T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

AKILLI ULAŞIM SİSTEMİNE YÖNELİK EN KISA YOL BULMA ALGORİTMALARININ PERFORMANS ANALİZİ

YÜKSEK LİSANS TEZİ

Gulkaiyr KALYBEK KYZY

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Prof. Dr. Cemil ÖZ**

Şubat 2019

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

AKILLI ULAŞIM SİSTEMİNE YÖNELİK EN KISA YOL BULMA ALGORİTMALARININ PERFORMANS ANALİZİ

YÜKSEK LİSANS TEZİ

Gulkaiyr KALYBEK KYZY

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 11.02.2019 tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

Prof. Dr.
Cemil ÖZ
Jüri Başkanı

Doç. Dr.
Nilüfer YURTAY
Üye

Doç. Dr.
Metin VARAN
Üye

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Gulkaiyr KALYBEK KYZY

11.02.2019

TEŞEKKÜR

Yüksek lisans eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, teşvik eden, aynı titizlikte beni yönlendiren değerli danışman hocam Prof. Dr. Cemil ÖZ'e ve araştırmada yardım eden Arş.Gör. Musa BALTA'ya teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

APSP	: All-Pair Shortest Path
AUS	: Akıllı Ulaşım Sistemi
DMS	: Değişken Mesaj Sistemi
DUA	: Duarouter
GPL	: General Public License
GPS	: Global Positioning System
GUI	: Graphical User Interface
ITS	: Intelligent Transportation System
İÇN	: İlgi Çekici Nokta
OD	: Origin-Destination
OSM	: Open Street Map
SSSP	: Single-source shortest path
SUMO	: Simulation of Urban Mobility
TraCI	: Traffic Control Interface

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Akıllı ulaşım sistemi, Dijkstra algoritması, A Yıldız algoritması , SUMO, Rota hesaplama

Günümüzdeki insanlar için en önemli olan problemlerdin biri ulaşımdır. Son dönemlerde karayolu ulaşımında trafik sıkışıklığı, trafik kazaları artmaya başlamıştır. Taşımacılık veya ulaştırma sektörü, bir yerden bir yere götüren veya taşıyan bir yasal kaynaktır. Zaman geçtikçe, ulaşım, yüksek kaza oranı, trafik sıkışıklığı, trafik ve hava kirliliği, vb. gibi birçok konuyla karşı karşıya gelmekteyiz. Bu karmaşıklık nedeniyle, araştırmacılar sanal teknolojileri Akıllı Ulaşım Sistemi olarak bilinen ulaşımla bütünleştiriyorlar. Bu tez, Akıllı Ulaşım Sistemi uygulamalarının, teknolojilerinin ve farklı alanlarının çok çeşitli parçalarını ele almaktadır. Kara Taşıtlarında Akıllı Ulaşım Sistemlerinin (AUS) tanıtılmasının trafik güvenliğini ve hareketliliğini önemli ölçüde arttırdığı tahmin edilmektedir. Trafik sorunu büyük şehirlerdeki insanlar için en önemli sorunlardan biridir ve devam etmektedir. İşe, okula vb yerlere varış ve çıkış saatlerindeki yoğunluk nedeniyle trafikte çok fazla zaman harcanmaktadır. Son zamanlarda artan trafik sıkışıklığının şiddetli etkisinden dolayı, insanlar trafikte zaman kaybı ve maaliyet artışı yaşıyorlar. Gerçek zamanlı trafik akışları veya geçmiş verilere dayanarak araçların seyahat rotaların optimize etmek için rota hesaplama algoritmaları önerilmiştir. En kısa yolu bulmak için Dijkstra algoritması, A Yıldız algoritması, Genetik algoritma, Floyd algoritması ve Karınca kolonisi algoritması gibi birçok algoritma kullanılır. Bu çalışmanın amacı, simülasyon yoluyla araçların hedeflerine ulaşmaları için en iyi yönlendirme algoritmalarının hangileri olduğunu bulmaktır. Çalışmada, iki algoritma üzerinde durulacaktır. Bu nedenle, iki algoritma türünün performans analizlerini karşılaştırarak hangisinin az kenarlara (uç) giderek en hızlı ve en az maliyetli şekilde amaca ulaşmasını araştırmaktır. Çalışma görsel SUMO simulasyon aracını kullanarak gerçekleştirilmiştir. Sonuca göre A Yıldız algoritmasının arama süresinin Dijkstra algoritmasından daha hızlı olduğunu göstermiştir.

PERFORMANCE ANALYSIS OF THE SHORTEST PATH ALGORITHMS FOR INTELLIGENT TRANSPORTATION SYSTEM

SUMMARY

Keywords: Intelligent transportation system, Dijkstra algorithm, A star algorithm, SUMO, Route calculation

The transport or transportation sector is a legal resource that transports or carries it from place to place. Over time, we face many problems, such as transportation, high accident rates, road congestion, traffic and air pollutionetc. Because of this complexity, researchers integrate virtual technologies into transportation, known as the Intelligent Transport System.This thesis deals with the various parts of the Intelligent Transport System applications, technologies and different fields. It is assumed that the introduction of intelligent transport systems (ITS) in ground vehicles will significantly increase traffic safety and mobility. The traffic problem is one of the most important problems for people in big cities and is still going on. Too much time spent in traffic due to the intensity of arrival and exit times to work, school, etc. Due to the serious consequences of an increase in traffic jams in recent times, people are losing time in traffic jams and increasing costs. Algorithms for calculating routes are proposed to optimize vehicle routes based on real-time traffic flows or historical data. To find the shortest path, many algorithms are used, such as the Dijkstra algorithm, the A star algorithm, the Genetic algorithm, the Floyd algorithm, and the Ant colony algorithm. The purpose of this study is to find out which of the best routing algorithms are tools for achieving goals through simulation. The thesis will focus on two algorithms. Therefore, by comparing the performance analysis of the two types of algorithms, it is necessary to investigate which one reaches the least edges (ends) and reach the goal in the least cost way. It was performed using the SUMO visual simulation tool. The result shows that the A star algorithm faster than Dijkstra's algorithm.

BÖLÜM 1. GİRİŞ

Ulaşım, insanlığın tarihsel gelişiminde çok önemli bir rol oynamıştır. İnsanlar ürünlerini satmaya, doğru ürünleri satın almaya, yeni yerler keşfetmeye ve sonuç olarak topluluklar arasındaki ekonomik, sosyal ve kültürel ilişkilerin zamanla gelişmesiyle, bu yüzyılın başına insanlarin motorlu taşıtlarla girmesi ulaşım endüstrisindeki radikal değişikliklere yol açmıştır [1]. Kentsel bölge insanları, hem iş hem de eğlence için şehir içinde seyahat etme yeteneğine dayanan bir yaşam tarzı izlemektedir. Bu ihtiyaçlar nedeniyle şehir içi yol planlaması son on yılda büyük bir sorun haline gelmiştir. Özellikle şehirlerin denetimsiz gelişimi ile ilgili tarihsel nedenlerden ötürü, dar yollarda park yeri eksikliği gibi sorunlar ve bazı kritik yerler için alternatif rotaların bulunmaması, şehir yol ağı planlamasını çok karmaşık bir sorun haline getirmektedir [1]. Bu gerçeğe bağlı olarak, trafik sıkıntısı çeken vatandaşların günlük yaşamında en fazla etkiye neden olan bir takım sorunlar ortaya çıkmaktadır. Bu özel nokta, sadece yaşam tarzının kalitesinde bir düşüşe katkıda bulunmamaktadır. Aynı zamanda bu özel nokta başkaları arasında huzursuzluk, stres gibi mevcut sağlık sorunlarının ağırlaştırılmasına katkıda bulunan kısa yolları tamamlamak için yolda harcanan zamanla ölçülebilir. Günümüzde sosyo-ekonomik olaylar hızla büyüdüğünde, ulaşım hizmetlerine olan talep aynı zamanda artmaktadır. Sonuç olarak, ulaşım sistemlerinde özellikle yollarda büyük tıkanıklar zaman kaybının ve fazla yakıt tüketiminin yanı sıra çevre kirliliğinin ve aynı zamanda kazaların meydana getirmesiyle sonuçlanmaktadır. Dünyada, özellikle gelişmiş ülkelerde, bu sorunları çözmek için uzun yıllar çalışmalar yapılmıştır. Taşımacılık sistemlerinin daha verimli ve güvenli bir şekilde kullanılmasını sağlayacak önlemler, sorunları minimum düzeye indirmek için önemli bir adım olmuştur [1]. Tüm dünyada da kentleşme ve araç sayısındaki hızlı artış, yollardaki trafik kontrol sistemlerinin önemini artırmaktadır. Şehirlerarası yolların günümüzün ihtiyaçlarını karşılayacak şekilde düzenlenmesi ve çoğalması da önem taşımaktadır ve ulaşım çok önemli bir rol oynamaktadır. Toplumda

kentsel bölgelerdeki taşıt araçların günlük olarak trafik sıkışıklığı ve giderek artan sayıda araçlar yol problemini şiddetlendirmektedir. Literatür, bu tıkanıklık sorununa çözüm önermeleri için çabalayan çok sayıda çalışma içermesine rağmen, bugünlerde sorun hala yaygındır. Trafik tıkanıklığı sorunu sosyal, ekonomik ve çevresel yaşam toplumun kalitesini etkilemektedir. Trafik sıkışıklığı sorunun hafifletmek için Akıllı Ulaşım Sistemleri (AUS), dünyadaki ulaşım sistemlerini iyileştirmek ve kolayklaştırmak için ortaya çıkmıştır [2].

Ulaşımda karar vermeden önce, iyi kararlar alabilmek için referans olması gerekir. Kullanılabilecek referanslar arasında teknik teoriler, literatür çalışmaları, mesleki değerlendirme veya trafik simülasyonu yer almaktadır. Simülasyon önemli bir araçtır çünkü simülasyon neredeyse mükemmel bir görsel modele gerçek bir durumu gösterebilir ve ayrıca alternatiflerin etkisini görsel olarak da gösterebilir [3]. Simülasyon yardımıyla başlangıç düğümden hedef düğüme doğru veya iki nokta arasındaki en kısa yolu bulmak için pek çok algoritmalar simüle edilmektedir. En kısa yolu bulmakta veya amaca kadar gitmek için yol bulma algoritmaları önerilmiştir. Önerilen algoritmaları kullanarak araçların seyahat ettiği zamanları hesaplayabiliriz ve seyahat zamanını hesaplayarak en kısa yolu da bulabiliriz.

1.1. Amaç

Bu çalışmanın amacı, simülasyon yoluyla, araçlar hedeflerine ulaşmaları için en iyi yönlendirme algoritmalarının performansını karşılaştırarak hangisi daha verimli olduğunu bulmak ve ardından kuyruk uzunlunluğu, gecikme süresi, ortalama harcanan zaman gibi karşılaştırmaları değerlendirmektir.

1.2. Kapsam

Bu tez çalışması 5 bölümden oluşmaktadır. 2. bölümde Kaynak Araştırması anlatılmıştır. Ayrıntılı olarak kullandığımız programlar, simülatörlör, rota hesaplama alanındaki uygulamalar hakkında bilgi sunulmaktadır. 3. bölümde tezde kullanılan Yol

Bulma algoritmaları anlatılmıştır. Tezin 4.bölümde simülasyonun gerkekleştirme adımları anlatılmıştır. Son 5. bölüm tezin sonuç kısmını kapsamaktadır.

BÖLÜM 2. KAYNAK ARAŞTIRMASI

2.1. Akıllı Ulaşım Sistemi Tanımı

Ulaşım, ulaşıma dahil bütün etkenleri (yol, altyapı, araç, kullanıcı) analiz, control, düzenleme ve yönlendirmede elektronik ve bilişim teknolojilerinin kullanıldığı sistemdir [4]. Nüfus artışı ve diğer sebeplerden dolayı, günlük trafik dolaşımındaki taşıt ve insan sayısındaki artış ile yol ve kavşak düzenlemelerinin ve denetimlerin yetersizliği kaçınılmaz olarak trafik sıkışıklığına yol açmaktadır. Sonuç olarak, çevre kirliliği, enerji ve zaman kaybı, mutsuz ve tahriş olmuş insanların sayısı artmaktadır. Bu özellikle büyük şehirlerde hayatı hayal bile edilemez hale getirmeye başlamıştır. Bu nedenlerden dolayı hayatımızı kolaylaştırmak için Akıllı Ulaşım Sistemi (AUS) kavramı girmiştir [5]. AUS ulaşım sistemlerinin modellenmesinde yenilikçi gelişmeler ve son kullanıcılara daha fazla bilgi ve güvenlik sağlayan trafik akışlarının düzenlenmesinde ve geleneksel nakliye sistemlerine kıyasla hareket katılımcıları arasındaki etkileşimi niteliksel olarak artıran akıllı bir sistemdir. Akıllı Ulaşım Sistemi (AUS) olarak adlandırılan bu kavram, hayatımızı iyileştirmek ve taşımayı daha güvenli, daha verimli, konforlu ve ekosisteme saygı duymak için yenilikçi çözümler sunmaktadır [5].

2.1.1. AUS'un hayatımızdaki yeri

Teknolojik gelişmeler, bilgisayarların farklı alanlarda farklı amaçlar ile kullanılmaya başlaması gibi etkenler ile günlük ihtiyaçlarda tercih edilen cihazlar, yöntemler de değişkenlik göstermektedir. Teknolojik gelişmeler, haberleşmeden ulaşıma, sağlıktan eğlenceye kadar hayatın her alanında "akıllı" sistemlerin kullanılmasını sağlamaktadır. Bu sistemler, hayatı kolaylaştırdığı gibi, zaman tasarrufu, insan gücünden tasarruf gibi avantajlar da sağlamaktadır [2].

Endüstriyel alanda da kullanılan bu akıllı sistemler günlük hayat içerisinde ulaşımda sıklıkla kişilerin karşılarına çıkmaktadır. Ulaşımda tercih edilen teknolojik gelişmelerin arasına akıllı ulaşım sistemleri yer almaktadır. Toplum için her ne kadar oldukça yeni bir kavram olsa da birçok alanda faydalı bir sistem olarak kullanımı giderek yaygınlaştırılmaktadır. Trafikte kişilerin karşısına çıkan birçok unsur, teknolojik gelişmeler sayesinde hayatın içerisinde yer almaktadır. İşlemleri kolaylaştıran, zaman tasarrufu sağlayan, trafiğin tıkanmasını engelleyen akıllı ulaşım sistemleri tek yönlü bir sistem değildir. Bu sistem içerisinde 4 farklı kol incelenebilmektedir. Akıllı ulaşım sistemlerinin kolları, ulaşım koordinasyon merkezi, yolcu, taşıt, yol düzenlemeleri olarak incelenebilmektedir.

- Koordinasyon merkezi için akıllı ulaşım sistemleri: acil durum yönetimi, trafik yönetimi, ücretli geçiş idaresi, ticari taşıtlar idaresi, bakım ve yapı idaresi, emisyon yönetimi, ulaşım / aktarma yönetimi, yük taşımacılığı yönetimi, arşiv veri yönetimi, toplu taşıma yönetimi, enformasyon servis sağlayıcısı
- Yolcular için akıllı ulaşım sistemleri: kişisel bilgi erişimi, uzaktan yolcu destek sistemi
- Yol için akıllı ulaşım sistemleri: paralı geçişlerde ücret toplama, park yönetimi, ticari taşıt kontrolleri, güvenlik gözlemesi
- Taşıtlar için akıllı ulaşım sistemleri: ticari taşıtlar, transit taşıtlar, bakım ve yapı makineleri.

2.1.2. AUS'un sağladığı faydalar

Akıllı ulaşım sistemleri kişilerin seyahat özgürlüğünü kullanırken daha güvenli bir şekilde seyahat etmelerine yardımcı olmaktadır. Bunun yanı sıra trafikte birçok alanda olumlu etkiler gösteren, hem trafik kuralları bakımından hem seyahat eden kişiler tarafından artık trafiğin ayrılmaz bir parçası olarak kabul edilen sistemlerdir.

- Akıllı ulaşım sistemlerinin kullanılmaya başlanması ile birlikte gişe gibi geçiş noktalarında, uzun araç kuyruklarının oluşması engellenmektedir.

- Her an kamera ve benzeri sistemler ile güvenlik tedbirleri alındığı için akıllı ulaşım sistemleri sayesinde trafik kurallarının ihlal edilmesi, en aza indirilebilmektedir.
- Yolcuların daha güvenli bir şekilde yolculuk yapması sağlanmaktadır.
- Ekonomik açıdan çok bir katkısının olmadığı düşünülse de akıllı ulaşım sistemleri, enerji verimliliğini sağladığı için ülke ekonomisine de büyük ölçülerde katkıda bulunmaktadır.
- Daha akıcı bir trafik sağlandığı için akıllı ulaşım sistemleri ile seyahat sürelerinin daha kısaltılması da hedeflenmektedir.

Akıllı Ulaşım Sistemlerinin (AUS) amacı, seyahat güvenliğini, yolcu rahatlığını artırmak, trafik sıkışıklığını azaltmak ve yakıt tüketimini azaltmak için bilgi teknolojisi, iletişim, sensör teknolojisi ve interneti ulaştırma sistemlerine uygulamaktır. Akıllı trafik kontrol sistemi AUS'un önemli bir parçasıdır. Akıllı ulaşım sistemleri, sadece tek yönlü değil, görüldüğü gibi çok yönlü faydalar sağlamaktadır. Hem yolcu hem sürücü hem araç hem genel trafik için önemli bir ihtiyaç haline gelmektedir. Her geçen gün trafiğe çıkan araç sayısının artması ile yoğunlaşan trafikte, düzenin sağlanması için önemli bir unsur olarak görülmektedir [5].

