Google Maps API Kullanarak Anlık Trafik Bilgisine Dayalı Gezgin Satıcı Problemi Uygulaması

Real-Time Application of Travelling Salesman Problem Using Google Maps API

Mehmet Hamza EROL[1], Faruk BULUT[2]

[1]İzmir Atatürk Lisesi, Konak İZMİR
mehmethamzaerol11@gmail.com

[2]Mühendislik ve Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü,
İstanbul Rumeli Üniversitesi
faruk.bulut@rumeli.edu.tr

Özetçe

Gezgin Satıcı Problemi, en az maliyet ile faklı konumlarda bulunan noktaların ziyaret edilmesinden sonra başlangıç noktasına dönülmesi olarak tanımlanır. Bu çalışmada, Google haritalarından alınan anlık trafik yoğunluk bilgisi kullanılarak en uygun Gezgin Satıcı Problemine ait rotanın hesaplandığı bir gerçek dünya uygulaması üzerinde çalışılmıştır. Geliştirilen masa üstü uygulamasında Tam Kapsamlı Arama, Sezgisel A-Star Arama, BitMask Dinamik Programlama, Dal-Sınır Algoritması ve Açgözlü Arama gibi farklı yöntemler bu problemde kullanılmış ve Açgözlü Arama yöntemi hariç diğer tüm teknikler aynı sonuçları vermiştir. Elde edilen en az süreli ya da en az kısa mesafeli rota, Google haritalarında gösterilerek ve farklı özellikler de eklenerek kullanıcı dostu bir ara yüz geliştirilmiştir.

Abstract

The Travelling Salesman Problem is defined as returning to the starting point after visiting all the points with the least cost. In this study, a real-world application that calculates the route of the Travelling Salesman Problem using the current traffic intensity information from Google Maps is prepared. Different methods such as Exhaustive Search, Heuristic A-Star Search, BitMask Dynamic Programming, Branch-and-Bound Algorithm and Greedy Search have been used in developing the desktop application. All these methods but Greedy Search have given the same routes. A user-friendly interface, displaying the shortest route in distance or duration on Google Maps, has been developed by adding different features.

1. Giriş

Faklı konumlarda bulunan *N* adet noktayı bisikletle, araç ile ya da yayan olarak ziyaret ettikten sonra geri dönmeniz gerekirse en uygun rotanız nasıl olur? İşte bu soru yaygın bilinen adıyla Gezgin Satıcı Problemidir (*Travelling Salesman Problem*, TSP). Bilindiği üzere TSP başlangıç noktası olarak seçilen bir noktadan başlayarak noktaların her birine birer defa uğrayıp başlangıç noktasına tekrar dönme işlemini en az maliyetle yapma problemdir. Günlük hayattan örnek olarak bir kargonun müşterilerine elindeki malları dağıtıp tekrar başladığı noktaya dönmesi ve bunu en az maliyet ile yapmaya çalışması bu problemin aslında önemli bir problem olduğunu gösterir. Bu durum gazete dağıtıcılarını, kuryeleri, bayramda akrabalarınız ziyaret etmek isteyen birini, müşterilerine mal veya hizmeti en kısa zamanda dağıtmak isteyen satıcıyı, turistik mekânları sırasıyla en az zamanda gezmek isteyen bir turist kafilesini ve kargo firmalarını yakından ilgilendiriyor.

TSP algoritması üzerine şu ana dek birçok bilimsel çalışma ve uygulama yapılmıştır. İlk olarak 1930'da bilimsel yöntemler kullanılarak uygun bir çözüm yöntemi önerilmiştir. Bu problem lineer bir çözüm yöntemi ile çözülemediği için NP-hard (*Non-Polynomial*) yani polinomsal olmayan problem grubunda değerlendirilmiştir. Bilinen en yaygın TSP çözüm teknikleri arasında Tam Kapsamlı arama (*Exhaustive Search*), Özyineleme (*Recursion*), Açgözlü arama (*Greedy Algorithm*), Dinamik Programlama (*Dynamic Programming*) ve Sezgisel Arama (*Heuristic Search*) yöntemleri vardır. Bu tarz algoritmaların tamamı en az maliyetli rotayı vermektedir [1].

