T.C.
VAN YÜZÜNCÜ YIL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
İSTATİSTİK ANABİLİM DALI

DAĞITILMIŞ MONTAJ HATTI PERMÜTASYON AKIŞ TİPİ ÇİZELGELEME PROBLEMİ İÇİN YENİ ÇÖZÜM YÖNTEMLERİ

YÜKSEK LİSANS TEZİ

HAZIRLAYAN: Mehmet Ali ARVAS
DANIŞMAN: Dr. Öğr. Üyesi Alper HAMZADAYI

VAN-2020

T.C.
VAN YÜZÜNCÜ YIL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
İSTATİSTİK ANABİLİM DALI

DAĞITILMIŞ MONTAJ HATTI PERMÜTASYON AKIŞ TİPİ ÇİZELGELEME PROBLEMİ İÇİN YENİ ÇÖZÜM YÖNTEMLERİ

YÜKSEK LİSANS TEZİ

HAZIRLAYAN: Mehmet Ali ARVAS

VAN-2020

KABUL VE ONAY SAYFASI

İstatistik Anabilim Dalı'nda Dr. Öğr. Üyesi Alper HAMZADAYI danışmanlığında, Mehmet Ali ARVAS tarafından sunulan "**Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi İçin Yeni Çözüm Yöntemleri**" isimli bu çalışma Lisansüstü Eğitim ve Öğretim Yönetmeliği'nin ilgili hükümleri gereğince 20/04/2020 tarihinde aşağıdaki jüri tarafından oy birliği ile başarılı bulunmuş ve Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan: Dr. Öğr. Üyesi Alper HAMZADAYI İmza:

Üye: Dr. Öğr. Üyesi Muhammed Hanifi VAN İmza:

Üye: Dr. Öğr. Üyesi Serkan ARAS İmza:

Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 15/05/2020 tarih ve 2020/27-V sayılı kararı ile onaylanmıştır.

Prof Dr. Suat ŞENSOY

Enstitü Müdürü

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

Mehmet Ali ARVAS

ÖZET

DAĞITILMIŞ MONTAJ HATTI PERMÜTASYON AKIŞ TİPİ ÇİZELGELEME PROBLEMİ İÇİN YENİ ÇÖZÜM YÖNTEMLERİ

ARVAS, Mehmet Ali
Yüksek Lisans Tezi, İstatistik Anabilim Dalı
Tez Danışmanı: Dr. Öğr. Üyesi Alper HAMZADAYI
Mayıs 2020, 65 sayfa

Bu tez çalışmasında hem dağıtılmış imalatın hem de dağıtılmış montaj sistemlerinin dikkate alındığı Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi ele alınmıştır. Üretim endüstrinin önemli bir problemi olan bu tür karma modelli montaj hattı problemlerindeki amaç, nihai ürünlerin elde edilme zamanını en aza indirmektir (diğer bir değişle makespanı en aza indirmektir). Bu tezde de söz konusu problem tipinin çözümünü en iyilemenin bazı farklı yolları tartışılmaktadır.

Bu kapsamda, ilgili problem tipinin daha etkin bir şekilde çözülebilmesi için yeni karma tam sayılı doğrusal programlama modelleri önerilmiştir. Ayrıca hâlihazırda literatürde var olan sezgisel/meta sezgisel yöntemlere ilave olarak da bu tezde yeni bir meta sezgisel yöntem tasarlanmış ve geliştirilmiştir.

Son olarak, önermiş olduğumuz matematiksel modeller, literatürde var olan aynı veri setleri ve aynı özellikteki bilgisayar kullanılarak ve yine literatürde var olan en iyi matematiksel modelle karşılaştırılmıştır. Yapılan deneyler sonucunda literatürde var olan en etkili matematiksel modelden daha etkili bir matematiksel model geliştirdiğimiz görülmüştür. Aynı şekilde tarafımızdan önerilen meta sezgisel yöntem ise, ilgili problem için oluşturulmuş 1710 tane örnek setine yönelik şu ana kadar tespit edilen en iyi çözümlerin sonuçları göz önünde bulundurularak istatistiksel karşılaştırma yapılmıştır. Yapılan deneyler neticesinde, problemin çözümüne yönelik şu ana kadar literatürde var olan yöntemlerden daha etkin bir meta sezgisel algoritma geliştirdiğimiz görülmüş ve ayrıca mevcut en iyi çözümlerden 40 tane daha iyi sonuç tarafımızca bulunmuş ve rapor edilmiştir.

Anahtar kelimeler: Dağıtılmış montaj hattı permütasyon akışı çizelgeleme problemi, Matematiksel modelleme, Sezgisel/meta sezgisel çözüm yöntemleri

ABSTRACT

NEW SOLUTION METHODS FOR THE DISTRIBUTED ASSEMBLY PERMUTATION FLOWSHOP SCHEDULING PROBLEM

ARVAS, Mehmet Ali
M.Sc. Thesis, Department of Statistics
Supervisor: Asst. Prof. Dr. Alper HAMZADAYI
May 2020, 65 pages

In this thesis, Distributed Assembly Line Permutation Flowshop Scheduling Problem, which deals with both distributed manufacturing and distributed assembly systems, is discussed. The purpose of such mixed production problems, which is an important problem of the manufacturing industry, is to minimize the time of obtaining the final products (in other words, to minimize makespan). In this thesis, some different ways to optimize the solution of the problem in question are discussed.

Within this framework, new mixed integer linear programming models are proposed in order to solve this problem more effectively. In addition to the heuristic / meta heuristic algorithms currently available in the literature, a new meta heuristic algorithm has been designed and developed in this thesis.

Finally, the proposed mathematical models have been compared with the best mathematical model in the literature, using the same datasets and computers with the same features. As a result of the experiments, it has been seen that we have developed a more effective mathematical model than the most effective mathematical model available in the literature. Similarly, the meta heuristic algorithm proposed by us has been statistically compared to the results of the best solutions ever founded for the 1710 data sets generated for the problem. As a result of the experiments, it has been observed that we have developed a more effective meta heuristic algorithm for the solution of the problem than the methods available in the literature so far, and also 40 better results than the best solutions available in the current literature have been found and reported.

Keywords: Distributed assembly permutation flow shop scheduling problem, Mathematical modelling, Heuristic/meta heuristic solution methods

ÖN SÖZ

Her şeyden önce, tez yazma sürecinde her türlü ilgi ve yardımlarını esirgemeyen danışman hocam Sayın Dr. Öğr. Üyesi Alper HAMZADAYI'ya beni Yüksek Lisans öğrencisi olarak kabul ettiği için ve tez yazma sürecinde gösterdiği sabır ve rehberlik için içten teşekkürlerimi sunarım. Ayrıca Yüksek Lisans yapmaya beni teşvik eden, bu süreçte değerli tavsiyelerini esirgemeyen Prof. Dr. Halit Eray ÇELİK'e ve bana her zaman destek olan aileme teşekkürlerimi sunarım.

2020

Mehmet Ali ARVAS

İÇİNDEKİLER

Sayfa

Sayfa

ÇİZELGELER LİSTESİ

Çizelge **Sayfa**

ŞEKİLLER LİSTESİ

Şekil **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış bazı kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	**Açıklama**
ATÇP	Akış Tipi Çizelgeleme Problemi
BR-ILS	Eğilimli-Rasegelieli-Tekrarlamalı Yerel Arama **(**Biased-Randomized - Iterated Local Search)
BS-HH	Geriye Dönük Arama Hiper-Sezgisel (Backtracking Search Hyper-Heuristic)
ÇGSP	Çoklu Gezgin Satıcı Problemi
DIWO	Kesikli Akın Eden Yabancı Ot Optimizasyonu **(**Discrete Invasive Weed Optimization)
DMPÇP	Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi
DPÇP	Dağıtılmış Permütasyon Akış Tipi Çizelgeleme Problemi
EDAMA	Dağıtım Tabanlı Memetik Algoritması **(**Distribution-Based Memetic Algorithm)
FFO	Meyve Sineği Optimizasyonu (Fruit Fly Optimization)
GA	Genetik Algoritma (Genetic Algorithm)
GD	Büyük Tufan (Great Deluge)
GS	Açgözlü Arama **(**Greedy Search)
HBBO	Hibrid Biyocoğrafyaya Tabanlı Optimizasyon (Hybrid Biogeography-Based Optimization)
IG	Tekrarlamalı Açgözlü Arama (Iterated Greedy)
KTDP	Karışık Tamsayılı Doğrusal Programlama
MPÇP	Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi
PATÇP	Permütasyon Akış Tipi Çizelgeleme Problemleri

Kısaltmalar	Açıklama
PSO	Parçacık Sürüsü Optimizasyonu (Particle Swarm Optimization)
RPD	Bağıl Yüzdece Sapma **(**Relative Percentage Deviation)
RS	Rastgele Arama **(**Random Search)
SA	Tavlama Benzetimi (Simulated Annealing)
SS	Saçılım Araması (Scatter Search)
SSBM	Tek Çözüm Üzerinden Arama Yapan Meta-Sezgisel **(**Solution Based Search Meta-Heuristics)
SSS	Tekli Arayıcıların Topluluğu **(**The Single Seekers Society)
TA	Eşik-Kabul **(**Threshold-Accepting)
TS	Tabu Araması **(**Tabu Search)
VND	Değişken Komşuluklu İniş **(**Variable Neighbourhood Descent)
VNS	Değişken Komşuluklu Arama **(**Variable Neighborhood Search)
WSA	Ağırlıklı Süperpozisyon Çekimi **(**Weighted Superposition Attraction)

EKLER DİZİNİ

Sayfa

1. GİRİŞ

Günümüzde imalat sanayi her zamankinden daha zor rekabet koşullarıyla karşı karşıyadır. Pazarın küreselleşmesi, ürün yaşam döngüsünün azalması, üretilecek ürüne yönelik yenilik taleplerinin artması, daha hızlı teslimat, daha kaliteli ürünler, yüksek düzeyde optimize edilmiş üretim süreçleri ile artan verimlilik bugün imalat sanayinin karşılaştığı başlıca zorluklardandır.

Dağıtılmış üretim sisteminde çeşitli üretim kapasitesine sahip fabrikalar, farklı veya aynı tipte makineler ile donatılmış olup, coğrafi olarak birçok farklı yere dağılmış durumdadır ve paralel olarak çalışmaktadırlar (Peklenik, 1992). Yapılan araştırmalar dağıtılmış üretim sisteminin, işletmelerin daha iyi ürün kalitesi, daha düşük üretim maliyetleri, azaltılmış yönetim riskleri elde etmesini sağladığını kanıtlamış ve aynı zamanda dağıtılmış sistemlerde üretim maliyetlerinin, tek bir üretim merkezinin bulunduğu sistemlerden daha az olduğu görülmektedir (Mahdavi ve ark., 2008). Örneğin, birçok gelişmiş ülkenin yurtiçi işgücü maliyetlerindeki artış nedeniyle bazı Asya ülkelerine paralel üretim merkezi kurduğunu görmekteyiz. Böylelikle tek bir tedarikçiye kıyasla birden fazla tedarikçiye dağıtılan iş yükleri, yönetim risklerini önemli ölçüde azaltmıştır.

Dağıtılmış imalat ve montaj sistemlerinin, günümüzdeki imalat sanayinin karşılaştığı zorlukları aşma noktasında sunmuş olduğu katkılar oldukça önemlidir. Bu sistemlerin her ikisini aynı üretim sisteminde birlikte kullanarak, birleşmenin sunmuş olduğu avantajlarla, endüstrileri daha güçlü, daha esnek hale getirilebilir. Üretim aşamasında karşılaşılan zorluklarla başa çıkmak için daha verimli ve etkili çözüm yolları geliştirilebilir.

