FEN BİLİMLERİ ENSTİTÜSÜ

TREN FRENLEME ENERJİSİNİN MAKSİMUM GERİ KAZANIMI İÇİN ZAMAN-PLANI OPTİMİZASYONU

Büşra TURAL

Danışman
Dr. Öğr. Üyesi Metin TURAN

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
İSTANBUL - 2020

KABUL VE ONAY SAYFASI

Büşra TURAL tarafından hazırlanan **" Tren Frenleme Enerjisinin Maksimum Geri Kazanımı İçin Zaman-Planı Optimizasyonu "** adlı tez çalışması 13/07/2020 tarihinde aşağıdaki jüri üyeleri önünde başarı ile savunularak, İstanbul Ticaret Üniversitesi Fen Bilimleri Enstitüsü **Bilgisayar Mühendisliği Anabilim Dalı**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman	**Dr. Öğr. Üyesi Metin TURAN** İstanbul Ticaret Üniversitesi	.....
Jüri Üyesi	**Dr. Öğr. Üyesi Alper ÖZPINAR** İstanbul Ticaret Üniversitesi	.....
Jüri Üyesi	**Doç Dr. Muhammed Ali AYDIN** İstanbul Üniversitesi	.....

Onay Tarihi : 24/07/2020

İstanbul Ticaret Üniversitesi, Fen Bilimleri Enstitüsünün 24.07.2020 tarih ve 2020/288 numaralı Yönetim Kurulu Kararının 1. maddesi gereğince, ders yüklerini ve tez yükümlülüğünü yerine getirdiği belirlenen "Büşra TURAL" (TC:53233624604) adlı öğrencinin mezun olmasına oy birliği ile karar verilmiştir.

Prof. Dr. Necip ŞİMŞEK
Enstitü Müdürü

AKADEMİK VE ETİK KURALLARA UYGUNLUK BEYANI

İstanbul Ticaret Üniversitesi, Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- Tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- Görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- Başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- Atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- Kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- Ve bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

01.06.2020

Büşra TURAL

İÇİNDEKİLER

Sayfa

ÖZET

Yüksek Lisans Tezi

TREN FRENLEME ENERJİSİNİN MAKSİMUM GERİ KAZANIMI İÇİN ZAMAN-PLANI OPTİMİZASYONU

Büşra TURAL

İstanbul Ticaret Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Metin TURAN

2020, 69 sayfa

Bu çalışmada, Metro İstanbul araçlarından zaman planı uyarlanarak maksimum enerji kazanımının optimize edilmesine yönelik araştırma sonuçları paylaşılmıştır. Yeniden enerji kazanımı (rejeneratif enerji), elektromanyetik frenleme yapan trenlerin ürettiği enerjiyi hatta hareket etmeye hazır durumunda bulunan diğer trenlere aktarması prensibine dayanmaktadır. Trenleri ivmelendiren enerji, kinetik ve potansiyel enerji olarak depolanır. Frenleme esnasında kinetik enerji elektrik enerjisine dönüştürülür, bu enerji geri iletilerek katenere verilir. Alıcı durumunda bulunan trenin de bu enerjiyi kullanmasıyla enerjinin geri kazanımı gerçekleştirilmiş olur. Yeniden enerji kazanımı elde etmenin en etkili yollarından birisi, trenlerin istasyonlarda bekleme sürelerinde düzenleme yaparak zaman-planı en iyileştirmesinin gerçekleştirilmesidir. Bunu sağlayacak bekleme sürelerini bulmak için genetik algoritma kullanılmıştır. Genetik algoritmalar, evrimsel sürece benzer şekilde çalışan arama ve en iyileştirme yöntemidir. Bu yöntem çok boyutlu ve karmaşık uzayda en iyinin hayatta kalması ilkesine göre en iyi çözümü aramaya dayanır. Her tekrar sonunda en iyi birkaç elit birey bir sonraki nesle aktarılmıştır. Her tekrarda toplam birey sayısı sabit tutulmuş, diğer bireyler ise elit bireylerin çaprazlanması sonucu veya rastgele üretilmesiyle oluşturulmuştur. Agresif mutasyon işlemi, istasyon bekleme sürelerindeki değişimin sıfıra eşit olmadığı durumlarda uygulanmıştır. Yapılan simülasyon sonucunda, referans çalışmaya göre (%60 rejeneratif kazanım) %30 civarında daha iyi sonuç elde edilmiştir (%78 rejeneratif kazanım)

Anahtar Kelimeler: Genetik algoritma, metro zaman-planı, optimizasyon, yeniden enerji kazanımı.

ABSTRACT

M.Sc. Thesis

TIME-PLAN OPTIMIZATION WİTH GENETIC ALGORITHM FOR REGAIN OF ENERGY FROM TRAIN TRACKS

Büşra TURAL

İstanbul Commerce University
Graduate School of Applied and Natural Sciences
Department of Computer Engineer

Supervisor: Asst. Prof. Dr. Metin TURAN

2020, 69 pages

In this study, the research results for optimizing the maximum energy gain are shared by adapting the time plan of Metro Istanbul vehicles. Regenerative energy recovery is based on the principle that energy produced by the trains which make electromagnetic brake is transferred to the other trains that are ready to move. One of the ways to re-energize is to arrange the waiting times of the trains at the stations and to realize the time-plan optimization. Genetic algorithm was used to find station dwell times. Genetic algorithms are search and optimization methods that work similarly to the evolutionary process. This method is based on seeking the best solution according to the principle of survival of the best in multi-dimensional and complex space. At the end of each repetition, several of the best elite individuals were transferred to the next generation In each repetition, the number of society individuals has been kept constant, while other individuals have been formed by crossing elite individuals or producing them randomly. Aggressive mutation was applied in cases where the change in station waiting times was not equal to zero. Result in of the simulation, around 26% better result compared to the reference study (60% regenerative energy recovery) was obtained (78% regenerative energy recovery).

Keywords: Genetic algorithm, metro time-table, optimization, regain of energy.

TEŞEKKÜR

Bu araştırma için beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan değerli Danışman Hocam Dr. Metin TURAN'a teşekkürlerimi sunarım.

Araştırmanın yürütülmesinde maddi ve manevi yardımlarını gördüğüm Sn. İbrahim Ethem DEMİRCİ olmak üzere tüm Metro İstanbul personeline teşekkür ederim.

Tezimin imalat aşamasındaki desteklerinde dolayı Metro İstanbul A.Ş. şirketine teşekkür ederim.

Tezimin her aşamasında beni yalnız bırakmayan aileme ve arkadaşlarıma sonsuz sevgi ve saygılarımı sunarım.

Büşra TURAL
İSTANBUL, 2020

ŞEKİLLER

Sayfa

ÇİZELGELER

Sayfa

SİMGELER VE KISALTMALAR

co	Coasting (Boşta gitme)
GA	Genetik Algoritma
km	Kilometre
kWh	Kilowatt saat
max.	Maxsimum
min.	Minimum
NLP	Doğal Dil İşleme
NP	Karmaşık Problem
sn	Saniye
t	Birim zaman
Ta	Total Acceleration (Toplam ivmelenme)
tb	Total Braking (Toplam frenleme)
UITP	The International Association of Public Transport

1 GİRİŞ

Günümüzde trafik sorunları ve fosil atıklardan kaynaklanan çevre sorunlarının artması ile birlikte toplu taşımaya verilen önem artmıştır. Toplu taşımanın önem kazanması ile metro, hafif raylı sistem ve tramvay gibi ulaşım araçlarına talep artmıştır. Raylı sistemler araç, inşaat ve elektrik tüketimi bakımından yüksek maliyetli sistemlerdir. Maliyetlerin düşürülebilmesi ve projenin geri dönüş süresinin kısaltılabilmesi yeni yatırım kararlarının verilmesi açısından önemlidir. Projelerin geri dönüş maliyetlerini azaltıcı yöntemlerden biri işletme maliyetlerini oluşturan en önemli kalemlerden enerji tüketiminin en aza indirgenmesidir. Bununla birlikte, hali hazırda şehirlerdeki raylı sistemlerin tükettikleri enerji miktarları oldukça yüksektir. Daha verimli enerji tüketimi için, raylı sistemler üzerinde enerji tasarrufu ile ilgili en iyileştirme (optimizasyon) çalışmaları büyük önem taşımaktadır. Sonuçta, küresel ısınmaya karşı çevreye fayda sağlamış olunurken, ayrıca işletmenin de tasarruf yapması sağlanmış olur.

Tasarruf yöntemlerinden bir diğeri ise trenlerin frenleme esnasında açığa çıkardıkaları kinetik enerjinin elektirik enerjisine çevirilerek ile hatta bulunan bir başka trenin bu elektirik enerjiyi kullanması prensibine dayanan rejeneratif frenleme enerjisinin kullanılmasıdır. Yapılan çalışmalar incelendiğinde raylı sistemlerde enerji tasarrufunda en etkili yöntemlerden birinide rejeneratif enerjinin geri kazanımı ile gerçekleştirildiği görülmüştür.

Bu çalışmada rejeneratif frenleme enerjisinin kullanımını maksimum yapmak için tren zaman çizelgelerinde değişiklikler yapılarak en uygun zaman çizelgesini elde etmek hedeflenmiştir. En uygun tren zaman çizelgelerini bulmak için ise genetik algortima kullanılmıştır. Yapılan çalışma beş bölümden oluşmaktadır.

Birinci bölüm giriş bölümüdür. Yapılan çalışmaya yönelik giriş yapılmıştır.

İkinci bölüm literatür özeti kısmıdır. Bu bölümde literatürdeki raylı sistemlerde enerji tasarrufunun önemi, bunun için yapılan faaliyetler, rejeneratif enerjinin kullanılmasının önemi, rejeneratif enerji kullanılmasıyla elde edilecek enerji tasarrufu ve tren çizelgelerinde yapılacak değişimler ile elde edilebilecek enerji kazancı hakkında yapılan çalışmalara değinilmiştir.

Üçüncü bölümde ilgili kavramlara değinilmiştir.

Dördüncü bölümde geliştirilen model hakkında bilgi verilmiş, geliştirilen algoritma açıklanmıştır.

Son bölüm olan sonuç ve öneriler kısmında ise elde edilen veriler yorumlanarak uygun rejeneratif enerji kazanımı için çizelgelemede dikkat edilmesi gereken noktalardan bahsedilmiştir.

2 LİTERATÜR ÖZETİ

Metro İstanbul 'da 2017 yılında 160 km hat bulunurken, 2019 yılında mevcut hat uzunluğu 233 km'ye ulaşmış, yakın gelecekte ise bu rakamın 1100 km'ye ulaşması hedeflenmektedir. 2019 yılı sonunda elde edilen verilere göre İstanbul'daki tüm metro hatları tarafından tüketilen enerji 300 kWh' tır. Raylı sistemlerde enerjinin yaklaşık olarak yarısını araçlar kullanırken, kalan kısım istasyonlardaki asansörlerde, merdivenlerde, havalandırma ve aydınlatmada kullanılmaktadır.

Metro İstanbul 'da 2019 yılında mevcut hat uzunluğu 233 km iken, inşaat halinde 221 km hat ve hedeflenen toplam hat uzunluğu 1100 km'dir. Raylı sistemlerde hat uzunluğu arttıkça tüketilen enerji miktarı da doğru orantıda artmaktadır. Tüketilen enerjinin büyük kısmını araçlar kullanırken, diğer kısmı ise istasyonlardaki merdiven, havalandırma ve aydınlatmada kullanılmaktadır.

Demir yolu sistemlerinde enerji verimliliğini artırmak için akıllı istasyon tasarımları, araçların ağırlıklarının azaltılması, enerji verimli sürüş stratejileri ve enerji optimizasyonu gibi geleneksel çalışmalar yürütülmektedir (González-Gil vd., 2014). Şehir içinde kullanılan metro, hafif metro ve tramvay gibi raylı sistem araçlarının iki istasyon arasındaki hızı 80-100 km/h'dir. Bu araçların hareket süreçleri üç aşamadan oluşmaktadır. Yüksek enerji gücü isteyen ivmelenme/hızlanma aşaması, daha düşük enerji gücü isteyen boşta gitme aşaması ve aracın rejeneratif enerji ürettiği frenleme aşamasıdır (Chen vd., 2005).

Bu üç süreç göz önünde bulundurulduğunda yapılabilecek önemli tasarruflardan biri, araçların boşta sürüş tekniği ile maksimum kullanılarak enerji verimliliğinin sağlanmasıdır. Bu yöntemde, araç iki istasyon arasında maksimum hıza ulaştıktan sonra, motorlar kapatılarak aracın sahip olduğu momentum ile birlikte hareketine devam etmesi sağlanmış olur (Wong ve Ho, 2004).

Wong ve Ho'nun (2004) uyguladıkları bu yöntemle makinist kabinlerine uyarıcı sistem kurmuş, Hong Kong KCRC ve Singapur MRT 'de çalışmalar yapmışlardır. Çalışmalar sonucunda Hong Kong KCRC ile %3 e varan bir enerji kazancı sağlanmıştır.

Bir diğer çalışmada, S. Açıkbaş (2008) raylı sistem araçlarında tahrik sistemi için kullanılan enerjinin hattın geometrisine, işletmenin mevcut koşullarına, araç ve cer gücü özelliklerine göre değiştiğini öne sürmüştür (Açıkbaş, 2008). S. Açıkbaş ve M. T. Söylemez Yaptıkları çalışma sonucunda, hatlarda kullanılan besleme geriliminin 750 V DC' den 1500 V DC ye çıkarılması ile birlikte %10 oranında bir enerji tasarrufu sağlandığını gözlemlemişlerdir (Açıkbaş ve Söylemez, 2004).

Enerji tasarrufuna ilişkin bir diğer yöntem ise rejeneratif frenlemenin kullanılmasına yöneliktir. Rejeneratif frenleme ile birlikte kinetik enerji elektrik enerjisine dönüştürülür ve bu yöntem demiryolu sistemlerinde kullanılır. Rejeneratif frenleme, frenlemeye geçen trenin ürettiği enerjinin kendi iç direnci ile yakılmadan hatta bulunan alıcı konumundaki trene aktarılması ve bu trenin bu enerjiyi kullanması ile gerçekleşir. Rejeneratif enerjinin kullanımı hattaki trenlerin sefer aralıkları (headway), sayısı ve konumu ile değişim göstermektedir. Trenin frenlemesi esnasında açığa çıkan enerji alıcı durumundaki başka bir tren tarafından kullanılmadığı veya depolanmadığı sürece bu enerji boşa gidecektir. Yapılan çalışmalar sonucunda raylı sistemlerde rejeneratif frenleme ile enerjinin %40 'a varan bir kısmının geri kazılabileceği ortaya çıkmıştır (Gunselmann ve Walter, 2005).

Rejeneratif frenleme sadece enerjiyi geri kazanmaz, aynı zamanda yakılarak ısı enerjisine dönüşmesi engellenen enerji ile, tünellerde ve istasyonlarda bulunan havalandırma sistemlerinin çalışma sıklıklarının düşürülmesi sağlanmış olur (Adinolfi vd., 1998).

Rejeneratif enerjinin kullanımı ile elde edilecek enerji kazancını artırmak üzere, trenlerin istasyonlarda frenleme veya istasyondan hızlanma eşleşmelerinin en

fazla süreye denk getirilmesi amaçlanmalıdır. Bunu yapabilmek için ise, trenlerin zaman çizelgelerinin üzerinde çalışılması gerekir. Bu oldukça karışık ve elle yapılması mümkün olmayan bir NP problemdir.

Rejeneratif frenleme çalışmalarına ilk olarak, 1985 yılında Asnis'in rejeneratif frenleme enerjisi ile ilgi araştırma yapması ile başlanmıştır (Asnis vd., 1985).

Gordon ve Lehrer (1998) ise, trenlerin koordine olarak çalışmasını inceleyerek, rejeneratif frenleme üzerinde nasıl bir etkisi olduğunu araştırmıştır (Gordon ve Lehrer, 1998).

Tren zaman çizelgesi oluşturulurken, hem işletmenin hem de yolcuların fayda sağlaması dikkate alınır. Amit ve Goldfard (1971) yaptıkları çalışmada tren zaman çizelgeleme problemi için optimizasyon tekniğini uygulamışlardır (Amit ve Goldfard, 1971). Bu çalışmayla beraber araştırmacılar, yolculuk süresi, gecikme süresi, işletme maliyeti, güvenilirlik, sağlamlık gibi farklı optimizasyon hedeflerine sahip algoritmalar önermeye başlamışlardır. Örneğin, Higgins ve arkadaşları (1996) gecikme süresini ve yakıt süresini en aza indiren bir optimizasyon modeli önermişlerdir (Higgins vd., 1996).

Ghoseiri ve arkadaşları (2004) ise, yolculuk süresi ve yakıt süresini en aza indirecek bir optimizasyon modeli önermişlerdir (Ghoseiri vd., 2004).

Ayrıca, Yang ve arkadaşları (2008) trenlerin gecikme sürelerini ve toplam yolculuk sürelerini en aza indirmek üzere, her bir istasyonda trene binen veya trenden inen yolcuları değişken olarak kabul eden bir çalışma gerçekleştirmişlerdir (Yang vd., 2008).

Albrecht (2004) yaptığı çalışmada, enerji tüketimini azaltmak için frenleyen ve ivmelenen trenlerin zaman tablolarını incelemiştir. Albrecht, trenlerin istasyonlardaki bekleme sürelerini artırarak, trenlerin zaman tablosunu yeniden düzenlemiştir. Bu yöntemde amaçlanan, trenlerin çalışma zamanlarını

en iyi şekilde planlayarak (koordine ederek) en uygun kombinasyonun bulunmasıdır (Albrecht, 2004).

Ramos ve arkadaşları (2008) zaman çizelgesinde optimizasyon işlemi gerçekleştirirken, trenlerin hızlanma ve frenleme zamanlarının örtüşmesinin maksimum olmasını istemişlerdir. Maksimum çakışmayı gerçekleştirebilmek için genetik algoritma kullanmışlardır. Bu çalışma, aynı trafoda bulunan ve karşılıklı iki istasyondaki trenlerin hareketi düşünülerek gerçekleştirilmiştir (Ramos vd., 2008).

Nasri ve arkadaşları (2010) genetik algoritma kullanarak, bir istasyonda bulunan iki treni koordine etmek için en iyi (optimum) yedek (reserve) zaman aralığını buldular. Trenlerin toplam elektrik enerjisi tüketimini hesaplamak için, aynı trafo merkezine sahip bir simülasyon da gerçekleştirdiler (Nasri, vd., 2010).

Bir başka çalışmada Peña-Alcaraz ve arkadaşları (2012), aynı trafodan beslenen trenlerin hızlanma ve ivmelenme zamanlarının örtüşmesini inceleyen bir model geliştirmişlerdir. Bu modelde, değişken bekleme süreleri yerine, çalışma süreleri göz önünde bulundurularak zaman tablosu oluşturulmuştur. Bu model daha sonra Madrid Metro hattında simüle edilmiş ve enerji tasarrufunda %7 oranında iyileşme gözlemlenmiştir (Peña-Alcaraz vd., 2012).

Yang ve arkadaşları (2013) diğer çalışmalardan farklı olarak, aynı istasyona ait zıt yönlü trenler üzerinde değil de birbirini takip eden aynı yönlü trenler üzerinde çalışmalarını gerçekleştirmiştir. Aynı yönde birbirini takip eden trenlerin hızlanma ve frenleme zamanlarının örtüşmesini incelemişlerdir. Bu çalışmada genetik algoritma kullanarak en iyi çözüme ulaşmayı hedeflemişlerdir. Çalışmalarını Pekin Yizhuang metro hattında simüle etmişlerdir. Simülasyon sonucunda, trenlerin ivmelenme ve frenlerin örtüşmeleri incelenmiş ve rejeneratif frenleme enerjisinden kazanılan enerjinin en yoğun saatlerde %22, normal saatlerde %15,2 arttığı gözlemlenmiştir (Yang, vd., 2013).