TSP problemi, Özyineleme yöntemi ile $O(N!)$ zaman karmaşıklığında çözülmektedir. Bilindiği üzere büyük *O* gösterimi, kullanılan çözüm algoritmasının zaman karmaşıklığını (*Time Complexity*) göstermektedir. Her ne kadar Özyineleme yöntemi optimizasyon çözüm tekniklerine göre her zaman için en iyi sonucu vermiş olsa da zaman karmaşıklığının büyük olması ve hesaplama süresinin uzun oluşu nedeni ile pek tercih edilmemektedir [2].

Diğer taraftan TSP problemi üzerine optimizasyon algoritmaları ile yapılmış çözüm teknikleri mevcuttur. Genetik Algoritmalar, Arı kolonisi (*Artificial Bee Colony*), Karınca Kolonisi (*Ant Colony*), Çok Katmanlı Algılayıcılar

(*Multi-Layer Perceptrons*), PSO (*Particle Swarm Optimization*), Meyve Sineği Optimizasyonu (*Fruit Fly*) gibi birçok metot şu ana dek bu problem üzerine uygulanmıştır [3]. Fakat optimizasyon yöntemleri, TSP için en iyi çözümü bulmayı garanti etmez, sadece mevcut çözümlerden birini bulur. Ziyaret edilmesi gereken konum sayısının artması durumunda hesaplama süresi de üstsel olarak artmaktadır. Bu nedenle TSP uygulamalarında, tüm durumların tek tek bulunup en iyi sonucun bulunması maliyetli bir seçenek olacağı için optimizasyon yöntemleri sıklıkla tercih edilmektedir.

Yapılan makale taramasında şu ana kadar TSP problemi üzerine geliştirilen en iyi çözüm tekniğinin BitMask Dinamik Programlama yöntemi olduğu görülmüştür [4]. BitMask yöntemi ile kısa bir hesaplama zamanı içerisinde en iyi sonuç hesaplanmaktadır.

Bu çalışmada TSP probleminin farklı algoritmik yöntemlerle çözülerek güncel hayata uyarlanması amaçlanmıştır. Projede, Google Maps API (*Application Programming Interface*) hizmetlerinden yararlanarak herkes tarafından kullanılabilecek bir uygulama Java platformunda geliştirilmiştir. Uygulama, anlık olarak Google haritalarından trafik yoğunluk bilgisini alarak ziyaret edilmesi gereken *N* adet noktanın en kısa zamanda ya da en kısa km ile gezilmesini sağlamaktadır. Anlın trafik bilgisi Google'ın API hizmetleri tarafından belirli bir limite kadar ücretsiz olarak verilmektedir. Bilindiği üzere API, bir uygulamaya veya servise ait kodların başka uygulamalarda kullanılmasını sağlayan ara yüz ve özellik olarak tanımlanır. API'lar genellikle kullanışlı ara yüzlerdir çünkü uygulama içinde bazı işlevleri yerine getirmek isteyen kullanıcılara sözdizimi kolaylaştırılmış bir şekilde istenen hizmetleri sunar.

Ayrıca hazırlanan görsel bir GUI (*Graphical User Interface*) uygulaması ile de kullanıcı dostu bir çalışma gerçekleştirilmiştir. Ayrıca çalışmada seyahat yöntemi ve aracı olarak yaya, bisiklet ve araç seçenekleri vardır.

Bu çalışmada Özyineleme yöntemi, sezgisel yaklaşımlarla iyileştirilmiş ve hesaplama zamanını olabildiğince en aza indirilmiştir. Aramada kullanılan Özyineleme fonksiyonu, belirlenen bazı kesme (budama) yöntemleri yardımıyla optimize edilmiştir. Bilindiği üzere iyileştirilen bu yöntem Dal-Sınır (*Branch & Bound*) algoritması olarak isimlendirilmektedir. Ayrıca A-Star gibi sezgisel arama yöntemleri ve Branch & Bound gibi kesmeli özyineleme algoritmaları ile farklı veri yapıları kullanılarak en optimal sonucun bulunması sağlanmıştır. Yapılan testlerde zamana ve mesafeye göre en uygun rota seçenekleri Google haritalarında gösterilmiştir. Ayrıca projenin amacı en uygun TSP rotasının tespit edilmesine yönelik olduğu için optimizasyon yöntemleri çalışmaya eklenmemiştir.