AUS'de mevcut ulaştırma sistemlerini iyileştirmek için büyük bir potansiyel vardır. Bilimsel raporlarda, AUS kavramının 1970'lere geri döndüğünü gösterdiği gibi, ancak 1994'te Paris'teki ilk AUS kongresinden sonra birçok ülke sadece mevcut trafik kontrol sistemlerini geliştirmek ve iyileştirmek için akıllı ulaşım sistemlerini uygulamaya başlamıştır. Endüstriyel ülkelerde büyük ölçüde araç navigasyon sistemlerin ve trafik bilgisi hizmetlerinin kullanımını başlamıştır. Araç bilgi ve iletişim sisteminin başarılı bir şekilde uygulanması, Japonya'yı dünyadaki fiili operasyonda en gelişmiş akıllı ulaşım sistemlerine sahip tek ülke yapmıştır [6]. Ulaştırma problemlerini çözmek için akıllı ulaşım sistemlerini teşvik etme çabaları, AUS uygulanmasını büyük ölçüde hızlandırmıştır. Aynı zamanda, bilgi teknolojilerindeki son gelişmeler AUS'un dağıtımını da hızlandırmıştır. AUS'un farklı araştırma alanları, ulaşım sistemlerinin geleceğini garanti eder ve sonunda uygulamasıyla yeni bir endüstri sektörü yaratacaktır. AUS'den elde edilen faydalar ve maliyetler, AUS

altyapılarının tüm yaşam döngüsü ve bilginin uygulanmasından dolayı değişken kullanıcı davranışları boyunca çeşitli yönlerden incelenir [7].

2.2. Trafik Simülatörleri

Simülatör: Araştırma veya operatör eğitimi amacıyla özel koşulları veya makinenin özelliklerini veya gerçek bir işlemi simüle eden cihaz veya sistemdir.

İncelenecek üç trafik simülatörü vardır, bunlar:

1. Kentsel Hareketlilik Simülasyonu (SUMO) (Sumo 0.32.0) [8].
   SUMO açık kaynaklı bir mikroskopik yol trafiği simülasyon paketidir.
2. Treiber'in Karayolu Trafiğinin Mikrosimülasyonu (Trafik Akışının Mikrozimülasyonu: html 5 versiyonu) [9].
   Bu, trafik dinamikleri ve trafik modelleme çalışmalarının araştırılması amacıyla Triber tarafından oluşturulan açık kaynaklı bir trafik simülatörüdür.
3. Aimsun (Aimsum 8.3) [10].

Statik trafik atama araçları, mezoskopik ve mikrosimülatör olan Aimsun simülatöründe bulunan üç tip taşıma modeli vardır. Bu üç simülatör, yazılım kategorileri, Kullanıcı Belgeleri ve Grafiksel Kullanıcı Arayüzü (GUI), Ağ trafik jeneratörleri ve araç modelleri / türleri ve büyük trafik ağları simülasyon yeteneği ve grafiksel sunum kalitesi temelinde incelenecektir. Kullanılacak simülatörleri seçmeden önce, ticari kullanım ya da açık kaynaklı olduklarını bilmek gereklidir. Açık kaynak en çok memnuniyetle karşılanır çünkü kullanıcıların herhangi bir kısıtlama olmaksızın kullanımı, çalışması ve değiştirmesi için hak sağlar. Bahsedilen üç trafik simülatörü arasında, hem SUMO hem de Treiber'in Karayolu Trafiği Mikrosimülasyonu kullanımı ücretsizdir, Aimsun ise 30 günlük deneme süresiyle sağlanan ticari bir yazılımdır. SUMO'nun ücretsiz kullanımı ve açık kaynağı ve Treiber'in Karayolu Trafiği Mikrosimülasyonu nedeniyle, kaynak kodları ücretsiz olarak indirilebilir ve değiştirilebilir.

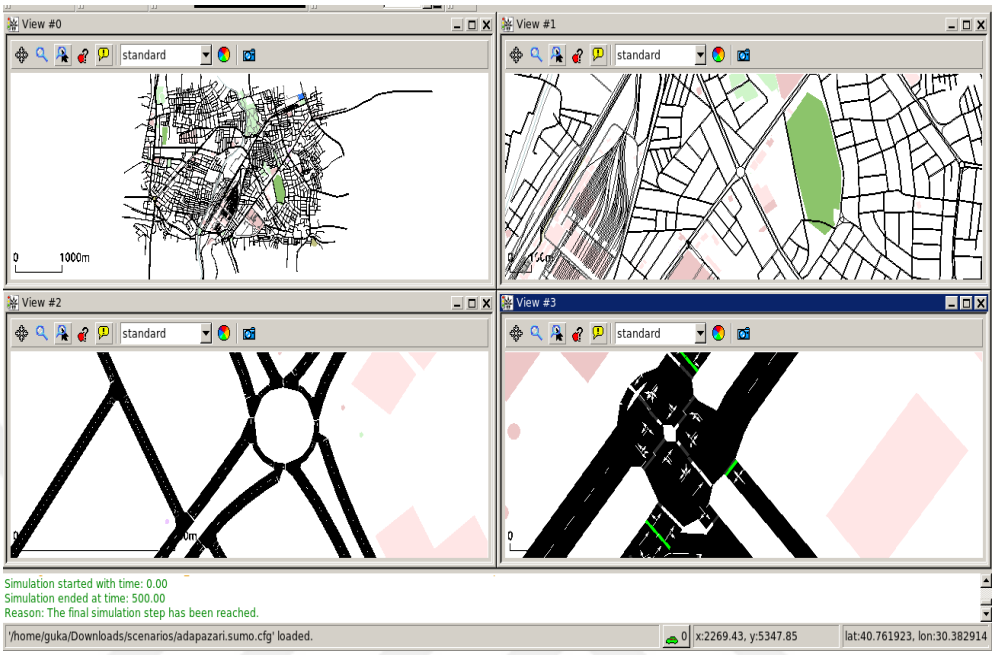
2.2.1. Kullanıcı belgeleri ve grafik kullanıcı arayüzü (GUI)

Trafik simülatörleri temelde kullanımı oldukça karmaşıktır ve ayrıca kullanıcı için kullanışlı olması için iyi bir kullanıcı arayüzü sağlaması önemlidir. Hem SUMO hem de Treiber'in Karayolu Trafiğinin Mikrosimülasyonu kullanıcı dokümantasyonlarına sahiptir. Bununla birlikte, Treiber'in Karayolu Trafiği kullanıcı belgelerinin mikrosimülasyonu, SUMO'nun kullanıcı belgelerine göre her bir özelliğe göre bölünerek ayrıştırılırken ayrıntılı değildir ve aynı zamanda daha iyi anlaşılabilmesi için örnekler verilmiştir. Aimsun G.Kotusevski ve K.A. Hawick (2009) tarafından yapılan araştırmaya göre, kullanım kılavuzu verilmemiş ve www.aimsun.com [10] adresindeki web sitesinden geçtikten sonra, eğitim yalnızca kullanıcı lisansı satın aldığında sağlanmıştır. Aimsun GUI açısından, sağlanan kullanıcı dokümantasyonu hem açık kaynak simülatörleri G.Kotusevski ve K.A. Hawick (2009)'a göre, nispeten anlaşılırdır [11]. Aimsun'daki trafik ağı, grafiksel ağ editörü kullanılarak elle çizilebilir. Araç tipleri için hem SUMO hem de Aimsun çeşitli araç tiplerini simüle edebilir, ancak Treiber'in Microsimulator cihazında, hem otomobilleri hem de kamyonları simüle edebilecek durumda değildir.

2.3. SUMO (Simulation of Urban Mobility)

Araçların yollarda nasıl hareket ettiğini simüle etmek için Kentsel Hareketlilik Simülasyonu (SUMO) simülatörü seçilmiştir. Bu simülatör, Alman Havacılık ve Uzay Merkezi'ndeki Ulaştırma Enstitüsü çalışanları tarafından geliştirilmiştir. Açık kaynaklıdır. Şekil 2.1.'de SUMO'nun görsel kullanıcı arayüzünü göstermektedir. SUMO, GPL kapsamında lisanslıdır [8]. SUMO, her bır araç seviyesinde çalıştığı mikroskopik bir simülatördür. Her araç simüle edilir. SUMO, ağları farklı kaynaklardan içe aktarma ve değiştirme, araç oluşturma ve bunlar için rota tanımlama konusunda birçok işlevsellik sunar. Ayrıca tanımlı ağlardaki en kısa yolu bulmak için kullanılabilecek iyi bilinen en kısa yol algoritmalarının bir uygulaması vardır. SUMO tek açık kaynaklı trafik simülatörü değil ama bu çalışmadada kullanmamızın sebebi ücretsiz ve farklı kaynaklardan haritaların alınmasına izin veriyor olmasıdır. Java

üzerinden simülasyon ile iletişim kurmak için uzak bir arayüze (TraCI) sahiptir ve Java algoritmalar ve deneyleri uygulamak için kullanılan bir dildir [12].



Şekil 2.1. SUMO'nun görsel kullanıcı arayüzü

SUMO, bir trafik simülasyonu hazırlamak ve gerçekleştirmek için gerekli tüm uygulamaları içerir (Ag ve rota içe aktarma, DUA, simülasyon).

SUMO'da yapılabilecek haraketler:

- Zaman aralıklı araç hareketi,
- Farklı araç tipleri,
- Şerit değiştirme ile çok şeritli sokaklar,
- Farklı yol kuralları, trafik ışıkları,
- Hızlı bir openGL grafik kullanıcı arayüzü,
- 10.000 gibi kenarlı ağları yönetir (sokaklar),
- Hızlı çalıştırma hızı (1 GHz'de 100.000 araç güncellemesi / s)
- Çalışma zamanında diğer uygulamalarla birlikte çalışabilirlik
- Ağ çapında, kenar tabanlı, araç tabanlı ve dedektör tabanlı çıktılar

Ağ ithalatı diğer araçlardan da mümkündür:

- Import VISUM, Vissim [13], Shapefiles, OSM, RoboCup, MATsim, open - DRIVE ve XML-Açıklamaları
- Eksik değerler sezgisel yoluyla belirlenir

Yönlendirme olanakları:

- Mikroskopik rotalar - her arac bir tanedir
- Farklı Dinamik Kullanıcı Atama algoritmaları

Yüksek taşınabilirlik:

- Sadece standart c ++ ve taşınabilir kütüphaneler kullanılır
- MS Windows ve ana Linux dağıtımları için paketler var

Yalnızca XML verilerinin kullanımıyla büyük çalışmalar yapılır.

- Açık kaynak (GPL)

Aşağıdaki operasyon sistemleri için kullanılabilir:

- MS Windows
- Linux
- Mac os işletim sistemi

2.3.1. SUMO'nun kullanıldığı alanlar

SUMO'yu pratikte anlamak için, SUMO'nun nasıl kullanılacağına dair literature bazı kullanım durumları hakkında genel bir bakış sunar. [14]'de dinamik bir rota planlama çerçevesi önerilmektedir. Rota algoritması, yol ağları gibi dinamik değişen ortamlarla etkileşime girebilir ve bir araca atanan rotayı trafik yönetim sistemlerinden alınan yeni güncellemelere göre uyarlar. En iyi rotayı güncelleme işlemi zaman zaman çalışır.

Ayrıca Dijkstra Algoritması için SUMO kullanılarak bir ağ tanımlamış, test edilmiş ve algoritmanın yol ağına erişmesini sağlayacak şekilde yollar oluşturmak için uygulanmıştır. Karayolu ağı üzerinde seyahat eden araçları simüle etmek için bir çerçeve (temel model) geliştirilmiştir. Çerçeve, simülasyonlar sırasında ve ilk olarak rotaları olan araçları tanımlamaya ve trafik yönetim sistemlerinden gelen güncellemelere göre algoritma bir araca atanan en iyi rotaları uyarlar. Çalışma, trafik simülasyonuna erişim sağlayan TraCI arayüzünü kullanıyor ve benzetilmiş araçların değerlerini çıkarmamıza ve rotalarını "çevrimiçi" yönetmemize olanak tanıyor [16]. Platform, yollardaki gerçek arabaların bu simülasyonun bir parçası olabileceği büyük ölçekli trafik simülasyonları oluşturmaya izin veriyor. Bu platformda, veriler aracın ağ geçidinden toplanır ve bir akıllı telefon kullanılarak sunucuya gönderilir. Yoldaki araçlardan toplanan bilgiler sunucuda işlenir ve davranış önerileri araçlara geri gönderilir (hızlar ve yeniden yönlendirmeler gibi).

2.3.1.1. SUMO'da simülasyon sonuçları

SUMO trafik simülasyonu senaryosu, rotalarını, seyahatlerini içeren tüm araçları ve ayrı bir dosyada tanımlanan yol ağını içeren bir dosya kullanılarak gerçekleştirilir. Simülasyon sırasında, araç başlangıç noktasından varış noktasına doğru yola çıkar, yolculuğu tamamlamak için gereken süre yoldaki trafik, trafik kazası, yoldaki ışıklar vb. gibi birçok faktörden etkilenir. Simülasyonun sonunda SUMO varış zamanı, varış konumu, yolculuk süresi, bekleme adımları, zaman kaybı (trafik sıkışıklığı, sinyallerin üzerinde durması nedeniyle), vb. bilgileri içeren bir çıktı dosyası oluşturur [15].

2.3.1.2. SUMO yol ağı tanımı

Bir SUMO ağ dosyası, bir haritanın trafikle ilgili bölümünü, araçların geçtiği yolları ve kavşakları tanımlar. SUMO'da bir yol ağı yönlendirilmiş bir grafik olarak temsil edilir, genellikle "düğümler" ve "kavşaklar" olarak adlamdırılıyor.

2.3.1.3. Kenarlar ve şeritler

Normal bir kenar, iki düğüm "kavşağı" arasındaki bağlantıdır ve her kenar, oluşturduğu şerit tanımlarını içerir.

2.3.1.4. Trafik ışığı programı

NETCONVERT [17], ağların hesaplanması sırasındaki trafik ışıkları ve kavşaklar için programlar üretir. Bu hesaplanan programlar gerçek programlardan farklıdır, ancak simülasyona gerçek trafik ışığı programları ile sağlanmasına izin verilmektedir.

2.3.1.5. Araç etkileşimi

Simüle edilen araçlar arasındaki iletişim, bir araba takip eden model ve bir şerit değiştirme modeli kullanılarak tanımlanır [18].

- Araba takip eden model: bir aracın hızını öndeki araca olan mesafeye göre nasıl uyarlayacağını belirler.
- Şerit değiştirme modeli: bir aracın şeritleri ne zaman değiştirmesi gerektiğini tanımlar, bu, aracın mevcut şeridinin bir sonraki bağlantıya sahip olmaması durumunda gerçekleşir.
- Rotanın kenarı: SUMO, herhangi bir aracın önündeki araçlarla çarpışmayı önleyebileceği anlamına gelir, çarpışmadan uzak olacak şekilde tasarlanmıştır.

2.3.1.6. Araç yapılandırma parametreleri

SUMO'da tanımlanan araçların birçok yapılandırma parametresi vardır; Aşağıda, önemli parametrelere genel kısa bir bakış yer almaktadır:

- Id: Aracın adı
- Route: atanmış rotanın kimliği. Rotalar ayrı bir dosyada tanımlanır. Her bir rota bağlı bir dizi kenardır.

- Depart: aracın ağa girmesi gereken zaman adımı
- DepartLane: aracın ağa girmesi gereken şerit
- DepartSpeed: Aracın şebekeye girme hızı
- ArrivalSpeed: Aracın şebekeden ayrılması gereken hız
- Type: Tip hem aracın fiziksel parametrelerini (uzunluk, renk gibi) hem de kullanılmış araba takip modelinin parametrelerini tanımlar.

2.4. Trafik Simülasyonu

Genel olarak simülasyon, bilgisayar modelinin oluşturulması ve zaman içinde hareket ettirilmesiyle elde edilen gerçek dünyanın bir kısmının dinamik temsili olarak tanımlanmaktadır [18]. Bilgisayar simülasyonu kullanımı, D.L.Gerlough, 1955 yılında California, Los Angeles Üniversitesi'nde "Genel amaçlı ayrık değişken bir bilgisayarda otoyol trafiğinin simülasyonu" başlıklı tezini yayınladığı zaman başladı [19]. Trafik araştırmasında, trafik modellerinin dört sınıfı vardır, ancak genellikle insanlar bunu üçe böler, örneğin (Şekil 2.3.).

2.4.1. Makroskopik simülasyon modeli

Makroskobik modeller 1950 yıllarında geliştirilmiştir. Makroskopik simülasyon modeli, trafik akışının, hızının ve yoğunluğunun deterministik ilişkisine dayanmaktadır. Makroskopik bir modeldeki simülasyon, bireysel araçları takip etmek yerine, bölüm bazında gerçekleştirilir. Aşağıdaki (Şekil 2.2.) şekilde katlı bir kavşak görülmektedir. Makroskopik modeller sisteme bu katlı kavşağa giren veya çıkan araçları, ortalama araç hızlarını ve toplam araç sayısını inceler [19].

