Tek Yönlü Dairesel Bir Üretim Hattında Sıra Bağımlı Aile Ayar Süreleri Yaklaşımı ile Çizelgeleme

Fatma Sibel ERYAŞAR

YÜKSEK LİSANS TEZİ

Endüstri Mühendisliği Anabilim Dalı

Ağustos 2010

Scheduling At A Unidirectional Circular Manufacturing Line With Sequence Dependent Family Setup Times

Fatma Sibel ERYAŞAR

MASTER OF SCIENCE THESIS

Department of Industrial Engineering

August 2010

Tek Yönlü Dairesel Bir Üretim Hattında Sıra Bağımlı Aile Ayar Süreleri Yaklaşımı ile Çizelgeleme

Fatma Sibel ERYAŞAR

Eskişehir Osmangazi Üniversitesi
Fen Bilimleri Enstitüsü
Lisansüstü Yönetmeliği Uyarınca
Endüstri Mühendisliği Anabilim Dalı
Endüstri Mühendisliği Bilim Dalında
YÜKSEK LİSANS TEZİ
Olarak Hazırlanmıştır

Danışman: Yrd.Doç.Dr. Servet Hasgül

Ağustos 2010

ONAY

Endüstri Mühendisliği Anabilim Dalı Yüksek Lisans öğrencisi Fatma Sibel ERYAŞAR'ın YÜKSEK LİSANS tezi olarak hazırladığı "Tek Yönlü Dairesel Bir Üretim Hattında Sıra Bağımlı Aile Ayar Süreleri Yaklaşımı ile Çizelgeleme" başlıklı bu çalışma, jürimizce lisansüstü yönetmeliğin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

Danışman **:** Yrd. Doç. Dr. Servet HASGÜL

İkinci Danışman **:** -

Yüksek Lisans Tez Savunma Jürisi:

Üye : Prof. Dr. Nihat YÜZÜGÜLLÜ

Üye : Doç. Dr. Nuray GİRGİNER

Üye : Yrd. Doç. Dr. Servet HASGÜL

Üye : Yrd. Doç. Dr. Aykut ARAPOĞLU

Üye : Yrd. Doç. Dr. İnci SARIÇİÇEK

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Prof. Dr. Nimetullah BURNAK

Enstitü Müdürü

ÖZET

Bu çalışma tek yönlü dairesel bir üretim hattına sahip bir üretim işletmesinde yaşanan bir çizelgeleme problemi üzerine yapılandırılmıştır. Problem iki aşamalı olarak analiz edilmiştir. İlk aşamada ürün aileleri seçimi problemi, ulaştırma modeline benzetilerek modellenmiştir. İkinci aşamada ise seçilen ürün ailelerinin çizelgelenmesi problemi, karesel atama problemine benzetilerek modellenmiştir. Yapılan denemelerle çizelgeleme problemi için geliştirilen modelin boyutunun problem boyutunun artmasıyla hızla arttığı gözlenmiştir. Bu nedenle problem çözümü için bir genetik algoritma geliştirilmiş ve önerilen bu genetik algoritma ile geliştirilen matematiksel modelin performansları, farklı boyuttaki problemler kullanılarak karşılaştırılmıştır. Sonuç olarak genetik algoritmanın her problem boyutu için uygun çözümler üretebildiği görülmüştür.

Anahtar Kelimeler: Çizelgeleme, sıra bağımlı aile ayar süresi, genetik algoritma

SUMMARY

This study is configured on a scheduling problem which appears in a unidirectional circular manufacturing line of a company. The problem is analyzed in two stages. At the first stage a mathematical model is developed for the problem of choosing job families as a transportation model. At the second stage, scheduling the chosen job families problem with sequence dependent family setup times is modelled as a quadratic assignment problem. Test problems showed that as the increasing number of job families, the size of the scheduling problem is increasing rapidly. Therefore a genetic algorithm is developed and the performance of the mathematical model and the genetic algorithm compared using various number of job families. In conclusion it is shown that the proposed genetic algorithm gives feasible solutions for each problem having different sizes.

Keywords: Scheduling, sequence dependent family setup times, genetic algorithm

TEŞEKKÜR

Gerek derslerimde gerekse tez çalışmalarında, bana danışmanlık eden Sayın Yrd. Doç. Dr. Servet HASGÜL' e teşekkür ederim.

Ayrıca her zaman yanımda olan ve desteklerini hep arkamda hissettiğim aileme sosuz sevgi ve teşekkürlerimi sunarım.

İÇİNDEKİLER

ÖZET **v**
SUMMARY **vi**
TEŞEKKÜR **vii**
İÇİNDEKİLER **viii**
ŞEKİLLER DİZİNİ **x**
ÇİZELGE DİZİNİ **xi**

1. GİRİŞ **1**

2. AİLE AYAR SÜRESİ YAKLAŞIMI İLE ÇİZELGELEME PROBLEMİ VE MEVCUT ÇALIŞMALAR **4**

2.1 Çizelgelemenin Tanımı ve Çizelgeleme Problemleri 4

2.2 Tanımlamalar ve gösterimler 5

2.3 Aile Ayar Süresi Yaklaşımı ile Çizelgeleme 7

2.4 Aile Ayar Süresi Yaklaşımı İle Çizelgeleme Alanında Yapılmış Mevcut Çalışmalar 9

3. TEK YÖNLÜ DAİRESEL BİR ÜRETİM HATTINDA AİLE AYAR SÜRESİ YAKLIŞIMI İLE ÇİZELGELEME PROBLEMİ **14**

3.1 Problemin Tanımlanması 14

3.2 Ürün Aileleri Seçimi Problemi ve Geliştirilen Matematiksel Model 15

3.3 Seçilen Ürün Gruplarının Çizelgelenmesi Problemi 18

3.3.1 Problemin sınıfı ve özellikleri 18

3.3.2 Varsayımlar 20

3.3.3 Ürün ailelerinin çizelgelenmesi için geliştirilen matematiksel model 20

4. İŞ AİLELERİ ÇİZELGELEME PROBLEMİNİN ÇÖZÜMÜNE İLİŞKİN GENETİK ALGORİTMA YAKLAŞIMI 24

4.1 Genetik Algoritmalar Hakkında Genel Bilgi 24

4.1.1 Kromozom gösterimi 26

4.1.2 Seçim prosedürü 27

4.1.3 Çaprazlama Çeşitleri 28

4.1.4 Mutasyon 30

4.1.5 Genetik algoritma parametreleri 31

4.1.6 Ürün Aileleri Çizelgeleme Problemine İlişkin Geliştirilen Genetik Algoritma 31

4.2 Geliştirilen Genetik Algoritmanın Performansı 32

5. SONUÇ VE ÖNERİLER 34

KAYNAKLAR DİZİNİ 36

ŞEKİLLER DİZİNİ

Şekil **Sayfa**

Şekil 3.1 Tek yönlü dairesel üretim hattı şeması 15

Şekil 3.2 Üretim hattı şeması üzerinde ürün ailelerinin rotalarının gösterimi. 19

Şekil 3.3 Problem boyutunun artmasıyla kısıt ve karar değişkeni sayılarındaki artışlar.23

Şekil 4.1 Konum tabanlı çaprazlama örneği 30

ÇİZELGE DİZİNİ

Çizelge **Sayfa**

Çizelge 4.1 Test problemleri üzerinde önerilen genetik algoritma ile LINGO 8.0' da geliştirilen matematiksel modelin performansları 33

BÖLÜM 1

1. GİRİŞ

Çizelgeleme, zaman kısıtı altında, gereken eylemlere kısıtlı kaynakların tahsis edilmesi ile ilgilenen ve bir veya daha fazla amacı eniyilemeye çalışan bir karar verme sürecidir (Pinedo, 2002).

Rekabet koşulları gitgide zorlaşan günümüz dünyasında kaynakların etkin kullanımı, bu zorlu yolda rakiplerden bir adım önde olmanın vazgeçilmez ve en önemli noktası haline gelmiştir. Gelinen bu nokta seri üretimin ekonomik avantajlarının fark edilmesine olanak sağlamış ve bu gelişmeler çizelgeleme alanında grup teknolojisi ile hücresel imalatı dikkat çeken konular haline getirmiştir.

İmalat sistemlerinde grup teknolojisinin sıkça kullanılmaya başlanması, araştırmalarda ürün aileleri çizelgeleme konusunun ortaya çıkmasını sağlamıştır.

Ürün aileleri çizelgeleme probleminde, işler ayar süresi gerekliliğine göre ailelere ayrılmaktadırlar. Ardışık olarak işlendiklerinde aralarında ayar süresi gereken işler ise farklı ailelerde yer almaktadırlar. Başka bir ifade ile kendisinden önce çizelgelenen iş ile sıradaki iş eğer aynı aileye ait ise bu iki iş arasında ayar süresi söz konusu olmazken, art arda işlenecek olan işler aynı aileye ait değil ise bu iki iş arasında ayar süresi söz konusu olmaktadır.

Bu çalışmada bir çikolata fabrikasında bulunan tek yönlü dairesel bir üretim hattında karşılaşılan bir çizelgeleme problemi ele alınmıştır. Çalışmaya konu olan üretim hattında dairesel yapısı ve proses planları sebebiyle üretim hattında aynı anda işlenebilecek ve aralarında ayar süresi gerekmeyen ürünlerden ürün aileleri oluşturulmuştur. Öncelikle ürünlere ilişkin talep miktarları belirlendikten sonra daha önce oluşturulmuş ürün aileleri içinden hangilerinin üretim için seçileceğine ve seçilen

ürün ailelerinin hangi sırada üretileceğine karar verilmektedir. Daha açık bir ifade ile mevcut ürün aileleri içinden talepleri karşılayacak şekilde hangisinin üretileceği problemi ve ardından üretimine karar verilen ürün ailelerinin çizelgelenmesi problemi ile karşılaşılmaktadır.

Mevcut işleyişte, işletmede bu seçim ve çizelgeleme işlemi karar vericinin sezgilerine dayanılarak yapılmaktadır. Sonuç olarak gereksiz stok oluşumu ve üretimde uzun süreli kopukluklara neden olan aksaklıklarla karşılaşılmaktadır.

Bu çalışmada ele alınan problem, üretim hattının yapısı itibariyle aynı anda aynı makineyi kullanmama esasına dayanılarak oluşturulmuş olan ürün aileleri arasından talep bilgilerine göre seçim yapma ve üretimine karar verilen ürün ailelerinin çizelgelenmesi problemi olmak üzere iki aşamalı olarak incelenmiştir. İlk aşamada ürün aileleri seçimi problemi, ulaştırma modeline benzetilerek modellenmiştir. İkinci aşamada ise seçilen ürün ailelerinin çizelgelenmesi problemi, karesel atama problemine benzetilerek modellenmiştir. Geliştirilen çizelgeleme modeli LINGO 8.0 programında, problemin farklı boyutları için çalıştırılmıştır. Yapılan bu denemelerle, çizelgeleme problemi için geliştirilen modelin karesel yapısı nedeniyle, problemin boyutunun ürün ailesi sayısının artmasıyla hızla arttığı gözlenmiştir. Bu nedenle problem çözümü için bir genetik algoritma geliştirilmiş ve önerilen bu genetik algoritma ile geliştirilen matematiksel modelin performansları, farklı boyuttaki problemler kullanılarak karşılaştırılmıştır.

Sonuç olarak LINGO 8.0' ın problem boyutu 30 iş ailesine ulaştıktan sonra uygun bir çözüm bulmada yetersiz kaldığı, buna karşılık önerilen genetik algoritmanın her problem boyutu için uygun çözümler üretebildiği görülmüştür.

Bu bilgiler ışığında çalışmanın birinci bölümünde konuya genel bir giriş yapılmış, ikinci bölümde aile ayar süreleri yaklaşımı hakkında bilgilere ve bu konuda literatürde yer alan çalışmalara yer verilmiştir. Üçüncü bölümde çalışmaya konu olan problem ayrıntıları ile ele alınmıştır ve incelenen probleme ilişkin geliştirilen matematiksel modele yer verilmiştir. Dördüncü bölümde ise problemin çözümü için

önerilen genetik algoritmanın ayrıntılarına ve genetik algoritmalar hakkında genel bilgilere yer verilmiştir. Beşinci bölümde yapılan çalışmanın sonuçları tartışılmıştır.

BÖLÜM 2

2. AİLE AYAR SÜRESİ YAKLAŞIMI İLE ÇİZELGELEME PROBLEMİ VE MEVCUT ÇALIŞMALAR

2.1 Çizelgelemenin Tanımı ve Çizelgeleme Problemleri

Çizelgeleme, üretim ve hizmet endüstrilerinde çok önemli rol oynayan bir karar verme problemidir. Mevcut rekabetçi çevrede etkili çizelgeleme, pazarda ayakta kalabilmek için bir gereklilik haline gelmiştir (Pinedo, 2002).

Çizelgeleme, istenilen çıktıların elde edilmesi için yerine getirilmesi gereken tüm eylemlerin kaynaklar ile eylemler arasındaki ilişkisel kısıtların ve zaman kısıtlarının sağlanarak seçilmesi, örgütlenmesi ve kaynak kullanımlarının zamanlaması sürecidir (Sipper ve Bulfin, 1997).

Pinedo (2002), çizelgelemeyi sistem kaynaklarının çeşitli işlere, görevlere ve faaliyetlere zaman temelinde tahsis edilmesi olarak tanımlamıştır. Kaynaklar ve eylemler çok değişik şekillerde olabilmektedir. Kaynaklar bir atölyedeki makineler, hava alanında kalkış pistleri, inşaat sahasında çalışanlar, bilgisayarda işlemci olabilirken, eylemler ürünlerin mamule dönüşüm sürecinde geçmesi gereken işlemler, hava alanındaki kalkış ve inişler, yapı projelerinin aşamamaları, bir bilgisayar programının çalışması olabilmektedir. Her eylemin farklı öncelikleri, başlama zamanları ya da teslim zamanları olabileceği gibi amaçlar da farklı şekillerde oluşabilmektedir (Pinedo, 2002).

Çizelgeleme hem imalat sektöründe hem de hizmet sektöründe önemli bir yere sahiptir. Rekabetin oldukça zorlaştığı günümüz şartlarında pazarda var olabilmede ve rakiplerle mücadele etmede etkin bir çizelgeleme yapabilmek, dolayısıyla iyi bir zaman

yönetimi yaparak işleri söz verilen ya da müşterinin istediği zamanda tamamlamak, organizasyonlara ayırt edici nitelik kazandırmaktadır.

2.2 Tanımlamalar ve gösterimler

Çizelgeleme problemlerinde sıklıkla kullanılan tanımlamalar ve gösterimlerden bazıları aşağıda verilmiştir.

- **İşlem süresi (p_{ij}) :** i. makinede işlenecek j işinin işlenmesi için gerekli süredir. Eğer j işinin işlemesinde makinelerden bağımsız ya da tek makina söz konusu ise i indisinin kullanılmasına gerek yoktur.

- **Hazır olma zamanı (r_j):** j işinin işlenebilmesi için işin işlenmeye hazır olması gerekmektedir ve bu tanımlama işin hazır olduğu zamanı belirtmektedir.

- **Teslim zamanı (d_j) :** j işinin müşteri tarafından istenen ya da müşteriye teslim edileceğine dair verilen zamanı ifade etmektedir.

- **Ağırlık (w_j) :** j işinin ağırlığı temel olarak j işinin diğer işlere göre öneminin bir ifadesidir.

- **Tamamlanma zamanı (C_j):** j işinin tamamlanma zamanını belirten bir ifadedir.

- **Akış süresi (F_j) :** j işinin hazır olma zamanı ile tamamlanma zamanı arasında geçen işin sistemde kaldığı süreyi belirtir.

- **Sapma (L_j) :** j işinin tamamlanma zamanı ile teslim zamanı arasında geçen süreyi belirtir, pozitif ya da negatif değer alabilir.

- **Erkenlik (E_j):** j işinin teslim zamanından önce tamamlanmasının ölçüsüdür.

- **Gecikme (T_j):** j işinin teslim zamanından sonra tamamlanmasının bir ölçüsüdür.

