MANİSA CELAL BAYAR ÜNİVERSİTESİ

HASAN FERDİ TURGUTLU TEKNOLOJİ FAKÜLTESİ

YAPAY ZEKA DERSİ PROJESİ

GEZGİN SATICI PROBLEMİ

Hazırlayanlar:

Damla DEMİR
Nilay Rabia BİLİR
Burakcan TİMUÇİN

1. GİRİŞ

Gezgin Satıcı Problemi (GSP), uygulama zenginliği nedeniyle en çok ilgi çeken problemlerden birisidir. Problem iki bilim adamı tarafından bağımsız olarak yaklaşık aynı zamanlarda ortaya koyulmuştur. Karl Menger 5 Şubat 1930 tarihinde Viyana'da bir matematik kolokyumunda, Hassler Whitney ise 1934 yılında Princeton Üniversitesi'nde bir seminerde problemi tanımlamışlardır. Problemin ilk matematiksel programlama gösterimi ve çözüm önerisi Dantzig ve arkadaşları tarafından verilmiştir.

GSP verilen n tane şehir için, her şehre bir kez uğramak koşuluyla tekrar başlangıç şehrine geri dönen en kısa (en az maliyetli) turu bulma problemidir. Tanımlanma açısından kolay görünse de çözülmesi zor bir problemdir. GSP'nin çözümü için birçok algoritma önerilmiştir. GSP'yi çözmek için üretilen yöntemleri kesin yöntemler ve sezgisel (heuristic) yöntemler olmak üzere iki gruba ayırabiliriz.

Kesin yöntemler verilen problem için kesin çözüm üretmektedirler, ancak hesaplanma açısından pahalıdırlar. Yani şehirlerin sayısı 20'yi aştığında problemi çözmek zorlaşır. Bu durumda sezgisel algoritmalar devreye girer. Sezgisel algoritmalar probleme tam olarak çözüm vermese bile yaklaşık bir çözüm verirler. Büyük boyutlu problemlerde sezgisel algoritmalar, kesin algoritmalara göre daha çok tercih edilmektedirler.

İşte biz de bundan yola çıkarak GSP için yeni bir sezgisel algoritma önerdik. Önerdiğimiz algoritma için JavaScript dilinde uygulamasını yapıp Ege Bölgesi illeri üzerinde denemeler yaptık.

2. PROJENİN AMACI

Bu projenin amacı, birçok uygulaması bulunan Gezgin Satıcı Problemi için yeni bir sezgisel algoritma önermek ve önerilen algoritmayı günlük hayatta uygulamaktır.

Önerilen algoritma en yakın komşu algoritmasının ve genetik algoritmasının birer genelleştirilmesi olarak geliştirilmiştir. Ayrıca her iki algoritma arasındaki farklılıkların gözlemlenmesi de projeye dahil edilmiştir.

3. GEZGİN SATICI PROBLEMİ

Gezgin Satıcı Problemi (GSP), matematik ve bilgisayar bilimleri alanlarında incelenen ve bilinen bir eniyileme problemidir. GSP, aralarındaki uzaklıklar bilinen N adet noktanın (şehir, parça veya düğüm gibi) her birisinden yalnız bir kez geçen en kısa veya en az maliyetli turu bulmayı hedefleyen bir problemdir. Çizge teorisinde ise, GSP "Verilen bir ağırlıklı çizgede (düğümler yani tepeler şehirleri; ayrıtlar ise şehirlerin arasındaki yolları göstermek; ağırlıklar da yolun maliyeti veya uzunluğu olmak üzere) en düşük maliyetli Hamilton Turu'nun bulunması" şeklinde tanımlanabilir.

Hamilton Turu: Bir çizge üzerindeki her tepeden sadece bir kez geçen ve başladığı noktada biten tura Hamilton turu denir.

Ege Bölgesi Haritası

Kısaca GSP, müşteriler, işler, şehirler, noktalar vb. için belirli bir güzergah ya da sıralama tayin etme problemidir. Nesnelerin minimum maliyet, uzaklık veya zaman alacak şekilde sıralanması gezgin satıcı problemlerine girmektedir.