Şekil 2.2. Katlı kavşak

2.4.2. Mezoskopik simülasyon modeli

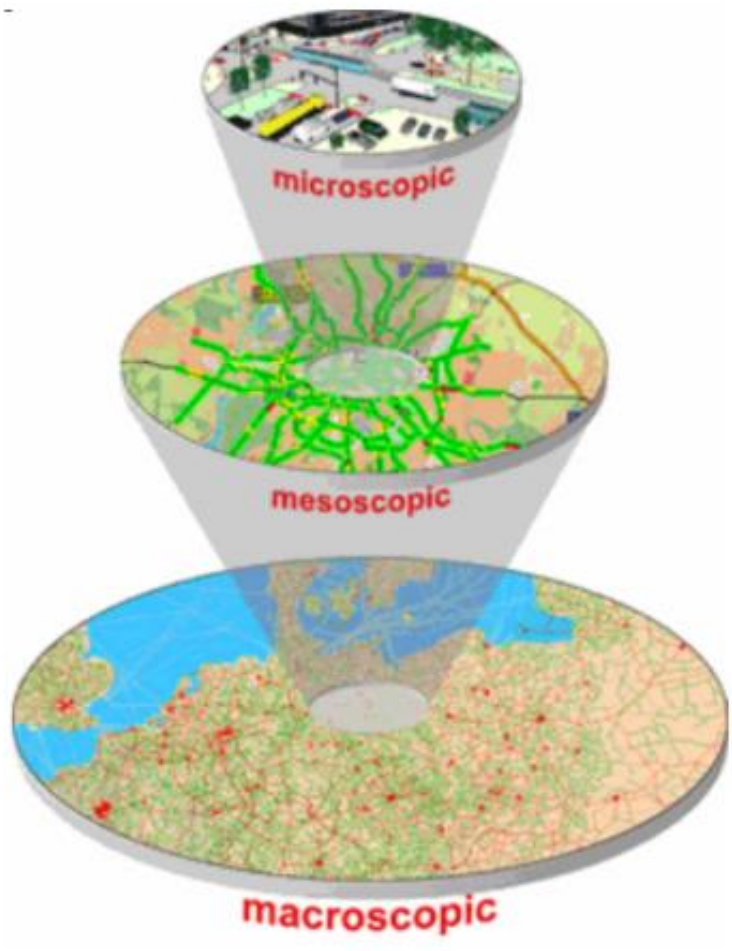
Mezoskopik simülasyon modelleri hem mikroskobik hem de makroskobik simülasyon modellerinin özelliklerini birleştirir. Bu tür simülasyonlardaki trafik akış birimi bireysel taşıttır ancak hareket makroskopik modun yaklaşımını kullanıyor. Mezoskopik simülasyon toplam düzeyde gerçekleşir ve dinamik hız / hacim ilişkisini dikkate almaz [19].

2.4.3. Mikroskobik simülasyon modeli

Mikroskobik modeller daha karmaşık modeller olarak kabul edilir, çünkü her aracın hareketini ayrı ayrı incelerler. Daha ayrıntılı bir model olarak, mikroskopik model, grafik mühendisleri için büyük önem taşımaktadır. Bir aracın hareketini değerlendirirken, farklı sürücü davranışınlarındam birçok değişkenin dikkate alınması gerekir. Bu değişkenler arasında en önemli parametre birbirini takip eden araçlar arasındaki aralıktır. Araç arasındaki zaman aralığı ikiye ayrılır.

1. Mesafe aralığı
2. Zaman aralığı

Tipik olarak, araçlar varışların istatistiksel bir dağılımını kullanarak bir ulaşım ağına girerler [19]. Mikroskobik model, araçların birbirleriyle ve altyapı ile nasıl hareket ettiklerini açıklamak için hızlandırma, hız yarlaması, şerit değiştirme, boşluk kabulü vb. için alt modeller içermektedir [20].



Şekil 2.3. Trafik modelinin seviyeleri

2.4.4. Tek seferlik yönlendirme yöntemleri

Başlangıçta, tüm sürücülerin planladıkları bir yolu vardır. Karayolu ağı için güncel trafik bilgisi olmadan, bir araç başlangıçta yönlendirildiği yolun üzerinde kalacaktır. Güncellenmiş trafik bilgisi almayan araçlar "tek seferde yönlendiren" araçlar olarak kabul edilir. Onların rotası, ilk rotalama sırasındaki bilinen bilgilere dayanır. Rotaları değerlendirmek için iki genel kriter vardır:

- Hangi rotanın en kısa mesafe olduğu
- Hangi rotanın en kısa süreyi alacağı

2.4.4.1. En kısa mesafe

Simülasyonlarımızda, en basit yönlendirme yöntemi rotaları yargılama kriterleri olarak en kısa mesafeyi kullanır. Yöntemin amacı, aracın başlangıç konumundan en az mesafe kaplayan hedefe giden yolu bulmaktır. Bu basit bir yöntemdir. Bu ağdaki yolların hepsi aynı hız sınırına sahip olduğu için, daha sonra tartışılacak olan en kısa bir zaman yönlendirme yöntemine eşdeğerdir. Bu tezdeki en kısa mesafe yönlendirmede SUMO aracı DUAROUTER [21] kullanılarak yönlendirilir. Aslında, araç rotaları başlangıçta SUMO'ya yerleştirildiğinde, en kısa mesafe yöntemi tüm araçlarda, uygun algoritmaların yol ağında iken onları yeniden yönlendirmesi beklentisiyle kullanılır.

2.4.4.2. En kısa zaman

Seyahat süresi, sürücülerin rotalarını seçmek için kullandıkları normal kriterlerdir. Bir rotanın seyahat süresini yargılamanın temel yolu, bir rotanın her bir bölümünün hız sınırlarını kullanmaktır. Bu, hızlandırılmış araçların simülasyonlarımızda çalışmasıdır. Ek olarak, simülasyonlarımızdaki "akıllı" yöntemlerin tümü, seyahat sürelerini rotaları yargılama ölçütü olarak kullanır ve her yol segmentindeki seyahat süresine yaklaşma yöntemiyle yeniden yönlendirilir. Gönderilen hız araçları bir seferlik araçlardır.

2.5. Rota Hesaplama

Artık mobil iletişim büyümekte, araçların sürüş koşullarıyla ilgili ek bilgileri paylaşma kabiliyeti genişletilmekte. Güncel verilerle canlı trafik haritasına bakmamız yararlıdır, ancak rota bilgi paylaşımı ileride bunları yansıtarak, bir adım daha ileri trafik haritaları geçirmeyi ummakta. [22]'da önerilen yol bilgisi paylaşım metodu, bir sürücünün diğer araçların planladığı rotaları bilmesi halinde, şu anda sıkışık olan yollardan ve araçlarından dolayı tıkanıklıklara yol açacak yollardan kaçınabileceği varsayımına dayanmaktadır. Bunun umut verici göründüğü bir senaryo, büyük bir otoyolda bir kaza olduğunda ortaya çıkar. Otoyolda çok sayıda araç var, arkalarında daha hızlı bir şekilde sıraya diziliyor. Tüm araçlar, kazadan önceki son çıkıştan çıkarsa, bir dar boğaza dönüşecek ve bazı şehir sokakları, daha fazla araç girmeye başladıkça hızla tıkanacaktır. Teorik olarak, rota bilgisi paylaşımı araçların veya bir trafik kontrol merkezinin yeni rotaları hesaplamasına ve farklı rotaları kooperatif olarak seçmesine izin verecek, böylece ilave araç sayısı alternatif yollarda dağıtılacaktır. Önerilen yöntem, rotaların karşılaştırılmasında kullanılacak araçlar için bir "İleriye Yönelik Trafik Hacmi" haritası yaratır. Bu harita, beklenen tıkanıklığı hesaba katmak için atfedilen bir değeri olan tüm yolların listesidir. Potansiyel Trafik Hacmi, mevcut uygun seyahat süresinin bir ürünüdür (Greenshield [23] modeline göre) ve Toplam Geçit Ağırlığı olarak adlandırılan bir değerdir. Toplam Geçit Ağırlığı, her bir aracın o yol segmentindeki bireysel Geçit Ağırlığının toplamıdır. Bir araç, rotadaki yol segmentlerinin sayısını belirtilen yol segmentine kadar sayarak ve yolun kalan rotasındaki toplam yol segmentine bölerek rotadaki her yol segmenti için Geçit ağırlığı hesaplar.

BÖLÜM 3. YOL BULMA ALGORİTMALARI

3.1. En Kısa Yol Bulma Algoritmaları

Günümüzde, mevcut en kısa yol problemlerini çözmek için birçok algoritma bulunmaktadır. Birçoğu öncekilerden gelişmiştir. Her biri sorunu farklı parametrelerle çözer. Aşağıdaki liste, grafik teorisi biliminin temelini oluşturan en kısa yol problemini çözen temel algoritmaları içermektedir [40]:

- Dijkstra algoritması
- A Yıldız algoritması
- Bellman-Ford algoritması
- Floyd-Warshall algoritması
- Genetik algoritma
- Floyd algoritması
- Karınca kolonisi algoritması

Dijkstra algoritması [24] ve Bellman-Ford algoritması [26, 27] tek kaynaklı en kısa yol (SSSP) problemini çözmektedir. SSSP problemi şu şekilde tanımlanabilir [25]: Her kenarda negatif olmayan maliyetleri olan ve seçilen kaynak düğümü $v \in V$ olan bir $G = (V, E)$ grafiği verilen v ile $\forall w \in V$ arasındaki en düşük maliyetli yolun maliyetini bulun. . Bir yolun maliyeti, basitçe yolun geçtiği kenarlardaki maliyetlerin toplamıdır. Dijkstra algoritması, negatif kenarların bulunmadığı bir grafikle çalışan açgözlü bir algoritmadır [24]. Bellman-Ford algoritması, Dijkstra algoritmasının açgözlü bir sürümü değil, negatif kenarlı grafiklerle çalışmanıza izin veriyor [26, 27]. Floyd-Warshall algoritması [28, 29] ve Johnson'ın algoritması [30] en kısa yol (APSP) problemini çözer. Johnson'ın algoritması, öncelikle tüm negatif kenarları pozitif olanlara dönüştürür ve ardından, Dijsktra algoritmasını grafikteki her düğümde

uygular. Seyrek grafikler için Johnson'ın algoritması, Floyd-Warshall algoritmasından [31] daha hızlı hesaplama süresi sağlar.

3.2. Dijkstra Algoritması

Algoritma 1956'da Edsger Wybe Dijkstra tarafından tasarlanmış ve 1959'da resmi yayınlanmıştır. Dijkstra'nın orijinal fikri iki düğüm arasındaki en kısa yolu bulmaktı [32]. Bununla birlikte zamanla, bilgisayar bilimcileri arasında Dijkstra algoritması, grafikteki tek bir düğümden diğer tüm düğümlere giden bir yol bulan bir algoritma olarak kabul edilmiştir [34]. Dijkstra algoritması uygulanan problemi çözmek için inşa edilmiştir: Bir noktadan diğerine mümkün olan en kısa yolu kullanarak ulaşılır [32]. En kısa yolu tanımlamak için kullanılan ana ölçüt ya zaman ya da mesafedir. Her iki miktar da sadece pozitif değerlere sahiptir. Bu kriterler grafik teorisine dönüştürülürse, grafiğin tüm kenarlarının da pozitif olması gerekir. Kenarlar negatifse, gerçek en kısa yol elde edilemez. Mevcut veriler, sadece grafikteki düğümler olan uygun değerleri içerecektir. Ayrıca grafik coğrafi haritayı temsil edecektir. Bu nedenle, Dijkstra algoritması, SSSP'yi çözmek için tercih edilen algoritma olacaktır [33]. Dijkstra algoritnması Link-State (Bağlantı-Duıumu) maıntığına dayanır. Link-State uzaklık , kapasite veya bant genişliği, yayılma gibi temel ölçüler verilebilir.

Dijkstra algoritmasının birçok çeşidi vardır [34]. Burada düğümlerin üç durumda bulunabileceği sunulan bir varyasyon var: taze, açık ve kapalı (Algoritma 1.). Her düğüm v için fonksiyon dist (v), başlangıç düğümünden s olan mesafeyi temsil eder. Ulaşılamaz düğümler için bu işlev tarafından döndürülen değer tanımsız olacaktır. Dist işlevinin yanında, düğümü geri s düğüme en kısa yoldan döndüren previous (v) işlevi de vardır. Dijkstra algoritmasını kullanarak, bir başlangıç düğümü ile grafikteki diğer herhangi bir düğüm arasındaki en kısa mesafeyi (veya en az çaba / en düşük maliyeti) belirlemek mümkündür. Algoritma fikri, başlangıç noktasından başlayarak en kısa mesafeyi sürekli olarak hesaplamak ve güncelleme yaparken daha uzun mesafeleri hariç tutmaktır. Algoritma aşağıdaki adımlardan oluşur:

1. "Sonsuz" mesafeli tüm düğümlerin başlatılması; 0 ile başlangıç düğümünün başlatılması
2. İlk düğümün mesafesini sabit, diğer tüm mesafeleri geçici olarak işaretlemek.
3. Başlangıç düğümünün aktif olarak ayarlanması.
4. Aktif düğümün tüm komşu düğümlerinin geçici mesafelerinin, kenarlarının ağırlıkları ile mesafesini toplayarak hesaplanması.Bu, mesafenin ayarlanma sayısıdır. Bu adıma ayrıca merkezi fikir denir.
5. Tahmini düğüm mesafesi mevcut olandan daha küçükse, mesafeyi güncelleyin ve geçerli düğümü öncekine ayarlayın. Bu adıma güncelleme adı verilir
6. Düğümün aktif olarak asgari geçici mesafe ile ayarlanma. Mesafesini kalıcı olarak işaretleyin.
7. Komşular hala geçici mesafelere sahip olan herhangi bir mesafe kalmayana kadar 4. ile 7. adımları tekrarlayın.

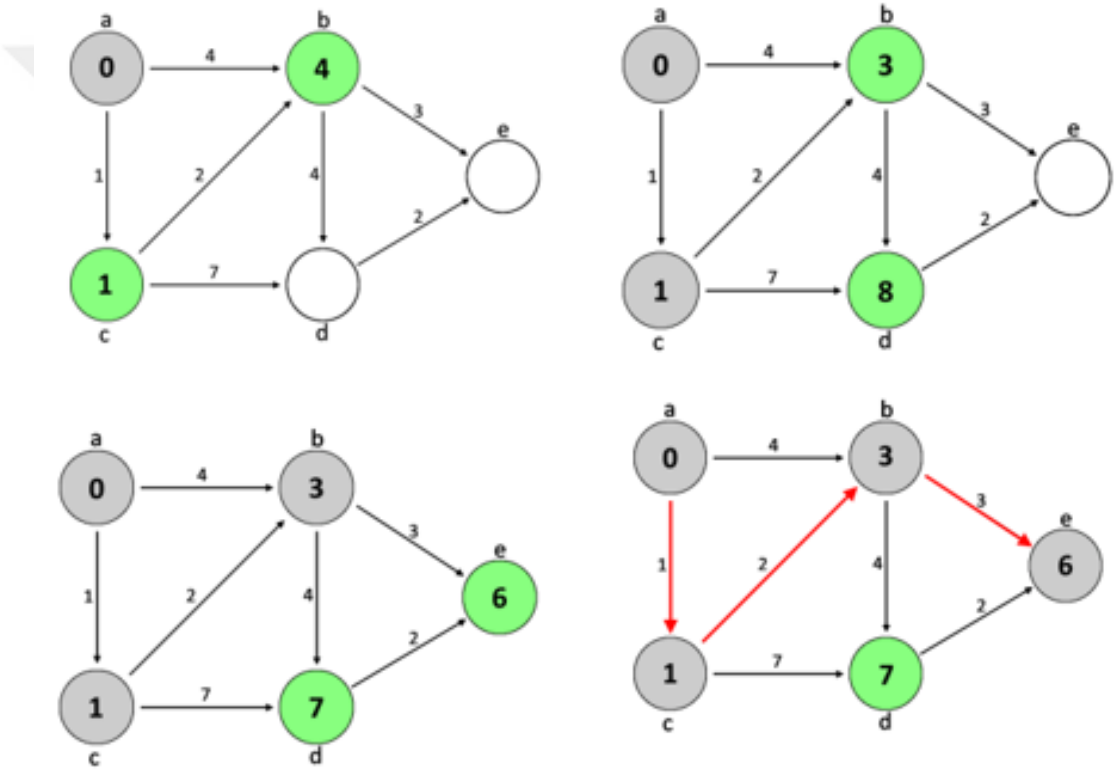
```
1   function Dijkstra(Graph, source):
2:  for each vertex v in Graph:                 // Başla
                                                // kaynak ile tepe noktası arasındaki
3:  dist[v] := infinity                         başlangıç mesafesi v sonsuzluğa eşit olarak
                                                ayarlanır
                                                // Kaynaktan optimum yoldan önceki
4:  previous[v] := undefined                    düğüm
5:  dist[s] := 0                                // Kaynaktan kaynağa olan mesafe
                                                // grafikteki tüm düğümler optimize
6:  Q := the set of all nodes in Graph          edilmemiş - bu nedenle Q'da
7:  while Q is not empty:                       // ana döngü
8:  u := node in Q with smallest dist[ ]
9:  remove u from Q
10: for each neighbor v of u:                   // v henüz Q'dan kaldırılmadı.
11: alt := dist[u] + dist_between(u, v)
12: if alt < dist[v]
13: dist[v] := alt
14: previous[v] := u
15: return previous[ ]
```

Algorithm 1. Dijkstra algoritması

Dijkstra algoritması Bellman-Ford algoritmasına çok benzer. Bu algoritmalar arasındaki temel fark, düğümlerin tekrar tekrar kontrol edilmesidir. Dijkstra algoritması bir düğümü kapattığında, bir daha asla kontrol edilmeyecektir. Bellman-

Fold algoritması her bir düğümden geçer ve grafikte negatif kenarların mevcut olması durumunda yolu yeniden hesaplar. Bu fazladan adım nedeniyle Dijkstra'dan daha yavaştır, ancak grafiğin geçerli olup olmadığını tespit edebilir.

