TREN SEFER PLANLAMASI İYİLEŞTİRME

Ömer ÜSTÜN
Mustafa Emir ALBAYRAK

HAVELSAN EKSPRES PROJESİ

Danışman
S. Çağatay YİĞİTER

Mayıs, 2022

TEŞEKKÜR

Bizi Havelsan Ekspres hackathonunda yarışma şansına layık gördükleri için organizasyon ekibine, yarışma boyunca sorularımızı cevaplayan mentörümüz S. Çağatay YİĞİTER'e, böyle kapsamlı ve güzel bir etkinlik için de Havelsan'a teşekkür ederiz.

Ömer ÜSTÜN
Mustafa Emir ALBAYRAK

İÇİNDEKİLER

ŞEKİL LİSTESİ

ÖZET

Tren Sefer Planlaması İyileştirme

Ömer ÜSTÜN
Mustafa Emir ALBAYRAK

Havelsan Ekspres Projesi

Danışman: S. Çağatay YİĞİTER

Ulaştırma alt sistemlerinden biri olan demiryolu, diğer ulaştırma alt sistemleriyle yoğun bir rekabet halinde bulunmaktadır. Yanlış planlamalar sonucu ülkemizde demiryolu ulaştırmasına olan talep, yolcu ve yük taşımacılığında karayolunun oldukça gerisinde kalmıştır. Demiryolunun pazar payını arttırması ve rekabetini devam ettirebilmesi için hizmet kalitesini arttırması gerekmektedir. Dakiklik ve güvenilirlik bir ulaştırma alt sisteminin kalitesini belirleyen ölçütlerin başında gelmektedir. Bu ölçütlerin istenilen seviyede tutulabilmesi de kısmen etkin trafik kontrolü ile sağlanabilir.

Trenler önceden hazırlanmış bir hareket planına uygun biçimde hareket etmektedir. Ancak beklenmedik bazı olayların gerçekleşmesi sonucu gecikmeler ve trenler arası çatışmalar meydana gelebilmektedir. Trafik kontrolü, trenler arası çatışmaları, gecikmeleri mümkün olduğunca azaltacak şekilde çözüp, yeni bir uygulanabilir çizelge hazırlamak için uygulanır. Problemin zorluk derecesi nedeniyle, problemin en az gecikme içeren çözümüne kabul edilebilir bir süre içerisinde ulaşılması imkânsızdır. Bu çalışmada, 5 dakika gibi kısa bir süre içerisinde uygulanabilir ve gecikme toplamının olabildiğince asgariye indirildiği bir çizelge hazırlamak için, genetik algoritmalar kullanılmıştır. Geliştirilen algoritmanın çözümleri, kesin dispeçer çözümleri ile karşılaştırıldığında, algoritma kısa sürede yeteri kadar iyi sonuçlar vermektedir. Algoritmanın uygulanması için geliştirilen bilgisayar programı, tren dispeçerleri için bir karar destek sistemi olarak da kullanılabilir.

Anahtar Kelimeler: Demiryolu trafik kontrolü, trenlerarası çatışmalar, yeniden çizelgeleme, genetik algoritmalar, yapay sinir ağları

ABSTRACT

Train Schedule Optimisation

Ömer ÜSTÜN
Mustafa Emir ALBAYRAK

Havelsan Ekspres Project

Advisor: S. Çağatay YİĞİTER

Railway, one of the transportation subsystems, is in intense competition with other transportation subsystems. As a result of the inaccurate planning, the demand for railway transportation in our country has lagged far behind the highway in passenger and freight transportation. The railway needs to increase its service quality in order to increase its market share and maintain its competition. Punctuality and reliability are the primary criteria that determine the quality of a transport subsystem. Keeping these criteria at the desired level can be partially achieved by effective traffic control.

