İNTERNET SERVİS SAĞLAYICILARININ MALİYET OPTİMİZASYONU

MEHMET EMİN KARAKAŞ

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ARALIK 2015

ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Osman EROĞUL
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Prof. Dr. Murat ALANYALI
Anabilim Dalı Başkanı (UHDE)

MEHMET EMİN KARAKAŞ tarafından hazırlanan İNTERNET SERVİS SAĞLAYICILARININ MALİYET OPTİMİZASYONU adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Yrd. Doç. Dr. Buğra ÇAŞKURLU
Tez Danışmanı

Tez Jüri Üyeleri

Başkan : Doç. Dr. Betül Atalay SATOĞLU _____

Üye : Yrd. Doç. Dr. Buğra ÇAŞKURLU _____

Üye : Prof. Dr. Süleyman Sadi SEFEROĞLU _____

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Mehmet Emin KARAKAŞ

Üniversitesi	**:**	**TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü	**:**	**Fen Bilimleri**
Anabilim Dalı	**:**	**Bilgisayar Mühendisliği**
Tez Danışmanı	**:**	**Yrd. Doç. Dr. Buğra ÇAŞKURLU**
Tez Türü ve Tarihi	**:**	**Yüksek Lisans Aralık 2015**

Mehmet Emin KARAKAŞ

İNTERNET SERVİS SAĞLAYICILARININ MALİYET OPTİMİZASYONU

ÖZET

Hızlı, güvenilir ve satın alınabilir internet çağımızın gerekliliklerindendir. Bu kapsamda Türkiye'deki internet servis sağlayıcı(İSS) firmaların ağ kurma verimlilikleri incelenmiştir. İSS'lerin kullanıcılara daha ucuz ve hızlı hizmet sağlamaları için maliyetlerinin nasıl düşürüleceği konusu araştırılmış ve belirleyici noktanın kazı maliyetleri olduğu tespit edilmiştir. İSS' lerin hizmet götürmek istedikleri yerler için en az kazı yaparak, ideal ağı oluşturmaları hedeflenmiştir. Bu kapsamda problem Steiner Ağacı Problemi(SAP) [1] olarak modellenmiş ve problemin çözümü için Prim'in yakınsama faktörü 2 olan algoritması [11] kullanılmıştır. Algoritmanın performansını test etmek için coğrafi data PostgreSQL [4] veritabanı yönetim sistemi ve Python [5] kullanılarak işlenebilir hale getirilmiştir. Çalışmamızda Türkiye'nin önde gelen İSS'lerinden birinden alınan gerçek veriler kullanılmıştır. İSS tarafından fiziksel olarak oluşturulmuş, kullanımda olan ağ ile geliştirmiş olduğumuz programın çıktıları karşılaştırılmıştır. Yapılan kazıların uzunluğunun yüzde kırk oranında azaldığı görülmüştür. ISS tarafından oluşturulan ağda 24.868 km kazı çalışması yapılmıştır, programın çıktısına göre ise 14.776 km kazı çalışması yapılmasının yeterli olacağı ortaya çıkmıştır.

Anahtar Kelimeler: İnternet Servis Sağlayıcı, Ağ, Steiner Ağaç Problemi, Prim'in MST Sezgisel Algoritması, CBS.

University	:	**TOBB University of Economics and Technology**
Institute	:	**Institute of Natural and Applied Sciences**
Science Programme	:	**Computer Engineering**
Supervisor	:	**Asst. Prof. Buğra ÇAŞKURLU**
Degree Awarded and Date	:	**M.Sc. December 2015**

Mehmet Emin KARAKAŞ

INTERNET SERVICE PROVIDERS COST OPTIMIZATION

ABSTRACT

Fast, reliable and affordable internet is a necessity of our age. In this thesis, the productivity of internet service provider (ISP) companies' in Turkey is examined. We studied how ISPs can lower the expenses to provide cheaper and faster service for their customers and our study revealed that excavation costs make up more that ninety percent of the total infrastructure costs. Thus, setting up the network so that the excavation costs are minimized would significantly reduce the infrastucture costs. The problem is modelled as the Steiner Tree Problem [1], which is known to be NP-hard. Since our graph is huge, using superpolynomial algorithms to solve the problem optimally would be computationally infeasible. Therefore; we used Prim's heuristic [11], which guarantees a solution within a factor of 2 of the optimum and runs rather fast. PostgreSQL [4] relational database management system and Python [5] programming language are the technologies we used in our implementation. In this work, real data obtained from one of the pioneer ISPs in Turkey is utilized. We have determined that 14.776 km long excavation is sufficient to serve all the customers of the ISP whose data we utilized. This is a significant improvement over their existing 24.868 km long network.

Keywords: Internet Service Provider, Network, Steiner Tree Problem, Prim's MST Heuristic Algorithm, GIS.

TEŞEKKÜR

Tez çalışmam sırasında bana olan her türlü desteği ve harcadığı tüm zamanlar için canım eşim Ayşen Biricik KARAKAŞ'a, isteğimi kırmayıp jürime katılmayı kabul eden Prof.Dr. Süleyman Sadi SEFEROĞLU'na ve hiç bir desteğini esirgemeyen tez danışmanım Yrd. Doç. Dr. Buğra ÇAŞKURLU'ya tüm samimiyetimle teşekkür ederim.

İÇİNDEKİLER

Sayfa

ŞEKİL LİSTESİ

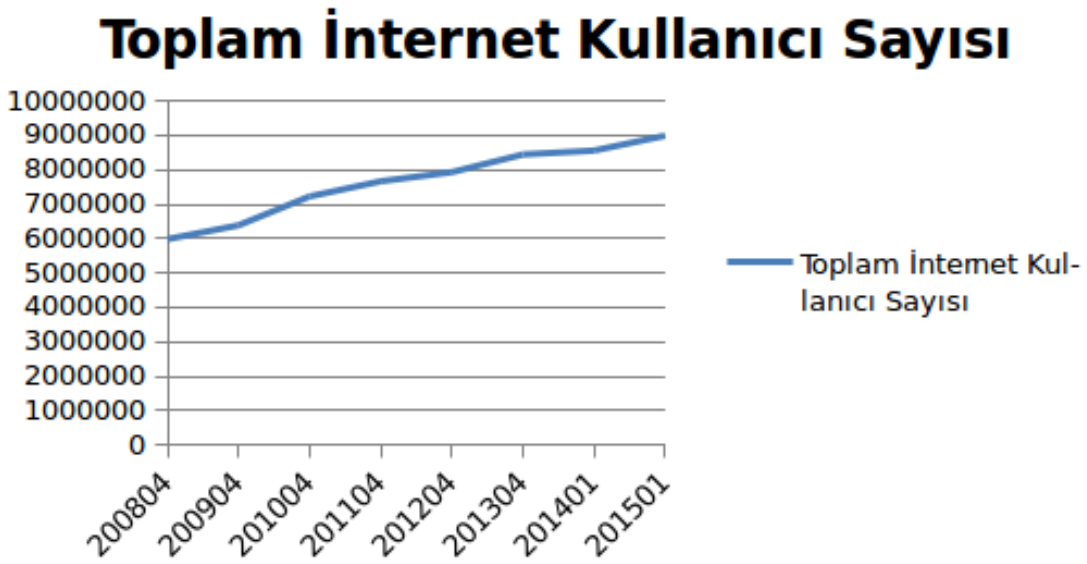
ÇİZELGE LİSTESİ

KISALTMALAR

Kısaltma	Açıklama
İSS - ISP	İnternet Servis Sağlayıcısı
SAP	Steiner Ağaç Problemi
BTK	Bilgi Teknolojileri ve İletişim Kurumu
CBS - GIS	Coğrafi Bilgi Sistemi
MST	Minumum Yayılan Ağaç
SAP	Steiner Ağaç Problemi
CSBM	Cadde, Sokak, Bulvar veya Meydan

1. GİRİŞ

İSS şirketleri belirli bir ücret karşılığında veya ücretsiz olarak internet ağına bağlanmanızı sağlayan şirketlerdir. Bu şirketler hizmet verdikleri tüm müşteri lokasyonlarını birbirine bağlayarak onların global ağa bağlanmalarını sağlamaktadırlar.



Şekil 1.1: Yıllara göre internet kullanıcı sayısı artışı

Şekil 1.1 BTK raporlarına [6] dayanarak Türkiye'deki mobil hariç internet kullanıcılarının sayısını göstermektedir. BTK raporları yılda dört kere çeyrekler şeklinde hazırlanan bir rapordur. Bu rapor açıklandığı çeyrek için internet kullanımı ve İSS şirketlerinin durumlarını gösterir.

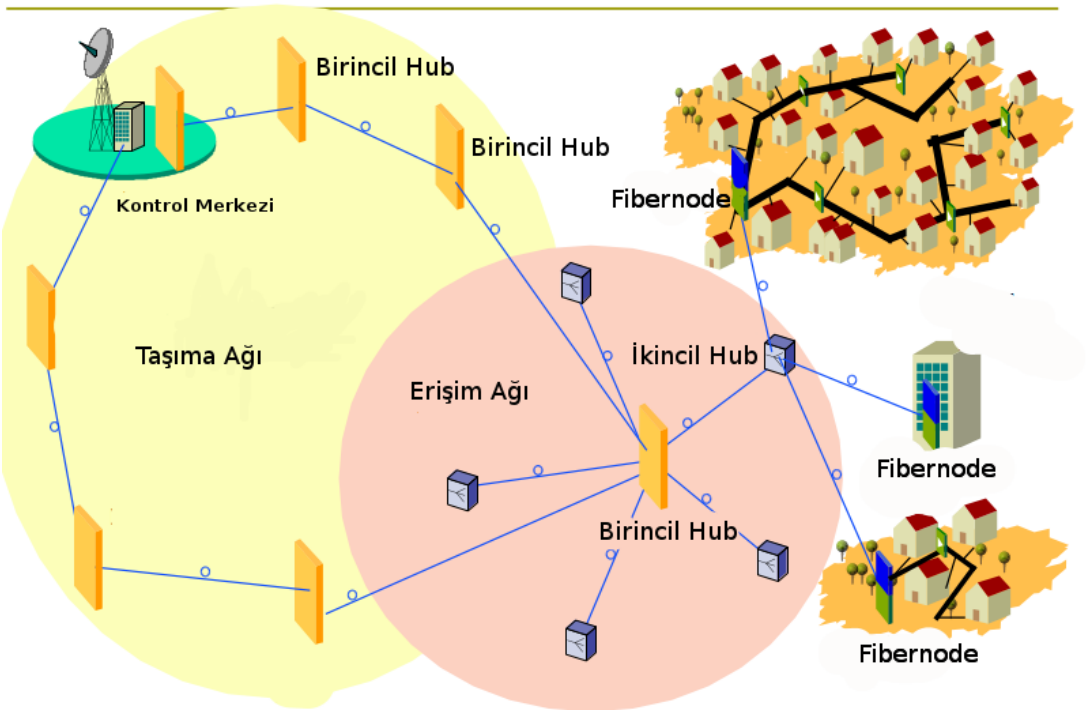
İSS sektöründe bir çok oyuncu bulunmaktadır. Bu oyuncuların pazardaki payları ağ genişlikleri ile doğru orantılıdır. Pazarın müşteri sayısındaki artış hızının azalmaya başladığı düşünülürse, ağınızın genişlemesinin pazar payınızı artıracağı bilinen bir gerçektir. Yani pazardaki müşteri sayısı artmıyorsa, müşteri sayınızın ve pazardaki yerinizi artırabilmenin yolu ağınızı genişleterek diğer firmaların pazar paylarını küçültmektir.