Ege Bölgesindeki şehirler arasındaki uzaklığın verildiği tablo aşağıdaki gibidir.

	İzmir	Aydın	Manisa	Muğla	Denizli	Uşak	Kütahya	Afyon
İzmir	**0**	126	35	225	224	211	333	327
Aydın	126	**0**	155	99	126	273	408	346
Manisa	35	155	**0**	254	208	195	317	311
Muğla	225	99	254	**0**	145	295	430	368
Denizli	224	126	208	145	**0**	150	285	223
Uşak	211	273	195	295	150	**0**	139	116
Kütahya	333	408	317	430	285	139	**0**	100
Afyon	327	346	311	368	223	116	100	**0**

3.1. PROBLEMİN YAPISI

Gezgin Satıcı Problemi, N tane şehir arasında, her şehre bir kez uğramak koşuluyla en kısa yolu kat edebilecek şekilde dolaşılacak turun bulunması mantığı ile oluşturulur.

Her şehre bir kez uğramak koşulu ile denenebilecek bütün ihtimallerden oluşan şehir turları arasındaki en kısa mesafeli turu bulmak amaçlanmaktadır. Bunun için ilk denenen tur en kısa mesafeli olarak kabul edilir. Sırasıyla diğer güzergahların uzunluklarıyla karşılaştırılır. Sonuçta en küçük uzunluğa sahip tur en iyi tur olarak belirlenir.

Problemin amacı, satıcıya bu en kısa yolu sunabilmektir.

1. Satıcının ilk şehirde, n - 1 değişik şehir arasında seçim hakkı vardır.
2. İkinci şehirde, n - 2 değişik şehir arasında seçim hakkı vardır.

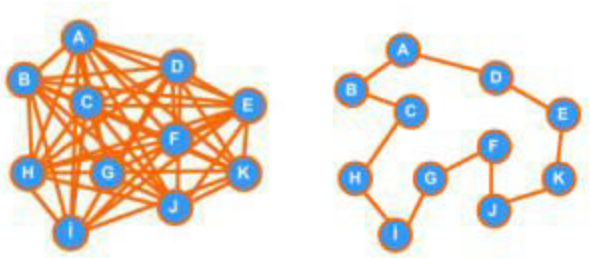
Bulunan her bir Hamilton yolu ters sırada da ziyaret edilebileceğinden satıcının (n-1)!/2 değişik tur arasından seçim yapması yeterli olacaktır. Örnek olarak,

- 7 şehir için (7-1)!/2
- 8 şehir için (8-1)!/2

ihtimaller vb. gibi söz konusudur. Yani, şehir sayısı arttığında işlem karmaşıklığı üstel olarak artmaktadır. Bu da problemin çözümünü zorlaştırmaktadır.

3.2. GEZGİN SATICI PROBLEMİ İÇİN ÇÖZÜM YÖNTEMLERİ

GSP için kesin çözüm denildiğinde, ilk akla gelen yöntem tüm mümkün olasılıkların denendiği sayımlama yöntemidir.



Gezgin satıcı probleminde 15 şehre kadar kesin sonuçlar kolayca bulunabilmektedir. Çözüm karmaşıklığı şehir sayısına üstel olarak bağımlı olduğu için 16 şehirden sonra imkansız hale gelmektedir. Mesela, 2001 yılında Almanya'nın 15112 adet şehrini gezen ve her şehirden yalnızca bir kez geçen en kısa yolu bulabilmek için Rice ve Princeton Üniversitelerindeki bilgisayarlar 22 yıldan daha fazla süre çalışarak yaklaşık 66000 km'lik güzergahı bulmuşlardır.

Görüldüğü gibi bu problemde, olasılıkların tamamının taranmasını gerektiren çözüm yönteminde şehirlerin sayısının artması ile problemin çözümü imkânsız hale gelmektedir. İlk aşamada çözülebilecek gibi görünen GSP problemi, belirli noktadan sonra çok uzun zaman gerektirmekte ve gittikçe çözülemez hale gelmektedir. Bu nedenle de problemin kısa zamanda değişik yaklaşımlarla çözümü düşünülmüştür. Çözüm uzayının yalnızca belirli bir kısmının tarandığı sezgisel yöntemler geliştirilmiştir. Çözüm uzayının tümü gözden geçirilmediği için sezgisel yöntemlerle optimum sonuca ulaşma garantisi yoktur.