Trains move in accordance with a pre-prepared action plan. However, as a result of some unexpected events, delays and conflicts between trains may occur. Traffic control is applied to resolve train-to-train conflicts in such a way as to reduce delays as much as possible and to prepare a new viable schedule. Due to the difficulty level of the problem, it is impossible to reach the solution with the least delay within an acceptable time. In this study, genetic algorithms are used to prepare a schedule that can be implemented in as little as 5 minutes and in which the sum of the delays is as small as possible. When the solutions of the developed algorithm are compared with the exact dispatcher solutions, the algorithm gives good enough results in a short time. The computer program developed for the implementation of the algorithm can also be used as a decision support system for train dispatchers.

Keywords: Railway traffic control, conflicts between trains, re-scheduling, genetic algorithms, neural networks

1
PROBLEM TANIMI

"Yol Optimizasyon Problemi" başlıklı PDF dosyasında teslim edilen problemi parçalarına ayırmak ve bu şekilde ilerlemek daha planlı ve verimli olacaktır.

1.1 Girdiler

Bilgi olarak üç tip trenin çeşitli özellikleri ve bu trenlerin çalıştıkları güzergahlar verilmiştir. Verilen tabloların sırasının birkaç yerde farklı olmasından kaynaklanabilecek kafa karışıklıklarını önlemek için tablolar yeniden düzenlenmiştir.

Her bir tip trenin sayısı ve istenen sefer planının gün olarak uzunluğu ise program başında kullanıcıdan girdi olarak alınacaktır.

1.2 Çıktılar

Girdilere karşılık tüm trenlere tren kodu atanması ve sefer planı tablolarının üretilmesi istenmiştir.

1.3 Zorluklar

Problemin konvansiyonel algoritmalarla kesin çözülmesinin zor olduğu aşikardır. Standart algoritmalar ile ancak kesin çözüme yaklaşılabilir.

Verilen süre de muntazam bir çözüm yetiştirmek için zorlayıcı olmuştur.

Problem PDF dosyasındaki tabloların tek formatta olmaması da anlaşılabilirliği düşürmektedir.

2
YAKLAŞIM

Çözüm için bütünleşik tekil bir kod yerine modüler yapılı fonksiyonel bir yaklaşım izlenmesine karar verilmiştir. Kullanım kolaylığı ve yorumlanan programlama dili olması sebebiyle derleme gecikmesi oluşturmamasından dolayı yazılım dili olarak Python tercih edilmiştir.

2.1 Kâr Optimizasyonu İçin Ağırlık Hesabı

Çakışma durumlarında hangi hatta öncelik verileceğine karar verilebilmesi için hatların kârlılıklarının karşılaştırılabilmesi gerekmektedir. Problemin birincil hedefi azami kâr olduğu için bu önceliklerin belirlenmesi önem arz etmektedir. Tren tipleri tüm ilgili parametreleri göz önüne alınarak sıralanmalıdır.

Bu sıralamayı yapabilmek için her hattın bir genel bakımlık süresi boyunca ne kadar kazanç sağladığı ve ne kadar zamanda bu kazancı oluşturduğu göz önünde bulundurulmuştur. Bu doğrultuda aşağıdaki genel ağırlık formülü türetilmiştir.

(GenelBakimKM/HatUzunlugu)((HatUzunlugu/TrenHizi) +(DurakSayisi*DuraktaBeklemeSuresi)) + SeferArasiBakimSuresi*

Verilen trenler bu formül ile incelendiğinde uzun vadede işletilmesi en fazla kârlıdan en az kârlıya doğru (i)ana hat treni, (ii)hızlı tren ve (iii)yük treni olduğu görülmüştür. Bu sebeple çakışma durumlarında öncelik bu sıraya göre verilecektir. Yük treni işleyiş kuralı itibariyle duraklarda bekletilemediğinden başlangıç durağındaki kalkış saati ileri alınarak geciktirilecektir.

2.2 Süre İşlemleri

Problemde farklı formatta verilen sürelerin kafa karıştırmaması ve süre işlemlerinin hatalı çıkmaması için girdi ve çıktı olarak kullanılan tüm sürelerin dakika olarak tutulması ve işleme sokulması kararlaştırılmış ve program da buna göre yazılmıştır.

