Gezgin Satıcı Problemi

Ömer AYILMAZDIR
191307014

Kocaeli Üniversitesi
Bilişim Sistemleri Mühendisliği
omeraylmazdr@gmail.com

Haktan AYDIN
191307065

Kocaeli Üniversitesi
Bilişim Sistemleri Mühendisliği
haktanbt@gmail.com

***Özet*—Bu proje kapsamında bir kuryenin gezgin satıcı problem çözümlenmiştir. Dağıtıcı gönderi adreslerine en kısa yoldan giderek zaman ve maliyet tasarrufu sağlayacaktır. C# Windows Form ile masaüstü program olarak geliştirdiğimiz bu projede kullanılabilirliği kolaylaştırmak adına basit ve sıkmayacak bir arayüz tasarladık. Amaç gerçekleştirilmesi gereken işi en stressiz bir şekilde halletmek. Satıcı belli bir şehirden başlayıp aynı şehre dönerek kentlerin her birini ziyaret etmeli ve en kısa turu bulmalıdır.**

Anahtarlar*—gezgin, satıcı, dijsktra, takip, yol, maliyet, (anahtar kelimeler***).**

I. GİRİŞ (*GEZGIN SATICI PROBLEMI*)

Günümüzün karmaşık ve zor koşulları problemlere hızlı ve kolay çözüm veren yeni çözüm yöntemleri arayışına neden olmuştur. Özellikle sert(hard) optimizasyon teknikleri yerine, yumuşak hesaplama (soft computing) ve evrimsel algoritma (evolutionary algorithm) kullanımı ön plana çıkmıştır. Evrimsel yaklaşımlardan olan genetik algoritmalar da, bu arayışlar içinde önemli bir yer tutmaya başlamıştır. Gezgin satıcı problemi (GSP) ilk olarak 1930'lu yıllarda matematiksel olarak tanımlanmıştır. Problem tanımı basit olmasına rağmen çözümü zordur. Problemde kullanılan şehir sayısının artışına paralel olarak çözüm uzayı genişlemekte, problemin çözüm zamanı ve çözüm zorluğu artış göstermektedir. Bu sebeple problem çözümünde analitik çözüm yöntemleri yetersiz kalmaktadır. Tüm çözüm uzayını taramak yerine mantıksal çıkarımlar ile çözüm uzayında kısmi taramalar yapan meta-sezgisel yöntemler problemin çözümünü garanti etmemekle beraber, çözüm maliyetini azaltmaktadır[1].

A. *Gezgin Satıcı Probleminin Çözümü*

Elinde ziyaret edilecek şehirler listesi bulunan bir satıcı kendi şehrinden başlayıp tüm şehirlere bir kez uğrayarak tekrar kendi şehrine dönmesi gerekiyorsa bu turu birçok farklı sırada tamamlayabilir. Satıcının tüm şehirlerin diğer şehirlere olan mesafelerini bildiğini düşünelim. Böylece minimum mesafeyi içeren şehir sırasını kolayca bulabileceği anlamına gelmemektedir. Minimum mesafeyi bulabilme problemine gezgin satıcı problemi (GSP) denilmektedir. GSP en dikkat çekici kombinatoriyel optimizasyon problemlerindendir. Tanımlaması basit, simülasyonunun zorluğu ile ün salmış ve hala etkili algoritmalar bulunmaya çalışılan bir problemdir. Problemde başlangıç şehri verilmişse, mümkün olan Hamilton yolları sayısı geriye kalan (n-1) adet şehrin yer değişmesine, yani (n-1)!'e eşit olmaktadır. Bu durumda problem basit olmasına rağmen, problemin çözümünü çözüm uzayının tamamını taramakla bulmak çok iyi bir yaklaşım değildir. Problemin en azından bir basit çözümünün olacağı kesindir. Bu sebeple gezgin satıcı problemlerinin çözümünde sezgisel ve meta sezgisel tekniklerin kullanılması etkin bir yoldur[2].