Algoritmanın temel dezavantajı, kör bir arama yapması ve gerekli kaynaklar üzerinde çok fazla zaman harcamasıdır. Diğer bir dezavantaj, negatif kenarları işleyememesidir. Bu, asiklik grafiklere yol açar ve çoğu zaman doğru en kısa yolu bulamaz. Dijkstra algoritmasında çıkan sonuç yolun grafını verir. Örnek olarak Şekil 3.1.'e bakabiliriz.



Şekil 3.1. Dijkstra algoritmasının uygulanması

3.2.1. Dijkstra algoritmasının kullanıldığı alanlar

- Google Haritalar'da kullanılır
- En kısa yolu bulmakta kullanılır
- Coğrafi Haritalarda kullanılır

- Grafiğin köşelerini ifade eden konumlarını bulmata kullanilir
- Konum arasındaki mesafe, kenarları ifade eder
- IP yönlendirmesinde İlk önce en kısa yolun açılması için kullanılır
- Telefon şebekesinde kullanılır.

3.3. A Yıldız Algoritması

Algoritma ilk olarak 1968'de Stanford Araştırma Enstitüsünde Peter Hart, Niels Nilsson ve Bertram Rafael tarafından tanımlanmıştır. Bu, uzayda çok sayıda sorunla ilgili açıklamaları bulmak için kullanılabilecek evrensel bir arama algoritmasıdır. Stratejik oyunlar ve en popüler en kısa yol bulma algoritmasıdır. A yıldız algoritması, bir yol bulmak için en iyi seçimdir, çünkü oldukça esnektir ve çok çeşitli bağlamlarda kullanılabilir [35]. Bu algoritma Dijkstra algoritmasının değiştirilmiş bir versiyonudur. Kullanım alanları çok yaygındır. Fakat bu forumda bizi ilgilendiren elbette oyun geliştirme eğilimli kullanımıdır. Grid düzleme sahip oyunlarda (2D RPG, 2D/3D RTS benzeri) kullanımı yaygındır. Ağırlıklı olarak "nesnelerin hedefledikleri noktaya olan yollarını bulması" amacıyla kullanılmaktadır. Yol bulma için iki yaklaşım vardır.

- Matematiksel yaklaşım
- Sezgisel yaklaşım

Matematiksel yaklaşım, hedeflere ulaşmak için düğümlerin düzenli olarak test edilmesini içerir. Sezgisel yaklaşım ise, daha verimli olan en kısa yol hakkında bilgi toplamak için problemin türü hakkında özel bilgiler kullanır [37]. A Yıldız algoritması iki yaklaşımı birleştirir, problem hakkında bilgi kullanır ve daha sonra bir yol bulmak için matematiksel bir çözüm belirler [36].

A Yıldız algoritması Dijkstra algoritmasının bir uzantısıdır. A Yıldız algoritması sezgisel yöntemlerle daha iyi performans (zamana bağlı olarak) sağlar. A Yıldız en kısa yolu bulmak için kullanılan en iyi ilk arama tekniğini takip eder, f (n) = g (n) + h (n). Bu değer sezgisel tahmin değerine eklenerek sezgisel fonksiyon hesaplanır. A Yıldız algoritması yol boyunca alternatif yol bölümlerinin bir öncelik sırasını koruyarak,

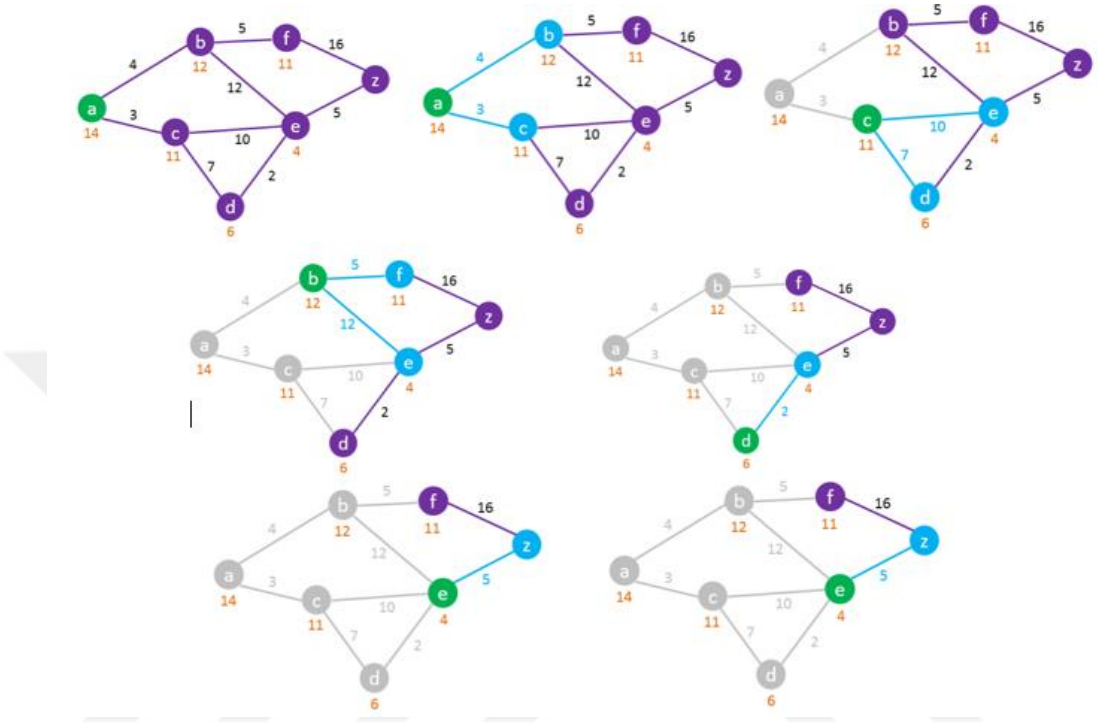
bilinen en düşük yolun bir yolunu izler. Herhangi bir noktada, geçiş yapılan yolun bir segmenti, karşılaşılan başka bir yol segmentinden daha yüksek bir maliyete sahipse, daha yüksek maliyetli yol segmentini terk eder ve bunun yerine daha düşük maliyetli yol segmentine geçer. Bu süreç hedefe ulaşılana kadar devam eder. En kısa yolu hesaplamak için sezgisel yol bulgusunu kullanırız. Hesaplama bir denge durumuna dönüşene kadar iterativ olarak yapılir [39].

Yol bulmada sezgisel kullanmazsak ve h'yi her zaman sıfır olarak değerlendirirsek, bu algoritmaya Dijkstra algoritması da denir [39]. A Yıldız yönlendirme algoritması, aramayı yönlendirmek, seyahat süresini sınırlamak için bir ölçüm kullanır ve genellikle Dijkstra'dan daha hızlıdır. Burada, metrik öklid mesafesi kullanılır.
A Yıldız algoritmasının pseudo kodu:

```
S ← Ø
Q ← s
h ← Ø
d[s] ← 0
while Q ≠ Ø do
 u ← EXTRACT-MIN(Q)
 If (u is the target)
 stop;
S ← S∪{u}
for each vertex v∈Adj[u] do
if v ∈ Q then
if d[v] > d[u] + w(u, v) then
d[v] ← d[u] + w(u, v)
h[v] ← h(v)
π [v] ← u
else
d[v] ← d[u] + w(u, v)
h[v] ← h(v)
π [v] ← u
Q ← v
```

Algoritma 2.

A Yıldız algoritması için basit bir örnek (Şekil 3.2.) ve aldığımız sonuç (Tablo 3.1.)



Şekil 3.2. A Yıldız algoritmasinin grafta uygulanması

Tablo 3.1. A Yıldız algoritmasının hesaplama sonucu

A'dan Kısa yol	Z'ye kadar Sezgisel mesafe	Toplam mesafe	Önceki düğüm
A-0	14	14	
B-4	12	4+12=16	A
C-3	11	3+11=14	A
A-C-D=3+7=10	6	10+6=16	C
A-C-D-E =3+7+2=12	4	12+4=16	D
A-B-F=4+5=9	11	9+11=20	B
A+C+D+E+Z= 3+7+2+5=17	0	17	E
Kısa yok-- A - C - D - E - Z uzunluğu 17			

3.4. Rotalamadaki Sezgisel Arama

En kısa yolu hesaplamak için sezgisel yol bulma özelliğini kullanılıyor. Hesaplama denge durumuna yaklaşıncaya kadar tekrarlı bir şekilde yapılıyor. Yolu hesaplamak için aşağıdaki denklem anahtar denklemdir:

$$f(n) = g(n) + h(n)$$

f (n) = n durumundan hedef duruma kadar hesaplanmış sezgisel fonksiyon.
h (n) = Sezgisel tahmin n durumundan hedef duruma kadar.
g (n) = n durumundan hedefe kadar yol maliyeti.

Aramaya başladığımızda, n'ye bitişik olan düğümleri kontrol ederiz ve hedefe ulaşana kadar genişleriz. n'den sonra açık bir liste yaratırız, başlangıçta listede n'in kendisi olan bir düğüm vardır sonra daha fazla düğüm eklenecek, bazı düğümler aradığımız yolda olacaktır, bazıları eklenmeyecek ve daha sonra kontrol edilecektir. n etrafındaki düğümler ilk önce kontrol edilecek ve ulaşılabilir olanlar açık listeye eklenecek ve n onların ana düğümü olacaktır. Daha sonra n açık listeden kapanış listesine çıkarılır. Bir sonraki adım açık listeden bir düğüm seçmemiz gerekir, en küçük f (n) değerine sahip olan seçilecek ve sonra hedefe ulaşılana kadar aynı süreçten geçecektir. Bununla birlikte, açık listedeki sırayı sıralamak için sezgisel yöntemi kullanımı bunun bize daima en kısa yolu vereceğini garanti edemeyiz [37].

- h (n) değeri 0 ise, A Yıldız algoritması Dijkstra algoritmasına dönüştürülür, burada en kısa yolun bulunması garanti edilir.
- h (n) değeri hedefe ulaşma maliyetinden düşükse, en kısa yol garanti edilir, ancak işlem yavaşlar.
- h (n) bir değere eşit ise, en kısa olanı çok hızlı bulunur, ancak böyle ideal bir sezgisel buluşu belirlemek kolay değildir.
- Eğer h (n) kaynaktan hedefe giden maliyetten büyükse, en kısa yol garanti edilmez, ancak işlem daha hızlı çalışır.

Normalde, hareket eden kurallara bağlı olarak uygulanacak üç sezgisel yöntem (bkz. Şekil 3.3.) vardır [38]:

3.4.1. Manhattan uzaklığı

h (n) = Cost * (abs (n.x-goal.x) + abs(n.y-goal.y))
h (n)- Sezgisel Maliyet Fonsiyonu
n.x – Mevcut düğüm x koordinatı
goal.x – Hedef düğüm x koordinatı
n.y – Mevcut düğüm y koordinatı
goal.y – Hedef düğüm y koordinatı

Manhattan uzaklığı Manhattan'ın New York'taki sokak coğrafyasına dayanmaktadır ve 4 hareket yönüne izin verdiğinde kullanıyoruz. Manhattant uzaklığın sezgisel maliyet hesaplamalarda kullanılıyor.

3.4.2. Öklid uzaklığı

Öklid uzalkığı sezgisel maliyet hesaplamada en çok kullanılan yöntemdir. Mevcut düğümden hedef düğüme olan dikey ve yatay uzaklıkları karelerinin toplamının kare kökü alınarak hesaplanır. Öklid uzaklığında maliyet Manhattan uzaklığına göre daha fazladır.

h (h (n) = sqrt(abs (cur.node.x – dest.x) + abs (cur.node.y – dest.y))
h(n)- Sezgisel Maliyet Fonsiyonu
cur.node.x – Mevcut düğüm x koordinatı
dest.x – Hedef düğüm x koordinatı
cur.node.y – Mevcut düğüm y koordinatı
dest.y – Hedef düğüm y koordinatı

Herhangi bir hareket yönünün izin verdiği ölçüde kullanırız.

3.4.3. Çapraz uzaklık (Dioganal)

Diagonal uzaklık sezgisel maliyeti hesaplama işlemi diğer yöntemlerden farklı bir şekilde gerçekleşmektedir. Toplam sezgisel maliyet, çapraz ve düz geçişler için ayrı ayrı hesap edildikten sonra elde edilir. Hareket 8 yönde olduğunda kullanılır.

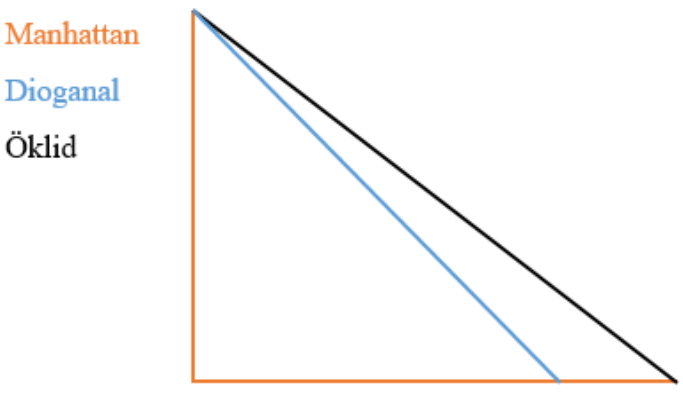
h (n) = cost * max(abs(n.x-goal.x), abs(n.y-goal.y))

Çapraz olarak hareket etmenin maliyeti yatay veya dikey olarak hareket etmeyle eşdeğerse, daha karmaşık bir seçenek olduğunda kullanılır.

h_diagonal (n) = min (abs (n.x-goal.x), abs (n.y-goal.y))
h_straight (n) = (abs (n.x-goal.x) + abs (n.y-goal.y))
h(n) = Diagonalcost * h_diagonal(n)+cost * (h_straight(n) - 2*h_diagonal(n)))
h_diagonal (n) = bir köşegen boyunca atabileceğiniz adımların sayısı.
h_straight (n) = Manhattan uzaklığı

Herhangi bir hareket yönüne izin verildiğinde kullanılır.

h(n) = cost * sqrt((n.x-goal.x)^2 + (n.y-goal.y)^2)

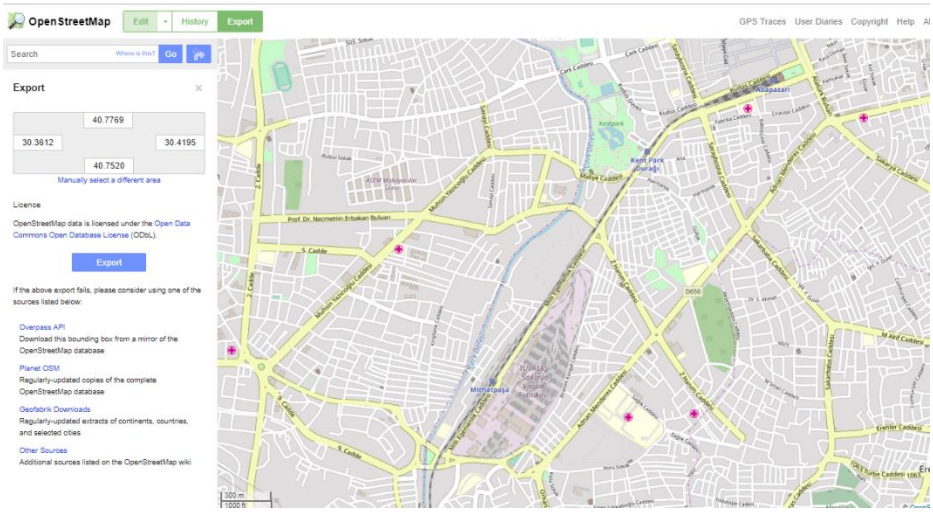


Şekil 3.3. Mesafeler örneği

BÖLÜM 4. SİMÜLASYON GERÇEKLEŞTİRME ADIMLARI

4.1. OpenStreetMap

OSM, dünyaya ait coğrafi verilerin yaratılıp serbestçe dağıtılabildiği bir projedir [42]. Ücretsiz olan diğer çevrimiçi haritalar, teknik ya da yasal kullanımdaki sınırlamalar nedeniyle yaratıcı ve üretken bir şekilde kullanılabilecek bir harita hizmeti oluşturmak amacıyla 2004 yılında ortaya çıkmıştır. O zamanlar, İngiltere'nin en prestijli üniversitelerinden biri olan University College London'da bir öğrencisi Steve Coast, İngiltere'nin askeri harita servisi tarafından kontrol edilen topografik haritalara sınırlı erişimden rahatsızlık duyarak ve Wikipedia'nın başarısından ilham alarak bir harita servisi geliştirmeye karar vermiş ve OSM'nin temellerini atmış [42]. Coğrafi konumlarıyla bir veritabanında saklanan veriler, Mapnik adlı bir tarayıcı kullanılarak görselleştirilir ve haritalara dönüştürülür. Bu harita, www.openstreetmap.org [43] adresinde mevcut olan açık katmanlı bir arayüz aracılığıyla kullanıcılara açıktır. Bir OSM görüntüsünün örneği Şekil 4.1.'de gösterilmiştir.