Bildirinin geriye kalan kısmı dört bölüm daha içermektedir. Takip eden 2. Bölümde kullanılan algoritmalara, 3. Bölümde elde edilen TSP bulgularına, 4. Bölümde tartışmalara ve son bölümde özete yer verilmiştir.

2. Yöntemler

Seyyar Satıcı Problemi, iki boyutlu düzlem üzerinde ziyaret edilmesi gereken N nokta için en az maliyetli çevrimin bulunması ile ilgilidir. Nokta sayısının artması ile çözüm maliyeti de üssel olarak artmaktadır. Bu nedenle TSP, NP zorluk derecesinde değerlendirilmektedir. Çalışmada kullanılan çözüm teknikerinin teorik açıklamaları kısaca şu şekildedir:

2.1. Tam Kapsamlı Arama (*Exhaustive Search*)

Bilgisayar bilimlerinde Kaba Kuvvet (*Brute Force*) yöntemi olarak da bilinen bu tekniğin Türkçe karşılığı Tam Kapsamlı Arama şeklindedir. Bu algoritma ile oluşabilecek tüm durumlar tek tek ele alınıp değerlendirilmekte ve alternatifler içerisinden en iyi olanı seçilmektedir. Her zaman en doğal ve en basit çözüm tekniği olarak kabul edilmekle birlikte en yavaş çözüm tekniği olarak da bilinmektedir. Çünkü eldeki durumlar üzerinde mantıklı elemeler yapılmaz ve işlem sayısı hiçbir yöntemle azaltılmaya çalışılmaz [5].

2.2. Açgözlü (*Greedy*) Arama

Türkçe karşılığı "Açgözlü" olan bu algoritmada, her adımda tüm seçenekler içerisinden en iyi olan seçilir ve değerlendirmeye alınır. Bu algoritma hızlı çalışmasına rağmen her zaman en iyi sonucu vermez hatta en optimale çok uzak bir sonuç da verebilir. Fakat bazı problem türlerinde ve verilen senaryolarda Açgözlü yöntemi doğru sonuca hızlı bir şekilde ulaşabilmektedir [6].

2.3. A-Star (A* *Heuristic*) Sezgisel Arama

A*, sezgisel bir arama algoritmasıdır ve en yaygın kullanım fonksiyonu şu şekildedir:

$$f(x) = h(x) + g(x) \tag{1}$$

A* yönteminde kullanılan $f(x)$ fonksiyonu $g(x)$ ve $h(x)$'in toplamlarıyla elde edilir. $g(x)$ fonksiyonu x durumuna gelene kadar yapılan maliyetlerin toplamını gösterir. $h(x)$ sezgisel fonksiyonu ise x durumundan sonra yapılabilecek tahmini masrafların toplamını gösterir. Herhangi bir aşamaya kadar elde edilen tüm X durumları içerisinden en iyi olanı üzerinden işlemler devam eder.

Problem çözümü aşamalarında elde edilen alternatifler $X=\{x_1, x_2, x_3, \ldots, x_N\}$ kümesinde tutulur. Çalışmamızda X kümesi Heap veri yapısına (Priority Queue) aktarılarak en iyi $f(x)$ fonksiyonuna sahip x durumu üzerinden işleme devam edilmiştir. Böylelikle adım adım en optimal sonuca sezgisel bir yöntemle ulaşılır [7].

2.4. Dal-Sınır (*Branch & Bound, B&B*) Yöntemi

Optimize edilmiş bir özyineleme tekniği olan B&B yöntemi, A-Star algoritmasına benzer bir yapıdadır. Her iki algoritmada da $h(x)$ ve $g(x)$ fonksiyonları bulunur ve aynı amaca hizmet eder. B&B'nin Türkçe karşılığı "Dallan ve Sınırla" şeklindedir ki bu da bize bu algoritmanın çalışma prensibini açıklamaktadır.