B. *Gezgin Satıcı Probleminin Matematiksel Modeli*

$$\text{Minimize: } Z = \sum_{i=1}^{n}\sum_{j=1,i\neq j}^{n} x(i,j)d(i,j) \quad (1)$$

Kısıtlar:

$$\sum_{j=1,j\neq i}^{n} x(i,j) = 1, i = 1,2,\dots,n \quad (2)$$

$$\sum_{i=1,i\neq j}^{n} x(i,j) = 1, j = 1,2,\dots,n \quad (3)$$

$$\sum_{i,j\in S,i\neq j} x(i,j) \leq |\,S\,| - 1, \forall\, S \subset \{1,2,\dots,n\} \quad (4)$$

$$x(i,j) = \begin{cases} 1, i \text{ noktasından } j \text{ noktasına gidiliyor ise} \\ 0, i \text{ noktasından } j \text{ noktasına gidilmiyor ise} \end{cases} \quad (5)$$

Şekil 1. Gezgin Satıcı Problemi Matematiksel Modeli

GSP'nin amaç fonksiyonu (1) numaralı eşitlikle verilmektedir. Burada d(i, j) ifadesi i ve j noktaları arasındaki mesafeyi göstermektedir. x(i, j) ise i noktasından j noktasına gidilip gidilmediğini ifade etmektedir. (2) ve (3) numaralı eşitlikler her bir noktaya yalnız bir kez uğranacağını garanti altına almaya yöneliktir. (2) numaralı eşitliğe göre her noktadan sadece bir kez çıkılacak, (3) numaralı eşitliğe göre her noktaya yalnızca bir kez gidilecektir. (4) numaralı eşitlik ise oluşabilecek alt turlardan kurtulmaya yönelik olan alt tur eleme kısıtıdır. (5) numaralı eşitlikte x(i, j)'nin 1 olması i noktasından j noktasına gidildiğini, 0 olması ise gidilmediğini göstermektedir[3].

II. DIJKSTRA ALGORITMASI

Dijkstra algoritması, örneğin yol ağlarını temsil edebilen bir grafikteki düğümler arasındaki en kısa yolları bulmak için bir algoritmadır. Bilgisayar bilimcisi Edsger W. Dijkstra tarafından 1956'da tasarlandı ve üç yıl sonra yayınlandı. Algoritma birçok varyantta mevcuttur. Dijkstra algoritması en kısayolu belirlerken Greedy(Açgözlü) yaklaşımını kullanır. Yani bir düğümden diğer bir düğüme geçerken olası en iyi yerel çözümü göz önüne alır. Her seferinde bir sonraki düğüme ilerleme Greedy yaklaşımına göre yapılır[4].

Dijkstra graf arama algoritmasının çalışma mantığı gereği belirlenen yüzeylerde başlangıç noktasından tüm geçiş noktalarına olan mesafeyi hesaplaması nedeni ile birden çok katlar arası geçişin mümkün olduğu çok katlı bir binada en kısa yol planlaması Dijkstra graf arama algoritması kullanılarak yapılmıştır. Graf arama algoritmaları içinde sınırlı alanlarda en kısa yolu bulurken, Dijkstra algoritması kullanılarak aramanın doğruluğu ve mevcut alternatifler içinde en kısa yolun bulunması açısından iyi bir performans elde edilir.[5].

A. *Dijkstra Algoritması Sözde Kodu*

1/	$b(i, j)$ ekle → kalıcı_liste;
2/	$r=1$;
3/	{ Döngü {
3/1/1	Etiketleme yap $[S_t,r]=[(S_r+p_{rt}),(r)]$; (r. düğümün tüm t komşuları için)
3/1/2	t düğümlerini ekle → geçici_liste;
3/1/3	Döngü bitir; }
3/2	Eğer { t. düğüm önceden etiketlenmişse ($[S_t,v]$)
3/2/1	Eğer { $(S_k+p_{kt})<S_t$ { $[S_t,v]\leftarrow[(S_r+p_{rt}),(r)]$;}}
3/3	Eğer { ($\forall$ A_ij ∈ kalıcı_liste) {bitir}
3/4	r=min(geçici_liste); }

Şekil 2. Dijkstra Algortimasının Sözde Kodu[5].

Bu algoritmada tüm düğümler için erişilme maliyeti hesaplanarak en kısa yolun bulunması garanti edilmiş olur. Daha iyi bir erişim maliyetinin olmadığı yani daha kısa bir yolun bulunmadığı durumda düğüm kalıcı listeye eklenir. Tüm düğümler kalıcı listeye eklendiğinde program durdurulur.

