PROGRAMLAMA LABORATUVARI 2
1. PROJE

Yunus Emre GÜL
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
yunemregul@gmail.com

Özet

Bu doküman Programlama Laboratuvarı 2 dersi 1. Projesi için çözümümü açıklamaya yönelik oluşturulmuştur. Dökümanda projenin tanımı, çözüme yönelik yapılan araştırmalar, kullanılan yöntemler, proje sürecinde karşılaşılan problemler, proje hazırlanırken kullanılan geliştirme ortamı ve kod bilgisi gibi programın oluşumunu açıklayan başlıklara yer verilmiştir. Doküman sonunda projemi hazırlarken kullandığım kaynaklar bulunmaktadır.

1. Proje Tanımı

1.1. Proje Tanımı

Projede bizden istenen merkezi Kocaeli olan yeni bir kargo firmasının gidilecek hedef şehirleri için en kısa rotayı oluşturan ve gösteren bir program yapmamızdır. Rotaların başlangıç ve bitiş noktası Kocaeli şehri olmalıdır.

Problem ‘Gezgin Satıcı Problemi’ ile aynı sayılabilir. Gezgin satıcı problemi de verilen şehirlere gidip başlangıç noktasına geri dönen en kısa rotayı bulmayı hedefler.

Gezgin satıcı problemi, dolayısıyla bu problem, NP hard olarak bilinen problemlerdendir. NP hard problemler polinom zamanda çözülemez. Çözümleri için optimizasyon gibi yaklaşımlar kullanılmaktadır.

1.2. İsterler

Proje tanımında belirtilen kurallar ve isterler aşağıdaki gibidir:

• Tüm sevkiyatların başlangıç ve bitiş noktası Kocaeli olmalıdır.

• Tek bir seferde maksimum 10 şehire teslimat yapılabilmelidir.

• Bir şehirden diğer şehre giderken komşu şehirler üzerinden hareket edilmelidir.

• Bir şehirden birden fazla kez geçilebilir.

• Bulunan çözüm o problem için en kısa mesafeli yolu bulabilmelidir.

• Aramada bulunan alternatif en kısa yollar en az maliyetliden başlayacak şekilde sıralı olarak yazdırılmalıdır.

• Bulunan en kısa 5 çözüm görsel bir harita üzerinde çizdirilmelidir. Bunlar arasındaki en iyi yol farklı renkle belirgin bir şekilde vurgulanmalıdır.

2. Araştırmalar ve Yöntem

Projeye ilk olarak TSP’nin ne olduğunu, NP hard problemlerin ne demek olduğunu araştırmakla başladım. Araştırmalarımdan yola çıkarak en kısa yol üretecek olan algoritmanın hiç bir zaman tek seferde en kısa yolu veremeyeceğini anladım. Verebilecek olsa NP hard bir probleme çözüm getirmiş olurdum ki bu da büyük ödülü olan zor, muhtemelen imkansız bir iş.

Çözüm üretecek algoritma tek seferde çözüm üretemeyeceğine göre üretilen bir çözümü sürekli olarak iyileştirmeye dayanabilir. Bu iyileştirme işlemine de optimizasyon dendiğini ve buna yönelik hali hazırda bulunan karınca kolonisi optimizasyonu ve genetik optimizasyon gibi algoritmalar olduğunu öğrendim.

İlk başta iki şehir düğümü arasındaki en kısa yolun nasıl bulunacağını ve ne gibi algoritmalar olduğunu araştırdım. Bir süre Dijkstra algoritmasını kullanmayı düşündüm ancak daha sonra Dijkstra’nın geliştirilmiş hali olan A* algoritmasını keşfettim ve bunu kullanmaya karar verdim. İki şehir arasındaki en kısa yolu bulacağım algoritma belliydi ancak verilen şehirler arasındaki en kısa rotayı oluşturacak başka bir yaklaşıma da ihtiyacım vardı. Bu da optimizasyon gibi algoritmalarla yapılıyor.