Yang ve arkadaşları (2014) geliştirdikleri bu modelde değişikliğe giderek, bu sefer aynı istasyonda zıt yöne giden trenlerin ivmelene ve frenleme zamanlarındaki örtüşmeyi incelemiştir. Ardışık aynı yöne giden trenleri dikkate almamışlardır. Bu çalışmalarında da bir önceki çalışmalarında olduğu gibi genetik algoritma kullanmışlardır. Çalışmanın sonucunda rejeneratif enerjiden elde edilen enerji kazancının %8,8 oranında olduğu tespit etmişler ve yolcuların istasyonlarda bekleme sürelerinde %3,2 oranında bir azalma yapılabileceğini göstermişlerdir (Yang vd., 2014).

Zhao ve arkadaşları (2014) tarafından, metro sistemlerinde zaman çizelgelerini optimize etmek için bir optimizasyon modeli önermiştir. Bu modelde amaç, çakışma süresi ile ölçülen rejeneratif enerji kazanımını en üst düzeye çıkarmak ve diğer yaklaşım ise toplam yolculuk süresini en aza indirerek yolcu memnuniyetini artırmaktır. İki hedef ağırlıklandırma yolu ile bir araya getirilmiş ve optimum zaman çizelgesini gerçekleştirmek için Benzetilmiş Tavlama (SA) yöntemi kullanılmıştır (Zhao, 2014).

Gong ve arkadaşları (2014) yakın zamanda gerçekleştirilen diğer bir önemli çalışmada, bekleme süresi kontrolü ile rejeneratif enerji kazanımını en üst düzeye çıkarmak için bir zaman çizelgesi optimizasyon modeli sunmuşlardır. Ayrıca, istenmeyen bir durum oluştuğunda tren işletiminin planlanan zaman çizelgesine dönmesini sağlamak için telafi edici bir sürüş stratejisi algoritması önermişlerdir. Optimum bir çözüm bulmak için Genetik Algoritma kullanmışlardır (Gong, 2014).

Xu ve arkadaşları (2016) mevcut çalışmalardan farklı olarak, çalışma süresini ve her istasyonda kalma süresini kontrol ederek yolcu süresini ve kullanılan enerjiyi en aza indirmek için, bir model geliştirmişlerdir. İki hedefi bir araya getirmek için doğrusal ağırlıklı uzlaşma yaklaşımı ve bulanık doğrusal programlama yaklaşımı geliştirmişler ve optimizasyon problemini çözmek için Genetik Algoritmadan faydalanmışlardır(Xu, 2016).

İbrahim Demirci (2018) yaptığı çalışmada, rejeneratif enerji tasarrufu sağlamak için istasyon bekleme sürelerini değişken olarak kullanmıştır. Trenlerin hızlanma ve frenleme esnasındaki örtüşme oranının en fazla olmasını sağlamak için genetik algoritma ile zaman çizelgesini en iyileştirmeye çalışmıştır. Çalışma, Metro İstanbul Hatlarından Kadıköy-Kartal hattında uygulanmış ve sonucunda yolcu yoğunluğunun en fazla olduğu saatlerde enerji kazanımının %46'dan %60 'a çıktığı, normal saatlerde ise enerji kazanımının %14'den %30'lara çıktığı gözlemlenmiştir (Demirci ve Celikoglu, 2018).

Literatürden de görüldüğü üzere rejeneratif enerjiden faydalanmak amacı ile birçok çalışma gerçekleştirilmiştir. Bu makalede ele alınan problem, İbrahim Demirci (2008) tarafından daha önce gerçekleştirilen çalışmanın devamı niteliğindedir. Bu makalede rejeneratif enerjinin kazanımını maksimum yapacak en iyi çözümünün bulunması için genetik algoritma ile bir çalışma yapılmıştır. Çalışmada istasyon bekleme sürelerinin minimum ve maksimum değerleri göz önüne alınarak zaman çizelgesi en iyileştirilmiştir. Bekleme sürelerinde değişim, toplam bekleme süreleri için ayrılan süreyi aşmayacak şekilde düzenlenmiştir. Farklı genetik algoritma parametre yöntemleri uygulanmıştır. Yapılan simülasyon sonucunda, referans çalışmaya göre (%60 rejeneratif kazanım) %26 civarında daha iyi sonuçlar elde edilmiştir (%76 rejeneratif kazanım).

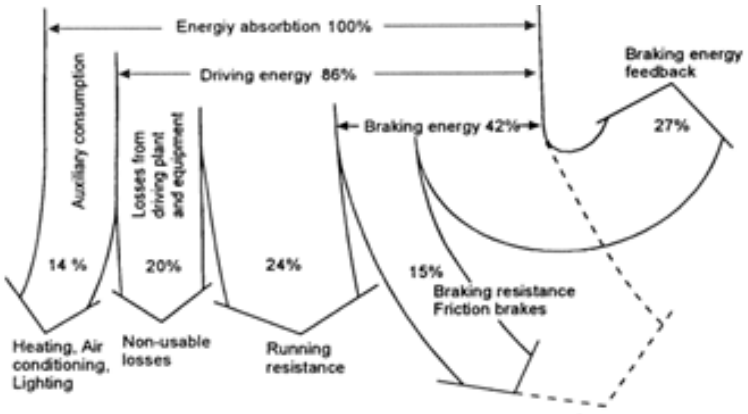
3 İLGİLİ KAVRAMLAR

3.1 Raylı Sistem Araçlarında Enerji Tüketimi

Raylı sistem araçlarında enerji tüketimi işletmeden işletmeye göre değişim gösterebilir. İşletmeler de enerji tüketimine sebep olan birden fazla faktör bulunmaktadır. Bunlar;

- Hatların besleme gerilimleri
- Hattın topografyası ve çevresel yapısı
- Trafolar, kompresörler vb. cihazlar
- Araçlara uygulanan sürüş stratejileri
- Araçların frenleme dirençleri
- Aydınlatma, havalandırma ve ısıtma için harcanan
- Araçların taşıdıkları yolcu sayısı

Araçlar frenleme esnasında hatta bulunan katenerden aldıkları enerjinin %42'sini elektrik enerjisine çevirmektedir. Araçların frenlemesi ile oluşan enerjinin yaklaşık olarak %27'si katener sistemine geri verilmektedir. Frenleme esnasında oluşan enerjinin %15'lik bir kısmı ise rezistörlerde harcanmaktadır (Amalgamated Report, 1995)(Şekil 3.1).



Şekil 3. 1 Raylı sistem araçlarında enerji tüketimi

3.2 Rejeneratif Frenleme ve Enerjinin Geri Kazanılması

Raylı sistem araçlarının enerji tasarrufu için kullandıkları yöntemlerden biri de rejeneratif frenleme ile elde edilen enerjinin kullanılmasıdır. Burada asıl önemli nokta frenlemeye geçen aracın üretmiş olduğu enerjinin kendi iç direnci ile yakılmadan önce kullanılmasıdır (Daniel Cornic, 2010). Araçların motorları frenleme anında üreteç olarak çalışmaktadır. Araçların frenleme esnasında oluşturdukları kinetik enerji elektrik enerjisine çevrilir. Kinetik enerjiden elektrik enerjisine dönüştürülen enerji trafoya iletilir. Bu enerji farklı yöntemlerle tekrar kullanılabilir. Elde edilen enerji istasyonlarda aydınlatma, havalandırma ve ısıtma sistemleri için kullanılabilir. Bir başka kullanım şekli de elde edilen bu enerjinin araç-üstü sistemlerine iletilmesi ve araç tarafından yardımcı konfor fonksiyonları için kullanılmasıdır. Rejeneratif frenleme ile kazanılan bu enerjinin tamamı kullanılamaz çünkü kazanılan enerji aracın yardımcı konfor fonksiyonlarının talep ettiği enerji miktarından daha fazladır. Diğer bir yöntem ise bu enerjinin trafodan o anda hatta bulunan ve ivmelenmekte olan araca iletilmesidir. Bu makalede rejeneratif frenleme ile kazanılan enerjinin hatta bulunan bir başka araca iletilmesine yönelik bir çalışma yapılmıştır.

Rejeneratif enerji ile kazanılacak toplam enerji, frenleme yapan araçların sayısı, araçların kurulu gücü, araçların frene başladığı esnadaki hızları, frenleme sıklığı gibi parametrelere bağlıdır (Açıkbaş ve Alataş, 2006).

3.3 Zaman Çizelgesi

Tren zaman çizelgelerinde trenlerin sefer aralıkları, yolcuların ihtiyaçlarına uygun hizmet verecek şekilde oluşturulmalıdır. Tren seferleri planlanırken amaçlanan, hangi trenin ne zaman hangi istasyonda olacağının, istasyonlarda ne kadar süre beklemesi gerektiğinin ve istasyonlar arasında oluşabilecek gecikmelerin tolere edilmesi için yedek zaman aralıklarının belirlenmesidir.

Zaman çizelgeleri ve rejeneratif enerjinin geri kazanımı birbiriyle ilişkilidir. Trenlerinin aralarındaki süre 3 dakika ve altında ise frenleme esnasında oluşan enerjinin %85 ile, %95'i diğer alıcı trenler tarafından kullanılmaktadır(UITP, 2005). Yapılan çalışmalarda trenler arasındaki sürenin daha fazla olduğu zaman aralıklarında rejeneratif enerjinin büyük kısmının fren rezistörlerinde yakıldığı görülmüştür (Martin, 1999).

Rejeneratif frenleme ile oluşan enerjinin kullanılmasını artırmak için trenlerin ivmelenme ve frenleme zamanlarının koordinasyonunun sağlanması gereklidir. Bunu başarmanın en etkin yollarından biri, istasyonda bekleme sürelerinin, trenlerin ivmelenme ve frenleme zamanlarının denk getirilecek biçimde düzenlenmesidir.

Paul 1999 yılında yaptığı çalışma ile tren sefer sürelerini %5 artırarak yaklaşık %20'lik bir enerji kazancı elde etmiştir (Jong ve Chang, 2005). Yine benzer bir çalışmada tren sürelerinin %10 artırılması sonucu, %10'luk bir enerji kazancı elde edilmiştir (Martin, 1999).

Tren zaman çizelgelerinde en yaygın kullanılan metot bekleme sürelerinde yapılacak değişiklikler üzerine kurulmuştur. Trenlerin seferleri için zaman tablosu oluşturulurken arızalardan veya yolculardan kaynaklı gecikmeler göz önünde bulundurulur ve maksimum bekleme süreleri de zaman çizelgelerine eklenir. Ekstra bekleme süreleri ile istasyon bekleme süreleri kullanılarak, enerji verimi elde edilecek sürüşler için istenilen bekleme zamanları kullanılabilir (Açıkbaş, 2008).

3.4 Optimizasyon

Optimizasyon, verilen amaç veya amaçlar için belirli kısıtlamaların sağlanarak en uygun çözümün elde edilme sürecidir. Bilim adamları yeni bir fikir ortaya koyar ve optimizasyon aracılığıyla bu fikir geliştirilir. Bir fikri etkileyen parametre veya bilgi elektronik formata dönüştürülebildiği sürece, bilgisayar

mükemmel bir optimizasyon aracıdır. Optimizasyon terminolojisinde her zaman en iyiye ulaşma arzusu söz konusudur.

En iyi tanımlaması, probleme, çözüm metoduna ve izin verilen toleransa bağlıdır. Geçmişten günümüze, karşılaşılan problemlerin çözülmesi amacıyla bir çok optimizasyon teknikleri geliştirilerek, değişik alanlara uygulanmıştır (Çunkaş, 2004).

Optimizasyon problemlerinin çözümünde klasik yöntemler olarak adlandırılan matematiksel yöntemler önceleri çok yaygın olarak kullanılmaktaydı. Bu tür yöntemlerin esnek olmaması ve matematiksel fonksiyonlarla tanımlama gereksinimi gibi dez avantajları, son zamanlarda, bilim adamlarında genel amaçlı ve performansı yüksek yöntemler geliştirme çabalarını artırmış ve doğadaki olaylardan esinlenmeye başlamışlardır. Tabiattaki olaylar temel alınarak geliştirilen optimizasyon algoritmaları sezgisel yöntemler olarak adlandırılmaktadır (Karaboğa, 2004). Bunlardan Genetik Algoritmalar(GA), Diferansiyel Evrim Algoritması ve Parçacık Sürü Optimizasyonu Algoritmaları, optimizasyon problemlerinde yaygın olarak kullanılmaktadır (Rahimpour, 2006).

3.4.1 Optimizasyonun aşamaları

Bir sistemin tasarlanma sürecinde, sorun gözlemlenerek önce ihtiyaçlar belirlenir. Daha sonra bu ihtiyaçlar doğrultusunda sistem tasarımı yapılarak denemelerle iyileştirilir. İhtiyaçlar belirlendikten sonra, sistemin çalışması dahil olmak üzere tüm aşamalarda optimizasyon devreye girer. Aşağıda optimizasyon aşamaları verilmiştir.

- Tasarım değişkenlerinin tanımlanması
- Hedef fonksiyonun belirlenmesi
- Kısıtların belirlenmesi
- Uygun optimizasyon metodunun seçilmesi ve uygulanması

3.4.2 Optimizasyonun kullanım alanları

Optimizasyon, karar verme süreçlerini hızlandırmakta ve karar kalitesini arttırmakta kullanılarak gerçek hayatta karşılaşılan problemlerin etkin, doğru ve gerçek zamanlı çözümünde yararlanılmaktadır. Aşağıda optimizasyon yönteminin kullanıldığı alanlar verilmiştir;

- Yapı-Araç İskeleti Dinamiği'ne ilişkin problemler
- Dizayn problemleri
- Ekonomi problemleri
- Zaman ve maliyet problemleri
- Operasyon araştırmaları
- Benzetim (Simülasyon) ile Senaryo Analizleri

3.5 Genetik Algoritma

Genetik algoritmalar çözüm uzayı büyük olan problemler için biyolojik evrim sürecinin bilgisayar ortamında kodlanması yoluyla, en iyi veya en iyiye yakın çözümün bulunmasına dayanan bir arama yöntemidir. Bu yaklaşım ilk olarak Holland tarafından 1975 yılında ortaya atılmıştır (Holland, 1992).

Genetik algoritmalar karmaşık problemlerin hızlı ve en iyiye yakın olarak çözülmesi için uygulanırlar. Büyük çözüm uzayına sahip problemlerde genetik algoritma kullanılarak kısa sürede çözüme ulaşılabilir. Genetik algoritmalar kısa sürede en iyi çözüme ulaşması ve global çözüm bulma açısından pek başarılı iken, yerel çözüm bulmada başarılı değildir (Gonzales ve Fernandez, 2000).

Genetik algoritmada ilk olarak problemin özelliklerine uygun kromozom belirlenir. Popülasyon içindeki bireyler, genellikle rastgele belirlenen kromozom değerleri ile oluşturulurlar. Popülasyonu oluşturan bireylerin kromozom değerlerinin her biri problemin çözümü için birer adaydır. Hangisinin daha iyi sonuç olduğu probleme özgü belirlenmiş bir iyilik fonksiyonu (fitness function) değeri ile belirlenir. Daha sonra popülasyonlarda

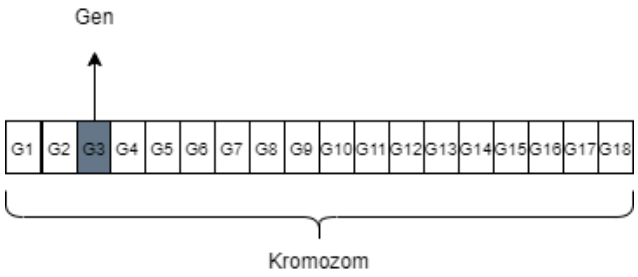
ki kromozomlar tıpkı evrim sürecinde olduğu gibi “çaprazlama”(crossover) ve “mutasyon” (muation) operatörlerine tabii tutularak yeni kromozomlar elde edilir. Bu işlemde amaç daha iyi kromozomlar oluşturmaktır. Bu işlemler, istenen yineleme sayısı boyunca veya tanımlanan hata eşiğine ulaşılana kadar tekrarlanır ve her döngü sonucunda yeni nesiller oluşturulur. Bir döngü sonucunda oluşturulan yeni bireylerden en iyileri “elit bireyler“ (elite chromosome) bir sonraki nesile doğrudan aktarılır ve popülasyon sayısı sabit tutulur (Gonzales ve Fernandez, 2000).

3.5.1 Genetik algoritma terimleri

Gen**:** Kendi başına anlamlı bilgi taşıyan en küçük birimdir.
Kromozom**:** (Birey) Birden fazla genin bir araya gelerek oluşturduğu dizidir.

- Kromozomlar, alternatif aday çözümleri gösterirler.
- Birey sayısı (kromozom) genelde sabittir.

Aşağıdaki Şekil 3.2’de kromozom ve gen gösterimi verilmiştir.



Şekil 3. 2 Gen ve kromozomun gösterimi

Popülasyon**:** Kromozomlardan oluşan topluluktur.

- Geçerli alternatif çözüm kümesidir.
- Popülasyondaki kromozom sayısı arttıkça çözüme ulaşma süresi (iterasyon sayısı) azalır.

Kodlama**:** Genetik algoritmalarda problemler uygulanmadan önce, verinin uygun şekilde kodlanması gerekmektedir.

Kurulan genetik modelin hızlı ve güvenilir çalışması için bu kodlamanın doğru yapılması gerekmektedir.

3.5.2 Genetik algoritmada kodlama türleri

Genetik algoritmada kodlama kromozomlarla temsil edilen çözümlerin nasıl oluşturulması gerektiğini bulmamızda yardımcı olur. Birden fazla kodlama türü mevcuttur.

3.5.2.1 İkili (Binary) kodlama

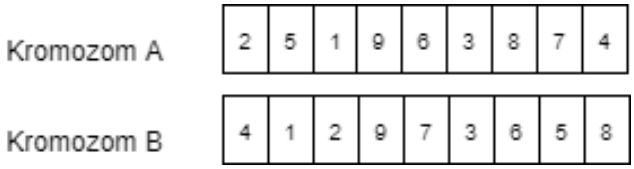
İkili kodalama kodlama yöntemlerinden en sık tercih edilen yöntemdir. İkili kodlamada bitler (genler) kullanılır; bitler 1 ya da 0 olmak üzere iki değer alabilirler. Kromozomların uzunluğu da problemin yapısına göre değişir. Örneğin 32 farklı karar değişkeni olan bir problem için kromozomun 2n=32; n=5 bit değeri vardır. 00010 11001 10101 şeklinde kromozomlar birleşerek bireyi oluşturur (Kahraman ., 2004).
Aşağıdaki Şekil 3.3'de ikili kodlamaya örnek verilmiştir.

1	0	1	1	1	0	1	0	1	0	1	0	0	1	1	0	1	0

Şekil 3. 3 İkili kodlama

3.5.2.2 Permütasyon kodlama

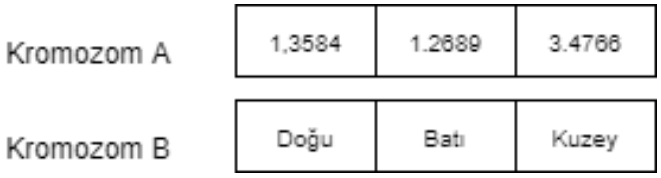
Bu kodlama yönteminde her kromozom bir sıra ile temsil edilmektedir. Bu yöntem daha çok sıralama problemlemlerin çözümünde kullanılır. Permütasyon kodlama, gezgin satıcı ve çizelgeleme promlemlerinde kullanılır. Kombinatorik problemler için çoğunlukla permütasyon kodlama kullanılmaktadır, fakat sözkonusu kodlamada genetik operatörlerden biri olan çaprazlama sırasında sorun çıkabilmekte ve bunun giderilebilmesi için onarıcı işlemlere gerek duyulmaktadır (Dursun, 2009). Aşağıdaki Şekil 3.4'de permütasyon kodlamaya örnek verilmiştir.