B. *Dijkstra Algoritması Kodu*

```
Dijkstra(G,w,s)                          //Başlat
{
     for(each u ∈ V)
     {
          d[u]= ∞;
          colur[u]=white;
     }
     d[s]=0;
     pred[s]=NIL;
     Q = (tüm köşeler kuyruk);

     while (Non-Empty(Q))        //Tüm köşeleri işle.
     {
          u=Extract-Min(Q);       //Yeni köşe bul.
          for (each v ∈ Adj[u] )
               if(d[u] + w(u, v) < d[v])
               {
                       d[v] = d[u] + w(u,v);
                       Azaltma-Anahtarı(Q, v, d[v]);
                       pred[v] = u;
               }
          color[u] = black;
     }
}
```

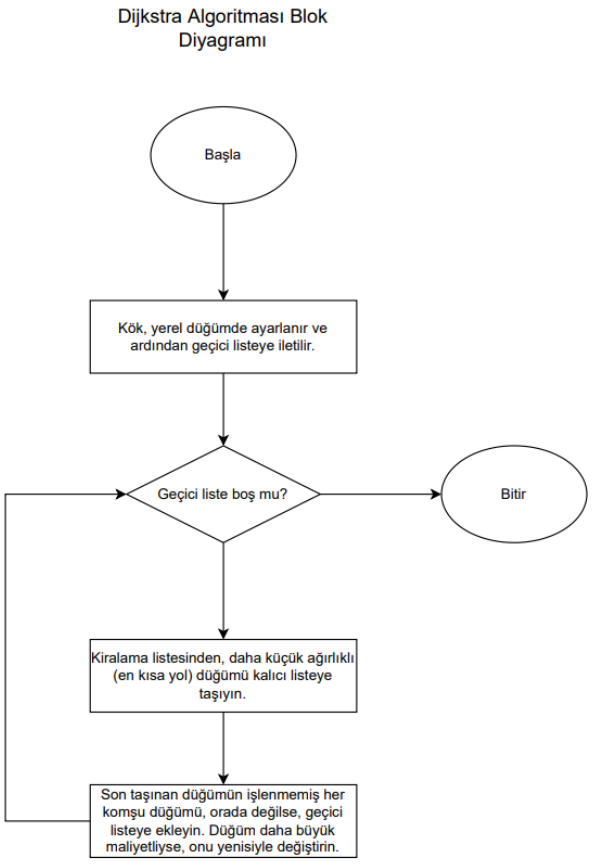
Şekil 3. Dijkstra Algoritması Kodu

- Başlangıç düğümü (node) olarak s düğümünü seçtiğimizi düşünelim.
- Algoritma başlangıçta bütün düğümlere henüz erişim olmadığını kabul ederek sonsuz (∞) değeri atar. Yani başlangıç durumunda henüz hiçbir düğüme gidemiyoruz.
- Ardından başlangıç düğümünün komşusu olan bütün düğümleri dolaşarak bu düğümlere ulaşım mesafesini güncellenir. Adj[s]= {a,b}, a ve b üzerindeki düğümleri güncellenir.
- Bu güncelleme işleminden sonra güncellenen düğümlerin komşularını günceller ve bütün düğümler güncellenene ve şekil üzerinde yeni bir güncelleme olmayana kadar bu işlem devam eder.
- Bu şekilde sonuç olarak en kısa maliyeti bulur ve kullanıcıya gösterir.

C. *Dijkstra Algoritmasının Çalışma Mantığı*

- Dijkstra'nın Algoritması temel olarak seçtiğiniz düğümde (kaynak düğüm) başlar ve bu düğüm ile grafikteki diğer tüm düğümler arasındaki en kısa yolu bulmak için grafiği analiz eder.
- Algoritma, her düğümden kaynak düğüme olan şu anda bilinen en kısa mesafeyi izler ve daha kısa bir yol bulursa bu değerleri güncelleştirir.
- Algoritma kaynak düğüm ile başka bir düğüm arasındaki en kısa yolu bulduğunda, bu düğüm "ziyaret edildi" olarak işaretlenir ve yola eklenir.
- İşlem, grafikteki tüm düğümler yola eklenene kadar devam eder. Bu şekilde, her düğüme ulaşmak için mümkün olan en kısa yolu izleyerek kaynak düğümü diğer tüm düğümlere bağlayan bir yolumuz olur.

D. *Dijkstra Algoritmasının Blok Diyagramı*



Şekil 3. Dijkstra Algoritması Akış Şeması.