Çizelgeleme problemleri makine ortamı, işlem özellikleri ve performans ölçütlerine göre sınıflandırılırlar. Bu sınıflandırma için de $\alpha/\beta/\gamma$ gösterimi kullanılır. Burada alfa(α) makine ortamını, beta(β) işlem özelliklerini ve gama(γ) da ulaşılmak istenen amacı ifade etmektedir. Makine ortamı makine sayısını, makine yerleşim tipleri gibi özellikleri belirtmek için kullanılır. İşlem özellikleri ise genel hatları ile problemin kısıtlarını tarifler. Sık kullanılan kısıtlara ait örnekler aşağıda belirtilmiştir:

- **Sıra bağımlı ayar süreleri (s_{ij}):** *i* ve *j* işleri arasında sıra bağımlı ayar süreleri söz konusu olduğunda ayar süreleri sıralamaya bağlı olarak farlılık gösteriyor demektir. Daha açık bir ifade ile *i* işinin ardından *j* işi işleneceğinde ortaya çıkacak ayar süresi *j* işinin ardından i işinin işlenmesi durumunda ortaya çıkacak ayar süresine eşit olmak durumunda değildir ($s_{ij} \neq s_{ji}$).
- **İşlerin bölünebilir olması (*prmp*):** işlerin bölünebilir olması bir işin işlenmeye başlanmasından tamamlanıncaya kadar tek bir makine tarafından işlenmesinin zorunlu olmadığını gösterir. Herhangi bir nedenle bölünebilir işin işlenmesine ara verilebilir ya da paralel başka bir makinede işlenmesine devam edilebilir.
- **Öncelik kısıtları (*prec*) :** öncelik kısıtları işlerin işlenebilmesi için öncüllük ve ardıllık ilişkilerinin söz konusu olduğu durumlarda ortaya çıkarlar.
- **Permutasyon (*prmu*):** bazı akış tipi üretim ortamlarında makineye gelen işlerin sırasının başlangıçtan bitişe kadar (İlk Giren İlk Çıkar) değişmemesi gereken durumu tarifler

Çizelgeleme problemlerinde sıklıkla kullanılan performans ölçütlerinden bazıları şöyle sıralanabilir:

- **En büyük tamamlanma zamanının (C_{max}):** en büyük tamamlanma zamanı sistemi terk eden son işin tamamlanma zamanını gösterir ve n adet iş olduğu varsayıldığında $max(C_1,..., C_n)$ ile ifade edilir.
- **En büyük sapma (L_{max}):** en kötü teslim zamanının ölçüsüdür ve n adet iş olduğu varsayıldığında $max(L_1,..., L_n)$ ile ifade edilir.

- **Toplam ağırlıklı tamamlanma zamanı ($\sum w_jC_j$):** çizelgede bulunan işlerin ağırlıklı tamamlanma zamanlarının toplamıdır.
- **Ağırlıklı geciken iş sayısı ($\sum w_jU_j$):** geciken iş sayısının ağırlıklandırılmış toplamını ifade eder.
- **Toplam gecikme ($\sum T_j$):** her işe ait gecikme zamanlarının toplamıdır.

2.3 Aile Ayar Süresi Yaklaşımı ile Çizelgeleme

Hücresel imalat (Cellular Manufacturing, CM), küçükten büyüğe tüm imalat ortamlarında, grup teknolojisinin en önemli uygulamalarından biri olmuştur. Hücresel imalat, atölye tipinin esnekliğini ve akış tipinin verimliliğini birleştirerek, seri üretimde olduğu gibi, küçük yığınlar halinde üretimde de ekonomik avantaj kazandırmaya olanak sağlamaktadır. Bu alandaki çalışmalar hücresel imalatın ayar sürelerini, süreç içi stok miktarlarını, taşıma maliyetlerini ve üretim planlama ve kontrol maliyetlerini azaltmada avantaj sağladığını kanıtlamıştır (Lin vd., 2009).

Endüstriyel imalat sistemlerinde grup teknolojisinin sıkça kullanılmaya başlanmasıyla, çalışmalarda ürün ailelerini çizelgeleme konusu aktif olarak işlenmeye başlamıştır (Webster ve Baker, 1995).

Hücresel imalat ortamlarında makineler ve işler benzerliklerine göre kendi içlerinde sınıflandırılırlar ve gruplara ayrılmaktadırlar. Bu gruplardan makine grupları bir imalat hücresini oluştururken, bu imalat hücrelerinde işlenebilecek parça grupları da ilgili imalat hücrelerine atanmaktadırlar.

Hücresel imalatta oluşturulan parça grupları ile ürün ailelerinin oluşturulma biçimleri itibariyle benzer özellik göstermektedir. Aile çizelgeleme modelinde de, grup teknolojisinde olduğu gibi, işler art arda işlendiklerinde ayar süresi gerekip gerekmemesine göre aileler oluşturacak şekilde gruplanmaktadırlar. Aynı ailede yer alan işler art arda işlendiklerinde aralarında ayar süresi gerekmemektedir. Daha açık bir ifade ile aynı aileye ait iki ayrı iş arka arkaya işlendiğinde ayar süresi oluşmazken, farklı ailelere ait iki iş art arda işlendiğinde ayar süresi gerekmektedir (Gen ve Cheng, 2000).

Örneğin işler aynı makinelerde işleneceklerse ve işlerin işlemleri arasında malzeme, kalıp değişimi gibi zaman alacak bir ayarlama söz konusu değil ise bu işler aynı ailede yer alabilmektedir. Eğer farklı maddelerden üretiliyor ve makinede her üründe farklı malzeme kullanımı söz konusu oluyor ise ya da makinenin temizlenmesi veya malzeme değişimi yapmak gerekli olabilmektedir. Bu durumda farklı maddelerden üretilen ürünler birbirinden farklı iş ailelerine aatanacaklardır ve bu farklı iş ailelerinin birinin üretiminden diğerine geçilirken makinede malzeme değişimi için ayar süresi gerekecektir.

Diğer bir deyişle iş ailesi, bir makinede art arda işlenecek işlerin oluşturduğu ve tek bir ayar süresini paylaşan en büyük kümedir. İş ailelerinin boyutu büyüdükçe, tek seferlik ayar süresi söz konusu olacağından, makine kullanım oranı yükselmektedir. Diğer taraftan büyük boyutlu ailelerin işlenmesinde, işlem süresi uzun olacağından, başka bir aileye ait bir işin işlenmesi gecikebilecektir. Bu nedenle oluşturulacak iş ailelerinin boyutunun belirlenmesi de büyük bir öneme sahiptir.

İş ailelerinin çizelgelenmesi probleminde, çizelgenin başlangıcında ve makine bir ailenin imalatından başka bir ailenin imalatına geçerken ayar süresi gerekecektir. Eğer ayar süreleri sıra bağımsız *(sequence independent)* ise o zaman *i.* makinede *f* ailesi için ayar süresi s_{if} ile ifade edilebilir. Diğer taraftan, sıra bağımlı *(sequence-dependent)* ayar süreleri söz konusu ise o zaman *g* ailesinin, *f* ailesinden sonra işlenecek olduğu varsayıldığında ayar süresi s_{fg} ile ifade edilir. Ayrıca her farklı *f, g, h* aileleri için sıra bağımlı ayar süresi, $s_{fh} \leq s_{fg} + s_{gh}$ ile açıklanan üçgen eşitsizliği *(triangle inequality)* varsayımına uymaktadır (Gosh ve Gupta, 1997).

Mitrofanov (1966) ve Burbidge (1971)'in bu alandaki ilk çalışmalarının ardından, imalat hücresi çizelgeleme problemleri alanında birçok çalışma yapılmıştır. İmalat hücresi çizelgeleme problemlerinin çözümü için, şu anda da geçerliliğini sürdüren algoritmalar genel olarak iki kategoride incelenebilmektedir:

- Kesin çözümler,
- Yakınsak sezgiseller.

Bu problemleri içerik olarak farklılaştırmak kolay olmasına rağmen, sıra bağımlı aile ayar süreleri yaklaşımı ile akış tipi imalat hücresi çizelgeleme problemi iki makine olması durumunda bile NP-zor problem sınıfında olduğu kanıtlanmıştır. Problem boyutunun artmasıyla hesaplama zamanı üstel olarak arttığı ve bu nedenle kombinatoryal karmaşıklık ve zaman kısıtları araştırmacıları, eniyiye yakın çözümler yakalayabilmeleri için yakınsak sezgisel algoritmalara yöneltmiştir (Lin vd., 2009).

2.4 Aile Ayar Süresi Yaklaşımı İle Çizelgeleme Alanında Yapılmış Mevcut Çalışmalar

Birçok aile çizelgeleme modeli için, aile içinde yer alan işlerin sıralamasının yapılması mümkündür. Örneğin, Monma ve Potts (1989), $1/s_{fg}/L_{max}$ problemi için ve teslim zamanından önce veya teslim zamanında biten işler için $1/s_{fg}/\sum w_j U_j$ probleminde, her ailedeki işler kendi içlerinde erken teslim kuralına *(earliest due date-EDD)* göre sıralandığında optimal bir çizelgenin mümkün olduğunu göstermişlerdir. $1/s_{fg}/\sum w_j U_j$ problemi için teslim zamanından önce ya da tam teslim tarihinde bitirilmesi gereken işler *(on-time jobs)* seçilir. $1/s_{fg}/\sum w_j C_j$ problemi için ise aile içinde işlerin en küçük ağırlıklı işlem süresi *(Weighted Smallest Processing Time-WSPT)* kuralına göre sıralamışlardır. Bu problemleri çözmek için farklı ailelerden sipariş edilmiş işleri bir çizelge oluşturmak için birleştirmişlerdir.

Aile ayar süresi yaklaşımı ile çizelgeleme problemlerinde dinamik programlama algoritmaları ileri ya da geri yaklaşımlı olmalarına göre farklılaştırılarak kullanılmıştır. İleri dinamik programlama algoritmalarında, başlangıçtaki kısmi çizelgeler daha önceki kısmi çizelgelerin sonuna bir iş ekleyerek oluşturulmuştur. Diğer taraftan geri yaklaşımlı dinamik programlama algoritmaları son kısmi çizelgenin başına bir iş eklenmesiyle oluşturulmuştur.

Monma ve Potts (1989), her ailede sıralanmış işlerin sayılarını, kısmi çizelgenin tamamlanma zamanının hesaplanmasını mümkün kılmak için "ağırlıklandırılmış geciken iş sayısının" bir durum değişkeni ve termin zamanından önce ya da termin zamanında tamamlanan kısmi işlerin tamamlanma zamanının bir fonksiyon değeri olduğu, ileri yaklaşımlı bir dinamik programlama algoritması geliştirmişlerdir. Bu

algoritmanın, $1/s_{fg}/L_{max}$ ve $1/s_{fg}/\sum w_jC_j$ problemleri için algoritmalara $O(F^2n^{F\wedge 2+2F})$ ve $W=\sum_{j=1}^{n} w_j$ *iken* $O(F^2n^FW)$ karmaşıklığa sahip olduğunu göstermişlerdir.

Bununla birlikte, Gosh ve Gupta (1994) ve Gosh (1997) ayar sürelerini durum değişkeni olmasını engellemek için geri yaklaşımlı dinamik programlama algoritması geliştirmişlerdir. Bu yaklaşımla, $1/s_{fg}/L_{max}$ ve $1/s_{fg}/\sum w_jC_j$ problemlerinin herbirinin çözüm zamanı karmaşıklığını $O(F^2n^F)$' e indirdiklerini göstermişlerdir.

$1/s_{fg}/L_{max}$ ve $1/s_{fg}/\sum U_j$ problemleri sabit bir F değeri için yukarıda belirtildiği gibi polinom zamanda çözülebilirdir. Girdilerin bir parçası olan aile sayısı F' in rassal seçilmiş değerleri için Bruno ve Downey (1978), $1/s_{fg}/L_{max}$ ve $1/s_{fg}/\sum U_j$ problemlerinin NP-zor olduğunu kanıtlamışlardır. Benzer şekilde $1/s_{fg}/\sum w_jC_j$ problemi sabit F değeri için polinom zamanda dinamik programlama ile çözülebilirdir ve Gosh (1994), $1/s_{fg}/\sum C_j$ probleminin rassal olarak seçilmiş F değerleri için "strongly" NP-zor olduğunu göstermiştir. Böylece $1/s_{fg}/\sum C_j$ ve $1/s_{fg}/\sum w_jC_j$ problemlerinin karmaşıklık derecesi rassal olarak seçilmiş aile sayısı olan F değerine bağlıdır.

Paralel makinalar için, Monma ve Potts (1989), $P2/s_f, pmtn/C_{max}$ probleminin NP-zor ve Webster (1997), $P/s_f/\sum C_j$ probleminin strongly NP- zor olduğunu göstermiştir.

Çok aşamalı problemler için, Kleinau (1993), $O2/s_f/C_{max}$ probleminin 3 aile için bile NP-zor olduğunu göstermiştir. Ayrıca $F2/s_f/C_{max}$ problemi için ise farklı aile sayısı değerleri için NP-zor olduğunu ve bu problem için optimum çözümün ancak bir permutasyon çizelge ile var olduğunu göstermiştir.

Bunlara ek olarak, Monma ve Potts (1989), her ailedeki işlerin Johnson algoritmasına göre sıralanabileceğini göstermişlerdir. Böylece, Potts ve Van Wassenhove' un (1992) gösterdiği gibi, sabit bir aile sayısı olan F değeri için $F2/s_f/C_{max}$ problem tipinin çözümü için polinom zamanlı dinamik programlama algoritması geliştirilmiştir.

Ayar süresi olmadan tek makinalı L_{max} problemi araştırılmış ve bu problemin en basit halinin, $1//L_{max}$, erken teslim kuralı ile çözümünün $O(nlogn)$ zamanda mümkün

olabileceği sonucuna ulaşılmıştır (Pinedo,2000). Aynı probleme öncelik kısıtı eklendiğinde ise *(1/prec/L_{max})* Lawler' s algoritması(1973) kullanılarak *O(n^2)* zamanda çözülebileceği hesaplanmıştır (Uzsoy ve Velasquez, 2008). Bruno ve Downey (1973), rassal oluşturulmuş *m* adet ailenin yer aldığı problemin NP-hard olduğunu göstermiştir.

Baker ve Magazine (2000), işlerin iş aileleri oluşturacak şekilde gruplandırıldığında ve bu iş ailelerinin kendi içlerinde erken teslim kuralına göre sıralandığında optimum çözümün bulunması için göz önünde bulundurulması gereken çizelge sayısının *n!*'den $\frac{n!}{n1\,!\,n2\,!\,n3\,!\ldots nf\,!}$'e düştüğünü kanıtlamıştır.

Uzsoy vd. (1991), *1/prec,* s_{ij} */* L_{max} problemi için bir dal sınır algoritması önermiştir. Bu yaklaşımla 15 iş' e kadar olan problemlerde çalıştırabilmişlerdir.

Webster ve Baker (1995), "Yığınlar için Erken Teslim Kuralı" adında erken teslim kuralına göre sıralanmış yığınların kendi aralarında erken teslim kuralına göre artan düzende sıralanmasıyla en büyük gecikmeyi *(L_{max})* hesaplayan bir sezgisel önermişlerdir.

Baker (1998), boşluk *(gap)* sezgiseli adında, grup teknolojisi tekniğiyle oluşturulmuş aile yığınlarını bölerek, çizelgeye her seferinde bir iş ekleyerek ilerleyen bir sezgisel önermişlerdir.

Hariri ve Potts (1997), *1/*s_f */*L_{max} problemi için bir dal-sınır algoritması önermişlerdir. Başlangıç alt sınır değeri, her ailenin ilk işi ile ilgili ayar süresi dışındaki tüm ayar süreleri yok sayılarak elde edilmiş ve nihai problem erken teslim kuralına göre sıralayarak çözülmüştür. Alt sınır değeri ise ailelerin iki veya daha fazla yığına bölünüp bölünmemesine göre bir prosedür tarafından belirlenmiştir. Ailedeki tüm işlerin tek bir yığında birleştirmiş, birleşik sezgisel yaklaşım geliştirmişlerdir. Aynı zamanda bir aileyi teslim zamanlarına göre iki yığında toplayan başka bir sezgisel algoritma da kullanmışlardır. Yaptıkları denemelerle 50 işe kadar olan problemleri çözmede etkili olmuşlardır. Hesaplama sonuçları 60 işe kadar algoritmanın başarılı sonuçlar verdiğini ortaya koymuştur.