Tez çalışmamız sırasında Türkiye'nin önde gelen İSS şirketlerinden birinden alınan gerçek veriler kullanılmıştır. Verilerini incelemiş olduğumuz şirket hizmet vermek istediği şehirlerde yerel ağ altyapısına sahiptir. Sahip olduğu yerel ağların birbirine bağlanması içim ise başka bir özel şirketten hizmet alımı yapılmaktadır. Şirket ağ yapısı incelendiğinde büyük şehirler veya birbirine uzak ilçeleri bulunan illerde birden fazla yerel ağ kurulduğu, diğer şehirlerde ise tek bir yerel ağ kurulduğu görülmektedir. Örneğin; Yalova'da tek bir yerel ağ bulunurken, İstanbul ilinde altı adet yerel ağ bulunmaktadır. İncelemiş olduğumuz şirket Türkiye genelinde yirmi dört ilde hizmet vermektedir. Her bir il kendi başına tek bir yerel ağ oluşturacak şekilde tasarlanmıştır ve bu yerel ağ büyükşehirlerde veya uzak ilçeleri olan illerde birden fazla yerel ağın birleşmesiyle oluşmuştur. Şirket Türkiye genelinde toplam 523.629 adet binaya hizmet götürmektedir. Her bir yerel ağ içerisinde kontrol merkezi denilen yapıdan yayın başlar ve yine kontrol merkezinde sona erer. Her bir yerel ağ içerisinde Metro Ethernet ile iletim sağlanmaktadır. Kontrol merkezinden çıkan yayın fiber optik kablolar aracılığı ile yayının dağıtımının yapıldığı düğümlere iletilir. Buradan sonra fiberoptik kablolar veya koaksiyel kablolar aracılığıyla yayının binalara dağıtımı gerçekleştirilir. Yerel ağların birbirine bağlanması başka bir şirket tarafından sağlanan dark fiber ağlarla sağlanmaktadır.

Şirketin ağını genişletmesi daha çok aboneye hizmetlerini ulaştırması anlamına gelmektedir. Şirketin ağını genişletirken maliyetlerini asgari düzeye indirmesi müşterilerine daha rekabetçi bir fiyattan hizmet vermesini sağlayacaktır. Şebeke genişletme işlemi iki şekilde yapılabilir. Bunlardan ilki mevcut bir yerel ağa ekleme yaparak genişletmektir. İkinci seçenek ise hizmet verilmeyen bir şehre yeni bir yerel ağ kurmaktır.

Tez çalışması sırasında datalarını incelediğimiz şirketin ağı bizim algoritmamız tarafından tekrar oluşturulmuştur. Ağ İSS şirketinin hizmet vermiş olduğu binaların tamamını kapsayacak şekilde inşa edilmiştir.

Şekil 1.2'deki Kontrol Merkezi tüm yayının başlangıç ve bitiş noktasıdır. Kullanıcıların internete çıkış noktası burasıdır. Kontrol merkezi global ağlara bağlanma noktasıdır. Buradan çıkan yayın Taşıma ağı adı verilen katmanla birlikte birincil hublara taşınmaktadır. Birincil hublar ikincil hublara verileri



Şekil 1.2: Lokal ağın genel görünüşü

taşımaktadır. İkincil hublardan veriler fibernode bölgelerine taşınmaktadır. Fiber node bölgeleri kendi içerisinde verilerin dağıtımını yapmaktadır. Bu çalışmanın amacı fibernode bölgelerindeki hizmet verilecek bütün binalara erişen en ucuz maliyetli (en az kazı gerektiren) ağ yapısının belirlenmesidir. Çalışmamızın fibernode bölgesindeki ağ yapısını baz almasının sebebi kazı maliyetlerinin çok büyük bir kısmının bu bölgelerdeki kazı çalışmalarına ait olmasıdır.

2. PROBLEM TANIMI

2.1 İSS'lerin Ağ Kurma Maliyetleri

2.1.1 Maliyet kalemleri

İSS'lerin altyapı kurma maliyetleri kazı maliyeti, kurum ödemeleri, pasif eleman maliyetleri ve aktif cihaz maliyetleri olmak üzere dört ana kalemden oluşmaktadır. Maliyet kalemlerine ilişkin aşağıda verilen bilgiler bir ISS'den alınmış olup Ankara ili için 2015 yılı gerçek verileridir.

2.1.1.1 Kazı Maliyeti

Fiziksel olarak yapılan kazı işleminin maliyetidir. Bu maliyet kalemi kazı için gerekli olan araçlar, asfaltlama ve sökme çalışmalarının toplam maliyetidir.

2.1.1.2 Kurum Ödemeleri

Fiberoptik hat / kablo döşemek için kamu kurum ve kuruluşlarına ödenen ücretler toplamıdır. Bu maliyet kalemi her il için ayrı olarak hesaplanmaktadır. Kazı çalışması yapılacak olan ilin bağlı bulunduğu belediye ruhsat veya diğer gider işlemleri için farklı fiyat politikaları izlemektedir. Çalışmamızda Ankara iline ait ücret ortalamalarına göre hesaplama yapılmıştır.

2.1.1.3 Pasif Eleman Maliyetleri

Fiberoptik hat döşemek için kullanılan pasif elemanların toplam maliyetidir. Bu maliyet kalemi fiberoptik kablo, kullanılan HDPE (fiberoptik kabloların içerisine döşenen poliüretan borular) gibi cihaz kategorisine girmeyen malzemelerin toplam maliyetidir.

2.1.1.4 Aktif Cihaz Maliyetleri

Ağ inşası sırasında kullanılan aktif cihazların maliyetidir. Erişim ve santral tarafında kullanılan anahtarlama ekipmanları, CMTS gibi internet erişimini yönlendiren cihazların toplam maliyetidir. Şebekede kullanılan tüm elektronik cihazlar bu kaleme girmektedir. Bu kalemin giderleri hizmet sayısı ile doğru orantılıdır. Yeni kazı çalışması yapılmadan hizmet verdiğiniz kişi sayısı arttırıldığı zaman bu maliyet kaleminde artış görülmektedir.

2.1.2 Maliyet Fonksiyonu

2.1.2.1 Kısaltmalar

- M: Verilecek hizmet sayısı toplamıdır.
- F: Toplam maliyeti ifade etmektedir. Maliyet fonksiyonun sonucudur.
- KM: Birinci ana maliyet kalemi olan kazı maliyeti toplamını ifade eder.
- KU: Toplam kazı uzunluğudur. Metre cinsinden ifade edilmektedir.
- KO: İkinci ana maliyet kalemi olan kurum ödemeleri toplamını gösterir.
- PE: Üçüncü ana maliyet kalemi olan ağda kullanılan pasif elemanların toplam maliyetini gösterir.
- AC: Dördüncü ana maliyet kalemi olan ağda kullanılan aktif cihazların toplam maliyetini gösterir.

2.1.2.2 Maliyet Fonksiyonu

Aktif Cihaz Maliyeti

$$AC = (M/1000) * 35000 \tag{2.1}$$

Her 1.000 adet hizmet için ortalama olarak 35.000TL aktif cihaz maliyeti bulunmaktadır. Toplam hizmet sayısının bine bölümünün 35.000 ile çarpımı tüm müşterilere hizmet vermek için gerekli aktif cihaz yatırım miktarını vermektedir.

Pasif Elemanların Maliyeti

$$PE = KU * 10 \tag{2.2}$$

Ağ kurma işlemi sırasında gerekli olan pasif cihazların toplam maliyetidir. Her bir metrelik kazı başına 10TL'lik pasif eleman maliyeti vardır.

Kurum Ödemeleri Maliyeti

$$KO = KU * 175 \tag{2.3}$$

Ağ kurma işlemi sırasında belediye ve devlet kurumlarına ödenmesi gereken maliyeti ifade etmektedir. Ankara ilinde her bir metrelik kazı için ortalama kurum ödemesi 175TL'dir. Bu yüzden toplam kurum ödemeleri kazı uzunluğunun 175 katı olarak hesaplanmıştır.

Kazı Maliyeti

$$KM = KU * 20 \tag{2.4}$$

Şirket verileri bir metrelik kazının ortalama maliyetinin 20TL olduğunu göstermektedir. Toplam kazı maliyeti toplam kazı uzunluğunun 20 katı olarak hesaplanmıştır.

Toplam Maliyet

$$F = AC + PE + KO + KM \tag{2.5}$$

Toplam ağ kurulum maliyeti yukarıda belirtilen dört ana maliyet kalemi olan kazı maliyetleri, kurum ödemeleri, pasif cihaz maliyetleri ve aktif cihaz maliyetlerinin toplamı olarak hesaplanmıştır.

Maliyet fonksiyonunun alt bileşenlerinden görülebileceği üzere ağ kurum maliyetini artıran ana etmen toplam kazı uzunluğudur. Dört ana maliyet kaleminden kazı maliyetleri, kurum ödemeleri ve pasif eleman maliyetleri kazı uzunluğu ile doğru orantılıdır. Aktif cihaz maliyetleri ise ISS'nin toplam müşteri sayısı ile doğru orantılıdır.

Bu tezde İSS'lerin maliyetlerinin toplam kazı uzunluğunu kısaltmak suretiyle ne kadar azaltılabileceği sorusu cevaplanmaya çalışılmıştır. Bu kapsamda, ISS'nin bütün müşterilerine hizmet verecek en kısa ağ yapısı eldeki kapsamlı coğrafi data ve müşteri datası kullanarak hesaplanmaya çalışılmıştır. Türkiye haritası üzerindeki her bir sokak başının ayrı bir düğümle belirtildiği ve sokakların kenarları oluşturduğu bir yönsüz çizge PostgreSQL veritabanı üzerinde oluşturulmuştur. Böylelikle çözüm aradığımız İSS'nin bütün müşterilerine hizmet verecek en kısa ağı bulma problemi, yukarıda belirtilen şekilde oluşturulan çizgede Steiner Ağacı probleminin çözümüne dönüştürülmüştür. Steiner Ağacı problemi NP-zor bir problem olduğundan bu problemi çözecek polinom zamanlı bir algoritma yoktur. Bu çalışmada problemin çözümü için Prim'in MST sezgisel algoritması[2] kullanılmıştır. Prim'in MST sezgisel algoritması hem hızlı çalışmakta, hem bütün müşterileri birbirine bağlayan bir ağ oluşturmakta, hem de oluşturduğu ağın toplam kazı uzunluğunun optimal ağın toplam kazı uzunluğunun iki katını geçmemesini garanti etmektedir. Algoritmayı çok büyük bir ağ girdisiyle koşturduğumuz için ileriki bölümlerde detaylı bir biçimde anlatılan bir takım performans artırıcı iyileştirmeler yapılmıştır. Algoritma gerçek veriler üzerinde koşturularak elde edilen sonuç İSS'nin mevcut ağı ile karşılaştırılmış ve yüzde kırk oranında azalma olduğu görülmüştür.

Bu konuda alan çalışması yapıldığı zaman örnek oluşturabilecek herhangi bir çalışmaya rastlanmamıştır. Bu yüzden yapılan çalışmaların tamamı bu tez çalışması kapsamında gerçekleştirilmiştir.

2.2 Problemin Tanımı ve Modelleme

2.2.1 Steiner Ağacı Problemi

Tanım ve Bilgi

Steiner ağacı problemi(SAP)[1] genellikle ağ dizaynı ve kablolama problemleri için kullanılmaktadır. Bu problemin iki girdisi vardır; bir yönsüz çizge ve bu yönsüz çizgenin düğümlerinin bir altkümesi. Problemin çıktısı ise çizgedeki kenarları kullanarak girdi olarak verilen düğümleri birbirine bağlayan en düşük maliyetli ağaçtır. Problem, kablolama, su borularının döşenmesi, ısıtma veya doğalgaz sistemlerinin döşenmesi gibi birçok gerçek hayat probleminin modellenmesinde kullanılmaktadır. Klasik bir kombinatoryal problem olan SAP problemi NP-zor bir problemdir ve dolayısıyla bu problemi optimal olarak çözecek polinom zamanlı bir algoritma yoktur.

2.3 Kullanılan Algoritmalar

2.3.1 Prim'in Minimum Yayılan Ağaç Algoritması

Minimum Yayılan Ağaç problemi ağ optimizasyonunda en çok kullanılan problemlerden biri olup Steiner Ağacı probleminin özel bir durumudur. Problemin girdisi $G = (V, E, W)$ şeklinde göstereceğimiz kenarları ağırlıklı bir yönsüz çizgedir. $V = \{v_1, v_2, \ldots, v_n\}$ G çizgesinin düğüm kümesini, $E = \{e_1, e_2, \ldots, e_m\}$ ise G çizgesinin kenar kümesini göstermektedir. $W = \{w_1, w_2, \ldots, w_m\}$ kümesi ise G çizgesinin kenar uzunluklarını göstermektedir. Minimum Yayılan Ağaç probleminde amaç çizgedeki bütün düğümleri birbirine en kısa yoldan bağlayan bir ağaç oluşturmaktır.