Diğer taraftan, üretim performansını önemli ölçüde etkileyebilecek birden fazla üretim sisteminden oluşan bu sistemleri optimize etmek, bunlardan sadece birini optimize etmekten daha karmaşıktır. Hiç şüphe yok ki söz konusu sistemin etkin bir şekilde programlanması üretim performansının artmasına neden olacaktır. Bu tez çalışmasında da, operasyonun verimliliğini en üst düzeye çıkarmak ve maliyetleri azaltmak için üretim sürecindeki iş yüklerini optimize etme işlemi çizelgeleme yoluyla gerçekleştirilmiştir.

Aynı zamanda son yıllarda aktif bir araştırma konusu haline gelen bu tür sistemleri çizelgelemek için yeni çözüm yöntemleri önerilmiştir.

1.1. Problemin Tanımı

Bu tezde, üretim sanayisinde hali hazırda kullanılmakta olan dağıtılmış imalat ve montaj sisteminden oluşan karma bir model incelenmiştir. Çalışılan model iki aşamadan oluşmaktadır: Birincisi farklı bileşenler üreten özdeş üretim fabrikaları, ikincisi ise daha önce imal edilen bileşenlerin, belirlenmiş olan montaj programları vasıtasıyla nihai ürünlere dönüştürüldüğü bir montaj aşamasıdır.

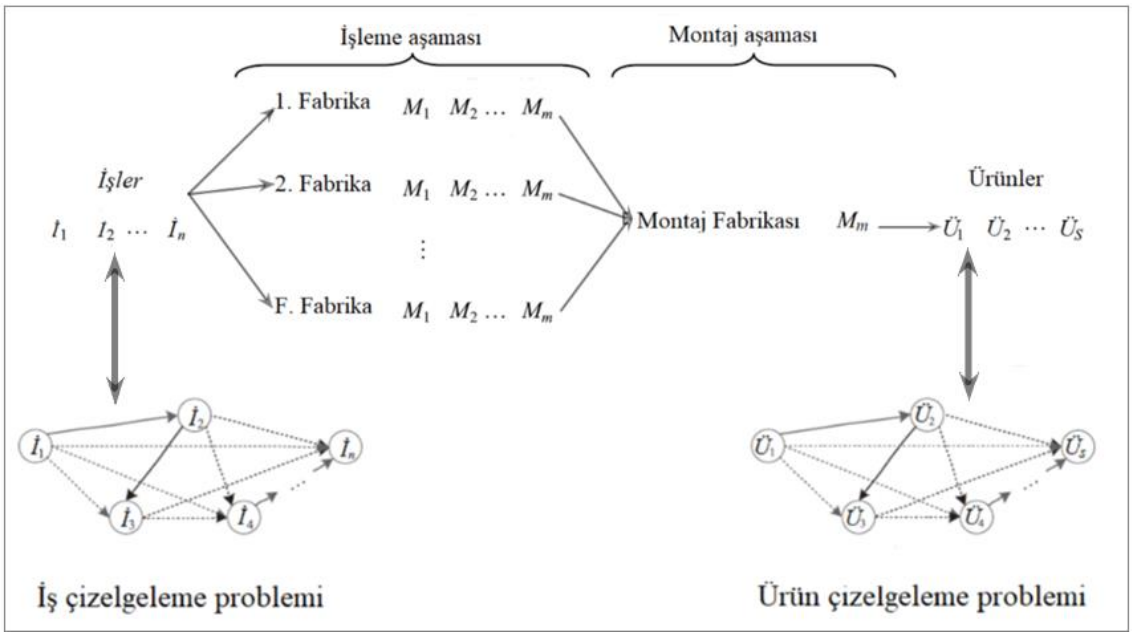
Her üretim sisteminin etkinliğinin değerlendirilmesi hususunda göz önünde bulundurulan en önemli kriterlerden biri toplam üretim süresidir. Bu problem tipinde de minimum üretim zamanını elde etmek ve genel performansı artırmak için nasıl planlama yapılması gerekliliğinin belirlenmesi önemli bir faktördür. Literatürde, üretim zamanının en küçüklenmesi konusunda bir çok çalışma yapılmış ve bu kriter çizelgeleme literatüründe kapsamlı bir şekilde incelenmiştir.

1.2. Tezin Amacı

Toplam üretim zamanını en aza indirgemek amacıyla yapılan üretim planlaması, imalat sistemlerindeki en önemli faktörlerden biridir. Bu tezin temel amacı ise montaj aşamasındaki ürünlerin üretim maliyetini minimuma indirecek üretim modellerini programlamaktır. Bu amaca, en son işi/ürünü mümkün olan en kısa sürede tamamlamaya çalışarak ulaşılacaktır.

Bu tezde gerçek hayat üretim sistemlerinde mevcut olan iki farklı problem eşzamanlı olarak ele alınmıştır. Bu problemler üretim sisteminde yer alan üretim ve montaj aşamalarıdır. İlk problemde, üretim fabrikalarının ilk aşaması olan akış atölyesi modellenmiştir. İşler ilk aşamada üretilir ve ikinci olarak üretilen ürünlerin tanımlanmış bir montaj programı ile montajı yapılarak nihai ürünler elde edilir. Bu sistem Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi (DMPÇP) olarak adlandırılır. Bu karmaşık yapılı problem tipini çözebilmek için tarafımızdan Karışık tamsayılı doğrusal programlama (KTDP) modeli geliştirilmiştir. Bu problem tipinin NP-Zor problemi olması

nedeniyle, ürünlerin montaj aşamasındaki üretim süresini en aza indirgemek için matematiksel modellerin yanında sezgisel yöntemler de sunulmaktadır. Üretim zamanını en aza indirmek için sadece montaj aşamasındaki ürünler dikkate alınmış ve bu sorunun şematik bir diyagramı Şekil 1.1'de gösterilmiştir.



Şekil 1.1. DMPÇP'nin şematik diyagramı.

Şekilden de görüldüğü üzere ikinci problem bir montaj sisteminden oluşur. İşler ilk aşamada işlenir ve ikinci aşama olan montaj aşamasında tanımlı bir montaj programı ile nihai ürünler elde edilir.

1.3. Yapılan Katkıların Özeti

Tez kapsamında yapılan katkılar aşağıda tanımlanmıştır:

- İki yeni matematiksel model önerilmiştir.
- Tekli arayıcıların topluluğu (SSS) algoritması DMPÇP için yeniden tasarlanmıştır.
- Küçük örneklerde KTDP modelini çözmek için IBM ILOG CPLEX ve Visual Studio C++ programları kullanılmıştır. Çözüm prosedürlerini incelemek ise için çeşitli test faktörleri uygulanmıştır.

- SSS algoritmasını test etmek için küçük ve büyük örnek kümeleri kullanılmış ve sonuçlar istatistiksel olarak analiz edilmiştir.
- 40 veri seti için yeni en iyi çözümler bulunmuştur.

1.4. Tezin Ana Hatları

Tezin geri kalanı aşağıdaki gibi düzenlenmiştir:

Bölüm 2: Tez için gerekli altyapı, çizelgeleme kavramları ve ardından çizelgeleme problemlerine genel bir bakış sunulmuştur. Dağıtılmış imalat ve montaj sistemlerinin bazı kavramları, optimizasyonun kısa bir tanıtımı ve son olarak çizelgeleme problemleri için bazı çözüm yöntemleri açıklanmıştır.

Bölüm 3: DMPÇP'nin ayrıntılı bir açıklaması sunulmaktadır.

Bölüm 4: Model 1 ve Model 2 adlı matematiksel modeller ve SSS meta-sezgisel algoritması sunulmuştur.

Bölüm 5: Matematiksel modeller ve meta-sezgisel algoritma test edilmiştir.
Tezde elde edilen tüm sonuçlar hakkında genel bir tartışma sunulmuştur.

Bölüm 6: Sonuçların tartışılmasını ve gelecekteki olası çalışmaları içeren genel bir sonuç bu son bölümde sunulmaktadır.

2. KAYNAK BİLDİRİŞLERİ

Üretim çizelgelemenin, üretim sistemindeki etkisinin ne denli önemli olduğunun anlaşılması nedeniyle, permütasyon akış tipi çizelgeleme problemleri (PATÇP) ile ilgili son zamanlarda literatürde çok sayıda çalışma yapıldığı görülmektedir. Geleneksel PATÇP'de amaç, toplam tamamlanma süresini en aza indirecek iş dizilimini bulmaktır. Ancak, dağıtılmış bir PATÇP'de (DPÇP), işlerin çizelgelemesinin yapılmasından önce, üretim merkezlerine hangi işlerin atanacağı konusunda alınması gereken ek bir karar vardır. Üretim planlamalarına etki edecek olan bu atama kararları optimizasyonun sağlanması açısından ek bir zorluktur. İlave olarak bir montaj aşaması da içeren PATÇP, montaj PATÇP'si (MPÇP) olarak bilinir. DPÇP'nin MPÇP ile doğal kombinasyonu, dağıtılmış montaj PATÇP'sini (DMPÇP) oluşturur. Böylece, DMPÇP dağıtılmış üretim aşamaları ile bir montaj aşamasından oluşur. Üretim aşaması, her biri PATÇP olarak modellenen dağıtılmış fabrikaları içermektedir (Naderi ve Ruiz, 2010). Montaj aşaması tek bir montaj makinesinden (atölyesinden) oluşur. Son zamanlarda, Komaki ve arkadaşları (2019), ikinci aşamada montaj operasyonu ile birlikte ilk aşamada akış atölyesi problemlerinin kombinasyonlarını dikkate alarak 126 yayını sınıflandırdıkları bir makale yayımlamışlardır. DMPÇP, literatürün bu revizyonuna dâhil edilmiştir. Bu yazarların da işaret ettiği gibi PATÇP'lerin montaj işlemleri ile kombinasyonu, yarı iletken imalatı, bilgisayar üretimi, gıda ve motorlu dişli imalatı endüstrileri de dahil olmak üzere farklı endüstrilerde görülebilmektedir. DMPÇP, literatüre Hatami ve ark. (2013) tarafından kazandırılmıştır. Bu yazarlar, karışık tamsayılı doğrusal programlama (KTDP) modelinin yanı sıra üç yapıcı algoritma ve değişken komşuluklu iniş (VND) algoritması önerdiler. Daha sonra DMPÇP için, Lin ve Zhang (2016) tarafından hibrit algoritma ve Wang ve Wang (2015) tarafından geliştirilen memetik algoritma çözüm yaklaşımı olarak önerilmiştir.

Son yıllarda, farklı yaklaşımların (genetik algoritmalar, doğadan ilham alan algoritmalar, memetik algoritmalar, vb.) hibridizasyonu da dâhil olmak üzere meta-sezgisel yaklaşımlarla ilgili literatürde önemli ölçüde gelişme sağlanmıştır. Hibrit

yaklaşımlar etkilerini arttırıyor olmalarına karşın, uzun ayarlama süreleri gerektiren çok sayıda parametreyi de içeriyor olmaları dezavantajlı tarafları olarak kabul edilmektedir. Son zamanlarda, Mladenovi´c ve ark. (2020), çözme yaklaşımlarını basitleştirme ve sezgisel olarak etkili kılan temel bileşenleri belirleme amacının *az ama öz* bir yaklaşım olduğunu ileri sürmüşlerdir. Daha sonra, bu çalışmada DMPÇP için yeni bir çözme yöntemi önerilmiştir.