Schutten, Van de Velde and Zijm (1996), $1/s_f, r_j/L_{max}$ problemi için bir dal sınır algoritması geliştirmişlerdir. Hazır olma zamanlarının varlığında, aile içindeki işlerin sırası hakkında herhangi bir sonuç bildirilmemiştir. Bu algoritmanın farklı yanı ayar sürelerini belirtmek için farazi işler kullanıyor olmasıdır. Hızlıca hesaplanabilen alt sınır değerleri, ayar sürelerinin gevşetilmesi, ilgili bölünebilir problemin çözülmesi ve ileri yönlü bir dallandırma kuralının kullanılmasıyla elde edilir. Hesaplama sonuçları bu algoritmanın 40 işe kadar olan örnekleri çözmede etkili olduğunu göstermiştir.

Mason and Anderson (1991) ile Crauwels, et al.. (1997), $1|s_f|\sum w_jC_j$ problemleri için dal-sınır algoritmaları geliştirmişlerdir. Mason ve Anderson ileri dallandırma yaparak dal-sınır arama ağacının boyutunu sınırlandıracak baskınlık kurallarını kapsamlı olarak kullanmışlardır.

Crauwels vd. (1997), yukarıda sözü geçen üç algoritmayı karşılaştırmışlardır. Bu algoritma 70' e kadar olan iş sayısının bulunduğu örneklerde başarılı olmuş ve Mason ve Anderson' un önerdiği algoritmaya üstünlük sağlamıştır.

$1/s_f/\sum w_jC_j$ problemi için yerel arama sezgiseli geliştiren iki çalışma bulunmaktadır. Mason (1991), her ailedeki ilk işi bilmenin, yığınları genelleştirilmiş SWPT kuralı kullanarak sıralanmasıyla oluşturulmuş bir çözümü mümkün kılar bilgisinden hareketle, bir genetik algoritma tasarlamıştır. Çalışmada standart genetik operatörlerin uygulandığı çözümlerin 0-1 tamsayılı ifadesini kullanmıştır.

Crauwels et al. (1997), birçok komşuluk arama sezgiselleri geliştirmişlerdir. Bir ailenin başında (bitişinde) bulunan işlerden bir alt aile seçen ve bu aileyi sıralamada daha ön pozisyonlara taşıyan bir komşuluk arama kullanmışlardır. Tavlama benzetimindeki sıcaklık ise periyodik bir desen takip etmektedir ve azalma/ alçalma *(descent)* algoritması her sıcaklık değişiminden önce uygulanmaktadır. 100 iş ve 20 aileye kadar olan örneklerin hesaplama sonuçları tüm yerel arama algoritmalarının en iyi sonuca çok yakın sonuçlara ulaştığını göstermiştir. En iyi sonuç aile sayısı az olduğunda çoklu başlangıçlı *(multi-start)* yasaklı arama algoritması ile, aile sayısı çok olduğunda ise Mason' ın önerdiği genetik algoritmayı yakaladığını göstermişlerdir.

Crauwels, et al.. (1996), $1/s_f/\sum U_j$ problemi için tavlama benzetimi, yasaklı arama ve genetik algoritma algoritmaları önermişlerdir. Komşuluk arama algoritmaları iş ya da aile komşuluklarından birini kullanarak çalışmaktadır. Komşuluk arama algoritmasında zamanında bitirilen işlerden birini alır, onun yerine geç kalan işlerden birini ekler. Böylece tüm işler zamanında tamamlanır. Aile komşuluk arama algoritması da benzer mantıkla çalışmaktadır. Ancak farklı olarak bir yığının tamamı geç kalmış işlerle yer değiştirir. Önerdikleri genetik algoritma ise her işi iki 0,1 tamsayılı elemanın temsil etmesi esasına dayanmaktadır. Bunlardan biri işin zamanında ya da geç tamamlandığını, diğeri ise o işin bir yığının son elemanı olup olmadığını temsil eder.

Zamanında tamamlanan işlerin çizelgesini elde etmek için, her zamanında tamamlanan iş ailesine ait bir teslim zamanı belirlenmiş ve aileler erken teslim kuralına göre sıralamışlardır. Problem çözümsüz olduğunda, yığın sayısını azaltmışlardır. Hesaplama sonuçları 50 iş, 10 aileye kadar olan örnekler için tüm sezgisellerin iyi çalıştığını göstermiştir. En kaliteli sonuçların ise genetik algoritma ile elde edildiği belirtmişlerdir.

Sotskov, et al. (1996), $F/s_f/C_{max}$ ve $F/s_f/C_j$ permutasyon tipi çizelgele ile sınırlandırılmış problemleri için komşuluk arama algoritmaları (tavlama benzetimi, yasaklı arama) geliştirmişlerdir. Komşuluk arama algoritması olarak Crauwels, et al.. (1997)'nin çalışmasındaki komşuluk arama algoritmasını kullanmışlardır. Hesaplama testleri, 80 işe kadar, 5 ve 10 makineli örneklerde en iyi sonuçların tavlama benzetimi ve yasaklı arama algoritması tarafından elde edildiğini göstermişlerdir.

BÖLÜM 3

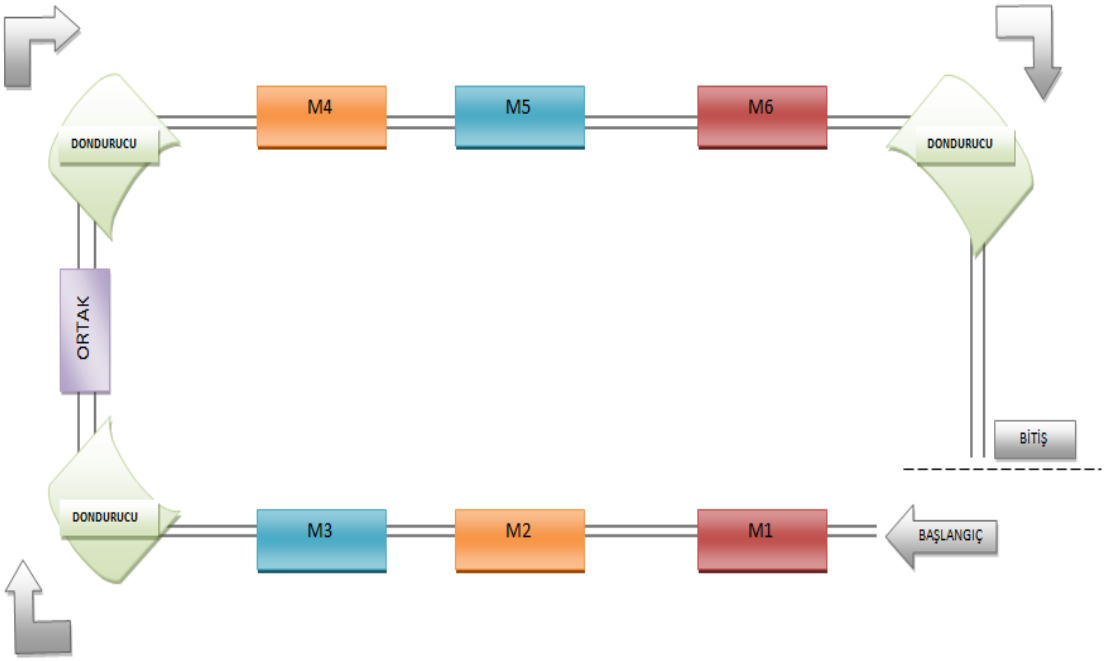
3. TEK YÖNLÜ DAİRESEL BİR ÜRETİM HATTINDA AİLE AYAR SÜRESİ YAKLIŞIMI İLE ÇİZELGELEME PROBLEMİ

3.1 Problemin Tanımlanması

Bu çalışmada tek yönlü dairesel bir üretim hattı ele alınmıştır (Şekil 3.1). Bu üretim hattı aynı anda en fazla üç ürünün üretimine olanak tanıyacak biçimde tasarlanmıştır. Bu noktadan hareketle çalışmanın yapıldığı çikolata işletmesinde, bu kısıta bağlı olarak hangi ürünlerin birlikte üretilebileceği üzerinde çalışılmış ve 105 adet ürün ailesi belirlenmiştir. Bu ürün grupları en fazla 3 farklı çeşitte üründen oluşmaktadır. Bu ürün grupları içinde yer alan ürünler, üretim hattında aynı anda işlenebildikleri için, arka arkaya işlendiklerinde herhangi bir ayar süresine ihtiyaç duyulmamaktadır. Ancak farklı ürün gruplarının arasında birinin işlenmesinden diğerine geçişlerde, sıra bağımlı olmak üzere, ayar süresine ihtiyaç duyulmaktadır. Bu nedenle belirlenen bu ürün grupları arasında farklı ürün gruplarının üretimine geçişlerde oluşan ayar süreleri de hesaplanmıştır.

Mevcut işleyişte ürünlere ilişkin talep bilgileri bölge temsilciliklerinden elde edilmektedir. Talep bilgilerinin elde edilmesiyle oluşturulmuş olan ürün gruplarından, talepleri karşılayacak şekilde, hangi ürün grubunun üretim için seçileceği ve seçilen ürün gruplarının hangi sırada üretileceği kararı verilmektedir. Mevcut işleyişte bu seçim ve çizelgeleme işlemi karar vericinin sezgilerine dayanılarak yapılmaktadır.

Bu işleyiş ilk bakışta sakıncalı görünmese de, üretim sürecinin tamamlanmasının ardından, talebi olmayan ürünlerin üretilmesi ve/ya talep edilen miktardan fazla üretilmesi sebebiyle gereksiz stok oluşumu problemi ile karşılaşılmaktadır. Ayrıca sezgisel olarak yapılan bu seçimde çizelgeleme aşamasında, ürün grupları arasında oluşan ayar süreleri de göz önüne alınmamaktadır.



Şekil 3.1 Tek yönlü dairesel üretim hattı şeması

Bu işleyiş ilk bakışta sakıncalı görünmese de, üretim sürecinin tamamlanmasının ardından, talebi olmayan ürünlerin üretilmesi ve/ya talep edilen miktardan fazla üretilmesi sebebiyle gereksiz stok oluşumu problemi ile karşılaşılmaktadır. Ayrıca sezgisel olarak yapılan bu seçimde çizelgeleme aşamasında, ürün grupları arasında oluşan ayar süreleri de göz önüne alınmamaktadır.

Bu çalışmada mevcut işleyişteki sözü edilen problem iki aşamalı olarak ele alınmıştır. İlk aşamada ürün aileleri içinden taleplere uygun seçim işlemi, ikinci aşamada ise seçilen ürün gruplarının çizelgelenmesi problemi ele alınmıştır.

3.2 Ürün Aileleri Seçimi Problemi ve Geliştirilen Matematiksel Model

Ürün aileleri seçimi problemi işletme tarafından önceden tanımlanmış 105 adet ürün ailesi arasından, ürün taleplerini karşılayacak şekilde seçim yapılmasını kapsar. Problemin bu aşaması ulaştırma modeline benzetilmiş ve bir matematiksel model geliştirilmiştir.

Ulaştırma modeli ürünlerin kaynaklardan (fabrika vb.) hedeflere (depo vb.) taşınmasıyla ilgilenir. Burada amaç bir taraftan hedefin talep gereksinimleri ve kaynakların arz miktarlarında denge sağlarken, diğer taraftan da her bir kaynaktan her bir hedefe yapılacak taşımaların toplam maliyetini enküçükleyecek taşıma miktarını belirlemektir (Taha, 2000).

Geliştirilen bu matematiksel modelde toplam üretim miktarını kaynaklardan hedeflere yapılacak taşıma miktarına benzetilmiştir. Başka bir ifade ile üretim programının oluşturulmasıyla talep bilgileri hedefleri, ürün ailelerinden seçim sonucunda elde edilecek miktar ise kaynak olarak düşünülmüş ve bu doğrultuda model geliştirilmiştir. Buna göre geliştirilen modelde kullanılan gösterimler şu şekildedir:

i ürün indisi

j ürün aileleri indisi

n toplam ürün sayısı

m toplam aile sayısı

Parametreler;

D_i i. ürünün talep miktarı

P_{ij} j. ailede bulunan i. üründen üretilen miktar

A_{ij} j. ailede yer alan i. ürün miktarı

K_i Üretim hattının i. Ürüne ilişkin kapasite değeri

C_i i. ürünü planlama devresi boyunca stokta bulundurma maliyeti (TL/br-devre)

Karar değişkeni;

$\boldsymbol{x_j}$ = j. ailenin tekrarlanma sayısı

Model I:

$$\sum_{j=1}^{m} A_{ij} * \boldsymbol{x_j} = P_{ij} \qquad i= 1,...,n \qquad (1)$$

$$\sum_{j=1}^{m} P_{ij} \geq D_i \qquad i= 1,...,n \qquad (2)$$

$$\sum_{j=1}^{m} P_{ij} \leq K_i \qquad i= 1,...,n \qquad (3)$$

$$P_{ij} \geq 0 \qquad i=1,...,n \qquad j=1,...,m \qquad (4)$$

$$X_j \geq 0, \text{ tamsayı} \qquad i=1,...,n \qquad j=1,...,m \qquad (5)$$

kısıtları altında

$$enk\ Z= \sum_{j=1}^{m} C_j *(\sum_{i=1}^{n} P_i - D_i) \qquad (6)$$

Problem, talebi karşılamak için yapılacak seçim sonucunda bir iş ailesinin üretiminin tekrarlanmasına olanak sağlamaktadır. Bu nedenle karar değişkeni x_{ij}, üretim için seçilen iş ailesinin üretiminin kaç kez tekrarlanacağını ifade etmektedir.

Model ürünlere ilişkin talep miktarının, üretim kapasitesini aşamayacağı varsayımı altında kurulmuştur. Modelde (1) numaralı kısıt seçim sonucunda elde edilen değerin ilgili ürüne ait toplam üretim miktarını tariflendiğini belirtir. Bir üründen üretilen miktar, ürünün yer aldığı ailelerdeki miktar ile ilgili ailelerin tekrarlanma sayılarının çarpımına eşittir. (2) numaralı kısıt ürünlerin taleplerinin mutlaka karşılanması gerektiğini tariflemektedir. (3) numaralı kısıt ürünlerin üretim miktarlarının hattın üretim kapasitesini aşamayacağını belirtmektedir. (4) numaralı kısıt bir üründen üretilecek miktarın sıfıra eşit veya sıfırdan büyük tamsayı değerler

alabileceğini ifade etmektedir. (5) numaralı kısıt ise karar değişkeninin sıfıra eşit veya sıfırdan büyük tamsayılı değerler alabileceğini ifade etmektedir. (6) numaralı ifade modelin amaç fonksiyonunu oluşturmaktadır. Amaç fonksiyonu seçim sonucunda oluşacak stok aktarım maliyetini enküçüklemek üzerine kurulmuştur.

3.3 Seçilen Ürün Gruplarının Çizelgelenmesi Problemi

Ürünlerin taleplerini karşılayacak şekilde ürün aileleri içinden seçim yapılması probleminin ardından seçilen ürün ailelerinin çizelgelenmesi aşaması gelmektedir. Başka bir ifade ile ürün aileleri seçim probleminin çözüm çıktıları ürün ailelerinin çizelgelenmesi probleminin girdilerini oluşturmaktadır.

