T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MELEZ SEZGiSEL VE META-SEZGİSEL ALGORİTMALAR KULLANARAK ÇİZELGELEME PROBLEMLERİNİN ÇÖZÜMLENMESİ

MEHMET FATİH USLU

DOKTORA TEZİ
ENDÜSTRİ MÜHENDİSLİĞİ ANABİLİM DALI
ENDÜSTRİ MÜHENDİSLİĞİ PROGRAMI

DANIŞMAN
PROF. DR. HÜSEYİN BAŞLIGİL

İSTANBUL, 2016

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

MELEZ SEZGiSEL VE META-SEZGİSEL ALGORİTMALAR KULLANARAK ÇİZELGELEME PROBLEMLERİNİN ÇÖZÜMLENMESİ

Mehmet Fatih Uslu tarafından hazırlanan tez çalışması 28.07.2016 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Endüstri Mühendisliği Anabilim Dalı'nda **DOKTORA TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Hüseyin BAŞLIGİL

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Prof. Dr. Hüseyin BAŞLIGİL

Yıldız Teknik Üniversitesi _____

Prof. Dr. Semih ÖNÜT

Yıldız Teknik Üniversitesi _____

Prof. Dr. Ali Fuat GÜNERİ

Yıldız Teknik Üniversitesi _____

Prof. Dr. İsmail ADAK

Yalova Üniversitesi _____

Doç. Dr. Murat BASKAK

İstanbul Teknik Üniversitesi _____

ÖNSÖZ

Bu çalışmamda tez danışmanlığımı yürüten değerli hocam Prof. Dr. Hüseyin BAŞLIGİL'e, tez çalışmalarımın izlenmesi ve değerlendirilmesinde katkılarını esirgemeyen değerli hocalarım Prof. Dr. Semih ÖNÜT ve Doç. Dr. Murat BASKAK'a, manevi destekleri için anne-babama, kardeşime, eşime, oğluma ve arkadaşlarıma teşekkür ederim.

Haziran, 2016

Mehmet Fatih USLU

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

C_{max}	Çizelgenin toplam tamamlanma zamanı
X_{ab}	Karınca kolonisi optimizasyonunda (KKO) en iyi çözüm
X_{rb}	KKO'da son restarttan sonraki en iyi çözüm
X_{ib}	KKO'da son iterasyondaki en iyi çözüm
A	KKO'da feromon oranı
B	KKO'da çekicilik oranı
τ_0	KKO'da başlangıç feromon seviyesi
ρ	KKO'da iz buharlaşma oranı
C	KKO'da çekicilik hesabı için bir sabit
Q	KKO'da feromon güncellemede kullanılan bir sabit
N	Örneklem büyüklüğü
X_k	KKO'da bir karıncanın çözümünün tamamlanma zamanı
G_k	KKO'da çözüm için ziyaret edilmesi gerekli düğümler
S_k	KKO'da bir sonraki adımda gidilebilecek düğümler
J	İş
M	Makine
P	Yapay Bağışıklık Sistemi (YBS)'de popülasyon büyüklüğü
J	YBS'de jenerasyon sayısı
K	YBS'de klonlama olasılığı
M	YBS'de mutasyon olasılığı
R	YBS'de reseptör olasılığı
H	YBS'de hafıza hücresi sayısı
P	Genetik Algoritma (GA)'da popülasyon büyüklüğü
G	GA'da jenerasyon sayısı
P_m	GA'da mutasyon olasılığı
P_c	GA'da çaprazlama olasılığı
S	GA'da seçilim tipi
M	GA'da mutasyon tipi
C	GA'da çaprazlama tipi
γ	Çizelgeleme problemlerinde amaç

KISALTMA LİSTESİ

AAGA	Adaptif Tavlama Genetik Algoritma
ARPD	Ortalama göreceli sapma yüzdesi
BAT	Yarasa
BS	Bulanık sayı
CogAff	Cognition and Affect
CPU	Merkezi işlem birimi
CWSA	Klasik Ağırlıklı Tavlama Benzetimi
DP	Dinamik Programlama
DPSO	Ayrık PSO
EPDT	Extended Prim-Dijkstra Trade-off
FL	Bulanık Mantık
FSA	Bulanık Tavlama Benzetimi
GA	Genetik Algoritma
GAP	Açıklık Sezgiseli
GIS	Coğrafi bilgi sistemleri
GPU	Grafik işlem birimi
H-CogAff	Human-like CogAff
KKO	Karınca Kolonisi Optimizasyonu
MATLAB	Matrix Laboratory
MIT	Massachusetts Teknoloji Enstitüsü
NC	KKO'da iterasyon sayacı
NEH	Nawaz-Enscore-Ham
NP-hard	Nondeterministik Turing makinası ile polinom zamanda çözülebilen sınıf
NWSA	Normalize ağırlıklı tavlama benzetimi
PSE	Parametre tarama deneyi
PSO	Parçacık sürüsü optimizasyonu
RKGA	Rastgele anahtarlı genetik algoritma
RTCG	Çalışma zamanlı kod üretimi
SDST	Sıra bağımlı hazırlık zamanlı
SMT	Yüzey montaj teknolojisi
SR	Başarı yüzdesi
TFT	Toplam akış zamanı
ÜBS	Üçgensel bulanık sayı
TFN	Yamuk bulanık sayı

YBS	Yapay Bağışıklık Sistemi

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

MELEZ SEZGiSEL VE META-SEZGİSEL ALGORİTMALAR KULLANARAK ÇİZELGELEME PROBLEMLERİNİN ÇÖZÜMLENMESİ

Mehmet Fatih USLU

Endüstri Mühendisliği Anabilim Dalı

Doktora Tezi

Tez Danışmanı: Prof. Dr. Hüseyin BAŞLIGİL

Bu çalışmada literatürde farklı türde çizelgeleme problemleri için geliştirilen Yapay Bağışıklık Sistemi, Karınca Kolonisi Optimizasyonu ve Genetik Algoritma gibi çeşitli algoritmalar bir araya getirilip, istenilen parametreler ile oluşturulan çizelgeleme problemlerinin bu algoritmalar ile çözdürülüp çeşitli grafiklerle performans kıyaslaması yapılabileceği açık kaynak kodlu bir internet sayfası arayüzü ile JavaScript kütüphanesi oluşturulmuştur. Daha sonra Çizelgeleme problemleri için Genetik Algoritma parametrelerinin optimizasyonu üzerine çalışılmıştır. Testlerde performansı iyi bulunan parametreler kullanılarak Genetik Algoritma ile Karınca Kolonisi Optimizasyonu farklı problem tipleri için melez çözümler haline getirilmiş, bunların performansları incelenmiştir.

Daha sonra makinaların doluluk durumlarının veya performanslarının bulanık sayılarla ifade edilebileceği ortamlar için çizelgeleme problemlerinde bir bulanık mantık yaklaşımı geliştirilmiştir. Bu yaklaşım, makinaların doluluk oranının dolu/boş olarak değil de belli bir yüzde olarak ifade edilmesini temel alır. Böylece makinalar bâzı durumlarda aynı anda birden fazla işi belli yüzdeleriyle işleyebilirler veya belli yüzdeye göre düşük performanslı çalışabilirler. Geliştirilen yaklaşımlar, programdaki algoritmalara uygulanmış; sonuçları değerlendirilmiştir.

Anahtar Kelimeler: Genetik Algoritma, Karınca Kolonisi Optimizasyonu, Yapay Bağışıklık Sistemi, Entegre Proses Planlama ve Çizelgeleme, Melez Sezgisel Algoritmalar, Bulanık Mantık.

ABSTRACT

RESOLVING SCHEDULING PROBLEMS WITH USING HYBRID HEURISTIC AND METAHEURISTIC ALGORITHMS

Mehmet Fatih USLU

Department of Industrial Engineering

Ph.D. Thesis

Adviser: Prof. Dr. Hüseyin BAŞLIGİL

In this thesis, various algorithms like Genetic Algorithm, Artificial Immune System and Ant Colony Optimization which aim to solve Scheduling problems are combined and an HTML page & open source Javascript Library is developed as an interface which allows users to compare their algorithms with others in graphical interface. Users can create various types of Scheduling problems and solve through these algorithms in this application. Also, Genetic Algorithm's parameters are optimized for Scheduling problems and via these parameters, a hybrid algorithm developed using Ant Colony Optimization and Genetic Algorithm.

Then, a fuzzy logic approach proposed for scheduling problems which machines can have fuzzy fullness or performance level. This approach based on expressing machine fullness or performance as a percentage rather than busy or empty. As a result of this approach, machines can process more than one job at the same time or process times can differ from thought before. This approach integrated in algorithms used in application.

Keywords: Genetic Algorithm, Ant Colony Optimization, Artificial Immune System, Integrated Process Planning and Scheduling, Hybrid Heuristics, Fuzzy Logic.

BÖLÜM 1

GİRİŞ

Çizelgeleme problemleri günlük hayatın birçok noktasında karşımıza çıkabilen bir optimizasyon problemidir. Basitçe, yapılması gereken işler için kaynakların işleri yapacak birimlere dağıtımının optimize edilmesi şeklinde özetlenebilir. Günlük hayatta yapmamız gereken işlerin optimizasyonu bizim açımızdan önemli değildir. Ancak son yüzyılda üretim tesislerinin hacimlerinin büyümesi ve işlerin karmaşıklık düzeyinin gittikçe artması, üretim tesislerindeki çizelgeleme optimizasyonunun önemini artırmış ve araştırmacılar bu alana eğilmişlerdir.

Yine geçtiğimiz yüzyılda bilgisayar teknolojisinin ilerlemesi ile optimizasyon işlemlerini bilgisayarlar üstlenmiş, böylece optimizasyonun kullanım alanları genişlemiştir. Ancak bütün bu gelişmelere rağmen çizelgeleme gibi bâzı kompleks problemlerde klasik optimizasyon yöntemlerinin başarısız olması ile sezgisel yaklaşımlar geliştirilmeye başlanmıştır. Bunlardan bazıları da doğadaki çeşitli olayların modellenmesinden esinlenen algoritmalardır. Sezgisel optimizasyon tekniklerinin ortak özelliklerinden birisi de problemin en iyi çözümünü bulmaya çalışmayıp, bunun yerine kabul edilebilir bir zaman dilimi içerisinde olabildiğince iyi bir çözüm bulma stratejisi gütmeleridir. [1]

Bu çalışmada da çeşitli sezgisel algoritmaların farklı tip çizelgeleme problemleri üzerindeki performansları incelenmiş, bu performanslardan hareketle melez yaklaşımlar geliştirilmiştir. Bunlar yapılırken farklı tip çizelgeleme problemleri üretilip çeşitli algoritmaların bu problemler üzerinde test edilip kıyaslanabildiği açık kaynak kodlu bir platform tasarlanmıştır. Daha sonra çizelgeleme problemleri üzerinde bir bulanık mantık yaklaşımı uygulanmış, elde edilen sonuçlar değerlendirilmiştir.

Birinci bölümde çalışmanın başında ilgili alanda yapılan literatür taraması aktarılmıştır. Gerçekleştirilen tarama ile çeşitli çizelgeleme problemleri için melez sezgisel ve meta-

sezgisel algoritmalar hakkında yeterli çalışma bulunmadığı Şekil 1.1 ve Şekil 1.3'te gösterildiği üzere görülmüş, bu alanda ilerlemeye karar verilmiştir.

İkinci bölümde Çizelgeleme problemleri ve türleri hakkında genel bir özet yazılmıştır. Çizelgeleme problemlerinin formülasyonu, çözümlerin Gantt diyagramında gösterilmeleri, farklı çizelgeleme problemi türlerinin birbirlerinden farkları ve nasıl ifade edilebilecekleri hakkında bilgiler verilmiştir.

Üçüncü bölümde Çizelgeleme problemleri hakkında literatürde önerilen belli başlı sezgisel algoritmalar ile tezde geliştirilen melez çözümler anlatılmıştır. Karınca kolonisi optimizasyonu, Genetik algoritma ve Yapay bağışıklık sistemi bu algoritmalardan öne çıkan birkaçıdır. Karınca kolonisi algoritması, karıncaların yiyecek arama yöntemlerinden ilham alarak çözüm bulmaya çalışan bir arama sezgiselidir. Çizelgeleme problemlerine uyarlanışı da bir çizelgeyi bir karıncanın yiyecek bulmak için gezindiği rota şeklinde simgelemesi ile olmaktadır. Yiyeceğe daha hızlı ulaşan karıncalar, gezdikleri rota üzerine çekici bir koku bırakarak diğerlerinin benzer rotalar seçmelerine yardımcı olmaktadırlar. Aynı şekilde daha iyi çizelgeler de, daha sonra oluşturulacak çizelgelerin de kendilerine benzer olmasını sağlamak üzere çizge üzerinde izler bırakırlar. Genetik algoritma, biyolojik evrimden ilham alan bir algoritmadır. Popülasyonlardaki genetik özellik frekansının, o özelliğin avantajlı olup olmaması durumuna göre sonraki nesillerde azalması veya artması durumundan faydalanarak çözüm arayan bir sezgiseldir. Çözümleri birer birey olarak simgeleyip, istenilen yönde olan çözümlere daha fazla hayatta kalma şansı verip kötü bireylerin yaşama olasılığı azaltılarak daha iyi çözümler elde edilmeye çalışılır. Yapay bağışıklık sistemi ise Genetik algoritma ile benzerlikler taşır. Vücudun antijenlere karşı direnç gösterme yöntemlerinden ilham alır. Çözümler kromozom yerine antikor olarak simgelenirler. Belli başlı mutasyonlar uygulanarak, iyi çözümlerin klonlanması ile daha iyi çözümlere ulaşılmaya çalışılır. Tez çalışmasında ayrıca Genetik algoritma ile Karınca kolonisi optimizasyonu'nun bir melezi önerilmiştir. Tez çalışması çerçevesinde özgün olarak önerilen melez model, iki algoritmanın da çözüm iyileştirmesi sonlanana kadar çalıştırılıp elde edilen sonucun diğer algoritmaya aktarılıp iyileştirilmeye çalışılması şeklinde olmaktadır.

Dördüncü bölümde ise Çizelgeleme problemlerinde bulanık mantık yaklaşımları modellenmeye çalışılmıştır. İki farklı yaklaşım önerilmiştir. Birinci yaklaşım, makinaların

performanslarının bulanıklaştırılması temellidir. Makinaların performanslarının çalışma süresi boyunca %100 olmayacağını, bir yamuk bulanık sayı (TFN) şeklinde başta %0 iken belli bir süre sonra %100'e çıkacağını daha sonra da proses bitiminde tekrar %0'a ineceğini öngörmüştür. Böylece proses bitimi başlangıçta planlanan sürede olmayacaktır. İkinci yaklaşım ise makinaların doluluk miktarlarını bulanıklaştırma temellidir. Tez çalışması çerçevesinde önerilen bu yaklaşıma göre işler, daha ufak parçalara bölünebilecek paketler halinde alınıp işlenen parçalar olarak kabul edilmiştir. Böylece bir işin parçalarının makinaya yerleştirilme süresi farkı, işin makinada işlenmesini bir yamuk bulanık sayı haline getirir. Bu yaklaşıma göre işlerin çalışma sürelerinin toplanması, birbirleriyle kıyaslanmaları veya enbüyüklenmeleri klasik metodlara göre farklılık gösterecektir. Böylece ortaya çıkan yeni durumların çözümü için yeni formülasyonlar önerilmiştir.

Beşinci bölümde yazılan Çizelgeleme paket programı anlatılmıştır. Program, kullandığı kütüphaneler dışında tamamen tez çalışması çerçevesinde özgün olarak kodlanmıştır. Mobil platformlar ve İnternet tarayıcıları için ayrı olarak yazılmıştır. Literatürde bulunan belli başlı algoritmalar programa eklenmiştir. Programa hızlı ve kolay bir şekilde örnek Çizelgeleme problemi oluşturma modülü yazılmış, böylece oluşturulan problemler üzerinde algoritmaları test etme imkanı oluşturulmuştur. Ayrıca bulunan çözümlerin bir Gantt diyagramı üzerinde gösterilmesini sağlayan bir modül yazılmıştır. Meta-sezgisel algoritmaların buldukları sonuçlar ve harcadıkları zamanın daha doğru şekilde kıyaslanması için bir zamana bağlı çözüm grafik gösterimi önerilmiş, buna uygun modül yazılmıştır. Ayrıca algoritmaları sayısal olarak kıyaslamak için ortalama sapma yüzdesi, başarı oranı gibi kriterleri gösteren bir tablo modülü eklenmiştir.

Altıncı bölümde yazılan program kullanılarak elde edilen bazı deneysel sonuçlar aktarılmıştır. İlk olarak yazılan Genetik algoritma ve Karınca kolonisi optimizasyonu için en iyi parametre değerleri belirlenilmiştir. Sonra bu parametre değerleri kullanılarak bu iki algoritma karşılaştırılmış, hangi problem tiplerinde hangi algoritmaların daha iyi çalıştıkları ortaya çıkarılmıştır. Bu değerler algoritmaların melezini oluşturmak üzere kullanılmıştır. Daha sonra Yapay bağışıklık sistemi ve Genetik algoritma, yazılan grafik gösterimi ve Sayısal karşılaştırma değerleri kullanılarak kıyaslanmıştır.

Daha sonra bulunan sonuçlar değerlendirilmiş, sonraki çalışmalar için tavsiyeler yazılmıştır.

1.1 Literatür Özeti

Araujo ve Nagano (2011) [2] sıra bağımlı hazırlık zamanlı beklemesiz akış tipi problemdeki işleri çizelgelemeyi tamamlanma zamanını enküçüklemek amacıyla irdelemişlerdir. Bu problem NP-hard olması ile ünlüdür ve bu probleme küçük bir katkıda bulunmuşlardır. Çalışmalarında GAP Heuristics (Açıklık Sezgiseli) adında yapısal özellik tabanlı yeni yapıcı sezgisel bir yöntem önermişlerdir. Önerilen yaklaşımın, Bianco, Dell'Olmo ve Giordani'nin önerdiği twos [3] ve Brown, Mcgarvey ve Ventura'nın önerdiği sıra bağımlı hazırlık zamanlı amaca adapte edilmiş sezgisel TRIPS (Üçlü) [4] gibi literatürdeki iyi bilinen 3 yöntemden çözüm kalitesi ve gereken hesaplama zamanı bakımından daha üstün olduğunu söylemişlerdir.

Pugazhenthi ve Xavior (2014) ise [5] N adet işin M adet makinede çalıştığı akış tipi zamanlama problemlerine tamamlanma zamanını enküçüklemek birincil amacı ile yaklaşmışlardır. Akış tipi zamanlama problemini modern üretim çerçevesinde çözmek için EPDT (Extended Prim-Dijkstra Tradeoff) sezgisel ve BAT (Yarasa) adında meta sezgisel yaklaşımlar önermişlerdir. Bu iki algoritmayı en az tamamlanma zamanına ulaşma konusunda daha fazla gelişme için Genetik Algoritma (GA) ile beraber uygulamışlardır. Bu yeni sezgisel yaklaşımların performanslarını ölçmek için MATLAB'da farklı büyüklükteki Taillard kıyaslama problemlerini çözdürmüşlerdir. Akış tipi problemler için önerilen GA uygulamalı FPDT sezgisel yaklaşımı ile GA uygulamalı BAT meta-sezgisel yaklaşımının, çizelgeleme problemlerinin çözümünde ve tamamlanma zamanını azaltacak daha iyi bir dizi bulmada etkili olduğunu söylemişlerdir.

Allahverdi ve Aldowaisan (2002) [6] toplam tamamlanma süresini gözönünde bulunduran çok makinalı beklemesiz akış tipi problemler için birkaç yeni sezgisel algoritma sunmuşlardır ve bu yeni yaklaşımlarının yeni geliştirilen genetik algoritma da dahil olmak üzere bilinen 3 yaklaşımdan hata performansı bakımından daha iyi olduğunu göstermişlerdir.

Laha ve Sapkal (2014) [7] çalışmalarında beklemesiz akış tipi çizelgelemedeki toplam akış zamanını enküçüklemek için bir sezgisel algoritma önermişlerdir. Hesaplamalarda, önerilen sezgisel yaklaşım, zaman karmaşası dışında iyi bilinen sezgisellerden üstün gelmiştir. İstatistiksel önem testleri yöntemin üstünlüğünü kanıtlamıştır.

Aldowaisan (2001) [8] makalesinde hazırlık sürelerinin sıra bağımsız olduğu ve işlem zamanlarından ayrı düşünüldüğü iki makineli beklemesiz akış tipi problemleri toplam akış zamanını enküçüklemek maksadıyla irdelemiştir. Hesaplanan sonuçlar yeni sunulan sezgisel yaklaşımın ve geliştirilen lokal baskınlığın üstünlüğünü göstermiştir.

Zhu ve arkadaşları (2007) [9] makalelerinde yeni bir melez sezgisel algoritma önermişlerdir. Önerdikleri algoritmada NEH (Nawaz-Enscore-Ham) ek operatorü yerine kullanılan FL methoduyla değiştirilmiş PH1p algoritmasını temel almışlardır. Avantajlı başlangıç değerleri oluşturmak için bir birleşik algoritma uygulamışlardır. Simülasyon sonuçları, bazı var olan algoritmalarla iyi bilinen performans ölçme testleri ve karşılaştırmaları temel alarak önerilen melez algoritmanın etkililiğini göstermiştir.

Kumar ve Singhal (2013) [10] makalelerinde sıra bağımlı hazırlık zamanlı ve iş bölmeli M adet makine ve N adet iş içeren akış tipi çizelgeleme problemleri için genetik algoritma ile ideal çizelge sıralaması bulmak üzerinde çalışmışlardır. Geleneksel ve genel olmak üzere iki tip vaka çalışması üzerinde duran yazarlar, optimize edilmiş tamamlanma zamanı değerine bir yerine birden fazla farklı iş sıralamaları ile erişilebileceğini ve iş bölmenin çizelgeleme işleminde tamamlanma zamanını azaltmada yardımcı olabileceğini göstermişlerdir.

Tseng ve Lin (2010) [11] makalelerinde beklemesiz akış tipi çizelgeleme problemlerini tamamlanma zamanı hedefi ile çözmek için melez bir genetik algoritma sunmuşlardır. Bu sunulan algoritma, genetik algoritma ile yeni bir yerel arama şemasını birleştirmiştir. Kullanılan yerel arama algoritması Eklemeli Arama (Insertion Search) ile Kesme ve Tamir ile Eklemeli Arama (Insertion Search with Cut-and-Repair) algoritmalarını birleştirmiştir. Yeni algoritma Carlier (1978) [12] tarafından sağlanan performans testlerindeki bütün problemler için aynı en iyi çözümü bulmuştur. Bununla birlikte Taillard (1990) [13] tarafından sağlanan 120 problemi de etkili bir şekilde çözmeyi başarmıştır.

Chaundry ve Mahmood (2012) [14] ise genetik algoritma kullanılan beklemesiz akış tipi çizelgeleme geliştirmişlerdir. Beklemesiz akış tipi çizelgeleme, imalat sistemlerinde yaygınca bulunan kısıtlı bir akış tipi çizelgelemedir. Bu araştırmada, genel amaçlı tablo tabanlı genetik algoritma kullanılan M sayıda makinede işlenen N sayıda iş için toplam tamamlanma zamanı enküçüklenmesi dikkate alınmıştır. Önerilen yaklaşım çözümü

literatürde zaten yayınlanmış olan problemlerle karşılaştırılmıştır. Önerilen yaklaşım tüm durumlar için en uygun çözümü üretmektedir. Ayrıca, genetik algoritmanın genel mantığını değiştirmeksizin aynı modeli kullanarak *herhangi* bir objektif fonksiyonun enküçüklenebileceğini de göstermektedir.

Aldowaisan ve Allahverdi (1998) [15] makalelerinde bir işin hazırlık süresi ile proses süresinin ayrı olduğu iki makineli beklemesiz akış tipi problemi üzerinde çalışmışlardır. Dikkate alınan performans ölçümleri toplam akış süresidir. Bir eliminasyon ölçütü geliştirmişler ve iki özel durumdan optimal çözümler elde etmişlerdir. Genel durum için sezgisel algoritma sunmuşlardır. Hesaplamalar algoritmanın iyi çözümler sağladığını göstermiştir. Barto, Bradtke ve Singh (1995) [16] DP (dinamik programlama) temelli Gerçek Zamanlı DP şeklinde isimlendirdikleri performansını deneyimle kendi kendine artırabilen bir algoritma sunmuşlardır. Önerdikleri algoritma Korf'un Gerçek Zamanlı A* Öğrenme algoritmasını belirsizlik içeren problemlere uygulanmıştır. Asenkron DP teorisinin sonuçlarına dayanarak kendi algoritmalarının farklı sınıftaki problemlerde optimal davranışa ulaştığını göstermişlerdir.

Bruner ve arkadaşları (2012) [17] çalışmalarında P2P içerik ağlarında kaynak keşfi için örümcek ağı kümelemesine dayanan ağ uyumlu özetleme algoritması sunmuşlardır. Bulunan kaynakların kesinliği geliştirilirken keşif sürecinde özetlemenin nasıl iyileştirileceğini tanımlamışlardır. Hassas geri çağırmaya dayanan metrikler, özetlenen içerik üzerinde oluşturulan sorguların belirli türlerinin kesinliğiyle karşılaştırılmıştır.

Wieczoreka ve arkadaşları (2009) [18] çalışmalarında çok ölçütlü ızgara iş akışı çizelgeleme problemine farklı yeni taksonomiler önermişlerdir ve bunu 5 yönlü incelemişlerdir: İş akış modeli, çizelgeleme ölçütü, çizelgeleme prosesi, kaynak modeli ve görev modeli şeklinde. Mevcut ilgili çalışmaları incelemişler ve önerilen taksonomilere göre sınıflandırmışlardır.

Cuevas-Tello ve arkadaşları (2010) [19] temsil eden gecikme ve aynı temel modelin düzensiz örneklenmiş ve gürültülü değişkenleri arasındaki zaman gecikmesindeki tahmin etme problemleri üzerine çalışmışlardır. Astronomik problemin içeriğinde çekirdek tabanlı tekniğin hiper parametre tahmini için evrimsel bir algoritma önermişler ve uygulamışlardır. Özetle uzak kuasardan ışığı saptıran iki kütleçekimsel merceklenmiş sinyal arasındaki zaman gecikmesindeki tahmin çalışılmıştır. Bu yöntem

daha doğru ve istikrarlı zaman gecikme tahminlerini sağlamıştır. Astronomide şu anki gelişmiş optik gözlem verisi için ve aynı zamanda benzer zaman serisi verilerini de içeren diğer disiplinlerde de bu yöntembilimin kolayca uygulanabileceğini göstermişlerdir.

Kashan ve arkadaşları (2009) [20] paralel makinelerdeki çizelgeleme problemi için Parçacık Sürü Optimizasyonu (PSO) algoritmasını kullanan ilk uygulamayı makalelerinde incelemişlerdir. Klasik PSO denklemleri ile kıyaslanabilecek düzeyde denklemler öne sürerek tamamlanma zamanını enküçüklemek için ayrık PSO (DPSO) algoritması önermişlerdir. Bunun yanında bu DPSO algoritmasının etkinliğini yerel arama sezgisel yaklaşımı ile hibritleştirerek incelemişlerdir. Bilgisayarlı hesaplamalar kullanılarak, önerilen DPSO algoritmasının gayet rekabetçi ve yerel arama sezgisel yaklaşımları ile hibritleştirilirken hızlı bir şekilde modellenebilir olduğunu göstermişlerdir.

Prot ve arkadaşları (2012) [21] makalelerinde endüstriyel bir atölye çizelgeleme problemini, çok modlu üretim hatlı bir atölye tipi olarak modellemişlerdir. Ugraştıkları problemde işlerin sıra-bağımlı hazırlık süreleri ve teslim tarihi gibi ek kısıtlar vardır. Problemde karar vericilerin amacı ise en büyük gecikmenin enküçüklenmesidir. Problemi çözmek için bir tabu arama prosedürü ile bu tabu arama prosedürünü ölçmek için geçerli bir alt sınır tanıtmışlardır.

Vidal ve arkadaşları (2014) [22] makalelerinde araç rota problemleri için verimli, uyguanabilir ve genel amaçlı bir algoritmanın geliştirilmesine ve bu alandaki zorlukların belirlenmesine bileşen tabanlı bir sezgisel yaklaşımla katkıda bulunmuşlardır. Geniş hesaplamalı deneyler sonucunda metodun literatürdeki en başarılı problem odaklı algoritmalar kadar veya onlardan daha iyi olduğunu dikkate değer bir performansını işaret ederek göstermişlerdir.

Pacini ve arkadaşları (2014) [23] çalışmalarında biyo-ilham kaynaklı tekniklerle Parametre Tarama Deneyleri (PSE) için dağıtılmış iş çizelgeleme adresleme çabalarını araştırmışlardır. Araştırılan materyallerin organize edilmesi ve analizi ve için bir taksonomi türetmişlerdir. Mevcut denemelerin güçlü ve zayıf yönlerini belirtmişlerdir. Bu alanda gelecekte yapılabilecek çalışmaları betimlemişlerdir.

Zandieh ve arkadaşları (2006) [24] çalışmalarında, genellikle sıra bağımlı hazırlık zamanlı (SDST) hibrit akış tipi çizelgeleme problemleriyle ilgilenmişlerdir. SDST hibrit akış tipi çizelgeleme için bir bağışıklık sistemi algoritması yaklaşımı açıklamışlardır. Hibrit akış tiplerini ve bağışık algoritmanın temel kavramlarını incelemişlerdir. Bağışıklık sistemi algoritmasının detaylarını açıklamış ve uygulamışlardır. Sonuçlar rastgele anahtarlı genetik algoritma (RKGA) ile karşılaştırılmış ve bağışıklık sistemi algoritması RKGA'dan üstün gelmiştir.

Klöckner ve arkadaşları (2012) [25] çalışmalarında grafik işlem birimi (GPU) çalışma zamanlı kod üretimini (RTCG) açıklamışlar ve bunun yararlılığını tartışmışlardır. Ayrıca GPU RTCG tekniğini destekleyen iki açık kaynaklı araç sunmuşlardır. Tekniğin başarılı bir şekilde uygulandığı örnekler gösterilmiştir.

Kuo ve Lin (2010) [26] çalışmalarında yüzey montaj teknolojisi (SMT) kurulum süresini azaltmak için genetik algoritmanın hibritine dayalı evrimsel tabanlı kümeleme algoritması ve parçacık sürü optimizasyonu algoritması (PSO) önermişlerdir. Sipariş kümeleme, aynı çizelgeye ait çizelgeleme sırası ile birlikte makinenin boşta kalma süresinin yanı sıra üretim süresini de azaltabileceğini ortaya koymuşlardır.

Burdett ve Kozan (2014) [27] çalışmalarında tren zamanlamaları oluşturma problemini ele almışlardır. Tren zaman tablosu oluşturma gecikmeler ve aksamalar açısından karmaşık bir problemdir. Etkilenen operasyonlar açısından gecikme etkisi tanımlamak için sayısal olarak verimli algoritmalar geliştirmişlerdir. Etkilenen araştırmaların ayarlamaları ve gecikme değerleri tren araştırmalarının ayırıcı grafik modeli ile yayılmıştır. Önerilen duyarlılık analizi sonuçları program sağlamlığını belirlemek için kullanılmıştır. Analizler proaktif çizelgeleme yaklaşımının bir parçası olarak kullanılabilen bilgiler sunmuştur. Etkilenen işlemler çizelge refinement için meta sezgisel yaklaşımları geliştirmede kullanılabilir.

Malczewski (2004) [28] çalışmasında 3 temel amacı vardır: planlamadaki Coğrafi Bilgi Sistemlerinin (GIS) gelişen rolü üzerine tarihsel bir perspektif ile birlikte coğrafi bilgi teknolojisine bir giriş sunmak, GIS tabanlı arazi kullanım elverişliliği haritalama ve modelleme için uygun metodlar ve teknikleri gözden geçirmek ve GIS tabanlı arazi kullanım elverişlilik analizinin eğilimlerini, zorluklarını ve olasılıklarını tanımlamak. Makalesinde GIS tabanlı arazi kullanım uygunluğu analizinin iki perspektifine

odaklanmıştır: tekno pozitivist perspektif ve sosya politik halk katılımlı perspektifler. GIS tabanlı arazı elverişlilik analizinin tartışılan problemlerini ve olasılıklarını açıklamıştır.

Mellit ve Kalogırou (2014) [29] çalışmalarında maksimum güç noktası izlemek için yapay zeka temelli metodların uygulamalarını gözden geçirmişlerdir. Çalışılan metodlar sinirsel ağlara, bulanık mantığa, genetik algoritma içeren evrimsel algoritmalara, parçacık sürüsü optimizasyonuna, karınca kolonisi optimizasyonuna ve diğer hibrit metodlara dayandırılmıştır. Programlanabilir mantık cihazlarındaki hızlı gelişmeler, dijital maksimum güç noktası izleme için fırsatlar sunmuştur. Bu alanda çalışan ve mühendisler, tasarımcılar ve bilim adamları için değerli bilgiler sunmuşlardır. Çalışmalarıyla yenilenebilir enerji sistemleri için gömülü zeka tekniklerinin gelişimindeki gelecek trendleri göstermişlerdir.

Shahabudeen ve Sivakumar (2008) [30] çalışmalarında genetik algoritma ve benzetimli tavlama tabanlı sezgısel geliştirmişler ve adaptif kanban sistemin tasarım parametrelerini ayarlamak için kullanmışlardır.

Dahmani ve arkadaşları (2013) [31] ise tek amaçlı 2 boyutlu vektör paketleme problemi (Mo2-DBPP) üzerine çalışmışlardır. Bu problemin çözümü için iki iteratif çözüm yaklaşımı önermişlerdir. Her iki yaklaşımda birkaç alt sınır, sezgiseller ve meta sezgiseller kullanmışlardır. İki yaklaşımın etkinliğini karşılaştırmak için literatürden alınan kriterler üzerinde hesaplamalı deneyler uygulanmıştır.

Sherali ve Driscoll (2000) [32] çalışmalarında tamsayılı lineer programlamadaki çokyüzlü yapılar ve cebirsel tekniklerin evrimini, bu tarz problemlerin çözümlerinde gelinen en son noktayı tartışmışlardır.

Barthélemy ve arkadaşları (2002) [33] çalışmalarında karar destek sistemlerinde insan etmeninin rolü ve yöneylem araştırmasında kullanılan ilişkili yardım elemanları üzerinde durmuşlardır. Amaçları, bilgi işlemlerinin son değişimi dikkate alındığında insan merkezli sistemler kapsamında yeni sorunları çözmek için kullanılabilen bâzı araçları (örneğin; fayda teorisi, bilişsel modelleme vs.) gözden geçirmek olmuştur.

Strang (2012) [34] ise Avustralya'daki bir madencilik şirketinde kamyon kuyruğu modeli deneysel durumu üzerine çalışmıştır. Özel amaçlı bir madencilik simülasyon-planlama yazılımı gizli varsayımlarla kullanılmıştır. Parametrik olmayan hipotez test teknikleri veri

dağılımlarını doğrulamak için uygulandı. Kolay kullanımlı elektronik tablo yazılımı, kuyruk modelleri için varsayımları doğrulamada kullanılmıştır.

Burgess ve Lefley (2001) [35] çalışmalarında, kullanım kolaylığı ve kesinliği açısından daha önce yayınlanan araştırmalarla karşılaştırıldığında, yazılım efor tahmininde genetik programlamanın potansiyelini değerlendirmeyi amaçlamışlardır.

Li ve arkadaşları (2011) [36] makalelerınde iki farklı veri madencilik tekniğini araştırmışlardır. Bu teknikler yapay sinir ağları ve ikili lojistik regresyon yöntemleridir. Sınav çizelgeleme problemleri için önerilen grafik tabanlı hiper-sezgisel tarafından oluşturulan çözümler üzerinde yaklaşımlarını değerlendirmişlerdir. Zaman karmaşıklığı analizi, yapay sinir ağları ve ikili lojistik regresyon yönteminin çalışmayı hızlandırdığını göstermiştir. Çalışmalarıyla daha gelişmiş bilgi-temelli karar destek sistemlerinin geliştirilmesine yardımcı olmuşlardır.

Dey ve arkadaşları (2014) [37] çalışmalarında çok seviyeli eşiklemeyi daha hızlı yapabilme amaçlı meta sezgiseller önermişlerdir. Altı farklı kuantum esinli meta sezgisel yöntemi önermek için kuantum mekanizması kullanmışlardır. Önerilen altı kuantum meta sezgisel yöntemin sonuçları konsensus sonuçları oluşturmak için ele alınmıştır. Friedman istatistiksel testi uygulanarak bu yöntemlerden üstün olan yöntem aranmıştır. Kuantum ilhamlı parçacık sürü optimizasyonu diğer yöntemlerden üstün gelmiştir. Önerilen yöntemlerin hesaplama karmaşıklıkları, bu yöntemlerin zaman aşımı verimliliği bulunması için açıklanmıştır.

Nahas ve arkadaşları (2007) [38] çalışmalarında iyi bilinen bir NP-hard problemi olan fazlalık paylaştırma probleminin (Redundancy Allocation Problem) çözümü için etkili bir algoritma sunmuşlardır. Sezgisel yaklaşım tasarımı düşüncelerinde, karınca kolonisi meta sezgisel optimizasyon yönteminden ve indirgenmiş tavan (Degraded Ceiling) yerel arama tekniğinden esinlenmişlerdir. İndirgenmiş tavan ile karınca kolonisi meta sezgiselinde yaptıkları melezleme iyi işlemiş ve fazlalık paylaştırmanın iyi bilinen sezgiseliyle rekabet edebilmiştir. Daha önceki araştırmalardan 33 test probleminin sayısal sonuçları rapor edilmiş ve karşılaştırılmıştır. Çalışmalarında önerdikleri yaklaşımla bulunan çözümlerin bilinen en iyi çözümlerden daha iyi olduğunu göstermişlerdir.

Kianfara ve arkadaşları (2012) [39] makalelerinde işlerin belirli olmayan dinamik gelişine ve sıra bağımlı hazırlık süresine dayanan esnek akış tipi sistemi üzerinde çalışmışlardır. Problemin amacı işlerin ortalama gecikmesini enküçükleyen bir çizelge belirlemektir. Problemin sınıfı NP-zor olduğundan yeni sevkiyat kuralı ve melez genetik algoritma geliştirilmiştir. Araştırmalarında sundukları iki yeni yöntem ve literatürde en çok kullanılan sevkiyat kuralları simülasyon modeli içinde birleştirilmiştir. Sonuçlar, çalışmalarında önerdikleri yöntemlerin geleneksel sevkiyat kurallarından daha iyi olduğunu göstermiştir.

Liu ve arkadaşları (2011) [40] makalelerinde atölye tipi planlama ve çizelgeleme problemi için adaptif tavlama genetik algoritma (Adaptive Annealing Genetic Algorithm - AAGA) üzerine çalışmışlardır. AAGA ile; tek parca, küçük grup veya özel üretimler için atölye tipi planlama ve çizelgeleme problemi üzerine uğraşmayı önermişlerdir. AAGA, adaptif mutasyon olasılığı genetik algoritmanın yakınsama hızını geliştirmiş ve yerel optimizasyonu önlemiştir. Boltzmann olasılık seçim mekanizması, erken gelişme oranını ve yerel yakınsama problemlerini çözebilmiş ve eşleşme bireyleri seçmek için uygulanmıştır. Sonuç olarak, AAGA tabanlı atölye tipi planlama ve çizelgeleme problemi tartışılmış ve Genetik Algoritma ile AAGA işlem sonuçları gösterilmiş, karşılaştırılmıştır.

Jolai ve arkadaşları (2013) [41] çalışmalarında beklemesiz, iki aşamalı, esnek akış tipi iki amaçlı çizelgeleme problemi üzerine yogunlaşmışlardır. Çalışmalarında gözönünde bulundurulan amaçlar, en az tamamlanma zamanı ve işlerin maksimum gecikmesi olmuştur. Bu problem NP-zor olarak bilinmektedir. İki amaçlı optimizasyonun tavlama benzetimine dayandırılan üç yöntemi CWSA (klasik ağırlıklı tavlama benzetimi), NWSA (normalize ağırlıklı tavlama benzetimi) ve FSA (bulanık tavlama benzetimi) olarak adlandırılır. Çalışmalarında Taguchi yöntemi ile çok amaçlı karar verme yaklaşımını karıştırarak yeni güvenilir bir yöntem önerilmiştir. Algoritmalarını küçük ve büyük ölçekli problemler üzerinde değerlendirmişlerdir. Performanslarını bağıl sapma endeksi açısından değerlendirmişlerdir. Çalışmanın sonucunda daha fazla çalışmanın potansiyel alanları vurgulanmıştır.

Sloman ve Chrisley (2005) [42] çalışmalarında fiziksel olguları yansıtan birinci derece, fiziksel olguların algılayıcılarını karakterize eden ikinci derece ontolojiler ile rekürsif

şekilde algılayıcıların algılayıcılarını karakterize eden üçüncü derece ontolojilerin sanal makinaların içeriği ile ilgisi üzerinde durmuşlardır. Hayvanlar ve robotlar gibi sanal ve fiziksel bileşik makinelerin bilimsel araştırmalar için gerekliliklerini incelemişlerdir. Reaktif, müzakereci ve meta-yönetim kategorileri birleştiren CogAff mimari şeması ile CogAff'in özel bir durumu olan ve insan benzeri sistemler için minimal mimari belirtme olarak kabul edilen H-CogAff'in kullanımlarını örneklendirmişlerdir.

Çizelge 1.1 Literatür taraması

YIL	BAŞLIK	YAZAR	YÖNTEM	PROBLEM
1995	Learning to act using real-time dynamic programming	Andrew G. Barto, Steven J. Bradtke, Satinder P. Singh	Gerçek Zamanlı Dinamik Programlama	gömülü sistemlerin performanslarını artırmak
2012	Network-aware summarisation for resource discovery in P2P-content networks	René Brunner, Agustín C. Caminero, Omer F. Rana, Felix Freitag, Leandro Navarro	Örümcek Ağı Tabanlı Özet Ağacı	büyük boyutlu peer-to-peer ağlarda içerik bulmak
2009	Towards a general model of the multi-criteria workflow scheduling on the grid	Marek Wieczorek, Andreas Hoheisel, Radu Prodan	Yeni bir Taksonomi	çok ölçütlü iş akış çizelgeleme
2010	Uncovering delayed patterns in noisy and irregularly sampled time series: An astronomy application	Juan C. Cuevas-Tello, Peter Tino, Somak Raychaudhury, Xin Yao, Markus Harva	Evrimsel Algoritma	iki sinyal arasındaki zaman gecikmesi tahmini
2009	A discrete particle swarm optimization algorithm for scheduling parallel machines	Ali Husseinzadeh Kashan, Behrooz Karimi	Ayrik PSO	paralel makine çizelgeleme
2012	Tabu search and lower bound for an industrial complex shop scheduling problem	D. Prot, O. Bellenguez-Morineau	Tabu Arama	endüstriyel kompleks atölye çizelgeleme

Çizelge 1.1 Literatür taraması (devamı)

2014	A unified solution framework for multi-attribute vehicle routing problems	Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Christian Prins	Birleşik Melez Genetik Arama	çok özellikli araç yönlendirme
2014	Distributed job scheduling based on Swarm Intelligence: A survey	Elina Pacini, Cristian Mateos, Carlos García Garino	Parçacık Zekâ	paylaştırılmış iş çizelgeleme
2006	An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times	M. Zandieh, S.M.T. Fatemi Ghomi, S.M. Moattar Husseini	Bağışıklık Algoritması	hibrit akış tipi çizelgeleme
2012	PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation	Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, Ahmed Fasih	PyCUDA ve PyOpenCL	GPU runtime kod üretimi
2010	Application of a hybrid of genetic algorithm and particle swarm optimization algorithm for order clustering	R.J. Kuo, L.M. Lin	melez GA + PSO	sıralı kümeleme
2014	Determining operations affected by delay in predictive train timetables	Robert Burdett, Erhan Kozan	duyarlılık analizi	train timetable gecikmelerinden etkilenen işlemleri belirlemek
2004	GIS-based land-use suitability analysis: a critical overview	Jacek Malczewski	-	CBS tabanlı arazi kullanımı uygunluk analizi

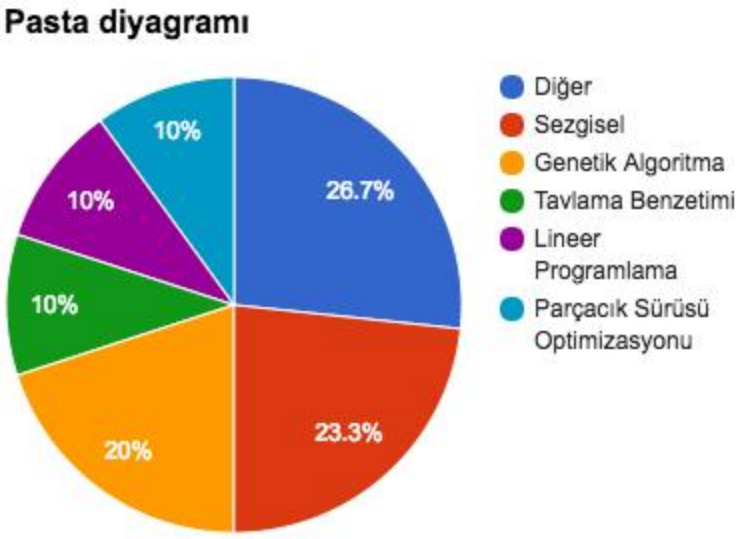
Çizelge 1.1 Literatür taraması (devamı)

2014	MPPT-based artificial intelligence techniques for photovoltaic systems and its implementation into field programmable gate array chips: Review of current status and future perspectives	Adel Mellit, Soteris A. Kalogirou	MPPT tabanlı AI teknikleri	fotovoltaik sistemlerin FPGA uygulaması
2008	Algorithm for the design of single-stage adaptive kanban system	P. Shahabudeen, G.D. Sivakumar	GA + tavlama benzetimli sezgisel	tek yerleşimli adaptif kanban sistemi
2013	Iterative approaches for solving a multi-objective 2-dimensional vector packing problem	Nadia Dahmani, François Clautiaux, Saoussen Krichen, El-Ghazali Talbi	iteratif sezgiseller, meta sezgiseller	çok amaçlı iki boyutlu vektör paketleme sorunu
2000	Evolution and state-of-the-art in integer programming	Hanif D. Sherali, Patrick J. Driscoll	tamsayılı doğrusal programlama	-
2002	Human centered processes and decision support systems	J.P. Barthelemy, R. Bisdorff, G. Coppin	yöneylem araştırması	karar destek sistemleri
2012	Importance of verifying queue model assumptions before planning with simulation software	Kenneth David Strang	-	Kuyruk model tahminlerini dogrulama
2001	Can genetic programming improve software effort estimation? A comparative evaluation	Jingpeng Li, Edmund K. Burke, Rong Zu	Genetik Programlama	yazılım çaba tahmini
2011	Integrating neural networks and logistic regression to underpin hyper-heuristic search	Jingpeng Li *, Edmund K. Burke, Rong Qu	sinir ağı + lojistik regresyon	hiper-sezgisel arama

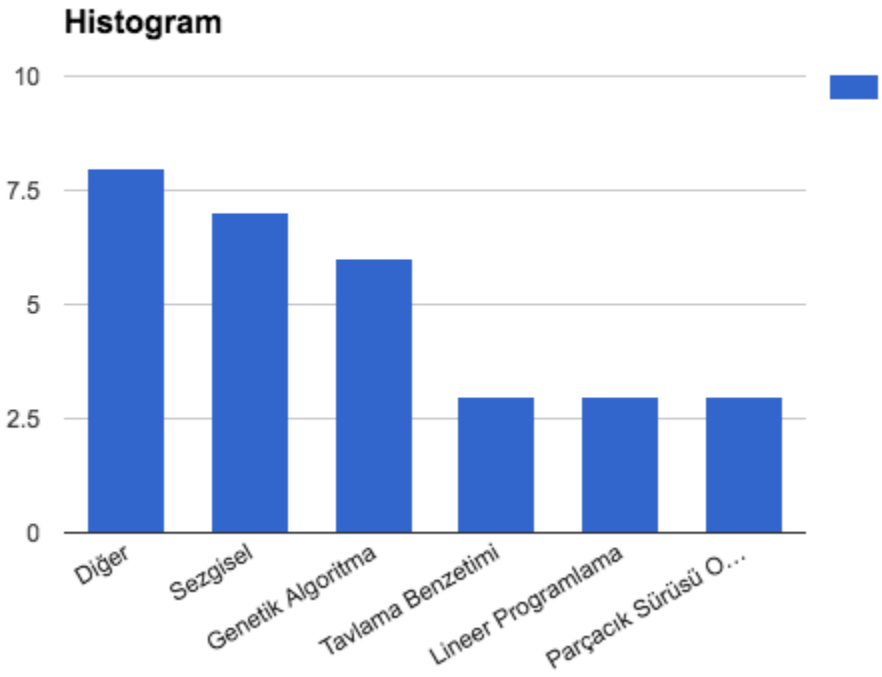
Çizelge 1.1 Literatür taraması (devamı)

2014	Multi-level thresholding using quantum inspired meta-heuristics	Sandip Dey, Indrajit Saha, Siddhartha Bhattacharyya, Ujjwal Maulik	kuantum ilhamlı meta sezgiseller	çok düzeyli eşikleme
2007	Coupling ant colony and the degraded ceiling algorithm for the redundancy allocation problem of series–parallel systems	Nabil Nahas, Mustapha Nourelfath, Daoud Ait-Kadi	karınca kolonisi + bozulmuş tavan algoritması	seri-paralel sistemlerde fazlalık tahsisi
2012	Study of stochastic sequence-dependent flexible flow shop via developing a dispatching rule and a hybrid GA	K. Kianfar, S.M.T. Fatemi Ghomi, A. Oroojlooy Jadid	melez GA + sevk kuralı	stotastik sıra bağımlı esnek akış tipi
2011	An adaptive annealing genetic algorithm for the job-shop planning and scheduling problem	Min Liu, Zhi-jiang Sun, Jun-wei Yan, Jing-song Kang	adaptif tavlama genetik algoritması	atölye tipi planlama ve çizelgeleme
2013	Bi-objective simulated annealing approaches for no-wait two-stage flexible flow shop scheduling problem	F. Jolai, H. Asefi, M. Rabiee, P. Ramezani	tek amaçlı benzetim tavlama	beklemesiz iki adımlı esnek akış tipi çizelgeleme
2005	More things than are dreamt of in your biology: Information-processing in biologically inspired robots	Ron Sun A. Sloman, R.L. Chrisley	cogAff	biyolojik ilhamlı robotlarla bilgi işleme

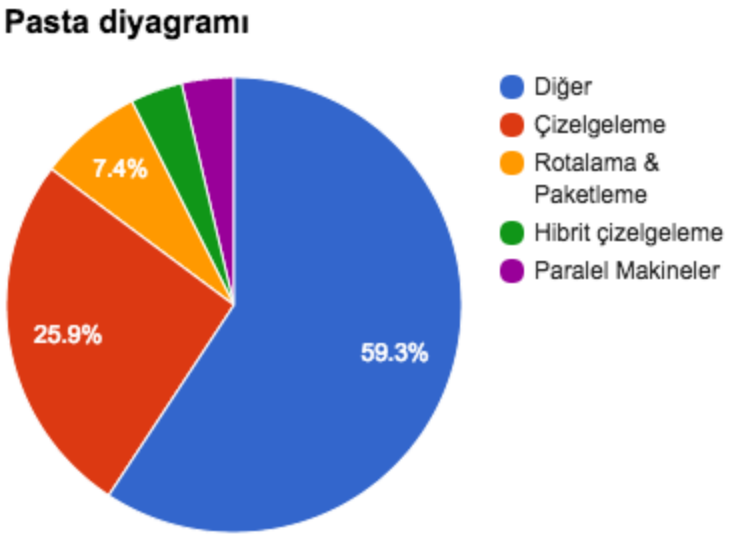
Şekil 1.1 ve Şekil 1.2'de yapılan literatür araştırmasında problemlerin çözümü için kullanılan algoritmaların pasta diyagramı ve histogramı görülmektedir. Şekil 1.3 ve Şekil 1.4'te ise literatürdeki problemlerin kullanım alanlarının pasta diyagramı ve histogramı görülmektedir. Bu grafiklerden Melez sezgisel algoritmaların literatürde yaygın olarak kullanılmadığı anlaşılmaktadır. Bu sebepten tezde bu yönde çalışılmaya devam edilmesine karar verilmiştir.