Lakin sefer tablolarındaki zamanların insan-okuyabilir formatta olması gerektiğinden sefer tabloları çıktı verilirken bu sürelerin tekrar saat formatına dönüştürülmesi gerekmiştir.

Bunun için girdi olarak dakika alıp çıktı olarak zaman damgası veren **zam_dam** fonksiyonu oluşturulmuştur. Bu fonksiyon gereken tüm yerlerde kolayca çağrılabilmesi için harici bir dosya olarak depolanmaktadır.

2.3 Kütüphaneler

Kullanılan kütüphaneleri mümkün olduğunca dahili kütüphaneler ile sınırlayarak ve harici kaynaklara başvurmayarak programın genel yükünün ve disk ayak izinin asgari olmasına çalışılmıştır.

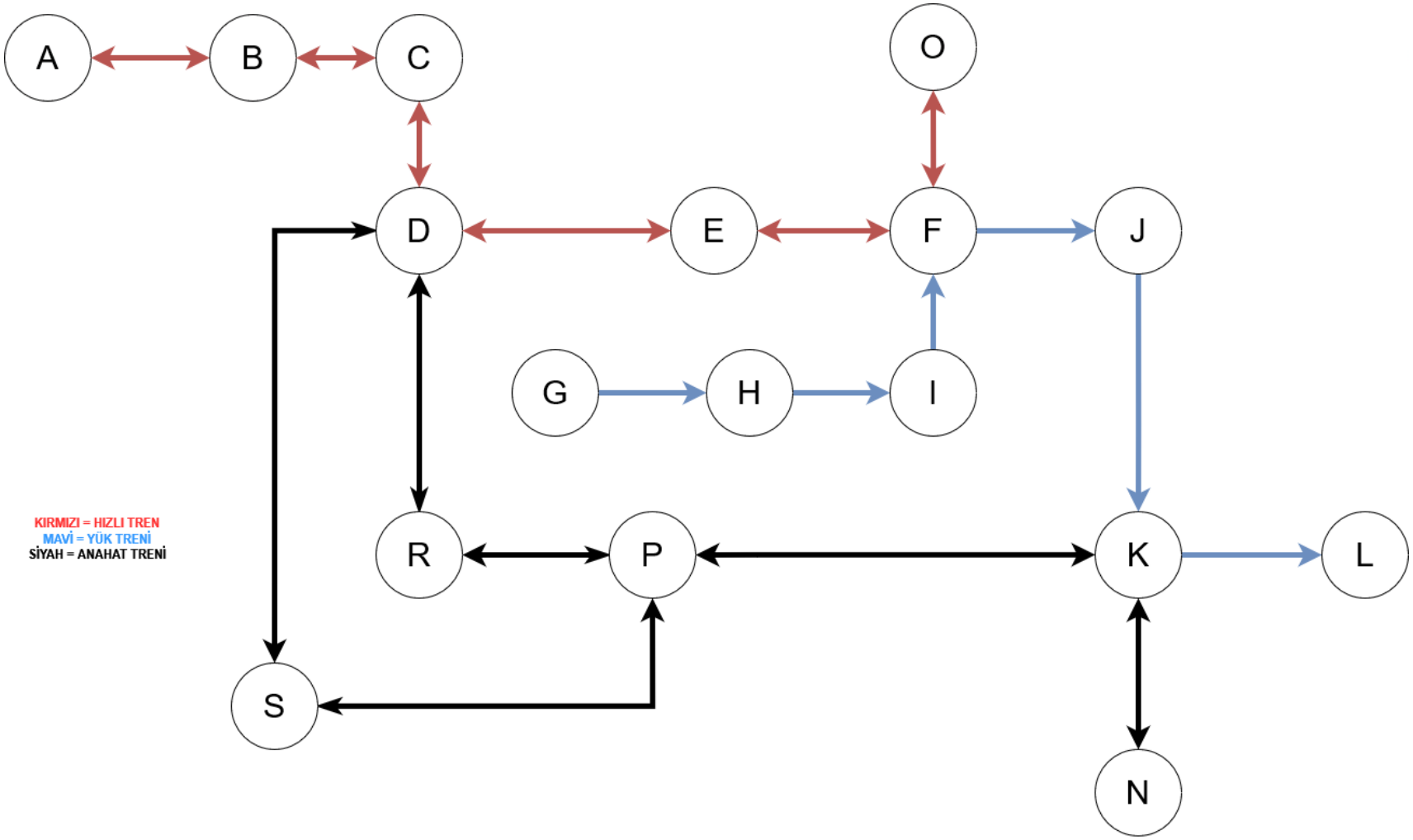
JSON dosyaları ile işlem yapılabilmesi için **json** ve programın yerel dosya dizinlerine erişebilmesi için **os** kütüphaneleri kullanılmıştır.