3.3.1 Problemin sınıfı ve özellikleri

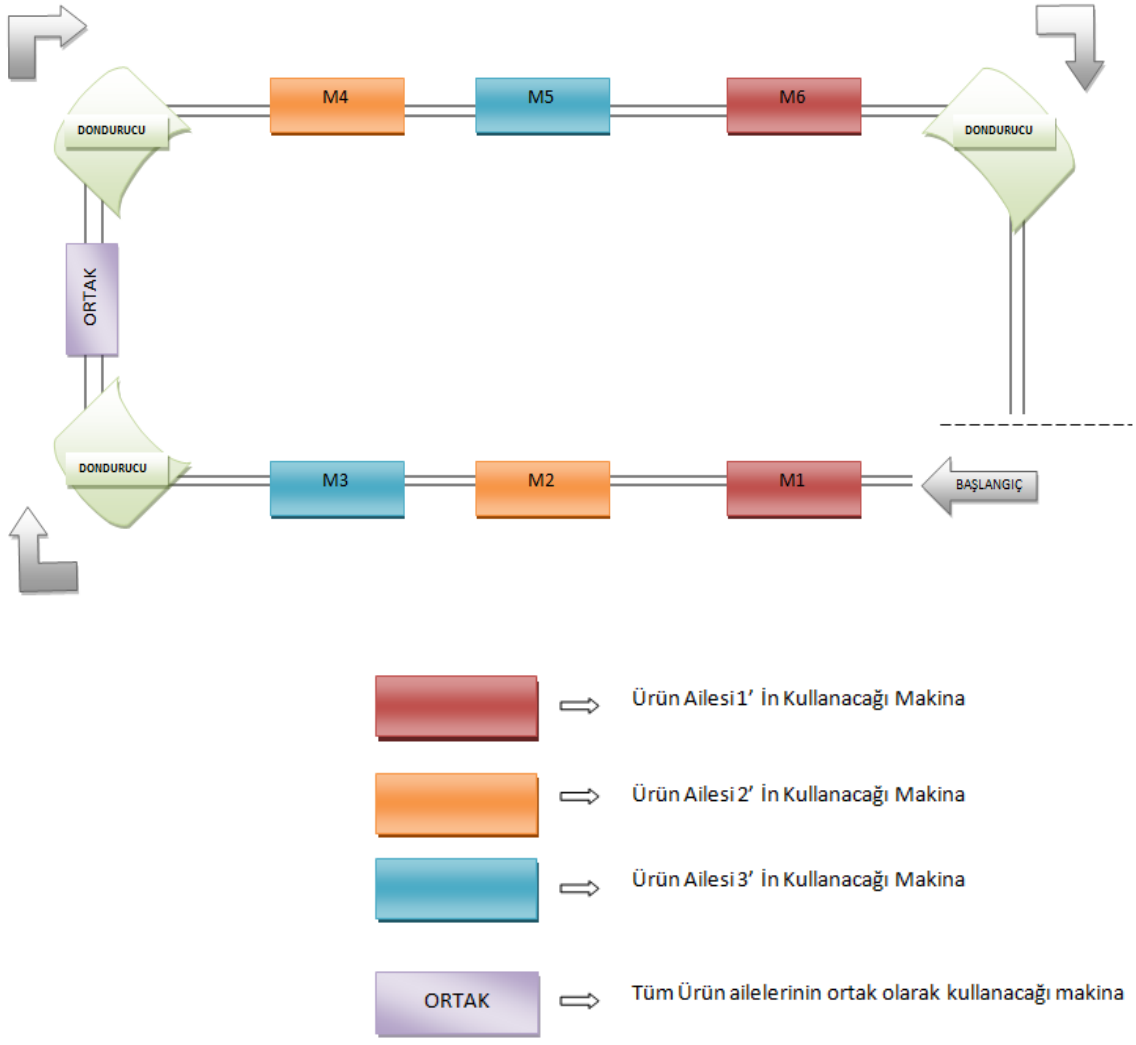
Çalışmaya söz konusu olan üretim hattı tek yönlü ve dairesel bir üretim hattıdır. Üretimine karar verilmiş ürün ailelerine ait kalıplar hatta verildikten sonra kalıplar hatta veriliş sıralarında üretimlerini tamamlamaktadırlar. Hattın dairesel olmasının verdiği bu özellik nedeniyle problem permutasyon akış tipi özelliği göstermektedir. Permutasyon akış tipinde makineler arasında sıra değişikliklerine izin verilmez, işlerin sırası baştan sona aynı kalır (Pinedo, 2000).

Diğer taraftan üretim hattının tek ve dairesel olması sebebiyle kalıpların hattı aynı sırada dolaşıyor olması ilk bakışta akla atölye tipi üretimi getirse de ürünlerin çeşitlerine bağlı olarak farklı makinelerde işlem görmeleri akış tipi üretim özelliğini göstermektedir.

Üretim hattında birden fazla makine vardır ve her çeşidin kullandığı makine sayısı birbirinden farklıdır. Ancak ayar süreleri ilgili ürün ailesinin temelinde belirlenmiştir.

Ürün aileleri içinde yer alan ürünler hatta aynı anda bulunmaları ve farklı makineleri kullanmaları sebebiyle herhangi bir ayar süresi söz konusu olmamaktadır. Ancak farklı ürün ailelerinin ardarda işlenmeleri durumunda üretim hattının ayarlanması zaman alacaktır ve bu ayar süresi sıra bağımlı ayar süresi özelliği göstermektedir. Başka

bir deyişle ayar süreleri üçgen eşitsizliğine uymaktadır. Ürün aileleri *f*, *g*, ve *h* için, s_{ij}, *i* ailesi ile *j* ailesi arasındaki ayar süresini tariflerken, $s_{fg} + s_{gh} \geq s_{fh}$ kuralı sağlanmaktadır (Gosh ve Gupta, 1997).



Şekil 3.2 Üretim hattı şeması üzerinde ürün ailelerinin rotalarının gösterimi.

3.3.2 Varsayımlar

Aile ayar süresi yaklaşımı ile çizelgeleme problemi ele alınırken, işletmedeki koşullar da gözlemlenerek problemin genel çatısını belirlemek adına bazı varsayımlar yapılmıştır. Bu varsayımlar aşağıda sıralanmıştır:

1. İş sayısı, işlem süreleri, iş ailesi sayısı, ayar süreleri bilinmektedir ve bu değerler negatif olmayan tamsayıdır.
2. Her işin hazır olma zamanı sıfırdır. Yani, listedeki her iş her an işlenmeye hazır durumdadır.
3. Üretim hattı başlangıçta olduğu varsayılan bir aile için (0 işi için) ayar süresi yapılmış durumdadır.
4. Aile içindeki işler arka arkaya işlenmek zorundadır ve bir ailenin işlenmesi o ailedeki tüm işlerin işlenmesi bitene kadar devam etmektedir.
5. İşler aynı anda sadece bir makinede gerçekleştirilebilir ve, işlem süresi tamamlanana kadar bölünmeden işlenir.
6. Her makine aynı anda yalnızca bir iş üzerinde işlem yapabilir.
7. Aile içindeki işler kendi aralarında erken teslim kuralına göre sıralanmıştır.
8. Bir iş ailesinin işlem süresi, o aile içinde yer alan tüm işlerin işlem sürelerinin toplamıdır.

3.3.3 Ürün ailelerinin çizelgelenmesi için geliştirilen matematiksel model

Ürün ailelerinin çizelgelenmesi probleminde üretimine karar verilen aileler, çizelgenin toplam tamamlanma zamanı enküçüklenecek şekilde sıralanacaklardır. Başka bir ifade ile çizelgeleme probleminde hangi ailenin hangi pozisyona atanacağının kararı verilecektir. Bu probleme ilişkin oluşturulan matematiksel modele ait parametreler ve karar değişkenleri şu şekilde belirlenmiştir:

i iş aileleri indisi $f \in i= \{1,2,...,n\}$

j işlerin atanacağı pozisyon indisi $k \in j=\{1,2,...,n\}$

p_i i. iş ailesinin işlem süresi

s_{if} İş ailesi i ile iş ailesi f arasında oluşan ayar süresi

Karar değişkenleri:

$$X_{ij} = \begin{cases} 1, i. \text{ ailenin ait } j. \text{ pozisyona atanması durumu} \\ 0, \text{d.d.} \end{cases}$$

$$Y_{ijfk} = \begin{cases} 1, j. \text{ pozisyona } i \text{ ailesi, } k. \text{ pozisyona } f \text{ ailesinin atanması durumu} \\ 0, \text{d.d.} \end{cases}$$

Model II:

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,.., m \qquad (1)$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad i = 1,.., n \qquad (2)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} Y_{ijfk} \geq x_{ij} + x_{fk} - 1 \qquad f \in i=\{1,..,n\},\ i \neq f\ ;$$

$$k \in j=\{1,..,m\},\ j+1=k \qquad (3)$$

$$X_{ij} \in \{0, 1\} \qquad (4)$$

kısıtları altında

$$enk\ Z = \sum_{i=1}^{n} \sum_{j=1}^{m} Y_{ijfk} * (s_{if} + p_i) \qquad (5)$$

Probleme konu olan ayar süresi sıra bağımlı özelliği sergilemektedir. Bu nedenle *i* ve *f*'' nin birbirinden farklı aileler olduğu varsayıldığında, bu iki aile arasında ayar süresinin (s_{if}) söz konusu olabilmesi için *f* ailesi, *i* ailesini takip eden pozisyona atanması gerekmektedir. Başka bir ifade ile *j* ve *k*' nın ardarda gelen pozisyonlar olduğu varsayıldığında ve *j*. pozisyona *i* iş ailesi $(x_{ij}=1)$, *k*. pozisyona da *f* ailesi atandığında $(x_{fk}=1)$ *i* ailesi ile *j* ailesi arasındaki ayar süresinden (s_{if}) bahsedilebilecektir.

Modelde bu kontrolü yapması ve amaç fonksiyonu karesel ifadeden kurtarmak için Y_{ijfk} şeklinde yeni ikil tamsayılı bir değişken tanımlanmıştır. Bu değişken, *k*. pozisyonun, *j*. pozisyonu takip ettiği varsayıldığında $(j+1=k)$, *j*. pozisyona *i* iş ailesi $(x_{ij}=1)$, *k*. pozisyona da *f* ailesi atandığında $(x_{fk}=1)$ ancak Y_{ijfk} değişkeni 1 değerini alabilecektir. Daha açık bir ifade ile birbirini izleyen pozisyonlara atanan iş ailelerinin aynı aileler mi yoksa farkı aileler mi olduğunu kontrol ederek buna göre ayar süresini hesaba dahil etmektedir.

Modelde (1) numaralı kısıtta her pozisyona yalnızca bir işin atanabileceği belirtilmektedir. (2) numaralı kısıtta her ailedeki işlerin yalnızca bir pozisyona atanabileceği ifade edilmektedir. (3) numaralı kısıt ise art arda gelen pozisyonlara $(j+1=k)$, farklı aileler atandığında $(i \neq f)$ bu iki aile arasında ayar süresi doğacağını belirtmektedir. (4) numaralı kısıt X_{ij} değişkeninin 0-1 tamsayılı değişken olduğunu belirtmektedir. (5) numaralı amaç fonksiyonu ise toplam tamamlanma zamanının enküçüklenmek istendiğini belirtmektedir. Tamamlanma zamanı ayar süreleri ile işlem sürelerinin toplamı olarak belirlenmiştir.

Şekil 3,3'te ürün aileleri sayısına göre kısıt sayıları ve kara değişkenlerinin sayıları bulunmaktadır. Şekil 3,3'ten de anlaşılacağı gibi problemin boyutu çizelgelenecek aile sayısının artmasıyla hızla artmaktadır. 60 aile ve üzerindeki problemlerde LINGO 8,0 programı bu sayıları hesaplamada yetersiz kalmaktadır.

Şekil 3.3 Problem boyutunun artmasıyla kısıt ve karar değişkeni sayılarındaki artışlar

BÖLÜM 4

4. İŞ AİLELERİ ÇİZELGELEME PROBLEMİNİN ÇÖZÜMÜNE İLİŞKİN GENETİK ALGORİTMA YAKLAŞIMI

4.1 Genetik Algoritmalar Hakkında Genel Bilgi

Evrimsel programlanın bir parçası olan genetik algoritmalar, Darwin' in evrim teorisinden esinlenilerek ortaya çıkmıştır. Genetik algoritmalar ilk olarak Michigan Üniversitesi'nden psikolog ve bilgisayar bilimleri uzmanı J. H. Holland tarafından 1975 yılında önerilmiştir. Genetik algoritmalar en iyinin korunumu ve doğal seçim ilkesinin benzetim yoluyla bilgisayarlara uygulanması ile elde edilen bir arama yöntemidir (Goldberg, 1989). Başka bir ifade ile genetik algoritmalar doğal seçim mekanizmasını taklit edip, en iyi çözüme ulaşmaya çalışan bir arama tekniğidir. Genetik algoritmaların temel teoremi uyum gücü yüksek olan kromozom parçalarının (schemata) hızla çoğalacağını ortaya koymaktadır (İşlier, 1997: Saraç' tan, 2007).

Genetik algoritmalar, geniş uygulama alanı, güçlü stokastik arama ve optimizasyon tekniği olarak, evrimsel algoritmaların en çok bilinen çeşidi olmuştur (Gen ve Cheng, 2000). Genetik algoritmalar, komşuluk arama metotlarının aksine, aynı anda birçok çözümle ilgilenir ve bu çözümlerin uyum fonksiyonu değerini hesaplarlar (Lin vd., 2009).

Genel olarak bir genetik algoritma, Michalewicz tarafından belirtildiği gibi, beş temel bileşenden oluşur (Gen ve Cheng, 2000):

1. Problemin çözümünün genetik gösterimi
2. Başlangıç popülasyon oluşturmak için bir yöntem

3. Uyum değeri ile ifade edilen, çözümlerin derecelendirilebileceği bir hesaplama fonksiyonu

4. Reprodüksiyon ile yeni bireyler oluşturacak genetik operatörler

5. Genetik algoritma parametrelerinin değerleri

Genetik algoritmalar kromozomların oluşturduğu popülasyonlar üzerinde çalışırlar. Kromozomlar genlerden oluşur. Genler üzerlerinde alel adı verilen değerleri taşırlar. Genetik biliminde genin konumunu lokus değeri belirtir ve bu değer genin fonksiyonundan farlı bir değerdir. Örneğin bir canlının mavi renkli göz geni ele alındığında, bu genin lokus değeri, yani kromozom içindeki konumu, 10 olabilir. Bu genin alel değeri ise mavi renk olması olacaktır. Genetik algoritmalar terminolojisinde kromozomlar farklı değerleri taşıyan özelliklerden ya da belirleyicilerden *(detectors)* oluşur. Her kromozom probleme ait potansiyel bir çözümü temsil eder. Ayrıca her kromozomun bir değeri vardır ve bu değere uyum değeri adı verilir (Gen ve Cheng, 2000).

Kromozomlar uyum değerlerine göre, genetik operatörler tarafından yeni bireyler oluşturmak için stokastik değişikliklere uğrarlar. İki tip değişiklik söz konusudur: çaprazlama ve mutasyon. Çaprazlama iki kromozomun parçalarını birleştirerek yeni bireyler oluşturur, mutasyon ise tek bir kromozom üzerinde değişiklikler yaratır (Gen ve Cheng, 2000). Çaprazlama ve mutasyon sonucunda oluşturulan yeni bireylerin uyum değerleri hesaplanır ve en iyi uyum değerine sahip kromozomlarla yeni bir populasyon oluşturulur. Bu işlemlerin tamamlanmasıyla bir ardıştırma *(iteration)* tamamlanmış olur. Karar vericilerin belirlediği ardıştırma sayısı tamamlanana kadar bu işlemler tekrarlanır ve ardıştırma sayısı tamamlandığında algoritma en iyi ya da en iyi çözüme yakın çözümü veren en iyi kromozomu sunar.

Arama stratejisinde iki önemli konu vardır: en iyi sonuca ulaşma (exploiting) ve arama uzayını keşfetme (exploring). Genetik algoritmalar, karmaşık bir alanda yönlendirilmiş (directed) rassal arama yaparlar. Genetik operatörler aslında kör arama (blind search) yaparlar: seçim operatörleri, genetik aramayı, çözüm uzayının bir alanına doğru yönlendirirler. Bu özellik nedeniyle bir gerçek hayat problemini tarifleyen bir

genetik algoritma geliştirmenin genel prensibi en iyiye ulaşma ve arama uzayını keşfetme arasında iyi bir denge kurmaktır. Bunu yapabilmenin yolu da genetik algoritmanın bileşenlerinin dikkatle incelemekten geçmektedir (Gen ve Cheng, 2000).

Genetik algoritmaların temel adımları kromozom gösterimi, seçim prosedürü, çaprazlama ve mutasyon çeşidinin belirlenmesi ile uyum fonksiyonun geliştirilmesinden oluşur.

4.1.1 Kromozom gösterimi

Genetik algoritmalar iki tip uzay üzerinde çalışırlar: genotip ve fenotip uzayı. Genetik operatörler genotip uzayı üzerinde çalışırlar, hesaplama ve seçim işlemleri ise fenotip uzayında çalışır. Bir genetik algoritma oluşturulmasında kilit nokta problemin çözümünün kromozom tarafından nasıl şifreleneceğinin belirlenmesidir. Başka bir ifade ile karakterlerin genotip uzayından fenotip uzayına nasıl aktarılacağının belirlenmesi önemlidir.