III. BIGO NOTASYONU

BigO gösterimi, argüman belirli bir değere veya sonsuzluğa doğru eğilim gösterdiğinde bir fonksiyonun sınırlayıcı davranışını tanımlayan matematiksel bir gösterimdir. Big O, Paul Bachmann, Edmund Landau, ve diğerleri tarafından icat edilen ve toplu olarak Bachmann-Landau gösterimi veya asimptotik gösterim olarak adlandırılan bir notasyon ailesinin bir üyesidir. O harfi, Bachmann tarafından Ordnung'u temsil etmek için seçildi, yani yaklaşım sırası. Bilgisayar bilimlerinde, algoritmaları, giriş boyutu büyüdükçe çalışma süresi veya alan gereksinimlerinin nasıl büyüdüğüne göre sınıflandırmak için Big O gösterimi kullanılır. Analitik sayılar teorisinde, Big O gösterimi genellikle aritmetik bir fonksiyon ile daha iyi anlaşılmış bir yaklaşım arasındaki farka olan sınırı ifade etmek için kullanılır; Böyle bir farkın ünlü bir örneği, asal sayı teoremindeki kalan terimdir. Big O gösterimi, benzer tahminler sağlamak için diğer birçok alanda da kullanılır.[6]

BigO gösterimi, fonksiyonları büyüme oranlarına göre karakterize eder: aynı büyüme hızına sahip farklı fonksiyonlar aynı O gösterimi kullanılarak temsil edilebilir. O harfi kullanılır, çünkü bir fonksiyonun büyüme hızına fonksiyonun sırası da denir. Bir fonksiyonun büyük O gösterimi açısından tanımlanması genellikle yalnızca fonksiyonun büyüme hızı üzerinde bir üst sınır sağlar.[6]

A. BigO ve Dijkstra Algoritması

Dijkstra'nın algoritması, tek bir kaynak problemini çözmek için genişlik-ilk aramayı (BFS) kullanır. Ancak, özgün BFS'den farklı olarak, normal bir ilk giren ilk çıkar kuyruğu yerine bir öncelik kuyruğu kullanır. Her öğenin önceliği, ona kaynağından ulaşmanın maliyetidir.[7]

Grafikte şunlar bulunur:

- Algoritmada v veya u
- İki düğümü birbirine bağlayan ağırlıklı kenarlar: (u, v) bir kenarı ve w(u, v) ağırlığını gösterir.
- Dağıtım – kaynak düğümden grafikteki her s düğüme minimum mesafelerin bir dizisi. Başlangıçta, ve diğer tüm düğümler v için , dist(v) = \inftydist(ler) = 0. DiziDağıtım, her düğüme en kısa mesafe bulunduğunda yeniden hesaplanır ve sonlandırılır.
- Q – grafikteki tüm düğümlerin öncelik sırası. İlerlemenin sonunda, Q boş olacak.
- S – algoritma tarafından hangi düğümlerin ziyaret edildiğini gösteren bir küme. İlerlemenin sonunda, S grafiğin tüm düğümlerini içerecektir.
- Düğümü en küçüğü v dist(v) ile açın Q. İlk çalıştırmada, kaynak düğüm s seçilecektir çünkü dist(ler) = 0 başlatmada.
- Ziyaret edildiğini belirtmek için, öğesine düğüm ekleyin.
- Geçerli düğümün her bitişik düğümü için değerleri aşağıdaki gibi güncelleştirinDağıtım: (1) eğer dist(v) + ağırlık(v, u) < dist(u), bu nedenle yeni minimum mesafe değerine güncelleyindist(u), (2) aksi takdirde dist(u).v
- İlerleme boş olduğunda Q veya başka bir deyişle, tüm düğümleri içerdiğinde S durur, bu da her düğümün ziyaret edildiği anlamına gelir.

B. Dijkstra Algoritmasının Zaman Karmaşıklığı Hesabı

- **Bu durum şu durumlarda gerçekleşir:** Verilen grafik G=(V, E) bir bitişiklik matrisi olarak temsil edilir. Burada w[u, v] kenarın (u, v)ağırlığını depolar.
- Öncelik kuyruğu sıralanmamış bir liste olarak temsil Q edilir.

