibi geniş ve karmaşık yapılarda üstün performans gösterir. Fakat işaret noktalarının dikkatli seçilmesi gerekir; uygun işaret noktaları seçilmezse verimlilik düşebilir. Haritalar gibi geniş veri yapılarına sahip uygulamalarda ALT algoritması, kısa süreli yol bulma işlemleri için mükemmel sonuçlar sağlar. Ancak, veri güncellemeleri ve dinamik grafiklerde işaret noktalarının sürekli yeniden düzenlenmesi gerektiği için, statik verilerle daha uyumlu bir yapıya sahiptir.



Şekil 6: Aynı girdide Dijkstra algoritması (sol), Manhattan alt sınırlarına sahip A∗search (orta) ve bir ALT algoritması (sağ) tarafından ziyaret edilen köşeler. (6)

**CH (Contraction Hierarchies) Algoritması:** CH algoritması, çok büyük grafiklerde daha hızlı sonuç almak için grafikteki düğümleri küçültme yöntemiyle çalışan bir algoritmadır. İlk olarak grafikteki düşük öncelikli düğümler belirlenir ve kaldırılır; bu işlem sırasında ağın bütünlüğünü korumak için gerekli bağlantılar yeniden düzenlenir. Bu sayede grafik daha basit bir yapıya ulaşır ve algoritmanın çalışma hızı artar. Bu işlemler "preprocessing" adı verilen ön işleme sürecinde yapılır. Ön işleme sırasında düğümler kademeli olarak küçültülerek sadece en önemli yollar bırakılır, böylece algoritma daha az düğümle çalışarak hızlı sonuçlar üretir. Bu özellikleriyle CH algoritması, özellikle yol ağları gibi büyük ölçekli veri kümelerinde çok etkili bir çözüm sunar. Ancak preprocessing süreci uzun sürebileceği için, veri sıklıkla değişiyorsa veya dinamik yapıda ise bu durum CH algoritmasının performansını düşürebilir.

**7.2 Tüm Algoritmaların Artı ve Eksi Yönleri**

1. **Dijkstra Algoritması**

**Artılar:**

Dijkstra algoritması her düğüme olan en kısa yolları doğru ve güvenilir şekilde hesaplar. Tüm düğümler için optimal yol bulma konusunda başarılıdır.Algoritma yapısı basit ve anlaşılırdır, bu da uygulama ve geliştirilmesini kolay hale getirir. Eğitim amaçlı birçok algoritma dersinde ilk öğretilen yol bulma algoritmalarından biridir.

**Eksiler:**

Dijkstra algoritması hedef odaklı olmadığından büyük grafiklerde çok fazla düğüm işlemek zorunda kalır, bu da onu yavaşlatabilir. Özellikle yalnızca iki düğüm arasındaki en kısa yolu arıyorsanız, Dijkstra fazla sayıda düğümü gereksiz yere işler. Dinamik veri yapılarında her değişimden sonra tüm algoritmanın yeniden çalışması gerekebilir, bu da işlem süresini uzatabilir.

1. **A Algoritması\***

**Artılar:**

A\* algoritması hedef odaklı bir yapıya sahiptir ve gereksiz düğümleri eleme kapasitesi sayesinde iki düğüm arasındaki en kısa yolu Dijkstra’dan daha hızlı bulabilir. Yol bulma ve navigasyon gibi dinamik durumlarda tercih edilmesinin ana nedenlerinden biri budur.Heuristic fonksiyonu sayesinde belirli bir hedefe odaklanabilir, bu da büyük grafiklerde daha etkili çalışmasını sağlar. Bu özellik, özellikle yol ağları gibi çok geniş grafiklerde büyük avantaj sağlar.

**Eksiler:**

A\* algoritmasının başarısı kullanılan heuristic fonksiyonunun doğruluğuna ve etkinliğine bağlıdır. Uygun bir heuristic belirlenmezse, algoritma gereksiz düğümleri işleyebilir ve performansı düşebilir. Heuristic ayarları genellikle grafik yapısına göre yeniden ayarlanmalıdır; bu nedenle algoritmanın farklı veri türlerinde uygulanması daha karmaşık hale gelebilir.

1. **ALT Algoritması**

**Artılar:**

ALT algoritması, işaret noktaları ve üçgen eşitsizliği sayesinde büyük grafiklerde hızlı çalışır ve yalnızca gerekli yolları işleyerek hedef düğüme odaklanır. Yol ağları gibi geniş çaplı grafiklerde en kısa yolu çok hızlı bir şekilde bulur. Bu özellik, geniş haritalarda kullanılan yol bulma işlemlerinde etkili sonuçlar elde edilmesini sağlar. ALT algoritması yol ağları gibi veri setlerinde kısa sürede çözüm üretebilir.