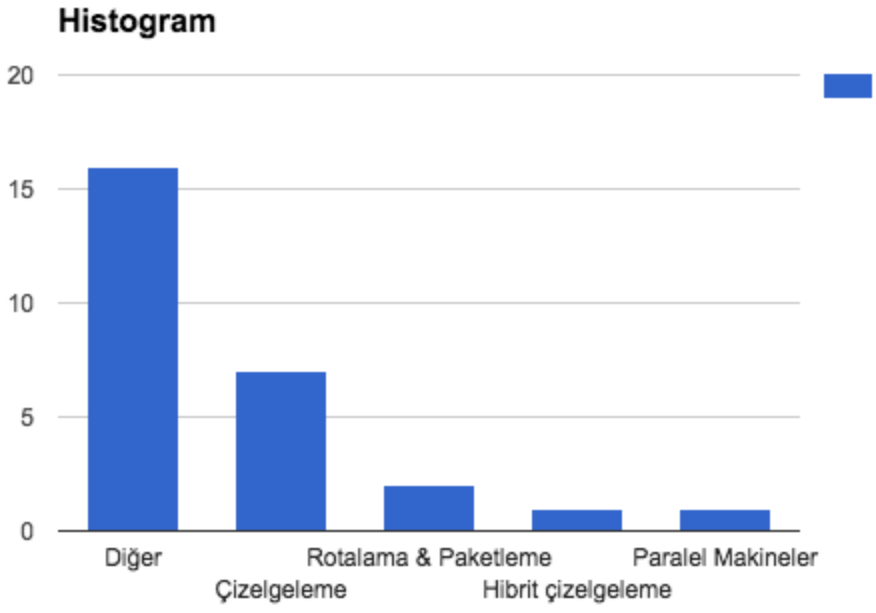
Şekil 1.1 Literatürde kullanılan algoritmaların pasta diyagramı



Şekil 1.2 Literatürde kullanılan algoritmaların histogramı



Şekil 1.3 Literatürde uğraşılan problemlerin pasta dağılımı



Şekil 1.4 Literatürde uğraşılan problemlerin histogramı

1.2 Tezin Amacı

Tezin amaçları aşağıdaki şekilde belirlenmiştir:

- İstenen özelliklerde rastgele bir çizelgeleme problemi oluşturabilecek, oluşturulan problemi yazılan algoritmaların aynı formatta okuyup, vereceği çıktıları Gantt diyagramı, grafik gösterimi ve çeşitli analiz yöntemları ile görselleştirebilecek bir platform oluşturmak.
- Literatürdeki belli başlı çizelgeleme algoritmalarını oluşturulan bu platform üzerinde kodlamak, kullanılır duruma getirmek.
- Kodlanan algoritmaların parametrelerini, birbirlerine üstünlüklerini test etmek.
- Melez sezgisel çözümlerin geliştirilmesi üzerine çalışmak.
- Bulanık mantık yaklaşımının çizelgeleme problemleri üzerinde uygulanabilirliğini araştırmak, geliştirilen yöntemleri test etmek.

1.3 Hipotez

Tezin ilk bölümünde literatürde farklı tipteki Çizelgeleme problemlerinin çözümünde kullanılan çeşitli algoritmalar bir araya getirilmiştir. Çizelgeleme problemleri üzerinde çalışmayı kolaylaştırmak ve ilerideki çalışmalara da yardımcı olmak amacıyla kullanıcıların istedikleri tipte Çizelgeleme problemi oluşturup düzenleyebileceği açık kaynak koldu bir platform yazılmıştır. Daha sonra literatürde geçen çeşitli algoritmalar bu platformda çalışacak şekilde kodlanmış, sisteme bütünleştirilmiştir.

Oluşturulan program üzerinden algoritmaların parametrelerinin çalışma performanslarına etkileri incelenmiş, Karınca Kolonisi Optimizasyonu ile Genetik Algoritma kullanılarak melez bir çözüm geliştirilmiş, performansı test edilmiştir.

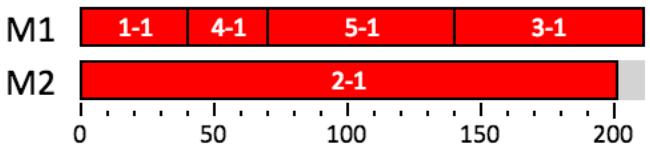
Daha sonra Bulanık mantık yaklaşımının Çizelgeleme problemleri üzerinde nasıl uygulanabileceği araştırılmış, birden fazla yaklaşım geliştirilmiştir. Geliştirilen yaklaşımların çözüme nasıl etki edeceği test edilmiştir.

BÖLÜM 2

PROSES PLANLAMA VE ÇİZELGELEME

Proses planlama ve çizelgeleme üretim sistemlerinin en önemli parçalarındandır. Proses planı bir ürün üretebilmek için gerekli hammaddeleri, bu hammaddelerin son ürün haline getirilmesi için gerekli operasyonları ve prosesleri içerir. Proses planının çıktısı makinaların tanımlanması, işlerin operasyonlarının ayarlanması gibi bölümlerden oluşur. Yani proses planlaması, ürün dizaynı ile üretimi arasındaki köprüdür denilebilir. Çizelgeleme ise proses planını girdi olarak alıp öncelik kısıtlarına uymak şartı ile bütün operasyonları makinalara yerleştirmekten sorumlu kısımdır. Çizelgeleme ise proseslerin planlanması ile aksiyona geçirilmesi arasındaki köprüdür denilebilir. Proses planlama ve çizelgeleme yakın ilişkili görünse de bunların entegrasyonu halen araştırma ve uygulama alanlarında önemli bir konudur. [43]

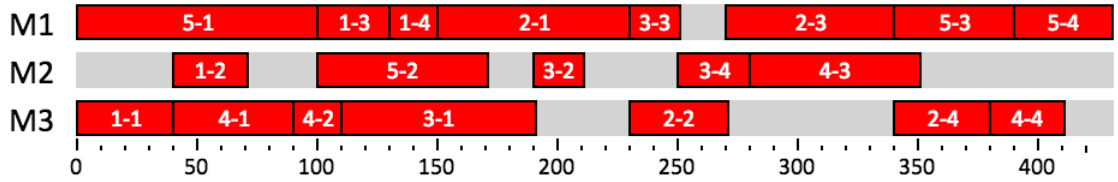
Çizelgeleme problemlerinin çeşitli tipleri vardır. Örneğin yapılması gereken işlerin belli bir öncelik sıralamasına bağlı operasyonları mevcut değilse tek aşamalı makine çizelgeleme, eğer işlerin öncelik sıralamalı operasyonları mevcutsa çok aşamalı makine çizelgeleme problemleri olarak adlandırılmaktadırlar. [44]



Şekil 2.1 İki makinalı tek aşamalı bir çizelgeleme problemi

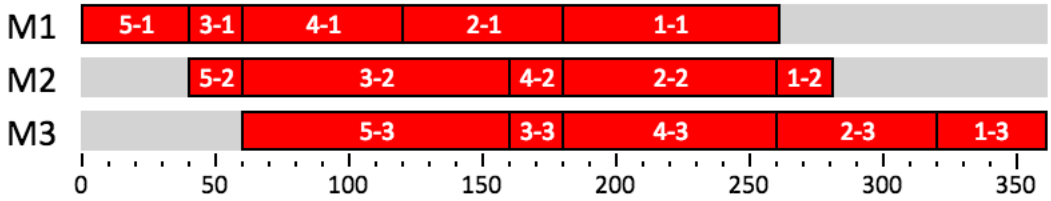
Çizelgeleme problemleri için geliştirilen çözümler Gantt diyagramı denilen bir gösterim ile görsel hale getirilebilirler. Bu gösterim yatayda zaman, dikeyde ise makinalardan oluşan bir şemadır. Her makinanın hangi zaman diliminde hangi iş ile yükleneceği bu diyagramdan okunabilir. Şekil 2.1'de böyle bir diyagram görülmektedir.

Tek aşamalı çizelgeleme problemleri aynı zamanda makina sayısına göre de tek makineli veya paralel makinalı çizelgeleme problemleri şeklinde sınıflara ayrılmaktadırlar. Örneğin Şekil 2.1'de görülen problem ayrıca paralel makinalı bir çizelgeleme problemidir. Paralel makinalı problemler aynı zamanda makinaların özdeş olup olmamasına göre de farklı sınıflara ayrılabilirler.



Şekil 2.2 Atölye tipi çizelgeleme problemi

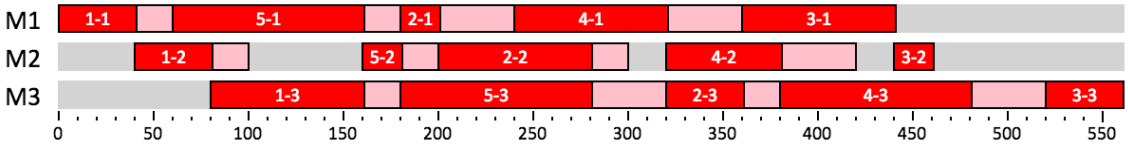
Çok aşamalı çizelgeleme problemleri de üretim tesisinin dizaynına göre farklı tiplerde olabilir. Bunlardan en bilinenleri akış tipi çizelgeleme ve atölye (sipâriş) tipi çizelgeleme problemleridir. Şekil 2.2 ve Şekil 2.3'te bunlara örnekler görülmektedir.



Şekil 2.3 Akış tipi çizelgeleme problemi

Çizelgeleme problemlerinin; problem tipi, ölçeği, performans ölçütü gibi özelliklerine göre üç alanlı notasyon adı verilen bir gösterim yöntemi vardır. Bu notasyona göre problem $\alpha|\beta|\gamma$ şeklinde üç değişkenle ifade edilebilir. α değişkeni makine konfigürasyonunu, β değişkeni varsa proses kısıtlamalarını, γ değişkeni ise

enküçüklenmeye çalışan amaç fonksiyonunu sembolize eder. Örneğin $1|S_{ij}|C_{max}$ ifadesi tek makineli toplam tamamlanma zamanını enküçükleyen sıra bağımlı hazırlık süresine sahip bir problemi gösterir. Örneğin Şekil 2.4'te 3 makineli sıra bağımlı hazırlık zamanlı akış tipi bir çizelgeleme problemi görülmektedir. [45]



Şekil 2.4 Sıra bağımlı hazırlık zamanlı bir akış tipi çizelgeleme problemi

BÖLÜM 3

ÇİZELGELEME ALGORİTMALARI

3.1 Karınca Kolonisi Optimizasyonu

Bu algoritmanın temel prensipleri C.W.Leung ve diğerlerinin [46] makalesinden alınmıştır.

Karınca kolonisi algoritması karıncaların yiyecek arama yöntemlerinden ilham almıştır. Karıncalar bir çizge (graph) üzerinde yiyecek araştırırlar. Yiyecek bulmanın anlamı çizelgeleme problemindeki bütün işlerin tamamlanmasıdır. Yiyeceğe ulaşan karıncalar geçtikleri güzergâhlara diğer karıncaların ilgisini çekecek bir koku (feromon) bırakırlar. Daha iyi yolları tercih eden karıncalar yiyeceğe daha hızlı ulaşırlar, bu da avantajlı yollarda daha fazla feromon birikmesine yol açar.

Bu algoritmada her iterasyonda, en kısa tamamlanma zamanlı (C_{max}) çizelgeyi oluşturan karınca eğer ki son sıfırlamadan sonraki en iyi çözümden daha iyi bir çözüm bulmuşsa feromon bırakır. Bu daha sonraki iterasyonlarda karıncaların daha kısa yolları takip etmelerini sağlar. Eğer karıncalar fazla feromon birikmesinden ötürü yerel en az değerde sıkışırlarsa algoritma feromon izlerini sıfırlayıp iterasyonları yeniden başlatıyor.

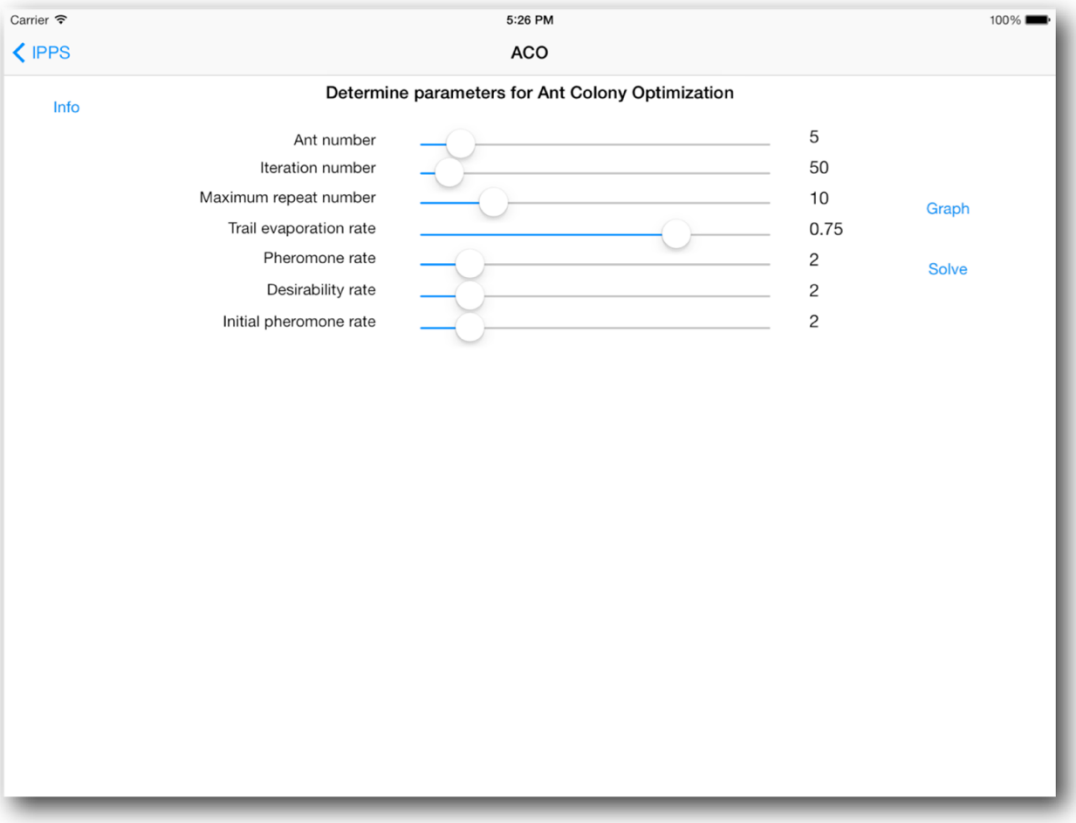
Karınca kolonisi algoritmasının, verimliliğini etkileyen bazı parametreleri vardır. Şekil 3.1'de bu parametrelerin belirlenme ekranını görmekteyiz.

Karınca sayısı: Ne kadar karıncanın çizgede bir çözüm arayacağını belirler. Karınca sayısı arttıkça daha iyi çözümler oluşturulur ancak daha fazla zaman kaybı ortaya çıkar. 1-50 arası değer alabilir.

İterasyon sayısı: Karıncaların kaç sefer çözüm arayacağının parametresi. Aynı şekilde iterasyon sayısı arttıkça çözümler iyileşir fakat zaman kaybı oluşur. 1-1000 arası tamsayı değer alabilir.

En fazla yineleme sayısı: Algoritmanın daha iyi bir çözümü kaç iterasyon boyunca bulamaması durumunda feromon izlerini sıfırlayacağını belirleyen parametre. 1-50 arası tamsayı değer alabilir. Küçültülmesi durumunda yerel en iyi değerde sıkışma süresi kısalmış olur.

İz buharlaşma oranı: Her iterasyon sonunda haritadaki feromon miktarları bu parametreye göre bir oranda buharlaştırılır. 0-1 arası değer alabilir. Yükseltilmesi durumunda yerel en iyi değerde sıkışma olasılığı azalabilir.



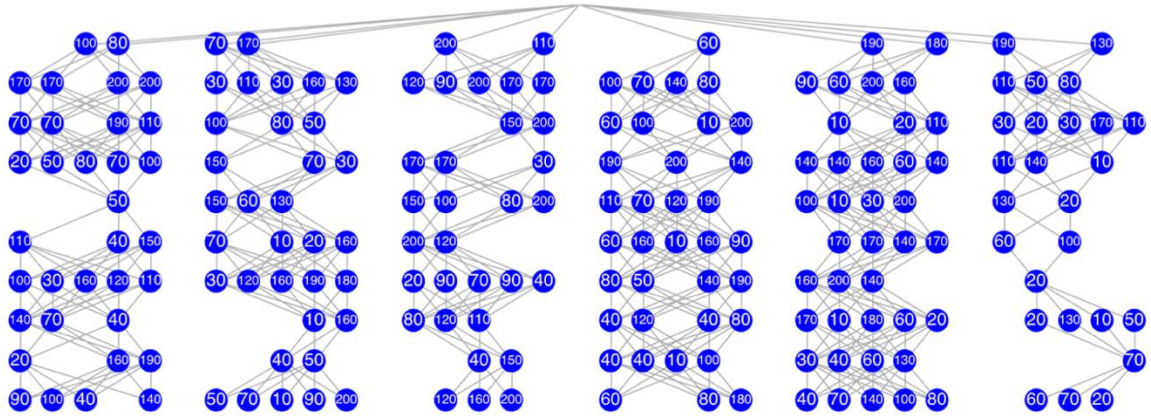
Şekil 3.1 Karınca Kolonisi Optimizasyonu sonuç ekranı

Feromon oranı: Karıncanın rota içinde yeni bir yöne (bir prosesi çizelgeye eklemeye) karar vermesinde haritadaki feromonun önem oranı. 1-10 arası tamsayı değer alabilir.

Çekicilik oranı: Karıncanın çizelgeye yeni bir proses eklemeye karar vermesinde o düğümün (prosesin) işlem zamanının etkisi. 1-10 arası tamsayı değer alabilir.

Başlangıç feromon oranı: Her feromon resetlemesi sonrası haritadaki yollarda bulunacak feromon miktarı. 1-10 arası tamsayı değer alabilir. Artırılması halinde sonradan oluşacak feromon değişmelerinin nisbi önemini azaltacaktır.

Şekil 3.2'de bir 6 iş - 9 proses - 5 makineli bir problemin grafik görünümünü görmekteyiz. Bu görünüm bir karıncanın nasıl çözüm aradığını gösteriyor. Görselde farklı işler arası yollar gösterilmemiş, sadece bir işin prosesleri arası yollar gösteriliyor.

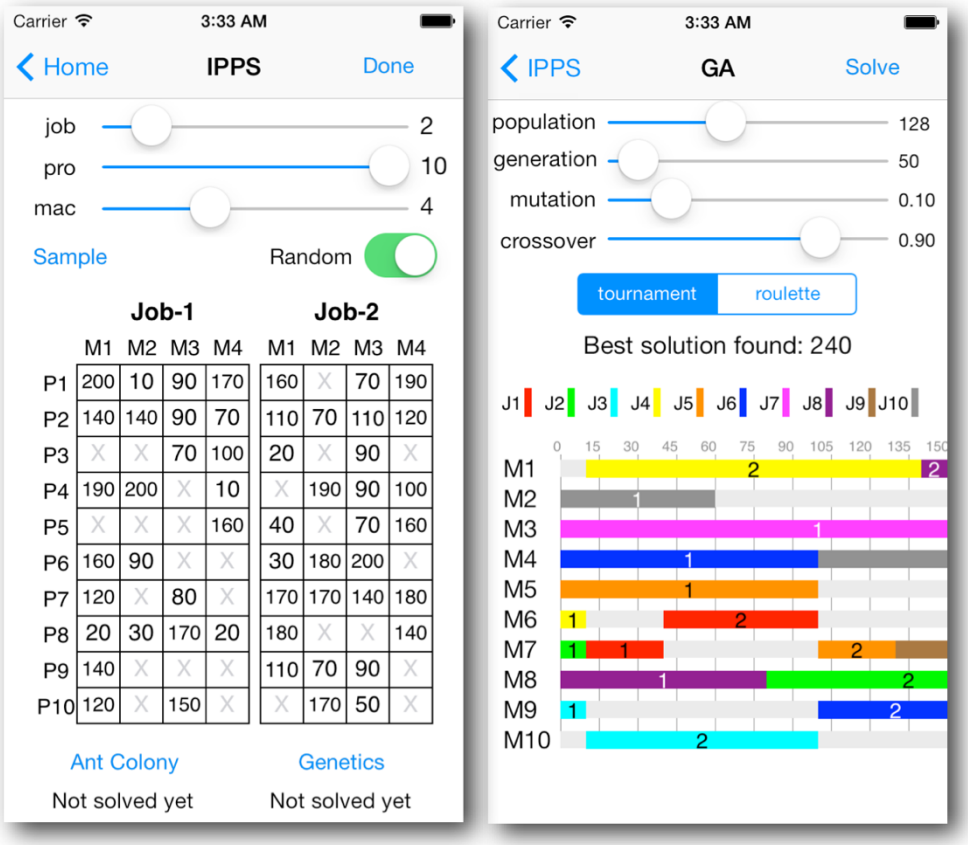


Şekil 3.2 Örnek bir problem için haritanın grafik gösterimi

Algoritmanın ayrıntılarında Node, Arc, Ant, Map ve Supervisor isimlerinde 5 sınıf bulunuyor. Node, bir prosesin bir makinede çalıştırılmasını simgeliyor. Karınca buradan geçerse bu rota için o proses ilgili makinede çalıştırılacak şekilde bir çözüm üretiliyor. Arc ise Node'lar arasındaki bağlantıları simgeliyor. Karınca bu Arc'lar üzerinde dolaşarak başka Node'lara ulaşabiliyor. Map sınıfı problem uzayının çizge halini simgeliyor. Supervisor ise problemin çözümünü yöneten fonksiyonları barındırıyor.

3.1.1 Yönetici

Supervisor (yönetici) sınıfı problemi yönetir. Problemin parametreleri burada tutulur. İçinde belli değişkenler vardır. Bunlar myMap, antList, winnerAnt, Xab, Xrb, Xib, maxNC, maxRepeatCnt, ant, NC, RepeatCnt, alpha, beta, rho, tho0, C, Q, machineCount, jobCount ve processCount olarak sıralanabilir. İki adet de fonksiyon vardır, bunların isimleri init ve solveIPPS şeklindedir.



Şekil 3.3 iPhone 4.0 inç ekran için uygulama dizaynı

myMap: Problemi içeren çizgedir.
antList: Çizgeyi gezecek karınca listesidir.
winnerAnt: En iyi çözümü bulan karıncadır.
Xab: En iyi çözümün C_{max} değeridir.
Xrb: Son resetten sonraki en iyi çözümün C_{max} değeridir.
Xib: O anki iterasyonda bulunan en iyi çözümün C_{max} değeridir.
maxNC: En fazla iterasyon sayısıdır.
maxRepeatCnt: İyi çözüm gelmediği takdirde resetleme için kaç iterasyon bekleneceğidir.

ant:	Karınca sayısıdır.
NC:	İterasyon sayacıdır.
RepeatCnt:	Resetleme bekleme sayacıdır.
alpha:	Feromon oranıdır.
beta:	Çekicilik oranıdır.
rho:	İterasyon sonrası feromon buharlaşma oranıdır.
tho0:	Başlangıç feromon miktarıdır.
C:	Çekicilik hesabı için kullanılan bir sabittir.
Q:	Feromon güncelleme için kullanılan bir sabittir.
machineCount:	Makine sayısıdır.
jobCount:	İş sayısıdır.
processCount:	Proses sayısıdır.
init():	Problem parametrelerini kaydeden fonksiyondur.
solveIPPS():	Çözümü başlatan fonksiyondur. Karıncaları yönetir, sonuçları toplar.

```
1. iterasyon:0'dan maksimumİterasyonSayısı'na kadar:
2.      Xib = sonsuz;
3.  karıncaListesi = boş;
4.      her karınca için:
5.          harita.başlangıçNoktası'na karınca koy;
6.          sonuç = karınca.çizelgeOluştur();
7.          Eğer (sonuç < Xib) ise:
8.              Xib = sonuç
9.          Eğer (Xib < Xrb) ise:
10.             Xrb = Xib;
11.   i: yayListesi içindeki her eleman için:
12.         harita.yayListesi[i].feromonSeviyesi *= (1-rho);
13.   i: karınca.yayK içindeki her eleman için:
14.         karınca.yayListesi[i].feromonSeviyesi =+= Q/Xrb;
15.   Eğer (Xrb < Xab) ise:
16.         eniyiKarınca = kazanan;
17.   Değilse:
18.         tekrarSayısı ++;
19.   r = rastgele(0,1);
20.   cp = log(NC)/log(maxNC);
21.   Eğer (tekrarSayısı >= maksimumTekrarSayısı VE r > cp) ise:
22.         tekrarSayısı = 0;
23.         Xrb = sonsuz;
24.         i: yayListesi içindeki her eleman için:
25.             harita.yayListesi[i].feromonSeviyesi = tho0;
26. eniyiKarınca'yı dön;
27. Bitir
```

3.1.2 Harita

Map (harita) sınıfı problem uzayının çizgesini simgeler. Çizge üzerindeki düğümler bir prosesin bir makinede çalıştırılmasını simgeler. Çizgedeki yaylar (Arc) düğümler

arasındaki yolları simgeler. Bu sınıfın jobList, nodeList, arcList, startNode ve finishNode isimli değişkenleri, init() isimli de tek bir fonksiyonu vardır.

3.1.3 Düğüm

Node (düğüm) sınıfı bir prosesin bir makinede işlenmesini simgeler. Düğümün bağlantıları da buradan sonra gidilebilecek başka düğümlerdir. Bu sınıfın arcList, jobID, processID, machineID ve time isimli değişkenleri ve init() isimli bir fonksiyonu vardır.

3.1.4 Yay

Arc (yay) sınıfı iki düğüm arasındaki bağlantıyı simgeler. Bu sınıfın sourceNode, destinationNode, pheromoneLevel ve desirability isimli değişkenleri, init() isimli bir de fonksiyonu vardır.

3.1.5 Karınca

Ant (karınca) sınıfı çizgede yiyecek (çözüm) arayan bir karıncayı simgeler. Bu sınıfın antID, currentNode, Xk, Gk, Sk, tabuK, arcK ve jobProcessQueue isimli değişkenleri; init(), constructSchedule(), nodeSelectionProcedure() ve getMakespanForRoute() isimli fonksiyonları vardır.

antID:	Karıncanın numarasıdır.
currentNode:	Karıncanın o an bulunduğu düğümdür.
Xk:	Karıncanın elde ettiği çözümün C_{max} değeridir.
Gk:	Ziyaret edilmesi gereken düğümlerdir.
Sk:	Bir sonraki admında ziyaret edilebilecek düğümlerdir.
tabuK:	Ziyaret edilen düğümlerdir.
arcK:	Ziyaret edilen bağlantılardır.
jobProcessQueue:	Tamamlanan proseslerdir.
init():	Karıncayı oluşturan fonksiyondur.

constructSchedule(): Her bir iterasyonda karıncaya bir çizelge oluşturan fonksiyondur.

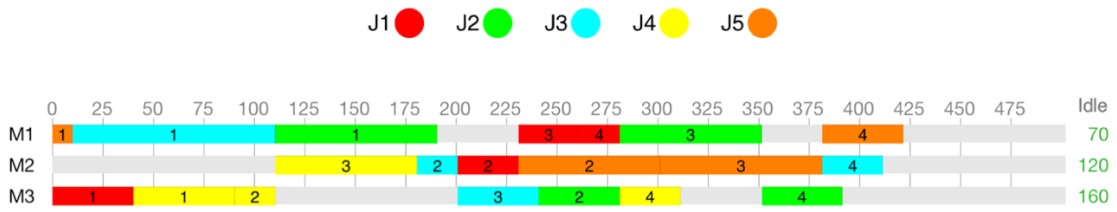
1. **Döngü** halinde yap:
2. i: Gk dizisi içindeki her eleman **için**:
3. **Eğer** (bulunanDüğüm.prosesID=Gk[i].prosesID VE
4. bulunanDüğüm.işID=Gk[i].işID) **ise**:
5. Gk.çıkar(i);
6. **Eğer**(Gk.uzunluk()=0) **ise**:
7. **Bitir**;
8. gelecekDüğüm = düğümSeçmeProsedürü();
9. tabuK.ekle(gelecekDüğüm);
10. Sk = gelecekDüğüm.yayListesi;
11. i: Sk içindeki her eleman **için**:
12. **Eğer**(işKuyruğu[Sk[i].hedefDüğüm.işID]==Sk[i].hedefDüğüm.prosesID) **ise**:
13. Sk.çıkar(i);
14. bulunanDüğüm = gelecekDüğüm;
15. Xk = CmaxHesapla(tabuK, makineSayısı, işSayısı);
16. Xk'yı dön;
17. **Bitir**

nodeSelectionProcedure(): Bir sonraki düğüme karar veren fonksiyondur.
getMakespanForRoute(): Çözümün C_{max}'ını hesaplayan fonksiyondur.

1. Cmax = 0;
2. i tabuList içerisindeki her eleman **için**:
3. başlangıçZamanı = MAKSİMUM(işBeklemesi[işID],makineBeklemesi[makineID]);
4. işBeklemesi[işID] = başlangıçZamanı + proses[i].zaman;
5. makineBeklemesi[makineID] = başlangıçZamanı + proses[i].zaman;
6. i:0'dan makineSayısı'na **kadar**:
7. **Eğer** (mmakineBeklemesi[i] > Cmax) **ise**:
8. Cmax = makineBeklemesi[i];
9. Cmax'ı dön;
10. **Bitir**

3.2 Genetik Algoritma

Uygulamada ayrıca Genetik Algoritma kullanılmıştır. Genetik Algoritma temel prensiplerini doğal seçilim ilkesinden alır. Belli sayıda kromozom içeren bir başlangıç populasyonu ile başlanır. Bir kromozom verilen çizelgeleme problemi için bir çözümü simgeler. Her iterasyonda kromozomlar kendi aralarında çaprazlanıp yeni kromozomlar oluştururlar. Her yeni nesilde uygunluk fonksiyonunda daha iyi sonuç veren kromozomlar yaşamaya devam ederler. Belli bir sayıda nesil sonrasında elde edilen en iyi çözüm, algoritmanın çözümü olur. Şekil 3.3'te uygulamanın örnek Genetik Algoritma çözüm ekranı görülmektedir. Şekil 3.4'te ise uygulamanın mobil versiyonu için küçültülmüş Genetik Algoritma ekranı görülmektedir.



Şekil 3.4 Örnek Çözüm

3.2.1 Gen Modeli

Bu çalışmadaki yaklaşımda bir gen; *jobID*, *processID*, *machineID* ve *runningTime* değişkenlerinden oluşur. Gene ait işin ve prosesin yine gene ait makinede çalıştırılmasının seçilmesi durumunda ilgili gen kromozom dizisine eklenir. Örneğin Şekil 3.4'de (5,1,1) geni çözümde olması gereken bir gendir çünkü çözümde 5. işin 1. prosesi 1. makinede çalışmıştır.

3.2.2 Kromozom Modeli

Çalışmadaki yaklaşımda bir kromozom bir çözümü (çizelgeyi) simgeler. Gen listesini ve çizelgeye eklenme sırasını içerir. Bir kromozom iki bölümden oluşur. İlk bölüm makine listesidir. Sırasıyla her prosesin hangi makinede çalıştırılacağını yazar. Örneğin Şekil

3.4’deki çözümde ilk işin ilk prosesinin 3. makinede çalışmış olduğu görülebilir. Demek ki bu çizelgeyi simgeleyen kromozom dizisinin ilk bölümü 3 ile başlamalıdır. Daha sonra 2 gelmelidir çünkü ilk işin 2. prosesi 2. makinede çalışmıştır. Şekil 3.4’ye göre kromozom listesinin ilk bölümü 3-2-1-1-1-3-1-3-1-2-3-2-3-3-2-3-1-2-2-1 dizisi şeklinde olmalıdır.

Kromozom modelinin ikinci bölümü, sıra listesinden oluşur. Bu liste de işlerin hangi sırayla çizelgeye ekleneceklerini simgeler. İşlerin hangi sırayla çizelgeye ekleneceği belirlenmezse çizelgeyi oluştururken belirsizlikler ortaya çıkabilir, ancak birden fazla sıra listesi de aynı çizelgeyi oluşturabilir. Örneğin Şekil 3.4’deki çözüme göre ilk eklenecek iş 1. iş veya 5. iş olabilir. Örneğin 5-3-2-1-4-4-4-3-3-1-1-1-5-5-2-2-4-2-3-5 listesi Şekil 3.4’deki çizelgeyi oluşturabilecek bir çizelgedir.

Sonuç olarak kromozom dizimiz toplamda şöyle olacaktır:
[3|2|1|1|1|3|1|3|1|2|3|2|3|3|2|3|1|2|2|1]
[5|3|2|1|4|4|4|3|3|1|1|1|5|5|2|2|4|2|3|5]

3.2.3 Algoritma Taslağı

Çalışmadaki yaklaşıma göre önce istenilen büyüklükte bir başlangıç populasyonu (kromozom dizisi) oluşturulur. Sonra kullanıcı tarafından verilen oranlara ve yöntemlere göre çaprazlama, mutasyon ve seçim prosedürleri, jenerasyon sayısı adedince uygulanır. Belli sayıda iterasyon sonrasında bulunan en iyi çözüm (kromozom) algoritmanın sonucu olarak dönülür.

Genetik algoritmanın verimliliğini etkileyen bazı parametreleri vardır:
Populasyon büyüklüğü (p): Ekosistemdeki kromozom sayısı. Artırılması durumunda daha iyi çözümler elde edilebilir ancak daha çok zaman harcanır. 8-300 arası değer alabilir.

Jenerasyon sayısı (g): Kaç nesil boyunca kromozomların çaprazlanacağını belirler. Aynı şekilde artması durumunda daha iyi sonuçlar elde edilir ancak daha fazla zaman alır. 1-1000 arası değer alabilir.

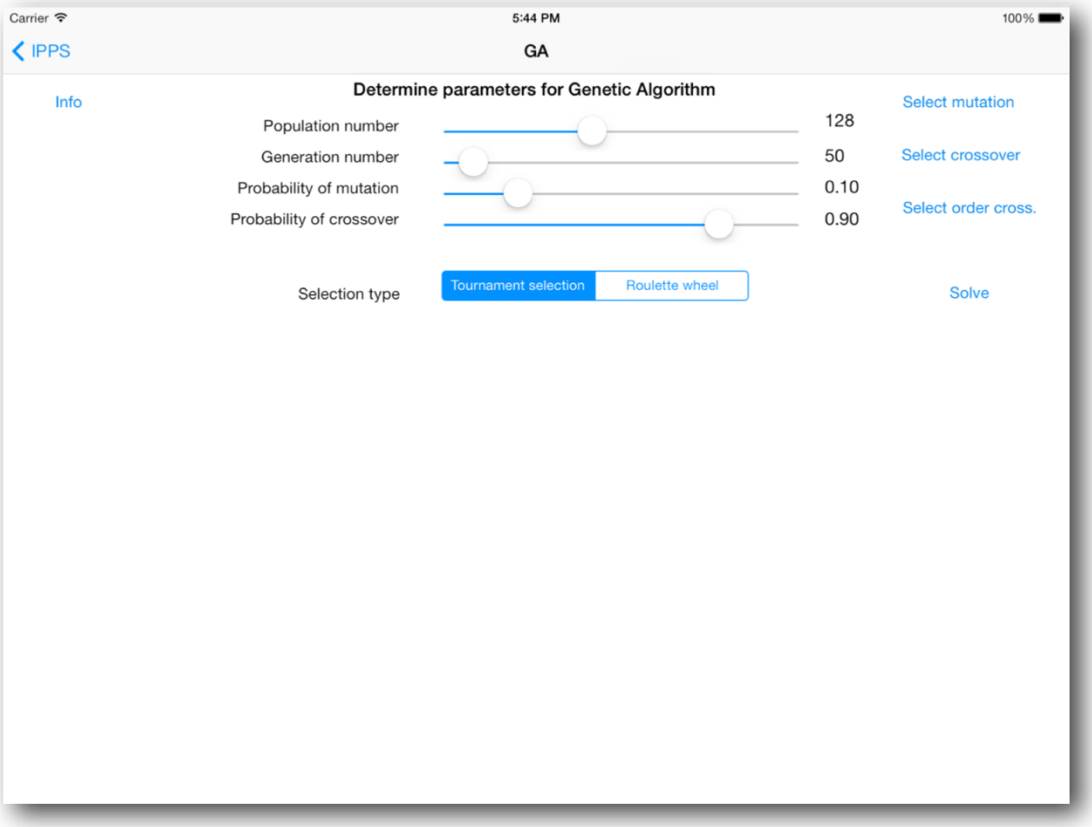
Mutasyon olasılığı (P_m): Her çaprazlama sonrasında kromozom dizilimlerinde oluşabilecek rassal değişikliklerin gerçekleşme olasılığı. 0.01 ile 0.5 arasında değer alabilir.

Çaprazlama olasılığı (P_c): Kromozomun bir iterasyonda çaprazlama yapma olasılığı. 0.5 ile 1 arası değer alabilir.

Seçilim tipi (s): Her nesil sonrasında hangi bireylerin yaşatılacağının belirlenme işlemleri. Turnuva seçilimi, rulet tekerleği gibi farklı seçenekleri vardır.

Mutasyon tipi (m): Ters mutasyon, komşu genlerin değiştirilmesi, rastgele iki genin değiştirilmesi, rastgele üç genin değiştirilmesi ve araya yerleştirme gibi seçenekleri mevcuttur.

Çaprazlama tipi (c): Tek nokta çaprazlama, çift noktalı çaprazlama ve çok noktalı çaprazlama gibi seçenekleri vardır.



Şekil 3.5 GA Çözüm Ekranı

Şekil 3.3'teki ikinci ekranda aynı zamanda Genetik Algoritma ile elde edilen örnek bir çözüm de görülmektedir. Bu çözümü oluşturan problem 10 proses ve 10 makineye sahiptir. Bu problemin çözümündeki parametrelerden populasyon büyüklüğü 128, jenerasyon sayısı 50, mutasyon olasılığı 0.1, çaprazlama olasılığı 0.9, seçilim tipi olarak ise turnuva seçilimi kullanılmıştır.

Genetik algoritma kısmında Gene (Gen), Chromosome (Kromozom), Genetics (Genetik) ve Ecosystem (Ekosistem) isimli dört adet sınıfımız bulunmaktadır. Gen sınıfı bir prosesin bir makinede çalışmasını simgeler. Kromozom sınıfı bir çizelgenin tamamını simgeler, gen dizisinden oluşur. Genetik sınıfı mutasyon ve çaprazlama tipleri gibi fonksiyonları içeren bir kütüphanedir.

3.2.4 Ekosistem

Ekosistem sınıfı çözümü yöneten sınıftır. Geneli problem parametreleri olan jobNumber, machineNumber, processNumber, generationNumber, populationNumber, currentGeneration, population, mutationRate, crossRate, mutationType, crossoverType, orderedCrossType, rouletteWheel, makespan ve solutionList adında değişkenlere sahiptir. Ayrıca crossPopulation, mutatePopulation, naturalSelectionRoulette, naturalSelectionTournament, sortPopulationWithMakespan ve solveIPPS adında da fonksiyonlara sahiptir.

jobNumber:	İş sayısıdır.
machineNumber:	Makina sayısıdır.
processNumber:	Proses sayısıdır.
generationNumber:	Jenerasyon (nesil) sayısıdır.
populationNumber:	Populasyon büyüklüğüdür.
currentGeneration:	O anki jenerasyon numarasıdır.
population:	Kromozom dizisinden oluşan populasyondur.
mutationRate:	Mutasyon oranıdır, 0-1 arası değer alabilir.
crossRate:	Çaprazlama oranı, 0-1 arası değer alabilir.
mutationType:	Mutasyon tipidir.
crossoverType:	Çaprazlama tipidir.
orderedCrossType:	Sıra çaprazlama tipidir.
rouletteWheel:	Seçilim yöntemidir.
makespan:	Bulunan en iyi çözümdür.
solutionList:	Bulunan en iyi çözümün gen listesidir.
init():	Verilen parametreler ile ekosistemi başlatır.
crossPopulation():	Populasyonu kendi içinde çaprazlar.
mutatePopulation():	Populasyonu mutasyona maruz bırakır.
naturalSelectionRoulette():	Rulet tekerleği seçilimini işletir.
naturalSelectionTournament():	Turnuva seçilimini işletir.
sortPopulationWithMakespan():	C_{max}'a göre populasyonu sıralar.
solveIPPS():	Problemi çözer.

1. Cmax = sonsuz;
2. bulunanJenerasyon:0'dan jenerasyonSayısı'na kadar:
3. popülasyonuSırala();
4. eğer(seçim yöntemi ruletÇarkı) ise:
5. ruletÇarkıUygula();
6. eğer(seçim yöntemi turnuvaSeçimi) ise:
7. turnuvaSeçimiUygula();
8. populasyonuÇaprazla();
9. populasyonaMutasyonUygula();
10. çözümListesi = popülasyon[0].genotip;
11. Cmax = popülasyon[0].CmaxHesapla();

3.2.5 Gen

Gen (Gene) sınıfı bir prosesin bir makinadaki işleyişini simgeler. 4 adet değişkeni vardır.

jobID İş numarasıdır.
processID İşe ait proses numarasıdır.
machineID Makina numarasıdır.
time Prosesin makinedeki çalışma süresidir.

3.2.6 Kromozom

Kromozom (Chromosome) sınıfı basitçe bir gen dizisinden ibarettir ve bir çizelgeyi simgeler. Genotype, makespan, jobNumber, processNumber, machineNumber, machine, order ve myEcosystem isimli değişkenleri; init, getMakespan, cross ve mutate isimli fonksiyonları vardır.

3.2.7 Genetik

Genetik (Genetics) sınıfı mutasyon ve çaprazlama fonksiyonlarının kütüphanesidir. Çeşitli mutasyon, çaprazlama ve seçilim fonksiyonları vardır:

- singlePointCrossover (Tek noktalı çaprazlama)

  Rastgele seçilen bir noktadan sona kadar kromozomların genlerini değiştirir.
- twoPointCrossover (İki noktalı çaprazlama)

Rastgele bir noktadan rastgele bir uzunluğa kadar kromozomların genlerini değiştirir.