Şekil 3. 4 Permütasyon kodlama

3.5.2.3 Değer kodlama

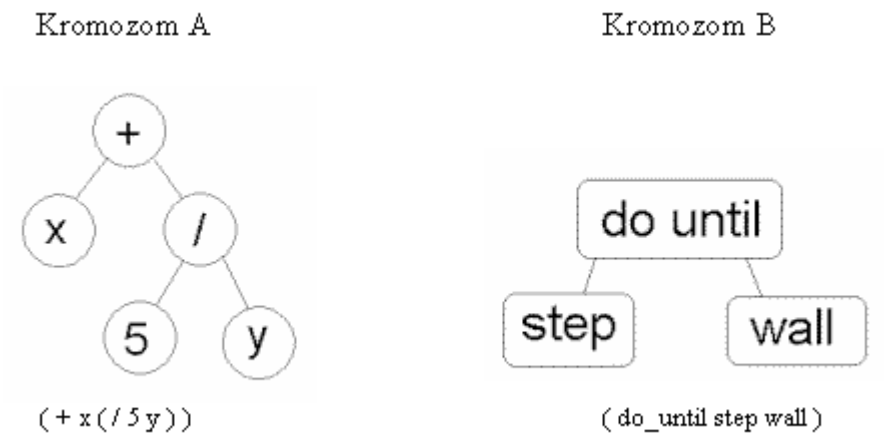
Bu kodlama yönteminde gerçel sayılar gibi karmaşık değerlerin kullanıldığı problemlerde, ikili kodlama daha zor olduğundan doğrudan değer kodlaması kullanılabilir. Aşağıdaki Şekil 3.5'de değer kodlamaya örnek verilmiştir.



Şekil 3. 5 Değer kodlama

3.5.2.4 Ağaç kodlama

Ağaç kodlama yöntemi daha çok gelişen, değişen programlar veya ifadeler için kullanılır. Örneğin GA ağaç kodlamada her kromozom, bazı nesnelerin (örneğin fonksiyonlar ya da programlama dilindeki komutlar gibi) ağaçlardır. Aşağıdaki Şekil 3.6'da ağaç kodlamalı kromozomlar örneği verilmiştir.



Şekil 3. 6 Ağaç kodlamalı kromozomlar örneği

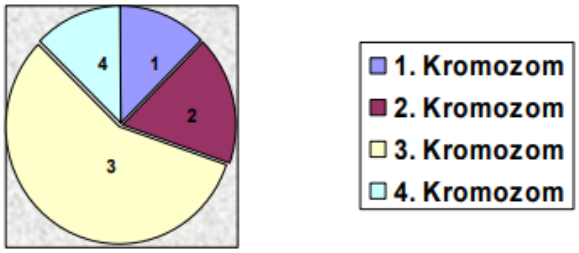
3.5.3 Genetik algoritmada şeçilim işlemi

Yeni topluluğu oluşturmak için mevcut topluluktan çaprazlama ve mutasyon işlemine tabi tutulacak bireylerin seçilmesi gerekir. Teoriye göre iyi olan bireyler yaşamını sürdürmeli ve bu bireylerden yeni bireyler oluşturulmalıdır. Bu nedenle tüm seçilim yöntemlerinde uygunluk değeri fazla olan bireylerin seçilme olasılığı daha yüksektir. En bilinen seçilim yöntemleri Rulet Seçilimi, Turnuva Seçilimi ve Sıralı Seçilimdir.

3.5.3.1 Rulet seçilim

Topluluktaki tüm bireylerin uygunluk değerleri toplanır ve her bireyin seçilme olasılığı, uygunluk değerinin bu toplam değere oranı kadardır.

Bireyler bir rulet tekerleğine yerleştiriliyormuş gibi düşünülür. Bireylerin uygunluk fonksiyonlarına göre rulet tekerleğinde kaplayacakları hacim belirlenir, uygunluk fonksiyonu yüksek olanların rulet tekerleğinde daha fazla hacimler kaplaması beklenir. Böylelikle rulet tekerleği çevrildiğinde uygunluk fonksiyonu yüksek olan bireylerin seçilme şansı yükselmiş olur (Kahraman ve Özdağlar, 2004). Aşağıdaki Şekil 3.7'da rulet tekeri seçimine örnek verilmiştir.



Şekil 3. 7 Rulet Tekeri Seçilimi

3.5.3.2 Sıralı seçilim

Sıralı seçimde en kötü uyumlulukta olan kromozoma 1 değeri, sonrakine 2 değeri, …, sonuncuya N (birey sayısı) verilir. Böylelikle seçilmede bunlara öncelik tanınmış olur. Bu şekilde onların da seçilme şansı artar. Ancak bu yöntem, çözümün daha geç yakınsamasına neden olabilir.

3.5.3.3 Turnuva seçilimi

Topluluk içerisinden rastgele k adet (3,5,7...) birey alınır. Bu bireylerin içerisinden uygunluk değeri en iyi olan birey seçilir. Uygunluk fonksiyonu yüksek olan birey tutulur geriye kalanlar atılır. Yeni topluluk bireyleri belli sayıdaki bireyler arasında yapılan yarışma sonucu oluşturulur (Kahraman ve Özdağlar, 2004).

3.5.4 Genetik algoritma operatörleri

GA'nın temel işleyişini oluşturan ve yürütücülüğünü belirleyen kısımdır. Kullanılan genetik operatörler, var olan popülasyon üzerine uygulanan işlemlerdir. Bu işlemlerin amacı daha iyi özelliğe sahip yeni nesiller üretmek ve arama algoritmasının alanını genişletmektir. Bu operatörler;

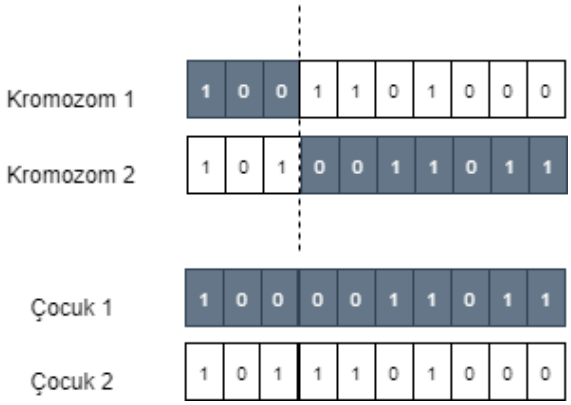
- Başlangıç Popülasyonu
- Uygunluk Değerinin Hesaplanması
- Yeniden Üretme
- Çaprazlama
- Mutasyon

3.5.4.1 Çaprazlama operatörü

Genetik algoritmadaki en önemli operatörlerden biri çaprazlamadır. Seçilen iki kromozomun çaprazlanması ile yeni bireyler oluşturulur. Çaprazlama işleminde amaç ebeveynlerin uygunluk değerinden daha yüksek iyilik fonksiyonu değerine sahip yeni bireyler oluşturmaktır. Literatürde birden fazla çaprazlama yöntemi mevcuttur. En kolay yöntem tek noktadan çaprazlamadır (Karasoy , BALLI). Bu yöntemde seçilen çaprazlama noktasına kadar genler bir ebeveynden alınırken, kalan genler ise diğer bir ebeveynden alınarak yeni birey oluşturulur. Bu yöntem çok noktalı olarak da uygulanabilir. Bu durumda tek noktalı çaprazlamaya benzemektedir ama çaprazlama noktaları rastgele belirlenir. Yeni birey ilk noktaya kalan olan genleri bir ebeveynden alırken, ikinci çaprazlama noktasına kadar olan genleri diğer ebeveynden alır ve sırayla diğer genler iki ebeveynden elde edilir. Çaprazlama işlemi olarak bu yöntemler haricinde "Pozisyona Dayalı Çaprazlama" (Murata vd., 1996), "Kısmi Planlı Çaprazlama" (Goldberg, 1989), "Sıraya Dayalı Çaprazlama" (Syswerda, 1991) gibi yöntemler de kullanılmaktadır.

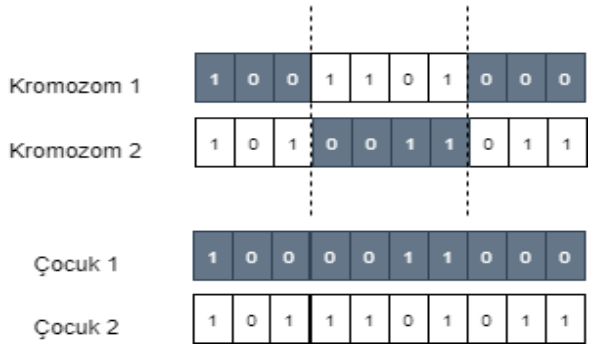
Çaprazlamada amaç, ata kromozomun yerlerini değiştirerek çocuk kromozomlar üretmek ve böylelikle zaten uygunluk değeri yüksek olan ata kromozomlardan daha yüksek uygunluklu çocuk kromozomlar üretmektir.

Tek Noktalı Çaprazlama**:** Aşağıdaki Şekil 3.8'da tek noktalı çaprazlama örnek ile açıklanmaya çalışılmıştır.



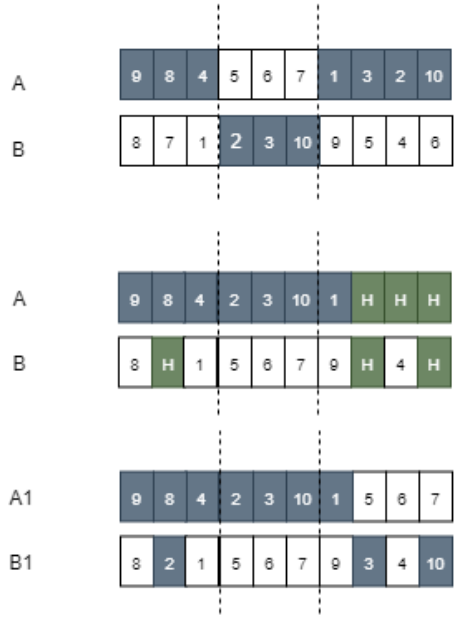
Şekil 3. 8 Tek noktalı çaprazlama

Çift Noktalı Çaprazlama: Aşağıdaki Şekil 3.9 ‘ de çift noktalı çaprazlama örnek ile açıklanmaya çalışılmıştır.



Şekil 3. 9 Çift noktalı çaprazlama

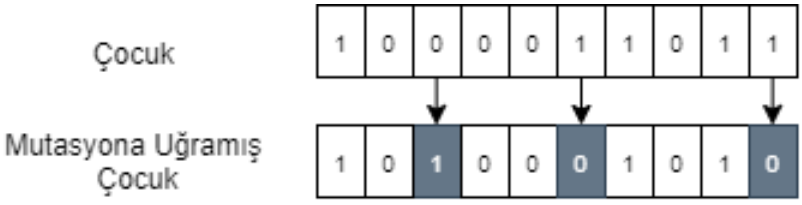
Sıralı Çaprazlama: Bu yöntemde çarprazlanacak olan genler arasında çarprazlanan kromozomların sırası taklit edilmektedirAşağıdaki şekil 3.10‘ de sıralı çaprazlama örnek ile açıklanmaya çalışılmıştır. Çaprazlama sıra ile soldan sağa doğru gerçekleştirilmiştir.



Şekil 3. 10 Sıralı çaprazlama

3.5.4.2 Mutasyon operatörü

Mutasyon operatörü genetik çeşitliliğin sağlanması açısından kullanılan bir genetik algoritma operatörüdür. Çaprazlama işlemi sonucunda oluşturulan kromozomların genlerinden bir veya birkaç tanesi değiştirilerek yeni kromozomlar elde edilir. Bu operatör oluşan tüm genlere uygulanamaz belli bir oranda uygulanır. Mutasyon oranının düşük olmasından dolayı etkileri kromozomlarda az hissedilir. Aşağıdaki şekil 3.11'de mutasyon örnek ile açıklanmıştır.



Şekil 3. 11 Mutasyon operatörünün uygulandığı kromozom

3.5.4.3 Elitzm operatörü

Çaprazlama mutasyon gibi operatörlerin uygulanması sonucunda en yüksek iyilik fonksiyonu değerine sahip bireyin bir sonraki nesile aktarılmasıdır.

3.5.4.4 Yeni neslin oluşturulması

Kromozomlar üzerinde çaprazlama ve mutasyon operatörlerinin uygulanması sonucunda yeni bireyler üretilir. Yeni kromozomların uygunluk değeri popülasyondaki diğer kromozomların iyilik fonksiyonu değeri ile karşılaştırılır. Yeni kromozomların uygunluk değeri diğer kromozomdan daha yüksek olduğu takdirde kromozomlar popülasyondan çıkarılır ve yerini yeni kromozomlar alır. Böylelikle nesil sayısı kadar birey üretilinceye kadar veya durdurma kriteri sağlanıncaya kadar devam eder (Yeo ve Agyei, 1998).

3.5.5 Genetik algoritmanın işleyişi

1.Adım: Olası çözümlerin kodlandığı bir çözüm grubu oluşturulur. Çözüm grubuna biyolojideki benzerliği nedeniyle popülasyon, çözümlerin kodları da kromozom olarak adlandırılır. Bu adıma popülasyonda bulunan birey sayısı belirleyerek başlanır. Bu sayı için bir standart yoktur. Genel olarak önerilen 100- 300 aralığında bir büyüklüktür. Büyüklük seçiminde yapılan işlemlerin karmaşıklığı ve aramanın derinliği önemlidir. Popülasyon bu işlemden sonra rasgele oluşturulur.

2. Adım: Her kromozomun ne kadar iyi olduğu bulunur. Kromozomların ne kadar iyi olduğunu bulan fonksiyona uygunluk fonksiyonu denir. Bu fonksiyon işletilerek kromozomların uygunluklarının bulunmasına ise hesaplama(evalution) adı verilir. Bu fonksiyon genetik algoritmanın beynini oluşturmaktadır. GA da probleme özel çalışan tek kısım bu fonksiyondır. Çoğu zaman GA nın başarısı bu fonksiyonun verimli ve hassas olmasına bağlı olmaktadır.

3. Adım: Bu kromozomları eşleyerek yeniden kopyalama ve değiştirme operatörleri uygulanır. Bu sayede yeni bir popülasyon oluşturulur. Kromozomların eşlenmesi kromozomların uygunluk değerlerine göre yapılır. Bu seçimi yapmak için rulet tekerleği seçimi, turnuva seçimi gibi seçme yöntemleri vardır.

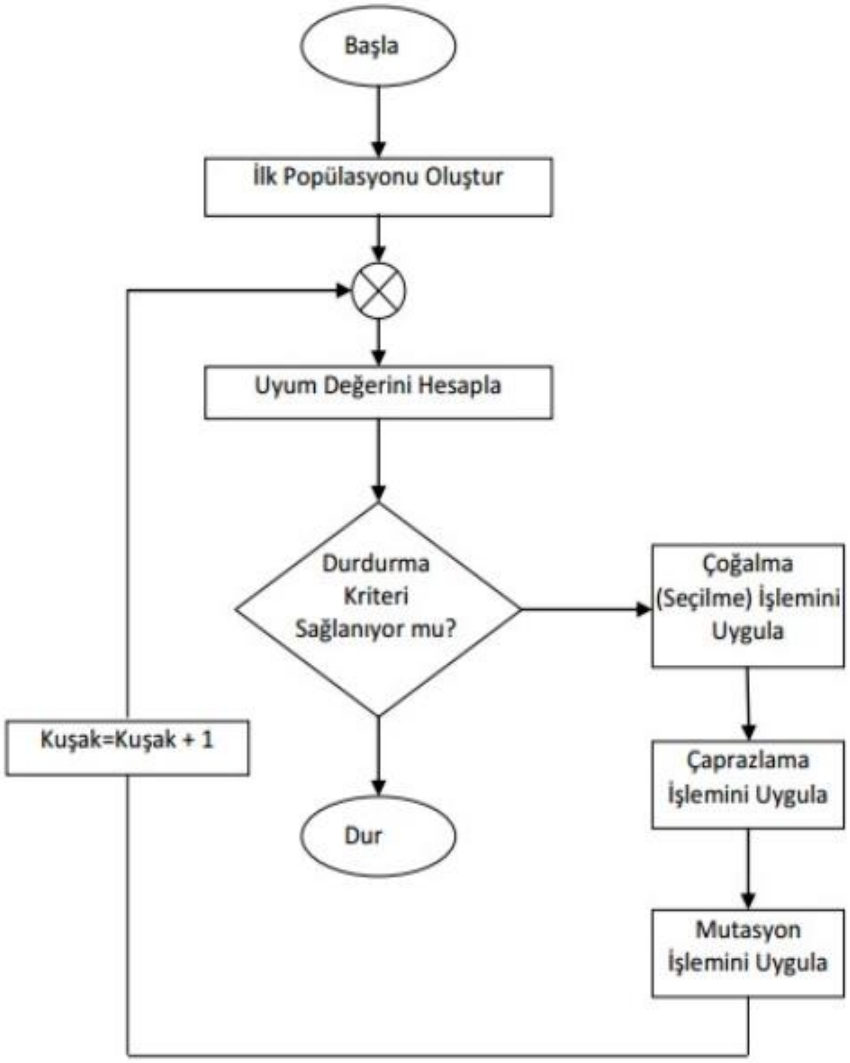
4. Adım: Yeni kromozomlara yer açmak için elit olmayan yani bir sonraki nesile aktarılmayacak kromozomlar kaldırılır. Eski kromozomlar çıkartılarak sabit büyüklükte bir popülasyon sağlanır.

5. Adım: Tüm kromozomların uygunlukları tekrar hesaplanır. Tüm kromozomlar yeniden hesaplanarak yeni popülasyon un başarısı bulunur.

6. Adım: GA iterasyon sayısı kadar çalıştırılarak çok sayıda popülasyon oluşturulup hesaplanır. Eğer iterasyon sayısı tamamlanmamış, istenilen uygunluk değerine ulaşılmamış ise 3. adıma gidilir.

7. Adım: O ana kadar bulunan en iyi kromozom sonuçtur. Çünkü popülasyon ların hesaplanmasında en iyi bireyler saklanmıştır.

Aşağıdaki şekil 3.12'de Genetik algoritmanın akış diyagramı verilmiştir.



Şekil 3. 12 Genetik algoritmann işleyişi

3.5.6 Genetik algoritmada parametre seçimi

Genetik algoritmada parametre seçilirken iki noktaya dikkat edilmelidir. Bunlar popülasyon büyüklüğü ve mutasyon olasılığıdır.
Popülasyon Büyüklüğü:

- Genetik algoritma kullanıcısı tarafından verilen en önemli kararlardan birisidir. Bu değer çok küçük olduğunda, genetik algoritma yerel en iyi değere takılabilmektedir.
- Popülasyonun çok büyük olması ise çözüme ulaşma zamanını arttırmaktadır.

Mutasyon Olasılığı:

- Mutasyon P(m) olasılığı ile bir kromozomdaki herhangi bir gende meydana gelebilir. Eğer mutasyon olasılığı artarsa, genetik arama rastsal bir aramaya dönüşür.

3.5.7 Genetik algoritmanın uygulama alanları

Genetik algoritmaların en uygun olduğu problemlere,

- Geleneksel yöntemler ile çözümü mümkün olamayan
- Çözüm süresi problemin büyüklüğü ile üstel orantılı olarak artanlardır.
- Bugüne kadar GA ile çözümüne çalışılan konulardan bazıları şunlardır.
- Yapay Sinir Ağları (artificial neural networks)
- Otomatik Programlama (automatic programming)
- Makine Öğrenmesi (machine learning)
- Ekonomi (economics)
- Popülasyon Genetiği (population genetics)
- Görüntü İşleme (image processing)

Örnek verilebilir.

4 ARAŞTIRMA BULGULARI VE TARTIŞMA

Uygulamada amaç rejeneratif enerjinin kazanımını maksimum yapabilmektir. Rejeneratif frenleme enerjisini kullanarak sağlanacak maksimum kazanç frenleyen tren sayısı kadar ivmelenen tren olduğunda gerçekleşir. Bunun için trenlerin ivmelenme ve frenleme yaptıkları zamanın örtüşmesinin maksimum olması gerekmektedir. İstasyon bekleme sürelerinde yapılacak değişimler ile trenlerin ivmelenme ve frenleme zamanlarının maksimum olduğu çözümün bulunması hedeflenmektedir. En uygun zaman çizelgesini bulabilmek elle yapılması zor bir NP problemidir.

En uygun çözümü verecek bekleme sürelerini bulabilmek için genetik algoritma kullanılmıştır.