Steiner Ağacı probleminde girdi olarak, kenarları ağırlıklı yönsüz bir çizge ve çizgenin düğüm kümesinin bir altkümesi verilmektedir. Bu altkümedeki düğümlerin birbirine en kısa yoldan bağlanması istenmektedir. Steiner Ağacı

problemini optimal olarak çözecek bir algoritma girdi olarak verilen herhangi bir düğüm altkümesini birbirine en kısa yoldan bağlayacak ağacı bulduğundan, girdi olarak kenarları ağırlıklı bir yönsüz çizgenin yanı sıra bu yönsüz çizgenin bütün düğümlerinden oluşan düğüm altkümesi verildiğinde o kenarları ağırlıklı yönsüz çizge üzerinde Minimum Yayılan Ağaç problemini çözmüş olacaktır. Bu indirgemede gösterildiği üzere Minimum Yayılan Ağaç problemi Steiner Ağacı probleminin özel bir durumudur. Steiner Ağacı problemi NP-zor bir problem olmasına karşın, Minimum Yayılan Ağaç problemi polinom zamanda çözülebilen bir problemdir. Minimum Yayılan Ağaç probleminin çözümünde en çok kullanılan iki algoritma Prim'in algoritması ve Kruskal'ın algoritmasıdır. İki algoritma da standart teknikler kullanılarak $O(m \cdot \lg n)$ zamanda çalıştırılabilir[12]. Minimum Yayılan Ağaç problemini optimal olarak çözen Prim'in algoritması, Steiner Ağacı problemi için ise bir yakınsama algoritması olup optimalin en fazla iki katı uzunluğunda bir ağaç oluşturmayı garanti etmektedir. Prim'in algoritmasının sözdekodu ve algoritmayı açıklayan bir örnek aşağıda verilmiştir.

2.3.1.1 Sözdekod

Algorithm 1 PRIM MST(G=(V,E,W))

Girdi: Kenarları ağırlıklı yönsüz bir çizge olan G

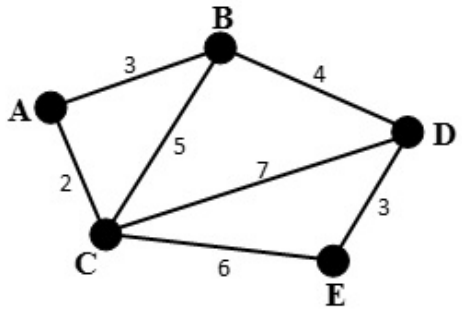
Çıktı: Kenarları ağırlıklı bir ağaç olan T

Adım 1: G çizgesine ait rastgele olarak seçilen bir düğüm T ağacına başlangıç düğümü olarak eklenir.

Adım 2: Her bir itarasyonda T çözüm ağacımıza ekli olan düğümlere komşu olan kenarlardan çözüm ağacına eklenmemiş düğüme giden en kısa olan kenar seçilir ve T ağacına eklenir.

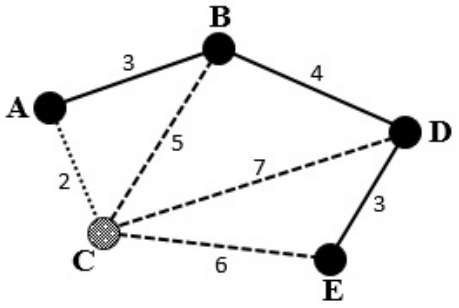
Adım 3: Adım 2 G çizgesindeki tüm düğümler çözüm çizgesinden oluşan ağaca bağlanıncaya kadar tekrar edilir.

2.3.1.2 Örnek



Şekil 2.1: Prim'in MST Algoritması Başlangıç Ağacı

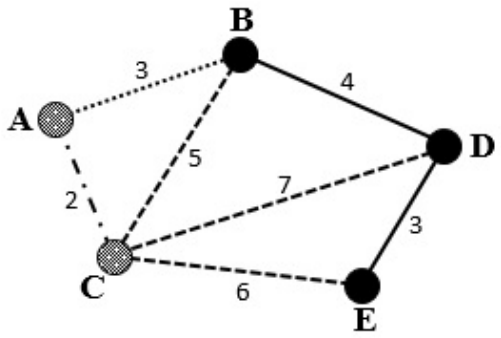
Şekil 2.1 girdimiz olan yönsüz ve negatif ağırlıklı kenarı bulunmayan $G = (V, E, W)$ çizgesini göstermektedir. G çizgesi üzerinde $V = \{A, B, C, D, E\}$ düğüm kümesini ve $E = \{(A, B), (A, C), (C, B), (B, D), (C, E), (D, E), (C, D)\}$ bu düğümleri birbirine bağlayan kenar kümesini göstermektedir. $W = \{3, 2, 5, 4, 6, 3, 7\}$ kümesi ise G çizgesinin kenar ağırlıklarını göstermektedir. $T = (V_2, E_2, W_2)$ ise çözüm ağacımızı göstermektedir ve başlangıçta içerisinde herhangi bir kenar veya düğüm bulunmamaktadır. V düğüm kümesinin içerisinden başlangıç düğümü olarak C rastgele seçilmiştir ve T çözüm ağacına eklenmiştir.



Şekil 2.2: Prim'in MST Algoritması 1. Adım

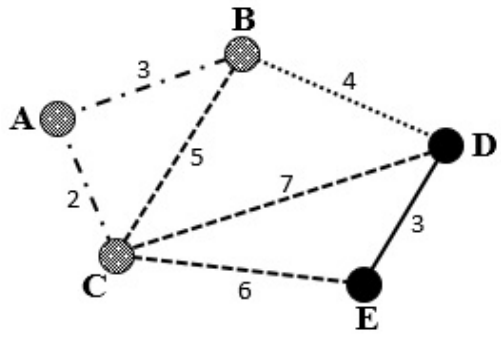
C düğümünün komşu kenar kümesi belirlenir ve ağırlıklarına göre karşılaştırma yapılır. Komşu kenar kümesi $\{(A, C), (C, B), (C, D), (C, E)\}$ ve kenar kümesinin ağırlıkları $\{2, 5, 7, 6\}$ birimdir. Şekil 2.2'deki kesikli çizgiler tüm komşulara giden

kenarları, sık kesikli çizgi ise ağaca eklenen kenarı ifade etmektedir. Kenar kümesinden en küçük ağırlığa sahip olan (A, C) kenarı seçilir ve çözüm ağacımıza eklenir. Yeni çözüm çizgemiz $T = (\{(A, C)\}, \{C, A\}, \{2\})$ şeklindedir.



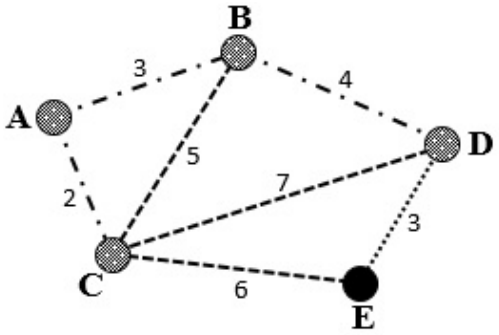
Şekil 2.3: Prim'in MST Algoritması 2. Adım

Çözüm ağacımızda bulunan $\{A, C\}$ düğümlerine komşu kenarlar belirlenir. (A, B) kenarının ağırlığı 3 birim, (C, B) kenarının ağırlığı 5 birim, (C, D) kenarının ağırlığı 7 birim ve (C, E) kenarının ağırlığı 6 birimdir. Bu kenarlar arasından ağırlığı en az olan (A, B) kenarı seçilir ve çözüm ağacımıza eklenir. Yeni çözüm ağacımız
$T = (\{C, A, B\}, \{(A, C), (A, B)\}, \{2, 3\})$ şeklindedir.



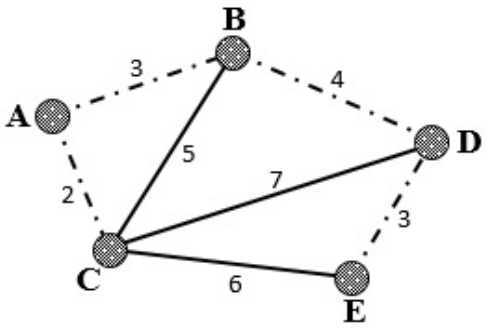
Şekil 2.4: Prim'in MST Algoritması 3. Adım

Algoritma çalışmaya devam edince $\{C, A, B\}$ düğümlerine komşu olan en az ağırlığa sahip (B, D) kenarı seçilir ve çözüm ağacımıza eklenir. Yeni çözüm çizgemiz $T = (\{C, A, B, D\}, \{(A, C), (A, B), (B, D)\}, \{2, 3, 4\})$ şeklindedir.



Şekil 2.5: Prim'in MST Algoritması 4. Adım

$\{A, B, C, D\}$ düğümlerine komşu kenarlar içerisindeki en az ağırlığa sahip (D, E) kenarı seçilir ve çözüm ağacımıza eklenir. Yeni çözüm ağacımız
$G_2 = (\{C, A, B, D, E\}, \{(A, C), (A, B), (B, D), (D, E)\}, \{2, 3, 4, 3\})$ şeklindedir.



Şekil 2.6: Prim'in MST Algoritması Sonucu

Algoritmanın çalışması tamamlandığında Şekil 2.6'da görülen ağaç ortaya çıkmaktadır. Şekil 2.6 üzerindeki noktalı kesikli çizgilerle belirtilen ağaç tüm düğümleri kapsayan en kısa ağaçtır. Böylelikle girdi olarak verilen G çizgesi üzerindeki düğümleri birleştiren en kısa ağaç oluşturulmuştur.

2.3.2 Dijkstra'nın En Kısa Yol Algoritması

Coğrafi veri üzerinde tanımı 2.3.4.1 de verilen iki nokta arasındaki uzunluğu bulma işlemi için kullanılmıştır. CBS sistemleri üzerinde iki nokta arasındaki

uzunluğun bulunabilmesi için tanımı 2.3.4.1'de verilen yönlendirme işleminin yapılması gerekmektedir. CBS sistemleri üzerinde iki nokta arasında birden fazla yol bulunabilir. Bu yollar arasındaki en kısa uzunluğa sahip yolun bulunması işlemi Dijsktranın en kısa yol algoritması kullanılarak bulunmaktadır. Ekler kısmında bulunan Bölüm B.1'de algoritmanın çalışması ayrıntılı olarak anlatılmıştır.

2.3.3 SAP Probleminin Yakınsama Algoritması ile Çözümü (Prim'in Minimum Yayılan Ağaç Sezgisel Algoritması)

2.3.3.1 Problemin Formülasyonu

Problemin girdisi $G = (V, E, W)$ şeklinde göstereceğimiz kenarları ağırlıklı bir yönsüz ve negatif ağırlıklı kenarı bulunmayan bir çizgedir. $V = \{v_1, v_2, \ldots, v_n\}$ G çizgesinin düğüm kümesini, $E = \{e_1, e_2, \ldots, e_m\}$ ise G çizgesinin kenar kümesini göstermektedir. $W = \{w_1, w_2, \ldots, w_m\}$ kümesi ise G çizgesinin kenar uzunluklarını göstermektedir. $L \subset V$; L düğüm kümesi V düğüm kümesinin bir alt kümesidir ve L düğüm kümesindeki düğümler terminal düğüm olarak adlandırılır. $T \subset G$; T ağacı ise G çizgesinin bir alt kümesidir ve sonuç kümesi olarak adlandırılır. Amaç L düğüm kümesindeki tüm terminal düğümleri kapsayacak şekilde en kısa uzunluğa sahip T ağacını oluşturmaktır.

2.3.4 Prim'in Minimum Yayılan Ağaç Sezgisel Algoritmasının İyileştirilmiş Hali

Prim'in MST algoritması sabit ağırlıklı kenar uzunlukları olan çizgeler üzerinde hızlı çalışmaktadır. Fakat bizim problemimizde coğrafi data üzerinde çalışmamızdan dolayı kenar uzunlukları algoritmanın işletilmesi sırasında hesaplanmaktadır. Algoritmanın işletilmesi sırasında Steiner Ağacı yardımcı düğüm sayımız gittikçe artmaktadır. Bundan dolayı hesaplama yapılan düğüm sayısı artmaktadır. Düğüm sayısının devamlı olarak artmasından kaynaklı olarak Prim'in MST

sezgisel algoritması çalışma zamanı bakımından yavaş kalmıştır. Tüm bu durumlardan dolayı algoritmanın çalışma zamanını iyileştirici yeni özellikler eklenmiştir.