- multiPointCrossover (Çok noktalı çaprazlama)

  singlePointCrossover fonksiyonunu birkaç kez uygular.
- positionBasedCrossover (Pozisyon tabanlı çaprazlama)

  Rastgele seçilen bazı noktaları bir ebeveynden, geri kalanları diğerinden alır.
- partialMappedCrossover (Parçalı haritalı çaprazlama)

  Önce twoPointCrossover'ı uygular, sonra yinelenen karakterleri eskisiyle değiştirir.
- linearOrderedCrossover():

  Bir circleCrossover varyantıdır. Her ebeveynden rastgele bir aralık seçer. Bunları çıkartır ve boş kalan pozisyonları belirler. Sonra boş pozisyonları doldurur.
- orderBasedCrossover (Sıra tabanlı çaprazlama)

  Önce rastgele noktalar seçilir, ardından birinci ebeveynin seçilen noktalarındaki genleri diğer ebeveynin kromozomundaki sağ komşusuna konulur. Sonra kullanılmayan genler kalan seçim noktalarına yerleştirilir.
- orderedCrossover (Sıralı çaprazlama)

  İki ebeveyn için de bir aralık belirlenir ve bu aralıktaki genler yer değiştirilir. Sonra tekrarlananan genler diğer ebeveyndeki gen sırasına göre yer değiştirilir.
- circleCrossover (Çember çaprazlama)

  İlk ebeveynin ilk geni seçilir. Sonra diğer ebevenynin aynı hizadaki geni seçilir. Sonra ilk ebeveynin, ikinci ebeveynin seçilen geninde yazan numaraya sahip geni bulunur ve onun karşısındaki gen seçilir. Bu proses başlangıçtaki gene ulaşıncaya kadar sürdürülür. Seçilen pozisyonlar kalır, diğer numaralar ikinci ebeveyndeki genlerle yer değiştirilir.

- mutationFloat():

Rastgele seçilen bir bölümü sola kaydırır, en soldaki geni de sağ tarafa alır.

- mutationReverse():

  Rastgele seçilen iki nokta arasındaki genlerin sıralamasını ters çevirir.

- mutationNeighbor():

  Rastgele seçilen bir geni sağdaki komşusu ile yer değiştirir.

- mutationArbitrary():

  Rastgele seçilen iki geni yer değiştirir.

- mutation3Arbitrary():

  Rastgele seçilen üç geni çember biçiminde yer değiştirir.

- repair():

  Bazı mutasyon ve çaprazlama fonksiyonları her problemin kromozom yapısına uygun olmayabilir. Dolayısıyla kromozomlarda bozulmalar meydana gelebilir. Bu fonksiyon zarar görmüş bir kromozomu tamir etmeye yarar.

- repairForOrder():

  Bu fonksiyon kromozomun iş sırasını simgeleyen kısmındaki bozulmaları tamir eder.

3.3 Yapay Bağışıklık Sistemi

Çalışmanın bu kısmında bir Yapay Bağışıklık Sistemi algoritması yazılmıştır. Yazılan algoritmanın temel prensipleri Orhan Engin ve Alper Döyen'in [47] makaleleri ile Nhu Binh Ho ve diğerlerinin [48] makalelerinden alınmıştır.

Yapay bağışıklık sistemi (YBS) algoritması bağışıklık sistemi mekanizmalarından ilham almış bir sezgisel problem çözme tekniğidir. Çizelgeleme problemlerinde kullanımı 20 seneden daha eskiye dayanır. Temel yaklaşımı önce her birisi birer çözümü simgeleyen rastgele Antikorlar üretip daha sonra bunları çeşitli mutasyonlarla iyileştirmeye çalışmak şeklindedir. Programda kullanılan Antikor modeli, bu çalışmadaki Genetik algoritmada kullanılan Kromozom modeli ile bire bir aynıdır.

3.3.1 Antikor Modeli

Antikorlar da kromozomlar gibi genlerden oluşurlar. Bir antikor iki bölümden oluşan bir gen dizisidir. Bu iki bölüm de çizelgeleme problemindeki toplam proses adedince genden oluşur. Yani bir problemde toplam **K** kadar iş var ve her bir iş **L** kadar prosesten oluşuyorsa bir antikorumuz toplam **2 x K x L** kadar gen içerecektir. Bu iki bölümden ilki sırayla proseslerin hangi makinede çalışacaklarını simgelerler. İkinci kısım da proseslerin hangi sırayla çizelgeye yerleştirileceklerini simgeler. Örneğin her işin **L** kadar prosesten oluştuğu bir problemde **k** numaralı işin **l** numaralı prosesinin antikorun ilk bölümündeki veya ikinci bölümündeki sırası (**k-1)*L + l** olacaktır. Antikorun ilk bölümünde **M** makineli bir problem için konuşursak **0** ile **M** arasında sayılar yeralacaktır ve örneğin herhangi bir gende **m** yazmakta ise bu bahsi geçen prosesin **m** numaralı makinede çalışması gerektiğini simgeler. Antikorun ikinci kısımda ise **K** işli bir problem için konuşursak **0** ile **K** arasında sayılar yeralacaktır ve örneğin **i**. sıradaki bir gende **k** yazması, çizelgeye **i**. sırada yerleştirilecek prosesin **k** numaralı işin sıradaki işlenmesi gereken prosesi olduğu demektir.

Örneğin 5 iş 4 proses ve 3 makineli bir problem için örnek bir antikorumuz aşağıdaki gibi olsun:

[3|2|1|1|1|3|1|3|1|2|3|2|3|3|2|3|1|2|2|1]

[5|3|2|1|4|4|4|3|3|1|1|1|5|5|2|2|4|2|3|5]

Burada antikor görüldüğü gibi 2 bölümden müteşekkildir ve her bölüm 5x4 = 20 gen içermektedir. Antikorun toplam uzunluğu ise 20x2 = 40 olmaktadır. Problemde 3 makina olduğu için antikorun ilk bölümündeki genler 1 ile 3 arasındaki sayılardan oluşmakta, 5 iş olduğu için ise antikorun ikinci bölümündeki genler 1 ile 5 arasındaki sayılardan oluşmaktadır. Örneğin 1. işin 2. prosesinin çalışacağı makina için antikorun ilk bölümündeki (1-1)*4+2 = 2 numaralı gene bakarız. Burada 2 yazdığından ötürü 2. işin 2. prosesi 2. makinada çalışacaktır. Veya 5. işin 1. Prosesine bakmak için (5-1)*4+1 = 17 olduğundan ilk bölümün 17. sırasındaki gene bakarız. Prosesleri çizelgelerken de ikinci bölümü kullanırız. Örneğin çizelgeye ilk olarak 5. işin konulacağını gördüğümüzden 5. işin çalıştırılma sırası gelen prosesini (burada ilk proses) çizelgeye

ekleriz. Daha sonra sırasıyla 3, 2, 1 ve 4 numaralı işlerin ilk proseslerini çizelgeledikten sonra antikorun sıradaki geninde yine 4 yazdığından bu sefer 4 numaralı işin 2. prosesini çizelgeye ekleriz.

Algoritma Parametreleri

Programa alınan parametreler sırasıyla şu şekildedir.

Populasyon büyüklüğü (P): Genetik algoritmadaki popülasyon büyüklüğü parametresi ile benzerdir. Başlangıçta kaç adet antikor oluşturulacağını belirler.

Jenerasyon sayısı (J): Genetik algoritmadaki jenerasyon sayısı ile benzerdir. Antikorların kaç sefer somatik hipermutasyon, reseptör denetimi gibi çeşitli mutasyonlara uğratılacaklarını belirler.

Klonlama olasılığı (K): Her bir jenerasyonda popülasyondaki antikorlardan bir miktarı rulet çarkı yöntemiyle klonlanmak üzere seçilir. Bu sayı da **KxP** formülü ile bulunur. Örneğin 100 adet antikorumuz var ve klonlama olasılığı 0.5 ise 100 x 0.5 = 50 adet antikor klonlanır.

Mutasyon olasılığı (M): Her bir jenerasyonda klonlanan antikorlardan ne kadarına somatik hipermutasyon uygulanacağını **M x K x P** ile belirler. Örneğin başlangıçta 100 tane antikor var, bunların 0.5 kadarı klonlanmış ve mutasyon olasılığı 0.1 ise 100 x 0.5 x 0.1 = 5 adet antikora somatik hipermutasyon uygulanır.

Reseptör olasılığı (R): Somatik hipermutasyonun ardından en kötü belli sayıda antikora reseptör denetimi adıyla bâzı mutasyonlar uygulanır. Böylece bir miktar iyileştirilmeleri umulur. En kötü kaç antikora reseptör denetimi uygulanacağı **R x P** formülü ile bulunur. Örneğin 100 antikor var ve reseptör olasılığı 0.1 ise 100 x 0.1 = 10 adet antikor reseptör denetimine gönderilir.

Hafıza hücresi sayısı (H): Her jenerasyonda klonlama, somatik hipermutasyon ve reseptör denetimi sonunda belli sayıdaki en kötü antikorun yerine aynı sayıdaki en iyi antikorlar kopyalanır. Bu sayıyı da hafıza hücresi sayısı belirler. Örneğin 100 antikor

varsa ve hafıza hücresi sayısı 10 ise uygunluk değeri en kötü 10 antikor silinip yerlerine uygunluk değeri en iyi olan 10 antikor kopyalanır.

3.3.2 Algoritma Taslağı

Önce gerekli parametreler kullanıcıdan alınır. Sonra popülasyon büyüklüğü adedince rastgele antikor oluşturulur. Daha sonra jenerasyon adedince şu işlemler tekrarlanır.

Rulet tekeri yöntemi ile klon sayısı kadar antikor seçilir ve kopyalanır. Seçilen antikorlardan mutasyon olasılığına göre belirlenen sayıdaki en kötü antikorlar Somatik Hipermutasyon işlemine göre güncellenir. Bu işlemden sonra klon sayısı ile popülasyon büyüklüğü farkı kadar yeni rastgele antikor oluşturulur ve klon listesine eklenir. Daha sonra klon listesinden reseptör olasılığına göre belilenen en kötü belli sayıda birey Reseptör Denetimi işlemine göre güncellenir. Son olarak elimizdeki listeden en iyi Hafıza Hücresi Sayısı kadar birey, aynı sayıdaki en kötü bireyin yerine yazılır.

Programın çalışmasına ilişkin pseudocode (yalancı kod) da şu şekildedir:

1. **Başla**
2. Populasyon büyüklüğü = p;
3. Jenerasyon sayısı = j;
4. Klonlama olasılığı = k;
5. Mutasyon olasılığı = m;
6. Reseptör olasılığı = r;
7. AntikorListesi = boş;
8. MinimumCmax = sonsuz;
9. 1'den p'ye **kadar**:
10. Rastgele Antikor oluştur;
11. AntikorListesi'ne rastgele antikor ekle;
12. 1'den j'ye **kadar**:
13. KlonSayısı = k*p;
14. KlonListesi = boş;
15. 1'den KlonSayısı'na **kadar**:
16. KlonListesi'ne RuletCarkı ile antikor ekle;
17. MutasyonSayısı = m*p;
18. 1'den MutasyonSayısı'na **kadar**:
19. KlonListesi'ne SomatikHipermutasyon uygula;
20. KlonSayısı'ndan p'ye **kadar**:
21. Rastgele Antikor oluştur;
22. KlonListesi'ne rastgele antikor ekle;
23. ReseptörSayısı = r*p;
24. KlonListesi'ne ReseptorDenetimi uygula;
25. i=1'den HafızaHucresi'ne **kadar**:
26. KlonListesi[p-i] = KlonListesi[i];
27. **eğer** (MinimumCmax > EnİyiKlon) **ise**:
28. MinimumCmax = EnİyiKlon;
29. **Bitir**

3.3.3 Somatik Hipermutasyon

Bu işlem klonlanan antikorlardan en kötü belli sayıdakine uygulanır. Her antikor için sırasıyla belli mutasyonlar uygulanıp daha iyi bir çizelge oluşana kadar devam edilir, oluşunca bırakılır. Eğer hiçbir mutasyon daha iyi bir çözüm oluşturmazsa antikor olduğu gibi bırakılır. Uygulanan mutasyon tipleri sırasıyla şunlardır:

Araya yerleştirme: Bu mutasyonda rastgele seçilen bir gen yerinden kaldırılıp yine rastgele seçilen başka bir pozisyona yerleştirilir ve aradaki genler ilk pozisyona doğru kaydırılır. Örneğin 12345 antikoru için 4 seçilip 1 ve 2 arasında yerleştirilirse antikorun yeni hali 14235 olur.

Keyfi iki gen değiştirme: Bu mutasyonda rasgele seçilen iki gen birbiriyle yerdeğiştirir. Diğer genlere dokunulmaz. Örneğin 12345 antikoru için 2 ve 4 seçilirse antikorun yeni hali 14325 olur.

Makine değiştirme mutasyonu: Bu mutasyonda bir genin çalıştığı makine değiştirilir. Örneğin 3 makinalı bir problem için antikor 123123 iken 4. gende mutasyon olursa antikorun yeni hali 123323 olabilir.

İki makine değiştirme mutasyonu: Bu mutasyonda iki adet genin çalıştığı makine değiştirilir. Örneğin 3 makinalı bir problem için antikor 123123 iken 2. ve 4. gende mutasyon olursa antikorun yeni hali 113323 olabilir.

3.3.4 Reseptör Denetimi

Bu işlem Somatik hipermutasyon sonrasında oluşan klon listesine, popülasyon numarasına eşitlenene kadar yeni rastgele antikor oluşturulduktan sonra en kötü belli sayıda antikora uygulanır. Yine Somatik hipermutasyonda olduğu gibi ard arda belli mutasyonlar uygulanır, antikor daha iyi olursa durulur. Hiçbir mutasyonla antikor daha iyi hale gelmezse, rastgele yeni bir antikor oluşturulur. İki antikor kıyaslanır, hangisi iyi ise onunla devam edilir. Uygulanan mutasyon tipleri sırası ile şunlardır:

Ters mutasyon: Bu mutasyonda antikorun belli iki bölgesi arası ters çevrilir. Örneğin antikorumuz 12345 ise ve 2. genden 4. gene kadar ters çevirelim dersek antikorumuzun yeni hali 14325 olur.

Makine değiştirme mutasyonu: Bu mutasyonda bir genin çalıştığı makine değiştirilir. Örneğin 3 makineli bir problem için antikor 123123 iken 4. gende mutasyon olursa antikorun yeni hali 123323 olabilir.

Makine sıfırlama mutasyonu: Bu mutasyonda bütün genlerin çalıştığı makineler en baştan rastgele belirlenir. Örneğin 3 makinalı bir problem için antikor 123123 iken yeni hali 312321 olabilir. Bu mutasyon antikorda büyük bir değişim oluşturacağından verimli antikorlara uygulanması büyük ihtimalle daha kötü antikorlar oluşmasına yol açacaktır. O yüzden baştaki küçük değişimlere tepki vermeyen antikorlarda denenmektedir.

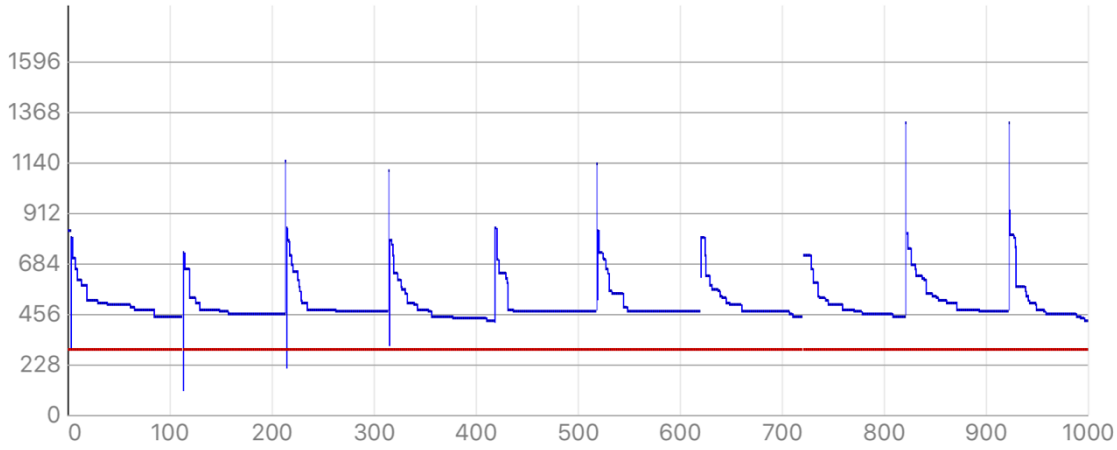
3.4 Melez Genetik Algoritma – Karınca Kolonisi Optimizasyonu

Genetik Algoritma (GA) ve Karınca Kolonisi Optimizasyonu (KKO) üzerinde ilerideki bölümlerde ayrıntılı olarak anlatılacak deneysel çalışmalar gerçekleştirildikten sonra bu parametreler temel alınarak bu iki algoritma melez hale getirilmeye çalışılmıştır.

İki algoritmanın hibriti geliştirilirken önce bir tanesi çalıştırılıp elde edilen sonuçları diğeri ile optimize etme mantığı temel alınmıştır. Birçok metasezgisel algoritmanın ortak bir sorunu, yerel optimum noktalarda sıkışıp çözüm alanının diğer bölgelerinde arama yapamamaktır. Bu durumu önlemek için türlü çözümler geliştirilmiştir. Bunlardan birisi de “Yakınsama’dan Kaçınma” [46] şeklinde adlandırılan yöntemdir. Bu yönteme göre eğer algoritmanın arama bölgesi daralmışsa ve bu daralmadan kendi dinamikleri ile kurtulamıyorsa arama bölgesini daraltan etmenler sıfırlanıp algoritma yeniden başlatılır. Bu yöntem bu çalışmada hem KKO hem GA’ya uygulanmıştır. KKO’da feromon oranları sıfırlanmış, GA’da ise popülasyon sıfırlanmıştır. Bu durum bir algoritmadan elde edilen sonuçları alıp diğeri ile optimize etmeyi zorlaştırıcı bir

etmendir. Çalışmada yapılan testler de bunu göstermiştir, algoritmalardan alınan sonuçları yerel olarak optimize etmek kısıtlı ölçüde iyileştirme sağlamaktadır.

Bu nedenle başlangıç algoritmasının hangisi olacağı önem taşımaktadır. Bu yüzden önerilen melez yaklaşım başlangıçta iki algoritmayı da paralel bir şekilde kısa bir süre çalıştırır ve elde ettikleri sonuçları gözlemler. Hangi algoritmanın elde ettiği sonuç daha iyi ise o algoritma ilk çalıştırılacak algoritma olarak seçilir, belli bir süre daha çalıştırılma sürdürülür.



Şekil 3.6 Genetik algoritma iterasyon grafiği

Örneğin Şekil 3.6'da Genetik Algoritma'nın 5 iş 5 proses 5 makine ölçeğindeki bir problem için 1000 iterasyonu görülmektedir. Yakınsamadan kaçınma uygulandığı için belli periyodlarla algoritma çalışmaya sıfırdan başlamıştır. Bu durum aynı zamanda şu demektir, aslında program belli sayıda iterasyon yapacak şekilde çalıştırıldığında elinde bulduğu en iyi çözüme yakın birçok sayıda çözüm bulunmaktadır. Bu çözümlere KKO'da X_{rb} (restart-based) demiştik, yani en son sıfırlamadan sonra bulunan en iyi çözümler. Melez yaklaşım da bu çözüm listesinden yararlanmaktadır. Örneğin önce GA sonra KKO çalıştırılacak olsun, melez yaklaşımda GA çalıştırıldığında elde edilen X_{rb} listesi saklı tutulur. Yani her sıfırlamadan önce listeye en son gelinen çözüm de eklenir. Böylece GA'nın çalışması bittiğinde elimizde bir miktar iyileştirilmiş çözüm simgeleyen kromozomlar bulunacaktır. Daha sonra KKO çalıştırılırken bu çözümlerin rotaları başlangıç feromonları olarak haritaya eklenecektir. Aynı "Yakınsamadan kaçınma" yaklaşımı KKO'da da kullanıldığından, Şekil 3.6'daki grafiğe benzer bir grafik KKO'da da

oluşacaktır. Böylece yine her sıfırlamadan sonra GA'dan elde edilen X_{rb} listesindeki bir sonraki kromozom, bir sonraki KKO sıfırlamasından önce haritaya başlangıç feromon seviyesi olarak yerleşecektir. Böylece örneğin GA'da bulunan 10 adet iyi çözüm, KKO'da tek tek iyileştirilmeye çalışılacaktır.

Tersine, eğer önce KKO'yu sonra GA'yı çalıştırımaya karar verilmiş ise, bu sefer de KKO'nun her sıfırlaması öncesi elde ettiği iyi rotalar, GA'nın her sıfırlamasında başlangıç popülasyonuna bir kromozom olarak eklenir. Böylece iki türlü de bir algoritmadan elde edilen sonuçlar bir diğeri ile iyileştirilmeye çalışılacaktır.

Programın taslağı şu şekildedir:

```
1. Başla
2.  zamanLimiti = 1000 ms;
3.      çözümSayısı = 10;
4.      eniyiGA = GA(zamanLimiti);
5.      eniyiKKO = KKO(zamanLimiti);
6.      Xrb = [];
7.      eniyiÇözüm = sonsuz;
8.      Eğer (eniyiGA < eniyiKO) ise:
9.          çözümSayısı kadar:
10.             Xrb.ekle(GA);
11.         i:0'dan çözümSayısı'na kadar:
12.             haritayaFeromonEkle(Xrb[i]);
13.         çözüm = KKO;
14.         Eğer (çözüm < enİyiÇözüm) ise:
15.             enİyiÇözüm = çözüm;
16.     Değilse:
17.         çözümSayısı kadar:
18.             Xrb.ekle(KKO);
19.         i:0'dan çözümSayısı'na kadar:
20.             popülasyonaKromozomEkle(Xrb[i]);
21.         GA;
22.         Eğer (çözüm < enİyiÇözüm) ise:
23.             enİyiÇözüm = çözüm;
24. Yazdır(enİyiÇözüm)
25. Bitir
```

BÖLÜM 4

BULANIK MANTIK YAKLAŞIMI

Bulanık mantık 1921 Bakü doğumlu Prof. Lütfi Aliasker Zade tarafından 1961 yılında ortaya atılmış bir yaklaşımdır. Geleneksel yöntemlerde kümelerin üyelik durumları kesin değerler alırken, bulanık mantık belli bir yüzde ile üyelik yaklaşımını getirmiştir. Dolayısıyla klasik yöntemler ile yapılan modellemelerden gerçeğe daha uygundur. [49]

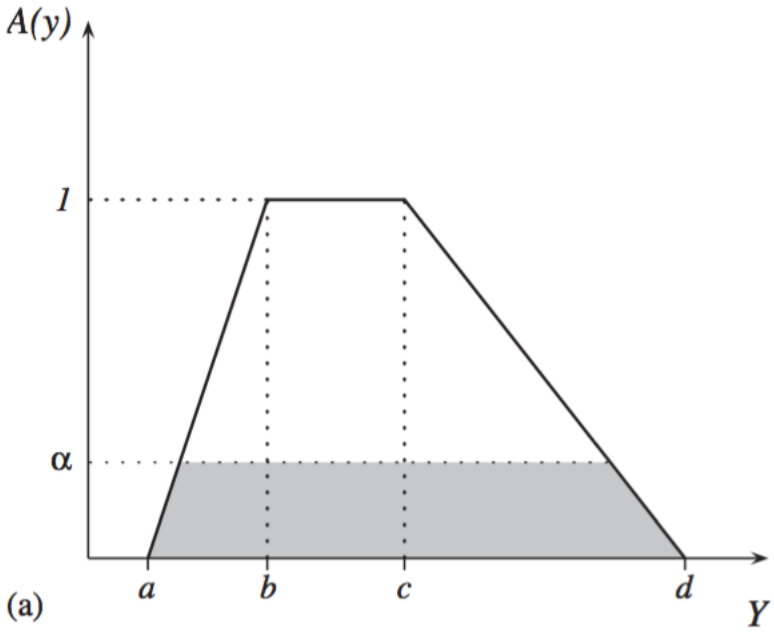
Bulanık yaklaşımda üyelik kümeleri klasik sayı tipleri yerine genelde Üçgensel Bulanık Sayı (ÜBS) ve Yamuk Bulanık Sayı (TFN) olarak kullanılır. ÜBS'ler (a_1,a_2,a_3) gibi 3 rakamla ifade edilir ve üyelik fonksiyonu şu şekildedir: [50]

$$\begin{array}{ll} 0 & x \leq a_1, \\ (x-a_1)/(a_2-a_1) & a_1 < x \leq a_2, \\ (a_3-x)/(a_3-a_2) & a_2 < x \leq a_3, \\ 0 & x > a_3. \end{array} \qquad (4.1)$$

Yamuk bulanık sayılar ise (a_1,a_2,a_3, a_4) 4 rakamla ifade edilir ve üyelik fonksiyonu şu şekildedir:

$$
\begin{array}{ll}
0 & x \le a_1, \\
(x-a_1)/(a_2-a_1) & a_1 < x \le a_2, \\
1 & a_2 < x \le a_3, \\
(a_4-x)/(a_4-a_3) & a_3 < x \le a_4, \\
0 & x > a_4.
\end{array} \tag{4.2}
$$

Örneğin Şekil 4.1'de bir TFN görülmektedir. Bu TFN'ye göre a'dan küçük veya d'den büyük değerler kümeye üye değilken, b-c arasındaki değerler de klasik modellemedeki gibi kümeye tam üyelik sağlamışlardır. Ancak a-b arasındaki veya c-d arasındaki sayılar kümeye belli bir derecede üyelik sağlarlar. Bunlara klasik modellemedeki gibi kesin üyedirler veya kesin üye değildirler demek mümkün değildir. Örneğin "genç insanlar" kümesi ile "yaşlı insanlar" kümesini ele alalım. 20 yaşındaki bir insanın genç olduğu veya 70 yaşındaki bir insanın yaşlı olduğu konusunda hepimiz hemfikiriz, ama gençlikten yaşlılığa kesin bir geçiş olmadığını da biliyoruz. Bu durumda örneğin 40 yaşındaki bir insan "genç insanlar" kümesine de "yaşlı insanlar" kümesine de bir miktar üyelik sağlayacaktır. Görüldüğü gibi bulanık mantık yaklaşımı birçok konuda gerçek hayata daha uygundur.

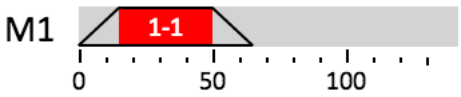


Şekil 4.1 Yamuk bulanık sayı örneği

4.1 Performans Bulanıklaştırma Yaklaşımı

Bu kısımda makinelerin proseslerin başlangıç ve bitişinde %100 performanslı çalışmayabileceği, makinaların performans – zaman grafiğinin bir YBS'ye benzeyeceği varsayılarak bir yaklaşım geliştirilmiştir.

Bu yaklaşımda kullanıcıdan bir operasyonun tamamlanma zamanı için klasik modellemede olduğu gibi tek bir sayı istenmemiş, bunun yerine (a,b,c) gibi 3 adet sayı girilmesi uygun görülmüştür. İlk sayı prosesin, eğer makine her anda %100 performansla çalışsa idi prosesin tamamlanacağı öngörülen süreyi; ikinci ve üçüncü sayılar ise makinanın performansının sırasıyla başlangıçta %100'e çıkacağı ve proses sonlanırken yineleme performansının %100'den düşmeye başlayacağı süreyi ifade eder.



Şekil 4.2 Performans temelli bulanık proses

Örneğin Şekil 4.2'de gördüğümüz TFN bir prosesi temsil ediyor. Başlangıçta makine düşük bir performans ile işleme başlamış, sonra performansı %100'e çıkmış, prosesin bitimi sırasında da yeniden düşmüştür.

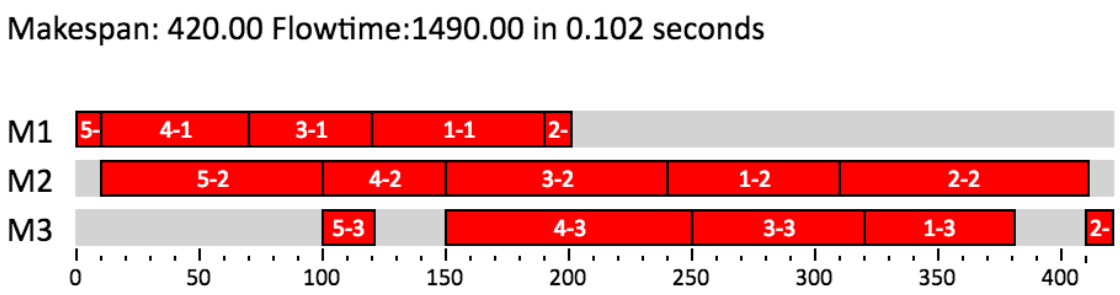
Bu durumdaki prosesleri işleme koyabilmek için durulaştırmak gerekmektedir. Durulaştırma için literatürde birçok yöntem önerilmiştir, ancak burada önemli bir farklılık vardır. Literatürdeki yöntemler TFN'leri tek bir sayıya indirger, bizim ise ihtiyacımız olan TFN'yi bir doğru parçasına indirgemektir. Veya şu şekilde de düşünülebilir, prosesin başlangıç ve bitişi ayrı ayrı durulaştırılmalıdır. Başlangıç ve bitişi de iki ayrı ÜBS olarak düşünülüp durulaştırılabilir.

Bu durumda durulaştırma için M =(l,m,u) şeklinde ifade edilen ÜBS'ler için

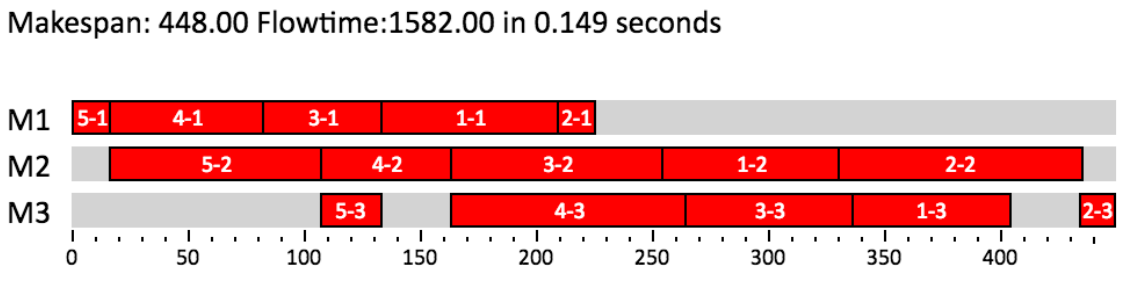
$$M_d = (l + 4m + u)/6 \qquad (4.3)$$

formülü kullanılabilir. [51] Bunun dışında da literatürde birçok durulaştırma yöntemi önerilmiştir, durulaştırma metodu olarak herhangi biri kullanılabiliri. Örneğin Şekil 4.3 ve Şekil 4.4'te görülen çözümler aynı problem için bir sezgisel algoritma (NEH) ile oluşturulmuş çözümlerdir. Görüldüğü gibi çizelge aynı olmasına rağmen gerek C_{max} gerekse Toplam Akış Zamanı farklıdır. Bunun sebebi Şekil 4.4'teki çözümün proses sürelerinin bulanık yaklaşım kullanılarak oluşturulup sonradan durulaştırılarak oluşturulmasıdır.

Burada bulanık yaklaşım kullanmak ile hedefimiz daha kısa tamamlanma zamanları elde etmek değil, daha gerçekçi sonuçlar bulmak olduğundan C_{max}'ın yüksek çıkması istenmeyen bir durum değildir. Önemli olan proses planı ve çizelgeleme yapıldıktan sonra üretime geçildiğinde yapılan hesaptan olabildiğince az sapılmasıdır.



Şekil 4.3 Klasik yaklaşım ile akış tipi çizelgeleme problemi



Şekil 4.4 Performans temelli bulanık yaklaşım ile akış tipi çizelgeleme problemi

4.2 Doluluk Bulanıklaştırma Yaklaşımı

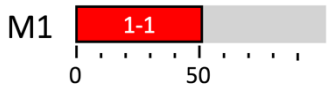
Bu kısımda makinelerin tamamen dolu veya tamamen boş olmayabileceği durumlar varsayılarak bir yaklaşım geliştirilmeye çalışılmıştır. Çizelgeleme problemlerinde prosesleri bölünemez varsayarız. Belli bir anda başlarlar ve belli bir anda biterler. Makinalar de belli bir t anında ya belli bir işi işletirler ve doludurlar veyahut hiçbir işi işletmezlerler ve boşturlar. Ancak günlük hayatta karşılaştığımız bazı çizelgeleme problemlerinde proseslerin bölünebileceği, dolayısıyla makinelerin %100 dolu veya boş olmadığı, örneğin %50 dolu olabileceği düşüncesiyle hareket etmek, dolayısıyla iş tamamlanma zamanlarının bulanık sayı olarak belirlenmesi bize hesaplama açısından kolaylık sağlayabilir.

Örneğin bir fırında ekmek, pide ve lahmacun gibi farklı tipteki yiyeceklerin pişirildiğini düşünelim. Doğal olarak fırının kapasitesi bir ekmekten, pideden veya lahmacundan çok daha büyük olacaktır. Aynı anda birçok yiyecek fırında pişirilebilecektir. Bu durumda fırının herhangi bir anda %100 dolu veya %100 boş oluşundan söz edemeyebiliriz. Fırına konulan yiyeceklerin konulma sürelerinin pişirme sürelerine oranının büyümesi halinde bu durumun bütün işlerin tamamlanma sürelerine etki edeceği aşikardır. Yani örneğin fırının kapasitesi 20 ekmek ise ve bir ekmek 15 saniyede fırına yerleştiriliyorsa bütün ekmekler toplam 5 dakikada fırına yerleştiriliyor demektir. Bir ekmeğin pişme süresi de 50 dakika ise ve fırında pişirmemiz gereken 20 ekmek varsa, ekmekleri sırayla makineye yerleştiririz. Böylece fırının doluluğu 300. saniyeye kadar her 15 saniyede bir %5 kadar artacaktır ve 5. dakikadan itibaren 50. dakikaya kadar %100 seviyesinde kalacaktır. Bu dakikada ilk konulan ekmek pişecek ve sonra yine her 15 saniyede bir pişen ekmekler çıkartılacağından fırının doluluğu %5 azalacaktır.

Toplam yerleştirme süresi = Fırın kapasitesi x Bir birim yiyeceğin yerleştirme süresi (4.4)

Toplam yerleştirme süresi sabit kalacak şekilde fırın kapasitesinin artıp, birim yiyeceğin yerleştirme süresinin düştüğünü düşünelim. Örneğin fırın kapasitesi 200 ekmek olsun ve 1 ekmek 1.5 saniyede yerleştirilsin. Bu durumda fırın doluluğu 5. dakikaya kadar

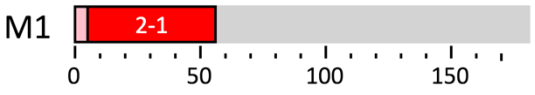
yaklaşık olarak lineer şekilde artacaktır ve yine 50. dakikadan itibaren 55. dakikaya kadar lineere yakın bir grafikle azalacaktır. Hesaplamalarda kolaylık olması açısından bu grafiği lineer kabul edebiliriz.



Şekil 4.5 Normal proses süresi

Örneğin işlerin makinaya konulma sürelerini ihmal etseydik 0. dakikadan itibaren bütün işlerin makineye konulduğunu ve 50. dakikada hepsinin bittiğini varsayabilirdik. Bu durumda Gantt diyagramımız Şekil 4.5'deki gibi olurdu ve işlerin bitiş süresini yeterince iyi simgelemezdi çünkü işlerin hepsi 50. dakikada bitmeyecektir ve 50. dakikada makinenin boşaldığını varsayıp yapacağımız bir çizelgeleme pratikte geçerli olmayacaktır.

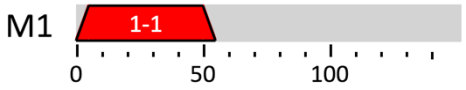
Eğer işlerin makinaya konulma süresini yine kesin (örneğin 5 dakika) bir hazırlık süresi şeklinde ifade etseydik, bu sefer Gantt diyagramımız Şekil 4.6'deki gibi olacaktı ve yine sistemin davranışını yeterince iyi modellememiş olacaktık çünkü makine 55. dakikaya kadar %100 dolu görünecekti. Halbuki gerçekte böyle olmayacaktır ve 50. dakikadan itibaren makina boşalmaya başlayacak, dolayısıyla bizim bir sonraki işi yüklemeye başlamamıza (örneğin bir miktar lahmacun sipârişi) bir miktar hazır olacaktır. Dolayısıyla bu durum dikkate alınarak daha iyi bir çizelgeleme yapılması mümkün olabilir.



Şekil 4.6 Normal hazirlik ve proses süresi

Görüldüğü gibi makinenin tamamen meşgul veya müsait olması dışında belli bir yüzde kadar dolu olabileceğini varsaydığımızda crisp hiçbir yöntem iyi bir modelleme

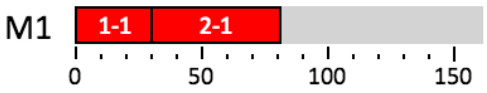
sağlamayacaktır. Daha iyi bir modelleme proses süresini bir bulanık sayı ile ifade etmek ile mümkündür. Örneğin prosesi modellemek için bir yamuk bulanık sayı (TFN) kullanılabilir. Bu sayının üyelik kümesi herhangi bir t anında makinenin ne kadar dolu olduğunu gösterecektir. Örneğin Şekil 4.7'teki yamuk bulanık sayı az önce verdiğimiz fırın örneğindeki ekmeklerin pişmeleri süresince makinenin doluluğunu simgeliyor. Makinemiz 0-5 ile 50-55 dakikaları arasında belli bir yüzdede dolu olacak, 5-50 arasında %100 dolu ve diğer durumlarda %100 boş olacaktır. Bu TFN'yi de (0,5,50,55) şeklinde ifade edebiliriz.



Şekil 4.7 Bulanık proses süresi

4.2.1 Bulanık Proses İşlemleri

Bir çizelgeleme işleminde temelde 3 adet işlem yaparız. Toplama, enbüyükleme ve karşılaştırma işlemi. Normal sayılar için bu işlemler gayet kolaydır. Örneğin 50 birim zamanda ve 30 birim zamanda biten iki prosesin tamamlanma süresi 30+50=80'dir. Örneğin Şekil 4.8'de bu iki prosesin çizelgelenişini görüyoruz.



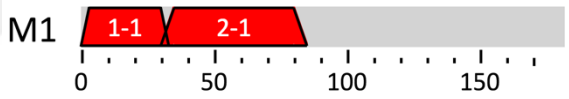
Şekil 4.8 Normal sayılı prosesler ile toplama işlemi

Ancak TFN proses süresi içeren işlemlerde bu toplama işlemi bu kadar basit olmayacaktır. Nasıl olacağını daha iyi anlamak için biraz önceki fırın örneğimize geri dönelim. Fırına yerleştirmesi toplam 5 dk süren ve bir tanesi 50 dakikada pişen ekmeklerden önce başka bir sipârişimiz daha olsun, örneğin lahmacun sipârişi. Bu sipâriş için toplam fırına yerleştirme süremiz 3 dakika ve bir lahmacunun pişme süresi de 30 dakika olsun. Bütün sipârişlerimiz için, sipâriş hacminin fırının kapasitesine eşit

olduğunu varsayıyoruz. Yani örneğin fırın 20 ekmek veya 30 lahmacun kapasiteli ise siparişlerin 20 ekmeklik veya 30 lahmacunluk olduğunu varsayıyoruz. Proseslerimizi daha hızlı göstermek için şu notasyonu kullanabiliriz.

(Bir birimin işlenme zamanı, Toplam yerleştirme zamanı)

Böylece iki işlemimiz (30,3) ve (50,5) olur. İlk sipârişimizin (lahmacun) fırına konmaya başladığını düşünelim. Bu süreç 3 dakika alacaktır. Dolayısıyla ilk lahmacunun 30. dakikada pişmesinden itibaren 33. dakikaya kadar bütün lahmacunlar pişip fırını boşaltacaklardır. Bu süreçte ekmeklerin fırına yerleştirilmesi başlayabilir ancak bitmeyecektir. Çünkü ekmeklerin fırına yerleştirilmeleri 5 dakika sürecektir. Dolayısıyla ekmeklerin fırına girmeleri 30-35. dakikalar arasında gerçekleşecektir. Dolayısıyla ekmeklerin pişmeleri de 80-85. dakikalar arasında tamamlanacaktır. yani bütün prosesin bitiş süreci (80,5)'lik tek bir proses gibi düşünülebilir. Bu işlemlerin tek makinada (fırın) işlenme zamanının Gantt diyagramı da Şekil 4.9'teki gibi olacaktır.

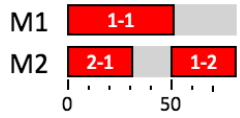


Şekil 4.9 TFN toplama işlemi

Tersine düşünüldüğünde de cevap aynıdır. Yani (30,3)+(50,5)=(50,5)+(30,3)=(80,5) olur. Burada da önce ekmekler pişecek ve pişme işleminin bitişi 50-55. dakikalar arasında olacaktır. Bu demektir ki lahmacunların konulması da 50. dakikada başlayacak ancak 53. dakikada tamamlanmayacak, 55. dakikaya kadar sarkacaktır çünkü fırın makinesini, ekmek işlemi kendisini terkettiği yüzdelik dilim kadar boştur ve herhangi bir t anında ancak bu kadarlık bir yüzde kadar diğer işlemin başlamasına açıktır. Bu işlemin formülasyonu da t_i, i. işlemin biriminin tamamlanma zamanı ve f_i de i. işlemin makineye yerleştirme süresi olmak kaydıyla şu şekilde tanımlanabilir.

$$(t_1,f_1)+(t_2,f_2)+....+(t_n,f_n) = (t_1+t_2+...+t_n+,\max(f_1,f_2,...,f_n)) \quad (4.5)$$

Bir diğer gerekli işlem de enbüyükleme (enbüyükleme) işlemidir. Buna temelde 2 yerde ihtiyaç duyarız. Birincisi iş beklemesi ile makine beklemesinin enbüyüklenmesinde, bir diğeri de çizelgelemenin toplam tamamlanma süresinin hesaplanmasında bütün makinaların bitiş sürelerinin en büyüğünün alınmasında karşımıza çıkar. Bu enbüyükleme işlemi, crisp sayılar için oldukça kolaydır.

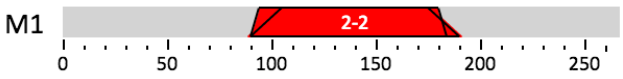


Şekil 4.10 Normal sayılar ile enbüyükleme işlemi

Örneğin Şekil 4.10'da 2. makine 30 birim zamanda boşalmasına rağmen 1. makinedeki işin bitip 2. makineye geçisi beklenmiş ve iş ancak 50. birim zamanda başlayabilmiş.

Ancak TFN'ler düşünüldüğünde maks(A,B)=C olacak şekilde iki sayının enbüyüklenmesi bir üçüncü sayıyı verebilir. Enbüyüklemenin mantığı herhangi bir işin başlamasının, diğer birden fazla işin bitmesine bağlı olması durumunu modellemek olduğundan ana düşüncemiz herhangi bir t anında iki işten devam etme yüzdesinin fazla olanının değerini almak olmalıdır. Dolayısıyla örneğin (50,5) ve (40,10) değerlerinde iki işin enbüyüklenmesini düşünürsek ilk iş bitmeye 50'de başlayıp 55'te sonlanacağından ve diğer iş bitmeye 40'ta başlayıp 50'de sonlanacağından ilk işin burada daha belirleyici olduğunu görürüz. Dolayısıyla bu işlem için şöyle bir formülasyon geliştirebiliriz:

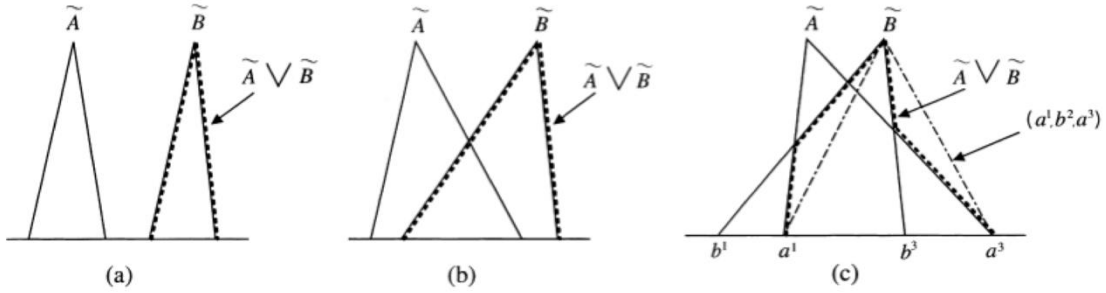
$$\max((t_1,f_1), (t_2,f_2), \dots , (t_n,f_n))=(\max(t_1,t_2,\dots,t_n) , \max(t_1+f_1,t_2+f_2,\dots,t_n+f_n)-\max(t_1,t_2,\dots,t_n)) \quad (4.6)$$



Şekil 4.11 TFN enbüyükleme örneği

Ancak bu formülasyon her zaman doğru sonuç vermeyecektir. Örneğin Şekil 4.11'de farklı makinede işleyen iki prosesin izdüşümünün üstüste gelmiş halini görmekteyiz. Gördüğümüz gibi proseslerin bitmeye başlama noktaları $t_1>t_2$ ancak bitmelerin sona

erme noktaları $t_2+f_2 > t_1+f_1$ olduğundan enbüyüklenmeleri bizim daha önce öngördüğümüz 4 noktalı TFN'lerden daha farklı olacaktır. Bu noktada oluşacak hata miktarlarını şimdilik ihmal ediyorum. Aslında yapılması gereken gerçek işlem, Şekil 4.12'de de görülebileceği gibi literatürde V işlemi olarak geçen işlemdir.



Şekil 4.12 Üçgensel bulanık sayılarda V işlemi [52]

Gerekli bir diğer işlem de karşılaştırma işlemidir. Karşılaştırma işleminin enbüyüklemeden farkı, iki seçenekten birisinin seçilmesinin gerekli olduğu yerlerde kullanmamızdır. Örneğin bir C_{max} probleminde iki farklı çizelgelemeyi karşılaştırmak için ikisinin de bitme zamanlarını kıyaslamak gereklidir. Bunun için de çizelgelemenin son kısmını bir üçgensel bulanık sayı gibi düşünüp ağırlık merkezini alabiliriz. Zaten üçgenimiz dik üçgen olduğundan ağırlık merkezi (t_1,f_1) sayısı düşünüldüğünde $t_1+f_1/3$ değerini alacaktır.

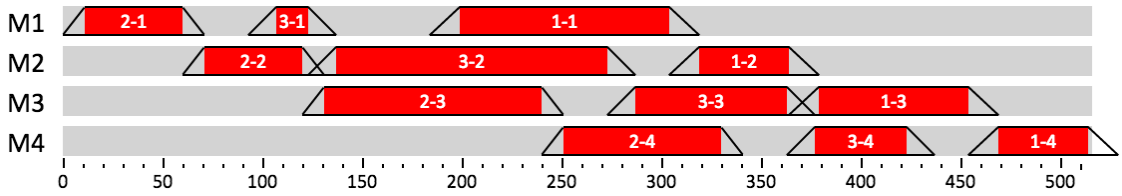
$$(t_1,f_1)>(t_2,f_2) <-> (t_1+f_1/3)>(t_2+f_2/3) \quad (4.7)$$

Örneğin (50,5) ve (45,15) zamanında biten iki farklı çizelgelememiz olsun. Buna göre $50+5/3<45+15/3$ olduğundan bu iki çizelgelemeden ikincisini büyük kabul edebiliriz.