4.1 Uygulama Bilgileri

4.1.1 Kavramlar

Trenlerin hareket süreleri dört kısıma ayrılmıştır. Bunlar;

• Bekleme: Trenlerin istasyondaki bekledikleri süredir.
• İvmelenme: Trenlerin istasyon kalkışından maksimum hıza ulaştığı ana kadar geçen süredir.
• Boşta Gitme: Trenlerin maksimum hıza ulaştıktan sonra bu hızda devam edilen veya boşta gidilen süredir.
• Frenleme: Trenlerin frenlemeye geçerek rejeneratif enerjinin üretildiği ve istasyonda durduğu ana kadar geçen süredir

4.1.2 Parametreler

4.1.2.1 Tur zamanı

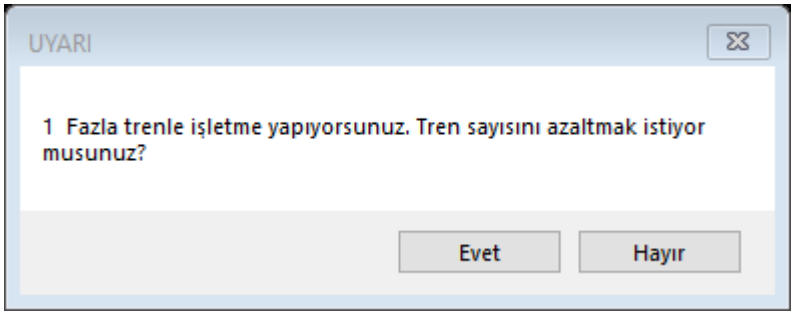
Tur zamanı trenin Kadiköy istasyonundan çıkıp Tavşantepe'ye ulaştıktan sonra tekrar Kadiköy istasyonuna gelinceye kadar geçirdiği süredir. Bu süre 75 – 76 dakikalar civarında bir zaman dilimini kapsar.

4.1.2.2 İstasyon

Kullanılan metro hattında 36 adet istasyon için kullanılmaktadır. Hat sonlarında ise trenlerin dönüşleri için kullanılan birer adet hat sonu dönüş peronları bulunmaktadır.

4.1.2.3 Tren sayısı

Hatta çalışan trenlerin sayısı 18 olarak seçilmiştir. Trenlerin sayısı seçime bağlı olarak artırılıp azaltılabilmektedir. Aynı zamanda geliştirilen uygulama daha az sayıda tren ile istenilen iki tren arası mesafe (Headway) sağlanabiliyorsa bunu kullanıcıya gösterir. Kullanıcının seçimi sonucunda tren sayısı azaltılır ve azalan tren sayısına bağlı olarak enerji tasarrufu gerçekleştirilmiş olur. Aşağıdaki Şekil 4.13'da fazla tren ile işletme yapıldığında kullanıcıya sunulan seçenek gösterilmektedir.



Şekil 4. 13 Fazla trenle işletme yapıldığının kullanıcıya belirtilmesi

4.1.2.4 Simülasyon süresi

Uygulamanın çalıştırılacağı toplam zamanı temsil eder. Simülasyon süresi için 18000 saniye (5 saat) seçilmiştir. Kullanıcı tarafından değiştirilebilmektedir.

4.1.2.5 İki tren arası mesafe (Headway)

İki tren arasındaki saniye cinsinden mesafedir. Uygulamada iki tren arasındaki mesafe 273 saniye olarak belirlenmiştir. Kullanıcı tarafından değiştirilebilmektedir.

4.1.2.6 Minimum ve maksimum bekleme süreleri

M4 hattında kullanılan veriler göz önüne alındığında istasyonlar için geçerli üç adet bekleme süresi olduğu görülmektedir. Minimum (-5) ve maksimum (+5) bekleme süreleri işletmeye daha esnek bir çalışma olanağı sunar. Tüm istasyonlar için belirlenen bu süreler sayesinde oluşabilecek özel koşullarda bir aksilik çıkmadan seferlerin zamanında işletilmesini sağlar.

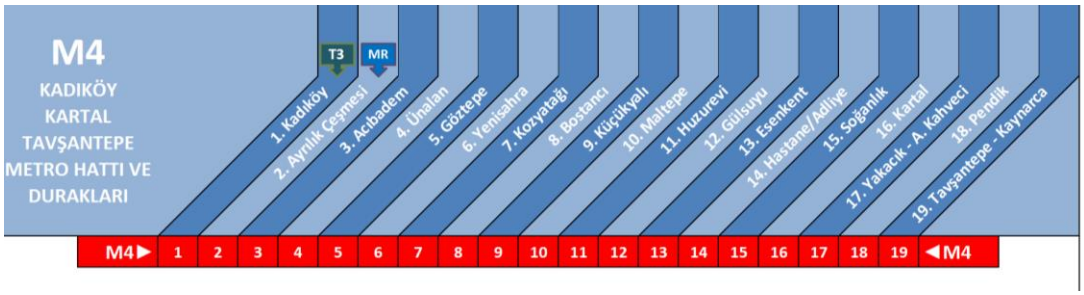
4.1.2.7 İstasyon bekleme süreleri ve optimizasyon

Geliştirilen uygulama ile istasyonların minimum ve maksimum bekleme sürelerinden yararlanarak, bekleme sürelerinde optimizasyon işlemi gerçekleştirilir. Optimizasyon işleminin amacı ise rejeneratif enerji kullanımının maksimum olmasını sağlamaktır.

İstasyonda bekleme sürelerinin azalması veya artması ile optimizasyon gerçekleştirilir. Uygulama, frenleme anındaki trenin ivmelenen bir başka trenle kesişmesini amaçlamaktadır.

4.2 Uygulamanın çalışma prensibi

Simülasyon için Metro İstanbul'un M4 Kadıköy-Tavşantepe Metro hattının gerçek verilerinden faydalanılmıştır. Aşağıdaki Şekil 4.12'de M4 Kadıköy-Tavşantepe metro hattı verilmiştir.



Şekil 4. 14 M4 Kadıköy-Tavşantepe metro hattı

Simülasyonda kullanılan tren sayısı 18 olarak seçilmiştir. İki tren arasındaki süre (Headway) 273 sn ve simülasyonun süresi ise 1800 sn olarak belirlenmiştir. (Çizelge 4.1)

Çizelge 4.1. Simülasyonda kullanılan veriler

İki Tren Arası Mesafe	273
Tren Sayısı	18
Simulasyon Süresi	18000

Kadıköy-Tavşantepe arasındaki tüm istasyonların, iki istasyon arası ivmelenme, boşta gitme, frenleme süreleri, istasyonlara ait minimum, maksimum ve normal bekleme süreleri hattan alınan verilere göre Excel tablosunda tutulmaktadır. Uygulama başlarken bu verileri okur ve işlemlerini bu tabloya göre gerçekleştirir (Çizelge 4.2.).

Çizelge 4.2. M4 Kadıköy-Tavşantepe metro hattı istasyon arası kullanılan veriler

İki İstasyon Arası	İstasyonda Bekleme Süresi	İstasyonda Minimum Bekleme Süresi	İstasyonda Maksimum Bekleme Süresi	Full İvmelenme Süresi	Boşta Gitme Süresi	Frenlemeye Geçtiği Süre	ta+co+tb
	DW	MinDW	MaxDW	TA	CO+CR	TB	Trip time
KAD-AYR	20	15	20	39	17	35	91
AYR-ACI	20	15	20	39	37	35	111
ACI-UNA	20	15	20	39	22	35	96
UNA-GOZ	20	15	20	39	2	35	76
GOZ-YEN	20	15	20	39	57	35	131
YEN-KOZ	20	15	20	39	20	35	94
KOZ-BOS	20	15	20	39	18	35	92
BOS-KUC	20	15	20	39	62	35	136
KUC-MAL	20	15	20	39	52	35	126
MAL-HUZ	20	15	20	39	0	35	74
HUZ-GUL	20	15	20	39	2	35	76
GUL-ESE	20	15	20	39	2	35	76
ESE-HAS	20	15	20	39	4	35	78
HAS-SOG	20	15	20	39	70	35	144
SOG-KAR	20	15	20	39	39	35	113
KAR-YAK	20	15	20	39	37	35	111
YAK-PEN	20	15	20	39	20	35	94
PEN-TAV	20	15	20	39	24	35	98
BEK-LE	120	115	125	0	0	0	
TAV-PEN	20	15	20	39	11	35	85
PEN-YAK	20	15	20	39	21	35	95
YAK-KAR	20	15	20	39	60	35	134
KAR-SOG	20	15	20	39	39	35	113
SOG-HAS	20	15	20	39	25	35	99
HAS-ESE	20	15	20	39	5	35	79
ESE-GUL	20	15	20	39	5	35	79
GUL-HUZ	20	15	20	39	8	35	82
HUZ-MAL	20	15	20	39	1	35	74
MAL-KUC	20	15	20	39	52	35	126
KUC-BOS	20	15	20	39	55	35	129
BOS-KOZ	20	15	20	39	18	35	92
KOZ-YEN	20	15	20	39	20	35	94
YEN-GOZ	20	15	20	39	44	35	118
GOZ-UNA	20	15	20	39	1	35	75
UNA-ACI	20	15	20	39	33	35	107
ACI-AYR	20	15	20	39	26	35	100
AYR-KAD	20	15	20	39	25	35	99
BEK-LE	120	115	125	0	0	0	
Toplam Bir Tur Süresi							3597

Excel dosyasının okunması ile birlikte ilk olarak iki istasyon arası yolculuk süresi(trip time) bulunur. Yolculuk süresi bir trenin bir istasyondan diğer istasyona giderken ivmelenme, frenleme ve boşta gitme sürelerinin toplanması ile bulunur. 18 istasyon, 1 adet dönüş istasyonu ve aynı zamanda trenlerin bir de ters yönlü haret etmesi ile birlikte toplamda 36 istasyon ve 2 dönüş peronu için yolculuk(trip time) hesaplanır. İstasyonlar arası yolculuk süresinin hesaplanıp toplanması ile birlikte toplam yolculuk süresi(total trip time) elde edilir. M4 hattıdnaki toplam bekleme süresinin 3597 sn olduğu Çizelge 4.3'de görülmektedir.

M4 hattının verilerine dayanarak 36(18*2) istasyon ve 2 dönüş peronu için istasyonlar arasındaki toplam bekleme süresi (dwell time) hesaplanır. M4 hattındaki toplam bekleme süresinin 960 sn olduğu Çizelge 4.3'de görülmektedir.

Bir trenin toplam tur süresi (total cycle time), toplam yolculuk süresi ve istasyonlar arasındaki toplam bekleme süresinin toplanması ile elde edilir. Bu sürede Çizelge 4.3'de görüldüğü gibi 4557 sn olarak bulunmuştur.

Simülasyonda belirlenen tren sayısı ve iki tren arasındaki sürenin (headway) çarpımı bize kullanabileceğimiz süreyi verir. Bu sürede Çizelge 4.3'de görüldüğü gibi 4914 sn olarak bulunmuştur.

Uygulamamızda amaç enerji kazancını maksimum yapacak süreleri bulmaktır. İstasyon bekleme sürelerinde yapabileceğimiz değişimi, toplam döngü süresiden kullanabileceğimiz toplam zamanı çıkarttığımızda elde ederiz. Bu süre Çizelge 4.3'de görüldüğü gibi 357 sn olarak bulunmuştur.

Çizelge 4.3. Bekleme sürelerinde yapılabilecek değişikliklerin hesaplanması

Değerler	Süreler
Toplam Tur Süresi	3597
Toplam Bekleme Süresi	960
Toplam Döndü Süresi	4557
İki Tren Arası Mesafe * Tren Sayısı	4914
Fark (İki Tren Arası Mesafe * Tren Sayısı -Toplam Döngü Süresi)	357

Excel dosyasındaki minimum ve maksimum bekleme sürelerinde değişiklik yapılarak optimizasyon yapmak istenmiştir. İstasyon bekleme sürelerindeki optimizasyon ile amaçlanan; trenlerin istasyonda bekleme sürelerinin azaltılıp artırılması ile ivmelenen trenlerle frenlemeye geçecek başka bir trenlerin kesişmesi sağlanmaktadır. Bu sayede rejeneratif enerji kazancının maksimum olması hedeflenmektedir.

Uygulamada kullanılan genetik algoritma parametreleri(popülasyon büyüklüğü, iterasyon sayısı vb.) başlangıçta Excel dosyasından okunmaktadır. Excel dosyasından okunan bu parametreler kullanıcı arayüzünde değiştirilebilmektedir.

4.3 İlk Popülasyonun Belirlenmesi

Problem de kullanılacak değişken istasyon bekleme süreleridir. Bu sebepten popülasyon istasyon bekleme sürelerinden oluşmakladır. Problemin büyüklüğü göz önüne alınarak başlangıç popülasyonu 20 bireyle oluşturulmuştur. Kromozom oluşturulurken dikkat edilmesi gereken önemli husus, istasyon bekleme sürelerindeki kısıttır. Bir trenin istasyon bekleme süresinin normal değeri 20 saniye olarak belirlenmiştir fakat trenlerin istasyonda bekleyebileceği minimum ve maksimum süreler ise sırasıyla 15 ve 25 saniyedir (-5 saniye, +5 saniye). Popülasyon oluşturulurken bekleme sürelerinin bu aralıkta olmasına dikkat edilmelidir.

Popülasyonun ilk bireyleri oluşturulurken, her istasyondaki bekleme süresini en iyileştirmek üzere, minimum (-5) ve maksimum (+5) değer aralığında rastgele bir tam sayı seçilir. Mevcut istasyon sayısı (38) kadar gen kullanılarak bireyin kromozomu oluşturulur.

4.4 İyilik Fonksiyonunun Oluşturulması

Uygulamanın simülasyon süresince, her t birim zamanı için (1 saniye) trenlerin anlık durum bilgisi; ivmelenmede (accelerating), frenlemede (breaking), boşta gidiyor (coasting) ve beklemede (dwelling) olmak üzere her tren için ayrı bir listede tutulmaktadır. Bu listeler kullanılarak, simülasyon süresinin t anında ivmelenen ve frenlenen tren sayıları bulunur. Aynı zaman dilimindeki frenlenen ve ivmelenen trenlerin sayıları birbiriyle karşılaştırılır. Frenleyen ve ivmelenen tren sayılarından minimum olanı maksimum rejeneratif enerji kullanımını ifade eder. Örneğin 100. saniyede frenleyen tren sayısı 5 iken, ivmelenen tren sayısı 3 ise 100. saniyede örtüşen tren sayısı 3 olarak (rejeneratif enerji kazancı 3 tren için sağlanır) bulunur.

Bu durum, tüm simülasyon süresi boyunca her t birim zamanında uygulanarak toplam örtüşen tren sayısı bulunur. İyilik fonksiyonu da örtüşen tren sayısının toplamının, tüm trenlerin toplam frenleme zamanına bölünmesi ile bulunur. Örneğin, simülasyon süresinin 18000 saniye olduğu ve hesaplamalar sonucunda toplam frenleme zamanının 102552 saniye ve toplam örtüşen trenlerin zamanının 71866 saniye olması durumunda iyilik fonksiyonu %0,70 (71866/102552) olarak bulunmuş olacaktır.

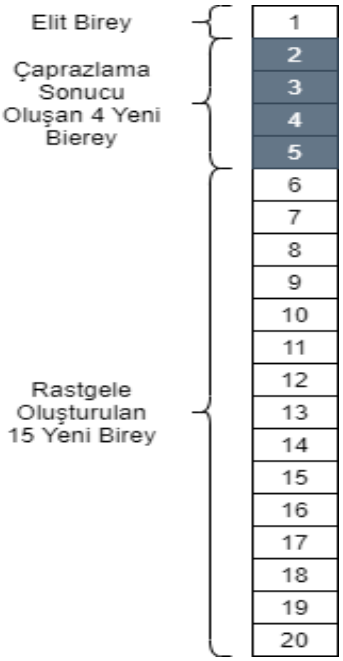
4.5 Popülasyonun Diğer Bireylerinin Belirlenmesi

Popülasyonun diğer bireylerinin belirlenmesi için çaprazlama, mutasyon ve elitizm gibi genetik algortima operatörleri kullnılmıştır. Yeni bireylerin oluşturulması için iki ayrı model geliştirilmiştir. Bu modelleri birbirinden ayıran nokta elit bireylerin belirlenmesi ve çaprazlama operatörlerindeki değişikliklerdir.

4.5.1 Elit bireyin belirlenmesi ve bir sonraki nesile aktarılması

4.5.1.1 Geliştirilen birinci modele göre elit bireyin belirlenip bir sonraki nesle aktarılması

İlk popülasyon bireyleri rastgele oluşturulduktan sonra, bu popülasyondaki bireyler yukarıda açıklanan iyilik fonksiyonu değerine göre sıralanır. En yüksek iyilik fonksiyonu değerine sahip olan birey bir sonraki nesle elit birey olarak aktarılır. Algoritmadaki elit bireylerin seçilme oranı %20 olarak belirlenmiştir (popülasyonda toplam 20 birey bulunmaktadır). Yerel maksimumlarda kalma riski oluşturmasına rağmen, algoritmanın sonuca hızlı yaklaşabilmesi adına bu değer çalışmada yüksek tutulmuştur. Yeni nesil bireylerinin mevcut popülasyondan oluşturulma şeması Şekil 3.15'de görülmektedir.

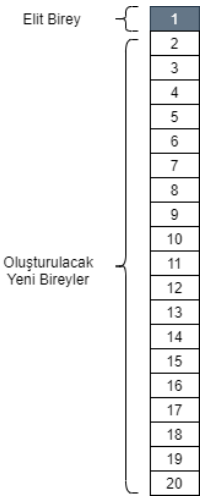


Şekil 4. 15 Geliştirilen birinci modele göre elit bireyin belirlenmesi

4.5.1.2 Geliştirilen ikinci modele göre elit bireyin belirlenip bir sonraki nesle aktarılması

İlk popülasyonda bireyler rastgele oluşturulur. Oluşturulan bireyler iyilik fonksiyonuna göre büyükten küçüğe göre sıralanır. En yüksek iyilik fonksiyonu

değerine sahip olan birey bir sonraki nesle elit birey olarak aktarılır. Algoritmadaki elit bireylerin seçilme oranı ise %5 olarak belirlenmiştir (popülasyonda toplam 20 birey bulunmaktadır) . Yerel maksimumlarda kalma riskini azaltmak üzere, elitlik oranı düşük tutulmuştur. Yeni nesil bireylerinin mevcut popülasyondan oluşturulma şeması Şekil 3.16'da görülmektedir.



Şekil 4. 16 Geliştirilen ikinci modele göre elit bireyin belirlenmesi

4.5.2 Çaprazlama operatörünün uygulanması

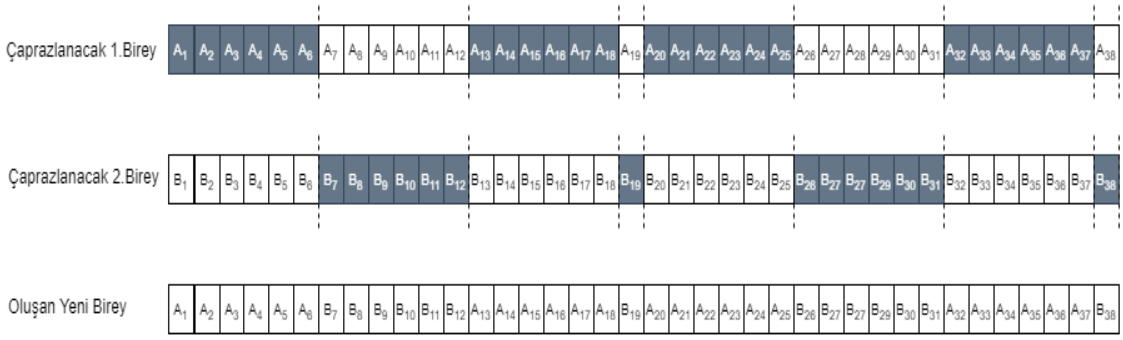
4.5.2.1 Geliştirilen birinci modele göre çaprazlama operatörünün uygulanması

İyilik fonksiyonuna göre sıralanıp, en yüksek sonucu veren 4 birey (20*0.2=4) elit olarak bir sonraki nesle aktarıldıktan sonra, diğer bireylerden 12 tanesi çaprazlama yolu ile oluşturulur. Çaprazlama için birden fazla noktadan çaprazlama yöntemi tercih edilmiştir. Çaprazlanacak bireyler elit olarak bir sonraki nesle aktarılan 4 birey olarak belirlenmiştir. Bu bireyler ile oluşabilecek tüm ikili kombinasyonlar göz önünde bulundurularak çaprazlama işlemi gerçekleştirilir (1-2,1-3,1-4,2-1,2-3,2-4,3-1,3-2,3-4,4-1,4-2,4-3).