Herhangi bir optimizasyon algoritmasını denemeden önce uzun bir süre probleme nasıl çözümler getirildiğini ve belki de bize verilen problemde gezgin satıcı probleminde olmayan bir kolaylık olabileceğini düşündüm ve araştırdım. Böyle bir kolaylığı bulamasam da bu araştırmaları yaparken ilk karşılaştığım, en kolay algoritma, TSP’ye her zaman doğru olmamakla beraber kimi durumlarda anında çözüm getiren açgözlü ‘en yakın komşu’ rota bulma algoritmasıydı.

En yakın komşu algoritması çok basit bir yaklaşımla başlanan şehire en yakın komşuya, daha sonra gidilen komşunun en yakın komşusuna giderek ve teslimat adresleri bitene kadar böyle devam ederek bir rota oluşturuyor. Oluşturulan rota açgözlü yaklaşıma dayandığı için ilk başta yapılan açgözlü seçimler daha sonra daha büyük sorunlara yol açabiliyor. Başlarda bu algoritmayı kullandım ve nasıl geliştirebileceğimi düşündüm.

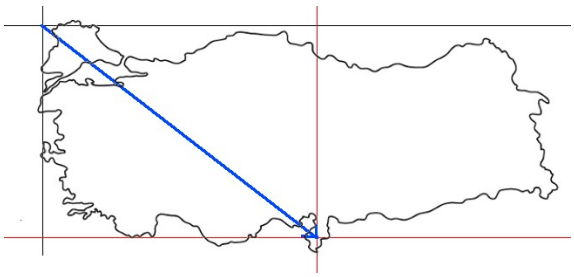
Bir süre 'en yakın komşu' algoritmasını geliştirip optimizasyon algoritmalarına hiç bulaşmadan problemin çözülebilir olacağını düşünsem de bir çözüm üretemedim. Optimizasyon algoritmalarını araştırdıktan sonra uygulaması kolay bir algoritma olan genetik algoritmayı uygulamayı seçtim.

Sadece genetik algoritmayı kullanmaktansa ilk başta bulduğum açgözlü en yakın komşu algoritmasını da genetik algoritmayla birleştirmenin iyi bir fikir olacağını düşündüm. Böylece kimi durumlarda anında en iyi çözümü bulabilen bir algoritmam, hatalı bir çözüm bulduğunda da daha iyi çözümleri araştıran genetik algoritmam olmuş oldu.

Şehirler arasındaki en kısa yolu ve teslimat adresleri arasındaki en kısa rotayı oluşturabileceğim belli olduktan sonra kalan ekrana nasıl çizdireceğimdi. Bunu da Swing kütüphanesi kullanarak yapmaya karar verdim.

2.1. Karşılaşılan Problemler

Çizim işlemleri için başladığımda karşılaştığım ilk problem 81 tane şehrin nasıl konumlandırılacağıydı. Şehirlerin enlem boylamlarını kullanmaya karar verdim. Buna göre şehirlerin enlem boylamının Türkiye'nin enlem boylamına göre olan konumunu ekrandaki piksel konumu olarak belirledim. Şekil 1' de Hatay ili için ekrandaki konumun nasıl belirlendiği görülebilir. Siyah çizgiler Türkiye'nin enlem-boylamı, kırmızı çizgiler Hatay'ın enlem boylamı, Hatay'ın konumundan Türkiye'nin konumunun çıkarılması mavi ok olarak kabul edilir. Çizim yaparken Türkiye'nin konumu ekranın sol üst köşesi olarak kabul edilir ve bu mavi vektöre göre hatay çizilebilir.



Şekil 1. Enlem-boylama göre ekrandaki konum belirleme

Karşılaştığım bir diğer problem de aynı anda hem çizim hem de rota üzerinde genetik algoritmayla optimizasyon yapınca kodun çalışması çizimi donduruyordu. Bunun sebebi de bir threadda aynı anda iki işlem yapılamamasından kaynaklanıyor. Yani hem çizimi yenileyip hem de aynı anda optimizasyon için döngü çalıştırmak olmuyordu. Bunun için optimizasyon yapan kodu farklı bir threadda çalıştırmaya karar verdim. Böylece aynı anda hem çizim yapıp hem de bir döngü çalıştırabiliyorum. Kullanıcıya da kesintisiz bir arayüz kontrolü sağlamış oluyorum.