Bu algoritmada özyinelemeli bir fonksiyon vardır. Her aşamada *h* ve *g* fonksiyonlarının değerleri temel alınarak bulunulan aşamadan varılabilecek tahmini en iyi sonuç elde edilir. Elde edilen bu sonuç daha önce bulunmuş ve en iyi sonuçtan daha kötü ise dallanma kesilir (budanır). Eğer durum tam tersi ise özyinelemeli fonksiyonun kendini çağırması yani dallanması işlemi devam eder. Son aşamada bulunan sonuç o ana kadar bulunan en iyi sonuçtan daha iyi ise sonuç güncellenir. Böylelikle sezgisel bir yöntemle sonuca işlemler sonunda ulaşılmış olunur [8].

2.5. BitMask Dinamik Programlama

Bilindiği üzere bilgisayarın arka planında her şey 0 ve 1 bitlerinden oluşmaktadır. Bu bitlerin farklı permütasyonlardaki dizilişleri ile veri kodlanır. Bu dizilişlerin oluşturduğu şekle BitMask denir. Örneğin 'A' harfinin bilgisayardaki ASCII kodu "01000001" şeklindedir. Bu dizilim bize ikilik tabanda bir sayıyı ifade eder ve onluk tabana çevrildiğinde 65 sayısı elde edilir.

Buradaki duruma benzer bir şekilde BitMask'ı kullanarak elimizdeki bazı durumları kodlayabiliriz. Örneğin *N* tane gezilecek noktadan gezilenlere 1, gezilmeyenlere 0 değeri verilir. Eğer gezilen ve gezilmeyen şehirler belli bir sırada dizilirse ikilik tabanda bir sayı elde edilir. TSP'nin çözümünde de BitMask, dinamik programlama algoritması ile birlikte kullanılmaktadır. Burada daha önceden sonucunu hesaplanmış durumları bir daha hesaplamamak için dinamik programlama ve hangi durumda olduğumuzu belirlemek için ise BitMask kullanılır [9].

BitMask Dinamik programlama yöntemi ile TSP uygulamasının zaman karmaşıklığı büyük O ölçütü ile $O(N^2 \times 2^N)$'e kadar indirilebilir.

3. Deneysel Uygulama ve Bulgular

Deneysel uygulama için Java programlama dili tercih edilmiştir. TSP probleminin çözümü için kullanılan algoritmalar hiçbir yerden kopyalanmadan tarafımızca Java dilinde yazılmıştır. Linux Ubuntu 14.04 ve Windows 10 işletim sisteminde yapılan deneysel uygulama aşağıdaki şu aşamalar ile gerçekleştirilmiştir:

Aşama 1. JDK (Java Development Kit) v8.111 64Bit yazılım geliştirme aracı ve Eclipse Neon v2 geliştirme editörü (IDE) bilgisayara kuruldu.

Aşama 2. GUI için yaygın ve zengin özellikleri bulunduğu için Swing tercih edildi. Görsel GUI uygulamaları için WindowBuilder aracı Eclipse'e kuruldu (https://eclipse.org/windowbuilder).

Aşama 3. Gradle v3.3'i Eclipse'e kurularak sonraki aşamalarda başka projelerin projemize eklenmesi sağlandı (https://gradle.org/gradle-download).

Aşama 4. "google-maps-services-0.1.17.jar" dosyası https://github.com/googlemaps/google-maps-services-java sitesinden indirildi projeye Gradle ile ilave edildi.

Aşama 5. Google Maps servislerinden yararlanabilmek için kendi Google hesabımız ile ücretsiz bir API Key https://developers.google.com/console adresinden alındı.

Aşama 6. Aynı sitede bulunan ve proje ile ilgili API hizmetlerinden aktifleştirilenler şunlardır:

- Directions API,
- Distance Matrix API,
- Elevation API,
- Geocoding API,
- Places API,
- Roads API,
- Time Zone API.

Aşama 7. Google Maps kütüphanesinde bulunan metotlar kullanılarak Google'dan anlık sonuçlar JSON (JavaScript Object Notation) veri biçiminde alındı ve deserialize edildi. Bilindiği üzere JSON, JavaScript tabanlı bir veri alış-veriş formatıdır.