2.3.4.1 Tanımlar

CBS sistemlerinde kullanılan ve algoritmamızın içerisinde yer alan bazı tanımlar ve bu tanımların açıklamaları aşağıda yer almaktadır.

Nokta

CBS sistemlerinde bulunan bir veri tipidir. İki boyutlu düzlem üzerinde x ve y koordinatları bulunan harita üzerinde tanımlanabilen lokasyon bilgisidir.

Çizgi

CBS sistemlerinde bulunan bir veri tipidir. Birbirleri ile sıralı şekilde birleştirilmiş olan noktalar grubudur. En az iki noktanın birleşimi ile oluşmaktadır.

Alan

CBS sistemlerinde bulunan bir veri tipidir. Harita üzerinde belirli bir alanı ifade etmektedir. Doğrusal bir veri içermez. İçerisinde bir çok nokta ve çizgi bulunur. Örnek olarak; mahalle sınırları,ilçe sınırları veya il sınırları verilebilir.

Kuş bakışı uzunluk

Harita üzerinde nokta veya çizginin diğer nokta veya çizgiye olan uzaklığıdır. Buradaki uzunluk yeryüzündeki engellerden etkilenmeyen havadan uzunluğudur.

Yönlendirme

Harita verisi üzerinde iki nokta arasındaki yolun cadde, sokak, bulvar veya meydan (CSBM) düzeyinde bulunması işlemidir. Navigasyon cihazlarının yaptığı işlemdir. İki nokta arasında CSBM'den geçen en kısa yolun tanımlanması işlemidir. Yani fiziksel olarak gidilebilecek bir yolun bulunması işlemidir.

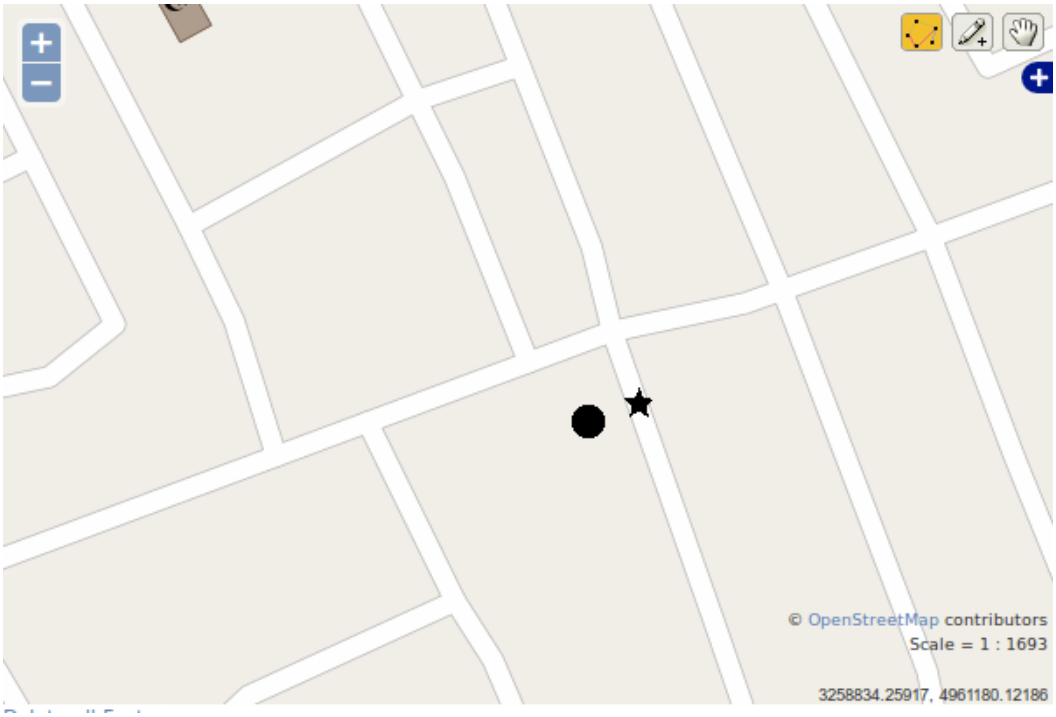
2.3.4.2 Algoritmanın Tanımı

MST problemi Steiner Ağaç Probleminin özelleşmiş halidir. SAP probleminin çözümü polinom zamanda mümkün değildir. Bu yüzden Prim'in MST sezgisel algoritması yakınsama algoritması olarak kullanılmıştır. Problemin çözümü için geliştirmiş olduğumuz algoritmanın temelini de Prim'in MST sezgisel algoritması oluşturmaktadır.

Oluşturmuş olduğumuz CBS veritabanında hizmet verilen iller için 2.072.539 adet cadde, sokak, bulvar veya meydan (CSBM) bulunmaktadır. Bu CSBM datası üzerinde çizge algoritmalarını çalıştırabilmemiz için bu datanın bir çizgeye dönüştürülmesi gerekmektedir. CBS sisteminde bir çizgi ile gösterilen bir CSBM'ın üzerinde bir veya birden fazla yol ayrımı olabilir. Bu yüzden oluşturduğumuz çizgede CSBM datasındaki her bir çizgiye karşılık gelen birden fazla düğüm veya kenar olabilir. Bu çizgilerin birbirleriyle olan ilişkileri ayrı bir tablo üzerinde tutulmaktadır. Bu dönüştürme işlemi sonrasında 1.477.960 adet nokta elde ettik. Bu noktalar algoritmamızı koşturacağımız çizge üzerindeki düğümleri oluşturmaktadır. Bu 1.477.960 adet düğümü birbirine bağlayan 2.072.400 adet kenar bulunmaktadır.

Hizmet verilmek istenen noktalar 523.629 adet binadan oluşmaktadır. Fiziksel olarak kurulacak ağın kazı çalışmaları CSBM üzerinden yapılmak zorundadır. Çünkü fiziksel olarak yapılacak kazılar devlete ait olan CSBM alanlarının içerisinden geçer. Bu nedenle hizmet verilecek binaları yönlendirme işlemine tabi tutabilmek için noktaların bağlı bulunduğu CSBM'nin bulunması gerekmektedir. Bu yüzden hizmet verilen binanın üzerinde bulunduğu CSBM üzerindeki en yakın nokta bina ile eşleştirilir. Bu indirgeme işlemi yapıldıktan sonra algoritmanın

birbirine bağlaması gereken 332.424 adet terminal düğüm bulunmaktadır. Şekil 2.7'de yuvarlak siyah nokta binanın konumunu göstermektedir. Bu noktayı CSBM düzeyine indirgediğimiz zaman Şekil 2.7'deki siyah yıldızla gösterilen nokta elde edilir. Bu işlemden sonra algoritma yıldız ile gösterilen nokta ile hesaplama işlemini gerçekleştirmektedir.



Şekil 2.7: CSBM düzeyine indirgeme işlemini gösterir harita

Üzerinde çalışmış olduğumuz problemi şu şekilde özetleyebiliriz. Bize içerisinde 1.477.960 adet düğüm bulunan ve bu düğümleri birbirine bağlayan 2.072.400 adet kenardan oluşan bir çizge verilmiştir. Verilen çizge yönsüz ve kenar ağırlıkları bulunan bir çizgedir. Bu çizgenin düğümlerinin bir alt kümesi olan 332.424 adet terminal düğüm bulunmaktadır. Bu terminal düğüm kümesinin verilen kenar kümesi kullanılarak en kısa uzunluğa sahip olacak şekilde birleştirilmesi ve çözüm ağacının oluşturulması gerekmektedir.

Prim'in MST sezgisel algoritması tek bir noktadan başlayarak büyüyen bir ağaç oluşturmaktadır. Algoritmanın her itarasyonunda bu ağacın içerisinde olmayan her terminal düğümün ağaca olan en kısa mesafesi en kısa olan

düğümün en kısa yoldan ağaca bağlanması gerekmektedir. Bizim problemimizde kenar ve terminal düğüm sayısının çok fazla olmasından dolayı algoritmayı hızlandırıcı iyileştirmeler yapılması gereksinimi duyulmuştur. Prim'in MST sezgisel algoritması ile geliştirmiş olduğumuz iyileştirilmiş MST algoritmasının çalışma zamanı karşılaştırmaları değerlendirme bölümünde sunulmuştur.

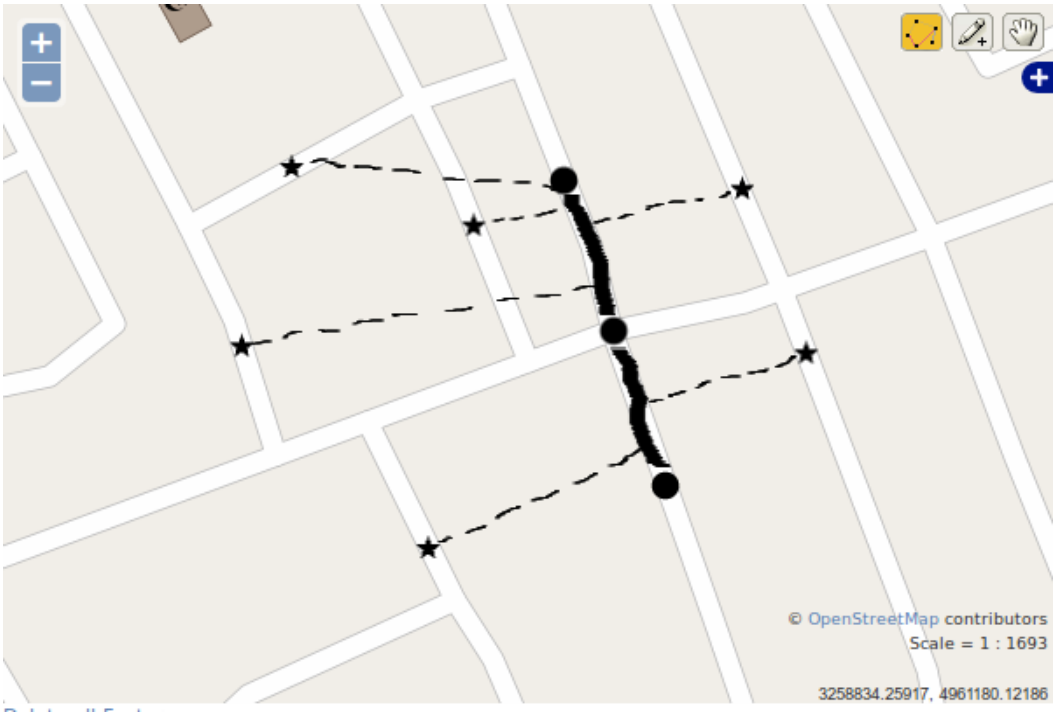
Algoritmanın işletilmesi sırasında iki düğüm arasındaki kenarın uzunluğu harita verisi üzerinde yönlendirme işlemi yapılarak bulunmaktadır. Bunun nedeni harita verisi üzerinde iki düğüm arasında doğrusal bir uzunluk bulunmamaktadır. Bir düğümden diğerine ulaşabilmek için normal hayattaki gibi CSBM'yi takip ederek gidilmesi gerekmektedir. Bir düğümden diğerine çok fazla sayıda yol bulunabilir. Harita üzerinde yönlendirme işlemi yaparak fiziksel olarak takip edebilecek bir yol bulunur. Bu yönlendirme işleminin hesaplanması sırasında Dijsktra'nın en kısa yol algoritması kullanılmaktadır.

Prim'in MST sezgisel algoritmasının işletilmesi sırasında çözüm ağacında bulunan her bir düğümün çözüm ağacına eklenmemiş olan düğümlere olan uzaklığının yönlendirme yapılarak hesaplanıp mesafesi en kısa olan düğümün çözüm ağacına eklenmesi gerekmektedir. Bu işlemleri gerçekleştirmek için CBS sisteminin bize sağlamış olduğu bazı fonksiyonlardan yararlanılmıştır.

Üzerinde çalıştığımız çizge çok fazla sayıda düğüm ve kenar içerdiği için Prim'in MST algoritmasının işlem mantığı olan çözüm çizgesinde bulunan düğümlere en yakın olan düğümün bulunması işlemin kısıtlanması gerekmektedir. Bunun için çözüm çizgemizde bulunan düğümleri CBS sisteminin özelliklerini kullanarak birleşik bir çizgi gurubu haline getiriyoruz ve bu çizgi grubuna kuş bakışı uzunluğu en az olan on adet çözüm kümesine eklenmemiş terminal düğümü buluyoruz. Beklentimiz yönlendirme işlemi yapıldığı zaman eklenecek olan terminal düğümün bu on noktanın içerisinde olmasıdır. Böylelikle en kısa uzunluğa sahip kenarı bulabilmek için yapacağımız karşılaştırma sayısını milyonlardan ona indirmiş oluyoruz. Bu işlem algoritmamızın çalışma zamanında ciddi bir azalma sağlamıştır.