Daha farklı bir kıyas yöntemi de çizelgelemelerin ne kadar süre diğerlerinden daha yüksek oranda bitmiş oldukları temelli geliştirilebilir. Bu durumda da örneğin 2 proses için düşünürsek P_1'in bitmiş olup P_2'nin devam ettiği alan ile P_2'nin bitmiş olup P_1'in devam ettiği alandan hangisi fazlaysa o proses daha uzun sürer diyebiliriz. Böyle bir kıyas yöntemi daha akla uygun geliyor. Bu yöntemin formülasyonu ise şu şekildedir:

$(t_1,f_1)>(t_2,f_2)$ <-> $(2t_1+f_1)>(2t_2+f_2)$ (4.8)

Ancak eğer problemimiz bir TFT enküçüklenmesini hedefleyen akış tipi çizelgeleme problemi ise kıyaslama yöntemimiz daha farklı olmalıdır çünkü o zaman bütün işlerin bitiş zamanlarının toplamı önemli olacaktır. Örneğin Şekil 4.13'da bir akış tipi beklemesiz (no-wait) çizelgeleme problemine doluluk bulanıklaştırma yaklaşımı ile oluşturulmuş bir çözüm görülüyor. Problem beklemesiz olduğu için her işin ilk operasyonu son operasyonun başlangıcına göre ileri kaydırılıyor ve bunu yaparken bulanık yaklaşım da gözönünde bulunduruluyor.



Şekil 4.13 Beklemesiz akış tipi bir çizelgeleme probleminde FL yaklaşımı

BÖLÜM 5

ÇİZELGELEME PAKET PROGRAMI

Bu bölümde çalışmadaki teorik çalışmanın pratiğe dökülmesi için geliştirilen açık kaynak kodlu Çizelgeleme paket programı anlatılmıştır. Program internet tarayıcısına sahip herhangi bir bilgisayarda çalıştırılabilir. Javascript ile yazılmıştır. Küçültülmüş bir versiyonu da Apple firmasının mobil cihazları için tasarlanmıştır.

5.1 Genel Bilgiler

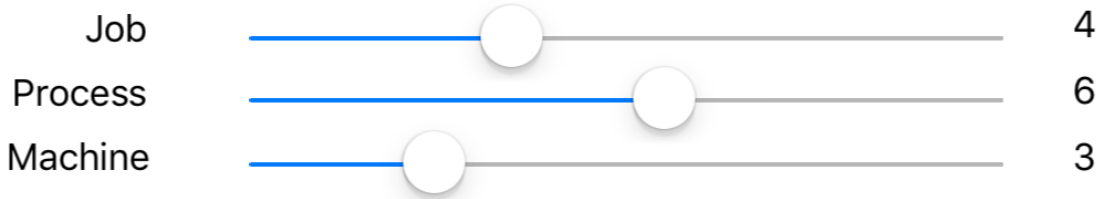
Program iki ayrı platformda yazılmıştır. Kapsamlı program internet tarayıcılarında çalışmak üzere HTML – CSS – Javascript dilleri kullanılarak geliştirilmiştir. Ana Javascript kütüphanesi olarak açık kaynak kodlu MIT lisanslı jQuery kütüphanesi kullanılmıştır. Geliştirme ortamı olarak da Jetbrains firmasının Intellij IDEA isimli yazılımı öğrenci lisansı ile kullanılmıştır.

Mobil platformlar için düşünülen program ise Apple firmasının mobil cihazlarında çalışması düşünülerek iOS işletim sistemi için Objective-C dilinde kodlanmıştır. Program kodlanması için geliştirme ortamı olarak XCode programı kullanılmıştır. Bütün bunları kullanabilmek için Apple firmasının geliştirici programına ücretli olarak kayıt olunmuştur.

5.2 Mobil Cihazlar için Program

Program bahsedildiği gibi Apple firmasının ürettiği iPhone ve iPad cihazları için tasarlanmıştır. Örnek bir iPad ekran görüntüsü Şekil 3.3'te ve örnek bir iPad ekran görüntüsü Şekil 3.5'te görülebilir.

Program 3 ana ekrandan oluşmaktadır: Giriş ekranı, Problem oluşturma ekranı ve Seçilen algoritma ile problem çözme ekranı. Problem oluşturma ekranında oluşturulacak problemin çeşitli özellikleri belirlenmektedir. Öncelikle problemin kaç iş, kaç proses ve kaç makine içereceği belirlenir. Şekil 5.1'de bu kısım görülmektedir. İş, proses ve makine sayısı 10 ile sınırlandırılmıştır.



Şekil 5.1 İş – Proses – Makina seçme bölümü

Daha sonra iş, proses ve makine sayıları belirlenen problemin her bir prosesin her bir makinede çalışma süresini içeren tablosu oluşturulmalıdır. Program kullanıcıya, isteğe göre boş bir tablo veya rastgele değerler ile doldurulmuş bir tablo sunar. Daha sonra kullanıcı tabloyu istediği gibi değiştirebilir.

	Job-1					Job-2					Job-3					Job-4				
	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5
P1	X	30	X	X	X	70	80	120	180	80	X	90	40	50	130	20	130	110	110	X
P2	X	50	X	190	X	190	30	60	120	X	60	50	70	110	110	60	180	140	X	120
P3	180	80	X	180	90	40	10	30	X	20	X	100	30	50	10	X	60	130	X	190
P4	160	200	40	120	X	10	160	80	20	200	90	20	30	60	40	190	X	110	90	130
P5	40	80	140	20	X	180	X	X	30	200	X	190	X	150	180	180	120	120	X	X
P6	100	20	140	150	100	20	130	X	50	150	X	100	X	40	20	20	180	X	70	X
P7	X	80	160	X	50	90	50	160	170	60	80	X	X	X	X	X	110	140	X	100

Şekil 5.2 Problem tablosu oluşturma bölümü

4 iş 7 proses 5 makineli bir problem için kullanıcının problem oluşturma tablosu Şekil 5.2'de görülmektedir. Burada herhangi bir kutucukta 'X' ifadesinin bulunması, o prosesin o makinada çalışmadığı anlamına gelir. Bunun yerine tercihen, çok büyük bir sayı da yazılabilir. Programın rastgele verdiği sayılar ise 20 ile 200 arasında 10 ile bölünebilen sayılar arasından Uniform (tekdüze) dağılım kullanılarak oluşturulmaktadır.

Daha sonra çözüm için tercih edilecek algoritma seçilir. Açılan yeni ekranda algoritmanın parametreleri yer almaktadır. Örneğin yine Şekil 3.5'te Genetik Algoritma için parametrelerin seçilip çözüm elde etme ekranı görülmektedir. Parametreler seçildikten sonra 'Solve' butonuna basılıp beklenir. Bu sürede algoritmanın bulduğu en iyi çözüm ve anlık çözümler grafik olarak görülebilir. Bu görüntü de Şekil 3.6'da görüldüğü gibidir.

Algoritma çalışmasını tamamladıktan sonra 'Show solution' butonuna basılırsa algoritmanın elde ettiği en iyi çözümün Gantt diyagramı ekrana gelir. Bu diyagramda her satır bir makineyi temsil eder, her satırın sonundaki sayı o makinenin bu çizelge uygulandığında ne kadar süre boş kalacağını gösterir. Her iş için farklı bir renk atanmıştır, işi temsil eden kısmın içinde yazan sayı da o prosesin bahsi geçen işin kaçıncı prosesi olduğunu simgeler. Böyle bir problem çözümü Şekil 5.3'te görülmektedir.

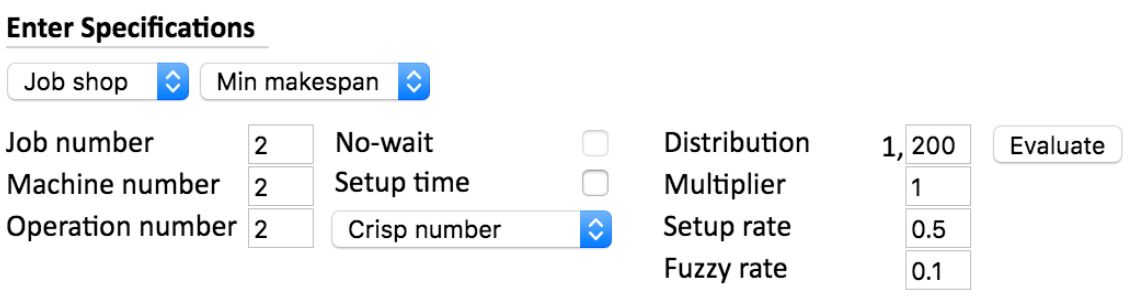
Şekil 5.3 10 iş – 3 proses – 10 makinalı bir problemin çözümü.

Program ayrıca iPhone ekranlarının küçüklüğü sebebiyle iPhone sürümleri için ufak görsel düzeltmelere sahiptir. Bunlar da Şekil 3.3'te görüldüğü gibidir.

5.3 Bilgisayarlar için Program

Programın esas sürümü bilgisayarlar için yazılan İnternet tarayıcısı tabanlı olan versiyonudur. Bu sürüm mobil versiyondan daha sonra geliştirilmiştir. Geliştirilme sebebi de dağıtım, çalıştırma ve modifiye etme kolaylığı ile tek bir cihaz türüne bağımlı kalmama isteğidir.

Bu sürüm daha sonra geliştirildiğinden mobil versiyona göre çok daha kapsamlıdır. Başlangıçta problemin atölye tipi mi akış tipi mi olduğu seçilir. Daha sonra eğer problem akış tipi ise enküçüklenecek etmenin 'Toplam akış süresi' veya tamamlanma zamanı olacağı seçilir. Daha sonra akış tipi çizelgeleme problemleri için problemin beklemesiz olma opsiyonu veya sıra bağımlı hazırlık süresi ekleme opsiyonu seçilebilir. İş, proses ve makina sayısı seçildikten sonra verilecek rastgele değerler için Uniform dağılımın sınırları değiştirilebilir. Uniform dağılım, rastgele seçilen sayının U(a,b) şeklinde verilen a ve b sınırları arasında aynı uzunluktaki herhangi iki aralığın içinde yer alma olasılığı eşit olan bir olasılık dağılımıdır. Programda her zaman için a=1 olup kullanıcı b'yi belirler. Bunun yanısıra dağıımdan çıkan sayıları belli bir katsayı ile çarpıp kullanma şansı da vardır. Yani örneğin 10 ile 200 arasındaki 10'a bölünen sayıları kullanmak istiyorsak b=20 seçilip katsayı olarak da 10 seçilebilir. Ayrıca sıra bağımlı hazırlık süreleri için proses zamanlarının belli bir yüzdesi de seçilebilir. Örneğin bu değer için 0.5 seçildiğinde hazırlık süreleri 5 ile 100 arasında 5'e bölünebilen sayılardan oluşacaktır.



Şekil 5.4 Problem özellikleri belirleme bölümü

Ayrıca bulanık yaklaşım için makine – proses süresi tablosu oluşturma opsiyonu da vardır. Bu durumda tercih edilen bulanık yaklaşımına göre her bir proses için 2 veya 3 değer girilmesi gerekecektir. Girilen ekstra değerler 4. bölümde anlatıldığı gibi prosesin bulanıklığı konusunda belirleyici olacaktır. Sıra bağımlı hazırlık süreleri için olduğu gibi bulanıklık zamanları da proses zamanlarının belli bir yüzdesi olarak rastgele belirlenebilir. Bütün bu belirlenen zamanlar sonradan istenildiği gibi değiştirilebilir şekildedir. Şekil 5.4'te problem özelliklerinin belirlenme ekranı görülebilir.

Şekil 5.5'te ise problemin proses zamanları belirleme bölümü görülmektedir. Bu bölüm 3 iş – 4 proses – 3 makineli, sıra bağımlı hazırlık zamanlı bir problem içindir. U(1,200) dağılımı kullanılmış, katsayı 1 olarak belirlenmiştir. Üstteki tablolar soldan sağa her bir iş için bir adet olmak üzere iş sayısı kadardır. Her bir satırı ise bir makineyi simgeler, bir tablonun bir sütunu ise bir prosesi simgeler.

Process Time Table

Job1

	P1	P2	P3	P4
M1	70	163	130	161
M2	1	97	155	7
M3	175	96	67	167

Job2

	P1	P2	P3	P4
M1	60	72	119	118
M2	70	61	159	51
M3	53	148	33	58

Job3

	P1	P2	P3	P4
M1	73	120	53	135
M2	109	133	1	159
M3	129	139	132	106

Setup Time Table

Machine1

1\2	J1	J2	J3
J1	0	16	55
J2	73	0	76
J3	19	77	0

Machine2

1\2	J1	J2	J3
J1	0	6	22
J2	46	0	87
J3	39	26	0

Machine3

1\2	J1	J2	J3
J1	0	16	64
J2	79	0	88
J3	13	20	0

Şekil 5.5 Proses zamanları belirleme bölümü

Alttaki tablolar ise sıra bağımlı hazırlık süreleri içindir. Her bir makine için bir tablo olmak üzere her tablo o makine için işlerin sıralamasına göre aralarına ne kadar hazırlık süresi ekleneceğini gösterir. Tablonun satır kısmındaki değer önceki işi, sütun

kısmındaki değer ise sonraki işi simgeler. Örneğin Şekil 5.5 için 1. makinede 1. işten sonra 2. iş geldiğinde aradaki gecikme S_{12} = 16 birim zaman olmaktadır.

Process Time Table

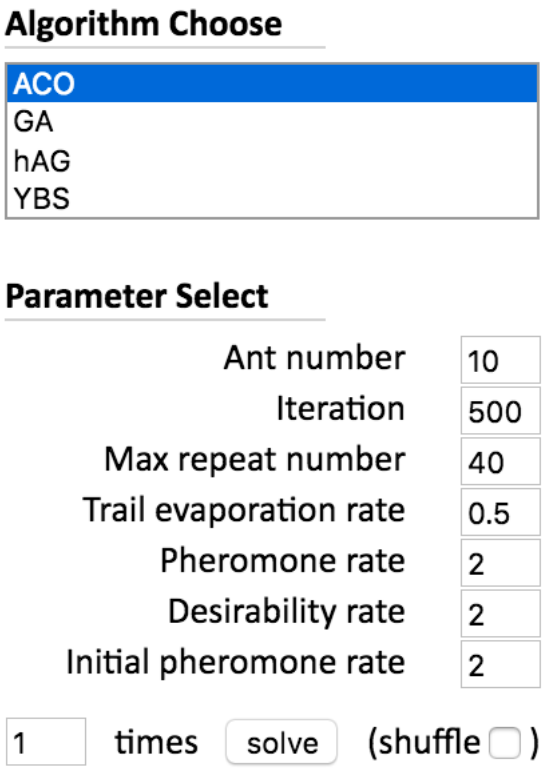
Job1

	P1			P2			P3			P4		
M1	145	2	6	116	1	4	185	4	2	28	9	5
M2	44	1	6	119	6	8	34	8	5	129	5	9
M3	89	2	5	154	4	3	97	8	7	63	6	4

Şekil 5.6 Bulanık proses zamanları için örnek

Ayrıca Şekil 5.6'da performans temelli bulanık yaklaşım için proses süresi seçimi bölümü gösterilmiştir. Görüldüğü gibi her proses için birden fazla değer vardır, ilk değer prosesin normal çalışma süresini, 2. ve 3. değerler ise bulanıklık seviyesini simgelemektedir.

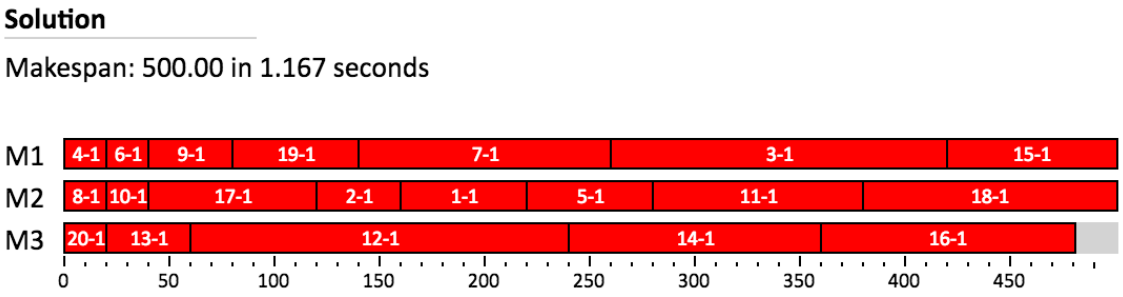
Daha sonra Algoritma seçim bölümünden bir veya daha fazla algoritma seçilip çözüm yaptırılabilir. Seçilen problem türü ve özelliklerine göre program uygun algoritmaları gösterecektir. Algoritma seçimi yapıldıktan sonra algoritmanın parametreleri seçilir. Sezgisel algoritmaları kıyaslamakta kullanılmak üzere bir problemi birden fazla çözdürme opsiyonu da bulunmaktadır.



Şekil 5.7 Algoritma seçim ve parametre belirleme bölümleri

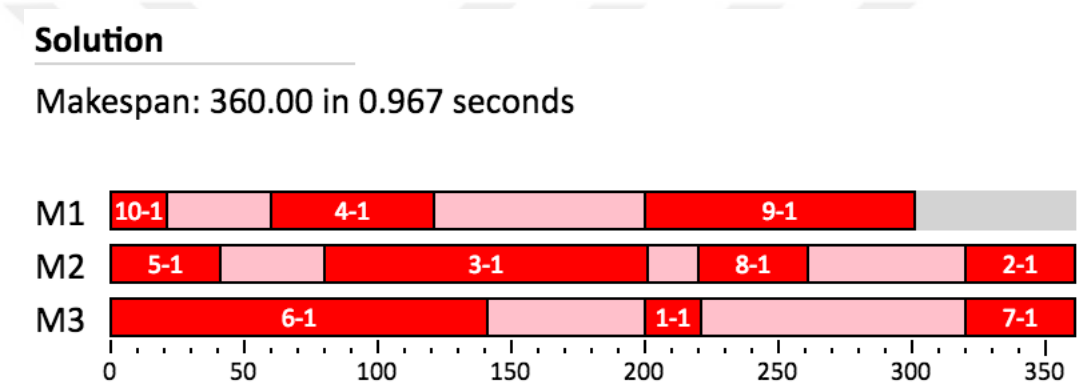
Şekil 5.7'de algoritma seçimi ve parametre belirleme kısımları görülmektedir. Kullanıcı KKO algoritmasını seçtiği için o algoritmaya ait parametreler gösterilmiştir. Eğer birden fazla algoritma seçilip, problem de yine birden fazla çözülsün denilirse, 'shuffle' (karıştırma) seçeneği aktif hale getirildiğinde program her seferinde başka bir problem ile çözüm arayacak, algoritmaları toplamda belli sayıda farklı problem ile kıyaslamış olacaktır.

Programın ana sürümünde, mobil sürümünde olduğu gibi iş veya proses sayısı 10 ile kısıtlanmamıştır. Bu yüzden mobil sürümde kullanılan Gantt diyagramının tasarımında değişikliğe gidilmiş, renk kullanımı kaldırılmıştır. Bunun yerine bütün işler kırmızı renkte gösterilmiş, iş numarası proses numarası ile birlikte işi simgeleyen bölümün üzerine yazılmıştır. Örneğin Şekil 5.8'de 20 iş içeren bir problem için Gantt diyagramı gösterilmiştir.



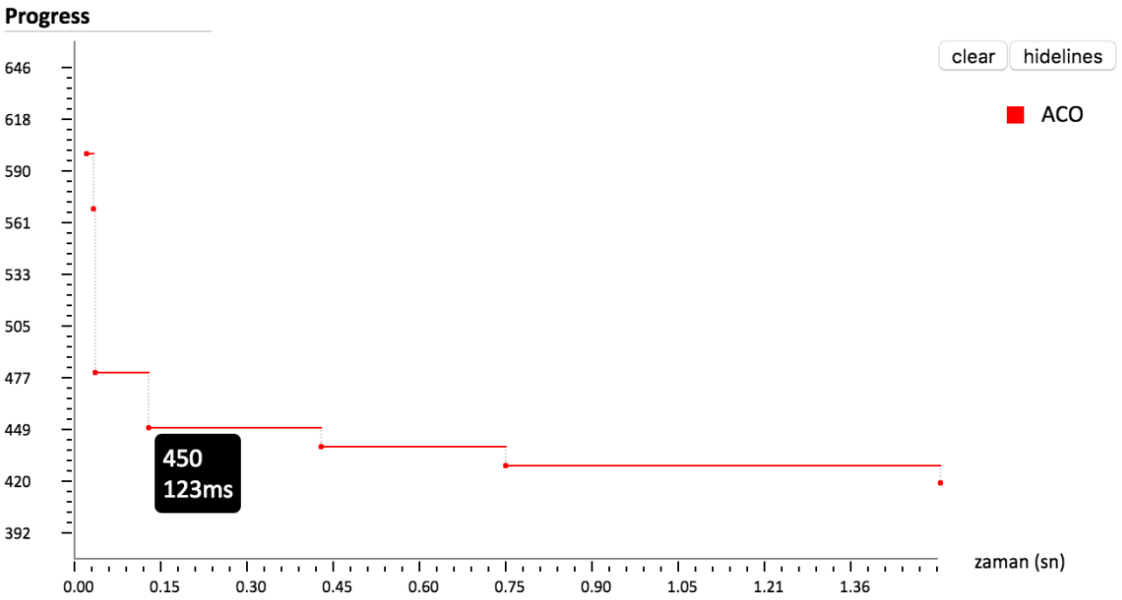
Şekil 5.8 20 işli bir çizelgeleme problemi çözümü

Sıra bağımlı hazırlık zamanlı çizelgeleme problemlerinde hazırlık süreleri ise pembe renkte gösterilmiştir. Örneğin Şekil 5.9'da 10 işli böyle bir problem görülmektedir.



Şekil 5.9 Sıra bağımlı hazırlık zamanlı bir problem çözümü

Algoritmaların performanslarını ölçmek amacıyla ayrıca bir grafik gösterimi de geliştirilmiştir. Bu gösterilmin avantajı algoritmanın sadece bulduğu en son sonucu değil, ayrıca bu sonucu bulma sürecinde ne kadar zamanda ne kadar iyi sonuçlar bulabildiğini de inceleme imkanı vermesidir. Örneğin Şekil 5.10'da bir KKO algoritmasının çalışma süresi içindeki performansı görülmektedir. İmleç ile grafiğin üzerine gelindiğinde algoritma çalışırken bulunan sonuçların bulunma anı ve C_{max} değeri gösterilmektedir. Örneğin yine Şekil 5.10'da 0.12 sn içinde 450 birim zamanlık bir çözüm bulunabilmiştir.



Şekil 5.10 Algoritma performans – zaman grafigi

Algoritma birden fazla çalıştırıldığında veya başka algoritmalar da aynı problem için olmak kaydıyla çalıştırıldıklarında aynı grafiğin üzerinde iz bırakırlar. Sağdaki 'clear' butonuna tıklanırsa veya problem değiştirilirse grafik sıfırlanır. Örneğin Şekil 6.36'da YBS ve GA kıyaslaması bu şekilde yapılmıştır. Yine sağdaki 'hideline' butonuna basıldığında bulunan noktalar birbiriyle bağlanmaz, yalnızca noktalar gösterilir.

Programda ayrıca sayısal olarak algoritmaların kıyaslanmasına yönelik bir kısım da vardır. Bu bölümde üç ayrı ölçüt gözönünde bulundurulmuştur. Bunlar sırasıyla; yapılan denemelerin yüzde kaçında hangi algoritmanın daha iyi sonuç bulduğu (SR - Success Rate), algoritmanın ortalama olarak bulunan en iyi sonuçtan sapma yüzdesi (ARPD - Average Relative Percentage Deviation) ve algoritmanın ortalama çalışma süresi (CPU) olarak bir tabloda herbiri bir sütunda gösterilirler.

Örneğin Şekil 5.11'de 4 tane algoritmanın bir problem üzerinde kıyaslanmasını görmekteyiz. Bu problem özelinde bütün denemelerde en iyi sonucu GA vermişken, ortalama sapma olarak GA'ya en yakın algoritma %7.75 sapma ile hACO-GA olarak ön plana çıkmış. En az süre kullanan algoritma ise ortalama 0.85 sn ile YBS algoritması olmuş.

Result			
5-3-J-50	SR	ARPD	CPU
ACO	0.00	10.35	1.75
GA	100.00	0.00	2.29
hACO-GA	0.00	7.75	4.33
AIS	0.00	61.12	0.85

Şekil 5.11 Algoritmaların sayısal kıyaslanma tablosu

BÖLÜM 6

DENEYSEL ÇALIŞMALAR

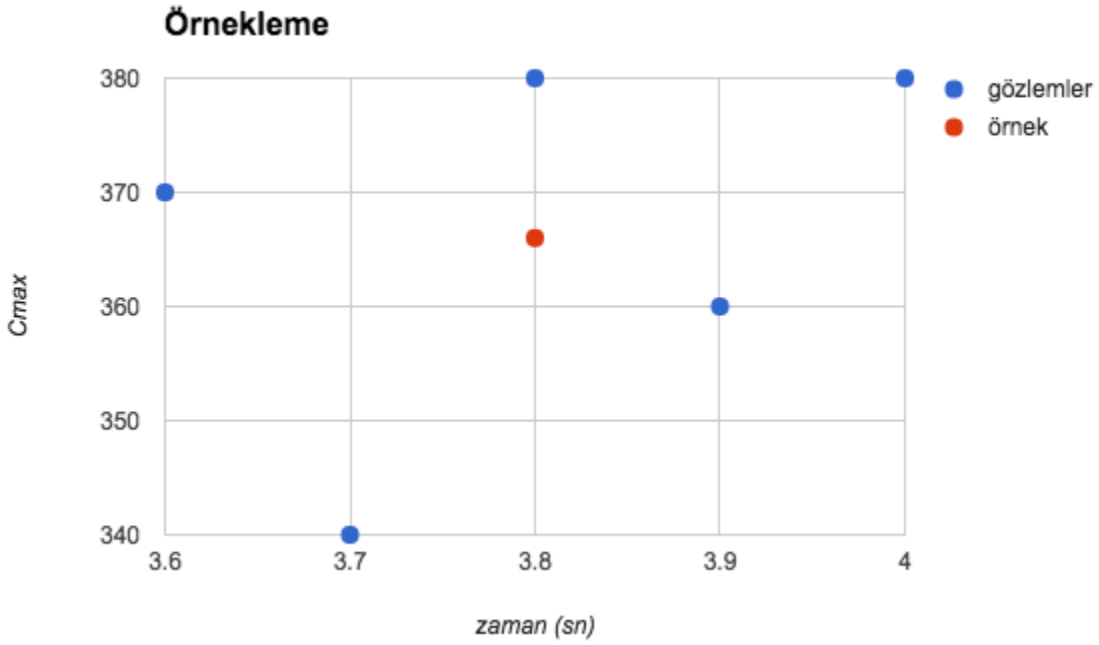
Yapılan testler parametrelerin değiştirilmesinin programın çalışma süresi ve bulunan en iyi çözümün C_{max} değerini nasıl etkilediğini gösterdi. Deneysel çalışma belli aşamalar halinde gerçekleştirildi. Öncelikle çeşitli algoritmaların parametrelerinin, o algoritmanın performansına etkileri araştırıldı. Daha sonra farklı ölçekteki problemler üzerinde algoritmaların birbirleriyle kıyaslamaları yapıldı.

6.1 Karınca Kolonisi Optimizasyonu Sonuçları

Bu bölümde parametrelerinin değiştirilmesinin Karınca Kolonisi Algoritması üzerine etkileri incelenmiştir. Bütün parametreler tek tek test edilmiştir, dolayısıyla bir parametre test edilirken diğerleri sabit tutulmuştur. Yapılan her beş gözlem sonucunun ortalaması tek bir örnek olarak alınmıştır. Örnekleme işlemi Şekil 6.1'de olduğu gibidir. Şekilde beş bağımsız çizelgenin ortalama işlemci zamanları ve C_{max} sonuçlarının ortalaması tek bir gözlem olarak alınmıştır.

6.1.1 Karınca Sayısını Değiştirmenin Etkileri

Bu karşılaştırmada karınca sayısının çalışma süresi ve C_{max} üzerindeki etkileri araştırılmıştır. Bütün diğer parametreler sabit tutulmuş ve örnek büyüklüğü *n=5* olarak alınmıştır, Tablo 1'de test sonuçları görülebilir.

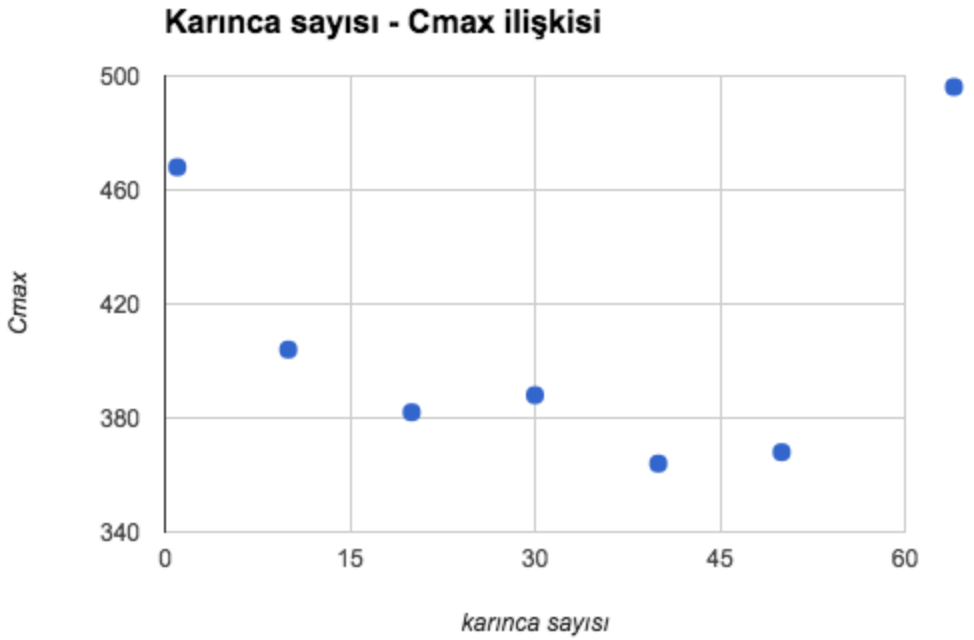


Şekil 6.1 Örnekleme

Çizelge 6.1 Karınca sayısını değiştirmenin C_{max} ve çalışma süresi üzerindeki etkileri

Karınca Sayısı	Çalışma süresi (sn)	C_{max}
1	0.12	468
10	0.74	404
20	1.36	382
30	2.1	388
40	2.92	364
50	3.56	368

Şekil 6.1, Çizelge 6.1'deki C_{max} verilerinin grafik halinde gösterimidir. Görüldüğü gibi karınca sayısını artırmak başta C_{max}'ı düşürse de bir süre sonra etki etmemektedir.



Şekil 6.2 Karınca sayısı ve C_{max} ilişkisi

Şekil 6.3, Çizelge 6.1'deki çalışma süresi verilerinin grafik halidir. Karınca sayısı arttıkça programın çalışma süresi lineer bir şekilde uzamaktadır.

Şekil 6.3 Karınca sayısı ve çalışma süresi ilişkisi

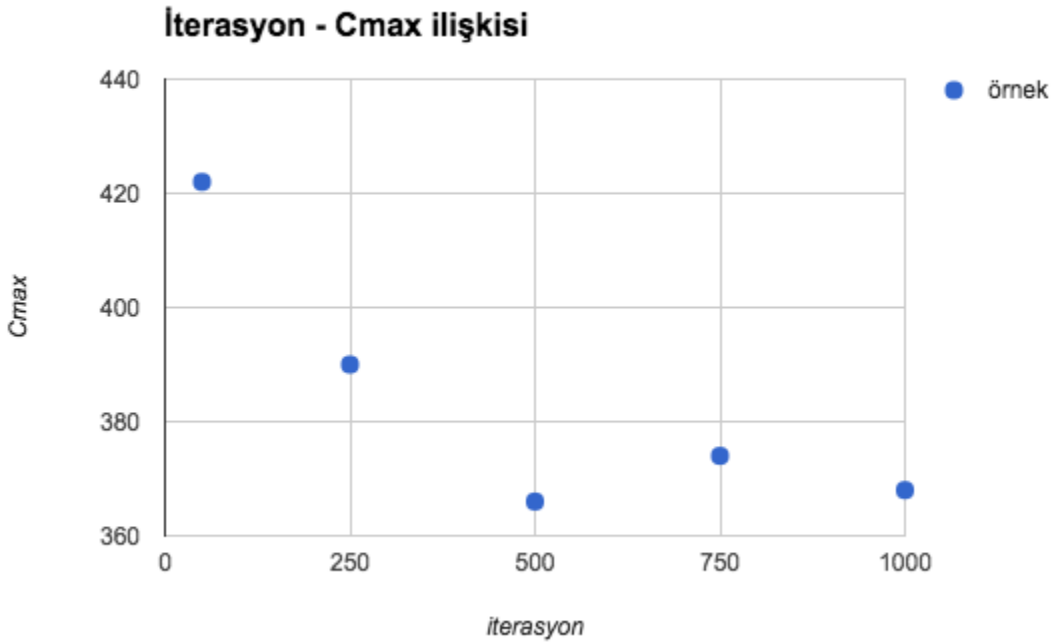
6.1.2 İterasyon Sayısını Değiştirmenin Etkileri

Bu karşılaştırmada iterasyon sayısının çalışma süresi ve C_{max} üzerindeki etkileri araştırılmıştır. Diğer parametreler sabit tutulmuş ve örnek büyüklüğü *n=5* olarak alımıştır. Ölçülen 5 ayrı iterasyon sayısı 50, 250, 500, 750 ve 1000 olarak belirlenmiştir. Karınca sayısı 5, en fazla yineleme sayısı 10, iz buharlaşma oranı 0.75, feromon oranı, çekicilik oranı ve başlangıç feromon oranı 2 olarak alınmıştır. Çizelge 6.2 test sonuçlarını gösterir.

Çizelge 6.2 İterasyon değişikliğinin C_{max} ve çalışma süresi üzerindeki etkileri

İterasyon	Çalışma süresi (sn)	C_{max}
50	0.48	422
250	1.78	390
500	3.8	366
750	5.82	374
1000	7.16	368

Şekil 6.4, iterasyon sayısı ve C_{max} ilişkisini gösteriyor. Şekilde görüldüğü gibi aralarında ters orantı var. 500 iterasyondan sonra C_{max} üzerinde belirgin bir iyileştirme etkisi görülmemiş.



Şekil 6.4 İterasyon ve C_{max} ilişkisi

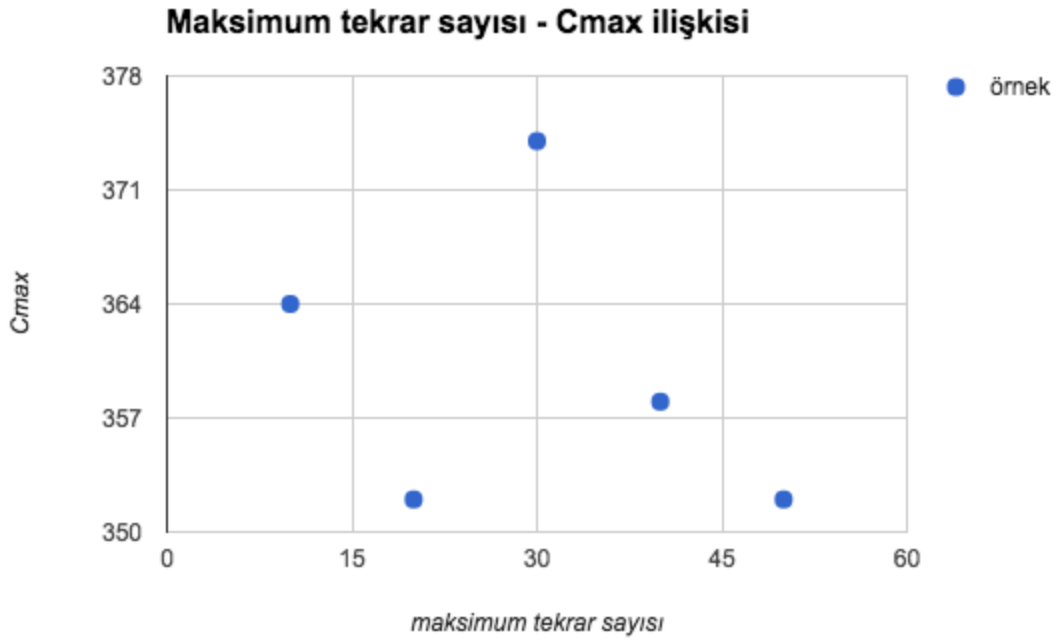
Şekil 6.5, Çizelge 6.2'deki sonuçların çalışma süresi ile ilgili kısmının grafik halidir. Zaman verisi saniye cinsindendir. Görüldüğü gibi iterasyon sayısı ve çalışma süresi arasında doğru orantı vardır.

Anlaşılacağı üzere iterasyon ve karınca sayısını artırmanın C_{max} ve çalışma süresi üzerindeki etkisi benzer oluyor. İkisi de çalışma süresi ile doğru orantılı ve bulunan çözümün C_{max} değeri ile ters orantılı.

Şekil 6.5 İterasyon ve çalışma süresi ilişkisi

6.1.3 En Fazla Yineleme Sayısını Değiştirmenin Etkileri

Bu karşılaştırmada, en fazla yineleme sayısını (maxNC) değiştirmenin C_{max} ve çalışma süresi üzerindeki etkileri incelenmiştir. Leung [46] makalesinde en fazla yineleme sayısının düşük değerlerinin düşük C_{max} sonuçları verdiğini söylemiştir. Şekil 6.6'da maxNC ile C_{max} ve programın çalışma süresi arasındaki ilişkiler görülmektedir. Bu çalışmada yapılan testlerde maxNC ile C_{max} arasında anlamlı bir korelasyon bulunamamıştır. Problem büyüdükçe çözümlerin iyileştirme süreleri uzayacağından daha büyük maxNC değerlerinin iyi sonuçlar vereceği tahmin edilebilir. Dolayısıyla problemin büyüklüğüne göre uygun maxNC değeri yapılacak testlerle belirlenebilir. Ayrıca yapılan testlerde maxNC'nin yükselmesi ile çalışma süresi bir miktar artmıştır ancak Şekil 6.7'de de görülebileceği gibi oransal olarak önemli bir fark ortaya çıkmamıştır.

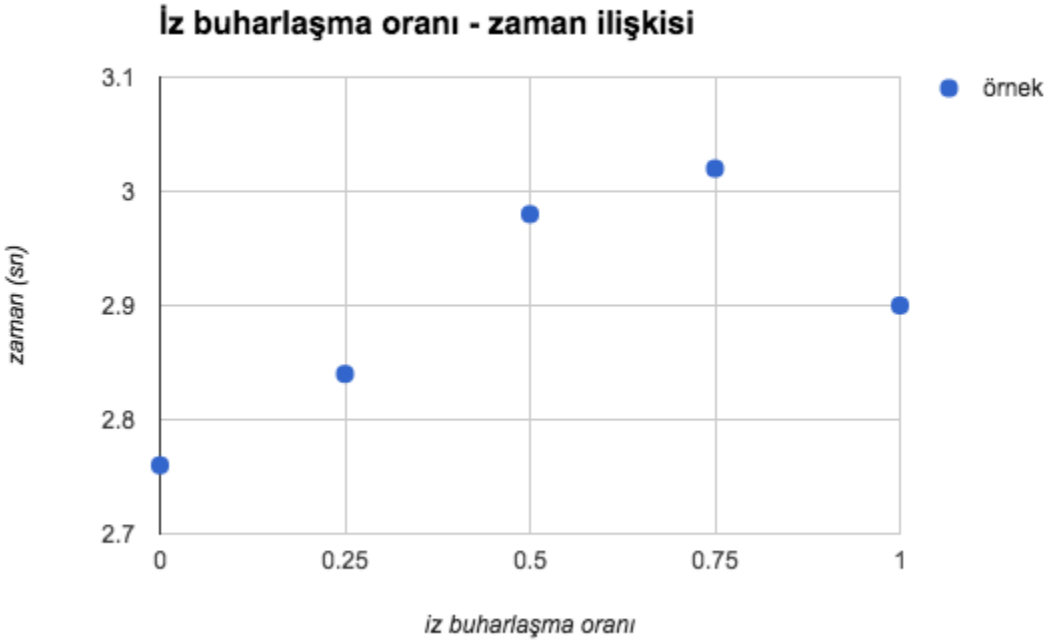


Şekil 6.6 En fazla yineleme sayısının C_{max} ile ilişkisi

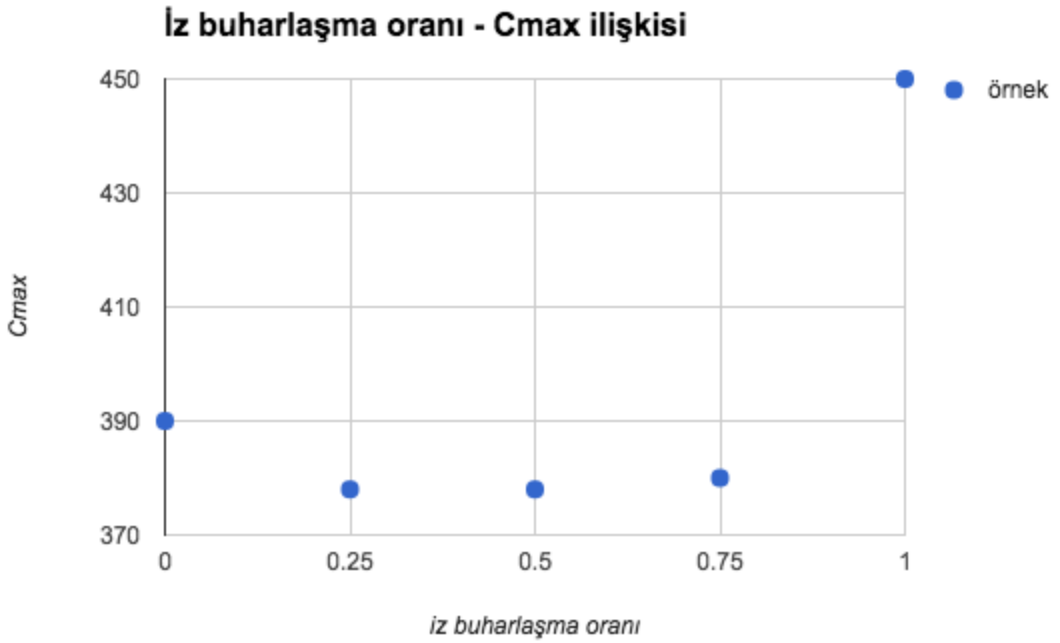
Şekil 6.7 En fazla yineleme sayısının çalışma süresi ile ilişkisi

6.1.4 İz Buharlaşma Oranını Değiştirmenin Etkileri

Bu karşılaştırmada, iz buharlaşma oranının (rho) C_{max} ve çalışma süresi üzerindeki etkileri incelenmiştir. İz buharlaşma oranı, her iterasyon sonrasında haritada biriken feromonların etkisinin azaltılma miktadıdır. Şekil 6.8 ve Şekil 6.9, C_{max} ve çalışma süresinin iz buharlaşma oranını nasıl değiştirdiği gösterilmektedir. İz buharlaşma oranı 1'e çok yaklaşırsa karıncalar öğrendikleri iyi yolları unutmaya başlayacaklarından daha kötü C_{max}'a sahip sonuçlar bulmaya başlayacaklardır.



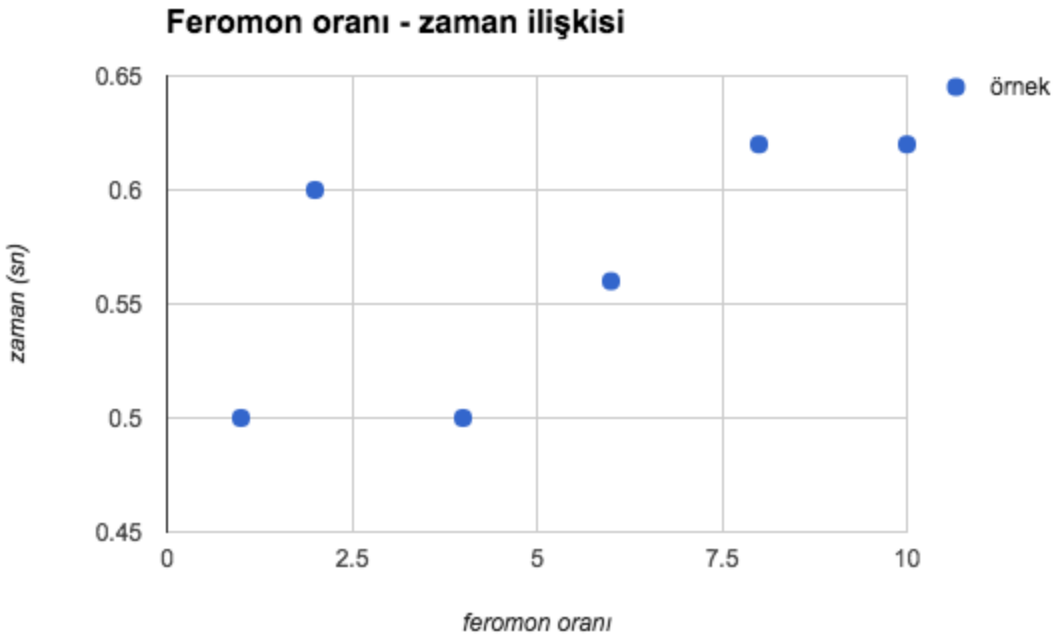
Şekil 6.8 İz buharlaşma oranının çalışma süresi ile ilişkisi



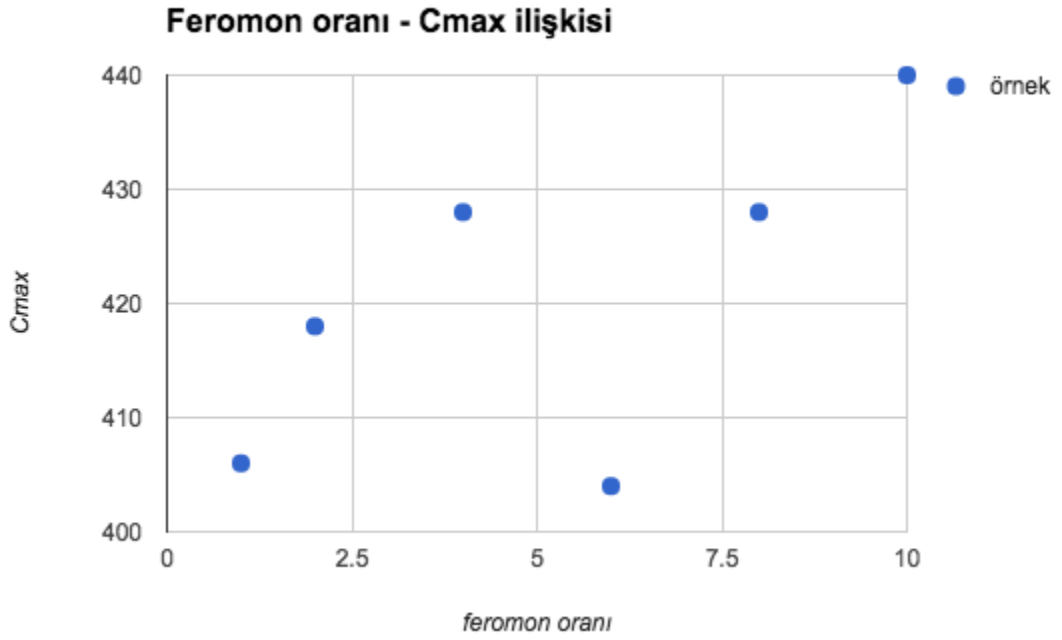
Şekil 6.9 İz buharlaşma oranının C_{max} ile ilişkisi

6.1.5 Feromon Oranını Değiştirmenin Etkileri

Bu bölümde, feromon oranını değiştirmenin C_{max} ve çalışma süresi ile ilişkisi araştırılmıştır. Şekil 6.10 ve Şekil 6.11'de feromon oranı ile C_{max} ve çalışma süresinin ilişkileri gösteriliyor. Nisbeten düşük feromon oranları daha iyi C_{max} değerine sahip çözümlere ulaşmayı kolaylaştırıyor.