Bir genin alel değerleri için kullanılabilecek şifreleme metotları şöyle sıralanabilir:

- 0-1 gösterimi
- Reel sayı gösterimi
- Tamsayı ya da alfabetik dizilim gösterimi
- Genel veri yapısı gösterimi

Endüstri mühendisliği dünyasındaki birçok problem için 0-1 gösteriminin kullanılması neredeyse imkânsızdır. Reel sayı gösterimi optimizasyon problemleri için kullanılabilecek en iyi yöntemdir. Bu konuda yapılan birçok çalışmada bu gösterim şeklinin, 0-1 gösteriminden daha iyi sonuçlar verdiğini ortaya koymuştur. Reel sayı gösteriminde genotip uzayının topolojik yapısı ile fenotip uzay yapısının aynı olması nedeniyle, etkin genetik operatörler oluşturmak kolaydır. Tamsayı ya da alfabetik dizilim gösterimi ise kombinatoryal optimizasyon problemleri için idealdir. Daha karmaşık yapılı gerçek hayat problemleri için ise genel veri yapısı gösteriminin kullanımı tavsiye edilmektedir (Gen ve Cheng, 2000).

4.1.2 Seçim prosedürü

Genetik algoritmaların temel prensibi Darwin' in doğal seçim teorisine dayanmaktadır. Seçim prosedürü genetik algoritmaya yol aldıran kuvvettir. Bu kuvvetin gücünün artması genetik aramanın daha erken sonlanmasına; bu kuvvetinin gücünün azalması ise evrimsel ilerlemenin ihtiyaç olandan daha yavaş olmasına neden olacaktır. Seçim baskısının şiddetinin artması ise arama uzayının daraltılması için gereklidir. Seçim, genetik aramayı çözüm uzayının umut vadeden kısımlarına yönlendirir.

Yaygın olarak kullanılan seçim prosedürü çeşitleri şöyledir:

- Rulet tekeri seçimi
- $(\mu + \beta)$ seçimi
- Turnuva seçimi
- Kararlı durum reprodüksiyonu (steady-state reproduction)
- Sıralama ve ölçeklendirme
- Paylaşım

Holland tarafından önerilen rulet tekeri seçim prosedürü, her kromozomu uyum değeri ile oranlayarak seçim olasılığı ve hayatta kalma olasılığı belirlemeye dayanır. $(\mu+\beta)$ seçimi ise ebeveynlerden ve yeni bireylerden en iyi kromozomları seçen deterministik bir prosedürdür. Bu iki metot kombinatoryal optimizasyon problemleri için idealdir (Gen ve Cheng, 2000).

Turnuva seçimi ise Goldberg vd. (1989) tarafından önerilen, rassal ve deterministik özellikleri barındıran seçim prosedürüdür. Bu metotta kromozom kümesi rassal olarak seçilir ve bu kümeden en iyi uyum değerine sahip kromozom reprodüksiyon için seçilir. Rassal olarak seçilen kromozom kümesi büyüklüğü turnuva

boyutunu belirler. Yaygın olarak kullanılan turnuva büyüklüğü 2' dir ve bu boyut 0-1 turnuva olarak adlandırılır (Gen ve Cheng, 2000).

Karalı durum reprodüksiyonu seçim prosedürü ise yeni popülasyon oluşturulurken ebeveynlerin yeni bireylerle yer değiştirmesi esasına dayanır.

Sıralama ve ölçeklendirme seçim prosedüründe, sıralama mekanizması amaç fonksiyonu değerlerini pozitif reel değerlere dönüştürerek, bu kromozomların hayatta kalma olasılıklarını hesaplar. Sıralama mekanizması ise amaç fonksiyonu değerini ihmal eder ve hayatta kalma olasılıkları için kromozomların sıralamasını kullanır. Popülasyonu en iyi kromozomdan başlayarak en kötüye doğru sıralar.

Paylaşım tekniği, Goldberg ve Richardson tarafından, çoklu fonksiyon optimizasyon problemleri için önerilmiştir (Gen ve Cheng, 2000).

4.1.3 Çaprazlama Çeşitleri

Çaprazlama, ebeveyn kromozomlardan seçilen genleri yeni bireye aktarır. Bu işlem rassal olarak bir çaprazlama noktası/ noktaları seçer ve bu noktadan itibaren seçilen gen parçalarının yer değiştirmesiyle yeni bireyi oluşturur. Kromozom gösterimlerine bağlı olarak çaprazlama çeşitleri değişiklik gösterir.

İkili kodlanmış (0-1) ve reel sayılı gösterim kullanan kromozomlarda tek noktalı, çift noktalı, tek biçimli (uniform) ve aritmetik çaprazlama en sık uygulanan çaprazlama çeşitleridir.

Tek noktalı çaprazlamada kromozom üzerinde bir çaprazlama noktası seçilir ve bu noktaya kadar olan genler birinci ebeveynden, bu noktadan sonraki genler ise diğer ebeveynden alınarak yeni birey oluşturulur.

Çift noktalı çaprazlamada ise iki kırılma noktası seçilir. İlk noktaya kadar olan genler birinci ebeveynden, iki nokta arasında kalan genler ikinci ebeveynden, ikinci noktadan sonra kalan genler ise tekrar birinci ebeveynden yeni bireye aktarılır.

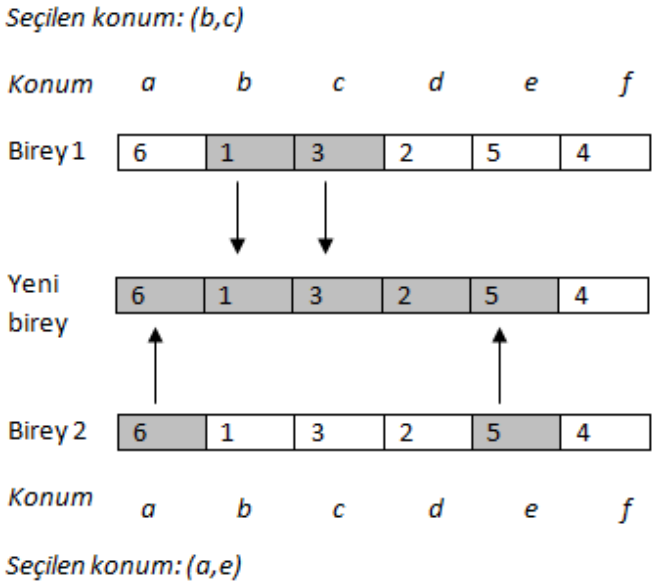
Tek biçimli çaprazlamada ise seçilen genler rassal olarak her iki ebeveynden yeni bireye aktarılır. Aritmetik çaprazlamada ise değişik aritmetik işlemler kullanılır (Nabiyev, 2005).

Dizilem gösteriminde sık kullanılan çaprazlama operatörleri parçalı eşleştirme çaprazlaması *(partially mapped crossover-PMX*), sıra çaprazlaması, konum tabanlı çaprazlamadır (Gen ve Cheng, 2000).

PMX çaprazlamada kromozom üzerinde iki kesim noktası belirlenir. Belirlenen bu iki nokta arasında kalan gen dizilemine eşleştirme kısmı denir. Belirlenen bu iki eşleştirme kısmı, yeni bireyleri oluşturmak için, karşılıklı yer değiştirir. Bu işlem sonucunda tekrarlı genleri engellemek için iki ebeveyn kromozomda tekrar eden genler belirlenir ve tekrar eden genlerin karşılıklı olarak yer değiştirir (Gen ve Cheng, 2000).

Sıra çaprazlamasında bir ebeveynden rassal olarak bir gen dizilemi seçilir. Belirlenen bu gen dizilemi yeni bireydeki aynı pozisyonlara aktarılır. İkinci ebeveynden seçilen gen dizilemindeki genler silinir. Bu işlemden sonra ikinci ebeveynde kalan genler yeni bireyin boş kalan pozisyonlarına baştan başlanarak aktarılır.

Konum tabanlı çaprazlamada ise bir ebeveynden rassal olarak pozisyonlar kümesi oluşturulur. Yeni bireye seçilen bu pozisyonlardaki genler yine aynı pozisyonlara atanır. İkinci ebeveynden, birinci ebeveyndeki seçilen pozisyondaki genler silinir ve kalan genler yeni bireye soldan sağa doğru aktarılır (Şekil 4.1).



Şekil 4.1 Konum tabanlı çaprazlama örneği

4.1.4 Mutasyon

İkili (0-1) gösterimi kullanılan kromozomlarda mutasyon işleminde rassal olarak seçilen gen0' dan 1' e ya da 1' den 0' a değiştirilir.

Dizilem gösteriminde ise ters çevirme, ekleme, yer değiştirme, karşılıklı yer değiştirme ve taşıma mutasyon operatörleri en sık kullanılanlardır.

Ters çevirme operatöründe bir kromozomda rassal olarak iki pozisyon seçilir ve bu pozisyonlar arasında kalan gen dizilemi ters çevrilir.

Ekleme operatöründe bir gen rassal olarak seçilir ve yine rassal olarak belirlenen bir pozisyona eklenir.

Yer değiştirme operatöründe ise rassal olarak bir gen dizilemi seçilir ve yine rassal olarak seçilen bir pozisyona eklenir.

Karşılıklı yer değiştirme operatöründe ise rassal olarak iki pozisyon seçilir ve bu pozisyonlardaki genler karşılıklı yer değiştirir.

Taşıma operatöründe bir gen yine rassal olarak seçilir ve istenen bir pozisyona taşınır.

4.1.5 Genetik algoritma parametreleri

Genetik algoritmaların performansları çaprazlama olasılığı, mutasyon olasılığı, popülasyon büyüklüğü ve ardıştırma sayısı tarafından etkilenmektedir.

- Çaprazlama olasılığı (P_c), çaprazlamanın sıklığını belirler. Çaprazlama olasılığının sıfır olması popülasyonun aynı kalması anlamına gelmektedir. Diğer yandan bu olasılık 1 olduğunda popülasyondaki tüm bireylerin çaprazlama yoluyla elde edilecektir. Bu durumda da iyi olan bilgilerin kaybolması riski gündeme gelecektir.

- Mutasyon olasılığı (P_m), mutasyonun sıklığını belirler. Mutasyon oranının düşük olması arama uzayının tamamının araştırmasını engelleme riski taşır. Bu oranın artması ise probleme rassallık özelliği kazandırır ve bireylerin değişmesini sağlar. Ancak bu oranın 1 olması da tüm bireyleri değişikliğe uğratacak ve faydalı bilgilerinde yok olmasına neden olabilir.

- Popülasyon büyüklüğü, popülasyonda kaç birey olduğunu belirtir. Popülasyondaki birey sayısı az ise çözüm uzayı yeterince araştırılamayacağı için yerel optimumum noktaya takılma olasılığı artar. Bu sayı çok arttığında ise neslin evrimleşmesi uzun süreceğinden algoritma da yavaşlayacaktır.

4.1.6 Ürün Aileleri Çizelgeleme Problemine İlişkin Geliştirilen Genetik Algoritma

Ürün aileleri çizelgeleme problemi için kromozom gösterimi, başlangıç popülasyonu, seçim prosedürü, çaprazlama ve mutasyon operatörleri ile durma koşulu aşağıdaki gibi tariflenmiştir:

- *Kromozom Gösterimi Ve Başlangıç Popülasyonu*: Kromozom gösterimi için tamsayılı gösterim kullanılmıştır ve iş ailelerinin sayıları kromozom

uzunluğunu belirlemektedir. Başlangıç popülasyonu rassal olarak üretilmiştir ve 10 bireyden oluşmaktadır.

- *Seçim Kriteri*: Bu çalışmada turnuva seçimi kullanılmıştır ve turnuva büyüklüğü 2 olarak belirlenmiştir.

- *Çaprazlama*: Çaprazlanmaya girme olasılığı (Pc) 0.40 olarak belirlenmiştir ve konum tabanlı çaprazlama kullanılmıştır.

- *Mutasyon*: Karşılıklı yer değiştirme operatörü kullanılmıştır, mutasyon olasılığı 0.01 olarak belirlenmiştir.

Önerilen genetik algoritma C# dilinde, ayar süreleri ile işlem sürelerini bir Excell dosyasından okuyacak şekilde tasarlanmıştır. Program kullanıcıdan ardıştırma sayısının girilmesini istemektedir. Bu ardıştırma sayısına ulaşıldığında ise o ana kadar bulunan çözümler içinden bulunan en iyi çözümü, yani kromozomu ve uyum değerini, yani toplam tamamlanma zamanını göstermektedir.

4.2 Geliştirilen Genetik Algoritmanın Performansı

Dairesel bir üretim hattında sıra bağımlı aile ayar süreleri yaklaşımı ile çizelgeleme problemi için geliştirilen matematiksel model ve genetik algoritma performansları çeşitli problem büyüklükleri için karşılaştırılmıştır. Her iki program da aynı veri kümesini kullanmıştır ve bu veri kümesi Excell'de rassal sayı türetme fonksiyonu yardımı ile oluşturulmuştur. Test problemlerinde problem büyüklüğü için üst sınır olarak 105 aile seçilmesinin nedeni bu çalışma kapsamında ele alınan problemde en fazla 105 adet ürün ailesinin bulunuyor olmasıdır.

LINGO 8.0'da geliştirilen matematiksel modelin ve önerilen genetik algoritmanın performansları farklı boyuttaki problemler için karşılaştırılmış ve sonuçlar Çizelge 4.1'de gösterilmiştir.

Çizelge 4.1 Test problemleri üzerinde önerilen genetik algoritma ile LINGO 8.0' da geliştirilen matematiksel modelin performansları

PROBLEM BOYUTU	LİNGO 8.0	GA
5	178	178
10	378	343
20	862	1052
30	-	1559
40	-	2258
50	-	3370
60	-	4123
70	-	4935
80	-	6874
90	-	7060
105	-	8906

Çizelge 4.1 incelendiğinde ele alınan test problemleri için önerilen genetik algoritmanın 5, 10 ve 20 iş ailesi için 10000 ardıştırma çalıştırılmıştır ve LINGO'da elde edilen bütünsel en iyi değeri yakaladığı görülmüştür. 30 ve üzeri ürün ailesinin çizelgelenmesi söz konusu olduğunda LINGO'nun problem boyutu nedeniyle çalışamadığı ve dolayısıyla uygun bir çözüm bulamadığı, buna karşılık önerilen genetik algoritmanın uygun çözümlere ulaşabildiği gözlenmektedir.

BÖLÜM 5

5. SONUÇ VE ÖNERİLER

Bu çalışma bir gıda işletmesinde bulunan tek yönlü dairesel bir üretim hattında yaşanan sıra bağımlı aile ayar süreleri yaklaşımı ile bir çizelgeleme problemi üzerine kurulmuştur. Bu probleme ek olarak, işletmede var olan dairesel üretim hattının yapısı itibariyle hatta aynı anda işlenebilen ürünlerin belirlenip, gruplanmasıyla oluşturulmuş 105 farklı ürün ailesi içinden talepleri karşılayacak şekilde seçim yapılması problemi de ele alınmıştır. Bu alt problem, ürünlerin kaynaklardan (fabrika vb.) hedeflere (depo vb.) taşınmasıyla ilgilenen, bir taraftan hedefin talep gereksinimleri ile kaynakların arz miktarları arasında denge sağlarken, diğer taraftan da her bir kaynaktan her bir hedefe yapılacak taşımaların toplam maliyetini enküçükleyecek taşıma miktarını belirlemenin amaçlandığı ulaştırma problemine benzetilerek modellenmiştir. Bu alt problemin çıktıları hangi ürün ailelerinin çizelgelenmesi gerektiği kararının verilmesini sağlamaktadır. Bu modelin çıktılarına dayanılarak çizelgeleme problemi yapılandırılmıştır.

Sıra bağımlı aile süreleri yaklaşımı ardarda işlendiklerinde aralarında ayar süresi gerektirmeyen işlerin bir aileyi oluşturduğu ve farklı ailelerin ardarda işlenmeleri durumu söz konusu olduğunda ise sıra bağımlı ayar süresine gerek olduğu durumlar için geliştirilmiştir. Üretimine karar verilen ürün ailelerinin çizelgelenmesi problemi de bu yaklaşım kullanılarak modellenmiş ve toplam tamamlanma zamanın enküçüklenmesi amaç edinilmiştir. Geliştirilen bu matematiksel model, LINGO 8.0 programında kodlanmıştır. Farklı ürün aileleri sayıları kullanılarak modelin performansı incelenmiştir. Bu inceleme sonucunda, aile sayılarının artmasıyla birlikte, değişken ve kısıt sayısının çok hızlı arttığı gözlenmiş ve programın 30 ve üzeri ailenin çizelgelenmesi durumu söz konusu olduğunda, çözüm bulamadığı sonucuna ulaşılmıştır. Bu nedenle modelin çözümüne ilişkin bir genetik algoritma geliştirilmiştir. Önerilen bu

genetik algoritmanın performansı incelenmiş ve LINGO modelinin performansı ile kıyaslanmıştır. Sonuç olarak problem boyutunun üst sınır değeri olan 105 aile sayısına ulaşıldığında da önerilen genetik algoritmanın uygun bir çözüm üretebildiği görülmüştür.