MPÇP'nin en çok incelenen versiyonunun, ilk aşamasında paralel üretim makinelerinin olduğu, ikinci aşamasında ise sadece bir montaj makinesinin olduğu ve toplam üretim zamanının en aza indirmeye çalışıldığı bir sistemden oluştuğu söylenebilir. Bu versiyonun ilk çalışmalarından biri Lee ve ark. (1993) tarafından yapılmıştır. Bu çalışmada yeni bir matematiksel model sunulmuş ve polinom olarak çözülebilen bazı problemler tartışılmıştır. Ayrıca dal-bağlı bir algoritma önerilmiş ve yaklaşık çözümler elde etmek için üç sezgisel yöntem tasarlanmıştır. Potts ve ark. (1995), sorunun NP-Zor olduğunu gösterirken, Hariri ve Potts (1997) daha düşük sınırlar tahmin etmiş ve egemenlik teoremlerini belirlemişlerdir. Sun ve ark. (2003) ilk aşamada iki paralel makine ile konuyu ele almış, Johnson'ın algoritmasına dayanan farklı sezgisel yöntemler önermiş ve büyük örnekleri daha etkin bir şekilde çözebildiklerini kanıtlamışlardır. Allahverdi ve Al-Anzi (2006), parçacık sürüsü optimizasyonu (PSO), tabu araması (TS) ve kurulum zamanları dahil olmak üzere sorunu çözmek için başka sezgisel algoritmalar sunmuştur. Liao ve ark. (2015) bir KTDP modeli sunmuş, optimal çözümler bulmak için bazı özellikler geliştirmiş ve sezgisel bir algoritma önermiştir. Komaki ve Kayvanfar (2015), bırakma sürelerini (release times) dikkate alarak 15 sezgisel yöntem geliştirmiş ve sorunu çözmek için bir alt sınır oluşturup bir meta-sezgisel öneride bulunmuştur.

İki aşamalı MPÇP'deki bazı çalışmalar, toplam tamamlanma süresi (Allahverdi ve Al-Anzi, 2009), toplam gecikme (Allahverdi ve Aydilek, 2015) ve gecikmeli işler (Allahverdi ve ark., 2016) gibi farklı amaç fonksiyonları kullanılarak gerçekleştirilmiştir. MPÇP'nin daha karmaşık bir versiyonu olan üç aşamalı MPÇP, Hatami ve ark. (2010) ve Komaki ve ark. (2017) tarafından çalışılmıştır. Bu problem, üretim ve montaj aşamaları arasında bir taşıma operasyonunu dikkate almıştır. Yani üç aşamalı olan bir problemin ikinci aşamasında taşıma operasyonunun gerçekleştirildiği

bir problem türüdür. Son olarak, montaj aşamasının da paralel makinelere sahip olduğu MPÇP'nin farklı bir versiyonu Mozdgir ve ark. (2013) ve Shoaardebili ve Fattahi (2015) tarafından çalışılmıştır. Bu çalışmalar, sırasıyla toplam tamamlanma süresinin en aza indirilmesini ve ağırlıklı gecikme ve erkenliğin (sum of weighted tardiness and earliness) toplamının en aza indirilmesini hedeflemektedir.

Yukarıda belirtilen araştırmalardan anlaşılacağı üzere, MPÇP son yıllarda büyük ilgi görmüş ve geniş çapta incelenmiştir. DPÇP ise oldukça yeni bir konudur ve araştırmacılar bunu incelemeye yeni başlamıştır. DPÇP, Naderi ve Ruiz (2010) tarafından ilk olarak çalışılmış olup toplam tamamlanma zamanını en aza indirgemek için farklı KTDP modelleri, yapıcı algoritmalar ve bir VND algoritması yapılan çalışmalarda önerilmiştir. Daha sonra, başka bilim adamları aynı sorunu çözmek için farklı yaklaşımlar önermişlerdir. Naderi ve Ruiz (2014) saçılım araması (SS) algoritmasını geliştirmiş ve Fernandez-Viagas ve Framinan (2015) sınırlı bir arama olan tekrarlamalı açgözlü arama (IG) algoritmasını tasarlamıştır. Daha sonra, Lin ve Ying (2016), DPAÇP'nin beklemesiz kısıtlı bir varyantını geliştirdiler. Bu yazarlar bir KTDP modeli ve çözme yaklaşımları olarak tekrarlamalı kokteyl açgözlü arama algoritması sundular. DPÇP'nin stokastik makine arızası dikkate alınarak stokastik versiyonu ilk kez Wang ve ark. (2016) tarafından çalışılmıştır. Bulanık mantık uyarlamalı strateji içeren bulanık mantık tabanlı hibrit EDA bu yazarlar tarafından önerilmiştir. Önerilen yöntem saf EDA ve GA ile karşılaştırılmıştır.

DPÇP'nin genelleştirilmiş bir versiyonu olan DMPÇP, ilk olarak Hatami ve ark. (2013) tarafından tanıtılmıştır. Bu yazarlar bir KTDP modeli önermişler ve toplam tamamlanma zamanını en aza indirmek için sezgisel bir metodun yanında bir de VND algoritması geliştirdiler. Benzer şekilde Hatami ve ark. (2015) DMPÇP'ye mümkün olduğunca günümüz sorunlarına daha gerçekçi çözümler bulsun diye sıraya bağlı kurulum süreleri eklediler. Bu yazarlar ayrıca farklı yapıcı sezgisel bir algoritma, bir VND algoritması ve bunu çözmek için bir IG algoritması geliştirdiler. Xiong ve Xing (2014), DMPÇP'nin farklı bir versiyonunu incelemişler ve burada her akış atölyesi fabrikası için bir montaj makinesi olacak şekilde problemi tasarlamışlardır. Değişken komşuluklu arama (VNS) ve bir GA'yla, azaltılmış bir VNS arasındaki bir hibridizasyon, çözme yaklaşımı olarak tasarlanmıştır. Ji ve ark. (2016), stokastik işleme

süreleri ve beklemesiz kısıtlar kullanarak DMPÇP'nin stokastik bir versiyonunu dikkate almıştır. Sorunu çözmek için simüle tavlama ile birleştirilmiş bir PSO çerçevesi kullanmışlardır. Ayrıca, Du ve ark. (2016) stokastik diziye bağlı kurulum süreleri ve stokastik salım süreleri ekleyerek Ji ve ark. (2016) tarafından analiz edilen problemi zenginleştirmiştir. Gonzalez-Neira ve ark. (2017) ise, stokastik işlem süreli olan versiyon için bir simheuristic yaklaşım önermiştir. Son zamanlarda, Sang ve ark. (2019), DMPÇP'deki akış süresini en aza indirmek için iki kesikli akın eden yabancı ot optimizasyonu (DIWO) algoritması geliştirmiştir.

DMPÇP için Lin ve Zhang (2016) hibrid biyocoğrafyaya tabanlı optimizasyon (HBBO) algoritması önermişlerdir. Aynı şekilde, Wang ve Wang (2015), dağıtım tabanlı memetik algoritmasını (EDAMA) ortaya koymuşlardır. Bu algoritmaların her birinin Hatami ve ark. (2013) tarafından sunulan ilk algoritmadan daha iyi performans gösterdiğine ilişkin karşılaştırmalar ilgili makalelerde yapılmıştır. Daha sonra Lin ve ark. (2017) tarafından geriye dönük arama hiper-sezgisel (BS-HH) algoritması, Pan ve ark. (2018) tarafından meyve sineği optimizasyonu (FFO) algoritması ve Ferone ve ark. (2019) tarafından eğilimli-rasegelieli-tekrarlamalı yerel arama (BR-ILS) algoritması geliştirilmiş ve her algoritma HBBO ve EDAMA ile karşılaştırılmıştır.

Bu yazarlar elde ettikleri sonuçlara göre daha etkin yöntemler keşfettiklerini göstermişlerdir. Ancak, BS-HH, FFO ve BR-ILS arasında henüz bir karşılaştırma yoktur. Bildiğimiz kadarıyla literatürde bu sorunu çözmek için şimdiye kadar başka bir çözme metodolojisi sunulmamıştır. Bu çalışmada ise, DMPÇP için şu ana kadar geliştirilmiş tüm algoritmalarla karşılaştırma yapılmış ve nispeten daha etkili algoritmalar geliştirdiğimiz görülmüştür.

2.1. Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi

Günümüzde, karmaşık tedarik zincirlerinin yönetimini geliştirmek, yirmi birinci yüzyıl küresel pazarında rekabet edebilmek için çok önemlidir. Tedarik zincirleri, teslim sürelerini azaltmak için koordine edilmesi gereken çok tesisli yapılardan oluşur. Bu bölüm, karmaşık tedarik zincirlerini modellemek ve incelemek için iki aşamalı bir Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi (DMPÇP)

önermiştir. Bu sorun, Dağıtılmış Permütasyon Akış Tipi Çizelgeleme Probleminin (DPÇP) genelleştirilmiş versiyonudur. DMPÇP'nin ilk aşaması *F* tane aynı üretim fabrikasından oluşmaktadır. Her biri, ikinci bir montaj aşamasında nihai ürünlere monte edilecek işleri üreten bir akış atölyesidir. Bu problem tipindeki amaç, nihai ürünlerin tamamlanma zamanını en aza indirmektir. Bu amaç doğrultusunda; önce Karışık Tamsayılı Doğrusal Programlama (KTDP) modeli sunulmuştur. Daha sonra Model 1 ve Model 2 isimli iki yeni matematiksel model ile SSS meta sezgisel algoritması açıklanmıştır. Son olarak; matematiksel modeller, şimdiye kadar keşfedilmiş en iyi matematiksel modelle, SSS algoritması ise şimdiye kadar üzerinde çalışma yaptığımız örneklerle ilgili en iyi sonuçları veren tüm sezgisel ve meta sezgisel algoritmalar ile karşılaştırılmıştır. Elde edilen sonuçlar, tarafımızdan sunulan matematiksel modellerin ve SSS algoritmasının, DMPÇP'yi çözmek için iyi bir performans sunduğunu göstermiştir.

2.1.1. Giriş

Tez çalışmamızda ele alacağımız birinci problem, her fabrika için akış planlamasını tasarlamak üzerine olacaktır. Akış Tipi Çizelgeleme Problemi (ATÇP), bir set *N* işinde yer alan her bir işin, her bir makinede işlenmesi gereken bir *M* makinesi setinden oluşur. İş başına düşen işlem sayısı, makine sayısına eşittir. Her işin, *i*'nci işlemi, *i*'nci makinede gerçekleşir. Bu nedenle, bir işin *i*'nci makinedeki işleminin başlaması için *i*-1 deki işleminin tamamlanmış olması ve *i*'nci makinanın da boş olması gerekir. Makinelerdeki her işin işlem süresi önceden bilinen ve negatif olmayan bir sayıdır.

ATÇP'lerde birtakım varsayımlar kabul edilmektedir:

- Tüm işler 0 zamanında işlenmeye uygundur;
- Makineler sürekli olarak çalışır durumdadır;
- Her makine bir kerede yalnızca bir işi işleyebilir;
- Her iş bir kerede yalnızca bir makinede işlenebilir;

- Belirli bir işin işlenmesi belirli bir makinede başladığında, kesintiye uğramaz ve işlem tamamlanıncaya kadar devam eder;
- Kurulum süreleri işlem sürelerine dahil edilir veya dikkate alınmaz.

ATÇP'de her makine için $n!$ olası iş dizini bulunmaktadır. Bu nedenle, m makineli akış atölyesi problemi için toplam çözüm sayısı $(n!)^m$'dir. Sorunu basitleştirmek için, tüm makinelerin aynı iş iznine sahip olduğu varsayılmaktadır. Başka bir deyişle, bir iş, makine 1'de j'ninci pozisyonundaysa, bu işin diğer tüm makinelerde de j'ninci konumunda olması gerekir. Bu varsayımla, ATÇP'ye, $n!$ olası çözümle birlikte Permütasyon Akış Tipi Çizelgeleme Problemi (PATÇP) denir.