Çaprazlama algoritması 36 istasyon ve 2 hat sonu dönüş noktası toplam 38 istasyon düşünülerek oluşturulmuştur. Çaprazlama işlemi uygulanırken hat sonu geri dönüş noktaları diğer kromozomlardan ayrı değerlendirilmiştir.

Geriye kalan 36 kromozom sırasıyla eşit olacak şekilde (36/6) altı gruba ayrılır ve hat sonu dönüş noktaları ile birlikte toplam 8 adet çaprazlama noktası elde edilmiş olur. Daha sonra yeni bireylerin kromozomlarının ilk kısmı birinci bireyin ilk çaprazlama noktasına kadar olan kromozomlarından, ikinci kısmı ikinci bireyin ikinci çaprazlama noktasına kadar olan kromozomlarından, üçüncü kısmı ilk bireyin üçüncü çaprazlama noktasına kadar olan kromozomlarından ve geri kalan kromozomlar ise sırasıyla benzer biçimde devam edilerek tamamlanır.

Elit olan ilk 4 bireyin kendi aralarında çaprazlanması sonucunda yeni 12 birey oluşturulmuştur. Popülasyon büyüklüğü 20 olarak belirlendiğinden, yeni popülasyon en elit 4 birey, çaprazlama sonucunda elde edilen 12 birey ve rastgele bir biçimde oluşturulan 4 bireyden oluşturulur. Rastgele bireylerin azlığı en iyiyi aramayı yavaşlatsa da, uygulanan agresif mutasyon operatörü yerel maksimumlardan çıkılmasını sağlamaktadır. Mutasyon operatörü, yeni bireylerin istasyonlardaki toplam bekleme süresi değişiminin sıfır olması kuralını sağlamak üzere aşağıdaki Şekil 3.17'deki gibi uygulanır.



Şekil 4. 17 Geliştirilen birinci modelde uygulanan çaprazlama operatörü

4.5.2.2 Geliştirilen ikinci modele göre çaprazlama operatörünün uygulanması

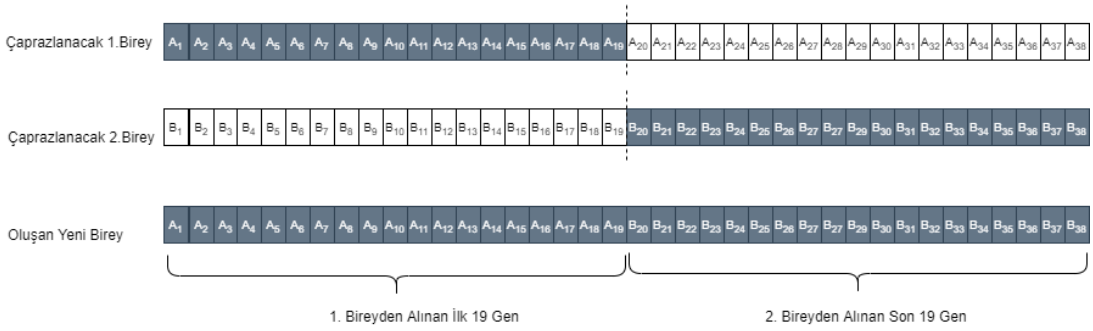
İyilik fonksiyonuna göre sıralandıktan sonra en yüksek sonucu veren birey (20*0,05=1) elit olarak bir sonraki nesle aktarılır. Daha sonra popülasyondaki diğer 4 birey çaprazlama yolu ile oluşturulur. Popülasyon büyüklüğü 20 olarak seçildiğinden diğer 15 birey rastgele olarak oluşturulur. Bu bireyler

istasyonlardaki toplam bekleme süresi değişimi sıfır olacak şekilde oluşturulur. Rastgele bireylerin çok olması en iyi aramayı hızlandırsa da öğrenmeyi yavaşlatmaktadır. Rastgele oluşturulan bireylerin fazla olması ise bizi yerel maksimum noktasından çıkarabilmektedir.

Çaprazlama için tek noktadan çaprazlama yöntemi tercih edilmiştir. Çaprazlanacak bireyler iyilik fonksiyonuna göre sıralanan bireyler arasından seçilmektedir. İyilik değeri en yüksek olan birey elit olarak aktarılır. Sıralamadaki diğer 4 birey ise atılır. 6,7,8,9 numaralı bireyler çaprazlama için seçilir (öğrenmeyi az tutmak üzere). Seçilen bu bireyler ile çaprazlanacak diğer 4 aday olarak, iyilik değeri en düşük sonucu veren 17,18,19,20 numaralı bireyler seçilir. Popülasyonda yeni nesil için çaprazlama uygulama yöntemi Şekil 3.18'de görülmektedir.

Şekil 4. 18 Geliştirilen ikinci modelde çaprazlama operatörünün uygulanması

Çaprazlama algoritmasında toplam 38 istasyon olduğundan 38 gen bulunmaktadır. Çaprazlama için tek noktadan çaprazlama yöntemi seçildiğinden ilk 19 gen bir bireyden alınırken diğer 19 gen başka bireyden alınmaktadır böylelikle çaprazlama sonucunda yeni birey elde edilmiş olur. Tek noktadan çaprazlamanın nasıl uygulandığı Şekil 3.19'da verilmiştir



Şekil 4. 19 Geliştirilen ikinci modelde uygulanan çaprazlama operatörü

Çaprazlama işlemi sonucu istasyon bekleme sürelerindeki toplam değişimin sıfır olması kuralının dışına çıkılmış olabilir. Çaprazlama sonucunda toplam istasyonda bekleme süresi sıfır olmayan bireyler mevcut ise, bu bireyler üzerinde mutasyon operatörü uygulanır.

4.6 Mutasyon Operatörünün Uygulanması İle Yeni Bireylerin Oluşturulması

Çaprazlama sonucu oluşturulan bireyler içinde toplam bekleme süresindeki değişimin sıfır olması beklenir. Toplam bekleme süresi değişiminin sıfır olmadığı durumlarda bireyler üzerinde mutasyon işlemi uygulanır.

Mutasyon işlemi istasyon bekleme sürelerindeki değişimin sıfıra eşit olmadığı durumlarda gerçekleşmektedir. Algoritma aşağıdaki gibi uygulanmaktadır.

İstasyon bekleme sürelerindeki değişim değeri bir değişkende (sayaç) tutulur.

- Bu değişken sıfırdan küçük ise;
  Bekleme sürelerindeki değişim, negatif değerli olan genlere (istasyonlara) uygulanır. Bu genlere sırayla bekleme sürelerinin toplamının tutulduğu sayaç sıfır olana kadar +1 eklenir. Tüm genler bir tur geçildiyse ve halen sayaç sıfır değil ise, bu işlem tekrarlanır.

- Bu değişken sıfırdan büyük ise;

  Bekleme sürelerindeki değişim bu sefer, pozitif değerli olduğu genlere (istasyonlara) uygulanır. Bu genlere sırayla bekleme sürelerinin toplamının tutulduğu sayaç sıfır olana kadar -1 eklenir. Tüm genler bir tur geçildiyse ve halen sayaç sıfır değil ise, bu işlem tekrarlanır.

4.7 Genetik Algoritmanın Sonlandırılması

Uygulamada algoritmayı sonlandırmak için iterasyon sayısı kullanılmıştır (hata oranı tespit edilmesi en iyi bilinmediğinden mümkün değildir). İstenen iterasyon sayısı sağlanana kadar tüm bu işlemler sırasıyla tekrarlanmaktadır.
Bu çalışmada, veri seti olarak Metro İstanbul hatlarından Kadıköy-Kartal hattının verileri kullanılmıştır. Bu hatta kullanılan trenlerin frenleme, ivmelenme, boşta gitme sürelerinin toplamı ile toplam yolculuk süresi 3597 saniyedir. Elde edilen bu süreye istasyonlarda toplam bekleme süresi toplamı olan 960 saniye eklendiğinde, bir trenin toplam parkur süresi 4557 saniyedir.
Simülasyonda toplam 18 trenin kullanıldığı öngörülmüştür. Ayrıca iki tren arasındaki mesafe 273 saniye ve toplam simülasyon süresi 18000 saniye olarak uygulanmıştır. Simülasyon tüm trenler işletilmeye başladığı andan itibaren başlatılışmış hesaplamalar bu süre zarfı içinde yapılmıştır.

4.7.1 Geliştirilen ilk model sonucu

Genetik algoritmanın 10 iterasyon (daha yüksek iterasyon sonuçlarında yapılan denemeler daha iyi sonuç vermemiştir) boyunca uygulanması sonucunda %76 bir enerji kazancı sağladığı gözlemlenmiştir. Referans alınan çalışmadaki [20] enerji kazancı miktarı %60 iken, önerilen yöntem ile mevcut durumdan yaklaşık %26 daha fazla enerji tasarrufu sağlandığı görülmektedir.

4.7.2 Geliştirilen ikinci model sonucu

Genetik algoritmanın 10 tekrar boyunca uygulanması sonucunda yaklaşık %76, 50 tekrar sonucunda %78 civarında bir enerji kazancı sağladığı gözlemlenmiştir. Tekrar sayısı arttırıldığında, sonuçlar değişmemiştir. Referans

alınan çalışmadaki [16] enerji kazancı miktarı %60 iken, önerilen yaklaşım ile yaklaşık %30 daha fazla enerji tasarrufu sağlandığı görülmektedir.

5 SONUÇ VE ÖNERİLER

Gerçekleştirilen bu çalışma ile raylı sistemlerde kullanılan rejeneratif frenleme enerjisinden maksimum fayda sağlayarak enerji tasarrufunun artırılması hedeflenmiştir. Rejeneratif frenleme enerjisi bir trenin frenlerken oluşturduğu enerjiyi hatta alıcı konumunda bulunan o an ivmelenen başka bir trene iletmesi ile gerçekleşmektedir. Bir hatta frenleyen tren sayısı ile ivmelenen tren sayısı ne kadar fazla eşleşirse rejeneratif enerjinin kullanımı da o kadar fazla olmaktadır. Çalışmada önerilen modelde, ivmelenen ve frenleyen trenlerin örtüşmesinin maksimum olmasını sağlamak üzere istasyon bekleme sürelerinde optimizasyon yapılmıştır. Genetik algoritma kullanarak enerji kazanımını maksimum yapacak istasyon bekleme süreleri bulunmuştur. Uygulama C# programlama dili ile kodlanarak geliştirilmiştir.

İstasyon bekleme süreleri belirlenirken gerçek uygulamada kullanılan iki kurala dikkat edilmiştir. İlk kural her istasyonda yeni bekleme süresi, mevcut bekleme süresinden minimum -5 saniye eksik ve maksimum +5 saniye fazla olabilir. Diğer kural ise, istasyonda bekleme sürelerinin toplamı, seyahat boyunca değişmemelidir.

Bu çalışma için Metro İstanbul hatlarından Kadıköy-Kartal hattı seçilmiş, önerilen model ile bu hattın verimliliğinin daha önce yapılan çalışmalara göre %30 daha fazla iyileştirilebileceği görülmüştür.

Gelecekte hattı besleyen trafolar göz önünde alınarak, istasyon bölgelerine göre daha detaylı ve gerçekçi modelleme yapılabilir. Algoritmada kullanılan çaprazlama ve mutasyon operatörlerinde farklı yaklaşımlarda bulunarak yeni modeller oluşturulabilir. Ayrıca istasyon bekleme sürelerindeki kısıt değerleri değiştirilerek sonuçlar gözlemlenebilir.

KAYNAKLAR

Açıkbaş S., Alataş A., 2006. Raylı Sistemlerde Enerji Verimli Sürüş ve Frenleme Enerjisinin Geri Kazanılması, Türkiye 10. Enerji Kongresi,27-30 Kasım, İstanbul.

Açıkbaş, S., 2008. Çok Hatlı Çok Araçlı Raylı Sistemlerde Enerji Tasarrufuna Yönelik Sürüş Kontrolü, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Doktora Tezi,İstanbul.

Adinolfi, A., Lamedica, R., Modesto, C., Prudenzi, A., Vimercati, S., 1998. Experimental Assesement of Energy Saving Due to Trains Regenerative Braking in an Electrified Subway Line, Transactions on Power Delivery.

Albrecht T., 2010. Reducing Power Peaks And Energy Consumption in Rail Transit Systems by Simultaneous Train Running Time Control. Transactions on State-of-the-art in Science and Engineering, 39.

Amalgamated Report, 1995. UITP, Reducing Energy Consumption in Underground Systems, International Metropolitan Railways Committe.

Amit, I., Goldfarb, D., 1971. The Timetable Problem For Railways. Developments in Operations Research, 2(1), 379-387.

Charnes, A., Miller, M.H., 1956. A Model for the Optimal Programming of Railway Freight Train Movements. Management Science, 3(1), 74-92.

Chen, J. F., Lin, R.L., Liu, Y.C., 2005. Optimization of an MRT Train Schedule: Reducing Maximum Traction Power by Using Genetic Algorithms, Transactions on Power Systems.

Cornic, D., 2010. Efficient Recovery of Braking Energy Through a Reversible DC Substation. in Electrical Systems for Aircraft, Railway and Ship Propulsion IEEE.

Çunkaş M., 2004. Elektrik Makinalarının Genetik Algoritmayla Optimizasyonu, Selçuk Üniversitesi, Fen Bilimleri Enstitüsi, Doktora Tezi, Konya.

Demirci, I.E., Celikoglu, H.B., 2018. Timetable Optimization for Utilization of Regenerative Braking Energy: A Single Line Case over Istanbul Metro Network. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC) IEEE, 19-21 October, Ankara, Turkey.

Dursun, P., 2009. Zaman Pencereli Araç Rotalama Problemi'nin Genetik Algoritma İle Modellenmesi. İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, İstanbul.

Dündar, S., Şahin, İ., 2013. Train Re-Scheduling with Genetic Algorithms and Artificial Neural Networks for Single-Track Railways. Transportation Research Part C: Emerging Technologies, 27, 1-15.

Ghoseiri, K., Szidarovszky, F., Asgharpour, M.J., 2004. A Multi-Objective Train Scheduling Model and Solution. Transportation Research Part B: Methodological, 38(10), 927-952.

Goldberg, D.E., 1989. Genetic Algorithms in Search. Optimization, and Machine Learning.

Gong, C., Zhang, S., Zhang, F., Jiang, J., Wang, X., 2014. An Integrated Energy-Efficient Operation Methodology for Metro Systems Based on a Real Case of Shanghai Metro Line One. Energies, 7(11), 7305-7329.

Gonzalez, E.L., Fernandez, M.A.R., 2000. Genetic Optimisation of a Fuzzy Distribution Model. International Journal of Physical Distribution, Logistics Management.

González-Gil, A., Palacin, R., Batty, P., 2013. Sustainable Urban Rail Systems: Strategies and Technologies for Optimal Management of Regenerative Braking Energy. Energy conversion and Management, 75, 374-388.

González-Gil, A., Palacin, R., Batty, P., Powell, J.P., 2014. A Systems Approach to Reduce Urban Rail Energy Consumption. Energy Conversion and Management, 80, 509-524.

Gunselmann, W., Godbersen, C., 2001. Double-layer Capacitors Store Surplus Braking Energy. Railway Gazette International, 1, 581.

Holland, J.H.,1975. Adaptation in Natural and Artificial Systems, 560 University of Michigan Press. Ann Arbor, MI, 561.

Kahraman, A., Özdağalar, D., 2004. Su Dağıtım Sistemlerinin Genetik Algoritma İle Optimizasyonu. Dokuz Eylül Üniversitesi Mühendislik Fakültesi Fen ve Mühendislik Dergisi, 6 (3), 1-18.

Karaboğa D., 2004. Yapay Zeka Optimizasyon Algoritmaları, Atlas Yayınları, Ankara.

Karasoy, O., Ballı, S., 2016. Google Maps ve Genetik Algoritmalarla GSP Çözümü İçin Öneri, Erişim Tarihi: 01.06.2020, https://ab.org.tr/ab16/bildiri/202.pdf

Lee, C.K., Chen, C.H., 2003. Scheduling of Train Driver for Taiwan Railway Administration. Journal of Eastern Asia Society of Transportation Studies, 5, 292-306.

Li, X., Lo, H.K. 2014. Energy Minimization in Dynamic Train Scheduling and Control for Metro Rail Operations. Transportation Research Part B, Methodological, 70, 269-284.

Martin, P., 1999. Train Performance And Simulation. In Winter Simulation Conference. 5-8 December, USA.

Murata, T., Ishibuchi, H., Tanaka, H., 1996. Genetic Algorithms for Fowshop Scheduling Problems. Computers and Industrial Engineering, 30(4), 1061-1071.

Nasri, A., Moghadam, M.F., Mokhtari, H., 2010. Timetable Optimization for Maximum Usage of Regenerative Energy of Braking in Electrical Railway Systems. In SPEEDAM 2010 IEEE.

Peña-Alcaraz, M., Fernández, A., Cucala, A. P., Ramos, A., Pecharromán, R.R., 2012. Optimal Underground Timetable Design Based on Power Flow for Maximizing the Use of Regenerative-Braking Energy. Proceedings of the Institution of Mechanical Engineers, Part F, Journal of Rail and Rapid Transit, 226(4), 397-408.

Rahimpour E., Rashtchi V., Pesaran M., 2007. Parameter Identification of Deep-Bar Induction Motors Using Genetic Algorithm, Electrical Engineering, 89(7), 547-552.

Ramos, A., Pena, M.T., Fernández, A., Cucala, P., 2008. Mathematical Programming Approach to Underground Timetabling Problem for Maximizing Time Synchronization. Dirección y Organización, (35), 88-95.

Syswerda, G., 1991. Scheduling Optimization Using Genetic Algorithms. Handbook of Genetic Algorithms, Van Nostrand Reimhold, New York.

Wrobel R., Mellor P.H., 2006. Particle Swarm Optimization for the Design of Brushless Permanent Magnet Machines, IEEE Industry Applications Conference, 41st IAS Annual Meeting.

Xu, X., Li, K., Li, X., 2016. A Multi-Objective Subway Timetable Optimization Approach With Minimum Passenger Time And Energy Consumption. Journal of Advanced Transportation, 50(1), 69-95.

Yang, X., Li, X., Gao, Z., Wang, H., Tang, T., 2012. A Cooperative Scheduling Model for Timetable Optimization in Subway Systems. IEEE Transactions on Intelligent Transportation Systems, 14(1), 438-447.

Yang, X., Ning, B., Li, X., Tang, T., 2014. A Two-Objective Timetable Optimization Model in Subway Systems. IEEE Transactions on Intelligent Transportation Systems, 15(5), 1913-1921.

Yeo, M.F., Agyei, E.O., 1998. Optimising Engineering Problems Using Genetic Algorithms. Engineering Computations, 15(2), 268-280.

Zhao, L., Li, K., Su, S., 2014. A Multi-Objective Timetable Optimization Model for Subway Systems. In Proceedings of the 2013 International Conference on Electrical and Information Technologies for Rail Transportation. Springer, Berlin, Heidelberg.