3.3. PROJEDE KULLANILAN ALGORİTMALAR

Bu projede GSP'nin çözümü için iki çeşit algoritma kullanılmıştır. Bunlar, En Yakın Komşu Algoritması (Nearest Neighbour Algorithm) ve Genetik (Genetic) Algoritması'dır.

- **En Yakın Komşu Algoritması (Nearest Neighbour Algortithm)**

En yakın komşu algoritması GSP için geliştirilmiş en basit ve en iyi bilinen sezgisel algoritmadır. Algoritmanın mantığı hep en yakın komşuya gitmeye dayanır. GSP için algoritmanın adımları aşağıdaki gibi olacaktır:

Adım 1. Rastgele şehir seç.
Adım 2. Alt tur oluşturmayan, en yakın ziyaret edilmemiş şehri bul ve oraya git.
Adım 3. Ziyaret edilmemiş şehir kaldı mı? Cevap evet ise, Adım 2'ye git.
Adım 4. Başlangıç şehre geri dön.
Adım 5. Algoritmayı sonlandır.

- **Genetik Algoritma (Genetic Algorithm)**

Bilgisayar bilimlerinin doğa bilimlerinden (biyoloji) öğrendiği ve kendi problemlerini çözmek için kullandığı bir yöntemdir. Bu algoritmada genetikte kullanılan temel 3 işlem kullanılır. Bu üç temel işlem;

1. Çaprazlama (Crossover)
2. Mutasyon (Mutation)
3. Başarılı Gen Seçimi (Selection)

Şehirleri izlerken izlediğimiz rotaların her birini birer kromozom olarak temsil ediyoruz. Buna göre, N şehir içeren bir örnek için farklı gezinme sıralarını ifade eden kromozomlarımız olacaktır. GSP için algoritmanın adımları aşağıdaki gibi olacaktır:

İlk olarak, belirli sayıda kromozomu rastgele şekilde yaratarak bir başlangıç popülasyonu oluşturacağız. Bu popülasyondan, en kısa mesafeyi sağlayan, yani çözüm için en uygun olan kromozomlardan belirli sayıda seçip birbiriyle ikişer ikişer çaprazlayarak yeni bireyler elde edeceğiz. Popülasyondaki en uzun mesafeli, yani en kötü bireyleri eleyeceğiz. Onlardan boşalan yere, çaprazlama sonucunda oluşan yeni bireyleri ekleyeceğiz.

Böylece yeni bir nesil oluşacak. Yeni nesildeki en iyi bireyleri seçip çaprazlama işlemlerini tekrarlayacağız. N adet nesil sonucunda bulduğumuz en iyi, en kısa mesafeli kromozom bizim çözümümüz olacaktır.

4. KAYNAK KODLAR

1. Ana fonksiyonun çalıştığı **Ana.js**

Bu dosya aracılığıyla diğer dosyalara erişim sağlanacak ve istenilen biçimde algoritmaların uygulanması sağlanacaktır. Window.onload olayı main() olayı ile eşdeğerdir.

```
let cizge = new Cizge();
//Cizge.js'de sınıf olarak yaratılmış Cizge'den nesne oluşturuldu
let devamEdenGenetik = undefined;
//Henüz değer ataması yapılmayan devam etmekte olan Genetik Algoritma değişkenimiz

window.onload = function(){
    let canvas = document.getElementById("noktalar_canvas");
    //İlk canvastan nesne yaratılır
    noktalariGoster(undefined, "noktalar_canvas");
    //İlk anda il seçimi olmadığı için nokta gözükmeyecektir
    canvas.addEventListener("mousedown", function(event){ //İl seçimi yapıldığı anda
        cizge.dugumEkle(event.offsetX, event.offsetY, yeniId());
        //Tıklanılan yerin x-y koordinatları alınır
        //Çizgedeki düğüme atanır
        noktalariGoster(cizge, "noktalar_canvas");
        //Canvas üzerinde seçilen her ilin noktasını göstermek için

        if(cizge.dugumIdleriniAl().length > 1) //Seçilen il sayısı 1den büyükse
        {
            document.getElementById("algorithms").style.display = "inline";
            //En yakın komşu ve genetik için kullanılan katmanı aktif hale getir
            if(devamEdenGenetik != undefined)    //Değer ataması olmuşsa
                clearInterval(devamEdenGenetik); //Değişken üzerinde yapılan işlemleri iptal eder

            //Aşağıdaki iki metot sayesinde En yakın komşu
            //ve genetik algoritmalara işletilmesi için
            //graph nesnesi gönderilir
            devamEdenGenetik = genetikOlustur(cizge);
            enYakinKomsuOlustur(cizge);
        }

    }, false);
};
```