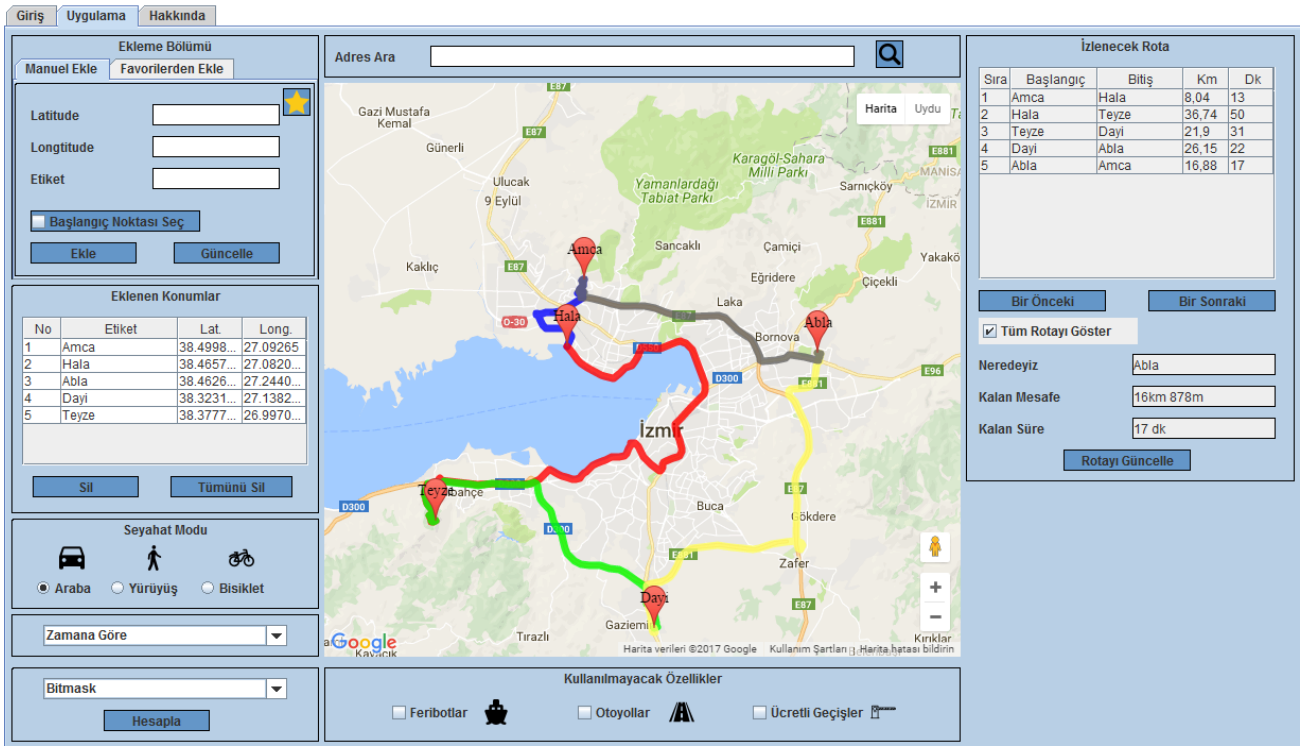
Aşama 8. Java GUI uygulaması, alınan anlık JSON verilerini TSP, B&B ve A-Star gibi algoritmalarda kullanarak en kısa süreli ve en kısa mesafeli rotayı buldu ve aynı GUI içerisinde ilgili harita üzerinde rota gösterildi.

Aşama 9. Anlık Google haritalarının GUI çalışmasına yapıştırılabilmesi ve haritalar üzerinde etkileşimli işlemlerin yapılabilmesi için JxMap API'lerinin JAR dosyaları projeye eklendi (https://www.teamdev.com/jxmaps).

JxMap API kütüphanesindeki metotlar kullanılarak otomatik konum ekleme yapıldı. Ayrıca elde edilen TSP sonuçları Google Maps üzerinde Marker'lar ile gösterildi.