Çalışmamızda, Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi incelenmiştir. Bu çalışma: Dağıtılmış Permütasyon Akış Tipi Çizelgeleme Problemi ve Montaj Hattı Akış Tipi Çizelgeleme Probleminin birleşimidir. Üretim ve montaj aşamalarından oluşmaktadır. İlk aşama, bir N kümesi işinin programlanması gereken aynı fabrikalardan oluşmaktadır. Tüm fabrikalar tüm işleri yapabilme kapasitesine sahiptir ve her fabrika M makineli PATÇP'dir. Fabrikaların aynı olduğu varsayılmaktadır. İşlem süreleri p_{ij}, $i \in M$, $j \in N$ olarak belirtilir. İkinci aşama, bir montaj makinesine (M_A) sahip olan ve farklı nihai ürünlerden oluşan bir T seti yapmak için belirlenmiş bir montaj programını kullanarak işleri yapan tek bir montaj fabrikasıdır. Her ürün tanımlanmış bir montaj programına sahiptir; başka bir deyişle, her ürün bazı tanımlanmış işlerden oluşur. N_h ve J_j, sırasıyla ürün h montaj programını ve ürün h montaj programına ait işleri temsil etmek için kullanılır, $N_h : \{J_j\}$, $j \in N_h$. Her h ürününün $| N_h |$ işi vardır ve j işi bir ürünün montajı için gereklidir. Bu nedenle, $\sum_{h=1}^{t} |Nh| = n.$ h ürününün montajı ancak fabrikalarda N_h'ye ait tüm işler tamamlandığında başlayabilir. Amacımız, montaj fabrikasındaki ürünlerin tamamlanma zamanını en aza indirmektir.

2.1.2. Hatami ve ark. (2013) tarafından önerilen matematiksel model

Tablo' da tezde sunduğumuz bütün modellerde yaygın olarak kullanılan parametreler ve indeksler sunulmuştur.

Çizelge 2.1. Modellerde kullanılan ortak parametreler ve indisler

Parametreler	Tanım
N	İş sayısı
M	Makine sayısı
F	Fabrika sayısı
P	Montajı yapılacak ürün sayısı
j, k	İşler için indeks (veya dizilimdeki işlerin sırası); $j, k \in \{1, 2,..., N\}$
i	Makineler için indeks; $i \in \{1, 2,...,M\}$
l, s	Ürünler için indeks; $l, s \in \{1, 2,...,P\}$
f	Fabrikalar için indeks; $f \in \{1, 2,..., F\}$
$O_{j,i}$	İş j'nin makine i deki operasyonu
$po_{j,i}$	Operasyon $O_{j,i}$'nin işlem zamanı
pp_s	s ürününün montaj zamanı
$G_{j,s}$	j işi s ürününe aitse 1 değerini alır, aksi taktirde 0 değerini alır
$bigM$	Yeterince büyük bir pozitif tamsayı

Hatami ve ark. (2013) tarafından önerilen matematiksel model sıralama tabanlı bir modeldir. Bu modelde yazarlar aşağıda verildiği gibi iki tane sıralama tabanlı ikili değişken ve iki sürekli değişken kullanmıştır.

$X_{k,j} \rightarrow k$ işi j işinden hemen önce işlenecekse 1 değerini alan; ve aksi takdirde 0 değerini alan ikili değişkendir.

$Y_{l,s} \rightarrow l$ ürününün montajı s ürününden hemen önce yapılacaksa 1 değerini alan; ve aksi takdirde 0 değerini alan ikili değişkendir.

$C_{j,i} \rightarrow i$. makinede j işinin tamamlanma süresini gösteren sürekli değişkendir.

$CA_s \rightarrow s$ ürününün tamamlanma süresini gösteren sürekli değişkendir.

Hatami ve ark. (2013) tarafından önerilen matematiksel model aşağıdaki gibidir.

Amaç fonksiyonu:

$$\text{En küçükle } C_{\max} \tag{2.1}$$

Kısıtlar:

$$\sum_{k=0}^{N} X_{k,j} = 1 \quad j \in \{1, \dots, N\} \mid j \neq k \tag{2.2}$$

$$\sum_{j=0}^{N} X_{k,j} \leq 1 \quad k \in \{1, \dots, N\} \mid k \neq j \tag{2.3}$$

$$\sum_{j=1}^{N} X_{0,j} = F \tag{2.4}$$

$$\sum_{k=1}^{N} X_{k,0} = F - 1 \tag{2.5}$$

$$X_{k,j} + X_{j,k} \leq 1 \quad k \in \{1, \dots, N-1\};\ j \in \{1, \dots, N\} \mid j \neq k, j > k \tag{2.6}$$

$$C_{j,i} \geq C_{j,i-1} + po_{j,i} \quad j \in \{1, \dots, N\};\ i \in \{1, \dots, M\} \tag{2.7}$$

$$C_{j,i} \geq C_{k,i} + po_{j,i} + (X_{k,j} - 1) \cdot bigM \quad k, j \in \{1, \dots, N\} \mid k \neq j;\ i \in \{1, \dots, M\} \tag{2.8}$$

$$\sum_{l=0, l \neq s}^{P} Y_{l,s} = 1 \quad s \in \{1, \dots, P\} \tag{2.9}$$

$$\sum_{s=1, l \neq s}^{P} Y_{l,s} \leq 1 \quad l \in \{1, \dots, P\} \tag{2.10}$$

$$Y_{l,s} + Y_{s,l} \leq 1 \quad l \in \{1, \dots, P-1\};\ s \in \{1, \dots, P\} \mid l \neq s, l > s \tag{2.11}$$

$$CA_S \geq (C_{j,M} \cdot G_{j,s}) + pp_S \quad j \in \{1, \dots, N\};\ s \in \{1, \dots, P\} \tag{2.12}$$

$$CA_S \geq CA_l + pp_S + (Y_{l,s} - 1) \cdot bigM \quad l, s \in \{0, \dots, P\} \mid l \neq s \tag{2.13}$$

$$C_{j,0} = 0 \quad j \in \{1, \dots, N\} \tag{2.14}$$

$$CA_0 = 0 \tag{2.15}$$

$$C_{max} \geq CA_S \quad s \in \{1, \dots, P\} \tag{2.16}$$

$$X_{k,j} \in \{0,1\} \quad k,j \in \{0,\dots,N\} \mid k \neq j \tag{2.17}$$

$$Y_{l,s} \in \{0,1\} \quad l,s \in \{0,\dots,P\} \mid l \neq s \tag{2.18}$$

$$C_{j,i} \geq 0 \quad j \in \{1,\dots,N\};\ i \in \{1,\dots,M\} \tag{2.19}$$

$$CA_S \geq 0 \quad s \in \{1,\dots,P\} \tag{2.20}$$

Modelin amacı çevrim zamanını en aza indirmektir Eş. 2.1. Kısıtlama seti Eş. 2.2, her bir *j* işinden hemen önce sadece bir tane önceki işin (*k*) olmasını kontrol eder ve sağlar. Kısıtlama kümesi Eş. 2.3, her bir *k* işinden sonra en fazla bir tane iş (*j*) gelebilmesini sağlar. Kısıtlama seti Eş.2.4, kukla iş 0'ın önceki iş olarak fabrika sayısı kadar (*F* defa) gözükmesini sağlar. Diğer bir değişle bu kısıt seti $X_{j,k}$ karar değişkeninde *j* pozisyonunu alan toplam 0 işlerinin sayısının *F* olmasını garanti eder. Kısıtlama seti Eş. 2.5 ayrıca kukla iş 0'ın *F*-1 kez izleyen iş olarak gözükmesini sağlar. Diğer bir değişle bu kısıt seti $X_{j,k}$ karar değişkeninde *k* pozisyonunu alan toplam 0 işlerinin sayısının *F*-1 olmasını sağlar. Kısıtlama seti Eş. 2.6 bir işin aynı anda başka bir işin hem önceki hem de sonraki işi olamayacağını kontrol eder. Kısıtlama seti Eş. 2.7, makine *i*-1'deki *j* işinin işleminin tamamlanması durumunda, bir sonraki makinede (makine *i*) *j* işinin işlenmesinin gerçekleşmesini sağlar. Kısıtlama seti Eş. 2.8, eğer *j* işi, *k* işinden hemen sonra işlenecekse, *i* makinasındaki *k* işi bitmeden, *j* işinin aynı makinedeki işleminin başlamamasını sağlar. Kısıtlama seti Eş. 2.9, her bir *s* ürününden hemen önce montajı yapılacak sadece bir tane önceki ürünün (*l*) olmasını kontrol eder ve sağlar. Kısıtlama kümesi Eş. 2.10, her bir *l* ürününden sonra en fazla bir tane ürünün (*s*) gelebilmesini sağlar. Kısıt seti Eş. 2.11, montajı yapılacak bir ürünün aynı anda montajı yapılacak başka bir ürünün hem öncül hem de ardılı olamayacağını kontrol eder. Kısıt Eş. 2.12, her *s* ürününün, en son makinadaki (*M*) tüm işleri tamamlanmadan, montajına başlanmayacağını belirtir. Kısıtlama seti Eş. 2.13, eğer *s* ürününün montajı, *l* ürününden hemen sonra yapılacaksa, *s* ürününün montaj makinesi üzerindeki işleminin başlaması için önce *l* ürününün işleminin bitmesi gerektiğini belirler. Kısıt seti Eş. 2.14 kukla makine olan 0'da *j* işinin tamamlanma zamanının 0 olmasını sağlar. Benzer şekilde kısıt seti Eş. 2.15 kukla ürün olan 0 in tamamlanma zamanının 0 olmasını sağlar. Kısıt seti Eş. 2.16, tamamlanma zamanını tanımlarken, kısıt setleri Eş. 2.17- Eş. 2.20 karar değişkenlerinin sınırlarını tanımlar.

Bu modelin önemli noktası fabrikalar için indeks kullanmamasıdır. Sıra tabanlı değişkenler bu nedenle bir dizi kukla işle birlikte kullanılır. Bu kukla işler tüm işleri alt iş kümelerine böler ve bunları her fabrikaya atar (yani, ilk kukla iş ile ikinci kukla iş arasında yer alan tüm işler ilk fabrikaya aittir). Örneğin, N = 8 ve F = 3 olan bir problemin olası çözümlerinden biri $X_{0,2}$ =$X_{2,3}$ =$X_{3,5}$ =$X_{5,0}$ =$X_{0,6}$ =$X_{6,1}$ =$X_{1,4}$ =$X_{4,0}$ =$X_{0,7}$ =$X_{7,8}$ = 1, o zaman dizi {0, 2, 3, 5, 0, 6, 1, 4, 0, 7, 8} şeklini alır ve burada kısmi iş dizileri {2, 3, 5}, {6, 1, 4} ve {7, 8} sırasıyla 1, 2 ve 3 fabrikalarına atanmış olur.

3. MATERYAL ve YÖNTEM

Önermiş olduğumuz matematiksel modellerin ve meta-sezgisel algoritmanın etkinliğini test etmek için, Hatami ve ark. (2013) tarafından oluşturulan iki kriter seti kullanılmıştır. Birinci örnek setinde iş sayısı N = {8, 12, 16, 20, 24}, makine sayısı M = {2, 3, 4, 5}, fabrika sayısı F = {2, 3, 4} ve ürün sayısı P = {2, 3, 4} olmak üzere 900 küçük boyutlu örnek vardır. İkinci örnek setinde ise, N = {100, 200, 500}, M = {5, 10, 20}, F = {4, 6, 8} ve P = {30, 40, 50} olmak üzere 810 büyük boyutlu örnek vardır. Önermiş olduğumuz matematiksel modeller test edilirken C-PLEX Studio 1271 ve Visual Studio 2015 - C++ yazılımlarına kodlama yapılmıştır. Tarafımızdan önerilen meta – heuristic algoritmanın performansı test edilirken ise MATLAB R2017a yazılımı kullanılmıştır. Söz konusu yazılımlar 2.30 GHz ve 8 GB RAM ile i5-4200U işlemci özelliklerine sahip bilgisayar aracılığıyla çalıştırılmıştır.