**Eksiler:**

ALT algoritmasında işaret noktalarının seçimi büyük bir önem taşır. Yanlış veya yetersiz işaret noktaları, algoritmanın hız ve verimlilik açısından geride kalmasına neden olabilir. Sık güncellenen veya dinamik grafiklerde, işaret noktalarının sürekli yeniden seçilmesi gerektiğinden, statik verilerle uyumlu çalışsa da dinamik durumlarda kullanımı zor olabilir.

1. **CH (Contraction Hierarchies) Algoritması**

**Artılar:**

CH algoritması, büyük grafiklerde düğüm küçültme ve preprocessing işlemi ile grafiği sadeleştirerek daha hızlı sonuçlar verir. Özellikle çok büyük yol ağlarında hızlı çözüm üretebilecek bir algoritmadır. Ön işleme sürecinde gereksiz düğümler kaldırıldığından, algoritma çok hızlı ve verimli çalışabilir. Bu özelliğiyle yol ağları gibi büyük veri setleri için idealdir.

**Eksiler:**

Preprocessing (ön işleme) süreci zaman alıcıdır ve sık değişen veya dinamik veri kümeleri için uygun değildir. Her değişiklik sonrası yeniden işleme yapılması gerektiği için, bu tür veri kümelerinde performansı düşebilir. Grafik sık güncelleniyorsa veya yeni düğümler ve yollar ekleniyorsa, CH algoritmasının ön işleme aşaması her seferinde tekrarlanmalı, bu da zaman kaybına yol açabilir.

### **8.0 Sonuç**

Bu analizde, dört ana en kısa yol algoritmasını (Dijkstra, A\*, Contraction Hierarchies (CH) ve ALT) karşılaştırdık. Her bir algoritma farklı avantaj ve dezavantajlara sahip olup, farklı uygulama türleri ve grafik yapıları için uygundur. Performanslarını, karmaşıklıklarını ve belirli senaryolara uygunluklarını anlamak, doğru yaklaşımı seçmek açısından önemlidir.

#### **8.1 Bulguların Özeti**

1. Dijkstra Algoritması: Daha küçük, pozitif ağırlıklı grafikler için uygundur, ancak büyük ölçekli ağlarda sezgisiz, kapsamlı arama yapması nedeniyle verimsiz hale gelir.
2. *A Algoritması*\*: Dijkstra’nın yaklaşımını sezgi kullanarak geliştirir ve hedefe daha yakın düğümleri önceliklendirir. Özellikle iyi bir sezgi ile büyük grafiklerde etkilidir, ancak performansı sezginin kalitesine bağlıdır.
3. Contraction Hierarchies (CH): Büyük, statik yol ağlarında, önemsiz düğümleri “sıkıştırarak” hızlı sorgu süreleri sağlar. Ön işlem kaynak yoğun olsa da, CH sık ve hızlı yol sorguları gerektiren uygulamalar için idealdir.
4. ALT Algoritması: A\* algoritmasını, büyük ve seyrek grafiklerde arama verimliliğini artırmak için landmarklar ile birleştirir. Performansı, landmarkların stratejik seçimlerine bağlı olup hız ve ön işleme gereksinimleri arasında denge sağlar.

#### **8.2 Gelecekteki Yönelimler**

* Dinamik Grafik Adaptasyonları: Bu algoritmaların, kenar ağırlıkları veya düğümlerin zamanla değişebildiği dinamik grafiklerdeki etkinliklerini artıracak çalışmalar yapılabilir. CH ve ALT gibi algoritmalar gerçek zamanlı güncellemeler için uyarlanabilir.
* *A İçin Sezgi Optimizasyonu*\*: A\* algoritması için uyarlanabilir sezgi fonksiyonlarının geliştirilmesi, özellikle evrensel bir sezginin uygun olmadığı durumlarda verimliliğini artırabilir.
* Hibrit Yaklaşımlar: CH, ALT ve diğer algoritmalardan öğeler birleştirilerek, hem yüksek hızlı sorgu süreleri hem de dinamik ortamlar için esneklik sağlayan hibrit yöntemler elde edilebilir.
* Makine Öğrenimi Entegrasyonu: Makine öğrenimi modelleri, grafik yapısına, veri boyutuna veya sorgu sıklığına göre en uygun algoritmayı seçerek yol bulmayı optimize etmeye yardımcı olabilir.