EKLER

EK A. Uygulama Kodları

EK B. Uygulama Ekran Görüntüleri

EK A. Uygulama Kodarı

Main.cs

```
using OfficeOpenXml;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using GAF;
using GAF.Operators;
using GAF.Extensions;
using System.Threading;

namespace Energy
{
   public partial class frmMain : Form
   {
      public List<Station> stationList = new List<Station>();  // istasyon bilgilerini tutan liste
      public List<Vehicle> vehicleList = new List<Vehicle>();  // tren bilgilerini tutan liste

      public List<Similarity> similarityList = new List<Similarity>(); // benzerlikleri tutan liste
      public List<Similarity> uniqueList = new List<Similarity>();
      public List<Similarity> uniqueListLast = new List<Similarity>();
      public int[] siralama = new int[20];
      public List<double> aa = new List<double>();
      public Dictionary<List<int>, double> fitnessResult = new Dictionary<List<int>, double>();

      public int headwayTime = 0;
      public int vehicleCount = 0;
      public double simulationTime = 0; // saniye cinsinden veri içerir.

      public int startSimulationTime = 0;
      public double totalSimulationTime = 0;
      public double totalTrackTime = 0; // parkur zamanını ifade eder.

      public double totalSimularTime = 0;
      public double totalBreakingTime = 0;
      public double totalAccelerationTime = 0;

      public double crossoverProbability = 0;
      public double mutationProbability = 0;
      public int elitismPercentage = 0;

      public int populationSize = 0;
      public int terminateCounter = 0;

      public int totalMinDwell;
      public int totalMaxDwell;
      public int totalDwell;

      public int minDwell;
      public int maxDwell;
      public int dwell;

      TrainCountWindow trainCountWindow;
      public int[] dwellTimeList;
      public static List<int> dwellList;
      Random random = new Random();

      public frmMain()
      {
         InitializeComponent();
```

```
        trainCountWindow = new TrainCountWindow();
        //trainCountWindow.Visible = false;
        ReadExcel(); // Excel okuma fonksiyonu

        // Toplam parkur süresini ekranda gösterme
        txtTotalTrackTime.Text = totalTrackTime.ToString();

        //Tren sayısını ekranda gösterme
        txtTrainCount.Text = vehicleCount.ToString();
        trainCountWindow.trainsCount(vehicleCount.ToString());
        trainCountWindow.Visible = false;

        //Headway süresini ekranda gösterme
        txtHeadway.Text = headwayTime.ToString();


        //Tren sayısı * Headdway süresini ekranda gösterme
        txtCalcTrainHeadway.Text = (vehicleCount * headwayTime).ToString();
        startSimulationTime = vehicleCount * headwayTime;
        //crossoverProbability ekranda gösterme
        txtCrossoverProbability.Text = crossoverProbability.ToString();

        //mutationProbability ekranda gösterme
        txtMutationProbability.Text = mutationProbability.ToString();

        // elitismPercentage ekranda gösterme
        txtElitismPercentage.Text = elitismPercentage.ToString();

        //populationSize ekranda gösterme
        txtPopulationSize.Text = populationSize.ToString();

        //Terminate ekranda gösterme
        txtTerminate.Text = terminateCounter.ToString();
        // simulasyon süresini ekranda gösterme
        txtSimulationTime.Text = simulationTime.ToString();

        // CrossoverType ekranda gösterme
        cmbCrossoverType.Items.Add("SinglePoint");
        cmbCrossoverType.Items.Add("DoublePoint");
        cmbCrossoverType.Items.Add("DoublePointOrdered");
        cmbCrossoverType.SelectedIndex = 1;

        if (headwayTime * vehicleCount < totalTrackTime)
        {
            MessageBox.Show("Headway time is not enough!" + Environment.NewLine + "HeadwayTime * VehicleCount : "
+ headwayTime * vehicleCount + " sn " + Environment.NewLine + "Total Cyclic Time : " + totalTrackTime + " sn " +
Environment.NewLine + "Difference : " + (headwayTime * vehicleCount - totalTrackTime + " sn "));
            System.Environment.Exit(-1);
        }
        else
        {
            totalSimulationTime = simulationTime + startSimulationTime;
            FindRegenerativeEnergy();
        }
    }
    public void CreateGA(int plusTime)
    {
        for (var p = 0; p < populationSize; p++)
        {
            CreateDwellTimeList(plusTime, stationList.Count, stationList).ToList(); // 1. populasyon
            EvaluateFitness(dwellList);
        }
        for (int i = 0; i < terminateCounter; i++)
        {
            List<List<int>> newPerson = new List<List<int>>();
            List<List<int>> newChromosome = new List<List<int>>();
            List<List<int>> noElitPersonList = new List<List<int>>();

            elitFind(newChromosome, noElitPersonList);
            fitnessResult.Clear();
            for (int s = 0; s < newChromosome.Count; s++)
            {
                mutation2(newChromosome[s], newPerson);
            }
```

```
            for (int q = 0; q < newPerson.Count; q++)
            {
                newChromosome.Clear();
                EvaluateFitness(newPerson[q]);
            }
        }
        Console.WriteLine(fitnessResult);
        resultWindow(fitnessResult);
    }

    int plusTime;
    private void btnGAF_Click(object sender, EventArgs e)
    {
        plusTime = headwayTime * trainCountWindow.trainCount - Convert.ToInt32(totalTrackTime);

        if (plusTime > headwayTime)
        {
            DialogResult dialog = new DialogResult();

            dialog = MessageBox.Show((int)plusTime / headwayTime + "  Fazla trenle işletme yapıyorsunuz. Tren sayısını
azaltmak istiyor musunuz?", "UYARI", MessageBoxButtons.YesNo);
            if (dialog == DialogResult.Yes)
            {

                this.txtTrainCount.Enabled = true;
                trainCountWindow.Visible = true;
            }
            else{
                CreateGA(plusTime);
            }
        }else
        {
            CreateGA(plusTime);
        }
    }

    public void randomChromosome(List<int> randomChromosomeList,List<List<int>> newChromosome)
    {
        bool contol = false;
        while (contol == false)
        {
            for (int i = 0; i < stationList.Count; i++)
            {
                int randomNumber = random.Next(stationList[i].lowerDwellTime, stationList[i].upperDwellTime);
                randomChromosomeList.Add(randomNumber);
            }
            int dwellControl = 0;

            int stationDwellTime = 0;
            int totalChangeDwellTime = 0;

            for (int j = 0; j < randomChromosomeList.Count; j++)
            {
                stationDwellTime = randomChromosomeList[j];
                totalChangeDwellTime = totalChangeDwellTime + stationDwellTime;
            }
            if (totalChangeDwellTime > plusTime)
            {
                contol = false;
            }
            if (plusTime > totalChangeDwellTime)
            {
                contol = true;
            }
            if (totalChangeDwellTime < 0)
            {
                while (totalChangeDwellTime != 0)
                {
                    int randomDwell = random.Next(0, randomChromosomeList.Count);
                    for (int j = 0; j < randomChromosomeList.Count; j++)
                    {
                        if (j == randomDwell)
                        {
                            dwellControl = randomChromosomeList[j] + 1;
```

```
                    if ((dwellControl >= -5) && (dwellControl <= +5))
                    {
                        randomChromosomeList[j] = dwellControl;
                        totalChangeDwellTime = totalChangeDwellTime + 1;
                    }
                }
            }
        }
        if (totalChangeDwellTime > 0)
        {
            while (totalChangeDwellTime != 0)
            {
                int randomDwell = random.Next(0, randomChromosomeList.Count);
                for (int j = 0; j < randomChromosomeList.Count; j++)
                {
                    if (j == randomDwell)
                    {
                        dwellControl = randomChromosomeList[j] - 1;
                        if ((dwellControl >= -5) && (dwellControl <= +5))
                        {
                            randomChromosomeList[j] = dwellControl;
                            totalChangeDwellTime = totalChangeDwellTime - 1;
                        }
                    }
                }
            }
        }
    }
    newChromosome.Add(randomChromosomeList);
}
//ilk populasyonu rastsal olarak oluşturan fonk.
private static IEnumerable<int> CreateDwellTimeList(int plusTime, int stationCount, List<Station> stationList)
//      private static IEnumerable<int> CreateDwellTimeList(int maxDwellTime, int stationCount)
{
    Random random = new Random();
    dwellList = new List<int>();
    bool contol = false;

    while (contol == false)
    {
        for (int i = 0; i < stationList.Count; i++)
        {
            int randomNumber = random.Next(stationList[i].lowerDwellTime, stationList[i].upperDwellTime);
            dwellList.Add(randomNumber);

        }
        int dwellControl = 0;

        int stationDwellTime = 0;
        int totalChangeDwellTime = 0;

        for (int j = 0; j < dwellList.Count; j++)
        {
            stationDwellTime = dwellList[j];
            totalChangeDwellTime = totalChangeDwellTime + stationDwellTime;
        }
        if (totalChangeDwellTime > plusTime)
        {
            contol = false;
        }
        if (plusTime > totalChangeDwellTime)
        {
            contol = true;
        }
        if (totalChangeDwellTime < 0)
        {
            while (totalChangeDwellTime != 0)
            {
                int randomDwell = random.Next(0, dwellList.Count);
                for (int j = 0; j < dwellList.Count; j++)
                {
                    if (j == randomDwell)
                    {
```

```
                    dwellControl = dwellList[j] + 1;
                    if ((dwellControl >= -5) && (dwellControl <= +5))
                    {
                        dwellList[j] = dwellControl;
                        totalChangeDwellTime = totalChangeDwellTime + 1;
                    }
                }
            }
        }
    }
    if (totalChangeDwellTime > 0)
    {
        while (totalChangeDwellTime != 0)
        {
            int randomDwell = random.Next(0, dwellList.Count);
            for (int j = 0; j < dwellList.Count; j++)
            {
                if (j == randomDwell)
                {
                    dwellControl = dwellList[j] - 1;
                    if ((dwellControl >= -5) && (dwellControl <= +5))
                    {
                        dwellList[j] = dwellControl;
                        totalChangeDwellTime = totalChangeDwellTime - 1;
                    }
                }
            }
        }
    }
}

int totalChangeDwellTime1 = 0;
int stationDwellTime1 = 0;
for (int e = 0; e < dwellList.Count; e++)
{
    stationDwellTime1 = dwellList[e];
    totalChangeDwellTime1 = totalChangeDwellTime1 + stationDwellTime1;
}
return dwellList;
}
public double EvaluateFitness(List<int> chromosome)
{
// uygunluk değeri hesaplama
Console.WriteLine(chromosome.ToString());//kromozomu yazdırıyorr bi bunu tutacak
double fitnessValue = -1;
try
{
    if (chromosome != null)
    {
        ClearStartValues();
        ReadExcel(); // Excel okuma fonksiyonu
        for (int i = 0; i < stationList.Count; i++)
        {
            Station station = stationList[i];
            stationList.Remove(stationList[i]);
            station.dwellTime += chromosome[i];
            stationList.Insert(i, station);
        }

        Thread thread = new Thread(() => fitnessValue = FindRegenerativeEnergyFitness());
        thread.Start();
        thread.Join();
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
fitnessResult.Add(chromosome, fitnessValue);
delta.Add(fitnessValue);
return fitnessValue;
}

public void ClearStartValues()
```

```
    {
        stationList.Clear();
        vehicleList.Clear();
        similarityList.Clear();
        uniqueList.Clear();
        uniqueListLast.Clear();
        headwayTime = 0;
        vehicleCount = 0;
        simulationTime = 0;
        totalTrackTime = 0;
        totalSimularTime = 0;
        totalBreakingTime = 0;
        totalAccelerationTime = 0;
        totalMinDwell = 0;
        totalMaxDwell = 0;
    }
    public void resultWindow(Dictionary<List<int>, double> fitnessResult)
    {
        frmResult result = new frmResult(fitnessResult);
        result.ShowDialog();
    }

    public List<List<int>> elitbirey = new List<List<int>>();

    public List<List<int>> nonelitbirey = new List<List<int>>();
    public List<List<int>> selectelit = new List<List<int>>();
    public List<double> delta = new List<double>();
    public List<double> deltas = new List<double>();


    public void elitFind(List<List<int>> newChromosome, List<List<int>> noElitPersonList)
    {
        List<List<int>> elitPerson = new List<List<int>>();
        List<List<int>> noElitPerson = new List<List<int>>();
        foreach (KeyValuePair<List<int>, double> kvp in fitnessResult)
        {
            Console.WriteLine("sırasız" + kvp.Value.ToString());
        }
        Dictionary<List<int>, double> asd = fitnessResult.OrderByDescending(key => key.Value).ToDictionary(z => z.Key,
y => y.Value);
        foreach (KeyValuePair<List<int>, double> kvp in asd)
        {
            selectelit.Add(kvp.Key);
            Console.WriteLine(kvp.Value.ToString());
        }
        for (int i = 0; i < (selectelit.Count); i++) //daha sonra dinamik yapılacak yüzdeiğiğe göre
        {
            if (i == 0)
            {
                elitPerson.Add(selectelit[i]);
            }
            if (i > 4 && i < 9)
            {
                noElitPerson.Add(selectelit[i]);
            }
            if (i > 15 && i < 20)
            {
                noElitPerson.Add(selectelit[i]);
            }
        }
        for (int i = 0; i < elitPerson.Count; i++)
        {
            newChromosome.Add(elitPerson[i]);
        }
        for (int i = 0; i < noElitPerson.Count; i++)
        {
            noElitPersonList.Add(noElitPerson[i]);
        }

        selectelit.Clear();
        noneCrossover(newChromosome, noElitPersonList);
        for (var p = 0; p < 15; p++)
        {
            List<int> randomChromosomeList = new List<int>();
```

```
            randomChromosome(randomChromosomeList,newChromosome);
        }
    }
    public void noneCrossover(List<List<int>> newChromosome, List<List<int>> noElitPersonList)
    {
        for (int i = 0; i < noElitPersonList.Count/2; i++)
        {
            List<int> firstHalf = new List<int>();
            List<int> secondHalf = new List<int>();
            List<int> cross = new List<int>();

            for (int z = 0; z < noElitPersonList[i].Count / 2; z++)
            {
                firstHalf.Add(noElitPersonList[i][z]);
            }
            for (int q = i + 4; q < noElitPersonList.Count; q++)
            {
                for (int w = noElitPersonList[i].Count / 2; w < noElitPersonList[i].Count; w++)
                {

                    secondHalf.Add(noElitPersonList[q][w]);
                }
                break;
            }

            cross = firstHalf.Concat(secondHalf).ToList();
            newChromosome.Add(cross);
        }
    }
/// <summary>
    /// Ratgele seçilen değerler ile dwell time değiştirilir mutasyona uğratılır
    /// Toplam dwelltime değerleri 0 olana kadar devam edilir.
    /// </summary>
    /// <param name="dwell4list"></param>
    public void mutation1(List<int> dwellList, List<List<int>> newPerson)
    {
        int dwellControl = 0;
        int stationDwellTime = 0;
        int totalChangeDwellTime = 0;
        Random random = new Random();


        for (int j = 0; j < dwellList.Count; j++)
        {
            stationDwellTime = dwellList[j];
            totalChangeDwellTime = totalChangeDwellTime + stationDwellTime;
        }
        Console.WriteLine(totalChangeDwellTime.ToString());
        if (totalChangeDwellTime < 0)
        {
            while (totalChangeDwellTime != 0)
            {
                int randomDwell = random.Next(0, dwellList.Count);
                for (int j = 0; j < dwellList.Count; j++)
                {
                    if (j == randomDwell)
                    {
                        dwellControl = dwellList[j] + 1;
                        if ((dwellControl >= -5) && (dwellControl <= +5))
                        {
                            dwellList[j] = dwellControl;
                            totalChangeDwellTime = totalChangeDwellTime + 1;
                        }
                    }
                }
            }
        }
        if (totalChangeDwellTime > 0)
        {
            while (totalChangeDwellTime != 0)
            {
                int randomDwell = random.Next(0, dwellList.Count);
                for (int j = 0; j < dwellList.Count; j++)
                {
```

```
            if (j == randomDwell)
            {
               dwellControl = dwellList[j] - 1;
               if ((dwellControl >= -5) && (dwellControl <= +5))
               {
                  dwellList[j] = dwellControl;
                  totalChangeDwellTime = totalChangeDwellTime - 1;
               }
            }
         }
      }
   }
   newPerson.Add(dwellList);
}
/// <summary>
/// Dwell time tomlam farkı kontrol edecek
/// Dwell time toplamları 0 dan küçük veya 0 dan büyük ise
/// Sıralı olarak sayacı artıracak veya azaltacak
/// </summary>
/// <param name="dwellList"></param>
public void mutation2(List<int> dwellList, List<List<int>> newPerson)
{
   int counter = 0;
   int stationDwellTime = 0;
   for (int j = 0; j < dwellList.Count; j++)
   {
      stationDwellTime = dwellList[j];
      counter = counter + stationDwellTime;
   }
   Console.WriteLine(counter.ToString());

   if (counter < 0)
   {
      while (counter < 0)
      {
         for (int j = 0; j < dwellList.Count; j++)
         {
            if (dwellList[j] < 0)
            {
               dwellList[j] = dwellList[j] + 1;
               counter++;
            }
         }
      }
   }
   if (counter > 0)
   {
      while (counter > 0)
      {
         for (int j = 0; j < dwellList.Count; j++)
         {
            if (dwellList[j] > 0)
            {
               dwellList[j] = dwellList[j] - 1;
               counter--;
            }
         }
      }
   }
   newPerson.Add(dwellList);
}
/// <summary>
/// Toplam fark negatif veya pozitif ise bunları içeren liste tutulur.
/// Listeden rastgele seçilerek +/-1 yapılır
/// counter 0 olana kadar devam edilir.
/// /// </summary>
/// <param name="dwellList"></param>
public void mutation3(List<int> dwellList, List<List<int>> newPerson)
{
   List<int> negativeDwell = new List<int>();
   List<int> possitiveDwell = new List<int>();
   Random random = new Random();
   int counter = 0;
   int stationDwellTime = 0;
```

```
            for (int j = 0; j < dwellList.Count; j++)
            {
                stationDwellTime = dwellList[j];
                counter = counter + stationDwellTime;
                if (dwellList[j] < 0)
                {
                    negativeDwell.Add(dwellList[j]);
                }
                if (dwellList[j] >= 0)
                {
                    possitiveDwell.Add(dwellList[j]);
                }
            }
            Console.WriteLine(counter.ToString());

            while (counter != 0)
            {
                if (counter < 0)
                {
                    int rnd = random.Next(0, negativeDwell.Count);

                    for (int i = 0; i < negativeDwell.Count; i++)
                    {
                        if (i == rnd)
                        {
                            negativeDwell[i] = negativeDwell[i] + 1;
                            counter++;
                        }

                    }
                }
                if (counter > 0)
                {
                    int rnd = random.Next(0, possitiveDwell.Count);

                    for (int i = 0; i < possitiveDwell.Count; i++)
                    {
                        if (i == rnd)
                        {
                            possitiveDwell[i] = possitiveDwell[i] - 1;
                            counter--;
                        }
                    }
                }
            }
            newPerson.Add(dwellList);
        }
        public void VechileData(List<Vehicle> vehicleList)
        {
            for (int i = 0; i < vehicleList.Count; i++)
            {
                lstVehicle.Items.Add("");
                lstVehicle.Items.Add("Train " + (i + 1) + " Values : ");

                int hcount = 0;

                for (int j = 0; j < vehicleList[i].vehicleTimeList.Count; j++)
                {
                    // bu bölüm ekrana boşluk eklemek için yazıldı
                    if (vehicleList[i].vehicleTimeList[j].headwayCount != hcount)
                    {
                        lstVehicle.Items.Add("");
                        hcount = vehicleList[i].vehicleTimeList[j].headwayCount;
                    }
                    lstVehicle.Items.Add("Train " + (i + 1) + " Departure " + vehicleList[i].vehicleTimeList[j].headwayCount + " " +
vehicleList[i].vehicleTimeList[j].stationName + " - " + vehicleList[i].vehicleTimeList[j].timeType + " - " +
vehicleList[i].vehicleTimeList[j].startValue + " - " + vehicleList[i].vehicleTimeList[j].finishValue);
                }
            }
        }
        public void FindRegenerativeEnergy()
        {
            Console.WriteLine(vehicleCount.ToString());
```

```
if (simulationTime != 0) // excel de simulasyon süresi girilmişse o zamana göre işlem yap
{
  for (int i = 0; i < vehicleCount; i++)
  {
    if (i * headwayTime < simulationTime)
    {
      Vehicle vehicle = new Vehicle(i * headwayTime, stationList, totalSimulationTime);
      vehicleList.Add(vehicle);
    }
  }
}
else // simülasyon süresi girilmemişse son tren parkuru tamamladığında bitir.
{
  //simulasyon süresi hesaplaması yapılacak

  for (int i = 0; i < stationList.Count; i++)
  {
    simulationTime += stationList[i].totalTime;
  }

  simulationTime += (vehicleCount - 1) * headwayTime;

  for (int i = 0; i < vehicleCount; i++)
  {
    if (i * headwayTime < simulationTime)
    {
      Vehicle vehicle = new Vehicle(i * headwayTime, stationList, totalSimulationTime);
      vehicleList.Add(vehicle);
    }
  }
}
//Tren verilerini ekrana yazdırma
VechileData(vehicleList);

int totalOverlop = 0;
for (int i = startSimulationTime; i < totalSimulationTime; i++)
{
  //her i anında yeni bir liste oluşturuyor buna trenlerin acceleration ve frenleme sayılarını ekliyor.
  List<String> braking = new List<String>();
  List<String> acceleration = new List<String>();
  List<int> cakisan = new List<int>();
  int overlap = 0;
  List<int> delta = new List<int>();
  int difference;

  for (int j = 0; j < vehicleList.Count; j++)
  {
    if (vehicleList[j].time.Count > i)
    {
      if (vehicleList[j].time[i] == "A") //n. saniyedeki tüm trenlerin durumunu tutar
      {
        acceleration.Add(j + ".tren " + i + "saniyede " + "A ");
        //  lstVehicleSimilarity.Items.Add(j + ".tren " + i + "saniyede " + "A ");
      }
      if (vehicleList[j].time[i] == "B")
      {
        braking.Add(j + ".tren " + i + "saniyede " + "B ");
        //lstVehicleSimilarity.Items.Add(j + ".tren " + i + "saniyede " + "B ");
      }
    }
    //çakışan tren sayısı
  }
  if (acceleration.Count >= braking.Count)
  {
    //cakisan.Add(braking.Count);
    overlap = braking.Count;
  }
  else
  {
    // cakisan.Add(acceleration.Count);
    overlap = acceleration.Count;
  }
  difference = System.Math.Abs(acceleration.Count - braking.Count);
  delta.Add(difference);
```

```
            totalOverlop = overlap + totalOverlop;
        }
        // Toplam frenleme süresini bulma
        for (int i = 0; i < vehicleList.Count; i++)
        {
            for (int j = 0; j < vehicleList[i].vehicleTimeList.Count; j++)
            {
                if (vehicleList[i].vehicleTimeList[j].timeType == Vehicle.TimeType.breaking)
                {
                    totalBreakingTime += vehicleList[i].vehicleTimeList[j].finishValue -
vehicleList[i].vehicleTimeList[j].startValue;
                }
            }
        }
        // Toplam frenleme süresini ekrana yazdırma
        txtTotalBreakingTime.Text = totalBreakingTime.ToString();

        // Toplam acceleration süresini bulma
        for (int i = 0; i < vehicleList.Count; i++)
        {
            for (int j = 0; j < vehicleList[i].vehicleTimeList.Count; j++)
            {
                if (vehicleList[i].vehicleTimeList[j].timeType == Vehicle.TimeType.acceleration)
                {
                    totalAccelerationTime += vehicleList[i].vehicleTimeList[j].finishValue -
vehicleList[i].vehicleTimeList[j].startValue;
                }
            }
        }
        // Toplam acceleration süresini ekrana yazdırma
        txtTotalAcceleration.Text = totalAccelerationTime.ToString();

        //toplam çakışma
        txtTotalSimularValue.Text = totalOverlop.ToString();
        // Toplam kazanç ekrana yazdırma
        txtGain.Text = (totalOverlop / totalBreakingTime).ToString("0.######");
    }

    public double FindRegenerativeEnergyFitness()
    {
        Console.WriteLine(vehicleCount.ToString());

        if (simulationTime != 0) // excel de simulasyon süresi girilmişse o zamana göre işlem yap
        {
            for (int i = 0; i < vehicleCount; i++)
            {
                if (i * headwayTime < simulationTime)
                {
                    Vehicle vehicle = new Vehicle(i * headwayTime, stationList, totalSimulationTime);
                    vehicleList.Add(vehicle);
                }
            }
        }
        else // simülasyon süresi girilmemişse son tren parkuru tamamladığında bitir.
        {
            //simulasyon süresi hesaplaması yapılacak

            for (int i = 0; i < stationList.Count; i++)
            {
                simulationTime += stationList[i].totalTime;
            }

            simulationTime += (vehicleCount - 1) * headwayTime;

            for (int i = 0; i < vehicleCount; i++)
            {
                if (i * headwayTime < simulationTime)
                {
                    Vehicle vehicle = new Vehicle(i * headwayTime, stationList, simulationTime);
                    vehicleList.Add(vehicle);
                }
            }
        }
        int totalOverlop = 0;
```

```
        for (int i = startSimulationTime; i < totalSimulationTime; i++)
        {
            //her i anında yeni bir liste oluşturuyor buna trenlerin acceleration ve frenleme sayılarını ekliyor.
            List<String> braking = new List<String>();
            List<String> acceleration = new List<String>();
            int overlap = 0;
            List<int> delta = new List<int>();
            int difference;
            for (int j = 0; j < vehicleList.Count; j++)
            {
                if (vehicleList[j].time.Count > i)
                {
                    if (vehicleList[j].time[i] == "A") //n. saniyedeki tüm trenlerin durumunu tutar
                    {
                        acceleration.Add(j + ".tren " + i + "saniyede " + "A ");
                        //  lstVehicleSimilarity.Items.Add(j + ".tren " + i + "saniyede " + "A ");
                    }
                    if (vehicleList[j].time[i] == "B")
                    {
                        braking.Add(j + ".tren " + i + "saniyede " + "B ");
                        //lstVehicleSimilarity.Items.Add(j + ".tren " + i + "saniyede " + "B ");
                    }
                }
                //çakışan tren sayısı
            }
            if (acceleration.Count >= braking.Count)// acceleration/hızlanma
            {
                //cakisan.Add(braking.Count);
                overlap = braking.Count;
            }
            else
            {
                // cakisan.Add(acceleration.Count);
                overlap = acceleration.Count;

            }
            difference = System.Math.Abs(acceleration.Count - braking.Count);
            delta.Add(difference);


            totalOverlop = overlap + totalOverlop;
        }
        for (int i = 0; i < vehicleList.Count; i++)
        {
            for (int j = 0; j < vehicleList[i].vehicleTimeList.Count; j++)
            {
                if (vehicleList[i].vehicleTimeList[j].timeType == Vehicle.TimeType.breaking)
                {
                    totalBreakingTime += vehicleList[i].vehicleTimeList[j].finishValue -
vehicleList[i].vehicleTimeList[j].startValue;
                }
            }
        }
        return totalOverlop / totalBreakingTime;

    }

    public struct Station
    {
        public string stationName { get; set; }
        public int dwellTime { get; set; }
        public int accelerationTime { get; set; }
        public int constantTime { get; set; }
        public int breakingTime { get; set; }
        public int totalTime { get; set; }
        public int minDwellTime { get; set; }
        public int maxDwellTime { get; set; }
        public int lowerDwellTime { get; set; }
        public int upperDwellTime { get; set; }

    }

    public enum Behaviour
```

```
    {
        dwell, acceleration, constant, breaking
    }

    public class Similarity
    {
        public int firstArrayNumber;
        public Behaviour firstArrayBehaviour;
        public int secondArrayNumber;
        public Behaviour secondArrayBehaviour;
        public double start;
        public double finish;
        public bool visible = true;       }

    private void ReadExcel()
    {
        try
        {
            string path = @"Input.xlsx";

            FileInfo contactFile = new FileInfo(path);
            using (ExcelPackage package = new ExcelPackage(contactFile))
            {
                ExcelWorksheet worksheet = package.Workbook.Worksheets[1];

                int rows = worksheet.Dimension.End.Row;

                for (int i = 7; i <= rows; i++)
                {
                    #region istasyon verileri okuma
                    if (worksheet.Cells[i, 1].Value != null)
                    {
                        if (worksheet.Cells[i, 1].Value.ToString().Contains('-')) // İçerideki dönüş süresi yazısını ve null değerleri
atlatmak için yazıldı
                        {
                            Station station = new Station();
                            station.stationName = worksheet.Cells[i, 1].Value.ToString();
                            station.dwellTime = Convert.ToInt32(worksheet.Cells[i, 3].Value.ToString());
                            station.minDwellTime = Convert.ToInt32(worksheet.Cells[i, 4].Value.ToString());
                            station.maxDwellTime = Convert.ToInt32(worksheet.Cells[i, 5].Value.ToString());
                            station.accelerationTime = Convert.ToInt32(worksheet.Cells[i, 6].Value.ToString());
                            station.constantTime = Convert.ToInt32(worksheet.Cells[i, 7].Value.ToString());
                            station.breakingTime = Convert.ToInt32(worksheet.Cells[i, 8].Value.ToString());
                            station.totalTime = station.dwellTime + station.accelerationTime + station.constantTime +
station.breakingTime;
                            station.upperDwellTime = Convert.ToInt32(worksheet.Cells[i, 5].Value.ToString()) -
Convert.ToInt32(worksheet.Cells[i, 3].Value.ToString());
                            station.lowerDwellTime = Convert.ToInt32(worksheet.Cells[i, 4].Value.ToString()) -
Convert.ToInt32(worksheet.Cells[i, 3].Value.ToString());

                            stationList.Add(station);
                        }
                    }
                    #endregion
                }

                //Headway süresi okuma
                if (worksheet.Cells[7, 22].Value != null)
                {
                    headwayTime = Convert.ToInt32(worksheet.Cells[7, 22].Value.ToString());
                    //Console.WriteLine("Headway Time : " + headwayTime);
                }

                //Tren sayısı okuma
                if (worksheet.Cells[8, 22].Value != null)
                {
                    if (trainCountWindow.approveCheck == true)
                    {
                        vehicleCount = trainCountWindow.trainCount;
                    }
                    else { vehicleCount = Convert.ToInt32(worksheet.Cells[8, 22].Value.ToString()); }

                    //Console.WriteLine("Vehicle Count : " + vehicleCount);
```

```
        }

        //Simülasyon süresi okuma
        if (worksheet.Cells[10, 22].Value != null)
        {
            simulationTime = Convert.ToInt32(worksheet.Cells[10, 22].Value.ToString());
            //Console.WriteLine("Simulation Time : " + simulationTime);
        }

        //CrossoverProbability okuma
        if (worksheet.Cells[14, 22].Value != null)
        {
            crossoverProbability = Convert.ToDouble(worksheet.Cells[14, 22].Value.ToString());
            //Console.WriteLine("crossoverProbability : " + crossoverProbability);
        }

        //mutationProbability okuma
        if (worksheet.Cells[15, 22].Value != null)
        {
            mutationProbability = Convert.ToDouble(worksheet.Cells[15, 22].Value.ToString());
            //Console.WriteLine("mutationProbability : " + mutationProbability);
        }

        //elitismPercentage okuma
        if (worksheet.Cells[16, 22].Value != null)
        {
            elitismPercentage = Convert.ToInt32(worksheet.Cells[16, 22].Value.ToString());
            //Console.WriteLine("elitismPercentage : " + elitismPercentage);
        }

        //populationSize okuma
        if (worksheet.Cells[17, 22].Value != null)
        {
            populationSize = Convert.ToInt32(worksheet.Cells[17, 22].Value.ToString());
            //Console.WriteLine("populationSize : " + populationSize);
        }

        //Terminate okuma
        if (worksheet.Cells[18, 22].Value != null)
        {
            terminateCounter = Convert.ToInt32(worksheet.Cells[18, 22].Value.ToString());
            //Console.WriteLine("terminate : " + terminateCounter);
        }

        // Toplam parkur süresini ölçme
        for (int i = 0; i < stationList.Count; i++)
        {
            totalTrackTime += stationList[i].totalTime;
        }
        //toplam min ve max sürelerini bulma
        for (int i = 0; i < stationList.Count(); i++)
        {

            minDwell = stationList[i].lowerDwellTime;
            totalMinDwell = totalMinDwell + minDwell;
            maxDwell = stationList[i].upperDwellTime;
            totalMaxDwell = totalMaxDwell + maxDwell;
            dwell = stationList[i].dwellTime;
            totalDwell = totalDwell + dwell;
        }
    }
  }
  catch (Exception ex)
  {
    MessageBox.Show("Excel reading error!" + Environment.NewLine + ex.ToString());
  }
}
private void btnExcel_Click(object sender, EventArgs e)
{
  try
  {
    ExcelPackage ExcelPkg = new ExcelPackage();
    ExcelWorksheet wsSheet1 = ExcelPkg.Workbook.Worksheets.Add("Energy");
```

```
            wsSheet1.Cells[1, 1].Value = "Tren Verileri";

            for (int i = 0; i < lstVehicle.Items.Count; i++)
            {
               wsSheet1.Cells[(i + 2), 1].Value = lstVehicle.Items[i].ToString();
            }

            wsSheet1.Cells[1, 2].Value = "Benzerlikler";

            //for (int i = 0; i < lstVehicleSimilarity.Items.Count; i++)
            //{
            //   wsSheet1.Cells[(i + 2), 2].Value = lstVehicleSimilarity.Items[i].ToString();
            //}

            wsSheet1.Cells[1, 3].Value = "Filtrelenmiş Benzerlikler";

            //for (int i = 0; i < lstVehicleSimilarityFiltered.Items.Count; i++)
            //{
            //   wsSheet1.Cells[(i + 2), 3].Value = lstVehicleSimilarityFiltered.Items[i].ToString();
            //}

            // Fit the columns according to its content
            wsSheet1.Column(1).AutoFit();
            wsSheet1.Column(2).AutoFit();
            wsSheet1.Column(3).AutoFit();

            wsSheet1.Protection.IsProtected = false;
            wsSheet1.Protection.AllowSelectLockedCells = false;
            string path = @"Output123.xlsx";
            ExcelPkg.SaveAs(new FileInfo(path));
         }
         catch (Exception ex)
         {
            MessageBox.Show("Excel reading error!" + Environment.NewLine + ex.ToString());
         }
      }

   }
}
```