Bir örnekle açıklayacak olursak; Şekil 2.8' de harita üzerinde üç adet çözüm ağacımıza eklenmiş terminal düğüm ve altı adet çözüm kümemize eklenmemiş

terminal düğüm görmekteyiz. Siyah noktalar çözüm kümemize eklenmiş düğümleri, yıldızlar ise çözüm kümesine eklenmemiş düğümleri göstermektedir. Çözüm ağacımıza eklenmiş üç adet düğümü bir çizgiyle birbirine bağlıyoruz. Şekil 2.8'de görülen düz çizgi bu çizgiyi ifade etmektedir. Çözüm ağacına eklenmemiş düğümler ise harita üzerinde yıldız şeklinde ifade edilmiştir ve kesikli çizgiler bu eklenmemiş terminal düğümlerin kuş bakışı uzunluklarını göstermektedir. On adet düğüm seçilirken bu kesikli çizgilerden en kısa olan on adet çizgiye ait düğüm seçilmektedir.



Şekil 2.8: Algoritmanın çözüm kümesine eklenecek düğümlerin seçilme işlemi

İkinci adım olarak seçmiş olduğumuz on adet çözüm kümemize eklenmemiş olan terminal düğüme kuş bakışı uzunluğu en kısa olan beş adet çözüm ağacına eklenmiş düğüm bulunur. Böylelikle yapmış olduğumuz bu sadeleştirme işlemi sayesinde binlerce karşılaştırma yerine belirlenen on düğümün her biri için sadece beş adet düğüm karşılaştırması yapılır.

Üçüncü adımda seçmiş olduğumuz on adet eklenmemiş terminal düğüm ve beş adet eklenmiş düğüm için teker teker Djisktra'nın en kısa yol algoritmasını

kullanarak yönlendirme yapıyoruz. Yönlendirme sırasında bir terminal düğümden diğer terminal düğüme ulaşabilmek için çok fazla sayıda yol olabilir. Yönlendirmenin hesaplanması tüm Türkiye düşünüldüğü zaman çok fazla zaman almaktadır. Sonuç olarak algoritmanın çalışma zamanı ne kadar kısa ise gerçek hayatta kullanılma olasılığı o kadar artmaktadır. Burada yönlendirme işlemini yaparken hesaplama yapmak istediğimiz her iki noktayı kapsayacak şekilde mahalle alanlarını birleştirerek yeni bir rota hesaplama kümesi oluşturuyoruz. Bu işlem sayesinde tüm Türkiye üzerinden yapılması gereken yönlendirme işlemi, hesaplama yapmak istediğimiz iki noktanın içinde yer aldığı mahalle alanları ile daraltılmıştır. Eğer yeni oluşturmuş olduğumuz küme içerisinde herhangi bir rota bulunamaz ise tüm Türkiye verisi üzerinde yönlendirme işlemi yapılır. Böylelikle bir yol bulunması işlemi garanti altına alınır. Yönlendirme işlemi yeni oluşturulan harita verisi üzerinde yapılarak çalışma zamanında ciddi bir azalma sağlanmıştır.

Elli itarasyondan sonra çözüm ağacına en yakın eklenmemiş terminal düğüm bulunarak çözüm ağacına eklenir. Her bir iterasyonda hesaplanan yönlendirme sonuçları veritabanı üzerinde saklanmaktadır ve aynı iki düğüm için tekrar hesaplama yapılmamaktadır. Başlangıçta 332.424 adet terminal düğümle işletilmeye başlanan algoritma tamamlandığı zaman 412.620 adet düğüm içeren bir ağaç oluşturur.

CSBM üzerindeki herhangi bir çizginin üzerindeki bir nokta çözüm ağacımıza eklendiği zaman çizginin üzerindeki tüm noktalar çözüm ağacımıza eklenir.

Bu işlemler tüm terminal düğümler çözüm ağacına eklenınceye kadar devam etmektedir. Tüm terminal düğümlerin eklenmesi tamamlandığı zaman hizmet verilen tüm binaların önüne kadar kablolama işlemi yapılmış olur. Bundan sonra algoritmamızın çalışması sona erer. Algoritmanın en fazla terminal düğüm sayısı kadar çalışması beklenmektedir. Fakat herhangi bir terminal düğüm çözüm ağacımıza eklendiği zaman üzerinde bulunduğu CSBM çizgisinin üzerindeki diğer noktalar da çözüm çizgemize otomatik olarak eklenir. Bu da çalışma zamanında bir azalmaya neden olacaktır.

2.3.4.3 Sözdekod

Algorithm 2 Geliştirmiş Olduğumuz Algoritma

Girdi: Kenarları ağılıklı ve yönsüz bir çizge olan G

Girdi: Terminal düğüm kümesi olan L

Çıktı: Kenarları ağılıklı ve yönsüz bir ağaç olan T

Adım 1: Boş bir T ağacı oluşturulur ve L terminal düğüm setinden rastgele bir düğüm seçilerek T ağacına eklenir

Adım 2: T ağacına eklenmemiş L düğüm kümesinden T ağacına en yakın 10 adet düğümü belirle

Adım 3: Bulunan 10 adet düğümün her biri için T ağacına eklenmiş bu düğüme en yakın olan 5 adet düğümü belirle

Adım 4: Eklenmemiş ve eklenmiş düğümler arasındaki Dijsktra'nın en kısa yol algoritmasını kullanarak yönlendirme işlemi yap

Adım 5: 50 iterasyon sonucunda saklanan köşelerden en az ağırlığa sahip olan düğümü T çözüm ağacına ekle

Adım 6: L düğüm kümesindeki tüm düğümler T ağacına eklenene kadar Adım 2'den Adım 5'e kadar olan işlemleri tekrarla

3. DENEYSEL ÇALIŞMA

3.1 Geliştirmiş Olduğumuz Yazılım

3.1.1 Kullanılan Teknolojiler

3.1.1.1 PostGIS

PostGIS[7] coğrafi nesneleri destekleyen PostgreSQL ilişkisel veritabanı yönetim sisteminin üzerinde çalışan bir open source eklentidir. Lokasyon ve coğrafi nesneler üzerinde SQL dilini kullanarak işlemler yapılması için tasarlanmıştır. CBS sistemlerinin kullandığı veri tiplerini ve fonksiyonlarını PostgreSQL veritabanı yönetim sisteminin kullanabilmesini sağlamaktadır.

3.1.1.2 Open Street Map

OpenStreetMap (OSM)[8] dünya üzerinde serbest düzenlenebilir bir harita oluşturmak için kurulan işbirliği projesidir.

1.6 milyon kayıtlı kullanıcı tarafından anket, GPS verileri, hava fotoğrafçılığı ve diğer serbest kaynaklar kullanılarak datalar toplanmaktadır.Daha sonra bu kalabalık data kümesi Açık Veritabanı Lisansı altında kullanılabilir hale getirilir.

3.1.1.3 Open Source Routing Machine(pgRouting)

pgRouting[9], PostGIS/PostgreSQL coğrafi veritabanı üzerinde coğrafi yönlendirme yapmayı sağlamaktadır. Üzerinde tanımlı olan birçok algoritma bulunmaktadır. Bu algoritmalardan herhangi biri kullanılarak iki nokta arasındaki yönlendirme işlemi yapılabilir.

3.1.1.4 Open Layers

OpenLayers[10] harita datalarının web tarayıcısında görüntülenmesine yarayan bir açık kaynak javascript kütüphanesidir.

3.1.2 Program Çalışma Ayrıntıları

Geliştirmiş olduğumuz yazılım kullanıcılara hesaplama sonuçlarını gösteren haritaların bulunduğu kullanıcı arayüzü ve algoritmayı işleten hesaplama motoru olmak üzere iki ana kısımdan oluşmaktadır.

Kullanıcılar için hazırlanan arayüzler django framework (python programlama dili) kullanılarak yazılmıştır. Haritalar içinse openlayer javascript kütüphanesi kullanılmıştır.

Kullanıcı arayüzlerinin iki adet görevi bulunmaktadır. Kullanıcıların hizmet götürülecek binaları harita üzerinde kaydedilmelerini sağlamaktadır. Harita üzerinden sunduğumuz veri girişi imkanı kullanıcıların verilere kolay bir şekilde erişmesini ve yeni lokasyonlar ekleyebilmelerini sağlamaktadır. İkinci görevi ise kullanıcıların oluşturulan ağları görsel olarak görebilmelerini sağlamaktır.

Hesaplama Motoru ise girişi yapılan veriler üzerinde iki temel işlevi yerine getirmektedir. Bunlardan ilki bir ağın sıfırdan oluşturulması işlemidir. Diğeri ise oluşturulmuş bir ağ üzerine yeni bir düğümün eklenmesi işlemidir.

Geliştirmiş olduğumuz hesaplama motoru programla dili olarak python kullanmaktadır. Data saklama yöntemi olarak ise CBS datalarını işlemek için özelleştirilmiş PostgreSQL veritabanı yönetim sisteminin bir eklentisi olan PostGIS kullanılmıştır. Ayrıca harita üzerinde yönlendirme yapabilmek için yine PostgreSQL veritabanı yönetim sisteminin bir eklentisi olan pgRouting kullanılmaktadır.

Hesaplama motoru hesaplama işlemini, verilerin yönlendirme yapılabilecek formata dönüştürülmesi ve algoritmanın bu data üzerinde işletilmesi olarak iki adımda gerçekleştirir.

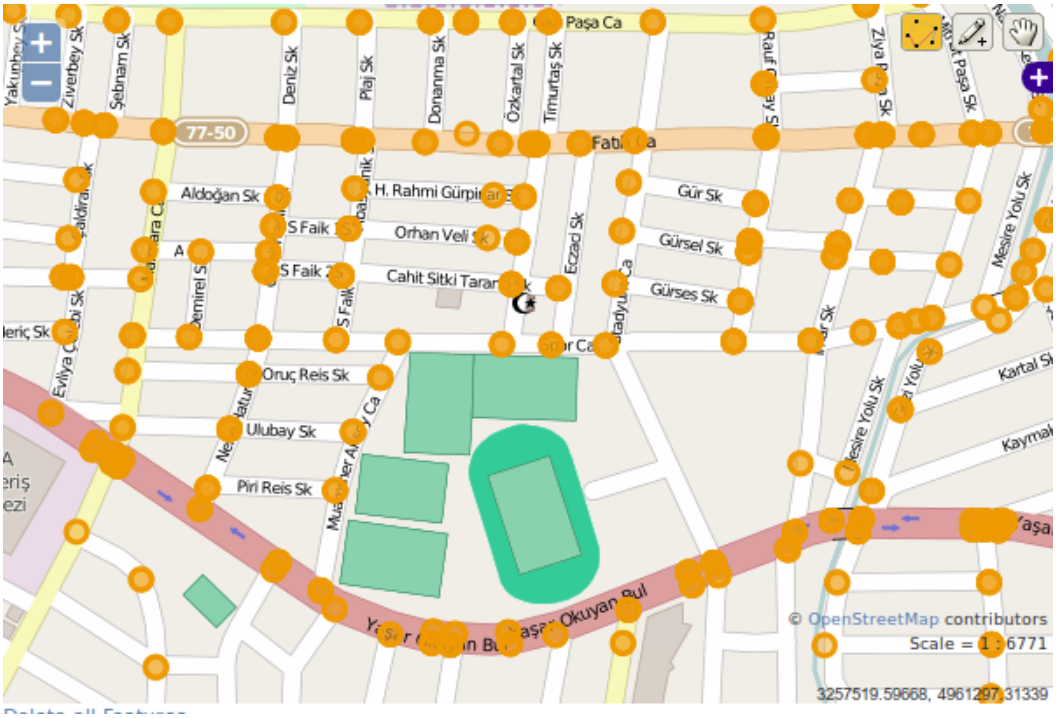
Verilerin yönlendirme yapılabilecek formata dönüştürülmesi

Harita dataları üzerinde yönlendirme işlemi cadde, sokak, meydan veya bulvarlar(CSBM) üzerinden yapılmak zorundadır. Hizmet verilecek noktaların koordinatları daha önce girişi yapılmış verilerdir. Şekil 3.1 hizmet verilecek olan binaların lokasyon bilgilerini gösterir harita örneğidir.