Yazılım tarafından hesaplanan en uygun TSP rotası, Şekil 1'de de görüldüğü üzere Google haritaları üzerinde görüntülenerek güzergâh takibinin daha kolay yapılması sağlandı.

Sol üst tarafta bulunan "Konum Ekleme Bölümü" Google haritasını kullanarak konum ekleme işine yarar. "Favorilerde Ekleme" bölümünde kullanıcı adı ve şifresi ile girildiğinde daha önce eklenmiş olan konumlar tekrar eklenebilir. Ortada tarafta bulunan Google haritası üzerine fare ikonu ile istenilen konum tıklandığında otomatik olarak enlem (*Latitude*) ve boylam (*Longtitude*) değerleri ilgili metin kutucuğuna yerleşmektedir. Ayrıca konum bilgisi Google harita üzerinde yazmaktadır. Kullanıcı sadece ilgili konumun daha kolay akılda tutulması için "Etiket" bilgisini klavye ile girmelidir. Eklenen her bir konum sol tarafta bulunan "Eklenen Konumlar" bölümünde sırasıyla belirecektir. Ayrıca eklenen bir konum daha sonra "Sil" butonu ile silinebilecek ya da "Güncelle" butonu ile etiket bilgisi değiştirilebilecektir. "Tümünü Sil" butonu ile eklenen tüm konumlar silinebilir. Konum ekleme esnasında eklenecek konum "Başlangıç Noktası Seç" butonu ile Başlangıç noktası olarak seçilebillir. Şekil 1'de görüldüğü üzere sırasıyla Amca, Teyze, Hala, Abla, Dayı gibi etiketli gerçek konumlar istenildiği kadar listeye eklenebilir.



Şekil 1: GUI Uygulama Ekranı

Sol alt tarafında "Seyahat Modu" bölümünde Araba, Bisiklet ve Yürüme seçeneklerinden biri seçilmelidir. Ardından "Algoritma Seçiniz" bölümünden istenilen bir TSP algoritması seçilebilir. Görüldüğü üzere TSP rotası farklı algoritmalar kullanılarak elde edilmiştir. "Hesaplama Modunu Seçiniz" ile en uygun rotanın km türünden mesafeye mi yoksa dakika cinsinden süreye göre mi olduğu belirlenebilir ve "Hesapla" butonuna basılabilir. İstenilirse feribotlar, otoyollar ve ücretli geçişler rotaya eklenmeyebilir.

Hesaplama işlemi sonucu orta bölümde bulunan "Rota" listesinde görülmektedir. Yapılan seçimlere göre sırasıyla takip edilmesi gereken rota liste halinde verilmektedir. Listede görüldüğü üzere Google'dan alınan anlık değerlere göre bir noktadan diğer noktaya varış mesafesi ve süresi listede verilmektedir.

Sağ tarafta ise toplam rota mesafesi ve anlık trafik yoğunluğuna göre harcanacak seyahat süresi dakika cinsinden iki ayrı pencerede belirtilmektedir. Ayrıca "Bir Sonraki" ve "Bir Önceki" Butonları ile konumumuz ve rotamız da güncellenmektedir. "Hesaplama Modu Seçimi" mesafeye göre değil de zamana göre yapıldığında elde edilen TSP rotasının bir öncekinden farklı çıktığı gözlemlenebilir. Aynı şekilde seyahat modu bisiklet veya yürüme şeklinde seçildiğinde de elde edilen rota bilgisi ve toplam mesafe ve dakika değerleri de değişmektedir.

4. Tartışma

Yapılan deneysel uygulamalarda projede kullanılan farklı algoritmalar ile aynı rota güzergâhını ve seyahat süresini bulmamız çalışmamızın doğruluğunu kanıtlar niteliktedir.

Şehir içerisinde veya dışarısında belirli yollarda kazı çalışması yapılabileceği gibi trafik kazaları neticesinde anlık yoğunluklar yaşanabilir. Bu tür anlık veriler Google haritaları tarafından kullanıcılarına ücretsiz olarak sunulması projemizin gerçekleştirilmesini mümkün kıldı. Bu sayede etkileşimi bir yazılım ile anlık trafik yoğunluk bilgisine dayalı iki farklı seyahat seçeneği sunulmuş oldu: birincisi en kısa mesafeli rota, diğeri ise anlık trafik yoğunluğuna göre an az süreli seyahat rotası.