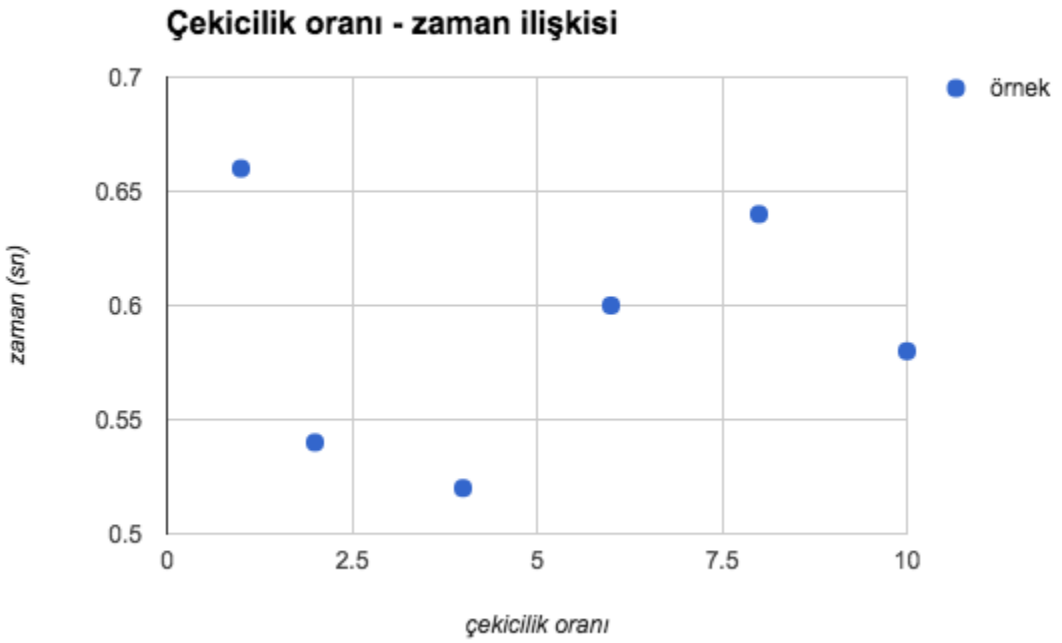
Şekil 6.10 Feromon oranının çalışma süresi ile ilişkisi



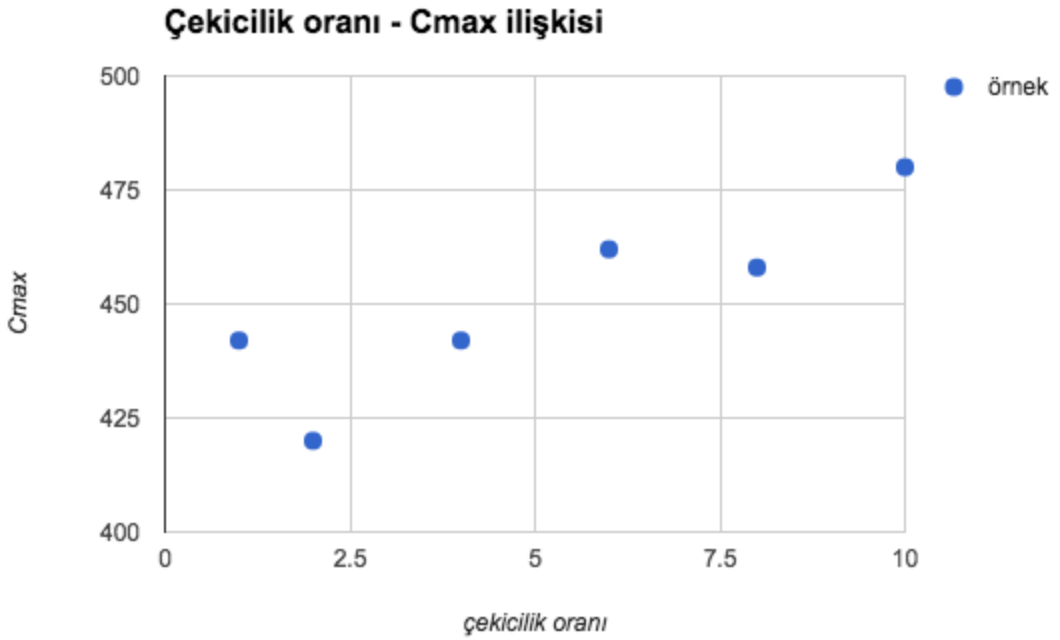
Şekil 6.11 Feromon oranının C_{max} ile ilişkisi

6.1.6 Çekicilik Oranını Değiştirmenin Etkileri

Bu bölümde çekicilik oranını değiştirmenin C_{max} ve çalışma süresi ile olan ilişkisi araştırılmıştır. Çekicilik oranı (desirability rate) karıncanın karşısına çıkan alternatif düğümlerden düşük işlem süresine sahip olanlara öncelik tanıma eğiliminin gücünü belirler. Şekil 6.12 ve Şekil 6.13'te de görülebileceği üzere yapılan testlerde çekicilik oranının küçük değerlerinde daha küçük C_{max}'a sahip sonuçlar bulunduğu görülmüştür. Aslında karıncanın bir sonraki prosese karar vermesi sürecinde feromon oranı ile çekicilik oranı birlikte kullanıldığından, feromon veya çekicilik oranının küçük değerlerinde iyi çözümler bulunması demek aslında birbirlerine oranları yakınken iyi çözümle bulunuyor olması şeklinde yorumlanabilir.



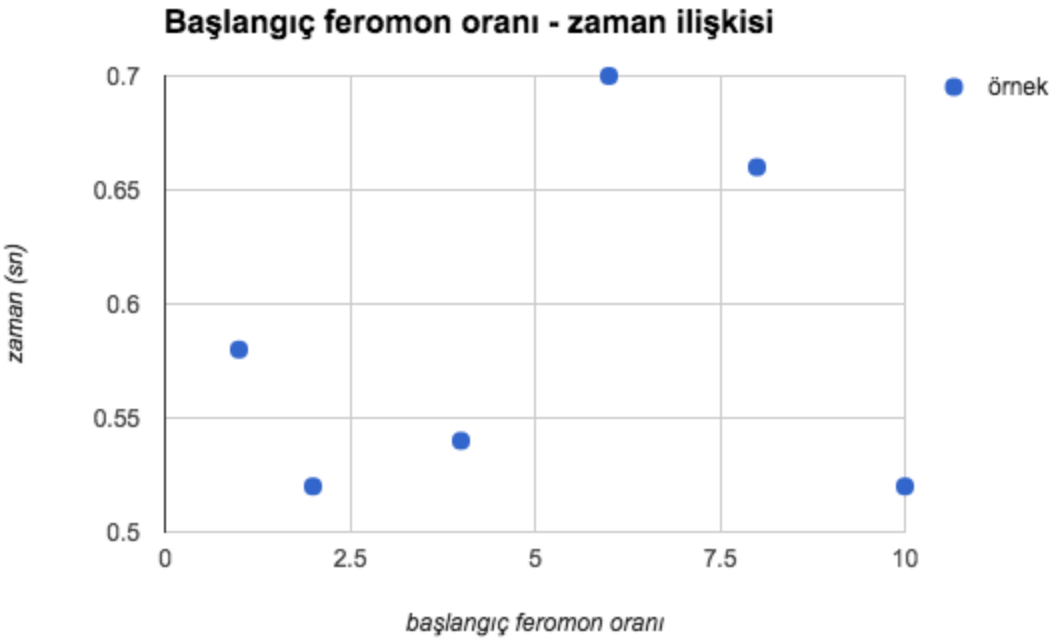
Şekil 6.12 Çekicilik oranının tamamlanma zamanı ile ilişkisi



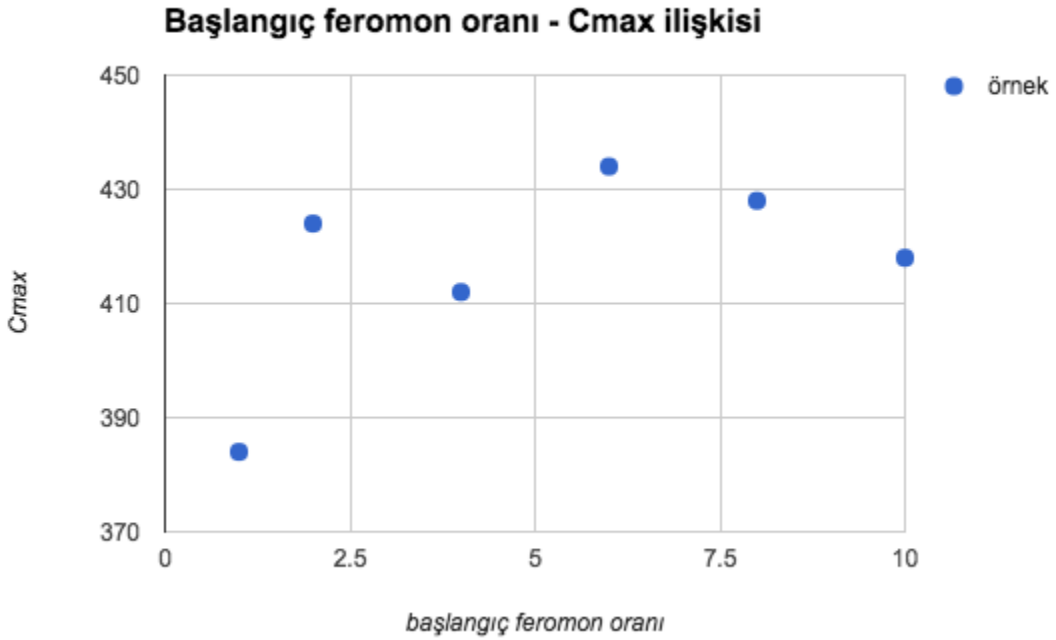
Şekil 6.13 Çekicilik oranının C_{max} ve tamamlanma zamanı ile ilişkisi

6.1.7 Başlangıç Feromon Oranını Değiştirmenin Etkileri

Bu bölümde başlangıç feromon oranının değiştirilmesinin C_{max} ve programın tamamlanma zamanı üzerindeki etkileri araştırılmıştır. Şekil 6.14 ve Şekil 6.15'te görülebileceği üzere bu parametrenin çalışma süresi üzerinde belirgin bir etkisi gözlemlenememiş ancak düşük başlangıç feromon değerlerinin daha iyi C_{max}'a sahip çözümler ortaya çıkarabileceği görülmüştür. Bu durum da haritada başlangıçta çok fazla feromon bulunmasının, karıncaların iyi yollara bıraktığı fazladan feromonun etkisini azaltabileceği şeklinde yorumlanabilir.



Şekil 6.14 Başlangıç feromon oranının çalışma süresi üzerindeki etkileri



Şekil 6.15 Başlangıç feromon oranının C_{max} üzerindeki etkileri

6.2 Genetik Algoritma Sonuçları

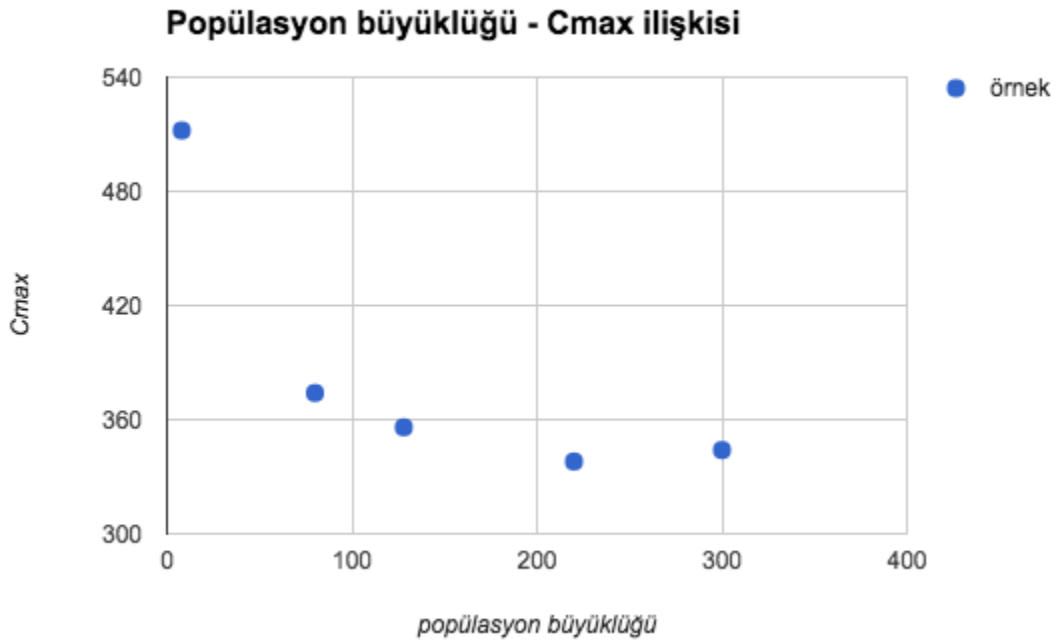
Deneysel çalışmaların bir diğer aşamasında Genetik Algoritma parametrelerinin etkileri araştırmıştır. Karınca kolonisi optimizasyonunda yapıldığı gibi bütün parametreler tek tek test edilmiş ve yine her 5 gözlem sonucu bir örnek olarak alınmıştır. Bir parametre test edilirken diğer parametreler şu şekilde alınmıştır:

Populasyon büyüklüğü:	128
Jenerasyon sayısı:	50
Mutasyon ihtimali:	0.1
Çaprazlama ihtimali:	0.9
Mutasyon tipi:	Ters mutasyon
Çaprazlama tipi:	Tek noktalı çaprazlama
Sıralı çaprazlama tipi:	Pozisyon tabanlı çaprazlama
Seçilim tipi:	Turnuva seçilimi

6.2.1 Popülasyon Büyüklüğünü Değiştirmenin Etkileri

Bu kısımda, popülasyon büyüklüğünün çalışma süresi ve bulunan en iyi çözümün C_{max} değeri üzerindeki etkileri incelenmiştir. Şekil 6.16 ve Şekil 6.17'de görülebileceği gibi popülasyon büyüklüğü arttıkça C_{max} değeri azalmış, ancak çalışma süresi uzamıştır.

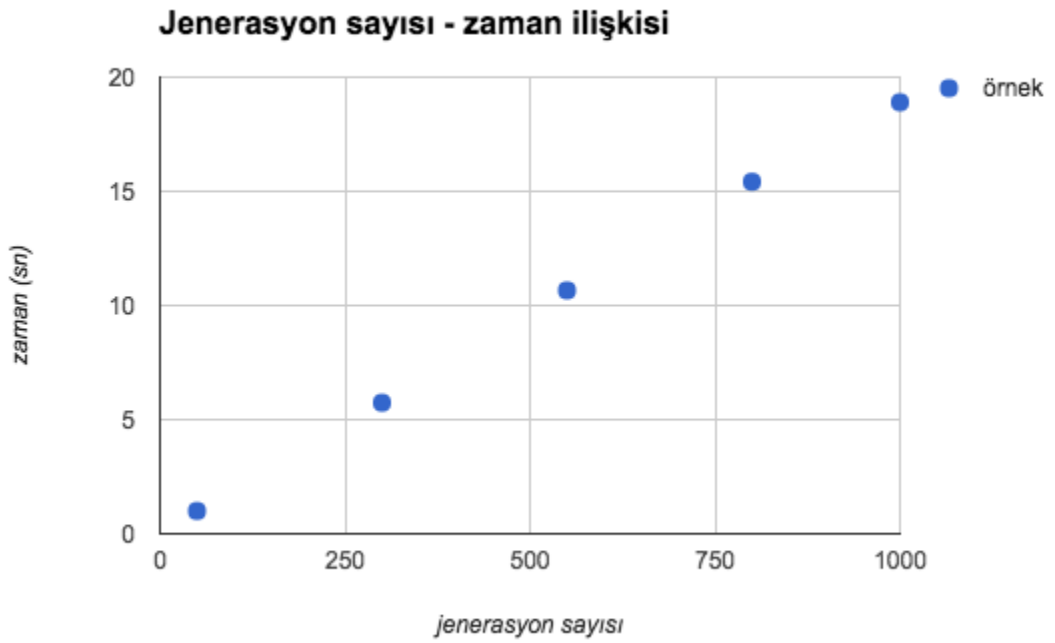
Şekil 6.16 Popülasyon büyüklüğünün çalışma süresi ile ilişkisi



Şekil 6.17 Popülasyon büyüklüğünün C_{max} ile ilişkisi

6.2.2 Jenerasyon Sayısının Değiştirmenin Etkileri

Bu kısımda jenerasyon sayısının çalışma süresi ve C_{max} değerini nasıl etkilediği araştırılmıştır. Şekil 6.18 ve Şekil 6.19'da görülebileceği üzere jenerasyon sayısı arttıkça programın çalışma süresi uzamaktadır. Ancak jenerasyon sayısı ile C_{max} arasında doğrudan bir ilişki bulunamamıştır. Bunun sebebi algoritmanın yerel optimum noktalarda tıkanması olarak gösterilebilir. Belli çözüm bölgelerinde sıkışan algoritma, nesil sayısının daha fazla artması ile bir iyileştirme sağlayamamaktadır. Bunun çözümü için belli miktarda nesil sonrasında iyileştirme sağlanamazsa popülasyon genetiği sıfırlanabilir.

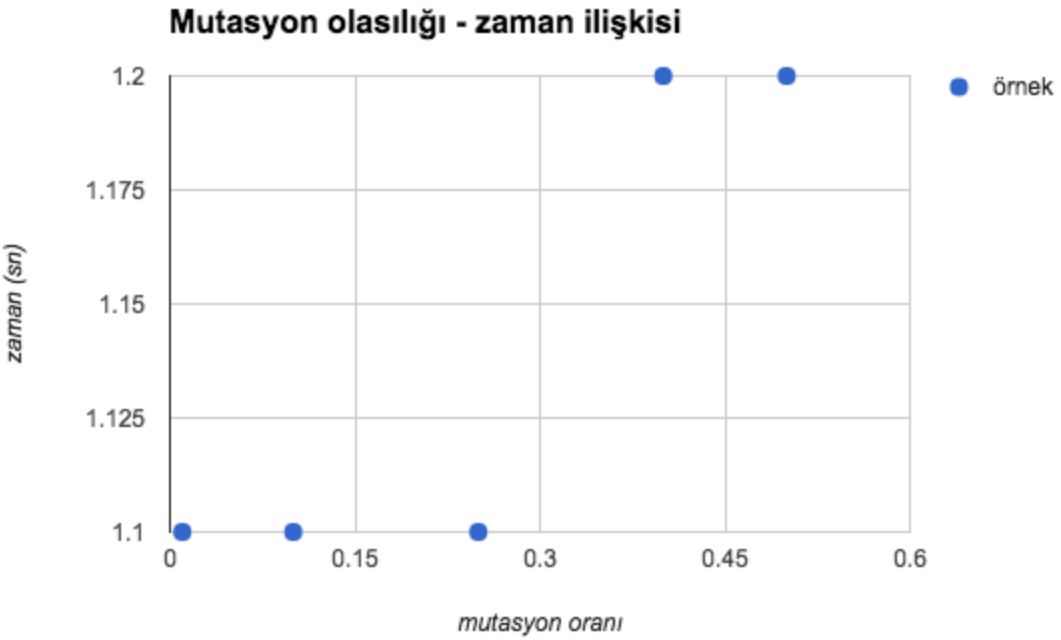


Şekil 6.18 Jenerasyon sayısı ile çalışma süresi ilişkisi

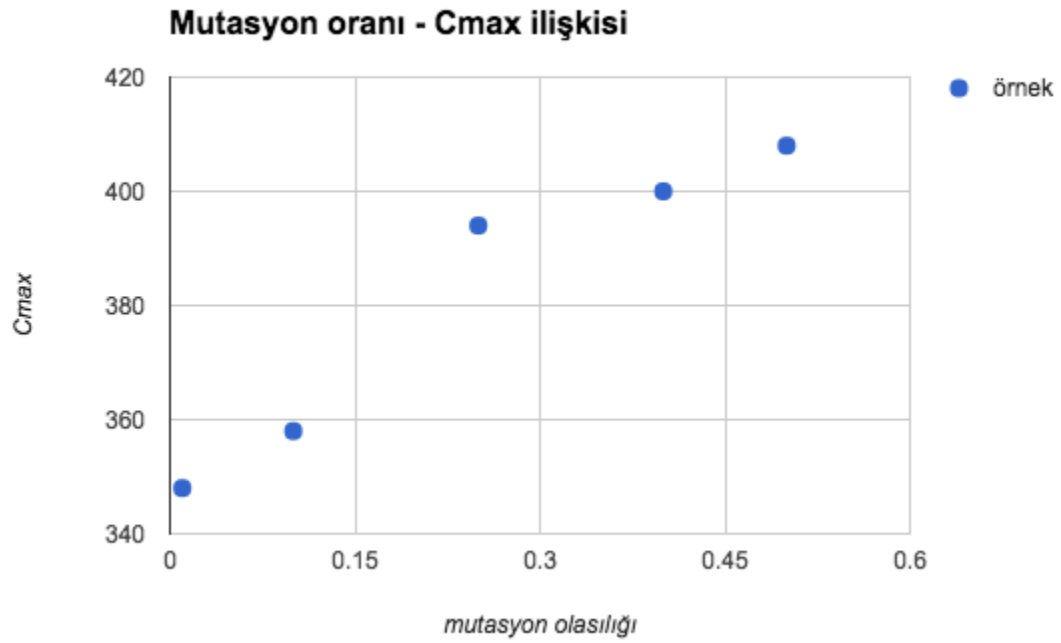
Şekil 6.19 Jenerasyon sayısı ile C_{max} ilişkisi

6.2.3 Mutasyon Olasılığını Değiştirmenin Etkileri

Bu kısımda, mutasyon olasılığının C_{max} ve çalışma süresi üzerindeki etkileri incelenmiştir. Şekil 6.20 ve Şekil 6.21'de görüleceği gibi mutasyon olasılığının yükseltilmesi daha çok zaman aldığı gibi C_{max} sonuçları üzerinde de olumsuz bir etki yapar. Bu beklenen bir durumdur, çünkü çok fazla mutasyon olması popülasyonun geliştirdiği iyi çözümlerin bozulmasına yol açabilir.



Şekil 6.20 Mutasyon olasılığının çalışma süresi üzerindeki etkileri



Şekil 6.21 Mutasyon olasılığının C_{max} üzerindeki etkileri

6.2.4 Çaprazlama Olasılığını Değiştirmenin Etkileri

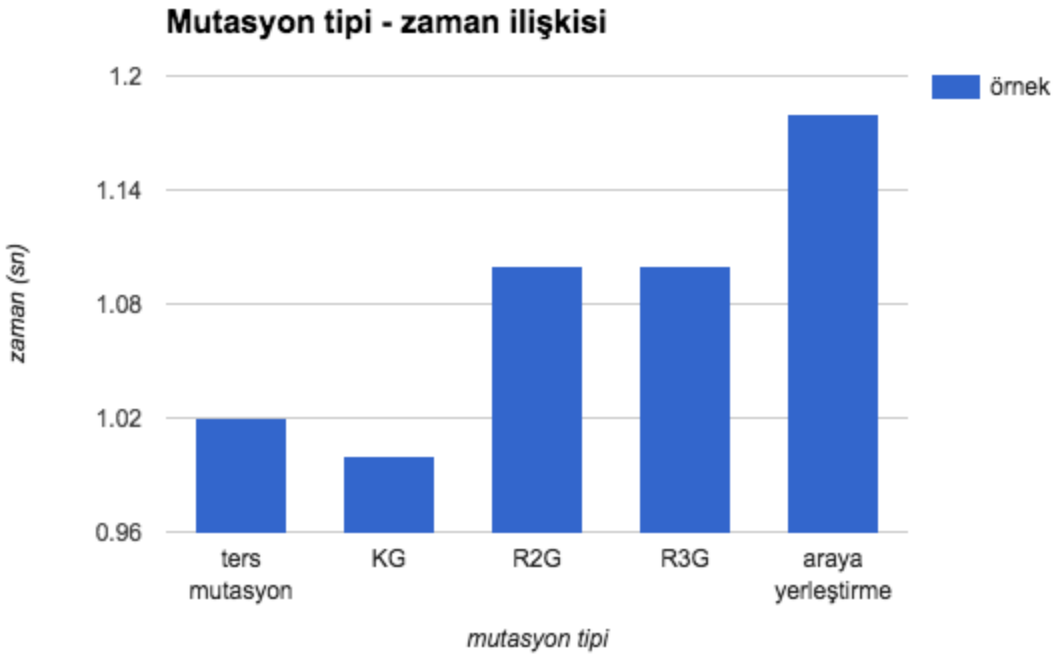
Bu kısımda, çaprazlama olasılığını değiştirmenin C_{max} ve çalışma süresi üzerindeki etkileri araştırılmıştır. Şekil 6.22 ve Şekil 6.23'te görüldüğü gibi çaprazlama olasılığını artırmak çalışma süresini uzatır, ancak C_{max} üzerinde belirgin bir etkiye sahip değildir.

Şekil 6.22 Çaprazlama olasılığının çalışma süresi üzerindeki etkileri

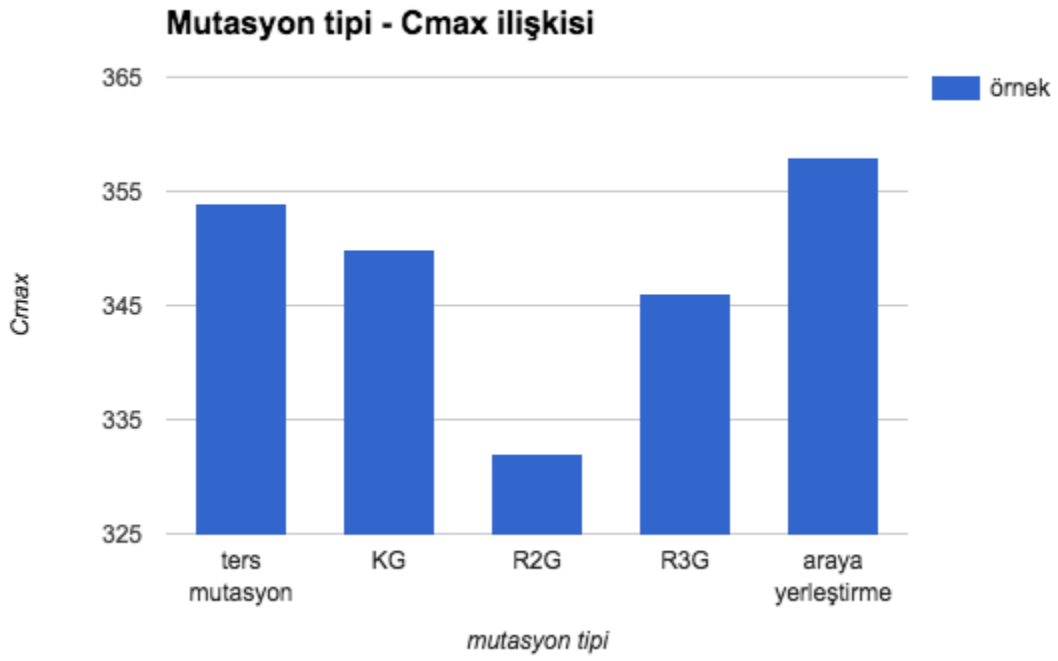
Şekil 6.23 Çaprazlama olasılığının C_{max} üzerindeki etkileri

6.2.5 Mutasyon Yöntemini Değiştirmenin Etkileri

Bu kıyaslamada mutasyon tipini değiştirmenin C_{max} ve çalışma süresi üzerindeki etkileri araştırılmıştır. Ele aldığımız mutasyon tipleri 'Ters mutasyon', 'Komşu iki genin değiştirilmesi' (KG), 'Rastgele iki genin değiştirilmesi' (R2G), 'Rastgele üç genin değiştirilmesi' (R3G) ve 'Araya yerleştirme'dir [53]. Şekil 6.24 ve Şekil 6.25'te görülebileceği gibi en iyi ortalama C_{max} değerleri 'Keyfi 2 genin değiştirilmesi' şeklinde uygulanan mutasyon tipi ile elde edilmiştir. En hızlı çalışma süresi 'Komşu iki genin değiştirilmesi' mutasyonunda, en kötü çalışma süresi ise 'Araya yerleştirme' isimli mutasyonda elde edilmiştir. Ancak mutasyonlar arası çalışma süresi farkı kayda değer ölçüde olmamıştır.



Şekil 6.24 Mutasyon tipi ile çalışma süresi ilişkisi



Şekil 6.25 Mutasyon tipi ile C_{max} ilişkisi

6.2.6 Çaprazlama Yöntemini Değiştirmenin Etkileri

Bu kıyaslamada çaprazlama tipini değiştirmenin C_{max} ve çalışma süresi üzerindeki etkileri araştırılmıştır. Şekil 6.26 ve Şekil 6.27'de görülebileceği üzere çalışma süresi açısından 'Tek noktalı çaprazlama' en iyi, 'Çok noktalı çaprazlama' ise en kötü sonucu vermiştir. Yine aynı şekilde görülebileceği gibi 'Tek noktalı çaprazlama' en kötü C_{max} değerlerini vermiş iken 'Çift noktalı çaprazlama' ile en iyi C_{max} değerleri elde edilmiştir.

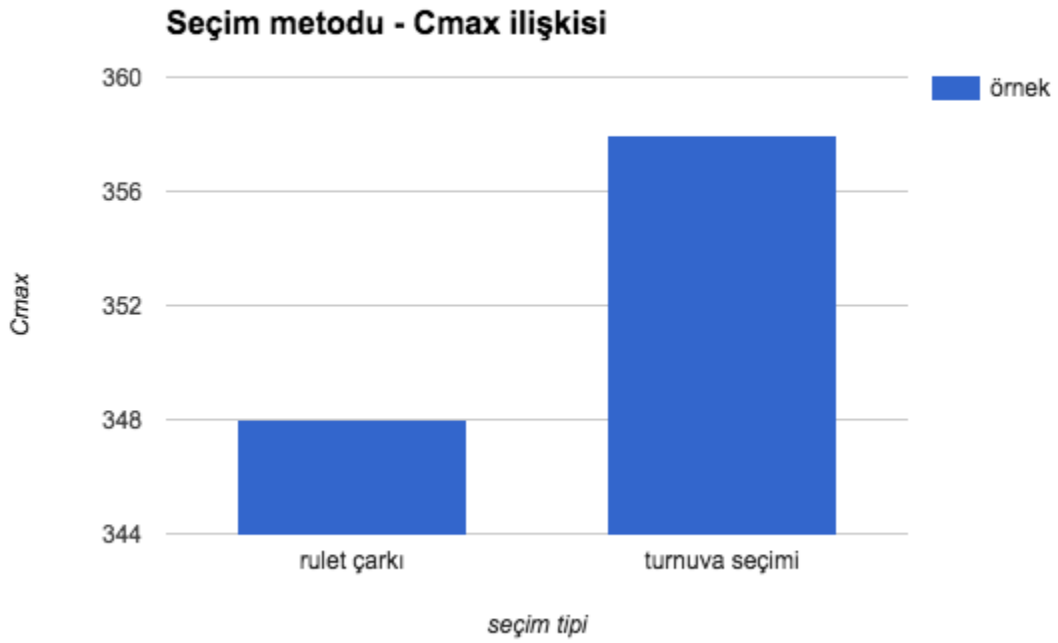
Şekil 6.26 Çaprazlamanın çalışma süresi ile ilişkisi

Şekil 6.27 Çaprazlamanın C_{max} ile ilişkisi

6.2.7 Seçim Yöntemini Değiştirmenin Etkileri

Bu kıyaslamada seçim yönteminin C_{max} ve çalışma zamanı ile ilişkisi araştırılmıştır. Şekil 6.28 ve Şekil 6.29'da görüleceği gibi 'Rulet seçimi' kullanıldığında çözümlerin ortalama C_{max} değeri bir miktar daha iyi gerçekleşmiştir.

Şekil 6.28 Seçim yöntemi ile tamamlanma zamanı ilişkisi



Şekil 6.29 Seçim yöntemi ile C_{max} ilişkisi

6.3 Genetik Algoritma – Karınca Kolonisi Optimizasyonu Kıyaslaması

Bu bölümde Karınca Kolonisi Optimizasyonu (KKO) ile Genetik Algoritma'nın (GA) farklı problem tiplerinde kıyaslanması gerçekleştirilmiştir. Hepsi atölye tipi olmak üzere altı farklı problem kullanılmıştır:

- Örnek problem (5 iş - 4 proses - 3 makine) *
- Random bir 5 iş - 5 proses - 5 makine problemi
- Random bir 10 iş - 5 proses - 5 makine problemi
- Random bir 5 iş - 10 proses - 5 makine problemi
- Random bir 5 iş - 5 proses - 10 makine problemi
- Random bir 10 iş - 10 proses - 10 makine problemi

* Örnek problem D. Y. Lee ve F. DiCesare'nin [54] makalelerinden alıntılanmış bir 5 iş - 4 proses - 3 makina problemidir.

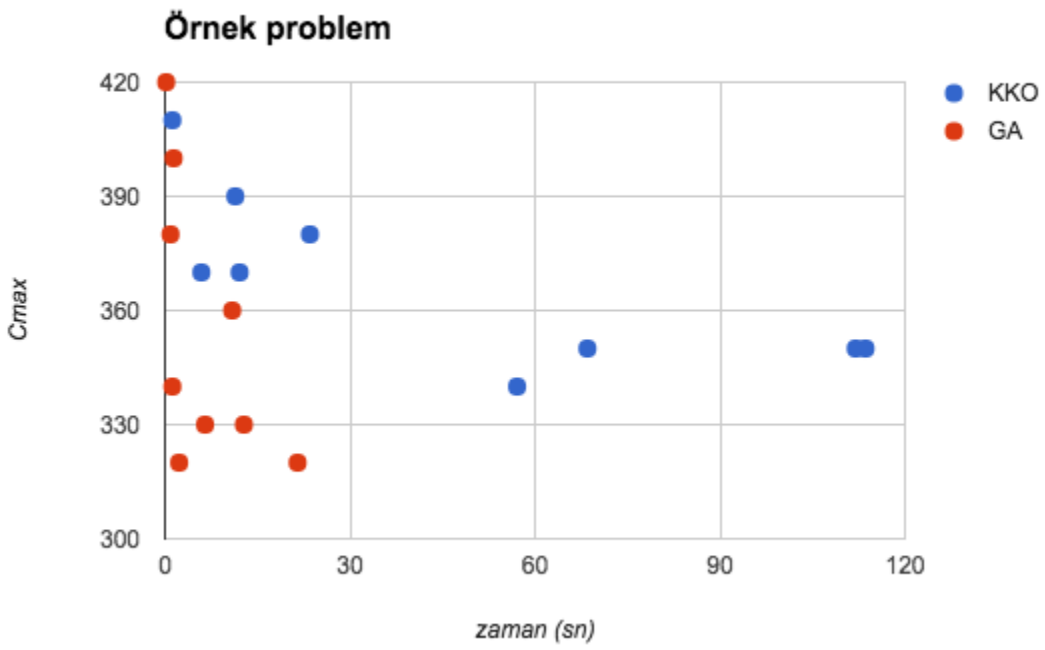
Her problemde 2 algoritma da birkaç farklı konfigürasyonda çalıştırılmıştır. Karınca Kolonisi Optimizasyonu'nda Karınca ve İterasyon sayısı değiştirilmiştir, Genetik Algoritma'da ise Popülasyon büyüklüğü ve Jenerasyon sayısı değiştirilmiştir. Diğer parametreler Çizelge 6.3'de görülebilir.

Çizelge 6.3 Karınca Kolonisi ve Genetik Algoritma parametreleri

KKO		GA	
En Fazla Yineleme Sayısı	10	Mutasyon Olasılığı	0.1
İz Buharlaşma Oranı	0.75	Çaprazlama Olasılığı	0.9
Feromon Oranı	2	Seçim Yöntemi	Rulet çarkı
Çekicilik Oranı	2	Mutasyon Tipi	Keyfi 2 geni değiştirme
Başlangıç Feromon Oranı	2	Çaprazlama Tipi	Çift noktalı çaprazlama

6.3.1 Örnek Problemde Kıyaslama

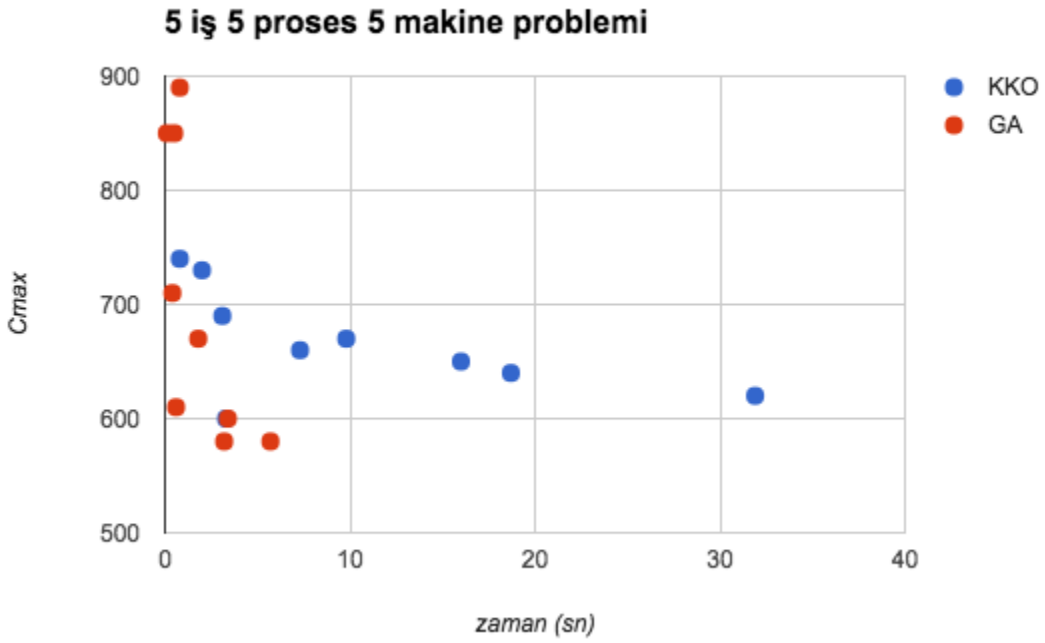
Bu kısımda 5 iş - 4 proses - 3 makinalı örnek problem kullanılarak iki algoritma karşılaştırılmıştır. Şekil 6.30'da bu kıyaslama sonuçları görülebilir. Grafikteki her bir nokta farklı bir konfigürasyonu göstermektedir. Kırmızı noktalar Genetik Algoritma'yı, mavi noktalar ise Karınca Kolonisi Algoritması'nı simgelemektedir. Grafiğin alt düzlemi algoritmanın çalışma süresini, sol düzlemi ise algoritmanın durduğunda bulduğu sonucun C_{max} değerini göstermektedir. Eşit zaman dilimlerinde daha iyi sonuçlar bulmasından hareketle Genetik Algoritma bu problem ölçeğinde daha başarılı olmuştur denebilir.



Şekil 6.30 KKO ve GA'nın Örnek problemde kıyaslanması

6.3.2 5 İş - 5 Proses - 5 Makineli Bir Problemde Kıyaslama

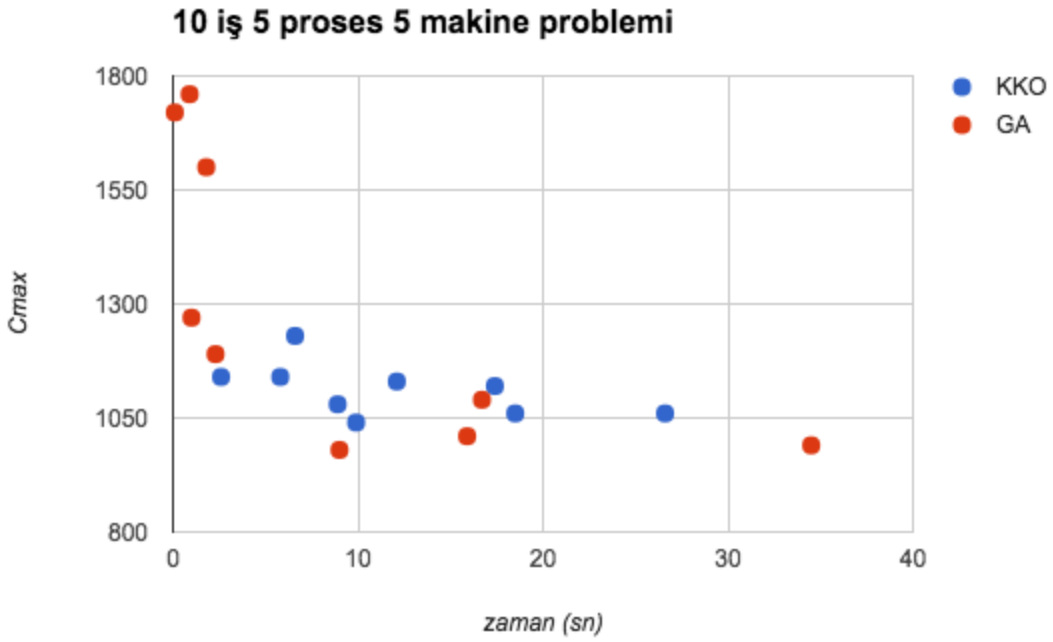
Bu kısımda iki algoritma 5 iş - 5 proses - 5 makineli bir problem üzerinden karşılaştırılmıştır. Şekil 6.31'de görülebilecek sonuçlara göre Genetik Algoritma daha hızlı ve daha iyi sonuçlar vermiştir.



Şekil 6.31 KKO ve GA'nın 5 iş 5 proses 5 makine probleminde kıyaslanması

6.3.3 10 İş - 5 Proses - 5 Makinalı Bir Problemde Kıyaslama

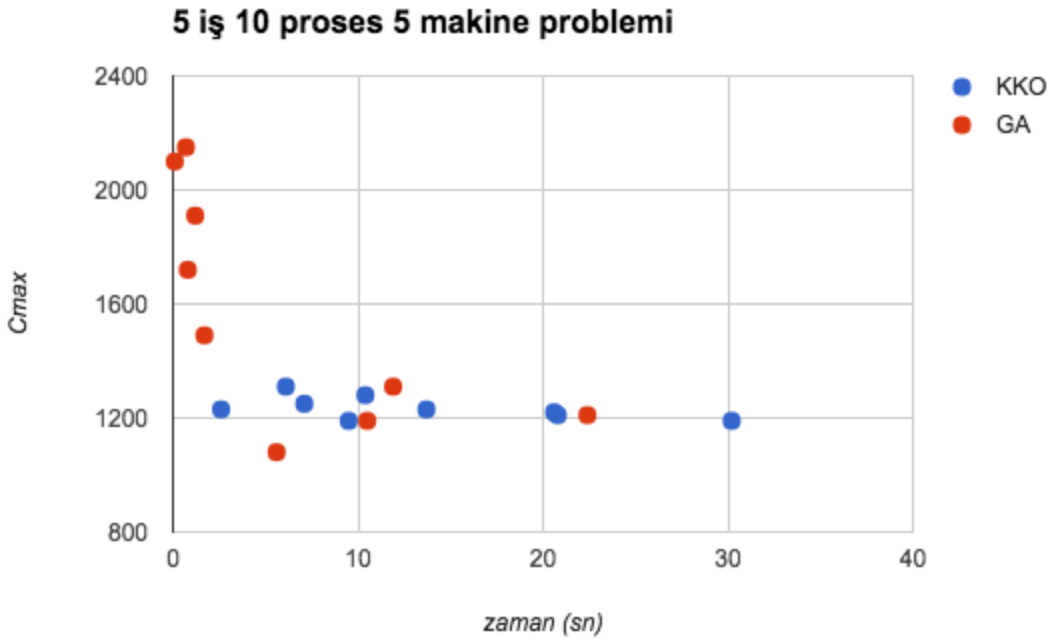
Bu kısımda iki algoritma random bir 10 iş - 5 proses - 5 makinalı problem ile kıyaslanmıştır. Şekil 6.32'de görülebilecek sonuçlara göre bu problem tipinde iki algoritma da aşağı yukarı benzer sonuçlar bulmulşlardır ancak Genetik Algoritma bir miktar daha iyi sonuçlar elde etmiştir.



Şekil 6.32 KKO ve GA'nın 10 iş 5 proses 5 makina probleminde kıyaslanması

6.3.4 5 İş - 10 Proses - 5 Makinalı Bir Problemde Kıyaslama

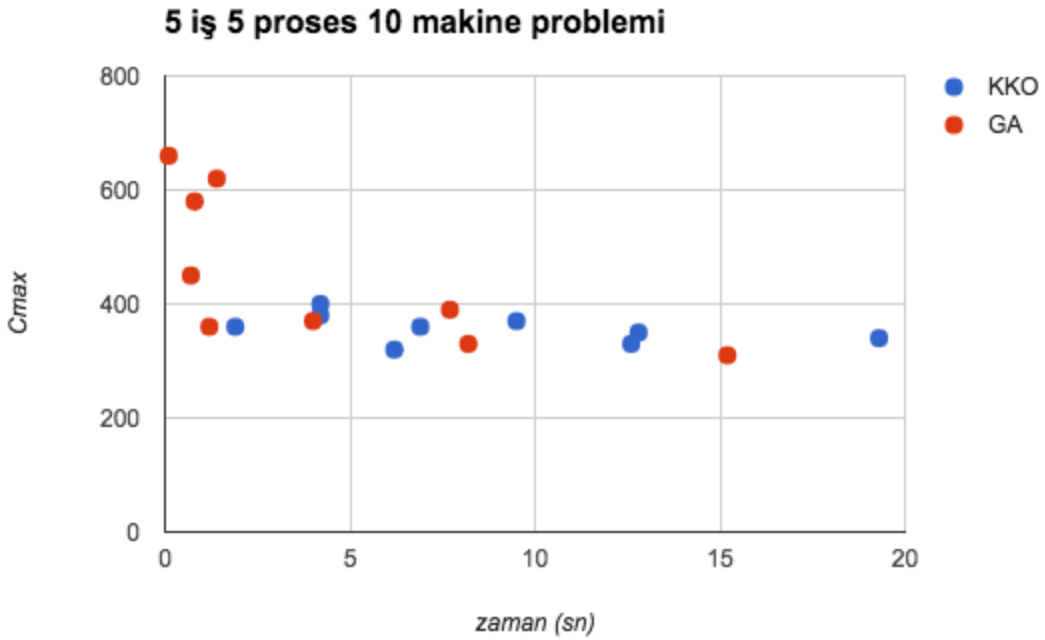
Bu kısımda iki algoritma 5 iş - 10 proses - 5 makinaya sahip rastgele bir problem üzerinden kıyaslanmışlardır. Şekil 6.33'de görüleceği üzere iki algoritma da benzer sonuçlar elde etmiştir. Bütün çözümler arasında en iyisi Genetik algoritma'ya ait olduğundan doğru konfigürasyonda GA'nın bu problem ölçeğinde daha iyi sonuçlar elde edebileceği düşünülmüştür.