Şekil 3.1: Hizmet götürülmek istenen her bir düğümün lokasyonunu gösterir harita görüntüsü.

Yönlendirme işlemi sadece CSBM üzerinden gerçekleşmektedir. Bu yüzden hizmet verilmek istenen noktaların bağlı bulundukları CSBM'ye ait en yakın çizgi bulunur. Bulunan CSBM çizgisi noktalar halinde parçalanır ve bu noktalar yazılım tarafından kaydedilir. Bu işlem sayesinde aynı CSBM üzerinde bulunan onlarca hizmet noktasına hizmet götürebilmek için hesap yapacağımız nokta sayısını birkaç noktaya indirgemiş oluruz. Yani terminal olarak kullanacağımız düğüm kümemizi CSBM düzeyine indirgemiş oluruz. Bu bize hesaplama sırasında ciddi manada zaman kazancı sağlamaktadır. Şekil 3.2; Şekil 3.1 ile gösterilen hizmet verilmek istenen binaların CSBM bazında indirgenmiş nokta haritasını

göstermektedir. Bu işlem sonucunda oluşturmuş olduğumuz nokta kümesi artık algoritmamızı işletmemiz için uygun formata gelmiş olur.



Şekil 3.2: Formatlanmış verileri gösterir harita görüntüsü.

Algoritmanın işletilmesi

Algoritmanın işletilmesi için arka planda çalışacak ikinci bir uygulama geliştirilmiştir. Bu uygulama paralel programlama yaparak tüm illerin aynı anda hesaplama işlemini yapmaktadır. Paralel programlama için python programlama dilinden faydalanılmıştır. Hesaplama işlemi ise PostgreSQL veritabanı üzerinde fonksiyonlar aracılığıyla yapılmaktadır. Bir önceki adımda oluşturduğumuz CSBM nokta kümesi terminal düğümler olarak, tüm Türkiye'de bulunan sokaklar ise işlem yapılacak düğüm kümesini ifade etmektedir. Algoritma Bölüm 2.3.4.2'de açıklaması bulunan şekilde çalıştırılmaktadır. Şekil 3.3 algoritmanın tamamlanmasından sonraki sonuçları gösterir harita örneğidir.

Şekil 3.3: Algoritmanın çalıştırıldıktan sonraki sonuçları gösterir harita görüntüsü.

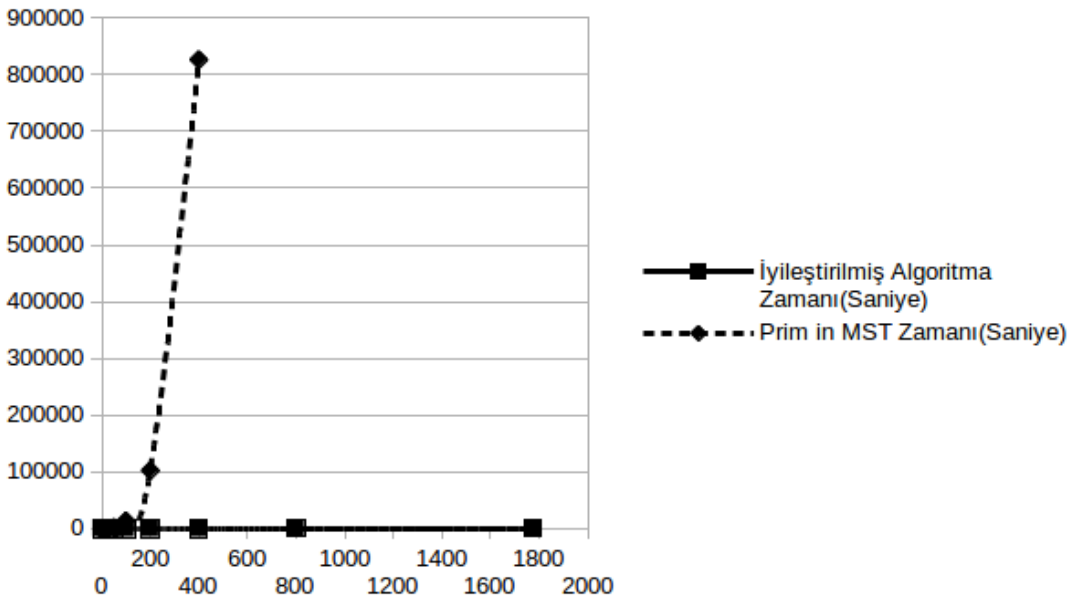
4. DEĞERLENDİRME

4.1 Çalışma Zamanı

Çizelge 4.1: Terminal düğüm sayılarına göre Prim'in MST algoritmasının geliştirmiş olduğumuz algoritma ile çalışma zamanlarının karşılaştırılmasıdır. Algoritma çalışma zamanları saniye cinsindendir.

Terminal Düğüm Sayısı	**İyileştirilmiş Algoritma Z.**	**Prim MST Z.**
0	0	0
20	21	170
50	43	1801
100	71	13265
200	89	102156
400	112	826482
800	154	
1776	181	

Çizelge 4.1'de belirlenmiş terminal düğüm setlerine göre çalışma zamanlarının karşılaştırılması yapılmıştır. Çizelge üzerindeki ilk kolon, üzerinde çalışılacak olan terminal düğüm sayısını göstermektedir. Her bir terminal düğüm seti için geliştirmiş olduğumuz iyileştirilmiş Prim'in MST sezgisel algoritması ile Prim'in MST sezgisel algoritması karşılaştırılmıştır. İkinci ve üçüncü kolonlarda yer alan değerler saniye cinsinden algoritmanın ilgili terminal düğüm seti için çalışma zamanını göstermektedir. Bu çizelgeye ait grafik Şekil 4.1 ile gösterilmiştir.



Şekil 4.1: Algoritmanın çalıştırıldıktan sonraki sonuçları gösterir harita görüntüsü.

Prim'in MST sezgisel algoritması çözüm ağacına eklenmemiş her bir terminal düğüm ile çözüm ağacına eklenmiş tüm düğümleri birbirine bağlayan kenarları bulmaktadır ve bu kenarlardan en kısa uzunluğu olan terminal düğümü çözüm kümesine eklemektedir. Setin içerisinde bulunan terminal düğüm sayısı arttıkça bu algoritmanın çalışma zamanı çok yüksek değerlere çıkmaktadır. Örneğin; 400 terminal düğüm için algoritma işletildiği zaman yaklaşık olarak 9,5 günde tamamlanmaktadır. Daha fazla terminal düğüm için hesaplama yapılamamıştır.

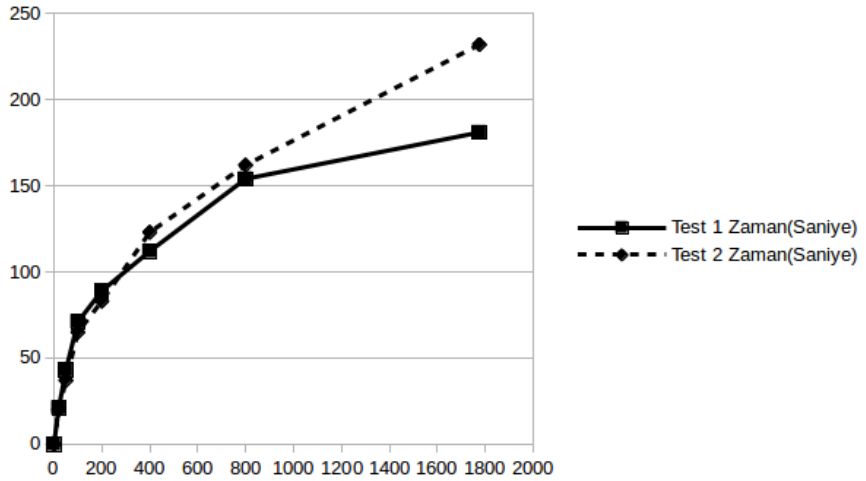
Geliştirmiş olduğumuz iyileştirilmiş Prim'in MST sezgisel algoritması ise ölçeklenebilir zamanda verilen terminal düğüm setini hesaplayabilmektedir. Her bir terminal düğümün çözüm kümesine eklenme süresi yaklaşık olarak aynı zamanı almaktadır.

İki algoritma arasındaki çalışma zamanı farkı Prim'in MST sezgisel yakınsama algoritmasının terminal düğüm sayısı artıkça çok hızlı bir şekilde çalışma zamanının artmasıdır. Geliştirmiş olduğumuz algoritma ise her bir terminal düğüm için yaklaşık olarak aynı hesaplama süresini sunmaktadır.

Çizelge 4.2: Geliştirmiş olduğumuz algoritma üzerinde terminal düğüm sayılarına göre yapılmış iki testin karşılaştırılması

Terminal Düğüm Sayısı(Adet)	**Test 1 Z(Saniye)**	**Test 2 Z(Saniye)**
0	0	0
20	21	20
50	43	37
100	71	65
200	89	83
400	112	123
800	154	162
1776	181	232

Geliştirmiş olduğumuz algoritmanın çalışma zamanındaki tutarlılığını göstermek için ikişer adet eşit sayıda terminal düğüm içeren terminal düğüm setleri oluşturulmuştur. Her bir terminal düğüm setinin içerdiği terminal düğümler faklıdır. Çizelge 4.2 verilen setlerdeki çalışma zamanlarının karşılaştırılmasını göstermektedir. Çizelge 4.2'deki ilk kolon çalışma setimizdeki terminal düğüm sayısını göstermektedir. İkinci ve üçüncü kolonlar ise yapılan iki testin çalışma zamanlarını saniye cinsinden göstermektedir. Çizelge 4.2'nın grafiğe çevrilmiş hali Şekil 4.2 ile gösterilmiştir.



Şekil 4.2: Algoritmanın çalıştırıldıktan sonraki sonuçları gösterir harita görüntüsü.

Çalıştırılan iki testin çalışma zamanı yaklaşık olarak aynıdır. İki test arasındaki ufak farklılık rastgele seçilen terminal düğüm kümesinin içerisindeki herhangi iki düğümün birbirleriyle olan uzaklığı, seçilen terminal düğümlerin aynı CSBM üzerinde olması gibi etkenlerden kaynaklanmaktadır. Algoritmamızın tanımında anlatıldığı gibi aynı CSBM çizgisi üzerindeki terminal düğümler herhangi biri eklendiği zaman çözüm kümesine otomatik olarak eklenmektedir.

Geliştirmiş olduğumuz algoritma hizmet verilen tüm illerdeki tüm hizmet verilen binaların ağ kurma işlemini sekiz saatte tamamlamıştır.

4.2 Sonuçlar

Çizelge 4.3 datalarını almış olduğumuz şirketin hizmet verdiği binalara hizmet götürülebilmesi için il bazında gerekli olan kazı uzunluklarını göstermektedir. Sonuçlar geliştirmiş olduğumuz algoritma tarafından hesaplanan değerlerdir.

Çizelge 4.3 başlığında kullanılan kısaltmaların açıklamaları:

TKU: Algoritmanın tamamlanmasının ardından yapılması gereken toplam kazı uzunluğu. Birim olarak metre kullanılmıştır.
FKU: Firma tarafından oluşturulmuş fiziksel ağın toplam kazı uzunluğu. Birim olarak metre kullanılmıştır.

Toplam Düğüm: Algoritmanın işletilmesi için gerekli olan CSBM düzeyine indirgenmiş düğüm sayısıdır.
Bina Sayısı: Hizmet verilecek binaların toplam sayısıdır.
Oran: Firmanın ağıyla, algoritmamızın çıktılarının karşılaştırılmasını göstermektedir.