Vehicle.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Energy
{
   public class Vehicle
   {
      public static int yLow = 0;
      public static int yHigh = 80;
      public int hCount = 0;
      public List<StationContent> vehicleTimeList = new List<StationContent>();
      public List<String> time = new List<String>();

      public Vehicle(double startTime, List<frmMain.Station> stationList, double simulationTime)
      {
         for (int i = 0; i < startTime; i++)
         {
            time.Add("0");
         }
         while (startTime < simulationTime)
         {
            //simulationtime 184910 start 4910 olunca tutmaya etikete başla
            hCount++;

            for (int i = 0; i < stationList.Count; i++)
            {
               StationContent t1 = new StationContent();
               t1.stationName = stationList[i].stationName;
```

```
t1.timeType = TimeType.dwell;
t1.startValue = startTime;
t1.finishValue = startTime + stationList[i].dwellTime;
t1.headwayCount = hCount;
if (t1.finishValue <= simulationTime)
{
   vehicleTimeList.Add(t1);
   for (int z = 0; z < stationList[i].dwellTime; z++)
   {
      time.Add("D");
   }
}
else
{
   t1.finishValue = simulationTime;
   startTime = simulationTime;
   vehicleTimeList.Add(t1);
   int cntl = 0;
   while (time.Count != startTime)
   {
      if (cntl == stationList[i].dwellTime)
      {
         break;
      }
      else
      {
         time.Add("D");
         cntl++;
      }
   }
   break;
}
StationContent t2 = new StationContent();
t2.stationName = stationList[i].stationName;
t2.timeType = TimeType.acceleration;
t2.startValue = t1.finishValue;
t2.finishValue = t1.finishValue + stationList[i].accelerationTime;
t2.headwayCount = hCount;
if (t2.finishValue <= simulationTime)
{
   vehicleTimeList.Add(t2);
   for (int z = 0; z < stationList[i].accelerationTime; z++)
   {
      time.Add("A");//hızlanma
   }
}
else
{
   t2.finishValue = simulationTime;
   startTime = simulationTime;
   vehicleTimeList.Add(t2);
   int cntrl = 0;
   while (time.Count != startTime)
   {
      if (cntrl == stationList[i].accelerationTime)
      {
         break;
      }
      else
      {
         time.Add("A");
         cntrl++;
      }
   }
   break;
}
StationContent t3 = new StationContent();
t3.stationName = stationList[i].stationName;
t3.timeType = TimeType.constant;
t3.startValue = t2.finishValue;
t3.finishValue = t2.finishValue + stationList[i].constantTime;
t3.headwayCount = hCount;
if (t3.finishValue <= simulationTime)
{
```

```
            vehicleTimeList.Add(t3);
            for (int z = 0; z < stationList[i].constantTime; z++)
            {
              time.Add("C");//boşta gitme
            }
          }
          else
          {
            t3.finishValue = simulationTime;
            startTime = simulationTime;
            vehicleTimeList.Add(t3);
            int cntrl = 0;

            while (time.Count != startTime)
            {
              if (cntrl == stationList[i].constantTime)
              {
                break;
              }
              else
              {
                time.Add("C");
                cntrl++;
              }
            }
            break;
          }
          StationContent t4 = new StationContent();
          t4.stationName = stationList[i].stationName;
          t4.timeType = TimeType.breaking;

          t4.startValue = t3.finishValue;
          t4.finishValue = t3.finishValue + stationList[i].breakingTime;
          t4.headwayCount = hCount;
          if (t4.finishValue <= simulationTime)
          {
            vehicleTimeList.Add(t4);
            for (int z = 0; z < stationList[i].breakingTime; z++)
            {
              time.Add("B");
            }
          }
          else
          {
            t4.finishValue = simulationTime;
            startTime = simulationTime;
            vehicleTimeList.Add(t4);
            int cntrl = 0;
            while (time.Count != startTime)
            {
              if (cntrl == stationList[i].breakingTime)
              {
                break;
              }
              else
              {
                time.Add("B");
                cntrl++;
              }
            }
            break;
          }
          startTime += stationList[i].totalTime;
        }
      }
    }
    public enum TimeType
    {
      dwell, acceleration, constant, breaking
    }
    public struct StationContent
    {
      public string stationName;
      public double startValue;
```

```
        public double finishValue;
        public TimeType timeType;
        public int headwayCount;
    }   }
}
```

Simulation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Energy
{

  public class Simulation
  {
    public string stationName { get; set; }
    public int dwellTime { get; set; }
    public int accelerationTime { get; set; }
    public int constantTime { get; set; }
    public int breakingTime { get; set; }
    public int totalTime { get; set; }
    public int minDwellTime { get; set; }
    public int maxDwellTime { get; set; }
    public int lowerDwellTime { get; set; }
    public int upperDwellTime { get; set; }

  }
}
```