Karşılaştığım problemlerden bir tanesi de kendi yazdığım bir kodun çok yavaş olmasıydı. Bu kod verilen iki şehir arasındaki mesafeyi döndürüyordu. Mesafeyi bulabilmek için tüm şehirlerin mesafesini gezmek gibi uygun olmayan bir yöntem kullanmıştım. Bu da 81x81 lik bir mesafe matrisini çok sık şekilde gezmek anlamına geliyordu. Projemin bundan dolayı biraz yavaş olduğunu projem bittiğinde VisualVM ile takip edince farkettim. Bu kısmı tüm diziyi gezmek yerine bir HashMap kullanıp anında mesafeye ulaşarak çözdüm. Bunu yapmam programı yaklaşık 20 kat hızlandırdı.

Multi-threading çözümünü kullanınca gelen bir sorunda bu oluşturduğum threaddan gelecek cevapları nasıl çizim threadına alacağım oldu. Örneğin optimizasyon yapan thread yeni bir en kısa yol bulduğunda bu yolu çizim yapan threadda gösterebilmem gerekti. Bu yüzden de 'observer pattern' olarak da bilinen bir yöntem kullandım. OptimizerListener isminde bir sınıf oluşturdum bu sınıfı rota optimize eden threaddan gelecek cevapları dinlemek amacıyla kullandım.

2.2. Kazanımlar

Bu projeden öncesinde genetik algoritma gibi algoritmaların yapay zeka uygulamaları gibi göreceli olarak karışık olacağını düşünüyordum. Ancak bu projeyle birlikte genetik algoritmanın kısmen rastgelelikle gelişim sağladığını ve uygulamasının kolay olduğunu öğrendim.

Yine bu projeden öncesinde duysam da hiç multithreading kullanmamıştım. Bu proje sayesinde hem öğrenmiş hem de uygulamış oldum.

Bu projeyi C gibi bir dilde yapacak olsam muhtemelen bağlı liste gibi yapılar kullanılması gerekirdi. Bağlı liste gibi yapıların java sınıflarıyla da temsil edilebileceğini deneyimlemek nesneye yönelik programlama hakkında gelişimime ve fikirlerime katkı sağladı.

Projeyle birlikte daha önce kullanmadığım HashMap, HashSet ve PriorityQueue gibi çeşitli veri yapılarını öğrenmiş oldum.

3. Geliştirme Ortamı

Projemi Linux sistemde, Intellij IDEA üzerinde geliştirdim. Projemin gelişimini ve versiyonlarını takip edebilmek için de Git versiyon kontrol sistemi kullandım.

Projemin hangi kısımda daha çok vakit harcadığını tespit edip o kısımları iyileştirmek için VisualVM kullandım. Detaylı bilgi 2.1. bölümde bulunabilir.

4. Kod Bilgisi

4.1. Programın Çalıştırılması

Kısım ektedir. [1](readme.pdf)

4.2. Algoritma

Bu kısımda projenin genel algoritmasına açıklık getireceğim.

Projeye çözüm getirmek için 2 farklı algoritma kullanacağıma karar verdim. Birinci algoritma iki şehir arasındaki en kısa yolu bulacak, ikincisi de teslimat şehirleri arasındaki en kısa rotayı oluşturmaya yönelik olacaktı. İkinci algoritma için 2. bölümde de belirttiğim gibi en yakın komşu algoritması ile genetik algoritmayı birleştirerek kullandım.

Kullanıcı arayüzden belirli şehirleri seçip rota bulmayı başlattıktan sonra ilk önce açgözlü en yakın komşu algoritması ile seçilen şehirler arasında bir rota oluşturuluyor. Yine 2. bölümde belirttiğim gibi bu yol açgözlü yaklaşım kullandığı için her zaman en kısa yol değil. Zaten en kısa yol olmasını da beklemiyoruz. Bu algoritma ile bulduğumuz yolu daha sonra genetik algoritma ile geliştiriyoruz. Böylece geliştirilecek yolumuz en baştan ya en iyisi ya da yine de iyi bir yol olmuş oluyor. Geliştirme işleminin böyle daha hızlı olabileceğini düşündüm.