  Sırasıyla grafikteki kenarların ve köşelerin sayısı olsun ve |V| olsun| E|. Daha sonra zaman karmaşıklığı hesaplanır:
- Tüm |V| köşeleri eklemek zaman Q alırO(|V|).
- Düğümü en azıyla Dağıtım kaldırmak zaman alır O(|V|) ve yalnızca O(1) yeniden hesaplama dist[u] ve güncelleme Q. Burada bir bitişiklik matrisi kullandığımız için, Dağıtım diziyi güncellemek için köşeler için |V| döngü yapmamız gerekir.
- Döngünün her yinelemesi için geçen süreO(|V|), döngü Q başına bir köşe silindiği için 'dir.
- Böylece, toplam zaman karmaşıklığı O(|V|) + O(|V|) \times O(|V|) = O(|V|^2).
- **Bu durum şu durumlarda geçerlidir:** Verilen grafik G=(V, E) bir bitişiklik listesi olarak temsil edilir.
- Öncelik kuyruğu ikili yığın veya Fibonacci yığını olarak temsil Q edilir.
- Köşelerin ilk öncelik kuyruğunu |V| oluşturmak zaman alırO(|V|).
- Bitişiklik listesi gösterimiyle, grafiğin tüm köşeleri BFS kullanılarak geçirilebilir. Bu nedenle, tüm köşelerin komşuları Dağıtım üzerinde yinelemek ve algoritmanın çalışması boyunca değerlerini güncellemek zaman alır O(|E|).
- Döngünün her yinelemesi için geçen süreO(|V|), döngü Q başına bir köşe kaldırıldığı içindir.
- İkili yığın veri yapısı, ayıklama-min (düğümü minimum Dağıtımdeğerle kaldırma) ve bir öğeyi O(log|V|) zamanında güncellememize (yeniden hesaplama dist[u]) olanak tanır.
- Bu nedenle, zaman karmaşıklığı, yani O(|V|) + O(|E| x log|V|) + O(|V| x log|V|)O((| E|+|V|) x log|V|) = O(|E| x log|V|), bağlı bir grafik |E|>=|V| - 1 olduğu gibi G.
- Daha sonra bir Fibonacci yığını kullanarak zaman karmaşıklığını hesaplayacağız. Fibonacci yığını, yeni bir eleman eklememize ve düğümü minimum Dağıtım in O(log|V)O(1) ile çıkarmamıza izin verir. Bu nedenle, zaman karmaşıklığı şöyle olacaktır:

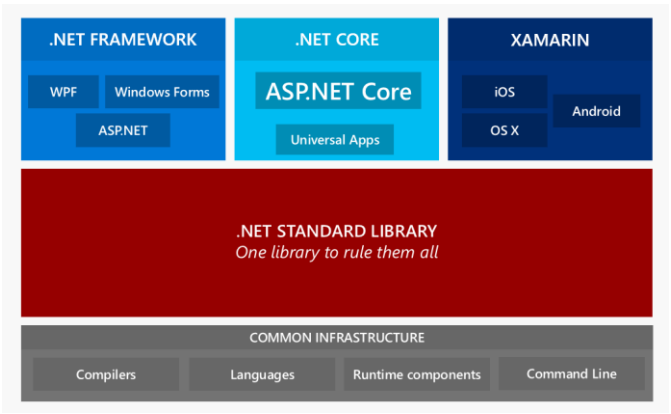
- Döngünün her yinelemesi ve ekstrakt-min için geçen süreO(|V|), döngü Q başına bir köşe kaldırıldığı içindir.
- Tüm köşelerin komşuları Dağıtım üzerinde yineleme yapmak ve algoritmanın çalıştırılması için değerlerini güncellemek zaman alır O(|E|). Her öncelik değeri güncelleştirmesi zaman aldığından O(log|V|), tüm Dağıtım hesaplama ve öncelik değeri güncelleştirmelerinin toplamı zaman alır O(|E| x log|V|).
- Böylece genel zaman karmaşıklığı O(|V| + |E| x log|V|) olur.

IV. *KULLANILAN YAZILIMSAL MIMARI*

Bu, geçen yüzyılda, daha spesifik olarak, altmışlı yılların sonunda, yazılım geliştirme şeklimizin binaları inşa etme şeklimize oldukça benzediğini öne sürdüklerinde oldu. Bu yüzden yazılım mimarisi adını aldık. Tıpkı bir mimarın bir binayı tasarlaması ve bu tasarıma dayanarak yapımını denetlemesi gibi, bir yazılım mimarının temel amacı da yazılım uygulamasının iyi uygulanmasını sağlamaktır; ve iyi bir uygulama, harika bir çözümün tasarımını gerektirir. .NET 6, C# 10 ile birlikte teslim edilir. Bu kadar çok platform ve cihazı hedefleyen .NET yaklaşımı göz önüne alındığında, C# şu anda dünyanın en çok kullanılan programlama dillerinden biridir ve küçük cihazlardan farklı işletim sistemlerinde ve ortamlarda büyük sunuculara kadar her şeyde çalışır[8].