2. Ara metotların çalıştığı **Fonksiyonlar.js**

```
let yeniId = function(){ //Düğümleri gezmek amacıyla
    let id = 0;          //kullanılan bir metot
    return function(){return id++;}
}();

let egeImg = document.getElementById("ege");

//Rastgele düğümlerle çizge oluşturmak için
function cizgeOlustur(points){
    let cizge = new Cizge();
    for(let i = 0; i < points; i++){
        cizge.dugumEkle(Math.random() * 300, Math.random() * 300, yeniId());
    }
    return cizge;
}

//En yakın komşu ve genetik algoritma haritalarının
//yollarının gösterilmesi amacıyla kullanılan metot
function yoluGoster(yol, cizge, canvasId, baslikId, baslikEki){
    var ctx = document.getElementById(canvasId).getContext("2d");
    ctx.fillStyle = "white";
    ctx.fillRect(0, 0, 1000, 1000);

    ctx.drawImage(egeImg, 0, 0, 300, 300);

    ctx.strokeStyle = "black";

    ctx.fillStyle = "red";
    ctx.beginPath();
    ctx.moveTo(cizge.dugumler[yol[0]].x, cizge.dugumler[yol[0]].y);
    ctx.fillRect(cizge.dugumler[yol[0]].x - 10, cizge.dugumler[yol[0]].y - 10, 20, 20);

    ctx.fillStyle = "blue";
    for(let i = 1; i < yol.length; i++)
    {
        ctx.lineTo(cizge.dugumler[yol[i]].x, cizge.dugumler[yol[i]].y);
        ctx.fillRect(cizge.dugumler[yol[i]].x - 3, cizge.dugumler[yol[i]].y - 3, 6, 6);
    }
    ctx.lineWidth = 3;
    ctx.stroke();
```

```
    let mesafe = cizge.yolMesafesiniAl(yol);
    document.getElementById(baslikId).innerText = baslikEki + mesafe + " MESAFE";
}

//İlleri seçtiğimiz haritanın noktalarını göstermek için kullanılan metot
function noktalariGoster(cizge, canvasId){
    var ctx = document.getElementById(canvasId).getContext("2d"); // İçerik tipi belirtilerek nesne alınıyor.
    ctx.fillStyle = "white"; //Arka plan rengi seçmek için
    ctx.fillRect(0, 0, 1000, 1000); //İçi dolu olarak çiziliyor.

    ctx.drawImage(egeImg, 0, 0, 300, 300); //Boyutları belirtilmiş bir şekilde resim eklenir.

    if(cizge != undefined){
        let idler = cizge.dugumIdleriniAl(); //Canvastaki her bir noktanın id'si alınır

        if(idler.length > 0) //Başlangıç noktası için
        {
            ctx.fillStyle = "red";
            ctx.moveTo(cizge.dugumler[idler[0]].x, cizge.dugumler[idler[0]].y);
            ctx.fillRect(cizge.dugumler[idler[0]].x - 10, cizge.dugumler[idler[0]].y - 10, 20, 20);
        }

        ctx.fillStyle = "blue";
        for(let i = 1; i < idler.length; i++) //Diğer noktalar için
        {
            ctx.moveTo(cizge.dugumler[idler[i]].x, cizge.dugumler[idler[i]].y);
            ctx.fillRect(cizge.dugumler[idler[i]].x - 3, cizge.dugumler[idler[i]].y - 3, 6, 6);
        }
    }
}
```