Şekil 6.33 KKO ve GA'nın 5 iş 10 proses 5 makina probleminde kıyaslanması

6.3.5 5 İş - 5 Proses - 10 Makineli Bir Problemde Kıyaslama

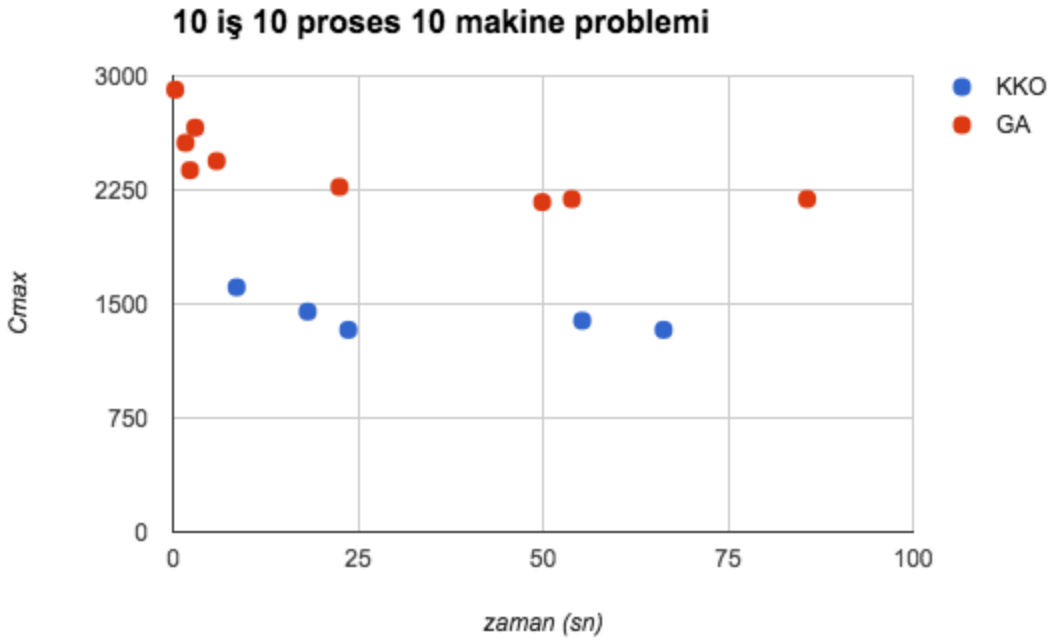
Bu kısımda iki algoritma 5 iş - 5 proses - 10 makinalı bir problemde kıyaslanmıştır. Şekil 6.34'te de görülebileceği üzere iki algoritma da benzer sonuçlar elde etmiştir. Yine en iyi çözüm Genetik Algoritma'ya ait olduğundan doğru konfigürasyonda Genetik Algoritma'nın daha iyi sonuçlar verebileceği düşünülmüştür.



Şekil 6.34 KKO ve GA'nın 5 iş 5 proses 10 makina probleminde kıyaslanması

6.3.6 10 İş - 10 Proses - 10 Makinalı Bir Problemde Kıyaslama

Bu bölümde iki algoritma 10 iş - 10 proses - 10 makinalı bir problemde karşılaştırılmıştır. Şekil 6.35'te görülebileceği üzere bu problemde Karınca Kolonisi belirgin şekilde daha iyi sonuçlar vermiştir.



Şekil 6.35 KKO ve GA'nın 10 iş 10 proses 10 makina probleminde kıyaslanması

6.3.7 GA - KKO Değerlendirmesi

İki algoritmanın da C_{max} ve çalışma süresi üzerinden parametre performansları ölçüldü. Karınca sayısı ile popülasyon büyüklüğünün, iterasyon ve jenerasyon sayısı ile benzer etki yaptığı gözlendi. Bu 4 parametrenin de programın çalışma süresini uzatıp sonuçları ise bir yere kada iyileştirdiği görüldü. Diğer parametreler de deneylerle belirlendi. Karınca Kolonisi Optimizasyonu'nda en fazla yineleme sayısı 10, iz buharlaşma oranı 0.25-0.75 arasında, feromon oranı, çekicilik oranı ve başlangıç feromon oranı 2 civarında iken iyi çözümler bulundu. Genetik Algoritma'da ise mutasyon oranı düşük, çaprazlama oranı yüksek, seçim yöntemi 'Rulet çarkı', mutasyon tipi 'Rasgele 2 geni değiştirme' iken ve çaprazlama tipi 'Çift noktalı çaprazlama' iken iyi sonuçlar bulundu.

Daha sonra iki algoritma birbirleri ile kıyaslandı. Bu noktada büyüdükçe performansı artıran parametreleri farklı sayılarda denenip sonuçlar elde edildi. Daha iyi C_{max} değerleri elde etmek veya aynı C_{max} değerlerini daha kısa sürelerde bulmak hedeflendi.

İş, proses ve makine sayıları değiştirilerek farklı Atölye tipi çizelgeleme problemleri oluşturuldu, sonuçlar bu problemler üzerinden elde edildi.

Elde edilen sonuçları genel olarak yorumlamak gerekirse, Çizelge 6.4'te de görülebileceği gibi iki adet yorum yapılabilir. Programın çalışma süresi arttıkça GA'nın performansı iyileşiyor. KKO programın çalışma süresi uzatıldığında aynı şekilde sonuçlarını iyileştiremiyor. Ayrıca problemin ölçeği büyüdükçe KKO, GA'ya göre daha iyi sonuçlar bulmaya başlıyor. GA küçük ölçekteki problemlerde daha başarılı ancak büyük ölçekteki problemlerde KKO ile rekabet edememeye başlıyor.

Çizelge 6.4 GA/KKO Algoritma tercih tablosu

Zaman & Ölçek	Küçük problem	Büyük problem
Uzun zaman	GA	GA/KKO
Kısa zaman	GA/KKO	KKO

6.4 GA – YBS Değerlendirmesi

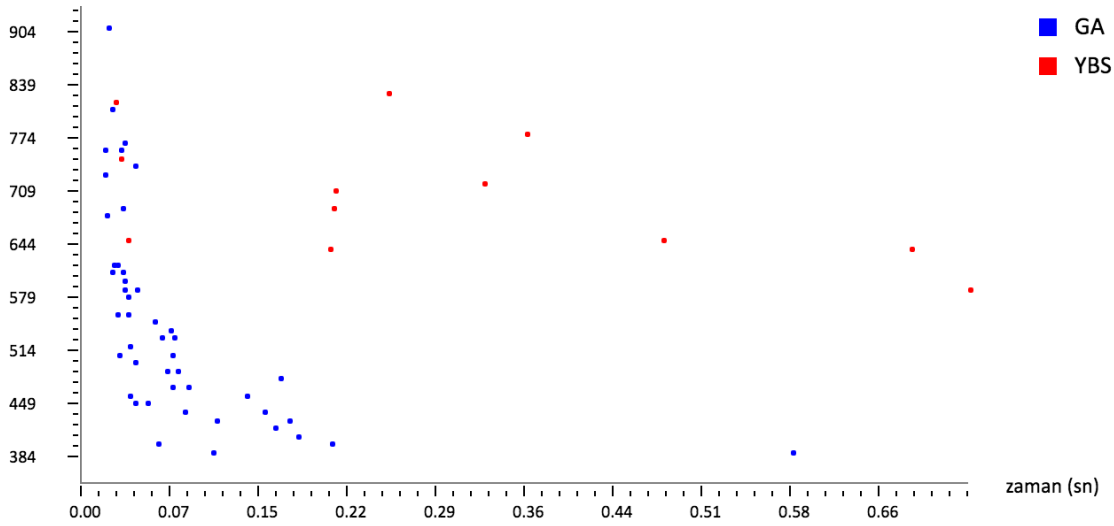
Verilen yönergelere göre YBS algoritması Javascipt dilinde yazılmış ve sistemle bütünleştirilmiştir. Daha sonra önceden yazılmış olan Genetik Algoritma (GA) ile belli problem ölçeklerinde kıyaslanmıştır. Her problem tipi için iki algoritma 5'er kez çalıştırılmış ve sonuçlar kaydedilmiştir. Algoritmaları kıyaslarken hem GA hem de YBS sezgisel algoritmalar olduğundan, algoritmaların çalışma süresini ve buldukları çözümün C_{max} değerini aynı anda değerlendirebilmek için bir C_{max} - zaman grafiği önerilmiştir. Algoritmalar çalışmaları esnasında herhangi bir anda buldukları daha iyi sonucu grafiğe kaydetmektedirler. Böylece algoritmaları kıyaslarken sadece en son buldukları sonucu değil, o sonuca nasıl bir hızda yaklaştıklarını görme şansımız olur ve bu durum bize daha iyi bir kıyas imkanı verir.

Bunun yanında yapılan denemelerin yüzde kaçında hangi algoritmanın daha iyi sonuç bulduğu (SR - Success Rate), algoritmanın ortalama olarak bulunan en iyi sonuçtan

sapma yüzdesi (ARPD - Average Relative Percentage Deviation) ve algoritmanın ortalama çalışma süresi (CPU) hesaplanmıştır.

6.4.1 Örnek Problem

Örnek problem makalelerde [54] algoritmaları başka makalelerde önerilen algoritmalarla kıyaslamak amacıyla kullanılan, testbed adıyla anılan problemlerden bir tanesi olup 5 iş - 4 proses - 3 makine problemidir.



Şekil 6.36 Örnek problem üzerinde GA ve YBS kıyaslaması

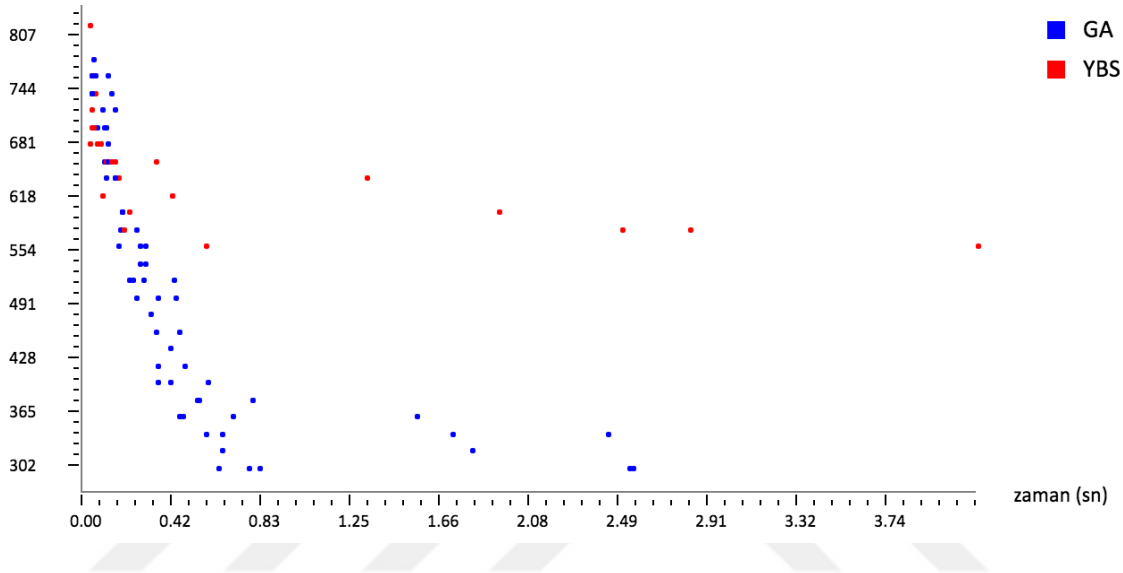
Yapılan kıyaslamalar sonucu Çizelge 6.5'te görülebilir. GA bütün denemelerde daha iyi sonuç bulmuş. YBS'nin sonuçları en iyi sonuçtan ortalama %60 civarında daha kötü olmuş. Çalıştıkları konfigürasyonlarda ise GA'nın ortalama çalışma süresi ise YBS'nin yaklaşık 2 katı olmuş. Şekil 6.36'da ise algoritmaların çalışmaları esnasında buldukları daha iyi sonuçların izleri görülüyor. Bu grafikte de GA'nın bariz üstünlüğü görülüyor.

Çizelge 6.5 Örnek problem GA - YBS kıyas tablosu

Algoritmalar	SR	ARPD	CPU
GA	100	0	1.10
YBS	0	59.30	0.46

6.4.2 5 iş 5 proses 5 makina problemi

Bu problem tipi örnek problemden biraz daha büyük ölçektedir. Probleme ilişkin veriler oluşturulurken Uniform dağılım kullanılmış ve yine iki algoritma da 5'er kez çalıştırılmıştır.



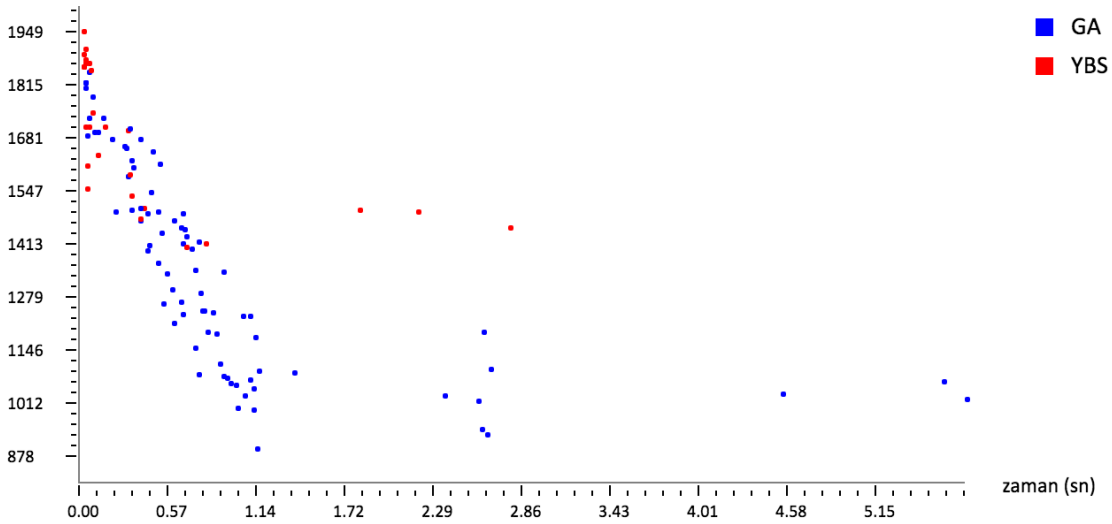
Şekil 6.37 5 iş 5 proses 5 makine problemi üzerinde GA ve YBS kıyaslaması

Yapılan denemeler sonucunda Çizelge 6.6'daki sonuç oluşmuştur. Bütün denemelerde GA'nın daha başarılı olduğu bu çizelgede ve ayrıca Şekil 6.37'de görülüyor. Ayrıca YBS yaklaşık %90 daha kötü sonuçlar bulurken, GA'nın yalnızca %25'i kadar zaman kullanmıştır.

Çizelge 6.6 5 iş 5 proses 5 makina problemi GA - YBS kıyas tablosu

Algoritmalar	SR	ARPD	CPU
GA	100	0	8.00
YBS	0	92	2.05

6.4.3 5 iş 10 proses 5 makina problemi



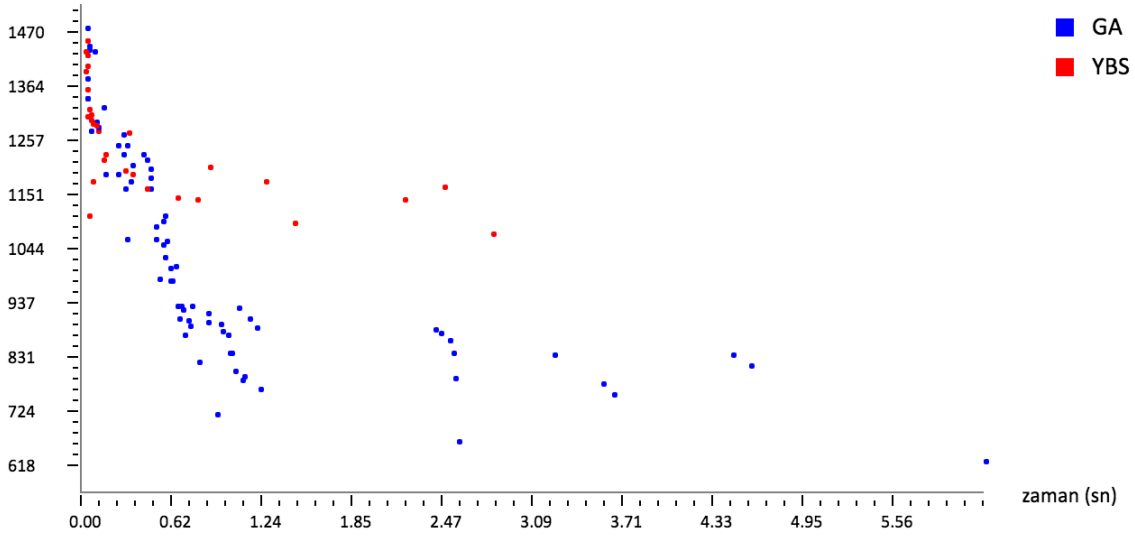
Şekil 6.38 5 iş 10 proses 5 makine problemi üzerinde GA ve YBS kıyaslaması

Daha sonra 5 iş 10 proses 5 makinalı bir problem üzerinde kıyaslama yapıldı. GA bütün denemelerde daha iyi sonuç bulurken süre kullanımı yaklaşık 6 kat daha fazla oldu. YBS'nin ortalama en iyi çözümden sapma miktarı %50 civarında gerçekleşti. Çizelge 6.7 ve Şekil 6.38'te bu karşılaştırmanın grafik ve nicel sonuçları görülüyor.

Çizelge 6.7 5 iş 10 proses 5 makina problemi GA - YBS kıyas tablosu

Algoritmalar	SR	ARPD	CPU
GA	100	0	10.80
YBS	0	48.93	1.64

6.4.4 5 iş 10 proses 10 makina problemi



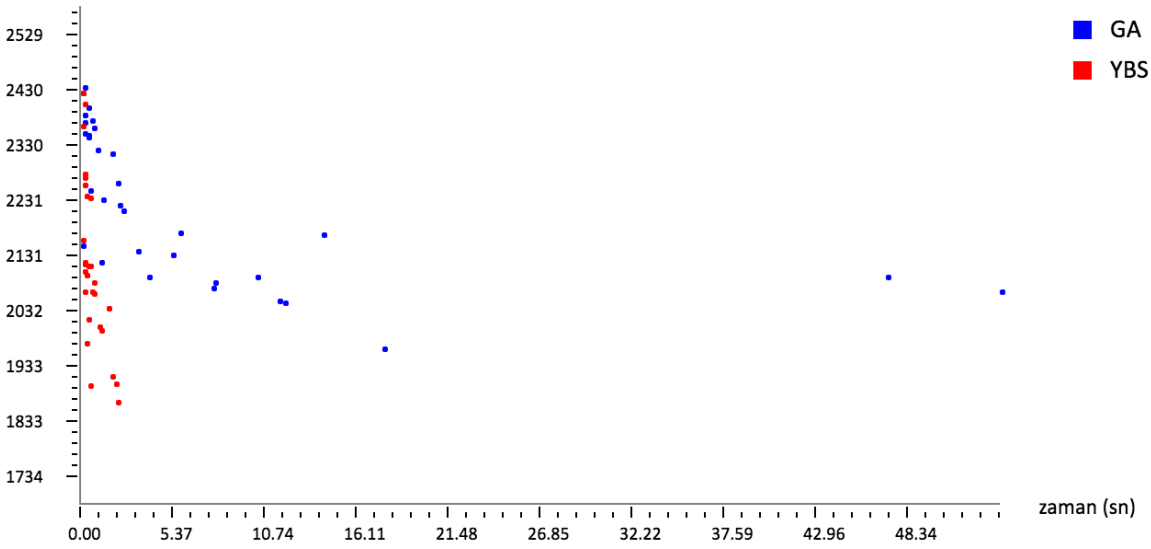
Şekil 6.39 5 iş 10 proses 5 makina problemi üzerinde GA ve YBS kıyaslaması

5 iş 10 proses 10 makinalı bir problemde daha kıyaslama yapıldı. Yapılan karşılaştırma sonucu Şekil 6.39'da görülüyor. GA, YBS'nin hiç inemediği C_{max} değerlerine inebilmiş. Aynı sonuç Çizelge 6.8'de de görülebilir. GA bütün denemelerde daha iyi sonuç bulurken, Yaklaşık 4 kat fazla zaman kullanmış. YBS'nin bulduğu sonuçlar ise ortalama olarak en iyi sonuçtan %54 daha kötü değer almış.

Çizelge 6.8 5 iş 10 proses 10 makina problemi GA - YBS kıyas tablosu

Algoritmalar	SR	ARPD	CPU
GA	100	0	9.12
YBS	0	53.52	1.94

6.4.5 10 iş 10 proses 10 makina problemi



Şekil 6.40 10 iş 10 proses 10 makina problemi üzerinde GA ve YBS kıyaslaması

Çizelge 6.9'da de görülebileceği gibi 10 iş 10 proses 10 makina ölçeğindeki problemlerde YBS algoritması GA'ya kıyasla daha başarılı olmuştur. Hem harcadığı zaman önemli ölçüde düşük, hem de bulduğu sonuçların en iyi çözümden ortalama sapması GA'ya kıyasla daha azdır. Şekil 6.40'ta bu durumu görebiliriz, YBS algoritması çok kısa sürelerde daha düşük C_{max}'lı çözümler bulabilmişlerdir.

Çizelge 6.9 10 iş 10 proses 10 makina problemi GA - YBS kıyas tablosu

Algoritmalar	SR	ARPD	CPU
GA	20	6.76	36.46
YBS	80	0.74	1.23

İki algoritma çeşitli ölçekteli Atölye tipi paralel makinalı çizelgeleme problemleri üzerinde karşılaştırılmıştır. Algoritmalar benzer parametreler ile çalıştırılmıştır. Yapılan

deneysel karşılaştırmalar sonunda YBS'nin GA'ya kıyasla küçük ölçekteki problemlerde daha başarısız olduğu, ancak büyük ölçekteki problemler sözkonusu olduğunda daha iyi çözümler bulabildiği görülmüştür. Ayrıca YBS'nin benzer parametreler ile çalıştırıldığında daha hızlı sonuçlandığı görülmüştür. Çizelge 6.10'da görülebileceği üzere büyük problemlerdeki performansının görece daha iyi oluşu ve hızlı sonuçlanması YBS algoritmasının Çizelgeleme problemlerinde kullanılabileceğini düşündürmüştür.

Çizelge 6.10 GA/YBS Algoritma tercih tablosu

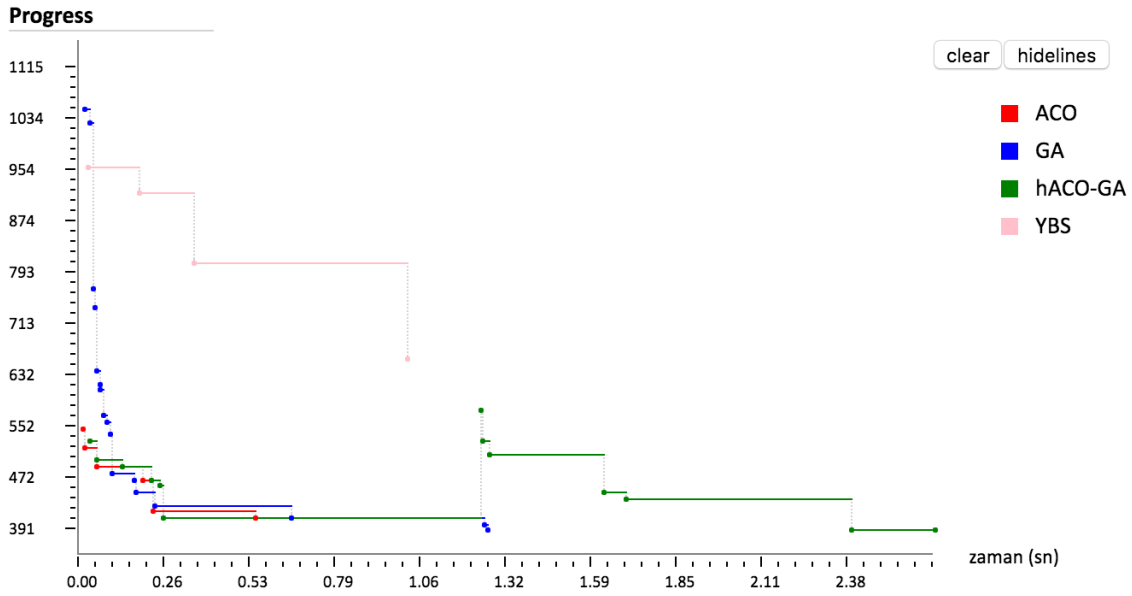
Zaman & Ölçek	Küçük problem	Büyük problem
Uzun zaman	GA	GA/YBS
Kısa zaman	GA/YBS	YBS

6.5 Bütün Algoritmaların Kıyaslaması

Çalışmanın bu son kısmında, hem yazılan çizelgeleme paket programının çizelgeleme problemleri üzerinde test yapmayı ne kadar kolaylaştırdığını gösterebilmek hem de geliştirilen hibrit yaklaşımın işlerliğini göstermek amacıyla atölye tipi çizelgeleme problemleri için gösterilen Yapay bağışıklık sistemi, Karınca kolonisi optimizasyonu, Genetik algoritma ve melez yaklaşım; biri büyük, biri küçük ölçekli olmak üzere 2 problem üzerinde kıyaslanmıştır.

6.5.1 Küçük ölçekli problem

Bu kısımdaki problem 5 iş, 4 proses, 3 makina ölçeğinde olup bahsi geçen 4 algoritma birer kez çalıştırılmıştır. Ortaya çıkan sonuçlar Şekil 6.41 ve Çizelge 6.11'de görülebilir. Genetik algoritma ve Melez yaklaşım küçük problemde iyi sonuç vermişlerdir. Karınca kolonisi optimizasyonu da kabul edilebilir bir çözüm vermiş ancak Yapay bağışıklık sisteminin sonucu diğerlerine kıyasla kötü olmuştur.



Şekil 6.41 Dört algoritmanın küçük ölçekli problemde kıyaslanması

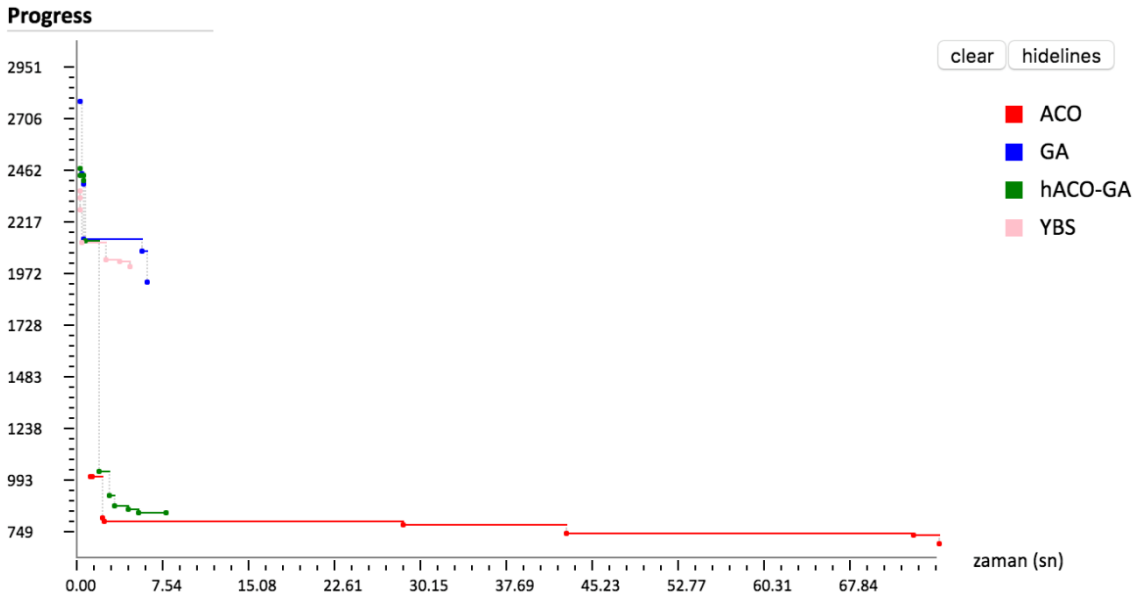
Çizelge 6.11 Dört algoritmanın küçük ölçekli problemdeki sonuçları

Algoritmalar	SR	ARPD	CPU
KKO	0	5.13	1.33
GA	100	0	2.06
Melez	100	0	2.66
YBS	0	69.23	1.01

6.5.2 Büyük ölçekli problem

Bu kısımdaki problem 10 iş, 10 proses, 10 makina ölçeğinde olup bahsi geçen 4 algoritma birer kez çalıştırılmıştır. Ortaya çıkan sonuçlar Şekil 6.42 ve Çizelge 6.12'de görülebilir. Karınca kolonisi optimizasyonu en iyi sonucu vermiş ancak çok fazla süre kullanmıştır. Şekil 6.42'ye baktığımızda melez yaklaşımın Karınca kolonisi optimizasyonu ile aynı zaman dilimlerinde benzer sonuçlar verdiklerini görüyoruz. Sonuç olarak melez yaklaşım çok daha az süre kullanarak yaklaşık olarak benzer bir sonuç üretmiştir. Genetik algoritma ve Yapay bağışıklık sistemi ise kullandıkları zaman az olmasına karşın ürettikleri sonuç bakımından diğer algoritmaların gerisinde kalmışlardır.

Şekil 6.42 Dört algoritmanın büyük ölçekli problemde kıyaslanması



Çizelge 6.12 Dört algoritmanın büyük ölçekli problemdeki sonuçları

Algoritmalar	SR	ARPD	CPU
KKO	100	0	125.53
GA	0	178.3	16.3
Melez	0	22.13	9
YBS	0	189.51	4.32

Sonuç olarak Çizelge 6.13'teki gibi algoritma tercih tablomuz üzerinde bütün algoritmaları gösterirsek melez yaklaşımın her problem tipi için kulanılabilir halde olduğunu da görürüz.

Çizelge 6.13 Bütün algoritmalar için tercih tablosu

Zaman & Ölçek	Küçük problem	Büyük problem
Uzun zaman	GA/Melez	KKO/Melez
Kısa zaman	GA/KKO/Melez	KKO/Melez

BÖLÜM 7

SONUÇ VE ÖNERİLER

Çizelgeleme problemleri, hesaplama karmaşıklıklarının yüksek olması sebebiyle boyutları büyüdükçe bir matematiksel formülasyon dahilinde çözülmeleri imkansız hale gelen problemlerdir. Üretim sistemlerinin verimliliklerini etkilemelerinden ötürü üretim tesislerinde optimize edilmeleri üzerine uğraşılan problemlerden olmuş ve Endüstri mühendisliğinin önemli konularından birisi haline gelmiştir.

Akademik çalışmaların başlangıç aşamalarındaki ortak bir sorun, literatürde daha önce yapılmış olan çalışmaların çıktılarını bir araya getirebilmek ve gerçekleştirilen çalışmaları öncülleri ile kıyaslayabilmektir. Birbirinden ayrı platformlarda, ayrı programlama dilleri ile yazılan veya farklı bilgisayarlarda farklı problemlerle çalıştırılan algoritmaların bir araya getirilmeleri ve birbirleri ile kıyaslanmaları zorlaşmaktadır.

Yapılan çalışma kapsamında öncelikle, çizelgeleme problemleri için öncelikle bahsedilen bu gibi durumlar için araştırmacıların işlerini kolaylaştıracak bir çizelgeleme paket programı tasarlanmıştır. Programın temel modüllerinden ilki, farklı tiplerde rastgele verilerle çizelgeleme problemleri oluşturabilen kısımdır. Örnek bir problem oluşturma ekranı Şekil 5.2'de görülebilir.

- Bu kısım kullanılarak Atöyle tipi veya Akış tipi bir çizelgeleme problemi oluşturulabilir. Oluşturulan probleme ait proses süreleri Uniform (tekdüze) dağılım ile rassal bir şekilde program tarafından doldurulur, ayrıca kullanıcının bu tabloları istediği şekilde düzenlemesine de imkan tanınır.

- Akış tipi problemler için enküçüklenmek istenen ölçüt tamamlanma zamanı veya toplam akış zamanı olarak belirlenebilir. Probleme sıra bağımlı, makinalara göre değişkenlik gösterebilen hazırlık zamanı tablosu eklenebilir. Her işin için öncelik kısıtlarına sahip istenilen sayıda proses tanımlanabilir. Ayrıca problem, işlerin prosesleri arasında beklemeye izin verilmeyen beklemesiz tipte de seçilebilir.
- Programın problemin proses sürelerine atayacağı tekdüze dağılıma sahip rassal değerlerin üst limiti de kullanıcı tarafından değiştirilebilir. Ayrıca bu değerler bir çarpan ile çarpılıp probleme de eklenebilir. Hazırlık zamanlarının üst limiti, proses zamanlarından bağımsız olarak belirlenebilir.
- Ayrıca çizelgeleme problemine bulanık yaklaşımlar uygulanmak istenirse, bunun için proses değer tablosu arayüzü program tarafından seçilen bulanık yaklaşıma uygun şekilde ayarlanır.

Problem oluşturulduktan sonra problem tipi için çalışmamızda belirtilen sistemde tanımlı algoritmaların listesi kullanıcıya sunulur. Kullanıcı bu liste içinden istediği bir veya daha fazla algoritmayı seçip bunların parametrelerini düzenleyebilir. Daha sonra algoritmaların kaç ayrı problemde, veya aynı problemde kaç kez çalıştırılması istenildiğini seçer. Meta-sezgisel algoritmalar her çalıştıklarında aynı sonucu vermezler, verdikleri sonuçlar bir miktar rassaldır. Dolayısıyla çalışmamızda kullanıcıya sunulan algoritmaların birden fazla çalıştırılmaları, daha iyi bir kıyas yapabilmek açısından tercih edilmelidir.

Çalışmamızdaki algoritmalar sonuçlandıklarında elde edilen sonuç programın Gantt diyagramı gösterim modülünde kullanıcıya gösterilir. Bu diyagramın yatay kısmında proseslerin çalıştığı birim zaman görülür. Dikey kısmında ise sırasıyla proses planında bulunan makinalar her birisi gri renkte yatay şeritler halinde görülmektedir. Bu şeritlerin üzerinde kırmızı renkli dikdörtgenler halinde çizelgelenen prosesler yer alır. Proseslerin üzerinde ise beyaz renkte prosesin ait olduğu işin numarası ve proses numarası bulunur. Çalışmamızdan önrek bir Gantt diyagramı modülü görüntüsü Şekil 5.8'de görülebilir.

Doktora çalışmamızda geliştirilen paket program kapsamında getirilen bir yeni yaklaşım da meta-sezgisel algoritmaları daha doğru kıyaslayabilmek için grafik gösterim kullanmaktır. Meta-sezgisel algoritmalar, geleneksel algoritmalar veya sezgisellerden farklı olarak belli bir prosedür çerçevesinde belli algoritmik adımları işleyip sonlanmazlar. Dolayısıyla sadece en son ürettiği çözüm ve bu çözümü bulduğu süreyi karşılaştırmak kullanıcıya yeterince iyi bir kıyas imkanı vermez. Meta-sezgisel algoritmalar, çalışmaları esnasında rastgele çözümler ile başlayıp sürekli bu çözümleri iyileştirmeye çalışır ve öngörülen bir süre, iterasyon veya çözüm sınırı koşuluna göre dururlar. Dolayısıyla meta-sezgisellerin çalışmaları esnasında ürettikleri ara çözümler de değerlidir ve kıyaslama için bu değerlerin de kullanılması daha doğrudur.

Çalışmamızdaki programın bir başka modülü de algoritmaların ürettikleri çözümün tamamlanma süresi ve çözümün bulunma zamanına ilişkin sayısal bazı veriler içeren sonuç tablosudur. Bu tablo literatürde de kullanılan bir karşılaştırma biçiminin paket programa uygulanmış halidir. Üç ana sütun içerir: Birincisi çalıştırılan algoritmaların, kıyas yapılan problemlerin ne kadarlık bir yüzdesinde en iyi sonucu bulduğunu gösteren "Başarı oranı"dır. İkincisi, algoritmaların her problem için, o problemde üretilen en iyi çözümden ne kadar yüzde ile saptıklarını belirten "Ortalama göreli sapma oranı"dır. Üçüncüsü ise algoritmanın ne kadar sürede sonlandığını gösteren "Çalışma süresi"dir.

Çalışmamızda yazılan program herhangi bir İnternet tarayıcısında çalışacak şekilde tasarlanmıştır. Böylece her yerden hızlı ve kolayca ulaşım sağlanabilir. Ayrıca programın akıllı telefon ve tablet gibi cihazlarda çalışabilecek bir mobil sürümü yazılmıştır, ancak bu sürüme bütün özellikler konulmamıştır.

Çalışmamızda bahsedilen program altyapısı hazırlandıktan sonra bir literatür araştırması yapılmış, çizelgeleme problemleri için önerilen birçok algoritma birçok farklı çalışmadan derlenmiştir. Derlenen bu algoritmalar yazılan programa tek tek kodlanmıştır. Ayrıca yapılan literatür araştırmasında, çizelgeleme problemlerinin, melez sezgisel algoritmalar ile çözümü hakkında yeterli sayıda çalışma olmadığı görülmüş ve bu alanda ilerlenilmesine karar verilmiştir. Doktora çalışmamızdaki programa eklenen belli başlı algoritmalar şunlardır:

- Karınca kolonisi optimizasyonu: Bu algoritma karınca sürüsünün birlikte yiyecek arama işlemini simüle eder. Karıncalar sürü halinde bir yiyecek ararken başta

etrafa dağılırlar. Bazıları rassal sebeplerle yiyeceğe daha erken ulaşır. Gezdikleri yerlere diğer karıncalara çekici gelen feromon ismi verilen bir çeşit koku bırakan bu karıncalar, diğerlerini kendi yollarına yönlendirmiş olurlar. Yiyeceğe erken ulaşan karıncalar geri dönerlerken daha fazla feromon bırakacakları için onların gittiği yollar sürüye daha çekici gelir ve diğerleri de benzer yolları tercih etmeye başlarlar. Böylece, feromon yoğun yollarda daha fazla karınca yiyecek arar. Dolayısıyla daha hızlı bir rota bulunması ihtimali yükselir. Karınca kolonisi optimizasyonu ile çizelgeleme problemleri çözülürken, yapılabilecek çizelgeler birer rota şeklinde sembolize edilir. Bir prosesin bir makinada işlenmesi bir düğüm olacak şekilde, düğümler arası yollardan oluşan büyük bir çizge içinde sanal karıncalar çözüm ararlar. Başlangıçta bir miktar rastgele bulunan çözümlerden en iyisinin rotasına bir miktar feromon bırakılır. Karıncalar rota seçerken yoldaki feromon oranı ve hedef düğümdeki prosesin çalışma zamanı gibi faktörlerin belli katsayılarla ağırlıklandırılması sonucunda karar verirler. Bu ağırlıklar da kullanıcı tarafından değiştirilebilir. Her turda belli sayıda karınca çözüm araması için çizgeye bırakılır ve tur sonunda en iyi çözüme göre feromon güncellemesi yapılır. Son iterasyon sonucunda bulunan en iyi çözüm de kullanıcıya gösterilir.

- Genetik algoritma: Bu algoritma biyolojik evrim teorisinden esinlenmiş bir algoritmadır. Popülasyonların zamana yayılmış yavaş çevresel değişimlere sonraki nesillerde genetik özellik frekanslarının genetik avantaj durumuna göre azalıp artması ile uyum sağlaması, ana mekanizmasıdır. Buna doğal seçilim denir. Genetik algoritma'da başlangıçta belli miktarda rastgele çözümler oluşturulur. Oluşturulan bu çözümler istenilen ölçüte uyumlu olup olmamasına göre bir sonraki nesilde yaşatılır veya öldürülür. Hayatta kalan çözümler kendi aralarında çaprazlanıp yeni çözümler oluştururlar, bu da genetik çeşitliliği sağlar. Ayrıca bir miktar çözüme de mutasyon uygulanır ve genetik çeşitlilik artırılır. Belli nesil sayısı sonunda, bulunan en iyi çözüm kullanıcıya gösterilir.
- Yapay bağışıklık sistemi: Bu algoritma, bağışıklık sistemimizin antijenlere karşı direnç sağlamaya adapte olmasından ilham alır ve Genetik algoritma ile benzerlikler taşır. Genetik algoritmadaki kromozomlar yerine antikorlar vardır,

bunlardan iyi olanları ile devam edilir ve bunlar klonlanır. Oluşan antikor listesine Somatik hipermutasyon veya Reseptör denetimi isimli belli mutasyon adımları uygulanır. Bundan sonra en kötü antikorlar silinip yerlerine en iyileri klonlanır ve bu işlem belli sayıda iterasyon boyunca sürdürülür. Sonuçta bulunan en iyi çözüm kullanıcıya gösterilir.

Çalışmamızdaki program oluşturulduktan ve literatürdeki belli başlı algoritmalar eklendikten sonra programın test altyapısı kullanılarak, Genetik algoritma ve Karınca kolonisi için parametreler iyileştirilmeye çalışılmıştır. Yapılan testler sonucunda bulunan optimum parametre değerleri Çizelge 6.3'te görüldüğü gibidir. Elde edilen bu parametre değerleri kullanılarak iki algoritma çeşitli problem ölçeklerinde birbirleriyle karşılaştırılmışlardır. Genetik algoritma küçük ölçekli problemlerde veya uzun süre hesaplama zamanı verilen durumlarda daha başarılı sonuç bulmuşken, Karınca kolonisi optimizasyonu daha kısa çalışma sürelerinde veya büyük problem ölçeklerinde daha başarılı olmuştur. Çalışmamızdaki bu sonuç da Çizelge 6.4'te görülebilir. Algoritmaların farklı durumlarda daha başarılı olmalarından hareketle bir melez yaklaşım geliştirilmiştir. Geliştirililen yaklaşımın ayrıntıları aşağıdaki gibidir:

- Melez Karınca kolonisi – Genetik algoritma: Bu algoritma çalışmada önerilen özgün bir melez yaklaşımıdır. Yaklaşımın temel mantığı bir algoritmadan üretilen sonucu diğerini kullanarak iyileştirmeye çalışmaktır. İki algoritma da çalıştırılırken belli süre sonunda iyileştirme sağlanamazsa, yerel bir çözümde tıkandığı düşünülerek sıfırlanır. Böylece bir algoritma sonlanıp diğeri çalışmaya başlandığında tek bir sonuç değil, elde edilen birçok yerel iyileştirilmiş sonuç diğer algoritmaya gönderilmeyi bekleyecektir. Hangi algoritmanın önce çalışacağı programın başlangıcında algoritma tarafından kararı verilen bir faktördür. Yapılan test çalışmaları sonucunda belli ölçekteki problemler için belli algoritmaların daha iyi çalıştığı görülmüştür. Örneğin 50 prosesten küçük problemlerde Genetik algoritma daha başarılı çalışırken daha büyük problemlerde Karınca kolonisi optimizasyonu'nun daha iyi çalıştığı gözlemlenmiştir. Böylece beraber başlatılan algoritmalardan verilen problem ölçeğine göre daha iyi sonuçlar bulmaya başlayan algoritma devam ettirilir, diğeri durdurulur. Böylece sınırlı yerel optimum iyileştirmesi durumundan bir

nebze kaçınılmış olur. İstenilen problem ölçeğinde daha iyi çalışan algoritma önce çalıştırılarak daha fazla iyileştirme sağlanabilir.

Çalışmamızda daha sonra Yapay bağışıklık sistemi ile Genetik algoritma çeşitli ölçeklerdeki problemlerde birbirleriyle kıyaslanmışlardır. Genetik algoritma küçük ölçekteki problemlerde daha başarılı çıkarken Yapay bağışıklık sistemi büyük ölçekli problemlerde daha iyi sonuç bulmuştur. Bu bölüme ilişkin sonuçlar Çizelge 6.10'da görülebilir.

Son olarak çalışmamızda geliştirilen melez yaklaşımın beklenildiği gibi hem küçük ölçekli hem büyük ölçekli problemlerde başarılı olup olmadığı test edilmiştir. Yapılan testlerde iki ölçekte de bulduğu sonuçların o ölçekte iyi çalışan algoritmanın sonuçları civarında olduğu görülmüştür. Farklı problem ölçeklerinde iyi sonuçlar bulabilmesi melez algoritmaları başarılı kılmaktadır. Bu kısıma ilişkin sonuçlar Çizelge 6.13'te görülebilir.

Çalışmanın diğer kısmında ise çizelgeleme problemleri için bulanık mantık yaklaşımları araştırılmıştır. İki farklı yaklaşım önerilmiştir, bunlardan birisi makinaların performanslarının değişkenliğinden hareketle işlem sürelerinin bulanıklaştırılması temelli, bir diğeri ise makinaların doluluk oranının bulanıklaştırılması temellidir. Performans temelli yaklaşımda makinaların proses başlangıç ve bitişlerinde %100 performansla çalışamayacağı varsayılır ve makinaların performans zaman grafiklerinin bir TFN şeklinde ifade edilebileceği varsayılır. Kullanıcı tarafından girilen performansın düşük olduğu bölge büyüklüğüne göre oluşan TFN'nin tekrar durulaştırılması sonucu başlangıçtaki proses zamanından farklı bir zaman bulunur. Bu işlem bütün proseslere uygulanır, böylece başlangıçtan farklı sürede sonlanan bir çizelge elde edilmiş olur. Burada amaç çizelgeyi daha hızlı bitirmek değil, gerçeğe daha uygun bir çizelge elde etmektir.

Doluluk temelli yaklaşım da iş paketlerinin tek bir iş olarak tanımlanabileceği ortamlarda geçerlidir. Bu durumda bir işin makinaya yerleştirilmesi belli bir süre alacaktır. Böylece makinaların dolulukları belli bir yüzde ile, dolayısıyla yine bir TFN ile ifade edilebilir. Bu varsayım sonucunda aynı anda birden fazla işin belli yüzdelerinin aynı anda bir makina tarafından yapılabilmesi mümkün hale gelir. Böylece işlerin uc uca eklenmesi için, iş ve makina beklemelerinin hesaplanması için veya algoritmaların

kıyaslanması için yeni işlemler tanımlanması gerekir. Çalışmada bunlar da formülize edilmiştir.

Tezde; çalışılan alanda daha sonra araştırma yapmayı düşünen araştırmacılar için getirilebilecek birçok öneri vardır. Bu öneriler, paket programın geliştirilmesi, başka sezgisel algoritmalar geliştirilmesi veya bulanık mantık konusunda yapılabilecekler şeklinde üç ana gruba ayrılabilir.