#### **8.3 Son Düşünceler**

Bu algoritmaların her biri yol bulmada önemli bir rol oynar ve lojistikten yapay zeka destekli navigasyona kadar geniş uygulama alanlarına sahiptir. Dijkstra ve A\* gibi geleneksel algoritmalar temelken, CH ve ALT gibi ilerlemeler gerçek zamanlı ve büyük ölçekli uygulamaların gelişen ihtiyaçlarını karşılar. Bu algoritmalar arasındaki seçim, hız, bellek ve esneklik gibi faktörlerin dengelenmesiyle belirlenmelidir. Veri hacmi ve karmaşıklığı arttıkça, gelecekteki yol bulma çözümlerinin, bu temel algoritmaların güçlü yönlerini birleştiren hibrit modeller ve uyarlanabilir yaklaşımlarla karakterize edilmesi muhtemeldir.

**REFERANSLAR**

1. Bing, Heming, and Lu Lai. "Improvement and application of Dijkstra algorithms." *Academic Journal of Computing & Information Science* 5.5 (2022): 97-102.
2. Dijkstra, Edsger W. "A note on two problems in connection with graphs." *Edsger Wybe Dijkstra: his life, work, and legacy*. 2022. 287-290.
3. Cevizci, Bekir. "Calculating the Shortest Path Using Dijkstra's Algorithm." Journal of Inquiry Based Activities 8.2 (2018): 70-85.
4. Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). Introduction to Algorithms. MIT Press
5. Jasika, Nadira, et al. "Dijkstra's shortest path algorithm serial and parallel execution performance analysis." *2012 proceedings of the 35th international convention MIPRO*. IEEE, 2012.
6. Bellman, R. (1958). On a routing problem. Quarterly of Applied Mathematics, 16(1):87–90 dijkstra
7. Iqbal, Mazhar, et al. "A fast and reliable Dijkstra algorithm for online shortest path." Int. J. Comput. Sci. Eng 5.12 (2018): 24-27.
8. Kai, Ni, Zhang Yao-ting, and Ma Yue-peng. "Shortest path analysis based on Dijkstra's algorithm in emergency response system." *TELKOMNIKA Indonesian Journal of Electrical Engineering* 12.5 (2014): 3476-3482.
9. Gutman, R. (2004). Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX), pages 100–111. SIAM.(a\*10)
10. Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968): 100-107.(a\*1)
11. Goldberg, A. V. and Harrelson, C. (2004). Computing the shortest path: A\* search meets graph theory. Technical report.(a\*2)
12. Mandow, Lawrence, and JL Pérez De la Cruz. "A New Approach to Multiobjective A\* Search." *IJCAI*. Vol. 8. 2005.(a\*3)
13. Och, Franz Josef, Nicola Ueffing, and Hermann Ney. "An efficient A\* search algorithm for statistical machine translation." *Proceedings of the ACL 2001 Workshop on Data-Driven Methods in Machine Translation*. 2001.[(a\*4)](https://aclanthology.org/W01-1408.pdf(a*5))
14. Zeng, Wei, and Richard L. Church. "Finding shortest paths on real road networks: the case for A." *International journal of geographical information science* 23.4 (2009): 531-543.(a\*5)
15. Delling, Daniel, and Dorothea Wagner. "Landmark-based routing in dynamic graphs." *Experimental Algorithms: 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007. Proceedings 6*. Springer Berlin Heidelberg, 2007.[(a\*6)](https://i11www.iti.kit.edu/extra/publications/dw-lbrdg-07.pdf(a*6))
16. Sanders, Peter, and Dominik Schultes. “Engineering fast route planning algorithms.” International Workshop on Experimental and Efficient Algorithms. Springer, Berlin, Heidelberg, 2007.(a\*7)
17. Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2009). Engineering route planning,(a\*8)
18. Delling, D., Sanders, P., Schultes, D., and Wagner, D. (2006). Highway hierarchies star. In 9th DIMACS Implementation Challenge. (a\*9)(alt2)(alt5) (ch10)
19. Goldberg, Andrew V., and Renato Fonseca F. Werneck. "Computing Point-to-Point Shortest Paths from External Memory." ALENEX/ANALCO 2 (2005).[(alt1)](https://www.researchgate.net/publication/220981995_Computing_Point-to-Point_Shortest_Paths_from_External_Memory(alt1))
20. Fuchs, Fabian. "On Preprocessing the ALT-Algorithm." Student thesis, Faculty of Computer Science, Institute for Theoretical Informatics (ITI), Karlsruhe Institute of Technology (KIT) 23 (2010).[(alt3)](https://www.fabianfuchs.com/fabianfuchs_ALT.pdf(alt3))
21. ALT Algorithms for Faster Route Planning in Road Networks (Fabian Fuchs) (alt4)
22. Geisberger, Robert, et al. "Contraction hierarchies: Faster and simpler hierarchical routing in road networks." Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7. Springer Berlin Heidelberg, 2008.[(ch1)](https://ae.iti.kit.edu/download/diploma_thesis_geisberger.pdf(ch1)) (ch6) (ch8) (ch9) (ch11)
23. John Lazarsfeld Tufts Comp 150 Alg ProjectSummer 2018 (ch2)
24. Bast, Hannah, et al. "Route planning in transportation networks." Algorithm engineering: Selected results and surveys (2016): 19-80.[(ch3)](https://arxiv.org/pdf/1504.05140(ch3))
25. Batz, G. Veit, et al. "Time-dependent contraction hierarchies." 2009 Proceedings of the Eleventh Workshop on Algorithm Engineering and Experiments (ALENEX). Society for Industrial and Applied Mathematics, 2009. (ch4)
26. Dibbelt, Julian, Ben Strasser, and Dorothea Wagner. "Customizable contraction hierarchies." Journal of Experimental Algorithmics (JEA) 21 (2016): 1-49. (ch5)
27. Schultes, Dominik, and Peter Sanders. “Dynamic highway-node routing.” International Workshop on Experimental and Efficient Algorithms. Springer, Berlin, Heidelberg, 2007.(ch7)
28. Holzer, Martin, Frank Schulz, and Dorothea Wagner. "Engineering multilevel overlay graphs for shortest-path queries." Journal of Experimental Algorithmics (JEA) 13 (2009): 2-5.(ch14)
29. Bauer, Reinhard, et al. “Preprocessing speed-up techniques is hard.” International Conference on Algorithms and Complexity. Springer, Berlin, Heidelberg, 2010.(ch13)
30. Abraham, Ittai, et al. “Highway dimension, shortest paths, and provably efficient algorithms.” Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2010.(ch12)
31. Goldberg, Andrew V., Haim Kaplan, and Renato F. Werneck. "Better landmarks within reach." International Workshop on Experimental and Efficient Algorithms. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.(ch15)