3. Çizge oluşturulması amacıyla kullanılan **Cizge.js**

```
let Cizge = class{
    constructor(){
        this.dugumler = {};
    }

    //Düğüm id'lerini almak için
    dugumIdleriniAl(){
        let cikis = [];
        for(let id in this.dugumler)
            cikis.push(id);
        return cikis;
    }

    //Bir düğüm eklemek için
    dugumEkle(x, y, id){
        this.dugumler[id] = {x:x, y:y};
    }

    //Yol uzunluğunu almak için
    yolMesafesiniAl(yol){
        let mesafe = 0;
        for(let i = 0; i < yol.length - 1; i++)
            mesafe += this.mesafeyiAl(yol[i], yol[i + 1]);
        return mesafe;
    }

    //Genetik algoritmanın kullandığı mesafe hesaplama metodu
    mesafeyiAl(dugumId, digerId)
    {
        return this.dugumMesafeleriniAl(this.dugumler[dugumId], this.dugumler[digerId]);
    }

    //mesafeyiAl ve mesafeleriAl metotlarının istediği hesaplamaları yapan metot
    dugumMesafeleriniAl(a, b){
        let x = Math.abs(b.x - a.x);
        let y = Math.abs(b.y - a.y);

        return x + y;
    }
```

```
//En yakın komşu algoritmasının kullandığı mesafe hesaplama metodu
mesafeleriAl(dugumId){

    let cikis = [];
    for(let digerId in this.dugumler)
    {
        if(dugumId != digerId)
            cikis.push({id:digerId, distance:this.dugumMesafeleriniAl(this.dugumler[digerId], this.dugumler[dugumId])});
    }
    return cikis;
}
```

4. En Yakın Komşu Algoritması'nın işletilmesi amacıyla oluşturulmuş **EnYakinKomsu.js**

```
//En yakın komşu yöntemiyle yolu oluşturulur.
function enYakinKomsuOlustur(cizge){ //Metot, argüman olarak bir çizge nesnesi alır.
    let dugumIdleri = cizge.dugumIdleriniAl();
    //Gidilen her il bir düğüm
    //olarak bu değişkene atanır.
    let yolDugumleri = [dugumIdleri[0]];
    //Başlangıç noktası hangisiyse o düğümden başlanması için
    //yeni bir değişkene ilgili atama sağlanır.
    while(true) //Sonsuz döngü ama listede ne kadar düğüm varsa o kadar döner.
    {
        let mesafeler = cizge.mesafeleriAl(yolDugumleri[yolDugumleri.length - 1]);
        //Her bir şehir ve o şehirden önceki şehirler arası mesafelerin
        //hesabı için değişken oluşturulur ve ilgili atama yapılır.
        mesafeler.sort(function(a, b){ return a.distance - b.distance; });
        //Döngü anında hangi şehirdeyse, bir önceki şehirle
        //arasındaki mesafeyi sıralamak açısından büyük olanı
        //küçükten çıkarmak için kullanılan sıralama metodu.
        mesafeler = mesafeler.filter(function(a){return yolDugumleri.indexOf(a.id) == -1;})
        //Başlangıç noktası seçildikten sonra gidilecek illerin
        //noktalarını ekleyebilmek için kullanılan filtreleme metodu.

        if(mesafeler.length == 0) //Her düğüm dolaşıldıysa döngüden çık
            break;

        yolDugumleri.push(mesafeler[0].id); //Anlık olarak ilk düğümü yığına ekler
    }

    yolDugumleri.push(yolDugumleri[0]);
    //Döngü bittiğinde yığına tekrardan başlangıç noktasını ekler

    yoluGoster(yolDugumleri, cizge, "canvasEYK", "eyk_h1", "EN YAKIN KOMŞU YÖNTEMİ: ");
    //Canvasta mesafe gösterilir
}
```