Çizelge 4.3 üzerindeki ilk kolon hesaplamanın yapılmış olduğu ili göstermektedir. İkinci kolon hesaplama yapılan bina sayısını. Üçüncü kolon Bölüm 3.1.2'de anlatılan indirgeme işlemi yapıldıktan sonraki terminal düğüm sayımızı göstermektedir. Dördüncü kolon algoritmamız çalıştıktan sonra ilgili ilde ağın

Çizelge 4.3: Algoritmanın sonuçlarının, firma tarafından oluşturulan fiziksel ağ ile karşılaştırılması

İl	**Bina Sayısı**	**Toplam Düğüm**	**TKU**	**FKU**	**Oran**
Adana	189376	19900	991237	1195862	17.11%
Ankara	663318	42082	2405742	4369447	44.94%
Antalya	134079	11186	531649	897859	40.79%
Balıkesir	58832	6356	240735	344693	30.16%
Bursa	181008	21480	762839	1107721	31.13%
Denizli	138859	13252	433306	845610	48.76%
Edirne	24031	2724	109996	211408	47.97%
Erzincan	25260	3860	128513	191437	32.87%
Erzurum	51181	4102	214149	276707	22.61%
Eskişehir	60779	4962	245564	365010	32.72%
Gaziantep	159115	16278	616998	894524	31.02%
İstanbul	1413259	85800	3783339	8399694	54.96%
İzmir	900086	28516	999231	1865721	46.44%
Kayseri	98810	6240	338540	383792	11.79%
Kocaeli	101861	11720	505963	702172	27.94%
Konya	205564	19790	927792	898487	-3.26%
Manisa	77017	4694	174021	277913	37.38%
Mersin	165357	10446	498852	544384	8.36%
Samsun	60735	5338	227513	222517	-2.25%
Tekirdağ	50111	5806	245278	331051	25.91%
Yalova	16335	1776	86217	127976	32.63%
Zonguldak	35752	4404	212651	361122	41.11%
Toplam	4810725	330712	14680125	24815107	40.84%

oluşturulması için gerekli olan kazı uzunluğunu vermektedir. Beşinci kolon firma tarafından fiziksel olarak oluşturulmuş ağdaki toplam kazı uzunluğunu göstermektedir. Son kolon ise geliştirdiğimiz algoritma ile firmanın ağının oransal olarak karşılaştırmasını göstermektedir.

Firmanın hizmet vermiş olduğu 22 il için oranlara bakıldığı zaman toplamda 40% bir başarım elde ettiğimiz gözükmektedir. İl bazında ise 2 il haricindeki tüm illerde algoritmanın daha verimli çalıştığı ve başarımın yüksek olduğu gözükmektedir.

5. SONUÇ

Yapmış olduğumuz çalışma kapsamında matematikte çok sık kullanılan problemlerden biri olan Steiner ağaç problemi gerçek hayatta kullanılabilir hale getirilmiştir. Firmaların maliyetlerini optimize etme şansı doğmuştur. Teknoloji ile fiziksel bir çalışmanın optimize edilebilirliği gösterilmiştir.

Ayrıca toplumsal açıdan bakıldığı zaman, kazı çalışması yapılması çevre ve ses kirliliğine neden olmaktadır. Bu çalışma sayesinde dolaylı yoldan daha az kazı çalışması yaparak kısmi olarak çevre ve ses kirliliğinin önüne geçmiş oluruz.

Firmalar tarafından yeni bir ağ oluşturulmak istenmesi veya yeni hizmet noktaları eklenmeden önce yapılacak çalışmanın görselleştirilmesi sağlanmıştır.

Çalışmanın bir sonraki aşamasında çeşitli demografik bilgiler harita üzerine eklenerek kazı çalışması yapılacak bölgede, kazı uzunluğu minimum olmasının yanı sıra yeni bir ağın inşa edilmesinin firmaya olan karlılığının karşılaştırılması gibi ekstra özellikler eklenebilir.

KAYNAKLAR

[1] Prömel, H. J. and Steger, A., The Steiner Tree Problem: A Tour through Graphs, Algorithms, and Complexity (Advanced Lectures in Mathematics), 2002

[2] Hwang, F. "On Steiner Minimal Trees with Rectilinear Distance" SIAM J. Appl. Math. SIAM Journal on Applied Mathematics, 104-114.

[3] Benjamin Zhan, F. "Three Fastest Shortest Path Algorithms on Real Road Networks: Data Structures and Procedures". Journal of Geographic Information and Decision Analysis, vol.1, no.1, pp. 70-82, 1997

[4] "Postgresql Dökümantasyon" erişim adresi: https://wiki.postgresql.org/wiki, erişim tarihi: Ekim 15, 2015.

[5] "Python 3.5.0 Dökümantasyon" erişim adresi: https://docs.python.org/3/, erişim tarihi: Ekim 15, 2015.

[6] "Bilgi Teknolojileri ve İletişim Kurumu - Pazar Verileri" erişim adresi: http://www.btk.gov.tr/tr-TR/Sayfalar/Pazar-Verileri, erişim tarihi: Haziran 14, 2014.

[7] "PostGIS 2.2.1dev Dökümantasyon" erişim adresi: http://postgis.net/docs/manual-2.2/, erişim tarihi: Ekim 15, 2015.

[8] "OpenStreetMap Hakkında" erişim adresi: https://www.openstreetmap.org/about, erişim tarihi: Ekim 15, 2015.

[9] "OpenStreetMap Dökümantasyon" erişim adresi: http://pgrouting.org/documentation.html, erişim tarihi: Ekim 15, 2015.

[10] "Openlayers Dökümantasyon" erişim adresi: http://openlayers.org/en/v3.10.1/doc/, erişim tarihi: Ekim 15, 2015.

[11] Cormen, Thomas H., Introduction to algorithms. 3rd ed., Cambridge, Mass.: MIT Press, 2009

[12] Kleinberg, J., Tardos, E., Algorithm design, Boston, Pearson/Addison-Wesley, 2006

EKLER

A. Veriler

A.1 Tablolar

```
CREATE TABLE routewcsbm.route (
    id SERIAL,
    csbm_id INTEGER NOT NULL,
    source_id INTEGER NOT NULL,
    target_id INTEGER NOT NULL,
    isk INTEGER NOT NULL,
    add BOOLEAN DEFAULT false NOT NULL,
    is_main BOOLEAN DEFAULT false NOT NULL,
    result_id INTEGER DEFAULT 1 NOT NULL,
    CONSTRAINT route_pkey PRIMARY KEY(id)
)
WITH (oids = false);

CREATE TABLE routewcsbm.route (
    id SERIAL,
    csbm_id INTEGER NOT NULL,
    source_id INTEGER NOT NULL,
    target_id INTEGER NOT NULL,
    isk INTEGER NOT NULL,
    add BOOLEAN DEFAULT false NOT NULL,
    is_main BOOLEAN DEFAULT false NOT NULL,
    result_id INTEGER DEFAULT 1 NOT NULL,
    CONSTRAINT route_pkey PRIMARY KEY(id)
)
WITH (oids = false);

CREATE TABLE routewcsbm.route_result (
```

```
    id INTEGER NOT NULL,
    name TEXT NOT NULL,
    geom public.geometry,
    geom point public.geometry,
    geom csbm public.geometry,
    geom result point public.geometry,
    uzunluk2 BIGINT,
    CONSTRAINT route result pkey PRIMARY KEY(id)
)
WITH (oids = false);

CREATE TABLE routewcsbm.route result temp (
    id SERIAL,
    geom public.geometry,
    result id INTEGER,
    uzunluk2 NUMERIC,
    CONSTRAINT route result temp pkey PRIMARY KEY(id)
)
WITH (oids = false);


CREATE TABLE routewcsbm.route temp (
    id SERIAL,
    x NUMERIC(1000,20) NOT NULL,
    y NUMERIC(1000,20) NOT NULL,
    geom point public.geometry NOT NULL,
    cost NUMERIC(1000,20),
    node INTEGER,
    edge INTEGER,
    add BOOLEAN NOT NULL,
    route id INTEGER,
    source id INTEGER,
    target id INTEGER,
    routing main point BOOLEAN DEFAULT false NOT NULL,
```

```
    is_main BOOLEAN DEFAULT false NOT NULL,
    result_id INTEGER DEFAULT 1 NOT NULL,
    "order" BIGINT,
    order2 INTEGER,
    CONSTRAINT route_temp_pkey PRIMARY KEY(id),
    CONSTRAINT route_temp_fk FOREIGN KEY (route_id)
    REFERENCES routewcsbm.route(id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE,
    CONSTRAINT route_temp_fk1 FOREIGN KEY (source_id)
    REFERENCES routewcsbm.route_temp(id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE,
    CONSTRAINT route_temp_fk2 FOREIGN KEY (target_id)
    REFERENCES routewcsbm.route_temp(id)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
    NOT DEFERRABLE
)
WITH (oids = false);
```