Çeşitli verilerin depolanması için JSON (JavaScript Object Notation) tercih edilmiştir. JSON'ın dahili anahtar yapısı istenen veriye rahatça erişilmesini kolaylaştırmaktadır.

Bunlar haricinde sadece dahili döngüler kullanılmıştır.

2.4 Çakışma Önleme

Olası çakışmaların önlenmesi için **timedelta** uygulanması uygun görülmüştür. Kesişim istasyonlarında çakışma olduğu durumda hatların ağırlıklarına ve tiplerine göre ya önceki duraktaki bekleme süreleri uzatılır ya da ilgili seferin başlangıç zamanı geciktirilir.



Şekil 2.1 Tren sisteminin şeması

3
MODÜLLER

Program, spesifik işleri yapan ve birbirleri arasında değişkenler aracılığıyla veri aktaran modüllere bölünmüştür.

3.1 Tren Etiketi

etiket_olustur olarak isimlendirdiğimiz bu fonksiyon girdi olarak her tipten trenlerin sayısını alır ve **trains.json** dosyasına her biri için tip etiketi ve çift haneli belirteç numarası ile kaydedilir. Örnek olarak birinci hızlı trenin kodu **HT01** olacaktır.

```
def etiket_olustur(HTsayisi, YTsayisi, ATsayisi):
    TrainCount = 1
    trains = 'trains.json'
    if path.isfile(trains) is True:
        os.remove(trains)
        print("Eski trenler kaldirildi!")
    if path.isfile(trains) is False:
        with open(trains, 'w') as f:
            json.dump([], f, indent=4)
            print("Yeni trenler olusturuldu!")
    with open(trains, 'r+') as f:
        data = json.load(f)
        for i in range(HTsayisi):
            x = {
                "id": TrainCount,
                "type": "speed",
                "label": "HT-" + "%02d"%(i+1)
            }
            data.append(x)
            TrainCount += 1
        for i in range(YTsayisi):
            x = {
                "id": TrainCount,
                "type": "freight",
```

```
                "label": "YT-" + "%02d"%(i+1)
            }
            data.append(x)
            TrainCount += 1
        for i in range(ATsayisi):
            x = {
                "id": TrainCount,
                "type": "mainline",
                "label": "AH-" + "%02d"%(i+1)
            }
            data.append(x)
            TrainCount += 1
        f.seek(0)
        json.dump(data, f, indent=4)
```

3.2 Sefer Çizelgesi

Her bir trenin sefer çizelgesi ayrışık olarak oluşturulmaktadır. Tren tipleri için ayrı parametrelere sahip sefer çıkarma fonksiyonları yazılmıştır. Bu fonksiyonlar her bir trenin tipini, tipinden kaynaklanan bekleme ve bakım sürelerini ve sefer başlangıç sırasını dikkate alarak çakışmasız seferler oluştururlar.

Örnek olarak hızlı tren seferi çıkartmak için kullandığımız **HTcikart** fonksiyonu aşağıda verilmiştir.