Öncelikle, çalışmamızda yazılan paket program, halihazırda daha sonraki araştırmacıların iş yükünü azaltmak ve çalışmalarını hızlandırmak üzere geliştirilmiştir. Ancak daha iyi çalışması için üzerine eklenebilecek birçok özellik vardır. Bunlardan bazıları şunlardır:

- Program İnternet tarayıcılarında çalışmak üzere Javascript dilinde yazılmıştır. Sahip olduğu birçok avantajdan dolayı Javascript dili tercih edilmiştir. Bu sayede herhangi bir internet sayfasında, paket program Dünya'nın her yerindeki insanların erişimine açık, istenildiği an güncellenebilir bir şekilde bulunabilir. İnsanlar, bilgisayarlarına bir şeyler indirmek zorunda kalmadan anında programı çalıştırabilir ve sonuçlarını görebilir. Ayrıca programın işlemci yükü sunucuda değil istemci cihazlarda olduğundan, çok fazla kişinin aynı anda programı çalıştırması bir sorun teşkil etmez. Ancak Javascript'in kendine has çalışma özellikleri sebebiyle programın çalışması esnasında sayfada güncelleme yapılamamaktadır. Bu da uzun süren iterasyonlarda kullanıcıların programın ne zaman sonlanacağını bilememesine yol açmaktadır. Eğer iterasyonlar döngü içinde değil de kısa bir süre durakladıktan sonra birbirini çağıran fonksiyonlar şeklinde tasarlanırsa iterasyonların çalışması esnasında kullanıcılar, anlık olarak bilgilendirilebilir, böylece kullanım deneyimi yükseltilmiş olur.
- Program rassal sayılar belirlerken Gauss dağılımı yerine tekdüze dağılım kullanmaktadır. Gerçek çizelgeleme problemlerine yakın olabilmesi sebebiyle rassal dağılımın hangi fonksiyon ile gerçekleştirileceğine dair kullanıcıya seçim şansı sunulması sağlanabilir.
- Diğer kullanıcıların, sisteme oluşturdukları problemleri kaydedebilecekleri veya kendi algoritmalarını ekleyebilecekleri bir altyapı yazılabilir. Böylece yazılan

program, çizelgeleme problemleri açısından güncel, hızlı ve kullanışlı bir literatür taraması aracı olabilir.

- Eklenen problemler, site dışına taşınmak veya dışarıdan problem eklemek için bir çizelgeleme problemi formatı haline getirilebilir. Örneğin XML dili bunun için tercih edilebilir.

Yukarıda bahsedilenlerin yanında çalışmamızdaki algoritma geliştirme kısmı için de yapılabilecek ek geliştirmeler mevcuttur.

- Örnağin farklı yaklaşımlı melez algoritmalar gerçekleştirilebilir. Meta-sezgisel algoritmaların çalışma biçiminden ötürü birinin yaptığı iyileştirmeyi diğerinin devam ettirmesi sınırlı bir iyileştirme sağlamaktadır. Bunun yerine algoritmayı yerel optimum noktalarından kurtarabilecek melez yaklaşımların tercih edilmesi daha iyi çözümlerin oluşturulmasını sağlayabilir.
- Melez yaklaşımlar yerine özgün bir çizelgeleme meta-sezgiseli tasarlanması da düşünülebilecek seçenekler arasındadır. Ayrıca şu anda programa eklenmemiş olan başka meta-sezgiseller de eklenebilir. Örneğin; Arı kolonisi optimizasyonu, Parçacık sürüsü optimizasyonu ve Tavlama benzetimi eklenebilecek algoritmalar arasındadır.
- Genetik algoritmaya ilişkin literatürde bulunan mutasyon ve çaprazlama yöntemlerinin henüz programda bulunmayanları da eklenebilir. Böylece kullanıcıya daha geniş bir Genetik algoritma opsiyonu sunulabilir.

Çalışmamızdaki Bulanık mantık yaklaşımı için de yapılabilecek ilerlemeler bulunmaktadır.

- Performans temelli Bulanık mantık yaklaşımında farklı durulaştırma metodlarının kıyaslanması yapılabilir.
- Doluluk temelli bulanık mantık yaklaşımında ise gerekli işlemler tanımlanırken ihmal edilen noktalar belirtilmiştir. Bu noktaların oluşturabileceği hata payları hesaplanabilir, daha az ihmal gerektiren daha kompleks işlem formülasyonları oluşturulabilir ve bu yaklaşım ile ortaya çıkacak çizelge farklılıklarına ilişkin istatistiksel bir veri ortaya konulabilir.

Çalışmamızda özetle;

1. Yapılan çalışmalar neticesinde çizelgeleme problemlerinin oluşturulması ve çözümü konusunda yapılmak istenen açık kaynak kodlu platform istenildiği gibi oluşturulmuştur.
2. Başarılı bir şekilde rastgele çizelgeleme problemi oluşturup kullanıcıların düzenleme yapmasına izin verebilir durumdadır.
3. Literatürde çizelgeleme problemleri için kullanılan Yapay bağışıklık sistemi, Karınca kolonisi optimizasyonu ve Genetik algoritma gibi belli başlı algoritmalar bu platforma yerleştirilmiş ve oluşturulan test arayüzü ile parametreleri optimize edilmiştir.
4. Daha sonra bu çalışmalar baz alınarak Karınca kolonisi optimizasyonu – Genetik algoritma melezi bir çözüm geliştirilmiş, performansı test edilmiştir. Yapılan testler sonucunda geliştirilen melez çözüm istenilen ölçüde başarılı bulunmuştur.
5. Bulank mantık yaklaşımı çizelgeleme problemlerine doluluk veya performans bulanıklaştırma gibi birkaç yöntem ile uygulanmış, geliştirilen yaklaşım açıklanmış, test edilmiştir. Geliştirilen bu platformun özellikle daha sonraki çizelgeleme çalışmalarında araştırmacılara dizayn, test ve karşılaştırma aşamalarında yardımcı olabileceği düşünülmüştür.

KAYNAKLAR

[1] Coşkun, A., (2007). ”Yapay zeka optimizasyon teknikleri: Literatür Değerlendirmesi”, Doğu Anadolu Bölgesi Araştırmaları.

[2] Araújoa, D.C. ve Nagano, M.S., (2011). “A new effective heuristic method for the no-wait flowshop with sequence-dependent setup times problem”, International Journal of Industrial Engineering Computations, 2: 155-166.

[3] Bianco, L., Dell’Olmo, P. ve Giordani, S., (1999). “Flow shop no-wait scheduling with sequence dependent setup times and release dates”, INFOR, 37: 3-19

[4] Brown, S.I., McGarvey, R.G. ve Ventura, J.A., (2004). “Total flowtime and makespan for a no-wait m-machine flowshop with set-up times separated”. Journal of the Operational Research Society, 55(6), 614-621.

[5] Pugazhenthi, R. ve Xavior, M.A., (2014). “Computation of Makespan Using Genetic Algorithm in a Flowshop”, American-Eurasian Journal of Scientific Research, 9: 105-113.

[6] Allahverdi, A. ve Aldowaisan, T., (2002). “Better Heuristics for M-Machine No-wait Flowshop with Total Completion Time Criterion”, The 6th Saudi Engineering Conference, 4: 551-560.

[7] Laha, D. ve Sapkal, S.U., (2014). “An improved heuristic to minimize total flow time for scheduling in the m-machine no-wait flow shop”, Computers & Industrial Engineering, 67: 36-43.

[8] Aldowaisan, T., (2001). “A new heuristic and dominance relations for no-wait flowshops with setups”, Computers & Operations Research, 28: 563-584.

[9] Zhu, X., Li, X. ve Wang, Q., (2008). “Hybrid Heuristic for m-Machine No-Wait Flowshops to Minimize Total Completion Time”, Computer Supported Cooperative Work in Design IV, 5236: 192-203.

[10] Kumar, G. ve Singhal, S., (2013). “Genetic Algorithm Optimization of Flowshop Scheduling Problem with Sequence Dependent Setup Time and Lot Splitting”, International Journal of Engineering, Business and Enterprise Applications (IJEBEA), 62-71.

[11] Tseng, L. ve Lin, Y., (2010). “A hybrid genetic algorithm for no-wait flowshop scheduling problem”, International Journal of Production Economics, 128:144-152.

[12] Carlier, J., (1978). “Ordonnancements a contraintes disjonctives”. RAIRO Recherche Operationnelle 12, 333–351

[13] Taillard, E., (1990). “Some efficient heuristic methods for the flow shop sequencing problem”. European Journal of Operations Research 47, 65–74.

[14] Chaudhry, I.A. ve Mahmood, S., (2012). “No-wait Flowshop Scheduling Using Genetic Algorithm”, No-wait Flowshop Scheduling Using Genetic Algorithm, 3: 4-6.

[15] Aldowaisan, T. ve Allahverdi, A., (1998). “Total flowtime in no-wait flowshops with separated setup times”, Computers & Operations Research, 25: 757-765.

[16] Barto, A.G., Bradtke, S.J. ve Singh, S.P., (1995). “Learning to act using real-time dynamic programming”, Artificial Intelligence, 72: 81-138.

[17] Bruner, R., Caminero, A.C., Rana, O.F., Freitag, F. ve Navarro, L., (2012). “Network-aware summarisation for resource discovery in P2P-content networks”, Future Generation Computer Systems, 28: 563-572.

[18] Wieczoreka, M., Hoheiselb, A. ve Prodana, R., (2009). “Towards a general model of the multi-criteria workflow scheduling on the grid”, Future Generation Computer Systems, 25: 237-256.

[19] Cuevas-Tello, J.C., Tiňo, P., Raychaudhury, S., Yao, X. ve Harva, M., (2010). “Uncovering delayed patterns in noisy and irregularly sampled time series: An astronomy application”, Pattern Recognition, 43: 1165-1179.

[20] Kashan, A.H. ve Karimi, B., (2009). “A discrete particle swarm optimization algorithm for scheduling parallel machines”, Computers & Industrial Engineering, 56: 216-223.

[21] Prot, D. ve Bellenguez-Morineau, O., (2012). “Tabu search and lower bound for an industrial complex shop scheduling problem”, Computers & Industrial Engineering, 62: 1109-1118.

[22] Vidal, T., Crainic, T.G., Gendreau, M. ve Prins, C., (2014). “A unified solution framework for multi-attribute vehicle routing problems”, European Journal of Operational Research, 234: 658-673.

[23] Pacini, E., Mateos, C. ve Garino, C.G., (2014). “Distributed job scheduling based on Swarm Intelligence: A survey”, Computers & Electrical Engineering, 40: 252-269.

[24] Zandieh, M., Ghomi, S.M.T.F. ve Husseini, S.M.M., (2006). “An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times”, Applied Mathematics and Computation, 180: 111-127.

[25] Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P. ve Fasih, A., (2012). “PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation”, Parallel Computing,38: 157-174.

[26] Kuo, R.J. ve Lin, L.M., (2010). “Application of a hybrid of genetic algorithm and particle swarm optimization algorithm for order clustering”, Decision Support Systems, 49: 451-462.

[27] Burdett, R. ve Kozan, E., (2014). “Determining operations affected by delay in predictive train timetables”, Computers & Operations Research, 41: 150-166.

[28] Malczewski, J., (2004). “GIS-based land-use suitability analysis: a critical overview“, Progress in Planning, 62: 3-65.

[29] Mellit, A. ve Kalogirou, S.A., (2014). “MPPT-based artificial intelligence techniques for photovoltaic systems and its implementation into field programmable gate array chips: Review of current status and future perspectives”, Energy, 70: 1-21.

[30] Shahabudeen, P. ve Sivakumar, G.D., (2008). “Algorithm for the design of single-stage adaptive kanban system”, Computers & Industrial Engineering, 54: 800-820.

[31] Dahmani, N., Clautiaux, F., Krichen, S. ve Talbi, E., (2013). “Iterative approaches for solving a multi-objective 2-dimensional vector packing problem”, Computers & Industrial Engineering,66: 158-170.

[32] Sherali, H.D. ve Driscoll, P.J., (2000). “Evolution and state-of-the-art in integer programming”, Journal of Computational and Applied Mathematics, 124: 319-340.

[33] Barthélemy, J.P., Bisdorff, R. ve Coppin, G., (2002). “Human centered processes and decision support systems”, European Journal of Operational Research, 136: 233-252.

[34] Strang, K.D., (2012). “Importance of verifying queue model assumptions before planning with simulation software”, European Journal of Operational Research, 218: 493-504.

[35] Burgess, C.J. ve Lefley, M., (2001). “Can genetic programming improve software effort estimation? A comparative evaluation”, Information and Software Technology, 43: 863-873.

[36] Li, J., Burke, E.K. ve Qu, R., (2011). “Integrating neural networks and logistic regression to underpin hyper-heuristic search”, Knowledge-Based Systems, 24: 322-330.

[37] Dey, S., Saha, I., Bhattacharyya, S. ve Maulik, U., (2014). “Multi-level thresholding using quantum inspired meta-heuristics”, Knowledge-Based Systems, 67: 373-400.

[38] Nahas, N., Nourelfath, M. ve Ait-Kadi, D., (2007). “Coupling ant colony and the degraded ceiling algorithm for the redundancy allocation problem of series–parallel systems”, Reliability Engineering & System Safety, 92: 211-222.

[39] Kianfar, K., Ghomi, S.M.T.F. ve Jadid, A.O., (2012). “Study of stochastic sequence-dependent flexible flow shop via developing a dispatching rule and a hybrid GA”, Engineering Applications of Artificial Intelligence, 25: 494-506.

[40] Liu, M., Sun, Z., Yan, J. ve Kang, J., (2011). "An adaptive annealing genetic algorithm for the job-shop planning and scheduling problem", Expert Systems with Applications, 38: 9248-9255.

[41] Jolai, F., Asefi, H., Rabiee, M. ve Ramezani, P., (2013). "Bi-objective simulated annealing approaches for no-wait two-stage flexible flow shop scheduling problem", Scientia Iranica, 20: 861-872.

[42] Sloman, A. ve Chrisley, R.L., (2005). "More things than are dreamt of in your biology: Information-processing in biologically inspired robots", Cognitive Systems Research, 6: 145-174.

[43] Sugimura, N., Hino, R. ve Moriwaki, T., (2001). "Integrated process planning and scheduling in holonic manufacturing systems", In Proceedings of IEEE international symposium on assembly and task planning soft research park, Fukuoka, Japan, 4, pp. 250–254.

[44] Baker., K.R. ve Trietsch, D., (2009). "Principles of sequencing and scheduling", John Wiley & Sons, New Jersey.

[45] Graham, R.L., Lawler, E.L., Lenstra, J.K. ve Kan, A.R., (1979). "Optimization and approximation in deterministic sequencing and scheduling: a survey", Annals of discrete mathematics, 5, 287-326.

[46] Leung, C.W., Wong, T.N., Mak, K.L. ve Fung, R.Y., (2010). "Integrated process planning and scheduling by an agent-based ant colony optimization", Computers & Industrial Engineering, 59(1), 166-180.

[47] Engin, O. ve Döyen, A., (2004). "A new approach to solve hybrid flow shop scheduling problems by artificial immune system", Future generation computer systems, 20(6), 1083-1095.

[48] Ho, N.B., Tay, J.C. ve Lai, E.M.K., (2007). "An effective architecture for learning and evolving flexible job-shop schedules", European Journal of Operational Research, 179(2), 316-333.

[49] Zadeh, L.A., (1965). "Fuzzy sets", Information and control, 8(3), 338-353.

[50] Lee, H.K., (2005). "First course on fuzzy theory and applications", Springer.

[51] Kwong, C.K. ve Bai, H., (2003). "Determining the importance weights for the customer requirements in QFD using a fuzzy AHP with an extent analysis approach", Iie Transactions, 35(7), 619-626.

[52] Sakawa, M. ve Kubota, R., (2000). "Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms", European Journal of Operational Research 120, 393 – 407.

[53] Murata, T., Ishibuchi, H. ve Tanaka, H., (1996). "Multi-objective genetic algorithm and its applications to flowshop scheduling", Computers & Industrial Engineering, 30(4), 957-968.

[54] Lee, D.Y. ve DiCesare, F. (1992). "FMS scheduling using Petri nets and heuristic search". In Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on (pp. 1057-1062). IEEE.

EK-A

KARINCA KOLONİSİ OPTİMİZASYONU

```
function Node(){
   this.jobID =-1;   this.machineID =-1;
   this.processID =-1;  this.processingTime =0;
   this.arcList = new Array();      this.arcIndex = new Array();
   this.superNode;   this.initWithVariable = function(jobID,processID,machineID,time){
      this.jobID = jobID;        this.machineID = machineID;
      this.processID = processID;      this.processingTime = time;}}
function SuperNode(){
   this.jobID = -1;  this.processID = -1;
   this.arcList = new Array();      this.initWithVariable = function(jobID,processID){
      this.jobID = jobID;        this.processID = processID;}}
function Arc(){
   this.sourceNode = new Node(); this.destinationNode = new Node();
   this.pheromoneLevel = 0;      this.desirability = 0;
   this.setupTime = 0;
   this.initWithVariable = function(sourceNode, destinationNode, pheromoneLevel,
desirabilityConstant, setupTime){
      this.sourceNode = sourceNode;       this.destinationNode = destinationNode;
      this.pheromoneLevel = pheromoneLevel;
      this.desirability = desirabilityConstant/(this.setupTime +
this.destinationNode.processingTime);
      this.setupTime = setupTime;}}
function Map(){
   this.jobList = new Array();      this.nodeList = new Array();
   this.arcList = new Array();      this.superNodeList = new Array();
   this.startNode = new Node();
   this.initWithVariable = function(nodeList,jobList,tho0,C,setupTimeList){
   this.nodeList = nodeList;
      this.jobList = jobList;
      for(var i=0;i<this.jobList.length;i++){
         for(var j=0;j<this.jobList[i][0].length;j++){
            var tempNode = this.jobList[i][0][j];
            var tempArc = new
Arc();            tempArc.initWithVariable(this.startNode,tempNode,tho0,C,setupTimeList[
j][i][i]);
            this.arcList.push(tempArc);
            this.startNode.arcList.push(tempArc);}}
      for(var i=0;i<this.jobList.length;i++){
         for(var j=0;j<this.jobList[i].length -1;j++){
            for(var k=0;k<this.jobList[i][j].length;k++){
                  for(var l=0;l<this.jobList[i][j+1].length;l++){
                     var node1 = this.jobList[i][j][k];
                     var node2 = this.jobList[i][j+1][l];
                     var tempArc = new Arc();
                     tempArc.initWithVariable(node1,node2,tho0,C,0);
                     this.arcList.push(tempArc);
                     node1.arcList.push(tempArc);}}}}
      for(var i=0;i<this.jobList.length;i++){
         for(var j=0;j<this.jobList[i].length;j++){
            for(var k=0;k<this.jobList[i][j].length;k++){
                  for(var x=0;x<this.jobList.length;x++){
                     if(x==i){
```

```
                        continue;}
                     else{
                        for(var y=0;y<this.jobList[x].length;y++){
                              for(var z=0;z<this.jobList[x][y].length;z++){
                                 var node1 = this.jobList[i][j][k];
                                 var node2 = this.jobList[x][y][z];
                                 var tempArc = new Arc();
                                 tempArc.initWithVariable(node1,node2,tho0,C,0);
                                 this.arcList.push(tempArc);
                                 node1.arcList.push(tempArc);}}}}}}}}
   this.printPheromoneLevel = function(){
      var string = '';
      for(var i=0;i<this.arcList.length;i++){
         string += this.arcList[i].pheromoneLevel.toFixed(1)+'-';}}}
function Ant(){
   this.antID; this.supervisor;
   this.Xk; this.Gk;
   this.Sk; this.tabuK = new Array();
   this.arcK = new Array();   this.jobProcessQueue = new Array();
   this.currentNode;    this.probabilityTable = new Array();
   this.probOrderTable = new Array();
   this.processMachineAssignList = new Array();
   this.machineOrderAssignList = new Array();
   this.machineLengthList = new Array();
   this.haco3;
   this.initWithVariable = function(startNode,Gk,supervisor,antID){
      this.Gk = Gk.slice(0);        this.Sk = startNode.arcList.slice(0);
      this.currentNode = startNode;       this.supervisor = supervisor;
      this.antID = antID;
      this.jobProcessQueue = new Array(this.supervisor.jobCount);
      for(var i=0;i<this.supervisor.jobCount;i++){
         this.jobProcessQueue[i]=0;}}
   this.getMakespan = function(arcList,tabuList,machineNumber,jobNumber){
      var makespan = 0;
      var machineIndex = new Array(machineNumber);
      var jobIndex = new Array(jobNumber);
      for(var i=0;i<machineNumber;i++){
         machineIndex[i]=0;}
      for(var i=0;i<jobNumber;i++){
         jobIndex[i]=0;}
      for(var i=0;i<tabuList.length;i++){
         var startTime =
Math.max(parseInt(jobIndex[tabuList[i].jobID]),parseInt(machineIndex[tabuList[i].machine
ID]));
         var setupTime = arcList[i].setupTime;
         jobIndex[tabuList[i].jobID] = startTime + setupTime +
tabuList[i].processingTime;
         machineIndex[tabuList[i].machineID] = startTime + setupTime +
tabuList[i].processingTime;}
      for(var i=0;i<machineNumber;i++){
         if(machineIndex[i]>makespan)
            makespan=machineIndex[i];}
      return makespan;}
   this.nodeSelectionProcedure = function(){
      var chanceVariable = Math.random();
      var totalChance = 0;
      var totalChanceRatio = 0;
      for(var i=0;i<this.Sk.length;i++){
         var tempChance = 0;
         var tempArc = this.Sk[i];
         tempChance =
Math.pow(tempArc.pheromoneLevel,this.supervisor.alpha)*Math.pow(tempArc.desirability,thi
s.supervisor.beta);
         totalChance += tempChance;}
      if(this.Sk.length == 0){
         alert("Ant is stucked. sk.length:"+this.Sk.length);
         return null;}
      for(var i=0;i<this.Sk.length;i++){
         var tempChanceRatio = 0;
         var tempArc = this.Sk[i];
         tempChanceRatio =
Math.pow(tempArc.pheromoneLevel,this.supervisor.alpha)*Math.pow(tempArc.desirability,thi
s.supervisor.beta)/totalChance;
         totalChanceRatio += tempChanceRatio;
         if(totalChanceRatio >= chanceVariable){
            this.arcK.push(tempArc);
            return tempArc.destinationNode;}}
```

```
      alert("Chance error");
      return null;}
   this.haco3NodeSelectionProcedure = function(){
      var random = Math.random();
      var pro = this.haco3.processNumber;
      var mac = this.haco3.machineNumber;
      if(random<this.haco3.q){
         var maxPheromone = 0;
         var nextNode;
         for(var j=0;j<this.Sk.length;j++){
            var tempStartNode;
            if(this.Sk[j].sourceNode.jobID == -1){
                  tempStartNode = 0;
            }else{
                  tempStartNode = this.Sk[j].sourceNode.jobID*pro*mac +
this.Sk[j].sourceNode.processID*mac + this.Sk[j].sourceNode.machineID + 1;}
            var tempFinishNode =this.Sk[j].destinationNode.jobID*pro*mac +
this.Sk[j].destinationNode.processID*mac + this.Sk[j].destinationNode.machineID;
            if(this.haco3.pheromoneMatrix[tempStartNode][tempFinishNode] >
maxPheromone){
                  maxPheromone =
this.haco3.pheromoneMatrix[tempStartNode][tempFinishNode];
                  this.arcK.push(this.Sk[j]);
                  nextNode = this.Sk[j].destinationNode;}}
         return nextNode;}
      else if(random<(this.haco3.q+this.haco3.d)){
         var nextNode;
         var totalPheromone = 0;
         for(var j=0;j<this.Sk.length;j++){
            if(this.Sk[j].sourceNode.jobID == -1){
                  tempStartNode = 0;
            }else{
                  tempStartNode = this.Sk[j].sourceNode.jobID*pro*mac +
this.Sk[j].sourceNode.processID*mac + this.Sk[j].sourceNode.machineID + 1;}
            var tempFinishNode =this.Sk[j].destinationNode.jobID*pro*mac +
this.Sk[j].destinationNode.processID*mac + this.Sk[j].destinationNode.machineID;
            totalPheromone +=
this.haco3.pheromoneMatrix[tempStartNode][tempFinishNode];}
         var random2 = Math.random()*totalPheromone;
         var cumulativePheromone = 0;
         for(var j=0;j<this.Sk.length;j++){
            if(this.Sk[j].sourceNode.jobID == -1){
               tempStartNode = 0;
            }else{
                  tempStartNode = this.Sk[j].sourceNode.jobID*pro*mac +
this.Sk[j].sourceNode.processID*mac + this.Sk[j].sourceNode.machineID + 1;}
            var tempFinishNode =this.Sk[j].destinationNode.jobID*pro*mac +
this.Sk[j].destinationNode.processID*mac + this.Sk[j].destinationNode.machineID;
            cumulativePheromone +=
this.haco3.pheromoneMatrix[tempStartNode][tempFinishNode];
            if(cumulativePheromone > random2 ){
                  maxPheromone =
this.haco3.pheromoneMatrix[tempStartNode][tempFinishNode];
                  this.arcK.push(this.Sk[j]);
                  nextNode = this.Sk[j].destinationNode;
                  return nextNode;}}}
      else{
         random = Math.floor(Math.random()*this.Sk.length);
         this.arcK.push(this.Sk[random]);
         return this.Sk[random].destinationNode;}}




   this.constructSchedule = function(type){
      while(true){
J"+this.currentNode.jobID+"P"+this.currentNode.processID);
         for(var i=0;i<this.Gk.length;i++){
            if(this.Gk[i].jobID == this.currentNode.jobID && this.Gk[i].processID ==
this.currentNode.processID){
                  this.Gk.splice(i,1);
                  i--;}}
         if(this.Gk.length == 0){
            break;}
         var nextNode;
         if(type == 0){
```

```
            nextNode = this.nodeSelectionProcedure();}
         else if(type == 3){
            nextNode = this.haco3NodeSelectionProcedure();}
         if(nextNode == null){
            alert("Node selection error. gk.length:"+this.Gk.length);
            console.log("Node selection error. gk.length:"+this.Gk.length);
            return -1;}
         this.jobProcessQueue[nextNode.jobID] = parseInt(nextNode.processID)+1;
         this.tabuK.push(nextNode);
         this.Sk = nextNode.arcList.slice(0);
         for(var i=0;i<this.Sk.length;i++){
            var node1 = this.Sk[i].destinationNode;
            if(this.jobProcessQueue[node1.jobID]!=node1.processID){
                  this.Sk.splice(i,1);
                  i--;}}
         this.currentNode = nextNode;}
      this.Xk =
this.getMakespan(this.arcK,this.tabuK,this.supervisor.machineCount,this.supervisor.jobCo
unt);
      if(this.tabuK.length != this.supervisor.jobCount*this.supervisor.processCount){
         alert("Tabu list is not feasible. tabuK:"+this.tabuK.length+"
job*pro:"+(this.supervisor.jobCount*this.supervisor.processCount));
         console.log("Tabu list is not feasible. tabuK:"+this.tabuK.length+"
job*pro:"+(this.supervisor.jobCount*this.supervisor.processCount));
         return -2;}
this.getMakespan(this.arcK,this.tabuK,this.supervisor.machineCount,this.supervisor.jobCo
unt);
      return this.Xk;}
   this.calculateProbability = function(){
      var tho0 = this.supervisor.tho0;
      var alpha = this.supervisor.alpha;
      var beta = this.supervisor.beta;
      var processTable = this.supervisor.processTable;
      var setupTable = this.supervisor.setupTable;
      this.machineLengthList = new Array(this.supervisor.machineCount);
      for(var i=0;i<this.supervisor.machineCount;i++){
         this.machineLengthList[i]=0;}
      this.probabilityTable = new
Array(this.supervisor.jobCount*this.supervisor.processCount);
      for(var i=0;i<this.supervisor.jobCount*this.supervisor.processCount;i++){
         this.probabilityTable[i] = new Array(this.supervisor.machineCount);
         var total = 0;
         var max = 0;
         var maxMachine;
         for(var j=0;j<this.supervisor.machineCount;j++){
            var job = Math.floor(i/this.supervisor.processCount);
            var process = i%this.supervisor.processCount;
            var pheromone = this.supervisor.machinePheromoneTable[j][i];
            this.probabilityTable[i][j] =
Math.pow(pheromone,alpha)*Math.pow(1/processTable[job][process][j],beta);
            total+=this.probabilityTable[i][j];
            if(this.probabilityTable[i][j] > max){
                  max = this.probabilityTable[i][j];
                  maxMachine = j;}}
         var totalChance = 0;
         var chanceArray = new Array(this.supervisor.machineCount);
         for(var j=0;j<this.supervisor.machineCount;j++){
            this.probabilityTable[i][j]/=total;
            totalChance += this.probabilityTable[i][j];
            chanceArray[j]= totalChance;}
         var random = Math.random();
         for(var j=0;j<this.supervisor.machineCount;j++){
            if(chanceArray[j]>random){
                  maxMachine = j;
                  break;}}
         this.processMachineAssignList[i]= maxMachine;
         this.machineLengthList[maxMachine]++;}
      this.machineOrderAssignList = new Array(this.supervisor.machineCount);
      for(var n=0;n<this.supervisor.machineCount ;n++){
         this.machineOrderAssignList[n] = new Array();
         for(var k=0;k<this.machineLengthList[n];k++){
            this.probOrderTable = new
Array(this.supervisor.jobCount*this.supervisor.processCount);
            for(var i=0;i<this.supervisor.jobCount*this.supervisor.processCount;i++){
                  this.probOrderTable[i]=0;}
            for(var i=0;i<this.supervisor.jobCount*this.supervisor.processCount;i++){
                  var job = Math.floor(i/this.supervisor.processCount);
```

```
                  var process = i%this.supervisor.processCount;
                  if(this.processMachineAssignList[i]==n &&
this.machineOrderAssignList[n].indexOf(i)==-1){
                     var tempSetupTime = 0;
                     for(var j=0;j<this.machineOrderAssignList[n].length;j++){
                        var currentJob = this.machineOrderAssignList[n][j];
                        if(j>0){
                              var previousJob = this.machineOrderAssignList[n][j-1];
                              tempSetupTime+=setupTable[0][previousJob][currentJob];
                        }
                        else{
                              tempSetupTime+=setupTable[0][currentJob][currentJob];}
                     }
                     var previousJob = job;
                     if(this.machineOrderAssignList[n].length>0){
                        previousJob =
this.machineOrderAssignList[n][this.machineOrderAssignList[n].length-1];  }

   tempSetupTime+=Math.max(setupTable[0][previousJob][job],1);
   tempSetupTime/= this.machineOrderAssignList[n].length+1;
                     var pheromone =
this.supervisor.orderPheromoneTable[previousJob][job];
                     this.probOrderTable[i] =
Math.pow(pheromone,alpha)*Math.pow(1/tempSetupTime,beta);}}
            var total = 0;
            for(var i=0;i<this.supervisor.jobCount*this.supervisor.processCount;i++){
                  total += this.probOrderTable[i];}
            var max = 0;
            var maxJob = -1;
            var totalChance = 0;
            var chanceArray = new
Array(this.supervisor.jobCount*this.supervisor.processCount);
            for(var i=0;i<this.supervisor.jobCount*this.supervisor.processCount;i++){
                  if(total!=0){
                     this.probOrderTable[i] /=total;
                     totalChance+= this.probOrderTable[i];
                     chanceArray[i]=totalChance;
                  }
            }
            var random = Math.random();
            for(var i=0;i<this.supervisor.jobCount*this.supervisor.processCount;i++){
                  if(total!=0){
                     if(chanceArray[i] > random){
                        maxJob = i;
                        break;}}}
            this.machineOrderAssignList[n].push(maxJob);}}}
   this.localSearch = function(localIteration){
      for(var it=0;it<localIteration;it++){
         var tempAnt = jQuery.extend({}, this);
         var random1 =
Math.floor(Math.random()*tempAnt.supervisor.jobCount*tempAnt.supervisor.processCount);
         var random2 =
Math.floor(Math.random()*tempAnt.supervisor.jobCount*tempAnt.supervisor.processCount);
         var job1 = parseInt(random1/tempAnt.supervisor.processCount);
         var pro1 = parseInt(random1%tempAnt.supervisor.processCount);
         var job2 = parseInt(random2/tempAnt.supervisor.processCount);
         var pro2 = parseInt(random2%tempAnt.supervisor.processCount);
         if(job1==job2 && pro1==pro2){
            Continue;  }
         var node1,node2,order1,order2;
         for(var j=0;j<tempAnt.tabuK.length;j++){
            if(tempAnt.tabuK[j].jobID == job1 && tempAnt.tabuK[j].processID == pro1){
                  node1 = tempAnt.tabuK[j];
                  order1 = j;
            }
            if(tempAnt.tabuK[j].jobID == job2 && tempAnt.tabuK[j].processID == pro2){
                  node2 = tempAnt.tabuK[j];
                  order2 = j;}}
         var newNode1 = tempAnt.supervisor.myMap.jobList[job2][pro2][node1.machineID];
         var newNode2 = tempAnt.supervisor.myMap.jobList[job1][pro1][node2.machineID];
         tempAnt.tabuK[order1] = newNode1;
         tempAnt.tabuK[order2] = newNode2;
         if(order1<tempAnt.tabuK.length-1){
            for(var j=0;j<tempAnt.tabuK[order1].arcList.length;j++){
                  if(tempAnt.tabuK[order1].arcList[j].destinationNode ==
tempAnt.tabuK[order1+1]){
```

```
                    tempAnt.arcK[order1] =
tempAnt.tabuK[order1].arcList[j];}}}
         if(order2<tempAnt.tabuK.length-1){
            for(var j=0;j<tempAnt.tabuK[order2].arcList.length;j++){
                  if(tempAnt.tabuK[order2].arcList[j].destinationNode ==
tempAnt.tabuK[order2+1]){
                    tempAnt.arcK[order2] =
tempAnt.tabuK[order2].arcList[j];}}}
         if(order1>0){
            for(var j=0;j<tempAnt.tabuK[order1-1].arcList.length;j++){
                  if(tempAnt.tabuK[order1-1].arcList[j].destinationNode ==
tempAnt.tabuK[order1]){
                    tempAnt.arcK[order1-1] = tempAnt.tabuK[order1-1].arcList[j];}}}
         if(order2>0){
            for(var j=0;j<tempAnt.tabuK[order2-1].arcList.length;j++){
                  if(tempAnt.tabuK[order2-1].arcList[j].destinationNode ==
tempAnt.tabuK[order2]){
                    tempAnt.arcK[order2-1] = tempAnt.tabuK[order2-1].arcList[j];}}}
         tempAnt.Xk =
tempAnt.getMakespan(tempAnt.arcK,tempAnt.tabuK,tempAnt.supervisor.machineCount,tempAnt.s
upervisor.jobCount);
         if(tempAnt.Xk < this.Xk){
            this.tabuK = tempAnt.tabuK.slice();
            this.Xk = tempAnt.Xk;}}
      return this.Xk;}}
function Supervisor(){
   this.problem;        this.fuzzy = false;
   this.myMap;    this.antList;
   this.winnerAnt;   this.winnerList = new Array();
   this.tmpWinner;
   this.Xab = Infinity;    this.Xrb = Infinity;
   this.Xib = Infinity;    this.maxNC = 500;
   this.maxRepeatCnt = 10; this.ant = 10;
   this.NC = 0;         this.repeatCnt = 0;
   this.alpha = 2;   this.beta = 1;
   this.rho = 0.75;  this.tho0 = 1;
   this.C;     this.Q;
   this.jobCount; this.processCount;
   this.machineCount;   this.processTable;
   this.setupTable;  this.solutionTime;
   this.progress; this.polygonID;
   This.status;      this.timeLimit = 2000;
   this.timeBuffer = 0;
   this.initialSolution = new Array();
        this.machinePheromoneTable;
   this.orderPheromoneTable;
   this.initWithVariable =
function(antNumber,maxNC,maxRepeatCnt,rho,alpha,beta,tho0,C,Q,jobCount,processCount,mach
ineCount,processTable,setupTable){
      this.ant = antNumber;      this.maxNC = maxNC;
      this.maxRepeatCnt = maxRepeatCnt;      this.rho = rho;
      this.alpha = alpha;     this.beta = beta;
      this.tho0 = tho0;       this.C = C;
      this.Q = Q;          this.jobCount = parseInt(jobCount);
      this.processCount = parseInt(processCount);
      this.machineCount = parseInt(machineCount);
      this.processTable = processTable;
      this.setupTable = setupTable;
      this.Xab = Infinity;       this.Xrb = Infinity;
      this.Xib = Infinity;       this.constructMap();
      this.antList = new Array();
      this.machinePheromoneTable = new Array(this.machineCount);
      this.orderPheromoneTable = new Array(this.jobCount);
      for(var i=0;i<this.machineCount;i++){
         this.machinePheromoneTable[i]=new Array(this.jobCount*this.processCount);
         for(var j=0;j<this.jobCount*this.processCount;j++){
            this.machinePheromoneTable[i][j]=this.tho0;}}
      for(var i=0;i<this.jobCount;i++){
         this.orderPheromoneTable[i]=new Array(this.jobCount);
         for(var j=0;j<this.jobCount;j++){
            this.orderPheromoneTable[i][j]=this.tho0;}}}
   this.constructMap = function(){
      var jobList = new Array(this.jobCount);
      var nodeList = new Array();
      for(var i=0;i<this.jobCount;i++){
         var tempProcessList = new Array(this.processCount);
         jobList[i] = tempProcessList;
```

```
         for(var j=0;j<this.processCount;j++){
            var tempMachineList = new Array();
            for(var k=0;k<this.machineCount;k++){
                  var value = parseInt(this.processTable[i][j][k]);
                  if(value!=-1){
                     var tempNode = new Node();
                     tempNode.initWithVariable(i,j,k,value);
                     nodeList.push(tempNode);
                     tempMachineList.push(tempNode);}
                  tempProcessList[j] = tempMachineList;}}}
      this.myMap = new Map();
      this.myMap.initWithVariable(nodeList,jobList,this.tho0,this.C,this.setupTable);
   }
   this.pheromoneUpdateWithChromosome = function(chromosome){
      var tempNode = this.myMap.startNode;
      for(var i=0;i<chromosome.genotype.length;i++){
         var job = chromosome.genotype[i].jobID;
         var pro = chromosome.genotype[i].processID;
         var mac = chromosome.genotype[i].machineID;
         for(var j in tempNode.arcList){
            if( tempNode.arcList[j].destinationNode.jobID == job &&
                  tempNode.arcList[j].destinationNode.processID == pro &&
                  tempNode.arcList[j].destinationNode.machineID == mac){
                  tempNode.arcList[j].pheromoneLevel += this.Q/chromosome.makespan;
                  tempNode = tempNode.arcList[j].destinationNode;
                  break;}}}}
   this.solveIPPS = function(){
      var date1 = new Date().getTime();
      var date2 = new Date().getTime();
      var tempResult = new Array(this.ant);
      var XibOrder = 0;
      if(this.initialSolution.length>0){
   this.pheromoneUpdateWithChromosome(this.initialSolution[this.initialSolution.length-
1]);
      }
      for(this.NC=0; this.NC<this.maxNC ;this.NC++){
         this.antList = new Array();
         this.Xib = Infinity;
         for(var i=0;i<this.ant;i++){
            var tempAnt = new Ant();
            tempAnt.initWithVariable(this.myMap.startNode,this.myMap.nodeList,this,i);
            this.antList.push(tempAnt);
            tempResult[i] = this.antList[i].constructSchedule(0);
            if(tempResult[i]>0 && this.Xib > tempResult[i]){
                  this.Xib = tempResult[i];
                  XibOrder = i;}}
         for(var j=0;j<this.myMap.arcList.length;j++){
            this.myMap.arcList[j].pheromoneLevel *=(1-this.rho);}
         if(this.Xrb > this.Xib){
            this.Xrb = this.Xib;
            var winner = this.antList[XibOrder];
            var tmpSol = new Solution();
            tmpSol.initWithAnt(winner);
            this.tmpWinner = tmpSol;
            for(var j=0;j<winner.arcK.length;j++){
                  winner.arcK[j].pheromoneLevel += this.Q/this.Xrb;}
            if(this.Xab > this.Xrb){
                  this.Xab = this.Xrb;
                  this.winnerAnt = jQuery.extend({}, winner);
                  date2 = new
Date().getTime();              this.progress.addDot(this.polygonID,date2-
date1+this.timeBuffer,this.winnerAnt.Xk);}}
         else {
            this.repeatCnt ++;}
         console.log(this.NC+"- Xib:"+this.Xib+" Xrb:"+this.Xrb+" Xab:"+this.Xab+"
repeat:"+this.repeatCnt);
         var r = Math.random();
         var cp = Math.log(this.NC)/Math.log(this.maxNC);
         if(this.repeatCnt >= this.maxRepeatCnt && r>cp){
            console.log("pheromone reset");
            this.winnerList.push(this.tmpWinner);
            this.repeatCnt = 0;
            this.Xrb = Infinity;
            for(var j=0;j<this.myMap.arcList.length;j++){
                  this.myMap.arcList[j].pheromoneLevel = this.tho0;  }
```

```
            if(this.initialSolution.length>0){
this.pheromoneUpdateWithChromosome(this.initialSolution[Math.floor(Math.random()*this.in
itialSolution.length)]);}}
         date2 = new Date().getTime();
         var dotList = this.progress.polygonList[this.polygonID].dotList;
         if((date2-date1)-dotList[dotList.length-1].time>this.timeLimit){
            this.winnerList.push(this.tmpWinner);
            break;}}
      date2 = new Date().getTime();
      this.solutionTime = date2 -date1;
      return this.winnerAnt.tabuK;}}
```