A.2 Prosedürler

```
CREATE OR REPLACE FUNCTION routewcsbm.route (
    g_result_id integer,
    g_clear boolean = false
)
RETURNS void AS
$body$
DECLARE

BEGIN

    IF g_clear IS TRUE THEN
        TRUNCATE TABLE routewcsbm.route_temp;
    END IF;
    INSERT INTO routewcsbm.route_temp(
        x, y, geom_point, cost, node, edge, add,
        route_id, source_id, target_id, routing_main_point,
        result_id
    )
    SELECT DISTINCT * FROM ((
        SELECT st_x(cvp.the_geom),
            st_y(cvp.the_geom),
            cvp.the_geom,
            0,
            r.source_id::INTEGER,
            NULL::INTEGER,
            FALSE,
            r.id,
            NULL::INTEGER,
            NULL::INTEGER,
            TRUE,
            r.result_id
```

```
        FROM routewcsbm.route r
    LEFT JOIN turksat.csbm_vertices_pgr cvp
        ON cvp.id = r.source_id
    WHERE r.result_id = g_result_id
    ORDER BY id ASC
) UNION ALL (
    SELECT st_x(cvp.the_geom),
        st_y(cvp.the_geom),
        cvp.the_geom,
        0,
        r.target_id::INTEGER,
        NULL::INTEGER,
        FALSE,
        r.id,
        NULL::INTEGER,
        NULL::INTEGER,
        TRUE,
        r.result_id
    FROM routewcsbm.route r
    LEFT JOIN turksat.csbm_vertices_pgr cvp ON cvp.id =
    r.target_id
    WHERE r.result_id = g_result_id
    ORDER BY id ASC
)) AS xx;

UPDATE routewcsbm.route_temp
SET source_id = id
WHERE result_id = g_result_id;


UPDATE routewcsbm.route_temp
SET is_main = TRUE, add = TRUE
WHERE id =
(
```

```
            SELECT id
            FROM routewcsbm.route_temp_rr
            WHERE rr.result_id = g_result_id
            LIMIT 1
        );
END;
$body$
LANGUAGE 'plpgsql'
VOLATILE
CALLED ON NULL INPUT
SECURITY INVOKER
COST 100;



CREATE OR REPLACE FUNCTION routewcsbm.routeatob (
    g_result_id integer
)
RETURNS void AS
$body$
DECLARE
    source INTEGER;
    target INTEGER;
    rec RECORD;
    rec2 RECORD;
    rec3 RECORD;
    line TEXT;
    costt NUMERIC = 0;
    costt2 NUMERIC = 0;
    lit_target INTEGER;
    lit_route_id INTEGER;
    lit_point INTEGER;
    lit_source INTEGER;
    l_network TEXT;
```

```
    l_geom geometry;
    l_seq BIGINT;
    linestring public.geometry;
BEGIN
    --raise notice '%',route.set_route(1,FALSE);
    line = '';
    --l_network = routewcsbm.get_route(g_seq_id);
    l_geom = (SELECT
                st_collect(geom_point)
            FROM routewcsbm.route_temp rt
            WHERE add IS TRUE AND
                result_id = g_result_id);
    FOR rec3 IN (
        SELECT r.geom_point as geom,
            r.id,
            r.node,
            r.route_id
        FROM routewcsbm.route_temp r
        WHERE r.add IS FALSE
            AND r.routing_main_point IS TRUE
            AND r.result_id = g_result_id
        ORDER BY r.geom_point <#> l_geom ASC
        LIMIT 10
    ) LOOP
        --raise notice '------------%------------';rec3.id;
        FOR rec2 IN (
            SELECT r.id,r.node,r.geom_point as geom,
            r.is_main
            FROM routewcsbm.route_temp r
            WHERE r.add IS TRUE AND
                r.result_id = g_result_id
            GROUP BY 1,2,3,4
            order by r.geom_point <#> rec3.geom ASC
```

```
    limit 5
) LOOP
    costt2 = NULL;
    SELECT INTO costt2 sum(t.cost)
    FROM routewcsbm.route_temp t
    WHERE (t.source_id = rec3.id
                AND t.target_id = rec2.id)
        OR (t.source_id = rec2.id
                AND t.target_id = rec3.id);

    --raise notice 'Basladi2';

    IF costt2 IS NULL THEN
        IF rec2.node IS NOT NULL THEN
            --raise notice 'Basladi3';
            target = rec2.node;
        ELSE
            --raise notice 'Basladi4';
            select INTO target p.id
            from turksat.csbm_vertices_pgr p
            order by p.the_geom <-> rec2.geom_point
                                    ASC
            limit 1;
        END IF;
        IF rec3.node IS NOT NULL THEN
            --raise notice 'Basladi6';
            source = rec3.node;
        ELSE
            --raise notice 'Basladi5';
            select into source p.id
            from turksat.csbm_vertices_pgr p
            order by p.the_geom <-> rec3.geom ASC
            limit 1;
        END IF;
```

```
BEGIN
    --raise notice 'Basladi7';
    INSERT INTO routewcsbm.route_temp(
        x, y, geom_point, cost, node, edge,
        add, route_id,source_id,target_id,
        result_id,"order")
    (
        SELECT st_x(w.the_geom),
            st_y(w.the_geom),
            w.the_geom,
            pgroute.cost,
            pgroute.node,
            pgroute.edge,
            FALSE,
            rec3.route_id,
            rec3.id,
            rec2.id,
            g_result_id,
            pgroute.seq
        FROM (
        SELECT seq,
              id1 AS node,
              id2 AS edge,
              cost
         FROM pgr_dijkstra(
         SELECT rs.id, source, target,
         length::double precision AS cost,
         linestring
         FROM turksat.csbm as rs
         JOIN turksat.mahalle_sokak_mh ON
         mh.il_sokak_kodu=rs.il_sokak_kodu
```

```
            WHERE mh.mahalle_id IN (
            SELECT
                    m.id
            FROM turksat.csbm as rs
            LEFT JOIN turksat.mahalle_sokak mh
            ON
            mh.il_sokak_kodu=rs.il_sokak_kodu
            LEFT JOIN turksat.mahalle m ON
            m.id = mh.mahalle_id
            WHERE rs.source IN
            (''||source||'', ''||target||'')
            OR rs.target IN
            (''||source||'', ''||target||'')
            )::TEXT, source::INTEGER,
            target::INTEGER, FALSE, false)
          ) AS pgroute
          LEFT JOIN
          turksat.csbm_vertices_pgr w
          ON w.id = pgroute.node
      );
EXCEPTION WHEN others THEN
      --raise notice 'Basladi8';
      INSERT INTO routewcsbm.route_temp(
          x, y, geom_point, cost, node, edge,
          add, route_id,source_id,target_id,
          result_id,"order")
      (
          SELECT st_x(w.the_geom),
              st_y(w.the_geom),
              w.the_geom,
              pgroute.cost,
              pgroute.node,
              pgroute.edge,
              FALSE,
```

```
                    rec3.route_id,
                    rec3.id,
                    rec2.id,
                    g_result_id,
                    pgroute.seq
                FROM (
                    SELECT seq,
                        id1 AS node,
                        id2 AS edge,
                        cost
                    FROM pgr_dijkstra(SELECT id,
                    source::integer,
                    target::integer,
                    length::double precision
                    AS cost, linestring
                    FROM turksat.csbm::TEXT,
                    source::INTEGER,
                    target::INTEGER,
                    FALSE, FALSE)
                ) AS pgroute
                LEFT JOIN
                turksat.csbm_vertices_pgr w
                ON w.id = pgroute.node
            );
            CONTINUE;
        END;

        SELECT INTO costt2 sum(t.cost)
        FROM routewcsbm.route_temp t
        WHERE t.source_id = rec3.id
        AND t.target_id = rec2.id;

END IF;
```

```
            IF costt2 < costt OR costt = 0 THEN
                lit_target = rec2.id;
                lit_point = rec3.id;
                lit_source = rec3.id;
                lit_route_id = rec3.route_id;
                costt = costt2;
            END IF;

        END LOOP;
    END LOOP;

    --raise notice '% %',lit_target,lit_source;
    INSERT INTO routewcsbm.route_result_temp
    (geom,result_id)(
        SELECT st_union(t1.linestring),g_result_id FROM (
            SELECT DISTINCT c.*
            FROM routewcsbm.route_temp rt
            LEFT JOIN routewcsbm.route r
            ON rt.route_id = r.id
            LEFT JOIN turksat.csbm c
            ON c.il_sokak_kodu = r.isk
            WHERE rt.id = lit_target
        ) t1
    );

    INSERT INTO routewcsbm.route_result_temp
    (geom,result_id)(
        SELECT st_makeline(geom_point),g_result_id from (
            SELECT r.geom_point
            FROM routewcsbm.route_temp r
            WHERE r.source_id = lit_source
            AND r.target_id = lit_target
            ORDER BY "order" ASC
```

```
    ) as t1

);


INSERT INTO routewcsbm.route_result_temp
(geom,result_id)(
    SELECT st_union(t1.linestring),g_result_id FROM (
        SELECT DISTINCT  c.*
        FROM routewcsbm.route_temp rt
        LEFT JOIN routewcsbm.route r
        ON rt.route_id = r.id
        LEFT JOIN turksat.csbm c
        ON c.il_sokak_kodu = r.isk
        WHERE rt.id = lit_source
    ) t1
);

l_seq =
nextval('routewcsbm.route_temp_order_id'::regclass);
raise notice 'order2_%',l_seq;

UPDATE routewcsbm.route_temp
SET add = TRUE, "order2" = l_seq
WHERE source_id = lit_source
AND target_id = lit_target
AND add IS FALSE;


UPDATE routewcsbm.route_temp
SET add = TRUE, "order2" = l_seq
WHERE id = lit_point;
```

```
    UPDATE routewcsbm.route temp
    SET "order2" = l.seq
    WHERE id = lit.target;


    UPDATE routewcsbm.route temp
    SET add = TRUE, "order2" = l.seq
    WHERE route.id IN (
        SELECT DISTINCT rou.id FROM routewcsbm.route ro
        LEFT JOIN routewcsbm.route rou ON ro.isk = rou.isk
        WHERE ro.id = lit.route.id
    )
    AND add IS FALSE
    AND routing.main.point IS TRUE
    AND result.id = g.result.id;

END;
$body$
LANGUAGE 'plpgsql'
VOLATILE
CALLED ON NULL INPUT
SECURITY INVOKER
COST 100;



CREATE OR REPLACE FUNCTION routewcsbm.set_route (
  g_result_id integer
)
RETURNS void AS
$body$
DECLARE
```

```
    l_record RECORD;
    l_record2 RECORD;
    l_record3 RECORD;
    l_sayac INTEGER = 0;
    line TEXT = '';
    line_main RECORD;
    line2 TEXT = '';
    line_point TEXT = '';
    line_multi_text = '';
    l_route_id BIGINT;

BEGIN
    line_multi = '';
    FOR l_record IN (
    SELECT DISTINCT st_astext(geom) as geom
    FROM (
            SELECT geom
            FROM routewcsbm.route_result_temp
            WHERE geom IS NOT NULL AND
                    result_id = g_result_id
            ORDER BY id ASC
          ) t1)
    LOOP
      if length(l_record.geom) > 45 THEN
        line_multi = line_multi || replace(
        replace(
        replace(l_record.geom,'MULTILINESTRING((','')
        ,'))',')')
        ,'LINESTRING','') || ',';
      END IF;
    END LOOP;
    UPDATE routewcsbm.route_result
    SET geom = st_geomfromtext('MULTILINESTRING(' ||
    substring(line_multi from 0 for length(line_multi) - 1)
```

```
|| '))')
WHERE id = g_result_id;

line_multi = '';

line = '';
l_sayac = 0;
FOR l_record IN (
  SELECT r.*
  FROM routewcsbm.route_temp r
  WHERE r.add is true AND
        r.result_id = g_result_id
  ORDER BY r.id ASC)
LOOP

  IF l_record.x IS NOT NULL
  AND l_record.y IS NOT NULL THEN
    if l_record.routing_main_point IS TRUE THEN
      line_multi = line_multi || '(' || l_record.x
      || '_' || l_record.y ||
        '),';
    end if;
    line = line || '(' || l_record.x
    || '_' || l_record.y || '),';
  END IF;
  l_sayac = l_sayac + 1;
END LOOP;
UPDATE routewcsbm.route_result
SET geom_point = st_geomfromtext('MULTIPOINT(' ||
substring(line_multi from 0 for length(line_multi) - 1)
|| '))'),
geom_result_point = st_geomfromtext('MULTIPOINT(' ||
substring(line from 0 for length(line) - 1) || '))')
WHERE id = g_result_id;
```

```
    line = '';
    FOR l_record3 IN (
      SELECT st_astext(c.linestring) as line
      FROM routewcsbm.route t
           LEFT JOIN turksat.csbm c ON c.id = t.csbm_id
      WHERE t.result_id = g_result_id)
    LOOP
       line = line || l_record3.line || ',';
    END LOOP;
    line = replace(line,'LINESTRING','');
    UPDATE routewcsbm.route_result
    SET geom_csbm = st_geomfromtext('MULTILINESTRING(' ||
    substring(line from 0 for length(line) - 1) || '))')
    WHERE id = g_result_id;

END;
$body$
LANGUAGE 'plpgsql'
VOLATILE
CALLED ON NULL INPUT
SECURITY INVOKER
COST 100;
```

B. Algoritma

B.1 Dijkstra'nın en kısa yol algoritması

B.1.1 Algoritmanın Çalışması

Elimizde yönsüz ve kenar ağırlıkları olan bir çizge $G = (V, E, W)$ olsun. Bu çizge üzerinde S başlangıç düğümüdür. Dijkstra algoritması tüm düğümlere adım adım geçici değerler atamaktadır. Geçici değer ataması iki düğüm arasındaki kenar uzunluğunun hesaplama yapılan düğümün değeri ile toplanarak bulunur. Algoritma adım adım tüm düğümleri kapsayacak şekilde genişler.

İlk olarak her düğüme bir geçici değer atanır. Başlangıç düğümü için bu değer sıfır, diğer tüm düğümler sonsuz olarak verilir. İlk düğüm ziyaret edildi olarak, diğer tüm düğümler ise ziyaret edilmedi olarak işaretlenir.

Hesaplama yapılan düğüm için, komşu olan ziyaret edilmemiş tüm düğümler geçici değerler atanır. Her bir düğüm için yeni hesaplanan geçici değer üzerinden karşılaştırma yapılır ve en küçük değer seçilir. Örnek olarak, hesaplama yapılan düğüm A ve geçici değeri 6 olsun, bağlantılı komşusu olan B'nin a ya olan uzaklığı 2, bu durumda B düğümünün geçici değeri $6 + 2 = 8$ olur. Eğer B düğümün önceki geçici değeri 8 den daha büyük ise geçici değer 8 olarak değiştirilir veya 8 den daha küçük ise eski değeri korunur. Hesaplama yapılan düğümün tüm komşularına geçici değerleri atandıktan sonra komşulardan en küçük değere sahip olan düğüm ziyaret edilen düğümler kümesine dahil edilir ve çözüm çizgemize eklenir.Bu durumda bu düğüm bir daha ziyaret edilmez.

Eğer komşu düğümlerin hepsi ziyaret edilmiş olarak işaretlenmiş veya ziyaret edilmemiş düğüm kümesi boş ise algoritma sona erer.

Diğer tüm durumlarda , en küçük geçici değere sahip yeni bir ziyaret edilmemiş düğüm seçilir, bu düğüm hesaplama yapılan düğüm olarak işaretlenir ve algoritma baştan koşturulur.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı	: KARAKAŞ, Mehmet Emin
Uyruğu	: T.C.
Doğum tarihi ve yeri	: 27.01.1986 Keçiborlu
Medeni hali	: Evli
Telefon	: 05053783331
Faks	:
e-mail	: mkarakas@etu.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet Tarihi
Y. Lisans	TOBB Ekonomi ve Teknoloji Üniversitesi	Devam
Lisans	Hacettepe Üniversitesi	2008

İş Deneyimi

Yıl	Yer	Görev
2009-	Türksat Uydu Haberleşme Kablo TV ve İşletme A.Ş.	Uzman

Yabancı Dil

İngilizce (Orta)

Yayınlar

KARAKAŞ, M.E., "İNTERNET SERVİS SAĞLAYICILARININ MALİYET OPTİMİZASYONU". Türkiye'de İnternet Haftası Konferansı Bildiriler Kitapçığı, 2015.