Yaptığımız çalışmada, belirtilen küçük ve büyük boyutlu örnek setleri kullanılarak, geliştirmiş olduğumuz matematiksel modellerin ve meta – sezgisel algoritmanın performansı test edilmiştir. Bu kapsamda matematiksel modeller sadece küçük örnek setleri kullanılarak test edilmiştir. Her bir örnek için algoritmanın çalışma süresi 600 saniye olarak sınırlandırılmıştır. Meta – sezgisel algoritma olan SSS algoritmasında ise hem küçük hem de büyük ölçekli örnekler test edilmiştir. Her bir örnek için algoritmanın çalışma süresi ise çözülecek örneğin makine ve iş sayısına bağlı olarak MxNx20 milisaniye ile kısıtlandırılmıştır.

3.1. Matematiksel Modeller

Hatami ve ark. (2013) tarafından önerilen matematiksel model (MILP) baz alınarak Model 1 ve Model 2 olmak üzere iki yeni matematiksel model geliştirilmiştir.

3.1.1. Model 1

Amaç Fonksiyonu: Eşitlik (2.1)

Kısıtlar:

$$\sum_{k=1}^{N} X_{k,0} = F \tag{3.1}$$

$$\sum_{j=0}^{N} X_{k,j} = 1 \quad k \in \{1, \dots, N\} | \, k \neq j \tag{3.2}$$

$$\sum_{s=1}^{P} Y_{0,s} = 1 \tag{3.3}$$

$$U_k - U_j + NX_{k,j} \leq N - 1 \quad k, j \in \{1, \dots, N\} \mid k \neq j \tag{3.4}$$

Kısıt setleri Eş. (2.2)- Eş. (2.4), ve Eş. (2.7)- Eş. (2.20)

Bu model, Çoklu Gezgin Satıcı Problemi (ÇGSP) için geliştirilen atama tabanlı lineer tam sayılı modelden esinlenilerek geliştirilmiştir (Bektas, 2006). ÇGSP'deki m adet araç bizim problemimizde *F* adet fabrikaya karşılık gelmektedir. ÇGSP'de herhangi bir araç yola çıktığında belli sayıdaki müşterileri gezip tekrardan geri gelmektedir. Bizim problemimizde ise herhangi bir fabrikaya atanan belli sayıda iş tamamlanıncaya kadar üretim devam etmektedir. Bizim problemimizdeki işler ÇGSP'deki müşterilere karşılık gelmektedir. Bu yeni modelde kısıtlama seti Eş. 3.1 kukla iş 0'ın *F* kez izleyen iş olarak gözükmesini sağlar. Diğer bir değişle bu kısıt seti $X_{j,k}$ karar değişkeninde k pozisyonunu alan toplam 0 işlerinin sayısının *F* olmasını garanti eder. Dikkat edilirse bu kısıt bu bağlamda Kısıt seti Eş. 5'den farklıdır. Kısıtlama kümesi Eş. 3.2, her bir k işinden sonra kesinlikle bir tane iş (j) gelmesini sağlar. Kısıtlama kümesi Eş. 3.3, kukla ürün olan 0'ın sıralamada diğer ürünlerden sadece bir tanesinden hemen önce olmasını garanti eder. Kısıt seti Eş. 3.4, ara düğümler arasında oluşturulan ve başlangıç noktasına bağlı olmayan dejenere turlar olan alt turları

önlemek için kullanılır. Bu kısıtlamalar, alt tur eliminasyon kısıtlamaları olarak adlandırılır. Eş. 3.4'deki alt tur engelleme kısıtı Kulkarni ve Bhave (1985) tarafından önerilmiştir.

3.1.2. Model 2

Amaç Fonksiyonu: Eşitlik (2.1)

Kısıtlar:

$$U_k - U_j + N\sum_{j=1}^{N} X_{k,j} + (N-2)\sum_{j=1}^{N} X_{j,k} \leq N-1 \quad (3.5)$$

Kısıt setleri Eş. 2.2- Eş. 2.4, Eş. 2.7- Eş. 2.20 ve Eş. 3.1- Eş. 3.3

Yukarıda önerdiğimiz modele benzer şekilde Desrochers ve Laporte (1991) tarfınfan daha güçlü olduğu önerilen başka bir alt tur eleme kısıt setini (Eş. 3.5) kullanarak başka bir model geliştirdik.

3.2. Meta Sezgisel Algoritma

Bu bölümde; büyük ölçekli Dağıtılmış Montaj Hattı Permütasyon Akışı Çizelgeleme Problemlerini (DMPÇP) çözebilmek için karmaşık ve sürekli optimizasyon problem tiplerini çözebilen ve bir meta-sezgisel algoritma olan tekli arayıcıların topluluğu (SSS) algoritması anlatılmıştır.

Baykasoğlu ve arkadaşlarının (2019) SSS algoritması; Dueck'in (1993) büyük tufan (GD) algoritması, Dueck'in (1990) eşik-kabul (TA) algoritması, Kirkpatrick ve Gelatt'in (1983) açgözlü arama (GS) algoritması, rastgele arama (RS) algoritması ve tavlama benzetimi (SA) algoritmasından oluşmaktadır. Karmaşık optimizasyon problemlerini çözmek için kullanılacak olacak olan bu farklı tek çözüm tabanlı sezgisel tarama algoritmaları genetik algoritma'nın (GA) temel yapısını (popülasyon yapısını) kullanarak bir araya gelmektedirler. Ayrıca, SSS algoritması bir bilgi paylaşım mekanizması olarak ağırlıklı süperpozisyon çekimi (WSA) algoritmasının "hedef noktası belirleme" prosedürünü kullanır. SSS algoritması, yukarıda belirtilen tekli

arayıcıları, özel durdurma koşulları karşılanana kadar tek bir çözümü geliştirmek için koordine eder ve daha sonra çoğaltma sürecine girmek için algoritmaların en iyi çözümlerini kullanır. Tekli arayıcılar, rastgele seçilmiş bir komşuluk mekanizmasını ve hedef nokta belirleme prosedürünü sırayla uygulayarak üzerlerine atanan algoritmaların özelliklerini kullanarak tekli çözümleri geliştirmeye çalışırlar. SSS algoritması, tekli arayıcıların, arama alanını kendi belirli parametre kümeleriyle ayrı ayrı keşfetmesini, bilgiyi çaprazlama yoluyla paylaşarak yeni çözümler üretmesini ve bilgiyi hedef noktası belirleme tabanlı komşuluk yapısı aracılığıyla entegre ederek arama işleminin yönlendirilmesini sağlar. Ardından, daha önce belirlenmiş durma koşulu gerçekleştiğinde kendini sonlandırır.

3.2.1. Çözüm gösterimi ve uygunluk hesabı

Çözümün gösteriminde permütasyon tabanlı bir yöntem kullanılır; yani, her çözüm (bireysel) bir tamsayı dizisi ile temsil edilir. Dizedeki her tam sayı bir işe karşılık gelir. Bir işin permütasyonu alındıktan sonra, toplam tamamlanma zamanı hesaplanır.

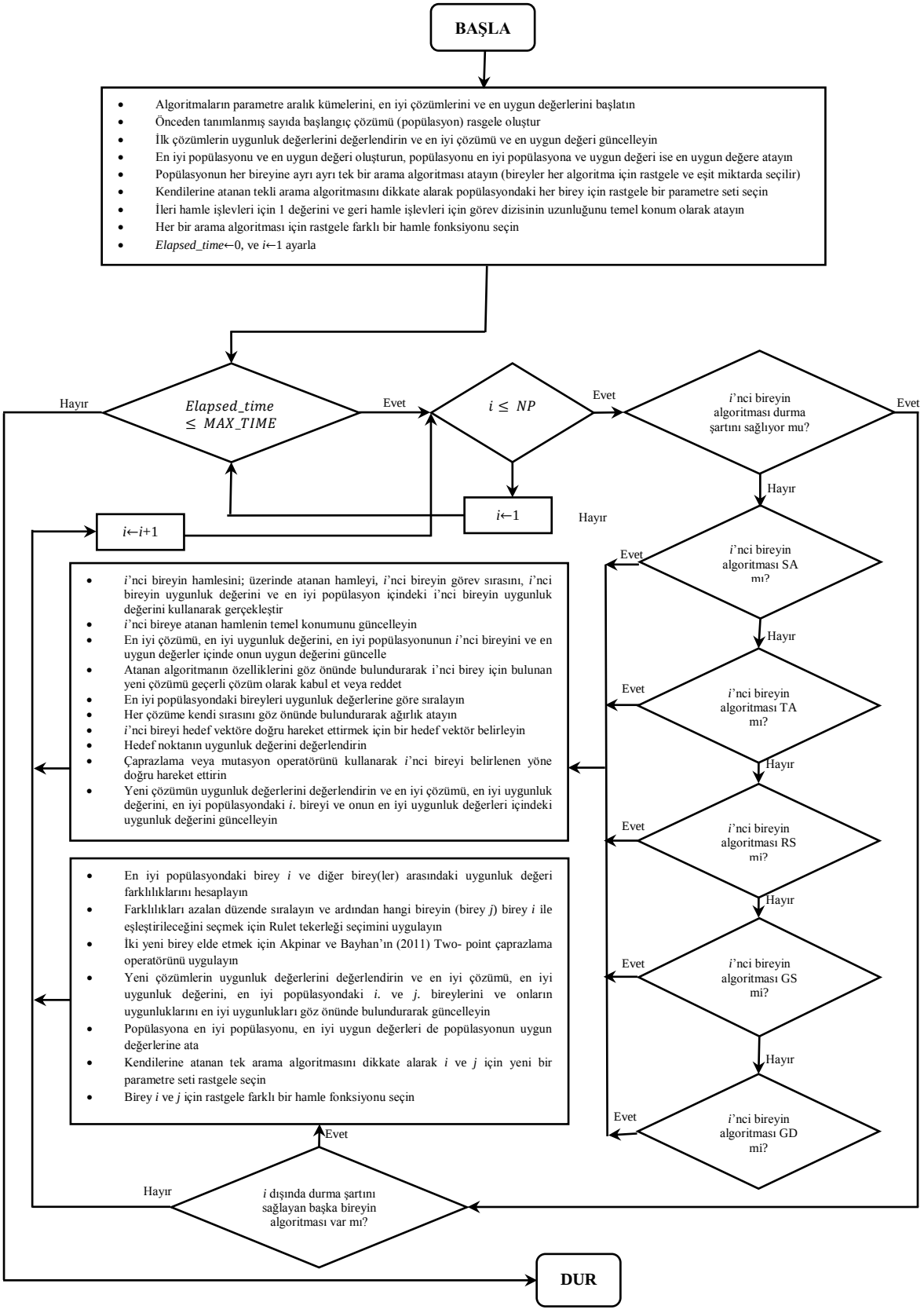
Çizelge 3.1. Algoritmaların Parametreleri

Algoritma	Parametreler	Açıklama
Great Deluge	*GD_ITERATION_MIN*	Maksimum dış iterasyon sayısının daha düşük seviyesi
	GD_ITERATION_MAX	Maksimum dış iterasyon sayısının üst seviyesi
	GD_INNER_MIN	Her seviyede deneme sayısının daha düşük seviyesi
	GD_INNER_MAX	Her bir seviyedeki deneme sayısının üst seviyesi
Threshold Accepting	*TA_MAIN_MIN*	Maksimum dış iterasyon sayısının daha düşük seviyesi
	TA_MAIN_MAX	Maksimum dış iterasyon sayısının üst seviyesi
	TA_INNER_MIN	Her eşik değerindeki deneme sayısının daha düşük seviyesi
	TA_INNER_MAX	Her bir eşik değerindeki deneme sayısının üst seviyesi
	THRESHOLDF_MIN	Başlangıç eşik değerinin daha düşük seviyesi
	THRESHOLDF_MAX	Başlangıç eşik değerinin üst seviyesi
Greedy Search	*GREEEDY_MIN*	Maksimum yineleme sayısının daha düşük seviyesi
	GREEEDY_MAX	Maksimum yineleme sayısının üst seviyesi
Random Search	*RANDOM_MIN*	Maksimum yineleme sayısının daha düşük seviyesi
	RANDOM_MAX	Maksimum yineleme sayısının üst seviyesi
Simulated Annealing	*INIT_TEMPT_MIN*	Başlangıç sıcaklığının düşük seviyesi
	INIT_TEMPT_MAX	Başlangıç sıcaklığının üst seviyesi
	SA_INNER_MIN	Maksimum iç iterasyon sayısının daha düşük seviyesi
	SA_INNER_MAX	Maksimum iç iterasyon sayısının üst seviyesi
	QUENCH_MIN	Düşük sıcaklık sönümleme oranı
	QUENCH_MAX	Sıcaklık sönüm oranının üst seviyesi