Şekil 4.1.OpenStreetMap görüntüsü

2004 yılından başlayan OpenStreetMap, çok az kısıtlama ile birçok farklı amaç için kullanılabilecek, düzenlenebilir ve serbestçe erişilebilir bir haritalama bilgisi veritabanıdır [44]. OpenStreetMap'in üç ana bileşeni vardır [45], bunlar düğüm, yol ve ilişkidir. İlişki yani iletişim, diğer üyelerden, yani düğümden ve yollardan oluşan OpenStreetMap veri yapısının en önemli unsurudur. OpenStreetMap popüler hale geliyor ve birçok hevesli kullanıcı bu kalabalık kaynaklı harekete katılıyor. Bu devrimci hareket, yaklaşık Tablo 4.1.'deki gibi istatistiğe sahip oluyor [46].

Tablo 4.1. OpenStreetMap 2019 istatistik raporu

Kıllanıcı Sayısı	5116532
Yüklenen GPS noktalarının sayısı	7130091147
Düğüm sayısı	4970069766
Yolların sayısı	555437662
İlişki sayısı	6498138

4.1.1. OpenStreetMap'ta kullanılan harita üretim tekniği

Adapazarı topolojisi tipik bir şehir alanıdır. SUMO'da yol ağı XML dosyaları kullanılarak modellenmiştir. Elle yazılabilir, düğümler ve kenarlar gibi harita ilkellerini tanımlayabilir ve daha sonra bunları birbirine bağlayabilir veya veriler doğrudan OpenStreetMap (OSM) gibi başka kaynaklardan alınabilir. OSM, tüm dünyaya harita sağlayan ücretsiz bir kullanıcı düzenleme projesidir. OpenStreetMap, isteyen herkes için sokak haritaları gibi ücretsiz coğrafi veriler oluşturur ve sağlar. Proje başlatıldı çünkü özgür olduğunu düşündüğünüz haritaların çoğu, insanları yaratıcı, üretken veya beklenmedik şekillerde kullanmalarını engelleyen, kullanımları konusunda yasal veya teknik kısıtlamalar içeriyor [47]. OSM sadece yol bilgisi sağlayabileceği için değil, yapı şekilleri, otoparklar, otobüs durakları ve diğer birçok İlgi Çekici Nokta (İÇN) gibi bilgileri de topladığı için seçilmiştir (bkz. Şekil 4.2.).

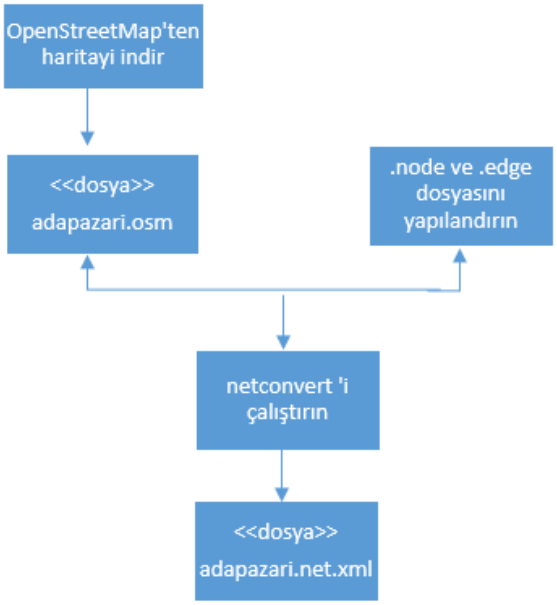
```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.6.1 (2534 thorn-01.openstreetmap.org)"
copyright="OpenStreetMap and contributors" attribution="http://
www.openstreetmap.org/copyright" license="http://opendatacommons.org/licenses/
odbl/1-0/">
 <bounds minlat="40.7538000" minlon="30.3621000" maxlat="40.7760000"
maxlon="30.4057000"/>
 <node id="26566501" visible="true" version="1" changeset="237344"
timestamp="2007-03-16T15:32:19Z" user="hakan" uid="4388" lat="40.7502300"
lon="30.3853800">
  <tag k="created_by" v="JOSM"/>
 </node>
 <node id="26566503" visible="true" version="1" changeset="237344"
timestamp="2007-03-16T15:32:20Z" user="hakan" uid="4388" lat="40.7506000"
lon="30.3851760">
```

Şekil 4.2. *.osm uzantılı dosya

4.1.2. OSM dosyasından *.net.xml dosyasına dönüştürme

Tüm bu farklı seçeneklerle sonuç *.osm adlı bir XML dosyası olur veya *.osm.xml. SUMO simülasyonu için kullanmak için NETCONVERT [17] aracını OSM dosyasını *.net.xml dosya formatına dönüştürmek için kullanabiliriz (bkz. Şekil 4.3.). Bunu yapmanın en basit yolu basit bir komuttur:

Netconvert –osm *.osm –o *.net.xml



Sekil 4.3. *.osm dosyasın *net.xml'e dönüştürme adımları

OSM'den indirdiğimiz harita ihtiyaç duyduğumuzdan daha fazla bilgi içeriyor. Bu yüzden otobüs durakları ve trafiği etkileyebilecek iş yerleri gibi ilgilendiklerimizi manuel olarak seçmeliyiz. Düğümler, yollar ve bunların ilişkileri ve meta veri etiketleri gibi OSM verilerini yükleme ve düzenleme olanağı sağlar [41]. Önceki aşamalarda geometrik nesneler değişmeden kalmıştı ve şimdi onları haritamıza renkli çokgenler olarak yerleştirmemiz gerekiyor, böylece onları kullanabilir ve SUMO-GUI'de görmeyi kolaylaştırabiliriz. SUMO'da şöyle tanımlanır:

< poly id =" < POLYGON_ID >"
type =" < TYPENAME >"
color =" < COLOR >"
fill =" < FILL_OPTION >"
layer =" < LAYER_NO >"
Shape =" [ ]*"/ >

Binaları ve park yerlerini OSM dosyasından SUMO dosyasına (Şekil 4.4.) (.poly.xml) dönüştürmek için POLYCONVERT kullanıyoruz [48].

```
    <poly id="636615956" type="landuse" color="194,194,130" fill="1"
layer="-3.00" shape="2317.23,4501.50 2330.27,4509.61 2337.74,4510.81
2343.85,4508.89 2344.75,4512.80 2340.48,4518.13 2339.10,4524.62 2339.21,4528.55
2334.13,4522.25 2332.34,4520.41 2326.70,4511.60 2322.13,4507.02
2317.23,4501.50"/>
    <poly id="636615957" type="landuse" color="194,194,130" fill="1"
layer="-3.00" shape="2294.66,4470.83 2288.22,4472.45 2282.89,4468.35
2284.26,4455.88 2290.02,4431.77 2293.20,4432.46 2292.02,4440.53 2291.79,4448.72
2293.33,4463.47 2294.66,4470.83"/>
```

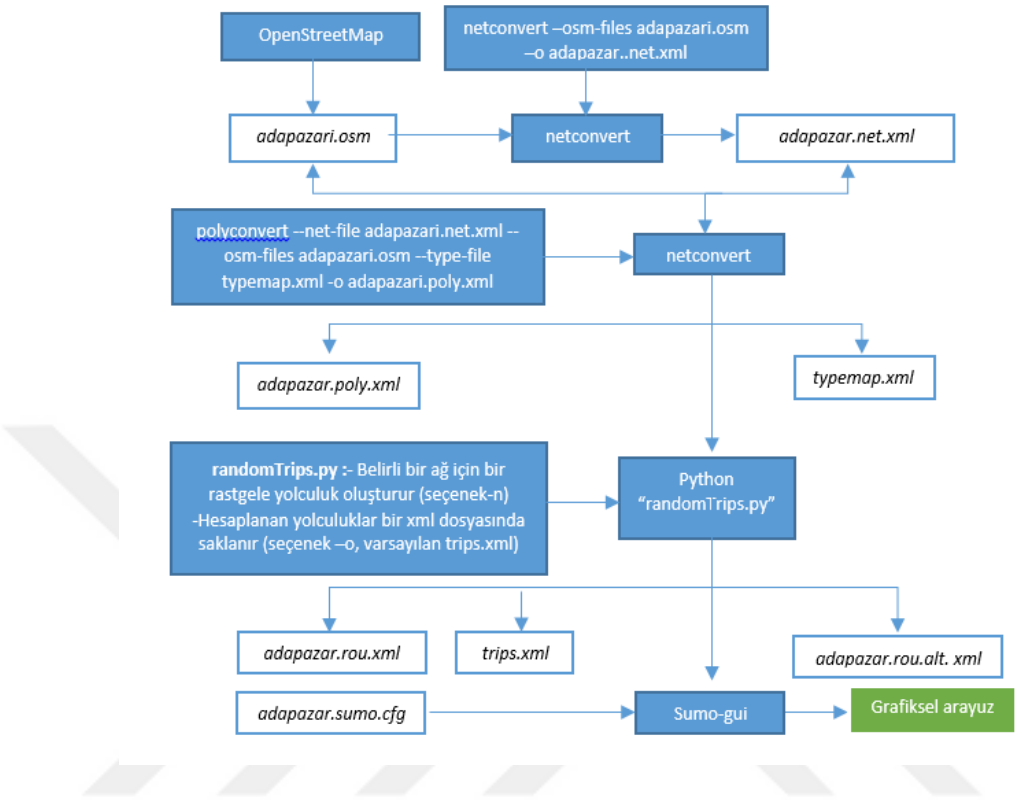
Şekil 4.4. *.poly.xml dosyası

Topolojiyi oluşturduktan sonra simüle edilecek haritadan Tablo 4.2.'deki bilgilere ulaşabiliriz.

Tablo 4.2. Topoloji bilgileri

Toplam düğüm sayısı	2945
Toplam kenar sayısı	6985
Toplam kenar uzunluğu [km]	441.81
Toplam şerit uzunluğu [km]	499.91

4.2. SUMO'nun Simülasyon Çalıştırma Adımları



Şekil 4.5. SUMO adımlarına genel bakış

SUMO paketinde kullanılan her dosya XML olarak kodlanmıştır. SUMO dosyalarının çoğu (yol ağı, rota veya talep, altyapı açıklamaları, vb.) SUMO'ya özeldir, hiçbir standarda uymuyor. SUMO simülasyonu çalıştırmanın iki yolu vardır. Birincisi sumo ve ikincisi de sumo-gui. Her ikisi de komut satırı programlarıdır. Aralarındaki fark, sumo simülasyonu grafiksel bir kullanıcı arayüzü olmadan arka planda çalıştırmasıdır. Bu durumda, tüm simülasyon arka planda çalışır. Sumo-gui ise çalışan simülasyonu gösteren grafiksel bir kullanıcı arayüzün sunar (bkz. Şekil 4.5.). Başlamak için gerekli temel iki farklı bilgi parçası aşağıdaki gibidir [49].

- Ağ Topolojisi
- Trafik Durumu Talebi

Bir Ağ Topolojisi, karayolları, demiryolları, yaya yolları, su yolları veya diğer hareketli araba, otobüs, tramvay, kamyon, tren, tekne veya insan ağını içerir. Trafik Durumu Talebi, ağ boyunca belirli bir düzende hareket eden arabaları, otobüsleri, tramvayları, kamyonları, trenleri, tekneleri veya insanları içerir. Bu iki gereksinim, bir simülasyonu çalıştırmak için tanımlanması gereken bir yapılandırma açısından birleştirilebilir.

4.2.1. Ağ topolojisi

Ağ dosyasını yanı sıra, ağ topolojisini (*.net.xml) bir veya daha fazla rota dosyasını tanımlayarak, araçların trafik şeklini açıklayan (*.rou.xml), ek dosyalar da (*.add.xml) belirleyebilirmişiz. Bu ek dosyalar isteğe bağlıdır ve örneğin: Ağa monte edilmiş endüksiyon döngülerinin veya diğer sensör türlerinin tanımlanması, ilgili bölgenin hava fotoğrafçılığını içeren bir arka plan görüntüsü, bina isimleri, alışveriş mağazaları gibi çevrenin daha net bir şekilde görüntülenmesini sağlar, veya simülasyonun görselleştirilmesinde yer almak istediğiniz diğer işaretler veya istediğiniz poligonal bölgeler simülasyon görselleştirmesinde temsil eder. SUMO'da bir XML dosyasında bir yapılandırma (configuration) tanımlanabilir (bkz. Şekil 4.6.)

```
<?xml version="1.0" encoding="iso-8859-1"?>

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.sf.net/xsd/sumoConfiguration.xsd">

    <input>
        <net-file value="adapazari.net.xml"/>
        <route-files value="adapazari.rou.xml"/>
        <additional-files value="adapazari.poly.xml"/>

    </input>

   <output>
      <netstate-dump value="adapazari.sumo.tr"/>
      <tripinfo-output value="output-tripinfos.xml"/>
      <emission-output value="output-emissions.xml"/>
      <vehroute-output value="output-vehroutes.xml"/>
<queue-output value="queue.xml"/>
    </output>

    <time>
        <begin value="0"/>
        <end value="500"/>
    </time>

</configuration>
```

Şekil 4.6. Yapılandırma örneği (*.sumo.cfg)

4.2.2. Trafik durumu talebi

Bir ağ oluşturduktan sonra, SUMO-GUI kullanılarak bir göz atılabilir, ancak hiçbir araba hareket ettirilemez. Ayrıca bir tür taşıt tanımına ve bunların ağ topolojisinde nasıl hareket ettiklerini de açıklanması gerekiyor. Buna trafik talebi denir. Trafik talebini anlamak için aşağıdaki terminolojiye ihtiyaç vardır [50].

Yolculuk – bir aracın (veya insanların) bir yerden bir yere hareket etmesidir.

Bir rota – genişletilmiş bir yolculuktur, yani bir rota tanımının yalnızca ilk ve son kenarı değil, aynı zamanda taşıtın başlangıç noktasından son varış noktasına kadar tüm kenarlarını geçeceği anlamına gelir. SUMO ve SUMO-GUI, araç trafiğine girdi olarak rotalara ihtiyaç duyar.

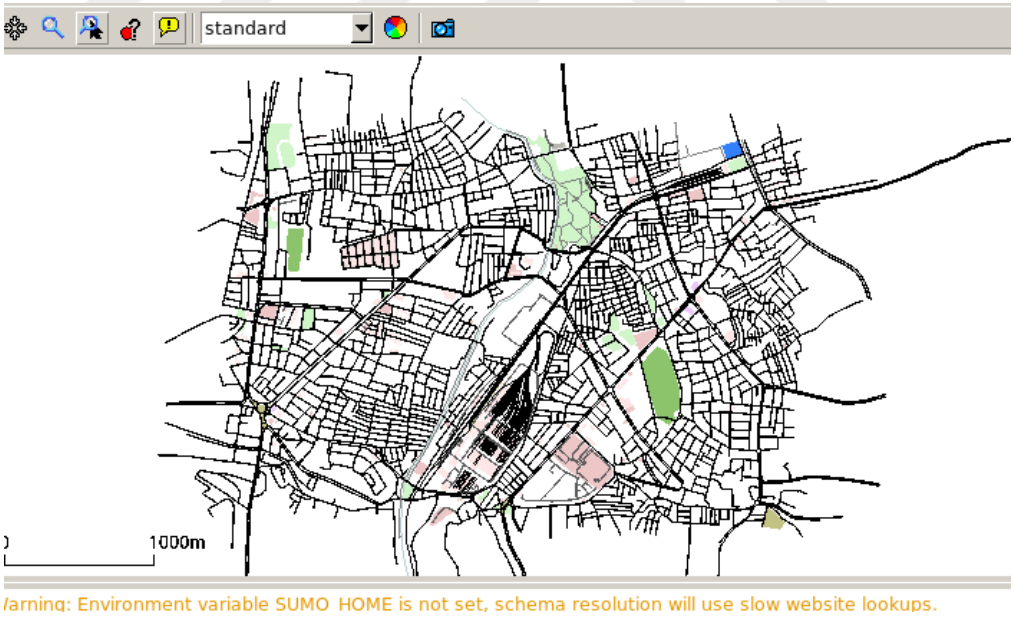
Şimdiye kadar SUMO rota oluşturmak için dört uygulama içeriyor.

DUAROUTER, Dijkstra'nın en kısa yollu algoritmasını kullanan rotaları veya tanımlarını diğer simülasyon paketlerinden ithal etmekten ve bilgi işlem rotalarından sorumludur. Ek olarak, simülasyonla birlikte DUAROUTER, C. Gawron tarafından formüle edilen dinamik kullanıcı atamasını hesaplayabilir. Başka bir seçenek olan JTRROUTER, kavşaklardaki geri dönüş akışlarını ve yüzdelerini kullanarak trafiği istatistiksel olarak simüle etmek istiyorsak kullanılabiliriz. Üçüncü bir seçenek olan OD2TRIPS, OD matrislerini (başlangıç / hedef matrisleri) triplere dönüştürmemize yardımcı olur. Son seçenek DFROUTER, seçilen gözlem noktalarından toplanan bir dizi ölçümden rotaları hesaplar. Rota dosyasını kendi içinde oluşturmayan ancak DUAROUTER ile birlikte kullanılabilecek bir seçenek, popülasyon istatistiklerini bir gezi dosyasına dönüştürmek ve ardından trafik talebi oluşturmak için DUAROUTER kullanmak için kullanılabilen ACTIVITYGEN programıdır [51].