Bundan sonraki çalışmalarda işletmenin kullanabileceği, gerekli verileri veri tabanından okuyup, çizelgeyi oluşturacak bir yazılım geliştirilmesi konusu işlenebilir. Ayrıca genetik algoritmanın operatörlerinde ve parametrelerinde değişikliklere gidilerek, diğer arama algoritmaları ve/ya hibrit yaklaşımlar kullanılarak problem çözülmeye çalışılabilir ve performansları kıyaslanabilir.

KAYNAKLAR DİZİNİ

Baker, K.R. and Magazine, M., 2000,Minimizing maximum lateness with job families, European Journal of Operational Research, 127(1), 126-39.

Baker, K.R., 1998, Heuristic procedures for scheduling job families with setups and due dates,The Amos Tuck School of Business Administration, Darmouth College.

Bruno, J.L. and Downey, P. J., 1978, Complexity of task sequencing with deadlines, setup times and changeover costs, SIAM Journal on computing 7, 393-404.

Chen, B.,1993, A better heuristic for preemptive parallel machine scheduling with batch setup times, SIAM Journal on Computing 22, 1303-1318.

Crauwels, H.A.J., Hairiri, A.M.A., Potts, C.N. and Van Wassenhove, L.N., 1997, Branch and bound algorithms for single machine scheduling with batch setup times to minimize total weighted completion time, Annals of Operations Research.

Crauwels, H.A.J., Potts, C.N. and Van Wassenhove, L.N., 1997, Local search heuristics for single machine scheduling with batch setup times to minimize total weighted completion time, Annals of Operations Research 70, 200-213.

Crauwels, H.A.J.,Potts, C.N., ve Van Wassenhove, L.N., 1996, Local search heuristics for single machine scheduling with batching the number of late jobs, European Journal of Operational Research 90, 200-213.

Gen, M. And Cheng, R., 2000, Genetic Algorithms and Engineering Optimization, John wiley&Sons, p. 491.

Ghosh, J.B. and Gupta, J.N.D., 1997, Batch scheduling to minimize maximum lateness, Operations research letters21, 77-80.

KAYNAKLAR DİZİNİ (devam ediyor)

Ghosh, J.B.,1994, Batch scheduling to minimizetotal completion time, Operations research letters 16,2 71-275.

Goldberg, D.E., 1989, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, p.381.

Hariri, A.M.A. and Potts, C.N., 1997, Single machine scheduling with batch setup times to minimize maximum lateness, Annals of Operations Research 70, 75-92.

İşlier, A.A., 1995, İmalat Problemlerinde Genetik Algoritmalar, Otomosyon Dergisi, Ocak, 94-98.

Kleinau, U.,1993, Two-machine shop scheduling problems with batch processing, Mathematical and Computer Modelling 17, 55-66.

Lawler, E.L., 1973, Optimal sequencing of a single machine subject to precedence constraints, Management Science, 19(5),544-6.

Lin, S.W., Ying, K.C., Lee and Z.J., 2008, Metaheuristics for scheduling a non-permutation flowline manufacturing cell with sequence dependent family setup times,Computer

Mason, A.J. and Anderson, E.J., 1991, Minimizing flow time on a single machine with job classes and setup times, Naval Research Logistics 38 , 333-350.

McNaughton, R.,1959, Scheduling with deadlines and loss functions, mnagement Science 6, 1-12.

Monma, C.L. and Potts, C.N.,1989, On the complexity of scheduling with batch setup times, Operations Research 37, 798-804.

Monma, C.L.and Potts, C.N.,1993, Analysis of heuristics for preemptive parallel machine scheduling with batch setup times, Operations Research 4193, 981-993.

KAYNAKLAR DİZİNİ (devam ediyor)

Moore, J.M., 1968, An n job, one machine sequencing algorithm for minimizing the number of late jobs, Management Science 15, 102-109.

Nabiyev, V.V., 2005, Yapay Zeka: Problemler, Yöntemler, Algoritma, Seçkin Yayınları, 759 s.

Pinedo, M.,2002, Scheduling: Theory, Algorithms and Systems, Prentice Hall, 378p.

Potts, C.N. and Van Wassenhove, L.N., 1992, Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity, Journal of the Operational Research Society 43, 395-406.

Saraç, T, 2007, Genelleştirilmiş Karesel Çoklu Sırt Çantası Problemi İçin Melez Bir Çözüm Yaklaşımı, Doktora Tezi, Osmangazi Üniversitesi Fen Bilimleri Enstittüsü, 141s.

Schutten, J.M.J. ,van de Velde, S.L. ve Zijm, W.H.M.,1996, Single-machine scheduling with release dates, due dates and family setup times, Management Science 42, 1165-1174.

Sotskov, Y.N., Tautenhahn, T. and Werner, F., 1996, Heuristics for permutation flow shop scheduling with batch setup times, Or Spektrum 18, 67-80.

Taha, H. 2000, Yöneylem Araştırması, Literatür Yayıncılık, p. 905.

Uzsoy, R., Martin-Vega,L.A., Lee,C.Y., Leonard, P.A., 1991, Production scheduling algorithms for a semiconductor test facility, IEEE Transactions on Semiconductor Manufacturing, 4, 270-80.

Webster, S.T. ,1997, The complexity of scheduling job families about a common due date, Operations Research Letters 20, 65-74.

Webster, S.T. and Baker, K.R., 1995, Scheduling groups of jobs on a single machine, Operations Research 43, 692-703.

EKLER DİZİNİ

EK.1 Önerilen Genetik Algoritmanın Program Kodları

EK.2 Matemetiksel Modelin ve Geliştirilen Matematiksel Modelin Kullandığı Veri dosyasının Bir Bölümü

EK.3 Geliştirilen Genetik Algoritmanın Kullanıcı Arayüzü

Ek- 1 Önerilen Genetik Algoritmanın Program Kodları

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;

namespace UrunAile
{
    class ga
    {
        public kromozom eniyiKromozom;
        public double eniyiUyumDegeri;
        public int CalisanIterasyonSayisi;//Toplamda kaç iterasyon
çalıştırıldığını tutacak

        public int eniyiKromozomIterasyonIndeksi;//en iyi kromozomun
kaçıncı nesilde bulunduğunu tutacak

        public ga()
        {
            eniyiKromozom = new kromozom();
            CalisanIterasyonSayisi = 1;
        }


        public populasyon turnuvaSecim(int turnuvaBuyuklugu,
populasyon p)
        {
            kromozom[] kr = new kromozom[Program.popBuyukluk];
            ArrayList kromozomlar = new ArrayList();
            for (int i = 0; i < p.populasyonBuyuklugu; i++)
            {
                if (i % turnuvaBuyuklugu == 0)
                {
                    kromozomlar.Clear();
                    for (int j = 0; j < turnuvaBuyuklugu; j++)
                    {

                        kromozomlar.Add(p.kromozomlar[i+j]);
                    }
                    double eniyiDeger=99999999;
                    kromozom eniyiKrom=new kromozom();
                    for(int a=0;a<kromozomlar.Count;a++)
                    {
                        kromozom kr2=new kromozom();

                        kr2=(kromozom)kromozomlar[a];
                        if (eniyiDeger > kr2.uyumFonksiyonuHesapla())
                        {
                            eniyiDeger = kr2.uyumFonksiyonuHesapla();
                            eniyiKrom = kr2;
                        }
                    }
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
                    for (int c = 0; c < kromozomlar.Count; c++)
                    {
                        kr[i + c] = eniyiKrom;
                    }
                }
            }
            populasyon pop = new populasyon(kr);
            return pop;

        }
        kromozom ciftiCaprazlaKonumTabanli(kromozom[] cift, int
caprazlanacakGenSayisi)
        {
            kromozom kr1 = new kromozom();
            kr1 = cift[0];//1. kromozomun koyasını oluştur
            kromozom kr2 = new kromozom();
            kr2 = cift[1];//2. kromozomun kopyasını oluştur
            kromozom cocuk = new kromozom();
            ArrayList cocukGenler = new ArrayList(kr1.Genler.Count);
            ArrayList genDeger = new ArrayList();

            Random rnd;
            ArrayList caprazlamaNoktalari = new
ArrayList(caprazlanacakGenSayisi);
            for (int i = 1; i < kr1.Genler.Count + 1; i++)
            {
                genDeger.Add(i);
            }
            for (int i = 0; i < caprazlanacakGenSayisi;
i++)//çaprazlama noktalarını rassal olarak oluştur
            {
                rnd = new Random();
                int cprNoktasi = rnd.Next(0, genDeger.Count-1);
                cprNoktasi = (int)genDeger[cprNoktasi];
                caprazlamaNoktalari.Add(cprNoktasi);
                genDeger.Remove(cprNoktasi);
            }

            caprazlamaNoktalari.Sort();//küçükten büyüğe sırala

            ArrayList caprazlamaNoktalariGenDegerleri = new
ArrayList(caprazlamaNoktalari.Count);
            ArrayList eksikDegerler = new
ArrayList(caprazlamaNoktalari.Count);
            for (int i = 0; i < kr2.Genler.Count; i++)
            {
                gen g = new gen();
                g = (gen)kr2.Genler[i];
                int deger = (int)g.genDeger;
                eksikDegerler.Add(deger);
                cocukGenler.Add(0);
            }
            for (int i = 0; i < caprazlamaNoktalari.Count; i++)
{
                gen g = new gen();
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
          g = (gen)kr1.Genler[(int)caprazlamaNoktalari[i]];
                caprazlamaNoktalariGenDegerleri.Add(g.genDeger);
                eksikDegerler.Remove(g.genDeger);
            }
            int cprIndeks = 0;
            for (int i = 0; i < kr1.Genler.Count; i++)//sabit kalacak
pozisyonların genlerini kopyala
            {
                if (caprazlamaNoktalari.Contains(i))
                {
                    gen g = new gen();
                    g.genDeger =
caprazlamaNoktalariGenDegerleri[cprIndeks];
                   cocukGenler[i]=g;
                    cprIndeks++;
                }
            }
            int eksikIndeks = 0;
            for (int i = 0; i < kr1.Genler.Count; i++)
            {
                if (!caprazlamaNoktalari.Contains(i))
                {
                    gen g=new gen();
                    g.genDeger=eksikDegerler[eksikIndeks];
                    eksikIndeks++;
                    cocukGenler[i] = g;
                }
            }

            cocuk = new kromozom(cocukGenler);
            Console.Write("\nKonum Tabanlı Çaprazlama Sabit
Pozisyonlar:");
            foreach (int i in caprazlamaNoktalari)
            {
                int a = i + 1;
                Console.Write(a.ToString() + " ");
            }
  Console.WriteLine("\n\nÇaprazlama Sonrası Kromozom:\n");
            cocuk.GenleriYazdir();
            return cocuk;
        }
        public void diziyiKaristir(ArrayList dizi)
        {
            ArrayList source = dizi;
            Random rnd = new Random();
            for (int inx = source.Count - 1; inx > 0; inx--)
            {
                int position = rnd.Next(inx + 1);
                object temp = source[inx];
                source[inx] = source[position];
                source[position] = temp;
            }
        }
        public populasyon konumTabanliCaprazla(populasyon p,double
caprazlamaOrani)
        {
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            double caprazlanacakGenSayisi =
Math.Round((p.kromozomlar[0].Genler.Count * caprazlamaOrani), 0);

            ArrayList pop = new ArrayList();
            for (int i = 1; i < p.populasyonBuyuklugu+1; i++)
            {
                pop.Add(i);
            }
            diziyiKaristir(pop);

            Dictionary<int, int> d = new Dictionary<int,
int>();//çaprazlama uygulanacak çiftleri oluşturuyoruz
            for (int i = 0; i < pop.Count-1; i+=2)
            {
                d.Add((int)pop[i], (int)pop[i + 1]);
            }//çiftler oluşturuldu
            ArrayList caprazlamaSonrasiKromozomlar = new
ArrayList(p.populasyonBuyuklugu);

 for (int i = 0; i < p.populasyonBuyuklugu; i++)
            {
                caprazlamaSonrasiKromozomlar.Add(0);
            }
            foreach(KeyValuePair<int, int> kvp in d) //bu fonksiyon
oluşturulan çiftlere göre çaprazlama işlemi yapılmış kromozomları
oluşturuyor
            {
                kromozom[] caprazlanacakCift = new kromozom[2];
                kromozom[] yeniCift=new kromozom[2];//key 2  value 6
               caprazlanacakCift[0]=p.kromozomlar[kvp.Key-1];
                caprazlanacakCift[1]=p.kromozomlar[kvp.Value-1];
                Console.WriteLine("\n\nÇaprazlanan Kromozom Çifti:" +
kvp.Key.ToString() + "-" + kvp.Value.ToString());
                Console.WriteLine("\nKromozom1:" + kvp.Key.ToString()
+ "\n"); caprazlanacakCift[0].GenleriYazdir();
                Console.WriteLine("\nKromozom2:" +
kvp.Value.ToString() + "\n"); caprazlanacakCift[1].GenleriYazdir();
                yeniCift[0] =
ciftiCaprazlaKonumTabanli(caprazlanacakCift,
(int)caprazlanacakGenSayisi);

                caprazlanacakCift[0]=p.kromozomlar[kvp.Value-1];
                caprazlanacakCift[1]=p.kromozomlar[kvp.Key-1];
                Console.WriteLine("\n\nÇaprazlanan Kromozom Çifti:" +
kvp.Value.ToString() + "-" + kvp.Key.ToString());
                Console.WriteLine("\nKromozom1:" +
kvp.Value.ToString() + "\n"); caprazlanacakCift[0].GenleriYazdir();
                Console.WriteLine("\nKromozom2:" +
kvp.Key.ToString()+"\n"); caprazlanacakCift[1].GenleriYazdir();
                yeniCift[1] =
ciftiCaprazlaKonumTabanli(caprazlanacakCift,
(int)caprazlanacakGenSayisi);
                caprazlamaSonrasiKromozomlar[kvp.Key-1] = yeniCift[0];
                caprazlamaSonrasiKromozomlar[kvp.Value-1] =
yeniCift[1];
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            }
            kromozom[] capSonrasiKromozomlar = new
kromozom[caprazlamaSonrasiKromozomlar.Count];
            Console.WriteLine("\nÇaprazlama Sonrası:\n----------------
---------------");
            for (int i = 0; i < caprazlamaSonrasiKromozomlar.Count;
i++)
            {
capSonrasiKromozomlar[i]=(kromozom)caprazlamaSonrasiKromozomlar[i];
                capSonrasiKromozomlar[i].GenleriYazdir();
            }
            p = new populasyon(capSonrasiKromozomlar);
           return p;