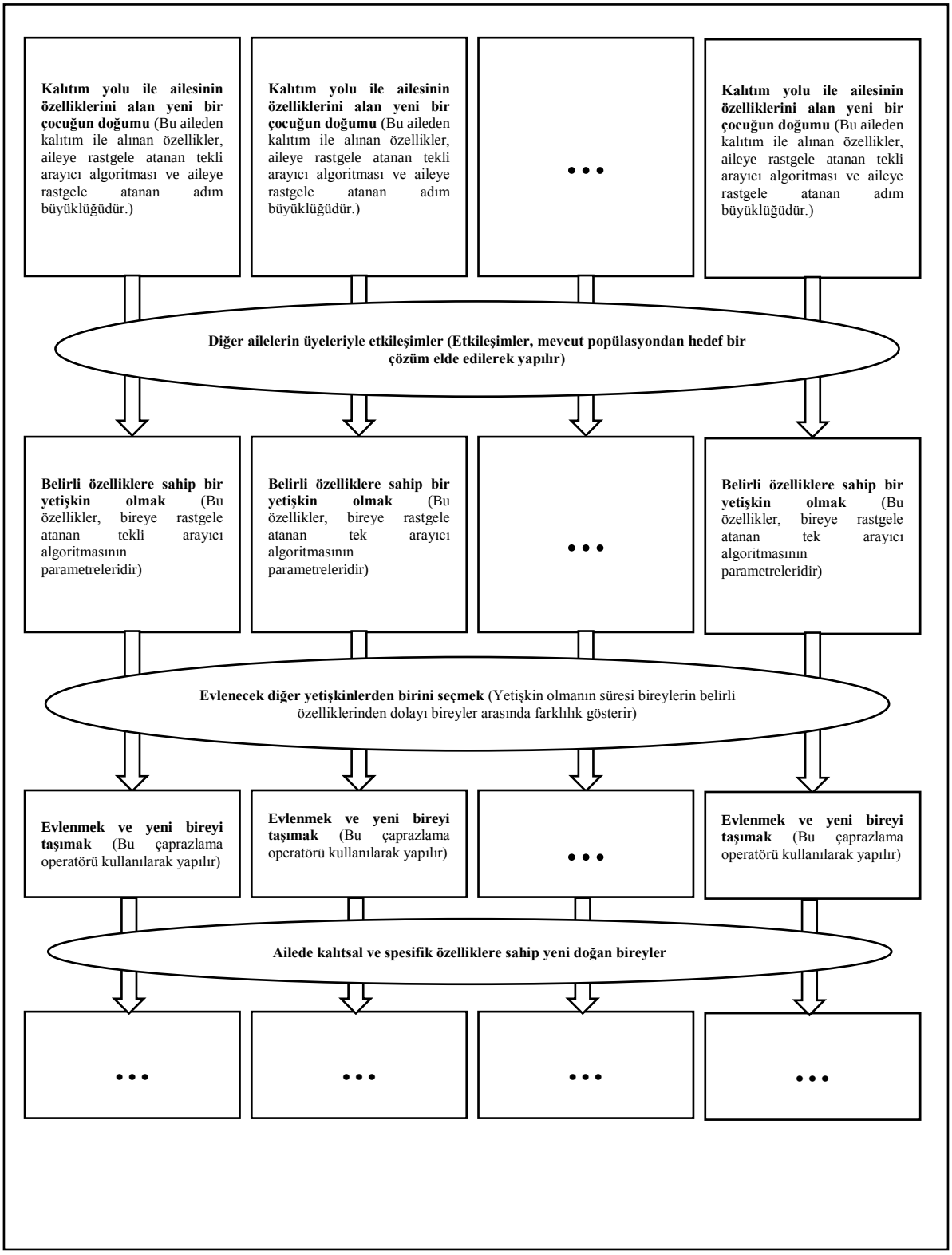
T_MIN	Son sıcaklık

Çizelge 3.2. Geliştirilen algoritmada kullanılan diğer parametreler, terimler ve değişkenler

Terimler & değişkenler	Açıklama
NA	Arama için kullanılan algoritma sayısı
NP	Popülasyondaki birey sayısı
MAX_TIME	Algoritma için verilen çalışma süresinin sınırı
τ	Kullanıcı tanımlı parametre
Population(.,.)	Başlangıç popülasyonu
F(.)	Başlangıç popülasyonun çözüm değerlerini saklayan matris
Best_Population(.,.)	En popülasyon
BF(.)	En iyi popülasyonun çözüm değerlerini saklayan matris
BestSolution(.)	Bulunan en iyi çözüm
BestFitness	En iyi çözümün çözüm değeri
character_matrix	NP satırları ve 6 sütundan oluşur. İlk sütunlar her NP'ye atanmış algoritmaları tutar, son sütunlar her NP'ye atanan hamleleri tutar ve diğer sütunlar ilgili algoritmalara atanan parametreleri tutar
move_position_matrix	24 hamlenin pozisyon durumunu tutar
seq_lenght	Veri setindeki toplam iş sayısı
OPTIMUM	Problemin optimum veya bilinen en iyi çözüm değeri


BAŞLA
Algoritmaların parametre aralık kümelerini, en iyi çözümlerini ve en uygun değerlerini başlatın
Önceden tanımlanmış sayıda başlangıç çözümü (popülasyon) rasgele oluştur
İlk çözümlerin uygunluk değerlerini değerlendirin ve en iyi çözümü ve en uygun değeri güncelleyin
En iyi popülasyonu ve en uygun değeri oluşturun, popülasyonu en iyi popülasyona ve uygun değeri ise en uygun değere atayın
Popülasyonun her bireyine ayrı ayrı tek bir arama algoritması atayın (bireyler her algoritma için rastgele ve eşit miktarda seçilir)
Kendilerine atanan tekli arama algoritmasını dikkate alarak popülasyondaki her birey için rastgele bir parametre seti seçin
İleri hamle işlevleri için 1 değerini ve geri hamle işlevleri için görev dizisinin uzunluğunu temel konum olarak atayın
Her bir arama algoritması için rastgele farklı bir hamle fonksiyonu seçin
Elapsed_time←0, ve i←1 ayarla
Elapsed_time ≤ MAX_TIME
Hayır
Evet
i ≤ NP
Evet
i'nci bireyin algoritması durma şartını sağlıyor mu?
Evet
Hayır
i←1
Hayır
i←i+1
i'nci bireyin algoritması SA mı?
Evet
Hayır
i'nci bireyin algoritması TA mı?
Evet
Hayır
i'nci bireyin algoritması RS mi?
Evet
Hayır
i'nci bireyin algoritması GS mi?
Evet
Hayır
i'nci bireyin algoritması GD mi?
Evet
i'nci bireyin hamlesini; üzerinde atanan hamleyi, i'nci bireyin görev sırasını, i'nci bireyin uygunluk değerini ve en iyi popülasyon içindeki i'nci bireyin uygunluk değerini kullanarak gerçekleştir
i'nci bireye atanan hamlenin temel konumunu güncelleyin
En iyi çözümü, en iyi uygunluk değerini, en iyi popülasyonunun i'nci bireyini ve en uygun değerler içinde onun uygun değerini güncelle
Atanan algoritmanın özelliklerini göz önünde bulundurarak i'nci birey için bulunan yeni çözümü geçerli çözüm olarak kabul et veya reddet
En iyi popülasyondaki bireyleri uygunluk değerlerine göre sıralayın
Her çözüme kendi sırasını göz önünde bulundurarak ağırlık atayın
i'nci bireyi hedef vektöre doğru hareket ettirmek için bir hedef vektör belirleyin
Hedef noktanın uygunluk değerini değerlendirin
Çaprazlama veya mutasyon operatörünü kullanarak i'nci bireyi belirlenen yöne doğru hareket ettirin
Yeni çözümün uygunluk değerlerini değerlendirin ve en iyi çözümü, en iyi uygunluk değerini, en iyi popülasyondaki i. bireyi ve onun en iyi uygunluk değerleri içindeki uygunluk değerini güncelleyin
En iyi popülasyondaki birey i ve diğer birey(ler) arasındaki uygunluk değeri farklılıklarını hesaplayın
Farklılıkları azalan düzende sıralayın ve ardından hangi bireyin (birey j) birey i ile eşleştirileceğini seçmek için Rulet tekerleği seçimini uygulayın
İki yeni birey elde etmek için Akpinar ve Bayhan'ın (2011) Two- point çaprazlama operatörünü uygulayın
Yeni çözümlerin uygunluk değerlerini değerlendirin ve en iyi çözümü, en iyi uygunluk değerini, en iyi popülasyondaki i. ve j. bireylerini ve onların uygunluklarını en iyi uygunlukları göz önünde bulundurarak güncelleyin
Popülasyona en iyi popülasyonu, en iyi uygun değerleri de popülasyonun uygun değerlerine ata
Kendilerine atanan tek arama algoritmasını dikkate alarak i ve j için yeni bir parametre seti rastgele seçin
Birey i ve j için rastgele farklı bir hamle fonksiyonu seçin
Evet
i dışında durma şartını sağlayan başka bireyin algoritması var mı?
Hayır
DUR

Şekil 3.1. SSS Algoritmasının temel adımları.



Şekil 3.2. SSS algoritmasının genel çerçevesi.

SSS algoritmasının temel adımları

1. Algoritma parametrelerini tanımla;
2. *algorithm_start_time* = Bilgisayarın zamanını al;
3. *Population (.,.)* Önceden tanımlanmış bir dizi ilk çözümü (NP) rastgele oluşturun;
4. *F* (.) = İlk çözümlerin uygunluk değerlerini değerlendirin;
5. *Best_Population* (:, :) = *Population* (:, :) ;
6. *BF* (:) = *F* (:);
7. *character_matrix* = zeros (*NP*, 6);
8. *counter* = 0;
9. for *i* = 1: *NA*
10. for *j* = 1: (*NP* / *NA*)
11. *character_matrix* (*counter*, 1) = *i*; // Popülasyondaki her birey için bir algoritma atayın
12. *counter* = *counter*+1;
13. endfor
14. endfor
15. for *i* = 1: *NP*
16. Call Function CHAR_ASSIGN (*i*, Algoritmaların parametreleri); // *i* bireyi için karakter atama
17. endfor
18. *move_position_matrix* = zeros (24, 1); //24, önceden tanımlanmış hamle sayısını temsil eder
19. for *i* = 1: 24 // önceden tanımlanmış hamle sayısı
20. if // *i* hamleyii ileri doğru bir hamle ise
21. *move_position_matrix* (*i*, 1) = 1;
22. else // geriye doğru hamle ise
23. *move_position_matrix* (*i*, 1) = *seq_lenght*;
24. endif
25. endfor

// Her bir kişiye rastgele farklı bir hamle atayın (character_matrix'in son sütunu her bir kişi için hamle bilgilerini tutar)