4.2.3. Görsel kılavuz

SUMO GUI versiyonu, orijinal modelleme paketini pencereli ve grafiksel bir arayüz kullanarak genişleterek uygulanmıştır. Geometrinin hızlı görselleştirilmesi için Qt Windowing [52] kütüphanesine ve openGL'ye [53] dayanmaktadır. Bu yaklaşım en yaygın işletim sistemlerinde kuruluma izin verir: UNIX, Windows ve Linux. Netconvert çalıştırıldıktan sonra SUMO ağı, SUMO GUI'de (sumo-gui –n adapazarı.net.xml) görselleştirilebilir. GUI kavşakların doğru bir şekilde birleştirildiğini doğrulamak ve (trafik simülasyonlarını tamamladıktan sonra) ağ içindeki araçların konumlarını önizlemek için yararlı bir araçtır. Çalışmakta olan örneğimizden SUMO sokak ağı aşağıda gösterilmiştir adapazari.sumocfg bir yapılandırma dosyası göz önüne alındığında, SUMO'da bir komut satırında basitçe yazarak bir simülasyon başlatılabilir (Şekil 4.7.):

sumo –c adapazari.sumocfg veya sumo-gui –c adapazari.sumo.cfg



Şekil 4.7. SUMO'da adapazarı sokak ağı

4.2.4. Rastgele trafik belirtme

SUMO simülasyonu için trafik oluşturmanın en kolay yolu rasgele oluşturulmuş trafik kullanmaktır. Kullanıcı, belirli bir ağ için rastgele veya değiştirilmiş bir başlangıç veya hedef kenar seçecek bir rastgele gezi kümesi oluşturmak için randomTrips.py (SUMO'daki tools dizininde bulunabilir) olan Python skriptini kullanabilir. Elde edilen geziler otomatik olarak çağrılabilen DUAROUTER için uygun bir XML dosyasında kaydedilir [51]. Geziler, yapılandırılabilir başlangıç ve bitiş zamanları tarafından tanımlanan bir aralıkta eşit olarak dağıtılır. Her yolculuk bir önek ve bir çalışan numaradan oluşan bir kimliğe sahiptir. Örneğin aşağıdaki komut ve çıktı olarak *.rou.xml (Şekil 4.8.):

randomTrips.py –n adapazari.net.xml –e 500 –r adapazari.rou.xml

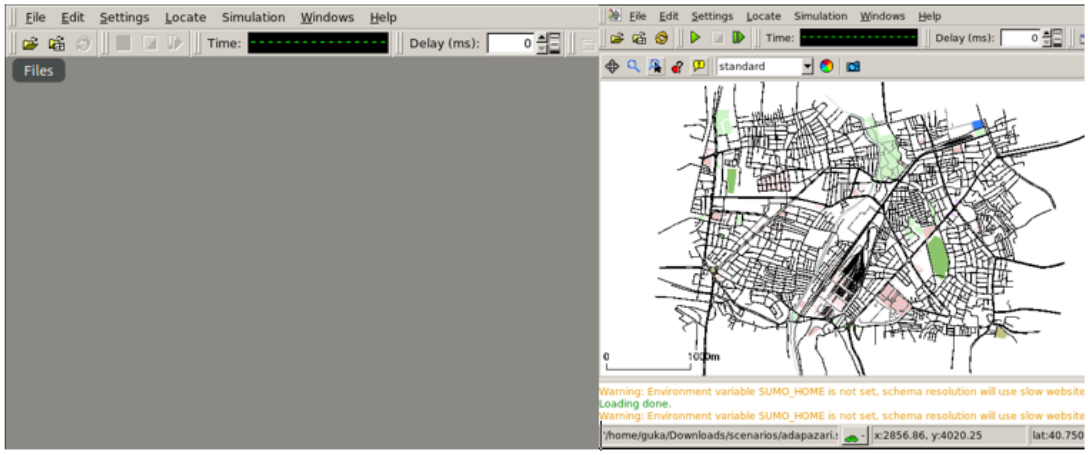
```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
    <vehicle id="0" depart="0.00">
        <route edges="215225054#20 326272006#18 326272006#19 326272006#20
326272006#21 326272006#22 326272006#23 326272006#24 326272006#25 326272006#26
326272006#27 326272006#28 326272006#29 326272006#30 326272006#31 326272006#32
326272006#33 326272006#34 326272006#35 194897789 637680657#1 637680657#2
140868470 -243832813#2 -243832813#1 -243832813#0 -140868446#3 -140868446#2
-140868446#1 -140868446#0 140868434 637681786#3 194744067 -48337967#5
-48337967#4 -48337967#3 -48337967#2 -48337967#1 -48337967#0 138222575#0
138222575#1 138222575#2 138222575#3 138222575#4 -140817759#1 491292367
491330755#2 138225034#0 138225034#1 138225034#2 140817684#0 140817684#1
215255753#0 -215255753#0"/>
    </vehicle>
```

Şekil 4.8. adapazari.rou.xml

BÖLÜM 5. SONUÇ

5.1. Simulasyon Sonuçları

Bir *.net.xml ve *.route.xml dosyalarını kullanarak, sumo'da sumo veya sumo-gui komutlarıyla tam bir simülasyon çalıştırmak mümkündür. Sumo-gui durumunda, kullanıcı ağda akan araçları görebilir ve araç akışının daha ayrıntılı bir görünümü için topolojinin belirli kısımlarını yakınlaştırabilir (Şekil 5.1.). Ancak, simülasyonun sona erene kadar hiçbir şey üretilmiyor. SUMO çok sayıda farklı önlemin üretilmesine izin verir.

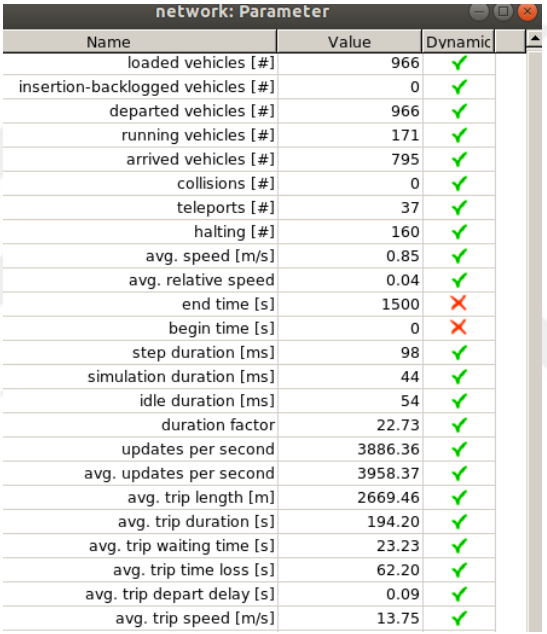


Şekil 5.1. sumo-gui çalıştırdığımızda aldığımız pencere

Varsayılan olarak hepsi devre dışıdır ve ayrı olarak çalıştırılmalıdır. Mevcut çıkış verilerinden bazıları (araç kaynak verisinin sıfırlanması, yol bilgileri, araç rota bilgileri ve simülasyon durumu istatistikleri) komut satırı parametreleri kullanılarak çalışır. Diğerleri simülasyon yapılandırmasına eklenen "ek dosyalar" içerisinde tanımlanmalıdır.

Ağ parametreleri (--duration-log.statistics) seçeneği açıkken, tüm araç gezileri için aşağıdaki ortalamalar mevcut olacaktır (Şekil 5.2.):

- RouteLength: ortalama rota uzunluğu
- Duration: ortalama seyahet süresi
- WaitingTime: ortalama bekleme süresi
- TimeLoss: istenenden daha yavaş hareket nedeniyle kaybedilen ortalama süre.



Sekil 5.2. Ağ parametreleri (Duration-log.statistics)

Bu seçeneği ayarlarken ve SUMO-GUI kullanıldığında, ağ parametresi iletişim kutusu ayrıca bu trafik önlemleri için ortalama bir çalışma gösterir.

5.1.1. Yeniden yönlendirme ve kısa yol bulma

SUMO simülatöründe aşağıdaki parametrelerle, araçlarda yeniden yönlendirme kullanmak mümkündür. Cihazi deterministik yönlendirme

(Device.rerouting.deterministic) ile yeniden yönlendirme kabiliyetinin ağ durumunun tahmininde ileriye bakması mümkün olabilir.

<processing>
 <routing-algorithm value=”astar”/>
</processing>
<routing>
<device.rerouting.probability value=”1.0”/>
<device.rerouting.period value=”1”/>
<device.rerouting.deterministic value=”true”/>
</routing>

Simülasyonun yeniden yönlendirme parametreleri Tablo 5.1.’de gösterilmiştir. Burda 50,100,500 ve 1000 sayıdaki araçları ayrı ayrı simüle edilerek aşağıdaki sonuçlar elde edilmiştir. Tablo 5.1.’deki sonuçlar simülasyona ilk giren araç için alınmıştır.

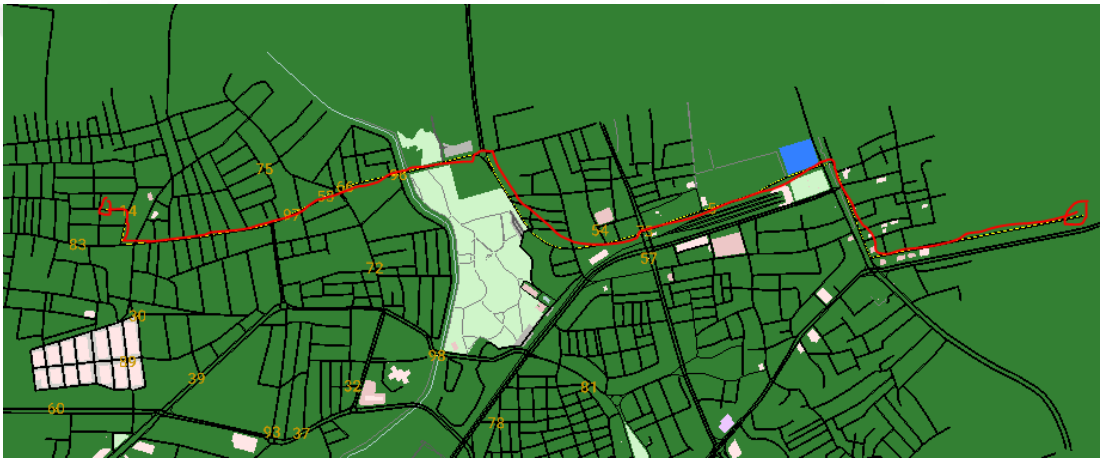
Tablo5.1. Yeniden yönlendirme

Araçların sayısı	Yeniden yönlendirmesiz		Yeniden yönlendirme algoritmaları			
			A Yıldız		Dıjkstra	
	Rota uzunluğu	Simülasyonu tamamlama zamanı	Rota uzunluğu	Simülasyonu tamamlama zamanı	Rota uzunluğu	Simülasyonu tamamlama zamanı
50	1654	731	654	460	670	470
100	2489	1366	577,82	454	577,82	454,03
500	5478	3922	671	936	689,12	997
1000	10828,17	25617	428,66	1750	430,66	1764

Örneğin 100 araçla simülasyonun sona ermesi için (time-end olarak) 500 saniye verdiğimizde yeniden yönlendirmesiz çalışmasında simülasyon alanında 83 araç giriyor ve onların hepsi verilen süre içinde hedeflerine ulaşamıyorlar. 680. saniyeden başlayarak araçlar hedeflerine ulaşıp alandan çıkmaya başlıyorlar. Yeniden yönlendirme yapıldığında araçların hepsi hedeflerine verilen süre içinde ulaşıyorlar.

Misal olarak 14 numaralı aracın ve rotasın verebiliriz. Aşağıda14 numaralı aracın rota bilgileri verilmiştir. Trip id=”14” depart=”14.00” from=”548201254#3" to="-215233549#2"/>

Şekil 5.3.’te ise 14 numaralı aracın en kısa yol bularak ve az zaman harcayarak hedefine ulaşıyor. Şekil 5.4.’te ise yeniden yönlendirme yapılmadan 14 numaralı aracın hedefine ulaşma rotası verilmiştir. Kırmızı ile işaretlenen yerler çıkış ve varış noktasıdır.



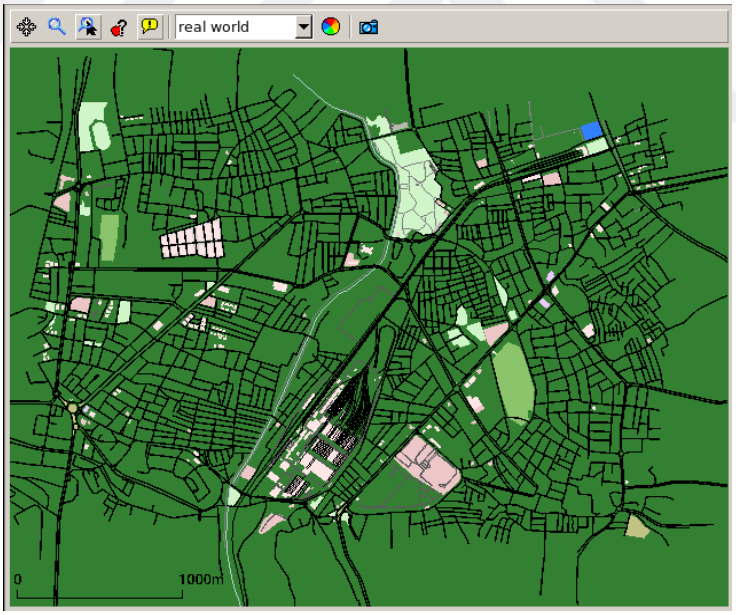
Şekil 5.3.14 numaralı aracın en kısa rotası

Şekil 5.4. 14 numaralı aracın yeniden yönlendirme yapılmadıgı rota

Tablo 5.1.'de yeniden yönlendirme yapıldığında araç en kısa yolu bularak hedefine ulaşıyor ve az süre harcanıyor. SUMO yeniden yönlendirme yapıldığında varsayılan olarak Dijkstra algoritmasında çalışıyor ve A Yıldız algoritması için ayrı kod ekleyerek çalıştırmamız gerekiyor. Simülasyonu çalıştırdığımızda A Yıldız algoritması, Dijkstra algoritmasından daha kısa sürede en kısa yolu bulmaktadır. Yeniden yönlendirme yapılmadan çalıştırıldığında siülasyon hesaplama süresi artmakta rota uzunluğu da yüksek çıkmaktadır.

5.1.2. SUMO çiktilari (outputtripinfos.xml)

SUMO, her simülasyon çalışması için çeşitli çıktılar (*.out.xml) üretmeye izin verir, görselleştirme SUMO-GUI (Şekil 5.5.) kullanılarak yapılır. Simülasyonun çıktısı ortalama bekleme süresi, seyahat bilgileri kullanılarak yapılmıştır. Bu dosya tipi XML olarak oluşturulur ve Şekil 5.6.'deki gibi < output-tripinfos.xml > olarak adlandırılır.



Şekil 5.5. Adapazari.net.xml-sumo-gui görüntüsü

```
<tripinfos xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/tripinfo_file.xsd">
    <tripinfo id="23" depart="23.00" departLane="-212149210#1_0"
departPos="5.10" departSpeed="0.00" departDelay="0.00" arrival="66.00"
arrivalLane="139679795#2_0" arrivalPos="116.02" arrivalSpeed="13.33"
duration="43.00" routeLength="654.73" waitSteps="0" timeLoss="9.51"
rerouteNo="0" devices="vehroute_23 tripinfo_23" vType="DEFAULT_VEHTYPE"
speedFactor="1.00" vaporized=""/>
```

Şekil 5.6. Sumo çıktısı output-tripinfos.xml.

Sımülasyonun çıktısında her bir aracın seyahet bilgisi verilmektedir.

Bizim inceleyeceğimiz parametrelere bakarsak (Şekil 5.6.)