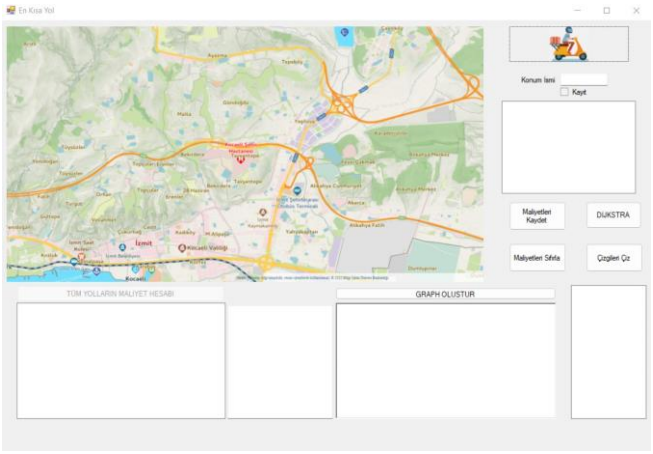
.NET, Microsoft tarafından tasarlanan ve geliştirilen bir yazılım çerçevesidir. .Net frameworkün ilk sürümü 2002 yılında gelen 1.0 idi. Kolay bir deyişle, C#, VB.Net vb. gibi farklı dillerde yazılmış programların derlenmesi ve yürütülmesi için kullanılan sanal bir makinedir. Form tabanlı uygulamalar, Web tabanlı uygulamalar ve Web servisleri geliştirmek için kullanılır. .Net platformunda çeşitli programlama dilleri vardır, VB.Net ve C# en yaygın olanlardır. Windows, telefon, web vb. İçin uygulamalar oluşturmak için kullanılır. Birçok işlevsellik sağlar ve ayrıca endüstri standartlarını destekler[8].

WinForms : Form – Tabanlı başvurular bu kategori altında değerlendirilir. Basit bir ifadeyle, dosya sistemini okuyan ve yazan istemci tabanlı uygulamaların bu kategoriye girdiğini söyleyebiliriz[8].

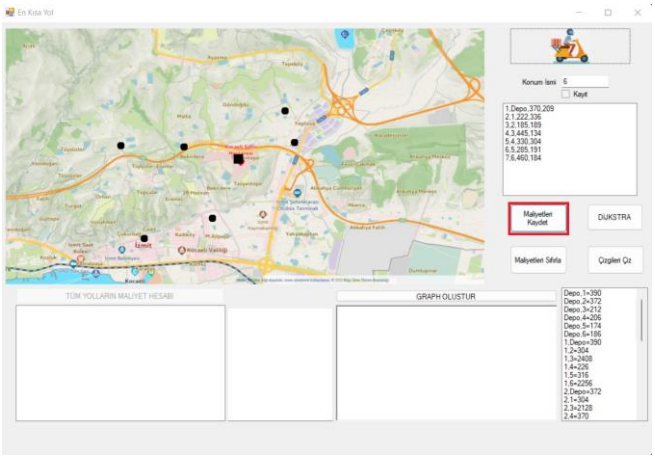


Şekil 4. Yazılımsal Mimari[9]

V. *UYGULAMANIN ÇALIŞMA MANTIĞI*

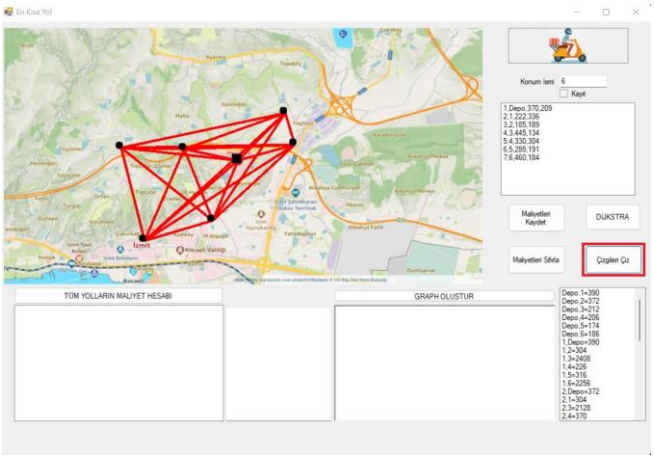


Şekil 5. Uygulama giriş ekranı.