26. for *i* = 1: *NA*
27. *character_matrix* (*i*, 6) = diğer kişilere atanmamış hamlelerden birini seçin
28. endfor
29. *crossover_decision* = zeros (*NP*, 1);
30. *parameter_count* = zeros (*NP*, 2);
31. While (1) //loop forever
32. for *i* = 1: *NP*
33. if *crossover_decision* (*i*, 1) == 0
34. *chromosome* (:) = *Population* (*i*, :);
35. *ALGORITHM* = *character_matrix* (*i*, 1);
36. switch *ALGORITHM*
37. case {1} *run great deluge algorithm ()*;
38. case {2} *run threshold accepting algorithm ()*;
39. case {3} *run greedy search algorithm ()*;
40. case {4} *run random search algorithm ()*;
41. case {5} *run simulated annealing algorithm ()*;
42. endswitch
43. if *f_value* < *BF* (*i*)
44. *BF* (*i*) = *f_value*;
45. *Best_Population* (*i*, :) = *new_chromosome* (:);
46. endif
47. [*target_vector*] = TARGET (*Best_Population* (:, :), *BF* (:), τ);
48. [*f_target*] = calc_fitness (*target_vector* (1, :));
49. if *f_target* < *BF* (*i*)

```
                                        BF (i) = f_target;
                                        Best_Population (i, :) = target_vector (1, :);
                                endif
                                if f_value > f_target
                                        [chr_n] = CROSSVER (new_chromosome, target_vector);
                                        [obj_n] = calc_fitness (chr_n);
                                else
                                        if rand () < exp (f_value - f_aggre)
                                                [chr_n] = CROSSVER (new_chromosome, target_vector);
                                                [obj_n] = calc_fitness (chr_n);
                                        endif
                                endif
                                Population (i, :) = chr_n (:);
                                F (i) = obj_n;
                                if obj_n < BF (i)
                                        BF (i) = obj_n;
                                        Best_Population (i, :) = chr_n (:);
                                endif
                                current_time = Take the current computer time;
                                if (current_time - algorithm_start_time) >= MAX_TIME
                                        break; // ALGORİTMAYI DURDUR
                                endif
                        endif
                endfor
                current_time = Bilgisayarın zamanını alma;
                if (current_time - algorithm_start_time) >= MAX_TIME
                        break; // ALGORİTMAYI DURDUR
                endif
                for i = 1: NP
                        if crossover_decision (i, 1) == 1 && crossover_decision değerine eşit olan herhangi bir
                           birey var mı (i hariç)
                          distance_list (:, 1) =; i bireyi ve crossover_decision değeri 1'e eşit olan diğer bireyler
                          arasındaki uygunluk değerlerinin farklılıklarını hesapla;
                          Sort distance_list (:, 1) azalan sırayla; // i bireyinden daha uzakta olan bireyin
                          seçilme şansı daha fazladır
                          sel_individual = Hangi bireyin i bireyi ile eşleştirileceğini belirlemek için
                          distance_list içindeki bireylere Rulet Tekerleği Seçimi uygulayın;
                          [child1, child2] = CROSSVER (individual sel_individual, individual i) i bireyi ve
                          sel_individual bireylerinden iki yeni birey elde etmek için İki noktalı geçiş operatörü
                          uygulayın;
                          crossover_decision (i, 1) = 0;
                           crossover_decision (sel_individual, 1) = 0;
                           f_value = child1'in uygunluk değerini hesapla;
                           Population (i, :)= child1 (:);
                           F (i) = f_value;
                                if f_value < BF (i)
                                        BF (i) = f_value;
                                        Best_Population (i, :) = child1 (:);
                                endif
                                f_value = child2' nin uygunluk değerini hesapla;
                                Population (sel_individual, :) = child2 (:);
                                F (sel_individual) = f_value;
                                if f_value < BF (sel_individual)
                                        BF (sel_individual) = f_value;
                                        Best_Population (sel_individual, :) = child1 (:);
                                endif
                        current_time = Bilgisayarın zamanını al;
                        if (current_time - algorithm_start_time) >= MAX_TIME
                                break; // ALGORİTMAYI DURDUR
                        endif
                        Call Function CHAR_ASSIGN (i, Algoritmaların Paremetreleri); // i bireyi için yeni
                        karakter ataması
```

```
105.                    Call Function CHAR_ASSIGN (sel_individual, Algoritmaların Parametreleri); // sel_individual bireyi için yeni karakter ataması
106.                    character_matrix (i, 6) = diğer kişilere atanmamış hamlelerden birini seçin // yeni hamle atama
107.                    character_matrix (sel_individual, 6) = diğer kişilere atanmamış hamlelerden birini seçin
108.            endif
109.    endfor
110.    Population (:, :) = Best_Population (:, :);
111.    F (:) = BF (:)
112.    current_time = Bilgisayarın zamanını al;
113.    if (current_time - algorithm_start_time) >= MAX_TIME
114.            break; // ALGORİTMAYI DURDUR
115.    endif
116. endwhile
117. BestSolution and BestFitness değerlerini bul ve onları raporla
```

3.2.2. Hedef noktası belirleme prosedürü

Orijinal SSS algoritması, yapay etkenlerin (artificial agent) arama yönlerini hedef nokta belirleme prosedürü ile belirler. Birleştirici özelliği olan SSS algoritması bir arama yönü belirleme prosedürü gerektirmemesine rağmen, algoritma, bu çalışma kapsamında benzer bir amaç için hedef noktası belirleme prosedürünü kullanır. Yani, birleştirici SSS algoritması her yinelemesi için bir hedef çözüm belirler ve bu çözümü ilgili tek çözüm üzerinden arama yapan meta-sezgiselin (SSBM) en iyi çözümüyle karşılaştırır. Hedef nokta ilgili SSBM'nin en iyi çözümünden daha iyi ise, algoritma ilgili en iyi çözümü hedef nokta ile değiştirir. Kombinatoryal SSS algoritması, hedef nokta mevcut çözümden daha iyi ise, hedef noktayı ve SSBM'nin mevcut çözümünü aşarak yeni bir çözüm üretir. Bu arada, kombinatoryal SSS algoritması, tüm algoritmanın aramasını bu bilgilerin rehberliğinde yönlendirmek için farklı prosedürler tarafından üretilen bilgileri paylaşır. Kombinatoryal SSS algoritması tarafından gerçekleştirilen hedef nokta belirleme prosedürünün özetlenmiş kodu aşağıda verilmiştir. Daha fazla detay için bakınız: Baykasoğlu ve Akpinar (2017), Onwubolu ve Davendra (2006).

```
function TARGET (c_Population, CF, τ)
// Uygunluk değerlerine göre çözümleri sıralayın
1. for k = 1: NP
2.              ek = CF (k);
3.              i = k;
4.              for j = k + 1: NP
5.                      if (CF (j) < ek)
6.                              ek = CF (j);
7.                              i = j;
```

```
8.                          endif
9.                  endfor
10.         temp = zeros (1, seq_lenght);
11.         for j = 1: seq_lenght
12.                 temp (1, j) = c_Population (k, j);
13.                 c_Population (k, j) = c_Population (i, j);
14.                 c_Population (i, j) = temp (1, j);
15.         endfor
16.         CF (i) = CF (k);
17.         CF (k) = ek;
18. endfor
// seq_lenght uzunluğunda bir olasılık vektörü oluşturun ve rastgele olarak bu vektördeki her bir hücreye 0 ile 1 arasında bir sayı atayın
19. aggre_prob_vector = zeros (1, seq_lenght);
20. for i =1: seq_lenght
21.                 aggre_prob_vector (1,i) = unifrnd (0,1);
22. endfor
// Her çözüme kendi sıralarını göz önünde bulundurarak bir ağırlık atayın
23. weight = zeros (1, NP);
24. for i = 1: NP
25.                 weight (1, i) = i^((−1)∗τ) ;
26. endfor
// Ağırlıklar ve olasılık vektörünü kullanarak hedef vektördeki her hücre için bir iş (görev) belirleme
target_vector = zeros (1, seq_lenght);
27. for loc = 1: seq_lenght
28.         for i =1: NP
29.                 if weight (1,i) >= aggre_prob_vector (1, loc)
30.                 task = c_Population (i, loc);
31.                         if // Eğer task daha önce hedef vektöre atanmamışsa
32.                                 task'ı aday görev listesine ekle
33.                         endif
34.                 endif
35.         endfor
36.         if // aday görev listesinde herhangi bir görev var mı?
37.                 Aday görev listesinde her görevin kaç kez tekrarlandığını bulun
38.                 Rulet tekerleği seçimini ve aday görev listesindeki yineleme numaralarını kullanarak konum
                    bulma için görevlerden birini seçin
39.                 target_vector (1, loc) = Seçilen görevi loc konumuna ekle
40.         else
41.                 Daha önce hedef vektöre atanmamış görevlerden birini rastgele seçin
42.                 target_vector (1, loc) = Seçilen görevi loc konumuna ekle
43.         endif
44. endfor
45. return target_vector
```

3.2.3. Hamle Fonksiyonları

Hamle 1: Bu hamlenin özetlenmiş kodu aşağıda verilmiştir.

```
function MOVE1 (chr_c, obj_c, best_ind_fitness, move_location) // ileri doğru değişim
1. obj_n = obj_c;
2. for i = 1: seq_lenght // Görev dizisindeki hücre sayısı
3.         if i != move_location
4.         base (:) = chr_c (:);
5.         dummy = base (move_location);
6.         base (move_location) = base (i);
7.         base (i) = dummy;
8.         [fitness_value] = calc_fitness (base (:));
```

9. if *obj_n* > *fitness_value*
10. *obj_n* = *fitness_value*; 11.*chr_n (:)* = *base (:)*;
12. endif
13. if *best_ind_fitness* > *obj_n*
14. break;
15. endif
16. endif
17. endfor
18. if *best_ind_fitness* > *obj_n* 19.*move_location* = 1;
20. else 21.*move_location* = *move_location* + 1;
22. endif
23. if *move_location* > *seq_lenght* 24.*move_location* = 1;
25. endif
26. return *obj_n*, *chr_n (:)* and *move_location*

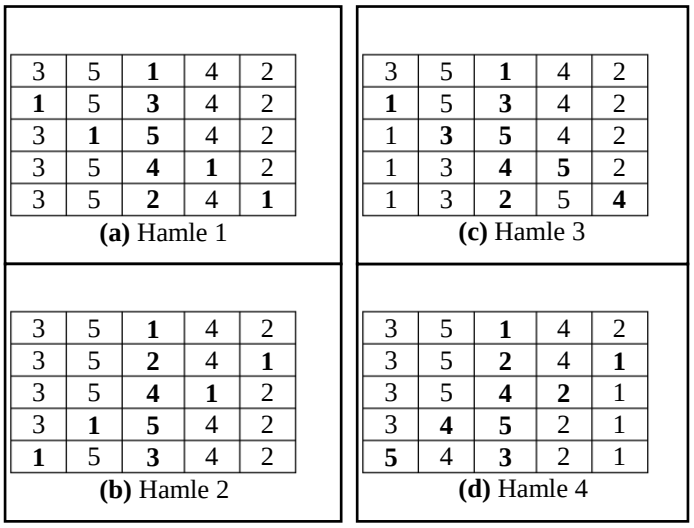
Bu hamle ikili yer değiştirme (swap) operatörünü temel alır ve deterministik ileri ikili yer değiştirme olarak adlandırılır. Görevin başlangıcından sonuna kadar (1'den sona kadar), taban konumunda bulunan (base position) görev, (taban konum, özetlenmiş koddaki *move_location* ile gösterilir) tabanın içeriğini ve sonraki konumları değiştirerek kendi konumu dışındaki diğer tüm konumlarda test edilir. Hamle işlemini gerçekleştiren tekli arama algoritması, en iyi popülasyonlardaki (*best_ind_fitness*) ilgili çözümlerden daha iyi bir çözüm bulursa, hamle içindeki arama sonlandırılır. Aksi takdirde, hamle, görev dizisindeki son konum değerlendirilene kadar yeni çözümleri değerlendirmeye devam eder. Hamle içinde bulunan çözümün kalitesine göre (*bf* ile gösterilir), taban konumu yeniden konumlandırılır (Özet kodda 18 ve 26 arasındaki çizgilere bakın). Hamle, en iyi popülasyonlardaki (*best_ind_fitness*) ilgili çözümlerden daha iyi bir çözüm bulursa, taban konum, görev sırasının ilk konumuna yerleştirilir. Aksi takdirde, görev dizisindeki konumu bir arttırılır.