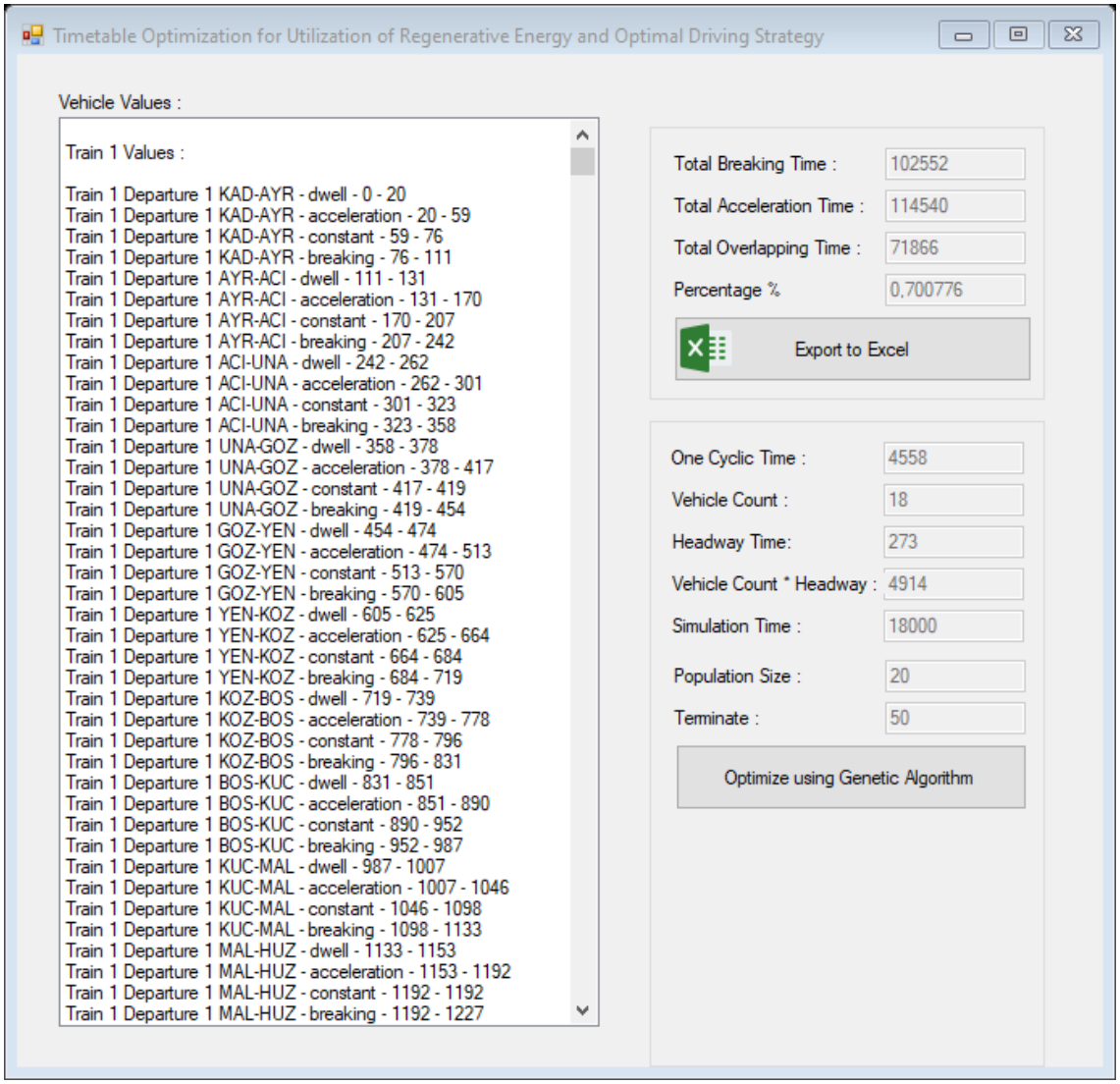
ExcelRead.cs

```
using OfficeOpenXml;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using GAF;
using GAF.Operators;
using GAF.Extensions;
using System.Threading;
namespace Energy
{
  public class ExcelRead : Simulation{
    public List<Simulation> stationList = new List<Simulation>();
    public int headwayTime = 0;
    public int vehicleCount = 0;
    public int simulationTime = 0;
    public double crossoverProbability = 0;
    public double mutationProbability = 0;
    public int elitismPercentage = 0;
    public int populationSize = 0;
    public int terminateCounter = 0;
    public int totalTrackTime = 0;
    public int totalMinDwell;
    public int totalMaxDwell;
    public int totalDwell;
    public int totalCyleTime;
    public int minDwell;
    public int maxDwell;
    public int dwell;
    public void content()
```

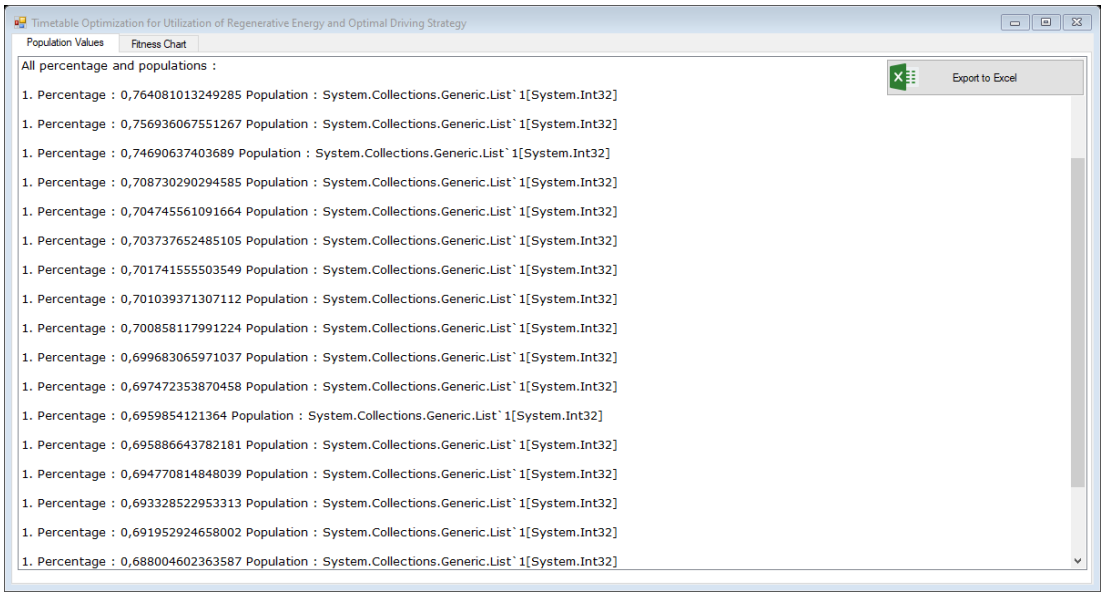
```
    { try{
        string path = @"Input.xlsx";
        FileInfo contactFile = new FileInfo(path);
        using (ExcelPackage package = new ExcelPackage(contactFile))
        {
          ExcelWorksheet worksheet = package.Workbook.Worksheets[1];
          int rows = worksheet.Dimension.End.Row;
          for (int i = 7; i <= rows; i++)
          {
            #region istasyon verileri okuma
            if (worksheet.Cells[i, 1].Value != null)
            {
              if (worksheet.Cells[i, 1].Value.ToString().Contains('-')) // İçerideki dönüş süresi yazısını ve null değerleri
atlatmak için yazıldı
                {
                  Simulation simulation = new Simulation();
                  simulation.stationName = worksheet.Cells[i, 1].Value.ToString();
                  simulation.dwellTime = Convert.ToInt32(worksheet.Cells[i, 3].Value.ToString());
                  simulation.minDwellTime = Convert.ToInt32(worksheet.Cells[i, 4].Value.ToString());
                  simulation.maxDwellTime = Convert.ToInt32(worksheet.Cells[i, 5].Value.ToString());
                  simulation.accelerationTime = Convert.ToInt32(worksheet.Cells[i, 6].Value.ToString());
                  simulation.constantTime = Convert.ToInt32(worksheet.Cells[i, 7].Value.ToString());
                  simulation.breakingTime = Convert.ToInt32(worksheet.Cells[i, 8].Value.ToString());
                  simulation.totalTime = simulation.dwellTime + simulation.accelerationTime + simulation.constantTime
+ simulation.breakingTime;
                  simulation.upperDwellTime = Convert.ToInt32(worksheet.Cells[i, 5].Value.ToString()) -
Convert.ToInt32(worksheet.Cells[i, 3].Value.ToString());
                  simulation.lowerDwellTime = Convert.ToInt32(worksheet.Cells[i, 4].Value.ToString()) -
Convert.ToInt32(worksheet.Cells[i, 3].Value.ToString());
                  stationList.Add(simulation);
                }
            }
            #endregion
          }
          //Headway süresi okuma
          if (worksheet.Cells[7, 22].Value != null) {
            headwayTime = Convert.ToInt32(worksheet.Cells[7, 22].Value.ToString());
          }
          //Tren sayısı okuma
          if (worksheet.Cells[8, 22].Value != null)
          {
            { vehicleCount = Convert.ToInt32(worksheet.Cells[8, 22].Value.ToString()); }
          }
          if (worksheet.Cells[11, 26].Value != null)
          {
            { totalDwell = Convert.ToInt32(worksheet.Cells[8, 22].Value.ToString()); }
          }
          //Simülasyon süresi okuma
          if (worksheet.Cells[10, 22].Value != null)
          {
            simulationTime = Convert.ToInt32(worksheet.Cells[10, 22].Value.ToString());
          }
          //CrossoverProbability okuma
          if (worksheet.Cells[14, 22].Value != null)
          {
            crossoverProbability = Convert.ToDouble(worksheet.Cells[14, 22].Value.ToString());
          }
          //populationSize okuma
          if (worksheet.Cells[17, 22].Value != null)
          {
            populationSize = Convert.ToInt32(worksheet.Cells[17, 22].Value.ToString());
          }
          //Terminate okuma
          if (worksheet.Cells[18, 22].Value != null)
          {
            terminateCounter = Convert.ToInt32(worksheet.Cells[18, 22].Value.ToString());
          }
          // Toplam parkur süresini ölçme
          for (int i = 0; i < stationList.Count; i++)
          {
            totalTrackTime += stationList[i].totalTime;
          }
          //toplam min ve max sürelerini bulma
          for (int i = 0; i < stationList.Count(); i++)
```

```
                {
                    minDwell = stationList[i].lowerDwellTime;
                    totalMinDwell = totalMinDwell + minDwell;
                    maxDwell = stationList[i].upperDwellTime;
                    totalMaxDwell = totalMaxDwell + maxDwell;
                    dwell = stationList[i].dwellTime;
                    totalDwell = totalDwell + dwell;
                }
                }
            }
        }
        catch (Exception ex) {
            MessageBox.Show("Excel reading error!" + Environment.NewLine + ex.ToString());
        }
    }
  }
}
```

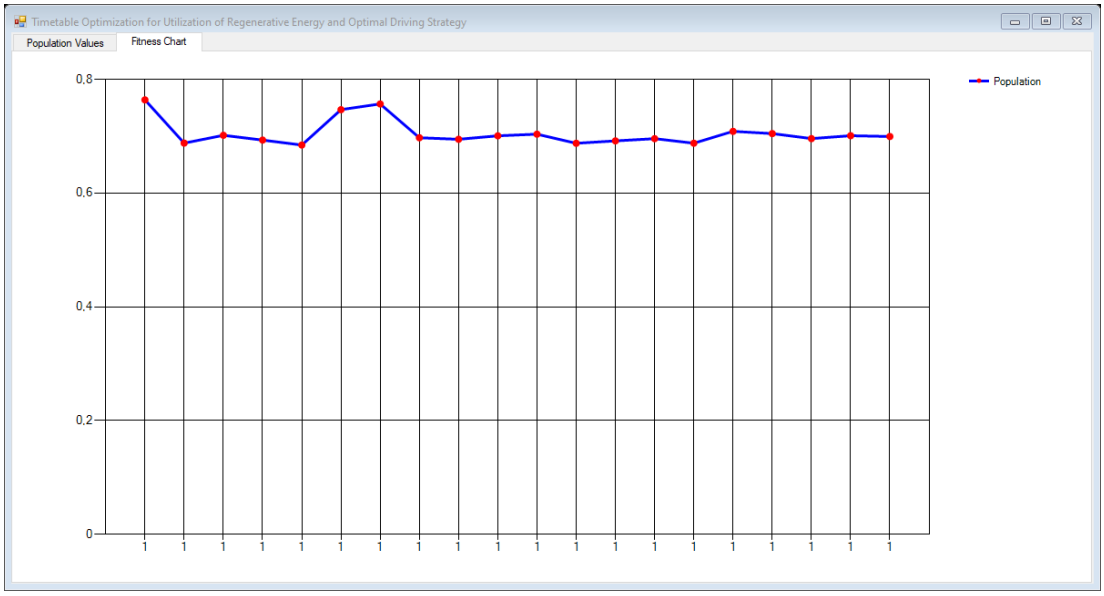
EK B. Uygulama Ekran Görüntüleri



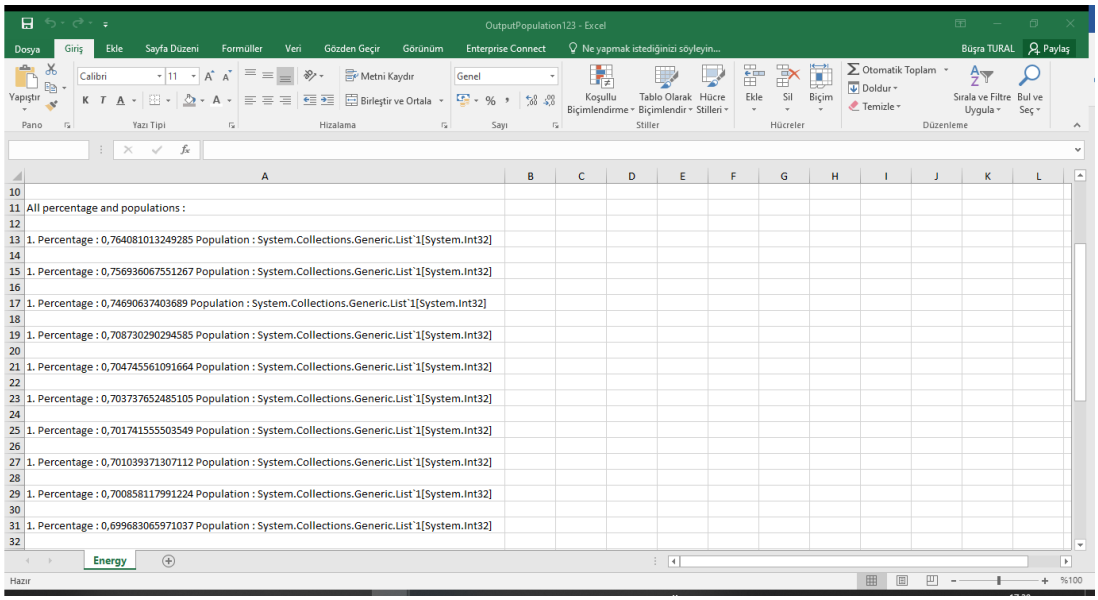
Şekil B. 20 Geliştirilen uygulamanın açılış ekran görüntüsü



Şekil B. 21 Genetik algoritma sonucu elde edilen popülasyonların iyilik fonksiyonların değerlerine göre sıralanması

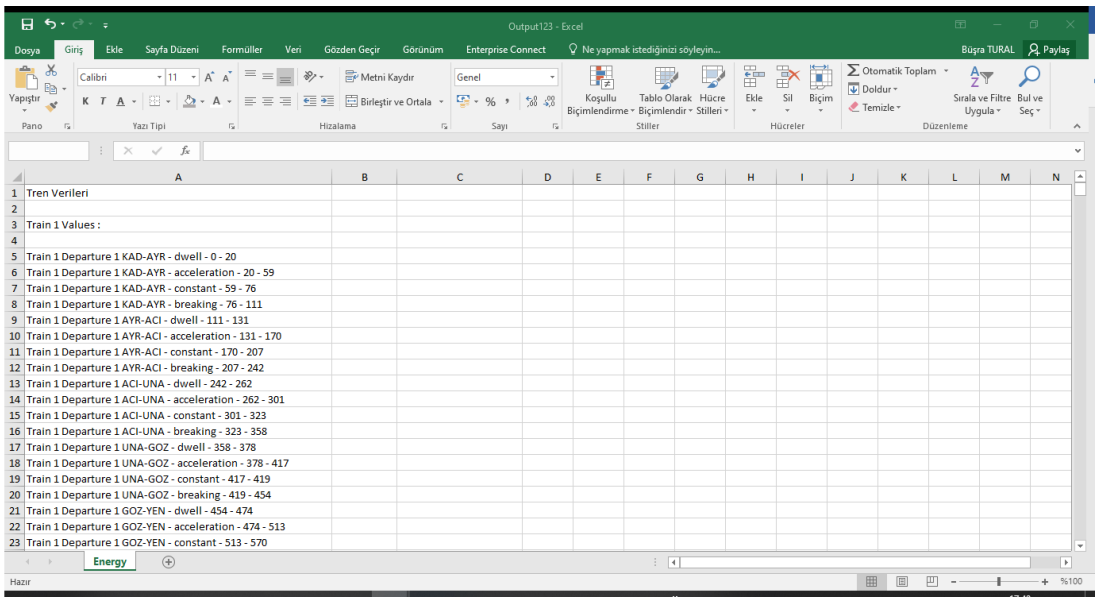


Şekil B. 22 Genetik algoritma sonucu elde edilen popülasyonların iyilik fonksiyonların değerlerinin grafiği



	A
10	
11	All percentage and populations :
12	
13	1. Percentage : 0,764081013249285 Population : System.Collections.Generic.List`1[System.Int32]
14	
15	1. Percentage : 0,756936067551267 Population : System.Collections.Generic.List`1[System.Int32]
16	
17	1. Percentage : 0,74690637403689 Population : System.Collections.Generic.List`1[System.Int32]
18	
19	1. Percentage : 0,708730290294585 Population : System.Collections.Generic.List`1[System.Int32]
20	
21	1. Percentage : 0,704745561091664 Population : System.Collections.Generic.List`1[System.Int32]
22	
23	1. Percentage : 0,703737652485105 Population : System.Collections.Generic.List`1[System.Int32]
24	
25	1. Percentage : 0,701741555503549 Population : System.Collections.Generic.List`1[System.Int32]
26	
27	1. Percentage : 0,701039371307112 Population : System.Collections.Generic.List`1[System.Int32]
28	
29	1. Percentage : 0,700858117991224 Population : System.Collections.Generic.List`1[System.Int32]
30	
31	1. Percentage : 0,699683065971037 Population : System.Collections.Generic.List`1[System.Int32]
32	

Şekil B. 23 Genetik algoritma sonucu elde edilen popülasyonların iyilik fonksiyonların değerlerinin excel dosyasına aktarılması



	A
1	Tren Verileri
2	
3	Train 1 Values :
4	
5	Train 1 Departure 1 KAD-AYR - dwell - 0 - 20
6	Train 1 Departure 1 KAD-AYR - acceleration - 20 - 59
7	Train 1 Departure 1 KAD-AYR - constant - 59 - 76
8	Train 1 Departure 1 KAD-AYR - breaking - 76 - 111
9	Train 1 Departure 1 AYR-ACI - dwell - 111 - 131
10	Train 1 Departure 1 AYR-ACI - acceleration - 131 - 170
11	Train 1 Departure 1 AYR-ACI - constant - 170 - 207
12	Train 1 Departure 1 AYR-ACI - breaking - 207 - 242
13	Train 1 Departure 1 ACI-UNA - dwell - 242 - 262
14	Train 1 Departure 1 ACI-UNA - acceleration - 262 - 301
15	Train 1 Departure 1 ACI-UNA - constant - 301 - 323
16	Train 1 Departure 1 ACI-UNA - breaking - 323 - 358
17	Train 1 Departure 1 UNA-GOZ - dwell - 358 - 378
18	Train 1 Departure 1 UNA-GOZ - acceleration - 378 - 417
19	Train 1 Departure 1 UNA-GOZ - constant - 417 - 419
20	Train 1 Departure 1 UNA-GOZ - breaking - 419 - 454
21	Train 1 Departure 1 GOZ-YEN - dwell - 454 - 474
22	Train 1 Departure 1 GOZ-YEN - acceleration - 474 - 513
23	Train 1 Departure 1 GOZ-YEN - constant - 513 - 570

Şekil B. 24 Trenlerin hareket detaylarının excel dosyasına aktarılması

ÖZGEÇMİŞ

Adı Soyadı : Büşra TURAL

Doğum Yeri ve Yılı : İSTANBUL, 15/09/1995

Medeni Hali : (Bekar)

Yabancı Dili : İngilizce

E-posta : busra.tural@istanbulticaret.edu.tr

Eğitim Durumu

Lise : Halit Armay Lisesi, 2013

Lisans : İstanbul Ticaret Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 2017

Yüksek Lisans : İstanbul Ticaret Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı

Mesleki Deneyim

Metro İstanbul A.Ş.
Ar-Ge Mühendisi 2018-…(devam ediyor)

Yayınları

Tural, B., Turan, M., 2020. Tren Frenleme Enerjisinin Maksimum Geri Kazanımı için Zaman-Planı Optimizasyonu, 2nd International Congress on Human-Computer Interaction, Optimization and Robotic Applications, 26-27 Haziran, Ankara.