5. Genetik Algoritması'nın işletilmesi amacıyla oluşturulmuş **Genetik.js**

```
//Başlamak için rastgele bir yol oluştur
function rastgeleYolOlustur(cizge){
    let dugumIdleri = cizge.dugumIdleriniAl();
    let yolDugumleri = [dugumIdleri.shift()]; //Baştan başla

    for(let i = 0; i < dugumIdleri.length; )
    {
        let a = Math.round(Math.random() * (dugumIdleri.length - 1));
        yolDugumleri.push(dugumIdleri[a]);
        dugumIdleri.splice(a, 1);
    }

    yolDugumleri.push(yolDugumleri[0]); //İlkiyle tekrar bitir

    return yolDugumleri;
}
```

```
//Yolu biraz mutasyona uğratın
function yoluMutasyonaUgrat(yol){
    let cikisYolu = yol.slice();
    let ilkVeSon = cikisYolu.pop();

    let mutasyonaUgrat = Math.ceil(Math.random() * ((yol.length - 1) / 50) + 1);
    mutasyonaUgrat = 1;

    for(let i = 0; i < mutasyonaUgrat; i++){
        let rastgeleIndis = Math.round(Math.random() * (cikisYolu.length - 1));
        let rastgeleDigerIndis = Math.round(Math.random() * (cikisYolu.length - 1));

        let eleman = cikisYolu[rastgeleIndis];
        cikisYolu.splice(rastgeleIndis, 1);
        cikisYolu.splice(rastgeleDigerIndis, 0, eleman);
    }

    //Başlangıç düğümüyle başla ve bitir
    cikisYolu.unshift(ilkVeSon);
    cikisYolu.push(ilkVeSon);

    return cikisYolu;
}
```

```
//Devam ettirmek için en uygun yolları bulun
function devamEdenYollariSec(yollar, cizge, adet){
    yollar.sort(function(a, b){return cizge.yolMesafesiniAl(a) - cizge.yolMesafesiniAl(b);});
    let ciftKayitOlmayan = [];
    for(let y in yollar)
        if(ciftKayitOlmayan.indexOf(yollar[y]) == -1)
            ciftKayitOlmayan.push(yollar[y]);
    return ciftKayitOlmayan.splice(0, adet);
}

//Yeni bir nesil yarat: YeniNesil = yavru -> mutasyon + ebeveynler -> seçim
function nesilOlustur(cizge, eskiNesil, turBasiYavru, turBasiGeriyeKalanlar, turBasiRastgele){
    let yeniNesil = [];
    for(let y = 0; y < eskiNesil.length; y++)
        for(let i = 0; i < turBasiYavru; i++)
            yeniNesil.push(yoluMutasyonaUgrat(eskiNesil[y]));
    for(let y = 0; y < eskiNesil.length; y++)
        yeniNesil.push(eskiNesil[y]); //Ayrıca aileleri yavrulara da ekleyin
    for(let i = 0; i < turBasiRastgele; i++)
        yeniNesil.push(rastgeleYolOlustur(cizge)); //Ayrıca rasgele yollar ekleyin
    let geriyeKalanlar = devamEdenYollariSec(yeniNesil, cizge, turBasiGeriyeKalanlar);
    return geriyeKalanlar;
}
```

```
//Genetik algoritmayı yapmak için ana program
function genetikOlustur(cizge){
    let nesilBasiYavru = 20;
    let turBasiGeriyeKalanlar = 20;
    let turBasiRastgele = 5;
    //İlk nesil bazı rastgele yollardır
    let nesil = [];
    for(let i = 0; i < turBasiGeriyeKalanlar; i++)
        nesil.push(rastgeleYolOlustur(cizge));
    //Nesiller boyunca devam eder
    let tur = 0;
    return setInterval(function(){
        nesil = nesilOlustur(cizge, nesil, nesilBasiYavru, turBasiGeriyeKalanlar, turBasiRastgele);
        console.log("Turun en iyisi " + tur++ + ": " + cizge.yolMesafesiniAl(nesil[0]));
        yoluGoster(nesil[0], cizge, "canvasGenetik", "genetik_h1", "GENETİK YÖNTEM: ");
    }, 1);
}
```

5. EKRAN ÇIKTISI