Birinci algoritmam olan iki şehir arasındaki en kısa mesafeyi bulan A* algoritması Dijkstra algoritmasının geliştirilmiş halidir. Dijkstra algoritmasına ek olarak komşulardaki gidilecek şehre karar verirken hedef şehre uzaklığını da değerlendirir. Böylece hedefe en yakın komşuya gitmiş olur. Hedef yönünde gitmeye çalışması da hızlı olmasını sağlıyor. Hedef yönünde gitmeyi sağlamak için komşulardaki gidilecek şehrin hedef şehre olan mesafesini de bilmek gerek. Yani bu algoritmayı kullanmak için her şehrin her şehre olan mesafesini bilmek ya da birbiriyle orantılı olacak şekilde tahmin edebilmek gerekir.

Başlarda bu tahmini iki şehirin enlem boylamına göre enlem boylamlar arasındaki mesafeyi bularak yaptım ama bu çok iyi sonuçlar vermiyor. Çünkü enlem boylamlar arasındaki mesafe kuşuçuşu ve bazen gerçekle orantılı olmuyor. Kimi şehir gerçekte daha uzakken daha yakın çıkabiliyor. Bu yüzden enlem boylama göre mesafe bulmayı bırakıp gerçek KGM mesafelerini kullanmaya karar verdim.

Genetik algoritma da canlıların doğadaki gelişimini baz alıyor. Canlılar nasıl çiftleşip gen çaprazlama ve mutasyon ile yeni kombinasyonlar oluşturuyor ve doğal seçilimle en iyileri hayatta kalıyorsa, genetik algoritma da aynen bunları yapıyor. Çalışma adımları şu şekilde:

- Nüfusu tercihe göre belirlenmiş bir başlangıç popülasyonu tanımlanır.

Bu proje için popülasyon bir rotayı temsil etmektedir. Örneğin 100 nüfuslu bir popülasyon 100 tane rota demektir. Projemdeki ilk popülasyon açgözlü en yakın komşu algoritmasının bulduğu yolun rastgele mutasyona uğratılmasıyla oluşturuluyor. Projemde deneme yanılma ile uygun popülasyon nüfusunun 300 olduğuna karar verdim ancak belki daha iyi rakamlar bulunabilir. Bu 300 değeri başlangıçta 300 tane mutasyonlu rota olduğunu ve her nüfusta 300 tane rota olduğu anlamına gelmektedir.

- Popülasyondaki her rota için o rotanın toplam mesafesi alınır ve o rotanın ‘fitness’ değeri olarak belirlenir.
- Yeni bir jenerasyon oluşturmaya başlanır. Yeni jenerasyonun üyeleri önceki jenerasyondaki en fit yani en iyi rotalardan rastgele seçilenler mutasyona ve çaprazlamaya uğratılarak oluşturulur.

Böylece yeni oluşturulan popülasyon önceki popülasyonun en iyilerinin hayatta kalmasıyla oluşmuş olur. Bu iki işlem sürekli tekrar edilerek rota istenen süre boyunca iyileştirilir.

4.3. İstatistik

Program kodu boşluksuz ve yorumsuz yaklaşık 893 satırdan oluşmaktadır. Kod düzenini sağlamak için yaklaşık 189 boş satır kullanılmıştır. Okuyucuya izlenim oluşturması için yaklaşık 221 yorum satırına yer verilmiştir.

Kaynakça

1. Gezgin satıcı problemi hakkında genel bilgi ve çözüm yöntemleri: https://en.wikipedia.org/wiki/Travelling_salesman_problem
2. En yakın komşu algoritması: https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

3. Genetik optimizasyon uygulanışı ve mantığı: https://youtu.be/hnxn6DtLYcY
4. Java'da A* algoritması uygulamak: https://www.baeldung.com/java-a-star-pathfinding
5. Java HashMap, HashSet gibi veri tipleri: https://www.w3schools.com/java/java_hashmap.asp https://www.javatpoint.com/java-hashset
6. Java Multithreading işlemleri: https://www.geeksforgeeks.org/multithreading-in-java/
7. Karşılaştığım çeşitli diğer problemler: https://stackoverflow.com/