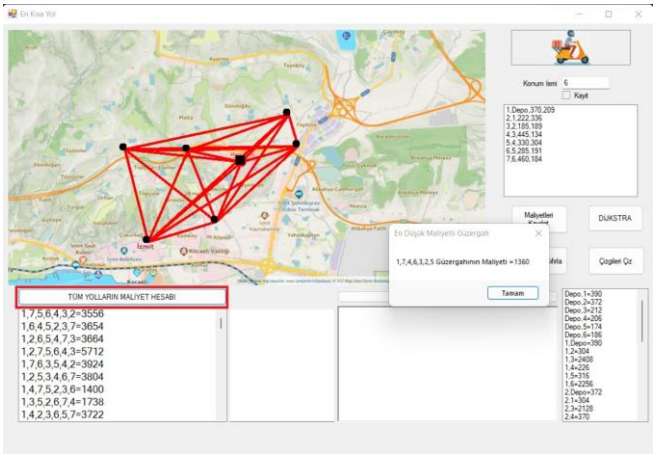
Şekil 6. Seçim ekranı.

Depo ve sipariş noktalarını seçip maliyetleri kaydet butonu ile her bir noktanın birbirine olan maliyetini hesaplıyoruz.



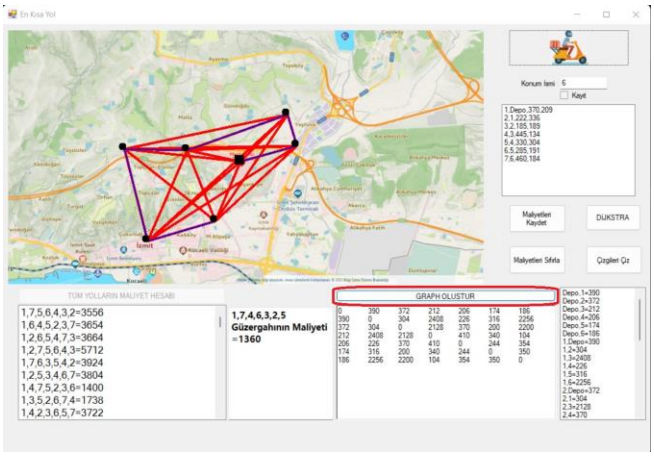
Şekil 7. Çizgileri çiz ekranı.

Çizgileri çiz ile her bir noktanın birbirine olan yolunu çiziyoruz.



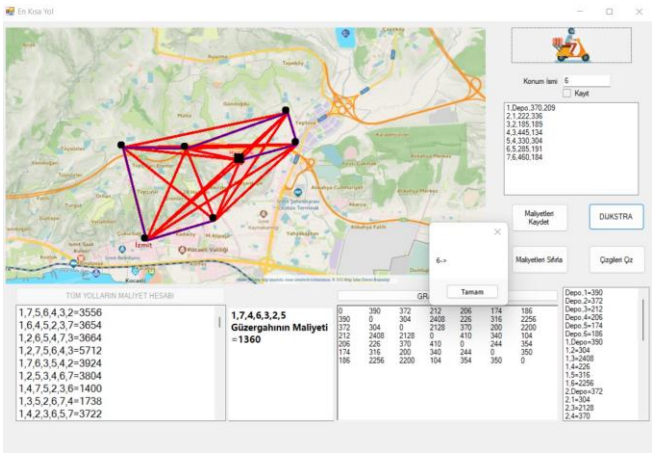
Şekil 8. Hesap ekranı.

Tüm yolların maliyet hesabı ile tüm olasılıklar arasında en kısa güzergahı tespit edip o yolu seçiyoruz.



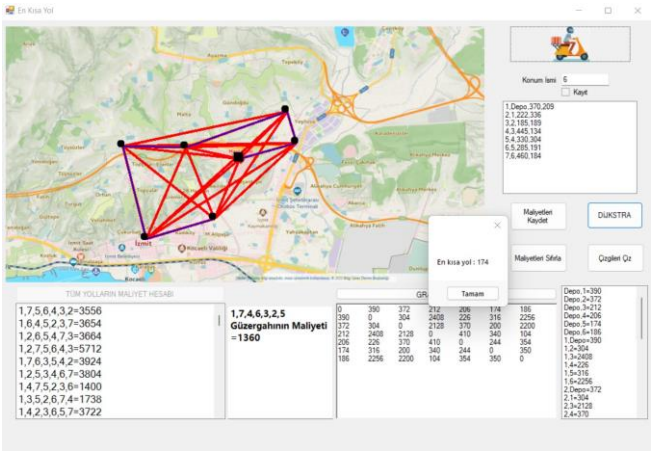
Şekil 9. Graph tablosu oluşturma ekranı.