EK-B

GENETİK ALGORİTMA

```
function Gene(){
   this.jobID;    this.processID;
   this.machineID;   this.processingTime;
   this.setupTime;
   this.initWithVariable = function(jobID,processID,machineID,processingTime){
      this.jobID = jobID;       this.processID = processID;
      this.machineID = machineID;      this.processingTime = processingTime;}}
function Chromosome(){
   this.ecosystem;   this.genotype;
   this.makespan; this.flowtime;       this.index;
   this.jobNumber;   this.processNumber;
   this.machineNumber;  this.machine;
   This.order;    this.initWithEcosystem = function(ecosystem){
      var job;       var machine;
      var process;       this.ecosystem = ecosystem;
      this.jobNumber = ecosystem.jobNumber;
      this.processNumber = ecosystem.processNumber;
      this.machineNumber = ecosystem.machineNumber;
      this.genotype = new Array();
      var jobIndex = new Array(ecosystem.jobNumber);
      this.index = new Array(ecosystem.jobNumber);
      this.machine = new Array(ecosystem.jobNumber*ecosystem.processNumber);
      this.order = new Array(ecosystem.jobNumber*ecosystem.processNumber);
      for(var i=0;i<ecosystem.jobNumber;i++){
         jobIndex[i] = 0;           this.index[i] = new Array();}
      for(var i=0;i<ecosystem.jobNumber*ecosystem.processNumber;i++){
         var toplam = 0;         var order = 0;
         for(var j=0;j<ecosystem.jobNumber;j++){
            toplam += jobIndex[j];}
         if(toplam == ecosystem.processNumber*ecosystem.jobNumber)
            break;
         do {
            job = Math.floor(Math.random() * ecosystem.jobNumber);            }
         while(jobIndex[job]>=ecosystem.processNumber);
         process = jobIndex[job];
         do {
            machine = Math.floor(Math.random() * ecosystem.machineNumber);
            order =
job*ecosystem.processNumber*ecosystem.machineNumber+process*ecosystem.machineNumber+mach
ine;}
         while(ecosystem.valueList[order]==-1);
         order =
job*ecosystem.processNumber*ecosystem.machineNumber+process*ecosystem.machineNumber+mach
ine;
         var tempGene = new Gene();
         tempGene.initWithVariable(job,process,machine,ecosystem.valueList[order]);
         this.order[i]=job;
         this.machine[job*ecosystem.processNumber + process] = machine;
         this.genotype.push(tempGene);          jobIndex[job]++;}}
   this.initWithVariable = function(machine,order,ecosystem){
      this.ecosystem = ecosystem;      this.jobNumber = ecosystem.jobNumber;
      this.processNumber = ecosystem.processNumber;
```

```
      this.machineNumber = ecosystem.machineNumber;
      this.genotype = new Array();
      var jobIndex = new Array(ecosystem.jobNumber);
      this.machine = new Array(ecosystem.jobNumber*ecosystem.processNumber);
      this.order = new Array(ecosystem.jobNumber*ecosystem.processNumber);
      for(var i=0;i<ecosystem.jobNumber;i++){
         jobIndex[i] = 0;}
      for(var i=0;i<ecosystem.jobNumber*ecosystem.processNumber;i++){
         var machineOrder = order[i]*ecosystem.processNumber + jobIndex[order[i]];
         var timeOrder = order[i]*ecosystem.processNumber*ecosystem.machineNumber +
jobIndex[order[i]]*ecosystem.machineNumber + machine[machineOrder];
         var tempGene = new
Gene();       tempGene.initWithVariable(order[i],jobIndex[order[i]],machine[machineOrder
],ecosystem.valueList[timeOrder]);
         this.genotype.push(tempGene);          jobIndex[order[i]]++;
         this.order[i]=order[i];          this.machine[i]=machine[i];}
      if(this.check==false){
         alert("error in ga.js:initwithvariable");}}
   this.check = function(){
      var processOrderList = new Array(this.jobNumber);
      for(var i=0;i<this.jobNumber;i++){
         processOrderList[i] = 0;}
      for(var i=0;i<this.order.length;i++){
         var job = this.order[i];         var process = processOrderList[job];
         processOrderList[job]++;
         var machine = this.machine[job*this.processNumber+process];
         if(job!=this.genotype[i].jobID || process!=this.genotype[i].processID ||
machine!=this.genotype[i].machineID){   return false;   }    }      return true;}
   this.balance = function(){       this.genotype = new Array();
      var processOrderList = new Array(this.jobNumber);
      for(var i=0;i<this.jobNumber;i++){
         processOrderList[i] = 0;}
      for(var i=0;i<this.order.length;i++){
         var job = this.order[i];         var process = processOrderList[job];
         processOrderList[job]++;
         var machine = this.machine[job*this.processNumber+process];
         var processingTime =
this.ecosystem.valueList[job*this.processNumber*this.machineNumber+process*this.machineN
umber+machine];
         var tempGene = new Gene();
         tempGene.initWithVariable(job,process,machine,processingTime);
         tempGene.processingTime = processingTime;
         this.genotype.push(tempGene);}}
   this.initWithSolution = function(solution){
      if(solution.problemType==1){                var valueList = new
Array(solution.jobNumber*solution.processNumber*solution.machineNumber);
         for(var i=0;i<solution.jobNumber;i++){
            for(var j=0;j<solution.processNumber;j++){
                  for(var k=0;k<solution.machineNumber;k++){
                     valueList[i*solution.processNumber*solution.machineNumber +
j*solution.machineNumber + k] = solution.processTimeList[i][j][k];}}}
         this.ecosystem = new
Ecosystem();        this.ecosystem.initWithVariable(solution.jobNumber,solution.processN
umber,solution.machineNumber,100,10,0.1,0.9,valueList,solution.setupTimeList,40,1,4,1,tr
ue);
         this.jobNumber = solution.jobNumber;
         this.processNumber = solution.processNumber;
         this.machineNumber = solution.machineNumber;
         this.makespan = solution.makespan;
         this.genotype = new Array();
            this.order = new Array(solution.jobNumber*solution.processNumber);
         for(var i=0;i<solution.jobOrder.length;i++){
            var tempGene = new Gene();
            tempGene.initWithVariable(solution.jobOrder[i],-1,-1,-1);
            this.genotype.push(tempGene);
            this.order[i] = solution.jobOrder[i];}}     else{
            var valueList = new
Array(solution.jobNumber*solution.processNumber*solution.machineNumber);
         for(var i=0;i<solution.jobNumber;i++){
            for(var j=0;j<solution.processNumber;j++){
                  for(var k=0;k<solution.machineNumber;k++){
                     valueList[i*solution.processNumber*solution.machineNumber +
j*solution.machineNumber + k] = solution.processTimeList[i][j][k];}}}
         this.ecosystem = new
Ecosystem();       this.ecosystem.initWithVariable(solution.jobNumber,solution.processNu
mber,solution.machineNumber,100,10,0.1,0.9,valueList,solution.setupTimeList,40,1,4,1,tru
e);
```

```
this.jobNumber = solution.jobNumber;      this.processNumber = solution.processNumber;
this.machineNumber = solution.machineNumber;       this.makespan = solution.makespan;
this.genotype = new Array();              this.machine = new
Array(solution.jobNumber*solution.processNumber);
         this.order = new Array(solution.jobNumber*solution.processNumber);
         for(var i=0;i<solution.processList.length;i++){
            var tempGene = new
Gene();            tempGene.initWithVariable(solution.processList[i].jobID,solution.proc
essList[i].processID,solution.processList[i].machineID,solution.processList[i].processin
gTime);
this.genotype.push(tempGene);             this.order[i] = solution.processList[i].jobID;
            this.machine[solution.processList[i].jobID*solution.processNumber +
solution.processList[i].processID] = solution.processList[i].machineID;    }  }}
   this.getMakespan = function(){      var makespan = 0;
      var machineIndex = new Array(machineNumber);
      var jobIndex = new Array(jobNumber);
      for(var i=0;i<machineNumber;i++){
         machineIndex[i]=0;   }
      for(var i=0;i<jobNumber;i++){
         jobIndex[i]=0;}
      for(var i=0;i<this.genotype.length;i++){
         var tempGene = this.genotype[i];          var startTime =
Math.max(parseInt(jobIndex[tempGene.jobID]),parseInt(machineIndex[tempGene.machineID]));
         var job = tempGene.jobID;           var previousJob = tempGene.jobID;
         var setupTime;          if(i>0){
            previousJob = this.genotype[i-1].jobID;  }
         setupTime =
this.ecosystem.setupTimeTable[tempGene.machineID][previousJob][job];
         tempGene.setupTime = setupTime;
         jobIndex[tempGene.jobID] = startTime + setupTime + tempGene.processingTime;
         machineIndex[tempGene.machineID] = startTime + setupTime +
tempGene.processingTime; }
      for(var i=0;i<machineNumber;i++){
         if(machineIndex[i]>makespan)
            makespan=machineIndex[i];}
      this.makespan = makespan;     return makespan;}
   this.cross = function(match){       var children = new Array();
      var G = new Genetics();       var descendant = null;
      var orderedDesc = null;
      switch (this.ecosystem.crossoverType){
         case 1:
            descendant =
G.singlePointCrossover(this.machine,match.machine,this.ecosystem);
            break;         case 2:
            descendant =
G.twoPointsCrossover(this.machine,match.machine,this.ecosystem);
            break;         case 3:
            descendant =
G.multiPointsCrossover(this.machine,match.machine,this.ecosystem);
            break;         case 4:
            descendant = G.uniformCrossover(this.machine,match.machine,this.ecosystem);
            break;         default :
            descendant =
G.multiPointsCrossover(this.machine,match.machine,this.ecosystem);
            break;      }
      switch (this.ecosystem.orderedCrossType){
         case 1:              orderedDesc =
G.positionBasedCrossover(this.order,match.order,this.ecosystem);
            break;         default :
            orderedDesc =
G.positionBasedCrossover(this.order,match.order,this.ecosystem);
            break;}
      var chromosome1 = new Chromosome();       var chromosome2 = new Chromosome();
      chromosome1.initWithVariable(descendant[0],orderedDesc[0],this.ecosystem);
      chromosome2.initWithVariable(descendant[1],orderedDesc[1],this.ecosystem);
      children.push(chromosome1);      children.push(chromosome2);
      if(chromosome1.check()==false){
         alert("error in ga.js:Genetics.cross("+this.ecosystem.crossoverType+")");}
      if(chromosome2.check()==false){
         alert("error in ga.js:Genetics.cross("+this.ecosystem.crossoverType+")");}
      return children;  }
   this.mutate = function(){
      var G = new Genetics();
      switch (this.ecosystem.mutationType){
         case 1:
               this.machine = G.mutationReverse(this.machine,this.ecosystem);
            this.order = G.mutationReverse(this.order,this.ecosystem);
```

```
            this.machine = G.repair(this.machine,this.ecosystem);
            break;         case 2:
            this.machine = G.mutationNeighbor(this.machine,this.ecosystem);
            this.order = G.mutationNeighbor(this.order,this.ecosystem);
            this.machine = G.repair(this.machine,this.ecosystem);
            break;         case 3:
            this.machine = G.mutationArbitrary(this.machine,this.ecosystem);
            this.order = G.mutationArbitrary(this.order,this.ecosystem);
            this.machine = G.repair(this.machine,this.ecosystem);
            break;         case 4:
            this.order = G.mutation3Arbitrary(this.order,this.ecosystem);
            break;         case 5:
            this.machine = G.mutationFloat(this.machine,this.ecosystem);
            this.order = G.mutationFloat(this.order,this.ecosystem);
            this.machine = G.repair(this.machine,this.ecosystem);
            break;         default:
            this.machine = G.mutationReverse(this.machine,this.ecosystem);
            this.order = G.mutationReverse(this.order,this.ecosystem);
            this.machine = G.repair(this.machine,this.ecosystem);
            break;      }
      this.balance();
      if(this.check()==false){
       alert("error in ga.js:Genetics.mutate("+this.ecosystem.mutationType+")");}}}
function Genetics(){
   this.singlePointCrossover = function(gene1,gene2,ecosystem){
      var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
      var descendant = new Array(2);
      for(var i=0;i<2;i++){
         descendant[i] = new Array(totalLength);}
      var start = Math.floor(Math.random() * totalLength);
      var length = totalLength-start-1;
      for(var i=0;i<start;i++){
         descendant[0][i]=gene1[i];          descendant[1][i]=gene2[i];       }
      for(var i=start;i<start+length;i++){
         descendant[0][i]=gene2[i];          descendant[1][i]=gene1[i];       }
      for(var i=start+length;i<totalLength;i++){
         descendant[0][i]=gene1[i];          descendant[1][i]=gene2[i];       }
      return descendant;   }
   this.twoPointsCrossover = function(gene1,gene2,ecosystem){
      var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
      var descendant = new Array(2);
      for(var i=0;i<2;i++){
         descendant[i] = new Array(totalLength);   }
      var start = Math.floor(Math.random() * totalLength);
      var length = Math.floor(Math.random() * (totalLength-start));
      for(var i=0;i<start;i++){
         descendant[0][i]=gene1[i];          descendant[1][i]=gene2[i];       }
      for(var i=start;i<start+length;i++){
         descendant[0][i]=gene2[i];          descendant[1][i]=gene1[i];       }
      for(var i=start+length;i<totalLength;i++){
         descendant[0][i]=gene1[i];          descendant[1][i]=gene2[i];       }
      return descendant;   }
   this.multiPointsCrossover = function(gene1,gene2,ecosystem){
      var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
      var descendant = new Array(2);
      for(var i=0;i<2;i++){
         descendant[i] = new Array(totalLength);      }
      var tempGene1 = gene1.slice(0);     var tempGene2 = gene2.slice(0);     do {
         var start = Math.floor(Math.random() * totalLength);
         var length = Math.floor(Math.random() * (totalLength-start));
         for(var i=0;i<start;i++){
            descendant[0][i]=tempGene1[i];            descendant[1][i]=tempGene2[i];}
         for(var i=start;i<start+length;i++){
            descendant[0][i]=tempGene2[i];            descendant[1][i]=tempGene1[i];
}
         for(var i=start+length;i<totalLength;i++){
            descendant[0][i]=tempGene1[i];            descendant[1][i]=tempGene2[i];}
         var tempGene1 = gene1.slice(0);          var tempGene2 = gene2.slice(0);}
      while(Math.random()<0.5);     return descendant;   }
   this.uniformCrossover = function(gene1,gene2,ecosystem){
      var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
      var descendant = new Array(2);
      descendant[0] = gene1.slice(0);     descendant[1] = gene2.slice(0);
      for(var i=0;i<totalLength;i++){
         if(Math.random()>0.5){
            descendant[0][i]=gene2[i];          descendant[1][i]=gene1[i]; }}
      return descendant;   }
```

```
   this.positionBasedCrossover = function(gene1,gene2,ecosystem){
      var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
      var descendant = new Array(2);
      for(var i=0;i<2;i++){
         descendant[i] = new Array(totalLength);      }
      var oneZeroList = new Array(totalLength);
      for(var i=0;i<totalLength;i++){
         oneZeroList[i]= Math.floor(Math.random()+0.5);
         if(oneZeroList[i]==0){
            descendant[0][i]=gene1[i];             descendant[1][i]=gene2[i];}
         else{       descendant[0][i]=-1;             descendant[1][i]=-1;}}
      for(var i=0;i<totalLength;i++){
         if(oneZeroList[i]==1){
            for(var j=0;j<totalLength;j++){
                  var flag = 0;
                  for(var k=0;k<totalLength && flag!=ecosystem.processNumber;k++){
                     if(gene2[j]==descendant[0][k]){
                        flag++;                         }                }
                  if(flag<ecosystem.processNumber){
                     descendant[0][i]=gene2[j];
                     break;                }              }           }     }
      for(var i=0;i<totalLength;i++){
         if(oneZeroList[i]==1){
            for(var j=0;j<totalLength;j++){
                  var flag = 0;
                  for(var k=0;k<totalLength && flag!=ecosystem.processNumber;k++){
                     if(gene1[j]==descendant[1][k]){
                        flag++;                         }              }
                  if(flag<ecosystem.processNumber){
                     descendant[1][i]=gene1[j];
                     break;                }              }           }     }
      return descendant;   }
   this.partialMappedCrossover = function(gene1,gene2,ecosystem){   //todo ecosystem-
independent yaz. gene1.length kullan
      var descendant = new Array(2);
      descendant[0] = gene1.slice(0);
   descendant[1] = gene2.slice(0);
      var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
      var start = Math.floor(Math.random() * totalLength);
      var length = Math.floor(Math.random() * (totalLength-start));
      for(var i=start;i<start+length;i++){
         descendant[0][i]=gene2[i];          descendant[1][i]=gene1[i];       }
      var map = new Array();
      for(var i=start;i<start+length;i++){
         var flag = 0;           for(var j=start;j<start+length;j++){


            if(descendant[1][j]==descendant[0][i]){
                  flag++;
                  break;               }           }
         if(flag==0){   //map olustur
            var tmp = new Array(2);
            tmp[0]=descendant[0][i];
            tmp[1]=descendant[1][i];
            for(var j=start;j<start+length;j++){
                  if(tmp[1]==descendant[0][j]){
                     tmp[1]=descendant[1][j];
                        j=start-1;                 }              }
            map.push(tmp);          }     }
      for(var i=0;i<start;i++){
         for(var j=0;j<map.length;j++){
            if(descendant[0][i]==map[j][0]){
                  descendant[0][i]=map[j][1];               }
            if(descendant[1][i]==map[j][1]){
                  descendant[1][i]=map[j][0];               }           }     }
      for(var i=start+length;i<totalLength;i++){
         for(var j=0;j<map.length;j++){
            if(descendant[0][i]==map[j][0]){
                  descendant[0][i]=map[j][1];               }
            if(descendant[1][i]==map[j][1]){
                  descendant[1][i]=map[j][0];               }           }     }
      return descendant;   }
   this.linearOrderedCrossover = function(gene1,gene2,ecosystem){
      var descendant = new Array(2);
      descendant[0] = gene1.slice(0);
   descendant[1] = gene2.slice(0);
      return descendant;   }
```

```
this.orderBasedCrossover = function(gene1,gene2,ecosystem){
   var descendant = new Array(2);
   descendant[0] = gene1.slice(0);
   descendant[1] = gene2.slice(0);
   return descendant;   }
this.orderedCrossover = function(gene1,gene2,ecosystem){
   var descendant = new Array(2);
   descendant[0] = gene1.slice(0);
   descendant[1] = gene2.slice(0);
   return descendant;   }
this.circleCrossover = function(gene1,gene2,ecosystem){
   var descendant = new Array(2);
   descendant[0] = gene1.slice(0);
   descendant[1] = gene2.slice(0);
   return descendant;   }
this.mutationReverse = function(chromosome,ecosystem){
   var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
   var result = chromosome.slice(0);
   var start = Math.floor(Math.random() * totalLength);
   var length = Math.floor(Math.random() * (totalLength-start));
   for(var i=0;i<length/2;i++){
      result[start+i]=chromosome[start+length-i-1];
      result[start+length-i-1]=chromosome[start+i];     }
   return result; }
this.mutationNeighbor = function(chromosome,ecosystem){
   var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
   var result = chromosome.slice(0);
   var start = Math.floor(Math.random() * (totalLength-1));
   result[start]=chromosome[start+1];
   result[start+1]=chromosome[start];
   return result; }
this.mutationArbitrary = function(chromosome,ecosystem){
   var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
   var result = chromosome.slice(0);
   var start = Math.floor(Math.random() * totalLength);
   var finish =Math.floor(Math.random() * totalLength);
   result[start] = chromosome[finish];
   result[finish] = chromosome[start];
   return result; }
this.mutation3Arbitrary = function(chromosome,ecosystem){
   var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
   var result = chromosome.slice(0);
   var first = Math.floor(Math.random() * totalLength);
   var second =Math.floor(Math.random() * totalLength);
   var third = Math.floor(Math.random() * totalLength);
if(first==second || first==third || second==third){
      return result;       }
   result[first] = chromosome[second];
   result[second] = chromosome[third];
   result[third] = chromosome[first];
   return result; }
this.mutationFloat = function(chromosome,ecosystem){
   var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
   var result = chromosome.slice(0);
   var start = Math.floor(Math.random() * totalLength);
   var finish =Math.floor(Math.random() * totalLength);
   if(finish>start){
      for(var i=start;i<finish;i++){
         result[i]=chromosome[i+1];          }
      result[finish]=chromosome[start];      }
   else if(start>finish){
      for(var i=finish;i<start;i++){
         result[i]=chromosome[i+1];          }
      result[start]=chromosome[finish];      }
   else{
      return result;       }
   return result; }
this.repair = function(chromosome,ecosystem){
   var totalLength = ecosystem.jobNumber*ecosystem.processNumber;
   for(var i=0;i<totalLength;i++){
      if(ecosystem.valueList[i*ecosystem.machineNumber+chromosome[i]]==-1){
         var machine;
         do{
               machine = Math.floor(Math.random() * ecosystem.machineNumber);
               chromosome[i]=machine;              }
         while(ecosystem.valueList[i*ecosystem.machineNumber+machine]==-1);  }}
   return chromosome;   }
```

```
   this.repeatControl = function(gene){
      for(var i=0;i<gene.length;i++)
         for(var j=i+1;j<gene.length;j++)
            if(gene[i]==gene[j])
                  return false;
      return true;   }}
function Ecosystem(){
   this.problem;        //problemin butun ozelliklerinin toplandigi object
   this.fuzzy = false;  // problem fuzzy mi?
   this.jobNumber;
   this.machineNumber;
   this.processNumber;
   this.valueList;      this.processTimeTable;
   this.setupTimeTable;
   this.problemType = 0;   //0 jobshop, 1 flowshop
   this.noWait = 0;     //0 wait, 1 no-wait
   this.generationNumber;
   this.populationNumber;
   this.currentGeneration;
   this.maxRepeatNumber;
   this.curRepeatNumber;
   this.Xrb;
   this.population;
   this.mutationRate;
   this.crossRate;
   this.mutationType;
   this.crossoverType;
   this.orderedCrossType;
   this.rouletteWheel;
   this.makespan;
   this.solutionList;
   this.solutionTime = 0;
   this.progress;
   this.polygonID;
   this.champion;
this.timeLimit = 1000;     // dongu time limit'i gecerse durdurayim.
   this.initialPopulation = new Array();     this.timeBuffer = 0;
   this.winnerList = new Array();      this.tmpWinner;
   this.initWithVariable =
function(jobNumber,processNumber,machineNumber,generationNumber,maxRepeatNumber,mutation
Rate,crossRate,valueList,setupTimeTable,populationNumber,mutationType,crossoverType,orde
redCrossoverType,rouletteWheel){
      this.jobNumber = parseInt(jobNumber);
      this.processNumber = parseInt(processNumber);
      this.machineNumber = parseInt(machineNumber);
      this.generationNumber = parseInt(generationNumber);
      this.populationNumber = parseInt(populationNumber);
      this.maxRepeatNumber = parseInt(maxRepeatNumber);
      this.mutationRate = parseFloat(mutationRate);
      this.crossRate = parseFloat(crossRate);
      this.mutationType = mutationType;
      this.crossoverType = crossoverType;
      this.orderedCrossType = orderedCrossoverType;
      this.rouletteWheel = rouletteWheel;
      this.valueList = valueList;
      this.setupTimeTable = setupTimeTable;  }
   this.sortPopulationWithMakespan = function(){
   for(var i=0;i<this.population.length;i++){
         this.population[i].makespan = this.population[i].getMakespan();     }
      this.population.sort(function(a,b){
         if(a.makespan > b.makespan) return 1;
         if(a.makespan < b.makespan) return -1;
         return 0;      });   }
   this.naturalSelectionTournament = function(){
      this.population.splice(this.population.length/2,this.population.length/2);   }
   this.naturalSelectionRoulette = function(){
      var importanceList = new Array(this.population.length);
      var total = 0;
      for(var i =0;i<this.population.length;i++){
         importanceList[i]=1/this.population[i].getMakespan();
         total+=importanceList[i];     }
      var random = 0;
      var i;
      for(i=0;random<total/2;i+=2){
         random+= importanceList[i]+importanceList[i+1];      }
      this.population.splice(i,this.population.length-i);   }
   this.crossPopulation = function(){
```

```
      var oldCount = this.population.length;
      for(var i=0;i<(this.populationNumber-oldCount)/2;i++){
         var mom = Math.floor(Math.random()*oldCount);
         var dad = Math.floor(Math.random()*oldCount);
         var children = this.population[mom].cross(this.population[dad]);
         this.population.push(children[0]);
         this.population.push(children[1]);     }}
   this.mutatePopulation = function(){
      for(var i=0;i<this.population.length;i++){
         var coin = Math.random();
         if(coin<this.mutationRate){
            this.population[i].mutate();           }     }  }
   this.solve = function(){
      if(this.problemType==0)
         this.solveFlowShop();
      else
         this.solveIPPS(); }
   this.solveIPPS = function(){
      var date1 = new Date().getTime();
      var date2 = new Date().getTime();
      this.makespan = Infinity;
      this.curRepeatNumber = 0;
      this.population = new Array();
      for(var i=0;i<this.populationNumber;i++){
         var tempChromosome = new Chromosome();
         if(i >= this.initialPopulation.length){
            tempChromosome.initWithEcosystem(this);        } else {
            tempChromosome.initWithSolution(this.initialPopulation[i]);
            console.log(tempChromosome);          }
         this.population.push(tempChromosome);    }
      this.champion = jQuery.extend({}, this.population[0]);  // object copy
      this.champion.getMakespan();
      for(this.currentGeneration = 0;this.currentGeneration<
this.generationNumber;this.currentGeneration++){
         this.sortPopulationWithMakespan();
         this.tmpWinner = jQuery.extend({}, this.population[0]);
            if(this.rouletteWheel){
            this.naturalSelectionRoulette();         }
         else{
            this.naturalSelectionTournament();          }
         this.crossPopulation();

   this.mutatePopulation();      console.log(this.currentGeneration,this.population[0].g
etMakespan(),this.champion.makespan);
         if(this.population[0].getMakespan()< this.champion.makespan){
            this.champion = jQuery.extend({}, this.population[0]);
            date2 = new
Date().getTime();            this.progress.addDot(this.polygonID,date2-
date1+this.timeBuffer,this.champion.makespan);
         }           else{
            this.curRepeatNumber++;          }
         if(this.curRepeatNumber > this.maxRepeatNumber){
            this.curRepeatNumber = 0;
            this.winnerList.push(this.tmpWinner);
            this.population = new Array();
            for(var i=0;i<this.populationNumber;i++){
               var tempChromosome = new Chromosome();
                  tempChromosome.initWithEcosystem(this);
                  this.population.push(tempChromosome);             }     }
         date2 = new Date().getTime();
                  var dotList = this.progress.polygonList[this.polygonID].dotList;
         if((date2-date1)-dotList[dotList.length-1].time>this.timeLimit){
            this.winnerList.push(this.tmpWinner);
            break;         }     }
      console.log(this.champion.genotype, this.champion.makespan);
      this.solutionList = this.champion.genotype;
      this.makespan = this.champion.makespan;
      date2 = new Date().getTime();
      this.solutionTime = date2 -date1;
      return this.solutionList;  }
   this.solveFlowShop = function(){
      var date1 = new Date().getTime();
      var date2 = new Date().getTime();
      this.makespan = Infinity;
      this.curRepeatNumber = 0;
      this.population = new Array();
      for(var i=0;i<this.populationNumber;i++){
```

```
var tempChromosome = new Chromosome();
tempChromosome.initWithEcosystem(this);
this.population.push(tempChromosome);     }  }}
```

EK-C

YAPAY BAĞIŞIKLIK SİSTEMİ

```
function Gene(){
    this.jobID;
    this.processID;
    this.machineID;
    this.processingTime;
    this.setupTime;
    this.initWithVariable = function(jobID,processID,machineID,processingTime){

        this.jobID = jobID;
        this.processID = processID;
        this.machineID = machineID;
        this.processingTime = processingTime;
    }}
function Antibody(){
    this.ecosystem;
    this.genotype;
    this.makespan = 0;
    this.flowtime;      //flowtime, flowshop cozumler icin gecerli
    this.index;
    this.jobNumber;             //problemdeki is sayisi
    this.processNumber;         //problemdeki proses sayisi
    this.machineNumber;         //problemdeki makine sayisi
    this.machine;
    this.order;
this.initWithProblem = function(problem){
        var job;                // random urettigim o anki siradaki is
        var machine;            // random urettigim siradaki is-prosesin makinesi
        var process;            // isin siradaki islenmemis prosesi
        this.ecosystem = problem;
        this.problem = problem;
        this.jobNumber = problem.jobNumber;
        this.processNumber = problem.processNumber;
        this.machineNumber = problem.machineNumber;
        this.genotype = new Array();            // olusturdugumuz genlerin dizisi
        var jobIndex = new Array(problem.jobNumber);
        this.index = new Array(problem.jobNumber);
        this.machine = new Array(problem.jobNumber*problem.processNumber);
        this.order = new Array(problem.jobNumber*problem.processNumber);
        for(var i=0;i<problem.jobNumber;i++){
            jobIndex[i] = 0;
            this.index[i] = new Array();
        }
        for(var i=0;i<problem.jobNumber*problem.processNumber;i++){
            var toplam = 0;
            var order = 0;
            for(var j=0;j<problem.jobNumber;j++){
                toplam += jobIndex[j];
            }
            if(toplam == problem.processNumber*problem.jobNumber)       break;
do {
                job = Math.floor(Math.random() * problem.jobNumber);
            }
```

```
            while(jobIndex[job]>=problem.processNumber);
            process = jobIndex[job];
do {
                machine = Math.floor(Math.random() * problem.machineNumber);
                order =
job*problem.processNumber*problem.machineNumber+process*problem.machineNumber+machine;
            }
            while(problem.valueList[order]==-1);
            order =
job*problem.processNumber*problem.machineNumber+process*problem.machineNumber+machine;
            var tempGene = new Gene();
            tempGene.initWithVariable(job,process,machine,problem.valueList[order]);
            this.order[i]=job;
            this.machine[job*problem.processNumber + process] = machine;
            this.genotype.push(tempGene);
            jobIndex[job]++;
        }
    }
    this.initWithVariable = function(machine,order,ecosystem){
          this.ecosystem = ecosystem;
        this.jobNumber = ecosystem.jobNumber;
        this.processNumber = ecosystem.processNumber;
        this.machineNumber = ecosystem.machineNumber;
        this.genotype = new Array();
        var jobIndex = new Array(ecosystem.jobNumber);
        this.machine = new Array(ecosystem.jobNumber*ecosystem.processNumber);
        this.order = new Array(ecosystem.jobNumber*ecosystem.processNumber);
        for(var i=0;i<ecosystem.jobNumber;i++){
            jobIndex[i] = 0;
        }
          for(var i=0;i<ecosystem.jobNumber*ecosystem.processNumber;i++){
            var machineOrder = order[i]*ecosystem.processNumber + jobIndex[order[i]];
            var timeOrder = order[i]*ecosystem.processNumber*ecosystem.machineNumber +
jobIndex[order[i]]*ecosystem.machineNumber + machine[machineOrder];
            var tempGene = new Gene();
tempGene.initWithVariable(order[i],jobIndex[order[i]],machine[machineOrder],ecosystem.va
lueList[timeOrder]);
            this.genotype.push(tempGene);
            jobIndex[order[i]]++;
            this.order[i]=order[i];
            this.machine[i]=machine[i];
        }
        if(this.check==false){
            console.log(this);
            alert("error in ga.js:initwithvariable");
        }}
    this.check = function(){
        var processOrderList = new Array(this.jobNumber);
        for(var i=0;i<this.jobNumber;i++){
            processOrderList[i] = 0;        }
        for(var i=0;i<this.order.length;i++){
            var job = this.order[i];
            var process = processOrderList[job];
            processOrderList[job]++;
            var machine = this.machine[job*this.processNumber+process];
            if(job!=this.genotype[i].jobID || process!=this.genotype[i].processID ||
machine!=this.genotype[i].machineID){
                return false;            }      }
        return true;  }
    this.balance = function(){
        this.genotype = new Array();
        var processOrderList = new Array(this.jobNumber);
        for(var i=0;i<this.jobNumber;i++){
            processOrderList[i] = 0;       }
        for(var i=0;i<this.order.length;i++){
            var job = this.order[i];
            var process = processOrderList[job];
            processOrderList[job]++;
            var machine = this.machine[job*this.processNumber+process];
            var processingTime =
this.ecosystem.valueList[job*this.processNumber*this.machineNumber+process*this.machineN
umber+machine];
            var tempGene = new Gene();
            tempGene.initWithVariable(job,process,machine,processingTime);
            tempGene.processingTime = processingTime;
            this.genotype.push(tempGene);        }    }
    this.initWithSolution = function(solution){
```

```
if(solution.problemType==1){    //flowshop
var valueList = new
Array(solution.jobNumber*solution.processNumber*solution.machineNumber);
            for(var i=0;i<solution.jobNumber;i++){
                for(var j=0;j<solution.processNumber;j++){
                    for(var k=0;k<solution.machineNumber;k++){
                        valueList[i*solution.processNumber*solution.machineNumber +
j*solution.machineNumber + k] = solution.processTimeList[i][j][k];}}  }
            this.ecosystem = new Ecosystem();
this.ecosystem.initWithVariable(solution.jobNumber,solution.processNumber,solution.machi
neNumber,100,10,0.1,0.9,valueList,solution.setupTimeList,40,1,4,1,true);
            this.jobNumber = solution.jobNumber;
            this.processNumber = solution.processNumber;
            this.machineNumber = solution.machineNumber;
            this.makespan = solution.makespan;

            this.genotype = new Array();
         this.order = new Array(solution.jobNumber*solution.processNumber);
            for(var i=0;i<solution.jobOrder.length;i++){
                var tempGene = new Gene();
                tempGene.initWithVariable(solution.jobOrder[i],-1,-1,-1);
                this.genotype.push(tempGene);
                this.order[i] = solution.jobOrder[i];}}
        else{                           //jobshop problemType=0
            var valueList = new
Array(solution.jobNumber*solution.processNumber*solution.machineNumber);
            for(var i=0;i<solution.jobNumber;i++){
                for(var j=0;j<solution.processNumber;j++){
                    for(var k=0;k<solution.machineNumber;k++){
                        valueList[i*solution.processNumber*solution.machineNumber +
j*solution.machineNumber + k] = solution.processTimeList[i][j][k];}}}
            this.ecosystem = new Ecosystem();
this.ecosystem.initWithVariable(solution.jobNumber,solution.processNumber,solution.machi
neNumber,100,10,0.1,0.9,valueList,solution.setupTimeList,40,1,4,1,true);
            this.jobNumber = solution.jobNumber;
            this.processNumber = solution.processNumber;
            this.machineNumber = solution.machineNumber;
            this.makespan = solution.makespan;
            this.genotype = new Array();
            this.machine = new Array(solution.jobNumber*solution.processNumber);
            this.order = new Array(solution.jobNumber*solution.processNumber);
            for(var i=0;i<solution.processList.length;i++){
                var tempGene = new Gene();
tempGene.initWithVariable(solution.processList[i].jobID,solution.processList[i].processI
D,solution.processList[i].machineID,solution.processList[i].processingTime);
                this.genotype.push(tempGene);

                this.order[i] = solution.processList[i].jobID;
                this.machine[solution.processList[i].jobID*solution.processNumber +
solution.processList[i].processID] = solution.processList[i].machineID;}}}
    this.getMakespan = function(){
        if(this.makespan!=0)
            return this.makespan;
        var makespan = 0;
        var machineIndex = new Array(machineNumber);
        var jobIndex = new Array(jobNumber);
        for(var i=0;i<machineNumber;i++){
            machineIndex[i]=0;}
        for(var i=0;i<jobNumber;i++){
            jobIndex[i]=0;}
        for(var i=0;i<this.genotype.length;i++){
            var tempGene = this.genotype[i];
            var startTime =
Math.max(parseInt(jobIndex[tempGene.jobID]),parseInt(machineIndex[tempGene.machineID]));
            var job = tempGene.jobID;
            var previousJob = tempGene.jobID;
            var setupTime;
            if(i>0){
                previousJob = this.genotype[i-1].jobID;  // }
         setupTime = this.problem.setupTimeList[tempGene.machineID][previousJob][job];
            tempGene.setupTime = setupTime;
            jobIndex[tempGene.jobID] = startTime + setupTime + tempGene.processingTime;
            machineIndex[tempGene.machineID] = startTime + setupTime +
tempGene.processingTime;}
        for(var i=0;i<machineNumber;i++){
            if(machineIndex[i]>makespan)
                makespan=machineIndex[i];}
```

```
        this.makespan = makespan;
        return makespan;}
    this.cross = function(match){
        var children = new Array();
        var G = new Genetics();
        var descendant = null;
        var orderedDesc = null;
        switch (this.ecosystem.crossoverType){
            case 1:
          descendant = G.singlePointCrossover(this.machine,match.machine,this.ecosystem);
                break;
            case 2:
                descendant =
G.twoPointsCrossover(this.machine,match.machine,this.ecosystem);
                break;
            case 3:
                descendant =
G.multiPointsCrossover(this.machine,match.machine,this.ecosystem);
                break;
            case 4:
                descendant =
G.uniformCrossover(this.machine,match.machine,this.ecosystem);
                break;
            default :
                descendant =
G.multiPointsCrossover(this.machine,match.machine,this.ecosystem);
                break;}
        switch (this.ecosystem.orderedCrossType){
            case 1:
          orderedDesc = G.positionBasedCrossover(this.order,match.order,this.ecosystem);
                break;
            default :
                orderedDesc =
G.positionBasedCrossover(this.order,match.order,this.ecosystem);
                break;
        var chromosome1 = new Antibody();
        var chromosome2 = new Antibody();
        chromosome1.initWithVariable(descendant[0],orderedDesc[0],this.ecosystem);
        chromosome2.initWithVariable(descendant[1],orderedDesc[1],this.ecosystem);
        children.push(chromosome1);
        children.push(chromosome2);
        if(chromosome1.check()==false){
          alert("error in ga.js:Genetics.cross("+this.ecosystem.crossoverType+")");}
        if(chromosome2.check()==false){
          alert("error in ga.js:Genetics.cross("+this.ecosystem.crossoverType+")");}
          return children;}
    this.mutate = function(){
        var G = new Genetics();
        switch (this.ecosystem.mutationType){
            case 1:
                   this.machine = G.mutationReverse(this.machine,this.ecosystem);
                this.order = G.mutationReverse(this.order,this.ecosystem);
                this.machine = G.repair(this.machine,this.ecosystem);
                break;
            case 2:
                this.machine = G.mutationNeighbor(this.machine,this.ecosystem);
                this.order = G.mutationNeighbor(this.order,this.ecosystem);
                this.machine = G.repair(this.machine,this.ecosystem);
                break;
            case 3:
                this.machine = G.mutationArbitrary(this.machine,this.ecosystem);
                this.order = G.mutationArbitrary(this.order,this.ecosystem);
                this.machine = G.repair(this.machine,this.ecosystem);
                break;
            case 4:
                this.order = G.mutation3Arbitrary(this.order,this.ecosystem);
                break;
            case 5:
                this.machine = G.mutationFloat(this.machine,this.ecosystem);
                this.order = G.mutationFloat(this.order,this.ecosystem);
                this.machine = G.repair(this.machine,this.ecosystem);
                break;
            default:
                this.machine = G.mutationReverse(this.machine,this.ecosystem);
                this.order = G.mutationReverse(this.order,this.ecosystem);
                this.machine = G.repair(this.machine,this.ecosystem);
                break;}
```

```
        this.balance();
        if(this.check()==false){
          alert("error in ga.js:Genetics.mutate("+this.ecosystem.mutationType+")");}}}
function AIS(){   //Artificial Immune System
    this.problem;           // problemin butun ozelliklerinin toplandigi object
    this.globalBest;        //Solution tipinden en iyi cozum
    this.progress;
    this.polygonID;
    this.solutionTime;
    this.populationNumber;
    this.generationNumber;
    this.mutationProbability;
    this.cloningProbability;
    this.memoryCellNumber;
    this.antibodyList;              // antibody populasyonum
    this.cloneFlag;                //
    this.currentGeneration;         // jenerasyon loop index degiskeni
    this.G = new Genetics();
    this.initWithVariable =
function(problem,population,generation,mutationProbability,cloningProbability,memoryCell
Number){
        this.problem = problem;
        this.populationNumber = parseInt(population);
        this.generationNumber = parseInt(generation);
        this.mutationProbability = parseFloat(mutationProbability);
        this.cloningProbability = parseFloat(cloningProbability);
        this.memoryCellNumber = parseInt(memoryCellNumber);}
    this.cloneSelectionRouletteWheel = function(cloneNumber){
         var cloneList = new Array();                                          //
        var fitnessList = new Array(this.antibodyList.length);               //
        var cumFitList = new Array(this.antibodyList.length);                //
        var total = 0;
        var maxCmax = 0                                          //
        for(var i=0;i<this.antibodyList.length;i++){
            this.cloneFlag[i]=0;                                 //
            if(this.antibodyList[i].getMakespan()>maxCmax)
                maxCmax = this.antibodyList[i].getMakespan();}
        for(var i =0;i<this.antibodyList.length;i++){            //
            fitnessList[i]= maxCmax - this.antibodyList[i].getMakespan() + 1;
            total+=fitnessList[i];}
        cumFitList[0] = fitnessList[0];
        for(var i=1;i<this.antibodyList.length;i++){
            cumFitList[i] = cumFitList[i-1]+fitnessList[i];         // }
        for(var cloneIndex = 0; cloneIndex<cloneNumber;cloneIndex++){       //
            var random = Math.floor(Math.random()*total);                //
            for(var i=0;i<this.antibodyList.length;i++){            //
if(cumFitList[i]> random)  {
                    this.cloneFlag[i]=1;
                    cloneList.push(this.antibodyList[i]);
                    break;}}}
        return cloneList;}
    this.sortWithMakespan = function(){
        for(var i=0;i<this.antibodyList.length;i++){                            //
            this.antibodyList[i].makespan = this.antibodyList[i].getMakespan();}
        this.antibodyList.sort(function(a,b){             // kucukten buyuge siralar
            if(a.makespan > b.makespan) return 1;
            if(a.makespan < b.makespan) return -1;
            return 0;});}
    this.sortClones = function (cloneList) {
        cloneList.sort(function(a,b){
            if(a.makespan > b.makespan) return 1;
            if(a.makespan < b.makespan) return -1;
            return 0;});}
    this.somaticHypermutation = function (cloneList) {
        this.sortClones(cloneList);
        var mutationNumber = Math.floor(cloneList.length*this.mutationProbability);
        for(var i=cloneList.length-mutationNumber;i<cloneList.length;i++){
            var tempAntibody = jQuery.extend({}, cloneList[i]);  // object copy
            this.G.mutationFloat(tempAntibody.order,this.problem);
            if(cloneList[i].makespan > tempAntibody.getMakespan()){
                cloneList[i] = tempAntibody;
                continue; }
            tempAntibody = jQuery.extend({}, cloneList[i]);  // object copy
            this.G.mutationArbitrary(tempAntibody.order,problem);
            if(cloneList[i].makespan > tempAntibody.getMakespan()){
                cloneList[i] = tempAntibody;
                continue;}
```

```
            tempAntibody = jQuery.extend({}, cloneList[i]);  // object copy
            this.G.mutationChangeMachine(tempAntibody.machine,problem);
            if(cloneList[i].makespan > tempAntibody.getMakespan()){
                cloneList[i] = tempAntibody;
                continue;}
            tempAntibody = jQuery.extend({}, cloneList[i]);  // object copy
            this.G.mutationChange2Machine(tempAntibody.machine,problem);
            if(cloneList[i].makespan > tempAntibody.getMakespan()){
                cloneList[i] = tempAntibody;
                continue;}}}
    this.receptorEditing = function () {
        var receptorList = new Array();
        for(var i=0;i<this.antibodyList.length;i++) {
            if (this.cloneFlag[i] == 0) {
          receptorList.push(this.antibodyList[i]);}}
        for(var i=0;i<receptorList.length;i++){
            var tempAntibody = jQuery.extend({}, receptorList[i]);  // object copy
            this.G.mutationReverse(tempAntibody.order,problem);
            if(receptorList[i].getMakespan() > tempAntibody.getMakespan()){
                receptorList[i] = tempAntibody;
                continue;
            }
            tempAntibody = jQuery.extend({}, receptorList[i]);  // object copy
            this.G.mutationResetMachine(tempAntibody.machine,problem);
            if(receptorList[i].getMakespan() > tempAntibody.getMakespan()){
                receptorList[i] = tempAntibody;
                continue;
            }
            tempAntibody = jQuery.extend({}, receptorList[i]);  // object copy
            this.G.mutationChangeMachine(tempAntibody.machine,problem);
            if(receptorList[i].getMakespan() > tempAntibody.getMakespan()){
                receptorList[i] = tempAntibody;
                continue;
            }
            receptorList[i] = new Antibody();
            receptorList[i].initWithProblem(this.problem);
            receptorList[i].getMakespan();
        }
        this.sortClones(receptorList);
        return receptorList;}
    this.solve = function(){
        var date1 = new Date().getTime();
        var date2 = new Date().getTime();
        console.log(date1);
        this.antibodyList = new Array();
        this.cloneFlag = new Array(this.populationNumber);
        for(var i=0;i<this.populationNumber;i++){
            var tempAntibody = new Antibody();
            tempAntibody.initWithProblem(this.problem);
            this.antibodyList.push(tempAntibody);}
        this.sortWithMakespan();
        this.globalBest = new Solution();
        console.log(this.antibodyList);
        this.globalBest.initWithAntibody(this.antibodyList[0]);
        for(this.currentGeneration = 0;this.currentGeneration<
this.generationNumber;this.currentGeneration++) {
            var cloneNumber =
Math.floor(this.antibodyList.length*this.cloningProbability);
            this.sortWithMakespan();
            var cloneList = this.cloneSelectionRouletteWheel(cloneNumber);
            this.somaticHypermutation(cloneList);
            var receptorList = this.receptorEditing();
            this.antibodyList =
cloneList.concat(receptorList.slice(0,this.populationNumber-cloneNumber));
            this.sortWithMakespan();
            for(var i=0;i<this.memoryCellNumber;i++){
               this.antibodyList[this.antibodyList.length-i-1] =  this.antibodyList[i];}
            if(this.antibodyList[0].makespan < this.globalBest.makespan){
                this.globalBest = new Solution();
                this.globalBest.initWithAntibody(this.antibodyList[0]);
                var date2 = new Date().getTime();
                this.globalBest.solutionTime = date2 -date1;
                this.solutionTime = date2-date1;
this.progress.addDot(this.polygonID,this.globalBest.solutionTime,this.globalBest.makespa
n);
console.log(date2-date1);}        }        return;    }}
```

HİBRİT GENETİK ALGORİTMA – KARINCA KOLONİSİ OPTİMİZASYONU

```
function hAG(){
   This.date1;       this.date2;
   this.solutionTime;      this.problem;
   this.globalBest;     this.jobNumber;
   this.processNumber;     this.machineNumber;
   this.fuzzy = false;     this.progress;
   this.polygonID;      this.initWithVariable = function(problem){
      this.problem = problem; }
   this.solve = function(){      this.date1 = new Date().getTime();
      var ecosystem = new Ecosystem();
      var population=100, generation=500, mutationProbability=0.1,
crossoverProbability=0.9, maxRepeatNumber=50;
      var selectionMethod= 1, mutationType= 3, crossoverType=2;
      var valueList = new Array();
      for (var i = 0; i < this.problem.jobNumber; i++) {
         for (var j = 0; j < this.problem.processNumber; j++) {
            for (var k = 0; k < this.problem.machineNumber; k++) {
                  valueList[i * this.problem.processNumber * this.problem.machineNumber
+ j * this.problem.machineNumber + k] =
this.problem.processTimeList[i][j][k];            }}}
      ecosystem.initWithVariable(this.problem.jobNumber, this.problem.processNumber,
this.problem.machineNumber, generation, maxRepeatNumber, mutationProbability,
crossoverProbability, valueList, this.problem.setupTimeList, population, mutationType,
crossoverType, 1, selectionMethod);
      ecosystem.fuzzy = fuzzy;
      ecosystem.progress = this.progress;
      ecosystem.polygonID = this.polygonID;
      ecosystem.timeLimit = 1000;
      ecosystem.solveIPPS();
      var tempSol = new Solution();
      tempSol.initWithChromosome(ecosystem.champion);
      this.globalBest = jQuery.extend({}, tempSol);
      var supervisor = new Supervisor();
      var ant=10, iteration=500, maxRepeat=40, rho=0.5, alpha= 2, beta= 2, tho0= 2,
C=100, Q= 1000;
      supervisor.initWithVariable(ant, iteration, maxRepeat, rho, alpha, beta, tho0, C,
Q, this.problem.jobNumber, this.problem.processNumber, this.problem.machineNumber,
this.problem.processTimeList, this.problem.setupTimeList);
      supervisor.fuzzy = problem.fuzzy;
      supervisor.progress = this.progress;
      supervisor.polygonID = this.polygonID;
      supervisor.timeLimit = 1000;
      supervisor.timeBuffer = ecosystem.solutionTime;
      supervisor.solveIPPS();
      tempSol = new Solution();
      tempSol.initWithAnt(supervisor.winnerAnt);
   if(supervisor.Xab < ecosystem.makespan){
         this.globalBest = jQuery.extend({}, tempSol);      }
      var timeBuffer = supervisor.solutionTime + supervisor.timeBuffer;
      if(supervisor.Xab < ecosystem.makespan){
         ecosystem = new Ecosystem();
         population=20, generation=10, mutationProbability=0.1,
crossoverProbability=0.9, maxRepeatNumber=50;
```

```
         selectionMethod= 1, mutationType= 3, crossoverType=2;
         valueList = new Array();
         for (var i = 0; i < this.problem.jobNumber; i++) {
            for (var j = 0; j < this.problem.processNumber; j++) {
                  for (var k = 0; k < this.problem.machineNumber; k++) {
                     valueList[i * this.problem.processNumber *
this.problem.machineNumber + j * this.problem.machineNumber + k] =
this.problem.processTimeList[i][j][k];               }}}
         ecosystem.initWithVariable(this.problem.jobNumber, this.problem.processNumber,
this.problem.machineNumber, generation, maxRepeatNumber, mutationProbability,
crossoverProbability, valueList, this.problem.setupTimeList, population, mutationType,
crossoverType, 1, selectionMethod);
         ecosystem.fuzzy = fuzzy;
         ecosystem.progress = this.progress;
         ecosystem.polygonID = this.polygonID;
         ecosystem.timeLimit = 1000;
         ecosystem.initialPopulation = supervisor.winnerList;
         ecosystem.timeBuffer = timeBuffer;
         ecosystem.solveIPPS();
         console.log(ecosystem);
         tempSol = new Solution();
         tempSol.initWithChromosome(ecosystem.champion);
         if(ecosystem.makespan < this.globalBest.makespan){
            this.globalBest = jQuery.extend({}, tempSol);        }     }     else{
            supervisor = new Supervisor();
         ant=10, iteration=500, maxRepeat=40, rho=0.5, alpha= 2, beta= 2, tho0= 2,
C=100, Q= 1000;
         supervisor.initWithVariable(ant, iteration, maxRepeat, rho, alpha, beta, tho0,
C, Q, this.problem.jobNumber, this.problem.processNumber, this.problem.machineNumber,
this.problem.processTimeList, this.problem.setupTimeList);
         supervisor.fuzzy = problem.fuzzy;
         supervisor.progress = this.progress;
         supervisor.polygonID = this.polygonID;
         supervisor.timeLimit = 1000;
         supervisor.initialSolution = ecosystem.winnerList;
         supervisor.timeBuffer = timeBuffer;
         supervisor.solveIPPS();
               tempSol = new Solution();
         tempSol.initWithAnt(supervisor.winnerAnt);
         if(supervisor.winnerAnt < this.globalBest.makespan){
            this.globalBest = jQuery.extend({}, tempSol);        }           //*/}
      this.date2 = new Date().getTime();
      this.solutionTime = this.date2 - this.date1; }}
```

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Mehmet Fatih USLU
Doğum Tarihi ve Yeri : 29.03.1988, Isparta
Yabancı Dili : İngilizce
E-posta : mfatihuslu@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Doktora	Endüstri Müh.	Yıldız Teknik Üniversitesi	2016
Y. Lisans	Bilgisayar Müh.	Fatih Üniversitesi	2011
Lisans	Bilgisayar Müh.	İstanbul Teknik Üniversitesi	2009
Lise	Sayısal	Yıldırım Han Anadolu Lisesi	2005

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2015	Yıldız Teknik Üniversitesi	Laboratuvar personeli
2013	Kavun teknoloji	iOS yazılımcı
2011	Valensas	iOS yazılımcı

YAYINLARI

Bildiri

1. 2016 Course Scheduler and Recommendation System for Students, AICT, Azerbaycan
2. 2016 A Meta-Heuristic Approach for IPPS Problem, FLINS, Fransa

ÖDÜLLERİ

1. 2002 TÜBİTAK İlköğretim Matematik Olimpiyatı, bronz madalya
2. 2004 TÜBİTAK Lise Bilgisayar Olimpiyatı, Akdeniz bölge 1.liği
3. 2004 TÜBİTAK Lise Fizik Olimpiyatı, Akdeniz bölge 3.lüğü
4. 2006 Boğaziçi Üniversitesi Robocode yarışması 3.lüğü
5. 2013 ODTÜ Bilgisayar programlama yarışması 3.lüğü
6. 2013 Koç Üniversitesi Bilgisayar programlama yarışması 3.lüğü