**Görsel Referanslar**

1. Cevizci, Bekir. "Calculating the Shortest Path Using Dijkstra's Algorithm." *Journal of Inquiry Based Activities* 8.2 (2018): 70-85.
2. Jasika, Nadira, et al. "Dijkstra's shortest path algorithm serial and parallel execution performance analysis." *2012 proceedings of the 35th international convention MIPRO*. IEEE, 2012.
3. Iqbal, Mazhar, et al. "A fast and reliable Dijkstra algorithm for online shortest path." Int. J. Comput. Sci. Eng 5.12 (2018): 24-27.
4. Kai, Ni, Zhang Yao-ting, and Ma Yue-peng. "Shortest path analysis based on Dijkstra's algorithm in emergency response system." TELKOMNIKA Indonesian Journal of Electrical Engineering 12.5 (2014): 3476-3482.
5. Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968): 100-107.
6. Goldberg, A. V. and Harrelson, C. (2004). Computing the shortest path: A\* search meets graph theory. Technical report.(a\*2)
7. Mandow, Lawrence, and JL Pérez De la Cruz. "A New Approach to Multiobjective A\* Search." *IJCAI*. Vol. 8. 2005.
8. Sanders, Peter, and Dominik Schultes. “Engineering fast route planning algorithms.” International Workshop on Experimental and Efficient Algorithms. Springer, Berlin, Heidelberg, 2007.
9. Fuchs, Fabian. "On Preprocessing the ALT-Algorithm." Student thesis, Faculty of Computer Science, Institut for Theoretical Informatics (ITI), Karlsruhe Institute of Technology (KIT) 23 (2010).
10. Geisberger, Robert, et al. "Contraction hierarchies: Faster and simpler hierarchical routing in road networks." Experimental Algorithms: 7th International Workshop, WEA 2008 Provincetown, MA, USA, May 30-June 1, 2008 Proceedings 7. Springer Berlin Heidelberg, 2008.
11. John Lazarsfeld Tufts Comp 150 Alg ProjectSummer 2018
12. Bast, Hannah, et al. "Route planning in transportation networks." Algorithm engineering: Selected results and surveys (2016): 19-80.
13. Dibbelt, Julian, Ben Strasser, and Dorothea Wagner. "Customizable contraction hierarchies." Journal of Experimental Algorithmics (JEA) 21 (2016): 1-49.