Graph oluştur butonu ile graph tablosu oluşturuyoruz.

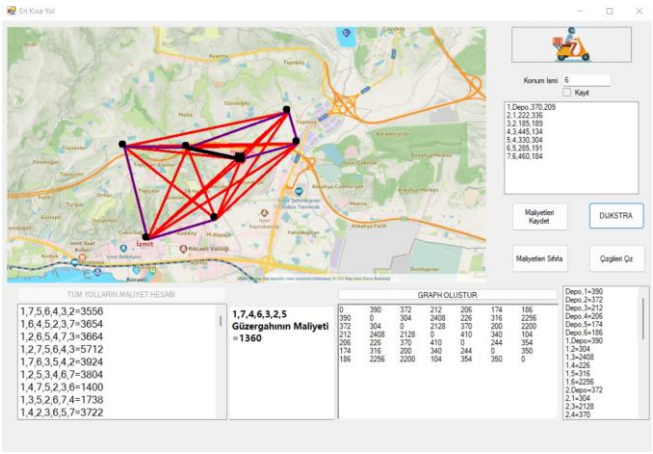


Şekil 10. Dijkstra.

DIJKSTRA butonu ile dijkstra algoritması kullanarak son sipariş noktası ile depo arasındaki en kısa rotayı buluyoruz.



Şekil 11. En kısa yolu gösteren ekran.



Şekil 12. Programın son ekran görüntüsü.

Programımızın son görüntüsü; Kırmızı çizgi = hiç geçilmemiş yollar. Mor çizgi = Siparişleri teslim ederken kullanılan yol yani en kısa yol, siyah çizgi = depoya dönüş için seçilen en kısa yol.

VI. *Karşılaştığımız Zorluklar ve Faydalar*

Projeyi geliştirirken öncelikle Dijkstra Algortimasının olası yönlerini araştırıp tartıştık bu süreçte algoritmanın projeye uyarlama konusunda bazı sorunlar yaşadık. Kullandığımız C# form ile Gezgin Satıcı Problemini bağdaştırmak biraz zamanımızı aldı kaynak sıkıntısı buna büyük bir etkendi. Proje raporu hazırlarken dijkstra algoritmasının şeması ve BigO notasyonu hesaplaması kısmında zorluklarla karşılaştık bir çok kaynak taradık. Nitekim verdiğimiz uğraşlar sonucunda problemin çözümüne kavuştuk. Bu problemin çözümünde geliştirdiğimiz proje bizlere algoritma kullanmanın yöntemlerini öğretti ve en kısa yol probleminin çözümünü kavramayı sağladı. Sonuç olarak Gezgin Satıcı Problemi konusunda fikirlerimize somutluk sağladık.

KAYNAKLAR

[1] S. Kuzu, O. Önay, U. Şen, M. Tunçer, B. F. Yıldırım, T. Keskintürk, Gezgin satıcı problemlerinin meta-sezgiseller ile çözümü, İstanbul Üniversitesi İşletme Fakültesi Dergisi, Cilt/Vol:43, Sayı/No:1, 2014, 1-27

[2] https://tr.wikipedia.org/wiki/Gezgin_sat%C4%B1c%C4%B1_problpro

[3] J. – Y. Potvin, Genetic Algorithms for the travelling salesman problem, forthcoming in Annals of Operations Research on "Metaheuristics in Combinatorial Optimization", eds. G. Laporte an I. H. Osman (1996)

[4] https://www.freecodecamp.org/news/dijkstras-shortest-path-algorithm-visual-introduction/#:~:text=Dijkstra's%20Algorithm%20finds%20the%20shortest,node%20and%20all%20other%20nodes.

[5] https://www.baeldung.com/cs/dijkstra-time-complexity

[6] https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/

[7] https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/

[8] https://www.c-sharpcorner.com/article/software-architecture-and-patterns/

[9] https://onnurkarakus.medium.com/nedir-bu-microsoft-net-core-2b0d91d3cb1e