Bu fonksiyon hızlı trenlerin hızını, doldurulması istenen toplam sefer süresini, başlangıçta art arda çıkarılacak olan seferlerin arasında olması istenen beklemeyi, tren kodunu, duraktaki bekleme süresini, genel bakıma girilmesi gereken KM sınırını, genel bakım süresini ve sefer sonrası bakım süresini girdi olarak alır.

```
def HTcikart(HT_max_hiz, sefer_suresi, gecikme, tren,
   durak_suresi, genel_bakimKM, genel_bakim, sefer_bakim):
    gecen_sure = 0 + gecikme
    gidilenKM = 0
    sefer = 0
    genel_bakim_dk = genel_bakim*60
    sefer_bakim_dk = sefer_bakim*60
    while gecen_sure <= sefer_suresi:
        if gidilenKM <= genel_bakimKM:
            if ((sefer)%2) == 1:
                sefer += 1
                print("Tren: " + tren + " A-O Guzergahi")
                print("Gun " + str((gecen_sure//1440)+1) + " -
   " + "Sefer " + str(sefer))
```

```
                for key in HT_route:
                    if HT_route[key] != 0:
                        print(key, HT_route[key], HT_max_hiz,
zam_dam(gecen_sure), zam_dam(gecen_sure + durak_suresi),
durak_suresi)
                        gecen_sure += durak_suresi + (
HT_distances[key]/HT_max_hiz)*60
                    else:
                        print(key, HT_route[key], HT_max_hiz,
gecen_sure, gecen_sure, durak_suresi)
                    gidilenKM += HT_distances[key]
            elif ((sefer)%2) == 0:
                sefer += 1
                print("Tren: " + tren + " O-A Guzergahi")
                print("Gun " + str((gecen_sure//1440)+1) + " -
" + "Sefer " + str(sefer))
                for key in HT_route_rev:
                    if HT_route_rev[key] != 0:
                        print(key, HT_route_rev[key],
HT_max_hiz, zam_dam(gecen_sure), zam_dam(gecen_sure +
durak_suresi), durak_suresi)
                        gecen_sure += durak_suresi + (
HT_distances_rev[key]/HT_max_hiz)*60
                    else:
                        print(key, HT_route_rev[key],
HT_max_hiz, gecen_sure, gecen_sure, durak_suresi)
                    gidilenKM += HT_distances[key]
            print("Sefer bakimi zamani, 2 saat bekleme")
            gecen_sure += sefer_bakim_dk
            print("Gidilen KM: " + str(gidilenKM))
            print("\n")

        elif gidilenKM > genel_bakimKM:
            print(tren + " Genel bakim zamani, 12 saat bekleme"
)
            print("\n")
            gecen_sure += genel_bakim_dk
            gidilenKM = 0
     return sefer
```

3.3 Zaman Damgası Oluşturulması

Zaman damgası oluşturma fonksiyonu tek girdi alır. Aldığı dakika sayısına ilk önce 1440 ile mod işlemi uygulayarak tam günleri eksiltir ve işlem hatasının önüne geçer. Saati bulmak için de kalan dakikaya 60 ile kalansız bölme uygular. Son olarak bu parçaları birleştirip **zaman_damgasi** değişkeninde depolayarak döndürür.

```
def zam_dam(dakikalar):
    dakika = floor(dakikalar%1440)
    toplam_saat = dakika // 60
    toplam_dakika = "%02d"%(dakika%60)
    zaman_damgasi = "{}:{}".format(toplam_saat, toplam_dakika)
    return zaman_damgasi
```

4
SONUÇ

Bu projede üç tip trene sahip tren hatlarının istenen süre zarfı için tren zaman çizelgesi oluşturabilmesine olanak sağlayan bir program oluşturulmuştur.

Gelecek iyileştirmelerde çakışma önleme algoritması geliştirilebilir, çıktı olarak verilen çizelgelerin PDF veya Excel tablosu olarak kaydedilme seçeneği eklenebilir ve son olarak da bir API oluşturularak tren süre ve yer sorgusu sağlayan bir sistem kurulabilir.