        }
        public void MutasyonIkiliYerDegistirme(populasyon p)
        {
            populasyon yeniPop = new populasyon(p.kromozomlar);
            int genPozisyon1=0;
            int genPozisyon2=0;
            ArrayList genDegerListe = new ArrayList();
            for (int i = 0; i < p.kromozomlar[0].Genler.Count; i++)
            {
                genDegerListe.Add(i);
            }
            Random rnd;
            rnd = new Random();
            foreach (kromozom k in yeniPop.kromozomlar)
            {
                int minDeger=1;
                int maxDeger=(int)yeniPop.kromozomlar[0].Genler.Count;
                genPozisyon1 = rnd.Next(minDeger,maxDeger);
                genDegerListe.Remove(genPozisyon1);
                genPozisyon2 = rnd.Next(minDeger,
genDegerListe.Count+1);
                genPozisyon2 = (int)genDegerListe[genPozisyon2-1];
                Console.WriteLine("\nMutasyon
Genleri:"+(genPozisyon1+1).ToString() + "," +
(genPozisyon2+1).ToString());
                Console.WriteLine("\nYeni Kromozom:");
                gen gen1=new gen();

                gen1=(gen)k.Genler[genPozisyon1];
                gen gen2 = new gen();
                gen2 = (gen)k.Genler[genPozisyon2];
                int gendeger1 = (int)gen2.genDeger;
                int gendeger2 = (int)gen1.genDeger;
                gen1.genDeger = gendeger1;
                gen2.genDeger = gendeger2;
                k.Genler[genPozisyon1] = gen1;
                k.Genler[genPozisyon2] = gen2;
                k.GenleriYazdir();
            }
        }
    }
}
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;//Genleri Arraylist olarak tarifliyoruz
namespace UrunAile
{
    class gen
    {
        public Random rnd;
        public object genDeger;//gen değeri sayı ya da harf
olabilir,genellemek amacıyla object yani nesne olarak tarifledik
        //bu fonksiyon istenen değer aralığında gen türetiyor
        public void RassalGenTuret(int min,int max)
        {
            Random Rnd = new Random();
            int genSayiDeger = Rnd.Next(min, max);
            this.genDeger=(object)genSayiDeger;//burada genDeger
değişkeni int tipinde,ama önceden object olarak tarfilendiğinden
burada tip dönüşümü yapılıyor
        }
        public void RassalGenTuret( ArrayList eksikDegerler,int
genIndeks)//bu fonksiyon öncekinin bir farklı versiyonu,burada henüz
türetilmemiş değer kümesinden gen değerleri türetilerek tekrarlar
engelleniyor
        {
            this.genDeger = (object)eksikDegerler[genIndeks];//burada
eksik değer kümesinden rastgele bir değer seçip genin değerine atadık

        }
        public void GenDegeriYazdir()
        {
            Console.Write("["+genDeger.ToString()+"]");
        }
    }
}

using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
using Excel=Microsoft.Office.Interop.Excel;

namespace UrunAile
{
    class kromozom
    {
        Random rnd;
        double[,] gecisMatris;
        public kromozom()
        {
            gecisMatris = new double[5, 5];
        }
        int kromozomUzunluk;
        public ArrayList Genler;
        public object minGenDeger;
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
        public object maxGenDeger;
        public kromozom(int uzunluk,int min,int max)//burada sadece
kromozom uzunluğu,min ve max değer verildiğinde kromozom sınıfı
verilen uzunlukta başlatılıyor
        {
            gecisMatris = new double[Program.veridizi.GetLength(0),
Program.veridizi.GetLength(1)];

            /*
            gecisMatris[1, 1] = 3;
            gecisMatris[1, 2] = 4;
            gecisMatris[1, 3] = 1;
            gecisMatris[1, 4] = 42;
            gecisMatris[1, 0] = 32;
            gecisMatris[2, 1] = 41;
            gecisMatris[2,2] = 13;
            gecisMatris[2, 3] = 14;
            gecisMatris[2, 4] = 32;
            gecisMatris[2, 0] = 46;
            gecisMatris[3, 1] = 63;
            gecisMatris[3, 2] = 47;
            gecisMatris[3, 3] = 38;
            gecisMatris[3, 4] = 44;
            gecisMatris[3, 0] = 32;
            gecisMatris[4, 1] = 124;
            gecisMatris[4, 2] = 33;
            gecisMatris[4, 3] = 45;
           gecisMatris[4, 4] = 37;
            gecisMatris[4, 0] = 94;
            gecisMatris[0, 1] = 83;
            gecisMatris[0, 2] = 49;
            gecisMatris[0, 3] = 35;
            gecisMatris[0, 4] = 64;
            gecisMatris[0, 0] = 37;
            */
            gecisMatris = Program.veridizi;
            minGenDeger = (object)min;
            maxGenDeger = (object)max;
            this.kromozomUzunluk = uzunluk;
            Genler = new ArrayList(kromozomUzunluk);
        }

        public kromozom(ArrayList genler)//burada genler verildiğinde
kromozom sınıfı başlatılıyor.Örneğin çaprazlamadan sonra kromozomlar
oluşturulurken bazı kromozomlar aynen korunacak.Onları sonraki nesilde
tekrar yaratmak için gen değerlerini aktarıyoruz.
        {
            gecisMatris = new double[Program.veridizi.GetLength(0),
Program.veridizi.GetLength(1)];
            /*
            gecisMatris[1, 1] = 3;
            gecisMatris[1, 2] = 4;
            gecisMatris[1, 3] = 1;
            gecisMatris[1, 4] = 42;
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            gecisMatris[1, 0] = 32;
            gecisMatris[2, 1] = 41;
            gecisMatris[2, 2] = 13;
            gecisMatris[2, 3] = 14;
            gecisMatris[2, 4] = 32;
            gecisMatris[2, 0] = 46;
            gecisMatris[3, 1] = 63;
            gecisMatris[3, 2] = 47;
            gecisMatris[3, 3] = 38;
            gecisMatris[3, 4] = 44;
            gecisMatris[3, 0] = 32;
            gecisMatris[4, 1] = 124;
            gecisMatris[4, 2] = 33;
            gecisMatris[4, 3] = 45;
            gecisMatris[4, 4] = 37;
            gecisMatris[4, 0] = 94;
            gecisMatris[0, 1] = 83;
            gecisMatris[0, 2] = 49;
            gecisMatris[0, 3] = 35;
            gecisMatris[0, 4] = 64;
            gecisMatris[0, 0] = 37;*/
           // minGenDeger = (object)min;
            //maxGenDeger = (object)max;
            gecisMatris = Program.veridizi;
            this.kromozomUzunluk = genler.Count;
            Genler = new ArrayList(kromozomUzunluk);
            this.Genler = genler;
        }
        public void GenleriYazdir()
        {
            foreach (gen g in Genler)//Genler dizisindeki her geni
ekrana yazdır
            {
                g.GenDegeriYazdir();
            }
            Console.WriteLine(" Uyum değeri:" +
this.uyumFonksiyonuHesapla().ToString());
        }
        public void genleriTuret()
        {
            Genler = new ArrayList();//genler dizisini sıfırlıyoruz
            ArrayList eksikGenler=new ArrayList();//genlerin dışında
bir de henüz türetilmemiş gen değerlerini tutan bir dizi
tanımladık,yeni genleri bu değerler içinden türeteceğiz

            for (int i = (int)minGenDeger; i < (int)maxGenDeger+1;
i++)//burada eksikGenler dizisini tüm olası değerler ile dolduruyoruz.
            {
                eksikGenler.Add(i);

            }
            rnd = new Random();
            for (int i = 0; i < kromozomUzunluk+1; i++)//kromozom
uzunluğu kadar gen türet
            {
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
                    gen g = new gen();
                    System.Threading.Thread.Sleep(5);

                    int genIndeks = rnd.Next(0, eksikGenler.Count);

                    g.RassalGenTuret(eksikGenler,genIndeks);//her bir
gen daha önceden türetilen gen değerlerinden farklı olacak

                Genler.Add(g);

                    eksikGenler.Remove((int)g.genDeger);

            }

        }
        public double uyumFonksiyonuHesapla()//daha önce tanımlanmış
geçiş matrisine göre sıralamanın uyum değerini hesaplıyor
        {
            double uyumDeger;
            uyumDeger = 0;
            for (int i = 0; i < Genler.Count-1; i++)
            {

                    gen g1 = new gen();
                    gen g2 = new gen();
                    g1 = (gen)Genler[i];//önceki gen
                    g2=(gen)Genler[i+1];//sonraki gen

                    uyumDeger += gecisMatris[(int)g1.genDeger-1,
(int)g2.genDeger-1];

            }
            return uyumDeger;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
namespace UrunAile
{
    class populasyon
    {
       public int populasyonBuyuklugu;
        public double eniyiUyumDegeri;
        public int eniyiUyumDegeriKromozomIndeksi;
        public kromozom[] kromozomlar;
        public int kromozomUzunluk;
        public ArrayList uyumDegerleri;
        public populasyon(kromozom[] kromozomlar)//verilen
kromozomlardan bir populasyon oluştur
        {
            this.eniyiUyumDegeri = 999999999;
            this.populasyonBuyuklugu = Program.popBuyukluk;
            this.kromozomlar=new kromozom[populasyonBuyuklugu];
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            this.kromozomlar=kromozomlar;
            this.kromozomUzunluk=kromozomlar[0].Genler.Count;
            this.uyumDegerleri = new ArrayList();

            for (int i = 0; i < kromozomlar.GetLength(0); i++)
//kromozomların uyum değerlerini bul
            {

this.uyumDegerleri.Add(kromozomlar[i].uyumFonksiyonuHesapla());
            }
            for (int i = 0; i < uyumDegerleri.Count; i++) //en iyi
uyum değeri olan kromozomu ve uyum değerini bul
            {

               if ((double)uyumDegerleri[i] < eniyiUyumDegeri)
                {

                    eniyiUyumDegeri = (double)uyumDegerleri[i];
                   eniyiUyumDegeriKromozomIndeksi = i;
                }
            }
        }
        public populasyon(int kromozomUzunluk, int maximum,int
minimum,int populasyonBuyuklugu)//verilen değerleri göre populasyon
oluştur
        {
            this.eniyiUyumDegeri = 999999999;
            this.populasyonBuyuklugu = Program.popBuyukluk;
            this.kromozomlar=new kromozom[populasyonBuyuklugu];
            this.kromozomUzunluk=kromozomUzunluk;
            this.uyumDegerleri = new ArrayList();
            for (int i = 0; i < kromozomlar.GetLength(0);
i++)//kromozomların uyum değerlerini bul
            {
                this.kromozomlar[i] = new kromozom(kromozomUzunluk-1,
minimum, maximum);

                this.kromozomlar[i].genleriTuret();

                double mevcutUyum;
                mevcutUyum = kromozomlar[i].uyumFonksiyonuHesapla();
                uyumDegerleri.Add(mevcutUyum);
            }
            for (int i = 0; i < uyumDegerleri.Count; i++) //en iyi
uyum değeri olan kromozomu ve uyum değerini bul
            {
                if ((double)uyumDegerleri[i] < eniyiUyumDegeri)
                {
                    eniyiUyumDegeri = (double)uyumDegerleri[i];
                    eniyiUyumDegeriKromozomIndeksi = i;
                }
            }
        }
        public void populasyonBaslat()
        {
            Console.WriteLine( "Başlangıç Populasyonu:\n");
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            foreach (kromozom kr in kromozomlar)
        {
                kr.GenleriYazdir();
            }

            Console.WriteLine( "-----------------------------------");
            Console.Write("En iyi kromozom:");

    kromozomlar[eniyiUyumDegeriKromozomIndeksi].GenleriYazdir();

using System;
using System.Collections.Generic;
using System.Text;
using Excel = Microsoft.Office.Interop.Excel;
namespace UrunAile
{
    class Program
    {
        public static double[,] veridizi;
        public static int popBuyukluk;
        public static void iterasyonYap(populasyon p, ga
g,System.Collections.ArrayList al)
        {
            g.CalisanIterasyonSayisi++;
            p = g.turnuvaSecim(2, p);
            Console.WriteLine("\nSeçim Sonrası:\n----------------------
----------");
            foreach (kromozom k in p.kromozomlar)
            {
                k.GenleriYazdir();
            }
            if
(p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi].uyumFonksiyonuHesapla
() < g.eniyiUyumDegeri)
            {
               g.eniyiKromozom =
p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi];
                g.eniyiUyumDegeri =
p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi].uyumFonksiyonuHesapla(
);
            }
            p = g.konumTabanliCaprazla(p, 0.4);
            Console.WriteLine("\nÇaprazlama Sonrası En İyi
Kromozom:\n");
p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi].GenleriYazdir();
            g.MutasyonIkiliYerDegistirme(p);
            Console.WriteLine("\nMutasyon Sonrası En İyi
Kromozom:\n");

p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi].GenleriYazdir();
            if
(p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi].uyumFonksiyonuHesapla
() < g.eniyiUyumDegeri)
            {
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
                g.eniyiKromozom =
p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi];
                g.eniyiUyumDegeri =
p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi].uyumFonksiyonuHesapla(
);
            }
            Console.WriteLine("\n" +
(g.CalisanIterasyonSayisi).ToString() + ". iterasyon sonucu\n---------
----------------------");
            Console.WriteLine("\nEn İyi Kromozom:");
            g.eniyiKromozom.GenleriYazdir();
     double uyum1 = g.eniyiKromozom.uyumFonksiyonuHesapla();
            al.Add(uyum1);
        }
       static void nesneBirak(object obj)
        {
            try
            {

System.Runtime.InteropServices.Marshal.ReleaseComObject(obj);
                obj = null;
            }
            catch (Exception ex)
            {
                obj = null;
                Console.WriteLine("Nesne bırakılamıyor " +
ex.ToString());
            }
            finally
            {
                GC.Collect();
            }
        }
        static void Main(string[] args)
        {
            popBuyukluk = 1;
            Console.WriteLine("Populasyon büyüklüğünü giriniz:");
            popBuyukluk = Int32.Parse(Console.ReadLine());
            while (popBuyukluk % 2 == 1)
            {
                Console.WriteLine("Lütfen Çift Sayı Giriniz:");
                popBuyukluk = Int32.Parse(Console.ReadLine());
            }
            Excel.Application xlApp;
            Excel.Workbook xlWorkBook;
            Excel.Worksheet xlWorkSheet;
            Excel.Range range;
            string str;
            int rCnt = 0;
            int cCnt = 0;
   xlApp = new Excel.ApplicationClass();
            xlApp.Visible = true;
            xlApp.UserControl = true;
            System.Globalization.CultureInfo oldCI =
System.Threading.Thread.CurrentThread.CurrentCulture;
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            oldCI = new System.Globalization.CultureInfo("en-US");
            System.Threading.Thread.CurrentThread.CurrentCulture =
oldCI;
            xlWorkBook =
xlApp.Workbooks.Open(System.IO.Directory.GetCurrentDirectory()+"/veri.
xls", 0, true, 5, "", "", true,
Microsoft.Office.Interop.Excel.XlPlatform.xlWindows, "\t", false,
false, 0, true, 1, 0);
            xlWorkSheet =
(Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);
            range = xlWorkSheet.UsedRange;
            veridizi=new double[range.Rows.Count,range.Columns.Count];
            for (int i = 1; i < range.Rows.Count + 1; i++)
            {
                for (int j = 1; j < range.Columns.Count + 1; j++)
                {
                    veridizi[i - 1, j - 1] =
(double)((Excel.Range)xlWorkSheet.Cells[i, j]).Value2;
                }
            }
            xlWorkBook.Close(true, null, null);
            xlApp.Quit();
            nesneBirak(xlWorkSheet);
            nesneBirak(xlWorkBook);
            nesneBirak(xlApp);

            System.Collections.ArrayList uyumDegerleri = new
System.Collections.ArrayList();
            Console.WriteLine("İterasyon sayısını giriniz:");
            int iterasyonSayisi = Int32.Parse(Console.ReadLine());
            int krUzunluk = veridizi.GetLength(0);
            populasyon p = new populasyon(krUzunluk, krUzunluk, 1,
popBuyukluk);//5 GEN uzunluğunda,her genin maximum değeri 5,minimum
değeri 1 olacak,populasyon büyüklüğü 10 olacak şekilde bir populasyon
oluştur.

            p.populasyonBaslat();
            ga g = new ga();
            g.eniyiKromozom =
p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi];
            g.eniyiUyumDegeri =
p.kromozomlar[p.eniyiUyumDegeriKromozomIndeksi].uyumFonksiyonuHesapla(
);
            populasyon secimSonrasi = g.turnuvaSecim(2,p);//Başlangıç
populasyonuna seçim işlemi yap


            Console.WriteLine("\nSeçim Sonrası:\n----------------------
----------");
            foreach (kromozom k in secimSonrasi.kromozomlar)
            {
                k.GenleriYazdir();
            }
            secimSonrasi = g.konumTabanliCaprazla(secimSonrasi, 0.4);
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            Console.WriteLine("\nÇaprazlama Sonrası En İyi
Kromozom:\n");

secimSonrasi.kromozomlar[secimSonrasi.eniyiUyumDegeriKromozomIndeksi].
GenleriYazdir();

            g.MutasyonIkiliYerDegistirme(secimSonrasi);

         Console.WriteLine("\nMutasyon Sonrası En İyi Kromozom:\n");

secimSonrasi.kromozomlar[secimSonrasi.eniyiUyumDegeriKromozomIndeksi].
GenleriYazdir();
            g.eniyiKromozom =
secimSonrasi.kromozomlar[secimSonrasi.eniyiUyumDegeriKromozomIndeksi];
            g.eniyiKromozomIterasyonIndeksi = 1;
            g.eniyiUyumDegeri =
secimSonrasi.kromozomlar[secimSonrasi.eniyiUyumDegeriKromozomIndeksi].
uyumFonksiyonuHesapla();