<tripinfo id="23"> simülasyona ilk olarak giren aracın numarasıdır

<duration="43.00" >23 numaralı aracın simülasyonun çalışmasından başlayarak hedefine ulaşana kadar harcanan süresidir

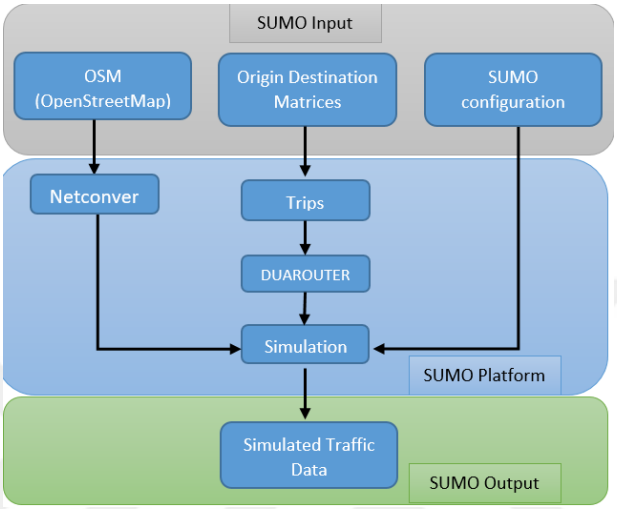
<routeLength="654.73" > 23 numaralı aracın rotasının uzunluğu

<timeLoss="9.51" > 23 numaralı aracın simülasyon sırasında ideal hızın altında sürüş nedeniyle kaybedilen zaman (ideal hız bireysel speedFactor'u içerir; kesişimlerden kaynaklanan yavaşlamalar vs. zaman kaybına neden olur, zamanlanmış duraklamalar sayılmaz)

Tablo 5.2. Algoritmaların süreye göre karşılaştırılması

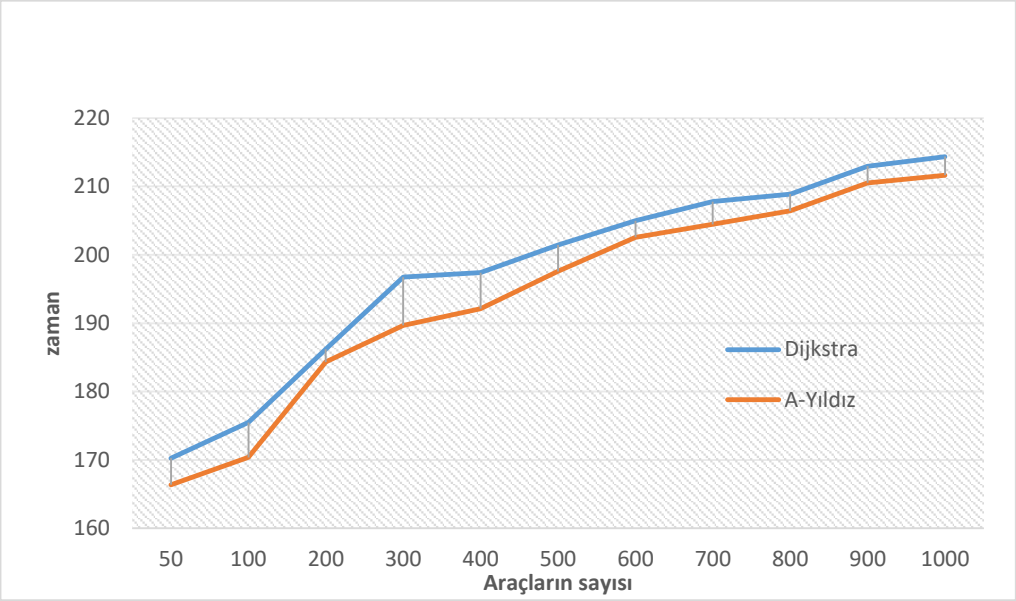
A –Yıldız ile Dijkstra algoritmalarının ortalama süre ve ortalama zaman kaybı performansi				
Araç Sayısı	Dijkstra		A Yıldız	
	Süre (Duratıon)	Zaman Kaybı (time loss)	Süre (Duratıon)	Zaman Kaybı (time loss)
50	170,25	39,29	166,37	36,73
100	175,48	40,28	170,37	36,70
200	186,20	42,76	184,32	41,37
300	196,73	44,79	189,68	42,85
400	197,38	44,94	192,12	43,5
500	201,45	45,97	197,57	44,78
600	204,98	47,07	202,57	44,99
700	207,79	46,99	204,45	45,05
800	208,88	47,57	206,42	46,58
900	212,97	49,39	210,52	47,78
1000	214,37	52,61	211,64	49,37

(Tablo 5.2.) Toplam harcanan sürelerin ortalamasın göstermektedir. Bu sonuçlar, simülasyonun çıktısını temsil etmektedir. Simülasyonu çalıştırdığımızda araçların sayısına göre seyahat sürelerinin değiştiği farkediliyor. Bu karşılaştırmada giriş parametreleri olarak haritaya, araçların sayıları alınmıştır (Şekil 5.7.).



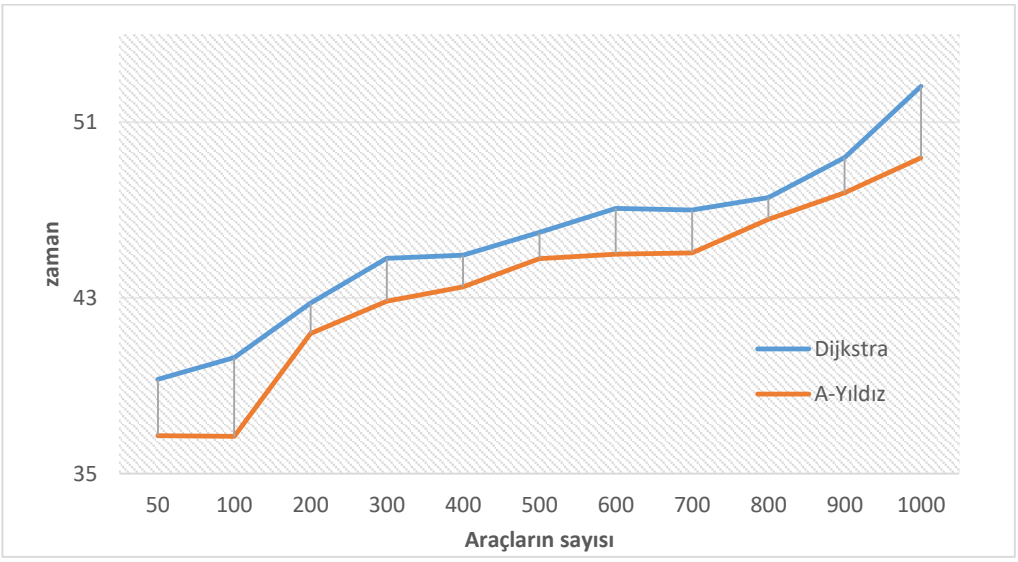
Şekil 5.7. Simülasyon girdi çıktı parametreleri

Burda A Yıldız ve Dijkstra algoritmaları uygulanarak simülasyon sırasında 1000 araç için ortalama veriler alınmıştır. Grafikte gösterildiği gibi A Yıldız algoritması kulanıldığında Dijkstra algoritmasına göre toplam simülasyon çalışma zamanının daga az sürede gerçeklestiği görünmektedir (Şekil 5.8.).



Şekil 5.8. Ortalama harcanan süre

Şekil 5.9.'da aldığımız grafiğe göre A Yıldız ve Dijkstra algoritmaları uygulanarak simülasyon sırasında 1000 araç için ortalama veriler alınmıştır. Grafikte gösterildiği gibi A Yıldız algoritması kulanıldığında Dijkstra algoritmasına göre ideal hızın altında sürüş yaparak az zaman kaybederek simülasyonu tamamlıyor.



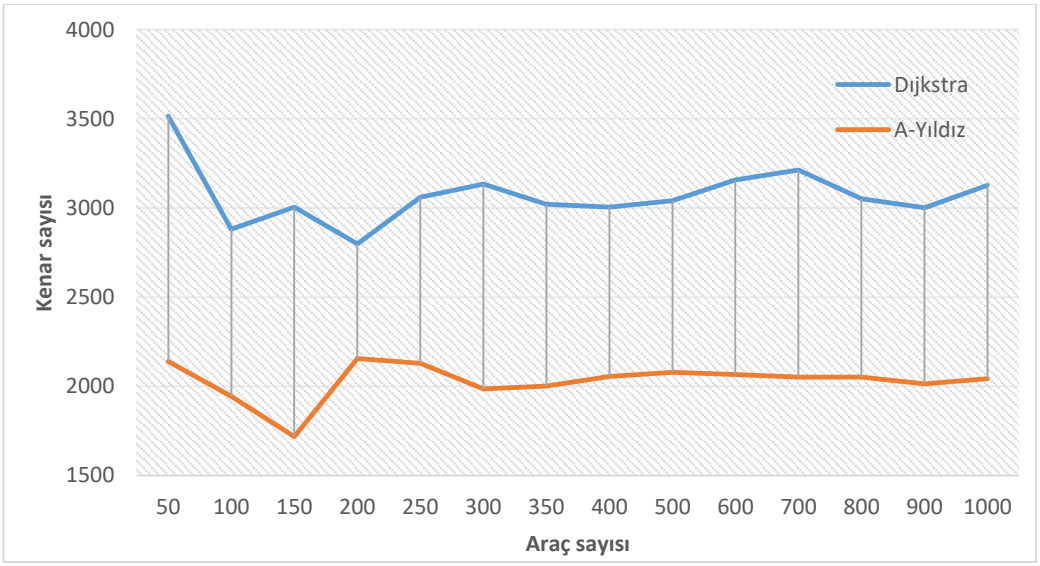
Şekil 5.9. Zaman kaybı karşılaştırılması

A Yıldız ile Dijkstra algoritmalarının 50 den 1000 araca kadar her aracın gittiği kenarın ortalama sayıları Tablo 5.3.'te gösteriliyor.

Tablo 5.3. Kenar sayıları

Araç sayıları	Dıjkstra	A Yıldız
	Ortalama Kenarlar	Ortalama Kenarlar
50	3515	2139
100	2881	1944
150	3004	1717
200	2798	2155
250	3061	2128
300	3134	1985
350	3021	2001
400	3005	2055
500	3040	2079
600	3157	2066
700	3212	2052
800	3051	2052
900	3001	2013
1000	3127	2042

Şekil 5.10.'a baktığımızda Dijkstra algoritmasında A Yıldız algoritmasına göre az kenara gidilmekteir.



Şekil 5. 10. Aracın gittiği kenarların sayılarının karşılaştırılması

Burda maliyet seyahet süresine göre hesaplanmıştır. Her aracın simülasyon sonuna kadar rotası vardır ve o rotadaki hedefine gidene kadar maliyet çıkarılmaktadır (Şekil 5.11.).

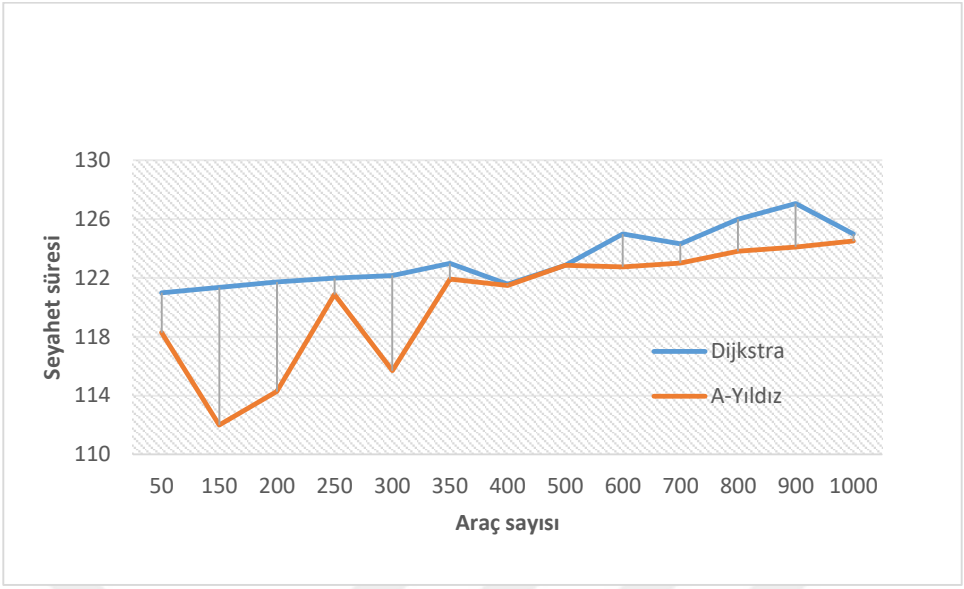
```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
    <vehicle id="0" depart="0.00">
        <routeDistribution last="0">
            <route cost="119.61" probability="1.00000000" edges="215378019#0
-215377962#0 -215377991#10 -215377991#9 215378012#0 215377926#7 -215378033#3
-215378033#2 -215378033#1 -215378033#0 -215378055#0 -215377927#0 551834704
637680653#1 637680653#2 491292391#1 491292391#2 491292391#3 -491292395
140010755#0 140010755#1 140010755#2 140010755#3 155249842#8 140817756
-140817757#3 -140817757#2 -140817757#1 -140817757#0 140817843#0 140817843#1
140817843#2 140817749#0 140817749#1 140817749#2 140817749#3 140817749#4
491330759#0 -140119082#6 -140119082#5 -140119082#4 215246119#0 215246128
-215246128"/>
        </routeDistribution>
    </vehicle>
```

Şekil 5.11. Yolların maliyet hesaplamasından aldığımız sonuçlar

Rota hesaplama seyahet zamanını için hesaplanan maliyet Tablo 5.4.'deki gibidir. Ortalama maliyet hasaplanmıştır ve iki algoritma için karşılaştırma grafiği Şekil 5.12.'de gösterilmiştir.

Tablo 5.4. Maliyet hesaplama

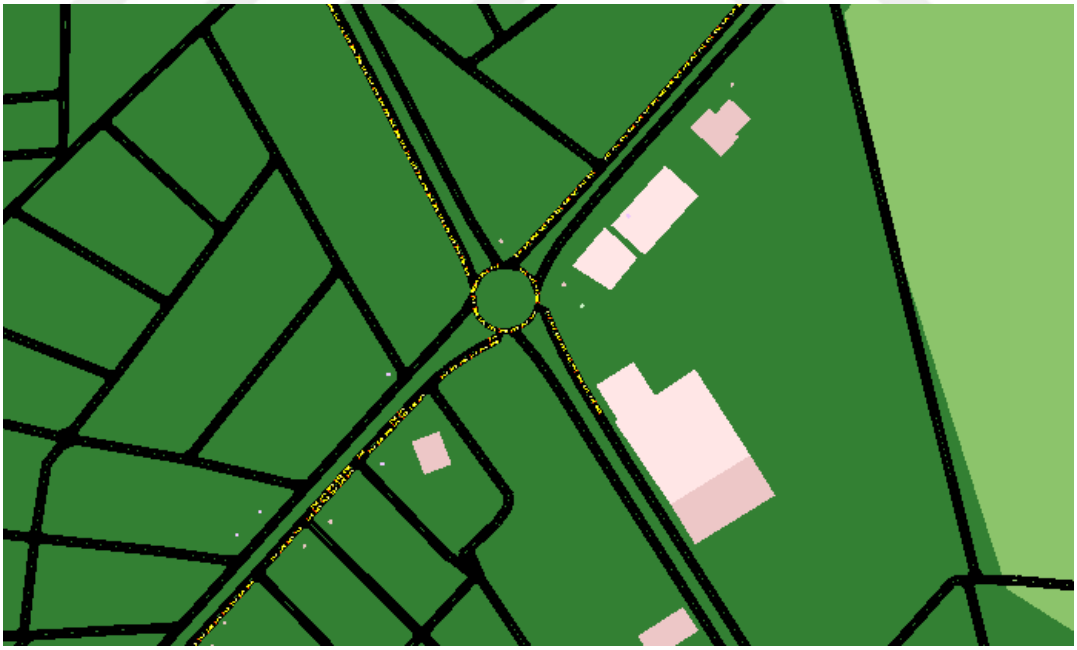
Ortalama Maliyet		
Araç sayısı	Dijkstra	A Yıldız
50	120,99	118,28
150	121,35	112,01
200	121,72	114,28
250	121,98	120,87
300	122,16	115,69
350	122,98	121,92
400	121,57	121,48
500	122,85	122,85
600	124,98	122,75
700	124,32	123,01
800	125,99	123,81
900	127,05	1241
1000	124,98	124,50



Şekil 5.12. Seyahet zamana göre maliyetlerin karşılaştırılması

5.1.3. Kuyruk

SUMO çalışırdığımızda araçların kuyruk gösterimi (Şekil 5.13.)



Şekil 5.13 Sumo'da kuyruk görünümü

SUMO ile “Queue-output” seçeneğini kullanırken tüm kuyrukların süreleri ve uzunlukları bir dosyaya gönderilir. Bu kuyruk verileri farklı sayıda araçların durumlarını karşılaştırmak için kullanılabilir [54]. Kuyruk çıktısını görmek için bkz. Şekil 5.14.

```
    <data timestep="73.00">
        <lanes>
            <lane id="-215378025#0_0" queueing_time="0.00"
queueing_length="0.00" queueing_length_experimental="5.18"/>
            <lane id="491292368#7_0" queueing_time="0.00"
queueing_length="0.00" queueing_length_experimental="5.17"/>
        </lanes>
    </data>
    <data timestep="74.00">
        <lanes>
            <lane id="-215378025#0_0" queueing_time="1.00"
queueing_length="5.11" queueing_length_experimental="5.11"/>
        </lanes>
    </data>
    <data timestep="75.00">
        <lanes/>
```

Şekil 5. 14. queue.output.xml dosya çıktısı

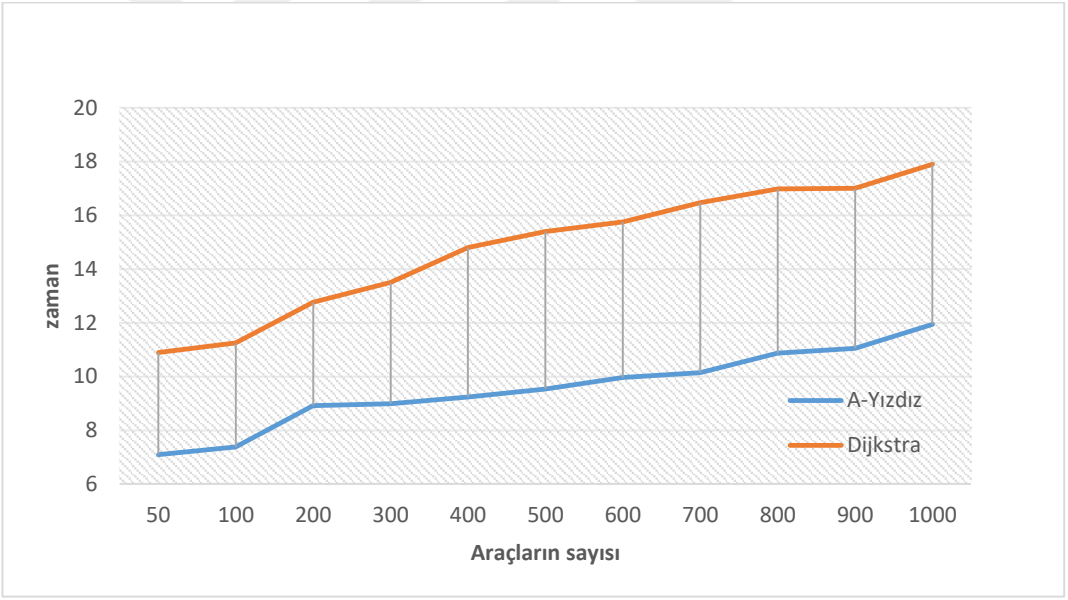
Kuyruk çıktısındaki parametrelerin açıklaması (Tablo 5.5.)