Zaman içerisinde yol üzerinde aniden bir trafik kazası olabilir veya kazı çalışması gibi nedenlerle bazı yollar trafiğe birden kapatılabilir. Kullanıcı çizilen rotayı izlerken rota dâhilindeki bir güzergâhta kazı çalışması, trafik kazası gibi günlük yaşamda sıkça karşılaşılan ve ulaşımı aksatan problemler meydana gelebilir. Bu gibi durumlarda kullanıcı belirlenmiş rotayı izlerken Google'dan alınan anlık verilere göre rota tekrar güncellenebilir. Ayrıca ziyaret edilmesi gereken yerler sırasıyla gezilirken bir sonraki gidilecek yere doğru hareket edilmeden önce TSP güzergâhı tekrar güncellenebilir.

Bu uygulama, turistik mekânların yayan veya bisiklet ile ziyaret edilmesinde de kullanım olanağı sağlamaktadır. Ayrıca farklı noktalara bisiklet turlu düzenlenirken ve yürüyüş sporu rotası belirlenirken bu uygulama kullanılabilir.

Bu projenin geliştirilmesi için daha sonra yapılabilecek bazı çalışmalar şunlar olabilir. GUI uygulaması, Android

veya IOS platformuna uyarlanabilir. Bu sayede cep telefonlarında kullanılabilmesi sağlanmış olur. Aynı çalışma JavaScript programlama dili ile de geliştirilip web tabanlı bir uygulama haline dönüştürülebilir. Bu sayede çalışma platformdan bağımsız olarak tarayıcı bir program ile herkes tarafından kullanılabilir hale getirilebilir.

TSP üzerinde çokça çalışılan bir konu olmuştur. Bu nedenle değişik algoritmalar, geliştirilen yöntemler bu çalışmaya ilave edilebilir. Var olan bazı algoritmaların iyileştirilmiş halleri de projeye eklenebilir.

Kaynakça

[1] K. L. Hoffman, M. Padberg, ve G. Rinaldi, "Traveling salesman problem", *In Encyclopedia of operations research and management science*, Springer US, 1573-1578, 2013.

[2] S. ÇOLAK, "Genetik Algoritmalar Yardımı ile Gezgin Satıcı Probleminin Çözümü Üzerine Bir Uygulama", *Çukurova Üniversitesi Sosyal Bilimler Enstitüsü Dergisi*, vol. 19, no. 3, 2010.

[3] A. Rao ve S. K. Hedge, "Literature survey on travelling salesman problem using genetic algorithms", *International Journal of Advanced Research in Eduation Technology* (IJARET), 2, 42-45, 2015.

[4] S. Urrutia, A. Milanés ve A. Løkketangen, "A dynamic programming based local search approach for the double traveling salesman problem with multiple stacks", *International Transactions in Operational Research,* vol 22, no.. 1, 61-75. 2015.

[5] O. Kullmann, "The Science of Brute Force", *Communications of the ACM*, 2017.

[6] M. H. Chen, C. W. Yu ve P. C. Chang, "Block Based Imperial Competitive Algorithm with Greedy Search for Traveling Salesman Problem", *International Scholarly and Scientific Research Innovation*, 8, 793-799, 2014.

[7] S. Aygün ve M. Akçay, "MATLAB Paralel Hesaplama Aracı ile A* Algoritmasının Rota Planlama İçin Analizi", *Genç Mühendisler Sempozyumu*, Türk Mühendisler Birliği, Mayıs, Ankara. 2015.

[8] M. Ç. MUTLU, "Kaynak dengeleme problemi için bir dal ve sınır algoritması (A branch and bound algorithm for resource leveling problem)", *Doktora Tezi, Orta Doğu Teknik Üniversitesi Fen Bilimleri Enstitüsü*, Ankara. 2010.

[9] C. Groër, B.D. Sullivan ve D. Weerapurage, INDDGO: "Integrated network decomposition & dynamic programming for graph optimization", *ORNL/TM-2012/176*, 2012.