Console.WriteLine("\n"+g.CalisanIterasyonSayisi.ToString()+".
iterasyon sonucu\n--------------------------------");
            Console.WriteLine("\nEn İyi Kromozom:");
           g.eniyiKromozom.GenleriYazdir();
            double uyum1 = g.eniyiKromozom.uyumFonksiyonuHesapla();
            uyumDegerleri.Add(uyum1);
            for (int i = 1; i < iterasyonSayisi; i++)
            {
                iterasyonYap(secimSonrasi, g,uyumDegerleri);

            }
            Console.WriteLine("\n" +
g.CalisanIterasyonSayisi.ToString() + " iterasyon sonucu bulunan en
iyi kromozom:\n");
            g.eniyiKromozom.GenleriYazdir();
            Console.ReadLine();

        }
    }
}
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
namespace UrunAile
{
    class ga
    {
        public kromozom eniyiKromozom;
        public double eniyiUyumDegeri;
        public int CalisanIterasyonSayisi;//Toplamda kaç iterasyon
çalıştırıldığını tutacak

        public int eniyiKromozomIterasyonIndeksi;//en iyi kromozomun
kaçıncı nesilde bulunduğunu tutacak
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
        public ga()
        {
            eniyiKromozom = new kromozom();
            CalisanIterasyonSayisi = 1;
        }
        public populasyon turnuvaSecim(int turnuvaBuyuklugu,
populasyon p)
        {
            kromozom[] kr = new kromozom[Program.popBuyukluk];
            //ArrayList krom = new ArrayList();
            ArrayList kromozomlar = new ArrayList();

            for (int i = 0; i < p.populasyonBuyuklugu; i++)
            {
                if (i % turnuvaBuyuklugu == 0)
                {
                    kromozomlar.Clear();
                    for (int j = 0; j < turnuvaBuyuklugu; j++)
                    {

                        kromozomlar.Add(p.kromozomlar[i+j]);
                    }
                    double eniyiDeger=99999999;
                    kromozom eniyiKrom=new kromozom();
                    for(int a=0;a<kromozomlar.Count;a++)
                    {
                        kromozom kr2=new kromozom();

                        kr2=(kromozom)kromozomlar[a];
                        if (eniyiDeger > kr2.uyumFonksiyonuHesapla())
                        {
                            eniyiDeger = kr2.uyumFonksiyonuHesapla();
                            eniyiKrom = kr2;
                        }

                    }
                    for (int c = 0; c < kromozomlar.Count; c++)
                    {
                        kr[i + c] = eniyiKrom;
                    }
                }
            }
            populasyon pop = new populasyon(kr);
            return pop;

        }
        kromozom ciftiCaprazlaKonumTabanli(kromozom[] cift, int
caprazlanacakGenSayisi)
        {
            kromozom kr1 = new kromozom();
            kr1 = cift[0];//1. kromozomun koyasını oluştur
            kromozom kr2 = new kromozom();
            kr2 = cift[1];//2. kromozomun kopyasını oluştur
            kromozom cocuk = new kromozom();
            ArrayList cocukGenler = new ArrayList(kr1.Genler.Count);
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            ArrayList genDeger = new ArrayList();

            Random rnd;
            ArrayList caprazlamaNoktalari = new
ArrayList(caprazlanacakGenSayisi);
            for (int i = 1; i < kr1.Genler.Count + 1; i++)
            {
                genDeger.Add(i);
            }
            for (int i = 0; i < caprazlanacakGenSayisi;
i++)//çaprazlama noktalarını rassal olarak oluştur
            {
                rnd = new Random();
                int cprNoktasi = rnd.Next(0, genDeger.Count-1);
                cprNoktasi = (int)genDeger[cprNoktasi];
                caprazlamaNoktalari.Add(cprNoktasi);
                genDeger.Remove(cprNoktasi);
            }

            caprazlamaNoktalari.Sort();//küçükten büyüğe sırala

            ArrayList caprazlamaNoktalariGenDegerleri = new
ArrayList(caprazlamaNoktalari.Count);
            ArrayList eksikDegerler = new
ArrayList(caprazlamaNoktalari.Count);
            for (int i = 0; i < kr2.Genler.Count; i++)
            {
                gen g = new gen();
                g = (gen)kr2.Genler[i];
                int deger = (int)g.genDeger;
                eksikDegerler.Add(deger);
                cocukGenler.Add(0);
            }
            for (int i = 0; i < caprazlamaNoktalari.Count;
i++)//çaprazlama noktalarındaki gen değerlerini ata
            {
                gen g = new gen();
                g = (gen)kr1.Genler[(int)caprazlamaNoktalari[i]];
                caprazlamaNoktalariGenDegerleri.Add(g.genDeger);
                eksikDegerler.Remove(g.genDeger);
            }
            int cprIndeks = 0;
            for (int i = 0; i < kr1.Genler.Count; i++)//sabit kalacak
pozisyonların genlerini kopyala
            {
                if (caprazlamaNoktalari.Contains(i))
                {
                    gen g = new gen();
                    g.genDeger =
caprazlamaNoktalariGenDegerleri[cprIndeks];
                    cocukGenler[i]=g;
                    cprIndeks++;
                }
            }
            int eksikIndeks = 0;
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            for (int i = 0; i < kr1.Genler.Count; i++)
            {
                if (!caprazlamaNoktalari.Contains(i))
                {
                    gen g=new gen();
                    g.genDeger=eksikDegerler[eksikIndeks];
                    eksikIndeks++;
                    cocukGenler[i] = g;
                }
            }

            cocuk = new kromozom(cocukGenler);
            Console.Write("\nKonum Tabanlı Çaprazlama Sabit
Pozisyonlar:");
            foreach (int i in caprazlamaNoktalari)
            {
                int a = i + 1;
                Console.Write(a.ToString() + " ");
            }
            Console.WriteLine("\n\nÇaprazlama Sonrası Kromozom:\n");
            cocuk.GenleriYazdir();
            return cocuk;
        }
        public void diziyiKaristir(ArrayList dizi)
        {
            ArrayList source = dizi;
            Random rnd = new Random();
            for (int inx = source.Count - 1; inx > 0; inx--)
            {
                int position = rnd.Next(inx + 1);
                object temp = source[inx];
                source[inx] = source[position];
                source[position] = temp;
            }
        }
        public populasyon konumTabanliCaprazla(populasyon p,double
caprazlamaOrani)
        {
            double caprazlanacakGenSayisi =
Math.Round((p.kromozomlar[0].Genler.Count * caprazlamaOrani), 0);

            ArrayList pop = new ArrayList();
            for (int i = 1; i < p.populasyonBuyuklugu+1; i++)
            {
                pop.Add(i);
            }
            diziyiKaristir(pop);
            Dictionary<int, int> d = new Dictionary<int,
int>();//çaprazlama uygulanacak çiftleri oluşturuyoruz
            for (int i = 0; i < pop.Count-1; i+=2)
            {
                d.Add((int)pop[i], (int)pop[i + 1]);
            }//çiftler oluşturuldu
            ArrayList caprazlamaSonrasiKromozomlar = new
ArrayList(p.populasyonBuyuklugu);
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
            for (int i = 0; i < p.populasyonBuyuklugu; i++)
            {
                caprazlamaSonrasiKromozomlar.Add(0);
            }
            foreach(KeyValuePair<int, int> kvp in d) //bu fonksiyon
oluşturulan çiftlere göre çaprazlama işlemi yapılmış kromozomları
oluşturuyor
            {
                kromozom[] caprazlanacakCift = new kromozom[2];
                kromozom[] yeniCift=new kromozom[2];//key 2  value 6
                caprazlanacakCift[0]=p.kromozomlar[kvp.Key-1];
                caprazlanacakCift[1]=p.kromozomlar[kvp.Value-1];
                Console.WriteLine("\n\nÇaprazlanan Kromozom Çifti:" +
kvp.Key.ToString() + "-" + kvp.Value.ToString());

                Console.WriteLine("\nKromozom1:" + kvp.Key.ToString()
+ "\n"); caprazlanacakCift[0].GenleriYazdir();

                Console.WriteLine("\nKromozom2:" +
kvp.Value.ToString() + "\n"); caprazlanacakCift[1].GenleriYazdir();
                yeniCift[0] =
ciftiCaprazlaKonumTabanli(caprazlanacakCift,
(int)caprazlanacakGenSayisi);

                caprazlanacakCift[0]=p.kromozomlar[kvp.Value-1];
                caprazlanacakCift[1]=p.kromozomlar[kvp.Key-1];
                Console.WriteLine("\n\nÇaprazlanan Kromozom Çifti:" +
kvp.Value.ToString() + "-" + kvp.Key.ToString());
                Console.WriteLine("\nKromozom1:" +
kvp.Value.ToString() + "\n"); caprazlanacakCift[0].GenleriYazdir();

                Console.WriteLine("\nKromozom2:" +
kvp.Key.ToString()+"\n"); caprazlanacakCift[1].GenleriYazdir();
                yeniCift[1] =
ciftiCaprazlaKonumTabanli(caprazlanacakCift,
(int)caprazlanacakGenSayisi);
                caprazlamaSonrasiKromozomlar[kvp.Key-1] = yeniCift[0];
                caprazlamaSonrasiKromozomlar[kvp.Value-1] =
yeniCift[1];
            }
            kromozom[] capSonrasiKromozomlar = new
kromozom[caprazlamaSonrasiKromozomlar.Count];
            Console.WriteLine("\nÇaprazlama Sonrası:\n----------------
---------------");

            for (int i = 0; i < caprazlamaSonrasiKromozomlar.Count;
i++)
            {
capSonrasiKromozomlar[i]=(kromozom)caprazlamaSonrasiKromozomlar[i];
                capSonrasiKromozomlar[i].GenleriYazdir();
            }
            p = new populasyon(capSonrasiKromozomlar);
            return p;

        }
```

Ek.1 (devam) Önerilen Genetik Algoritmanın Program Kodları

```
        public void MutasyonIkiliYerDegistirme(populasyon p)
        {

            populasyon yeniPop = new populasyon(p.kromozomlar);
            int genPozisyon1=0;
            int genPozisyon2=0;
            ArrayList genDegerListe = new ArrayList();
            for (int i = 0; i < p.kromozomlar[0].Genler.Count; i++)
            {
                genDegerListe.Add(i);
            }
            Random rnd;
            rnd = new Random();
            foreach (kromozom k in yeniPop.kromozomlar)
            {

                int minDeger=1;

                int maxDeger=(int)yeniPop.kromozomlar[0].Genler.Count;

                genPozisyon1 = rnd.Next(minDeger,maxDeger);
                genDegerListe.Remove(genPozisyon1);
                genPozisyon2 = rnd.Next(minDeger,
genDegerListe.Count+1);
                genPozisyon2 = (int)genDegerListe[genPozisyon2-1];
                Console.WriteLine("\nMutasyon
Genleri:"+(genPozisyon1+1).ToString() + "," +
(genPozisyon2+1).ToString());
                Console.WriteLine("\nYeni Kromozom:");
                gen gen1=new gen();
                gen1=(gen)k.Genler[genPozisyon1];
                gen gen2 = new gen();
                gen2 = (gen)k.Genler[genPozisyon2];
                int gendeger1 = (int)gen2.genDeger;
                int gendeger2 = (int)gen1.genDeger;
                gen1.genDeger = gendeger1;
                gen2.genDeger = gendeger2;
                k.Genler[genPozisyon1] = gen1;
                k.Genler[genPozisyon2] = gen2;
                k.GenleriYazdir();
            }

        }
    }
}
```

Ek.2 Matemetiksel Modelin ve Geliştirilan Matematiksel Modelin Kullandığı Veri dosyasının bir bölümü

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
1		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
2	1	999	57	10	148	118	108	175	137	183	187	17	43	3	96	172	124	115	7	110	12	22	22	31	110	194	184	40	76	84
3	2	38	999	174	137	145	84	104	106	98	48	165	88	42	197	167	130	171	195	62	85	193	131	195	52	2	56	3	192	88
4	3	69	45	999	150	189	165	151	180	22	20	179	46	169	172	175	30	16	164	74	185	47	129	158	171	15	16	151	12	190
5	4	60	172	153	999	125	158	194	192	34	146	155	147	31	30	195	67	106	179	16	120	4	175	144	22	178	5	36	65	187
6	5	161	91	3	119	999	167	29	70	156	70	65	136	199	58	21	184	39	136	188	175	70	200	119	58	52	172	34	60	192
7	6	88	38	200	2	44	999	130	24	189	14	130	116	46	160	117	144	113	156	115	45	71	164	89	115	199	98	57	119	110
8	7	135	58	50	34	199	47	999	170	117	46	33	143	168	171	138	112	103	4	130	182	150	100	57	127	137	121	175	111	145
9	8	88	185	51	72	167	121	75	999	45	140	89	143	19	4	80	142	89	87	156	23	22	27	17	36	100	63	47	111	152
10	9	8	159	120	67	183	67	21	134	999	155	194	47	185	160	124	73	102	152	96	107	51	46	47	198	31	6	22	32	143
11	10	100	1	113	113	133	64	177	127	1	999	73	76	134	79	61	188	114	106	63	10	119	104	17	135	20	141	166	101	49
12	11	178	9	79	30	159	85	178	20	6	196	999	164	48	113	22	98	29	170	71	155	191	107	11	18	109	146	123	12	168
13	12	156	39	119	17	26	192	38	14	144	181	129	999	70	166	166	12	97	168	66	172	47	179	140	161	188	175	41	62	63
14	13	186	156	155	187	172	90	81	7	60	69	80	63	999	128	53	195	18	143	40	61	15	190	94	87	97	128	21	87	105
15	14	57	185	154	169	107	2	119	145	19	199	89	86	95	999	194	117	148	11	156	190	84	195	192	147	12	47	124	151	34
16	15	180	40	150	30	142	31	181	98	81	108	186	156	94	116	999	82	177	45	116	15	10	129	71	172	165	140	124	12	141
17	16	163	80	133	13	88	161	98	123	39	197	82	170	136	98	151	999	46	13	155	42	178	80	28	142	24	119	101	156	163
18	17	153	158	2	13	70	177	102	53	158	67	182	61	131	66	143	20	999	53	197	159	93	147	52	54	28	13	156	20	106
19	18	47	58	199	75	55	108	10	28	189	194	24	124	196	141	98	83	88	999	132	166	3	182	176	64	24	159	126	31	34
20	19	108	93	41	137	165	126	51	62	31	123	85	165	115	69	98	128	125	2	999	92	23	17	166	122	151	23	126	157	198
21	20	165	99	61	46	42	33	33	53	126	142	73	47	57	107	47	137	128	133	67	999	2	188	31	25	141	96	113	63	46
22	21	137	132	122	78	111	28	15	18	131	123	79	177	88	26	181	162	141	146	85	21	999	85	126	97	70	27	200	44	6
23	22	78	150	116	194	155	200	98	116	161	156	52	32	174	86	67	42	192	65	102	1	140	999	133	16	198	12	194	145	101
24	23	163	194	11	160	133	132	97	182	139	194	196	84	60	151	5	16	58	116	134	84	8	137	999	1	36	104	19	20	166
25	24	125	54	108	171	36	174	140	186	167	159	107	75	153	47	10	14	17	84	51	49	11	94	17	999	42	63	59	134	12
26	25	108	8	105	42	134	2	1	88	50	117	111	121	3	11	71	102	15	137	67	148	165	35	131	47	999	88	171	5	23
27	26	67	131	163	45	107	17	193	89	17	120	156	47	163	143	118	109	47	45	60	136	9	171	64	92	191	999	98	47	128
28	27	14	125	179	39	177	104	32	121	186	89	42	13	94	30	16	40	25	31	29	113	17	119	199	103	71	122	999	125	98
29	28	83	123	27	68	14	67	94	97	99	116	4	121	74	18	130	50	50	27	133	56	115	22	168	134	172	53	106	999	40
30	29	169	16	193	180	55	109	184	164	59	166	29	137	180	56	93	189	176	160	189	49	104	146	27	134	149	112	9	178	999

Ek.3 Geliştirilen Genetik Algoritmanın Kullanıcı Arayüzü