Tablo 5.5. Kuyruk çıktısındaki parametrelerin açıklaması

Adı	Tipi	Açıklama
time_step	(simülasyon) saniye	Bu timestep öğesindeki değerlerle açıklanan zaman adımı
id	id	Şeritin id si
queueing_time	saniye	Kuyruk nedeniyle araçların toplam bekleme süresi
queueing_length	metre	Kavşaktan hattaki son araca kadar olan uzunluk
queueing_length_experimental	metre	Hızı 5 km / s'den düşük son araca kadar kuyruğun uzunluğu

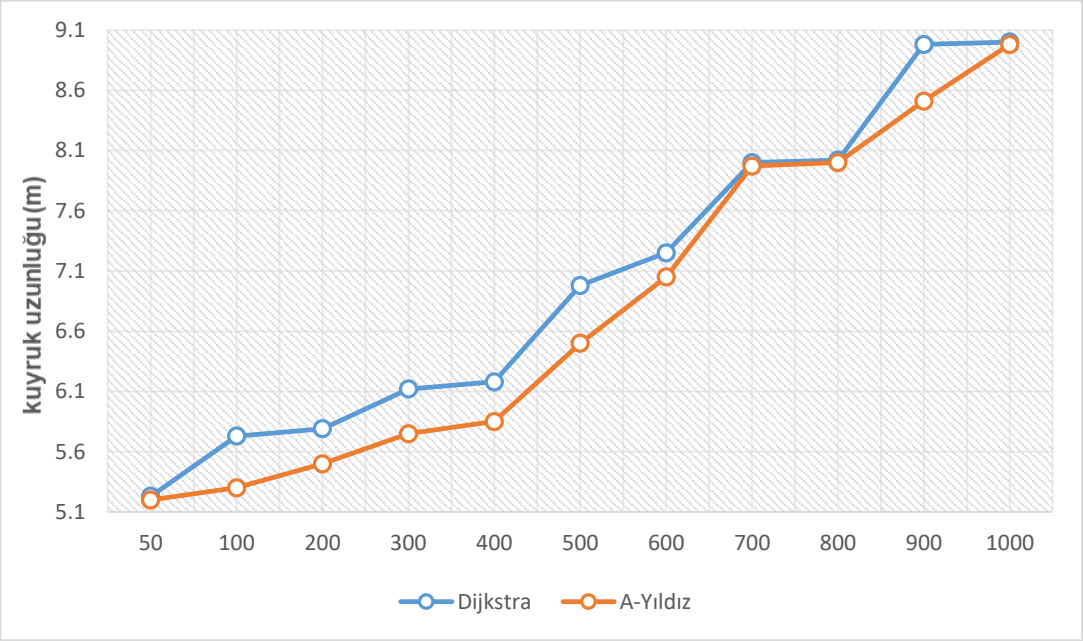
Tablo 5.6. Kuyruk karşılaştırma performansı

A Yıldız ile Dijkstra algoritmalarının ortalama kuyruk performansi				
Araç Sayısı	Dijkstra		A Yıldız	
	Ortalama kuyruk zamanı	Ortalama kuyruk uzunluğu	Ortalama kuyruk zamanı	Ortalama kuyruk uzunluğu
50	10,90	5,23	7,1	5,2
100	11,25	5,73	7,38	5,3
200	12,76	5,79	8,92	5,5
300	13,5	6,12	8,99	5,75
400	14,8	6,18	9,24	5,85
500	15,4	6,98	9,54	6,5
600	15,76	7,25	9,97	7,05
700	16,47	8	1,15	7,97
800	16,98	8,02	10,87	8
900	17	8,98	11,05	8,51
1000	17,9	9	11,94	8,98



Şekil 5.15. Kuyrukta beklenen zaman grafiği

Kuyruk uzunluğu, kavşağa gitmek için beklemek üzere durdurulan araç sayısıdır. Kuyruk uzunluğu araba, otobüs ve kamyon (ağır vasıta) içerir.



Şekil 5.16. Kuyruk uzunluğu

SUMO'da kuyruk her türlü durumlarda simüle edilebilir. Bu tezde yapılan kuyruk çalışması en basit bir şekilde tamamlanmıştır. Simülasyonu farklı sayıda araç sayısını girerek ve simülasyonu tamamlamak için belirli zaman ayırılarak kuyruk bekleme süresi, kuyruk uzunluğu hesaplandı. Grafik olarak Şekil 5.15 ve Şekil 5.16'da verilmistir.

5.2. Sonuç

Bu tez çalışmasında simülasyon yoluyla, araçlarin hedeflerine ulaşmaları için yönlendirme algoritmalarının performansları karşılastırmalı olarak incelenmiş daha verimli olan algoritma belirlenmiş, daha sonrada kuyruk uzunlunluğu, gecikme süresi, ortalama harcanan zaman gibi karşılaştırmalar yapılmıştır.

Simülasyon toplam 1000 araç üzerinde yapılmıştır. Yönlendirme algoritmaları için yapılan değerlendirme çalışması, gerçek ulaşım ağına ve gerçekçi trafik yüküne dayalı olarak gerçekleştirilmiştir. Trafik simülasyonu SUMO üzerinde simüle edilmiştir. Araçların gittiği kenar sayısı, araç sayısı, yolun uzunluğu, verilen süre, kuyruk, en kısa yolu bulma gibi parametreler kullanılmıştır. Kısa yol bulma algoritmalarından A Yıldız

ve Dijkstra algoritmaları seçilerek SUMO ve OpenStreetMap (OSM) kullanılarak şehir trafiği gerçege uygun olarak simüle edilmiştir.

Yapılan simülasyonun sonuçlarına dayanarak, şu sonuçlara varılabilir:

- Dijkstra algoritması ve A Yıldız algoritması oldukça hızlı bir sürede en iyi sonuçları sağlamıştır.
- Dijkstra ve A Yıldız algoritmaları yeniden yönlendirme yapıldığında en iyi sonuçları göstererek en kısa yolu bulmuştur.
- Dijkstra algoritmasının ve A Yıldız algoritmasının harcanan zamanlarını karşılaştırdığımızda az bir farkla A Yıldız algoritması en iyi sonuçları sağlamıştır.
- Değişken tıkanıklık seviyesi ve rastgele trafikteki araçların sayısı değiştirilerek dinamik yol ağlarında Dijkstra ve A Yıldız algoritmasının verimliliği simüle edilmiştir.
- Kuyruk karşılaştırma performansında A Yıldız algoritması Dijkstra algoritmasına çok yakın değerler alarak öne çıkmıştır.

Bu çalışmada A Yıldız ve Dijikstra algoritmalarının performansı ve guvenirliği ispatlanmıştır. Performans değerlendirmesi çalışmaları için elde ettiğimiz sonuçlara bakarak en kısa yol bulmak için A Yıldız algoritması daha performanslı olduğu görülmuştür.

KAYNAKLAR

[1] Aydemir T., Başlangıç - Son matrisinin İzmir'deki Dönel Kavşak Giriş Kapasitesi Üzerindeki Etkisinin Belirlenmesi, Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü, İnşaat Mühendisliği Bölümü, Ulaştırma Anabilim Dalı, Yüksek Lisans Tezi, 2006.

[2] Tufan, H., Akıllı Ulaşım Sistemleri Uygulamaları Ve Türkiye İçin Bir Aus Mimarisi Önerisi, Ulaştırma ve Haberleşme Uzmanlığı Tezi, Ankara, 2014.

[3] Rafael, R., P. ve Manuel, S., L., Algorithm for shortest path search in Geographic Information Systems by using reduced graphs, Rodríguez-Puente and Lazo-Cortés SpringerPlus, 2:291, 2013

[4] Yıldırım, İ., Afet acil durum yönetimi ile ilgili akıllı ulaşım sistemleri uygulamaları üzerine bir araştırma. Yüksek Lisans Tezi. Bahçeşehir Üniversitesi Fen Bilimleri Enstitüsü, 2016.

[5] Japan Society of Traffic Engineers, Intelligent Transport Systems (ITS), Maruzen Press, Tokyo, 1997.

[6] Iguchi, M.A., Perspective on ITS deployment, JSAE Review 23, pp: 173-176., 2002.

[7] Hayashi H. ve Morisugi H., International comparison of background concept and methodology of transportation project appraisal, Transport Policy, 7 (1): 73-88., 2000.

[8] SUMO - Simulation of urban mobility, www.sumo.sourceforge.net., Erişim Tarihi: 11.11.2018.

[9] Treiber Microsimulation of road traffic applet, www.traffic-simulation.de., Erişim Tarihi: 12.11.2018.

[10] TSS-Traffic Simulation Systems: Aimsun website, www.aimsun.com., Erişim Tarihi: 12.11.2018.

[11] Kotusevski, G. ve Hawick, K.A., Review of Traffic Simulation Software Computer Science, Institute of Information and Mathematical Sciences, Massey University, Albany, NS 102-904, Auckland, New Zealand, 2009.

[12] SUMO, www.sumo.dlr.de/wiki/TraCI., Erişim Tarihi: 25.09.2018.

[13] Ptv-Group, www.ptvgroup.com., Erişim Tarihi: 25.11.2018.

[14] Djahel ve Murphy, J., A comparative study of vehicles' routing algorithms for route planning in smart cities in Vehicular Traffic Management for Smart Cities (VTM), Dublin, 2012.

[15] Wynita M., Rodrigo H., Emanuele C., Florian H., Kay M., Robert, N. S., A Large-Scale SUMO-Based Emulation Platform., IEEE Transactions on Intelligent Transportation Systems, 2015.

[16] Behrisch, M, Bieker, L., Erdmann, J., Krajzewicz, D., SUMO-Simulation of Urban MObility, SIMUL : The Third International Conference on Advances in System Simulation, Institute of Transportation Systems German Aerospace Center Rutherfordstr. 2, 12489 Berlin, Germany, 2011.

[17] Sumo-NETCONVERT, www.sumo.dlr.de/wiki/NETCONVERT., Erişim Tarihi: 29.11.2018.

[18] Taşıtların Tanımı, Taşıt Çeşitleri ve Rotalar, www.sumo.dlr.de/wiki/Definition_of_Vehicles,_Vehicle_Types,_and_Routes#CarFollowing_Models., Erişim Tarihi: 29.11.2018.

[19] Alexiadis, V., Jeannotte, K., Chandra, A., Traffic Analysis Toolbox Volume I: Traffic Analysis Tools Primer Oackland, CA: US., Department of Transportation, 2004.

[20] Olstam, J.J., A Model for Simulation and Generation of Surrounding Vehicles in Driving Simulators Norrkoping: UniTryck, 2005.

[21] DUAROUTER, www.sumo.dlr.de/wiki/DUAROUTER., Erişim Tarihi: 09.12.2018.

[22] Tomohisa, Y., Kiyoshi, I., Koichi, K. ve Hideyuki, N., Smoth Traffic Flow with a Cooperative Car Navigation System. Utrecht, Netherlands: AAMAS, 2005.

[23] Traffic stream models, www.civil.iitb.ac.in/~vmtom/1100_LnTse/503_lnTse/plain., Erişim Tarihi: 09.12.2018.

[24] Dijkstra, E. W., Numerische Mathematik 1, chapter A Note on Two Problems in Connexion with Graphs., Mathematisch Centrum, Amsterdam, pp. 269–271., 1959.

[25] Henzinger, M. , Krinninger, S., Nanongkai, D., A Subquadratic Time Algorithm for Decre-mental Single-Source Shortest Paths. SODA , pages 1053–1072, 2014.

[26] Bellman, R.E., On a Routing Problem., The Quarterly of Applied Mathematics, volume 16, pp. 87–90., 1958.

[27] Ford, L. R., Fulkerson, D. R., Flows in Networks. Princeton, NJ: Princeton University Press, 1962.

[28] Warshall, S., A theorem on Boolean matrices. Journal of the ACM, volume 9, no. 1,pp. 11–12., 1962.

[29] Floyd, R. W., Algorithm 97: Shortest Path. Communications of the ACM, volume 5, p. 345, 1956.

[30] Johnson, D. B., Efficient Algorithms For Shortest Paths in Sparse Networks., Journal of the Association for Computing Machinery, volume 24,: pp. 1–13., 1977.

[31] Cormen, T. H., Leiserson, C. E., Introduction to Algorithms. MIT Press and McGraw-Hill, second edition edition, 636–640 pp. , 2001.

[32] Frana, P., An Interview with Edsger W. Dijkstra. Communications of the ACM, pp. 41–47., 2010.

[33] Rishab, A., Rishabh , Y., Raghav, A., O.V., Gnana, S., Dijkstra's Algorithm for Shortest Path Identification in Reconfigurable Microgrid, Joutnal of Engineering and Applied Science 13(3)717-720, 2018.

[34] Mehlhorn, K., Sanders, P., Algorithms and Data Structures: The Basic Toolbox. Springer, 2008.

[35] Anbuselvi, R., Bhuvaneswaran, R.S., Simulation of path finding algorithm a Bird's Eye perspective, 2009.

[36] A* Algorithm, By Warren Russell, www.sfu.ca/~arashr/warren.pdf., Erişim Tarihi: 27.12.2018.

[37] Rouse, M., Flocking like it's 1999., www.whatis.techtarget. com/definition/heuristic., Erişim Tarihi: 24.10.2018.

[38] Yiqing, W., An Urban Mobility Overlay for Evaluating Cellular Driven Vehicle Teleoperation. Institutionen för informationsteknologi Department of Information Technology. IT 17 085 Examensarbete 30 hp, 2017.

[39] Cormen, T. H., Leiserson, C. E.,. Rivest, R. L. ve Stein, C., Section 24.3: Dijkstra's algorithm In Introduction to Algorithms, Second Edition, 2013.

[40] Cormen, T. H., Algorithms Unlocked, MIT Press, Cambridge, 2013.

[41] Dias, J., Abreu, P.H., Silva, D.S., Gustavo, F., Penousal, M., ve Ant´onio, L., Preparing Data for Urban Traffic Simulation using SUMO, P´olo II - Pinhal de Marrocos, 3030-290 Coimbra, Portugal

[42] Ünen, H.C., Orkut, M.Y., Onur, G., Özgür Harita : OpenStreetMap-TMMOB Coğrafi Bilgi Sistemleri Kongresi, 2013.

[43] OpenStreetMap, www.openstreetmap.org., Erişim Tarihi: 10.09.2018.

[44] Neis, P. ve Zipf, A., Analyzing the contributor activity of a volunteered geographic information project – the case of OpenStreetMap, ISPRS International Journal of Geo-Information, Vol. 1, No. 2, pp.146–165, 2012.

[45] Haklay, M., Weber, P., OpenStreetMap: user-generated street maps, Pervasive Computing, Vol. 7, No. 4, pp.12–18, IEEE, 2008.

[46] OpenStreetMap.org (2019) OpenStreetMapStatistics, www.openstreetmap.org/stats/data_stats.html., Erişim Tarihi: 07.01.2019.

[47] OpenStreetMap, www.sumo.dlr.de/wiki/Networks/Import/OpenStreetMap., Erişim Tarihi: 07.09.2018.

[48] POLYCONVERT, www.sumo.dlr.de/wiki/POLYCONVERT., Erişim Tarihi: 07.09.2018.

[49] Xml DOSYALARI, www.sumo.dlr.de/wiki/Tools/Xml., Erişim Tarihi: 07.10.2018.

[50] SUMO, www.sumo.dlr.de., Erişim Tarihi: 07.09.2018.

[51] Rastgele Trafik Belirtme, www.sumo.dlr.de/wiki/Tools/Trip., Erişim Tarihi: 09.09.2018.

[52] TrollTech.2002, Troll Tech Homepage, www.trolltech.com., Erişim Tarihi: 29.12.2018.

[53] OpenGL.org.2002, OpenGL.org Homepage, www.opengl.org., Erişim Tarihi: 30.12.2018.

[54] Kuyruk, www.sumo.dlr.de/wiki/Simulation/Output/QueueOutput., Erişim Tarihi: 20.12.2018.

ÖZGEÇMİŞ

Gulkaiyr Kalybek kyzy, 19.04.1992'de Kırgızistan'da doğdu. İlk, orta ve lise eğitimini Kırgızistan'da Narın bölgesinde tamamladı. 2010 yılında Kök-Car Lisesinden mezun oldu. 2010 yılında Kırgızistan başkenti Bişkek şehrinde Kırgızistan-Türkiye "Manas" Üniversitesi Uygulamali Matematik ve Enformatik Bölümü'ne başladı. 2015 yılında Kırgızistan-Türkiye "Manas" Üniversitesinden başarıyla mezun oldu. 2015 yılında Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Bölümü'nde yüksek lisans eğitimine başladı.