Hamle 2**:** Bu hamle aynı zamanda ikili yer değiştirme operatörünü temel alır ve deterministik geriye doğru ikili yer değiştirme olarak adlandırılır. Hamle 1 öğesinden farklı olarak, bu hamle, aramayı görevin son konumundan ilk sıraya doğru başlatır. Hamle 1'e benzer şekilde, en iyi popülasyonlardaki (*best_ind_fitness*) ilgili çözümlerden daha iyi bir çözüm olduğunda, hamle arama işlemi sonlandırılır. Aksi takdirde, hamle, görev dizisindeki ilk konum değerlendirilene kadar yeni çözümleri değerlendirmeye devam eder. Hamle, en iyi popülasyonlardaki ilgili çözümlerden daha iyi bir çözüm bulursa, taban konum, görev dizisinin son konumuna yerleştirilir. Aksi takdirde, görev

dizisindeki konumu bir azalır. Bu hamle ve hamle 1 arasındaki farklar aşağıdaki özet kodda gösterilmiştir.

```
function MOVE2 (chr_c, obj_c, best_ind_fitness, move_location) // geriye doğru ikili yer değiştirme
1. obj_n = obj_c;
2. for i = seq_lenght : -1: 1
…
17. endfor
18. if best_ind_fitness > bf 19.move_location = seq_lenght;
20. else 21.move_location = move_location - 1;
22. endif
23. if move_location == 0 24.move_location = seq_lenght;
25. endif
26. return obj_n, chr_n (:) and move_location
```

Hamle 3 ve Hamle 4: Eğer yeni bir hamle, içinde hamle 1 ve hamle 2 olan hamleden sonra gerçekleşirse, önceki hamleler dikkate alınmaz. Yani, sonraki hamleler sadece önceden verilen görev sırasına (*chr_c*) göre belirlenir ve taban konum ile sonraki konum arasındaki içerik her seferinde önceden verilen görev sırasına göre değiştirilir. Hamle 3 ve hamle 4, önceden belirlenen görev sırasına göre yapılmış hamleleri kaydeder. Böylece, bu hamleler önceki hamleleri dikkate alarak yeni hamleler yapar. Bu fark dışında, sırasıyla hamle 3 ve hamle 4 sırasıyla hamle 1 ve hamle 2'nin karşılığıdır. Hamle 1, 2, 3 ve 4'teki tüm olası hamlelere bir örnek Şekil 3.3.'te verilmiştir. Bu örnekte, görev dizisinin üçüncü yeri taban pozisyon olarak kabul edilir ve hamlelerdeki ilk satırlar, önceden verilen görev dizilerini temsil eder.



Şekil 3.3. Hamle 1, 2, 3 ve 4'teki tüm hamle kombinasyonlarına bir örnek.

Hamle 5, 6, 7 ve 8: Hamle 1-4’de kullanılan ikili yer değiştirme operatörlerinden farklı olarak hamle 5-8’de ekleme operatörünü kullanır. Bu farkın dışında, hamle 1, 2, 3 ve 4 sırasıyla hamle 5, 6, 7 ve 8’in karşılığıdır. Hamle 5, 6, 7 ve 8'deki tüm hamle kombinasyonlarına bir örnek Şekil 3.4.'te verilmiştir.

3	5	**1**	4	2	Yerleştirildikten sonra ileri kaydırılmış hücreler					Oluşturulan yeni çözümler				
1						3	5	4	2	**1**	3	5	4	2
	1				3		5	4	2	3	**1**	5	4	2
			1		2	3	5		4	2	3	5	**1**	4
				1	2	3	5	4		2	3	5	4	**1**

(a)

Yerleştirildikten sonra geri kaydırılmış hücreler					3	5	**1**	4	2	Oluşturulan yeni çözümler				
3	5	4	2						**1**	3	5	4	2	**1**
3	5	4		2				**1**		3	5	4	**1**	2
5		4	2	3		**1**				5	**1**	4	2	3
	2	4	5	3	**1**					**1**	5	4	2	3

(b)

3	5	**1**	4	2	Yerleştirildikten sonra ileri kaydırılmış hücreler					Oluşturulan yeni çözümler				
1						3	5	4	2	1	3	**5**	4	2
	5				1		3	4	2	1	5	**3**	4	2
			3		2	1	5		4	2	1	**5**	3	4
				5						4	2	1	3	5

(c)

Yerleştirildikten sonra geri kaydırılmış hücreler					3	5	**1**	4	2	Oluşturulan yeni çözümler				
3	5	4	2						**1**	3	5	**4**	2	1
3	5	2		1				**4**		3	5	**2**	4	1
5		4	1	3		**2**				5	2	**4**	1	3
	2	1	3	5	**4**					4	2	1	3	5

(d)

Şekil 3.4. Hamle 5, 6, 7 ve 8'deki tüm hamle kombinasyonlarına bir örnek.

Hamle 9 ve 10: Hamle 9 ve 10 ikili yer değiştirme operatörünü temel alır ve sırasıyla hamle 3 ve 4'ün stokastik versiyonlarıdır. Hamle 3 ve 4’teki tüm yeni çözümler, bir

sonraki hamleden önce mevcut çözümler olarak kabul edilmektedir. Bu hamlelerde, bu durum öncekilerin ve şu anki çözümlerin çözüm kalitesine bağlıdır. Bir çözüm, bir hamleden sonra mevcut çözümden daha iyiyse, geçerli görev dizisi olarak kabul edilir. Aksi takdirde, kabul oranına ve mevcut çözüm ile hamleden sonra elde edilen çözüm arasındaki göreceli sapmaya bağlı olarak yeni görev sırası, geçerli görev sırası olarak kabul edilir. Yani, kabul oranı yeni çözümün (hamle sonrası) görevdeki çözümün yerini alıp almayacağına karar verir. Daha kötü çözümlerin kabul koşulları aşağıda verilmektedir.

$$acceptace\_rate \geq \frac{\text{yeni çözümün uygunluk değeri} - \text{ mevcut çözümün uygunluk değeri}}{\text{mevcut çözümün uygunluk değeri}}$$

Bu çalışmada kabul oranı 0,1 olarak alınmış ve hamle 3 ile hamle 9 arasındaki farklar aşağıdaki özetlenmiş kodda gösterilmiştir.

```
function MOVE9 (chr_c, obj_c, best_ind_fitness, move_location) // ileri ikili yer değiştirme
1. for i = 1: seq_lenght
2.                 if i != move_location
3.        base (:) = chr_c (:); 4.dummy = base (move_location); 5.base (move_location) = base (i); 6.base (i) =
dummy; 7.[fitness_value] = calc_fitness (base (:));
8.                         if obj_c > fitness_value
9.                                 obj_c = fitness_value; 10.chr_c (:) = base (:);
11.                        else
12.                        if  acceptance_rate>= (fitness_value - obj_c) / obj_c
13.                                obj_c = fitness_value; 14.chr_c (:) = base (:);
15.                        endif
16.                endif
17.                if obj_c < best_ind_fitness
18.                        break;
19.                endif
20.       endif
21. endfor
22. if best_ind_fitness > obj_c 23.move_location = 1;
24. else 25.move_location = move_location + 1;
26. endif
27. if move_location > seq_lenght 28.move_location = 1;
29. endif
30. obj_n = obj_c; 31. chr_n (:) = chr_c (:);
32. return obj_n, chr_n (:) and move_location
```

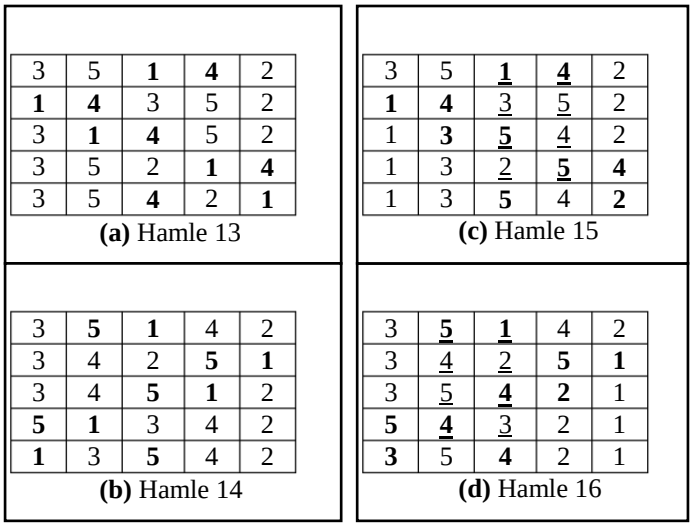
Hamle 11 ve 12: Hamle 11 ve 12 ekleme operatörünü ve sırasıyla hamle 7 ve 8'in stokastik versiyonlarını temel alır.

Hamle 13: Bu hamle, hamle 1'in blok versiyonudur. Taban pozisyonuna (*move_location*) dayanarak, görev dizisinin taban pozisyonunu ve ileri yön pozisyonlarını kullanarak rastgele oluşur ve bloke olur, taban pozisyon dizisindeki son görev sırasına ulaşana kadar hamleler yapar. Blok uzunluğu, taban konumu da dahil olmak üzere en az iki ve en fazla üç konumda bozulmayacak şekilde oluşturulur. Bu hamlenin özet kodu aşağıda verilmiştir.

```
function MOVE13 (chr_c, obj_c, best_ind_fitness, move_location) // ileri blok ikili yer değiştirme
1. obj_n = obj_c; 2. block_start_point = move_location;
3.          if (seq_lenght - move_location) > 2
4.          rnd_nbr = round(unifrnd (1, 2 )); 5.block_end_point = move_location + rnd_nbr; 6.block_lenght = 1+
            rnd_nbr;
7.          else
8.          block_end_point = move_location +1; 9.block_lenght = 2;
10.          endif
11. block_list = zeros(1, block_lenght);
12. for loc = 1: seq_lenght
13.          if loc != move_location
14.                    base(:)=chr_c (:); 15.counter = 1;
16.                    for i= block_start_point: block_end_point
17.                              block_list (1,counter) = base (1,i); 18.counter = counter + 1; 19.dummy(1,i) = 0;
20.                    endfor
21.                    rest_list = zeros (1, seq_lenght - block_lenght); 22.inside_end = loc + (block_lenght - 1); 23.
                       bcounter = 1; 24.rcounter =1;
25.                    for i = loc: inside_end
26.                              if i <= seq_lenght
27.                                        if base (1,i) ~ = 0
28.                                        rest_list (1,rcounter) = base (1,i); 29.rcounter = rcounter + 1; 30.base
                                           (1,i)=block_list (1,bcounter); 31.block_list (1,bcounter)=0;
                                           32.bcounter=bcounter+1;
33.                                        else
34.                                        base (1,i)=block_list (1,bcounter); 35.block_list (1,bcounter)=0;
                                           36.bcounter=bcounter+1;
37.                                        endif
38.                              else
39.                                        for m=1: block_lenght
40.                                                  if block_list (1,m)~=0
41.                                                            for k=1: seq_lenght
42.                                                                      if base (1,k) == 0
43.                                                                      base (1,k) = block_list (1,m); 44.
                                                                         block_list (1,m)=0;
45.                                                                      endif
46.                                                            endfor
47.                                                  endif
48.                                        endfor
49.                              endif
50.                    endfor
51.                    for i =1: rcounter-1
52.                              for k =1: seq_lenght
53.                                        if base (1,k)==0
54.                                                  base (1,k)= rest_list (1,i); 55.break;
56.                                        endif
57.                              endfor
58.                    endfor
59.                    [fitness_value] = calc_fitness (base (:));
60.                    if obj_n > fitness_value
```

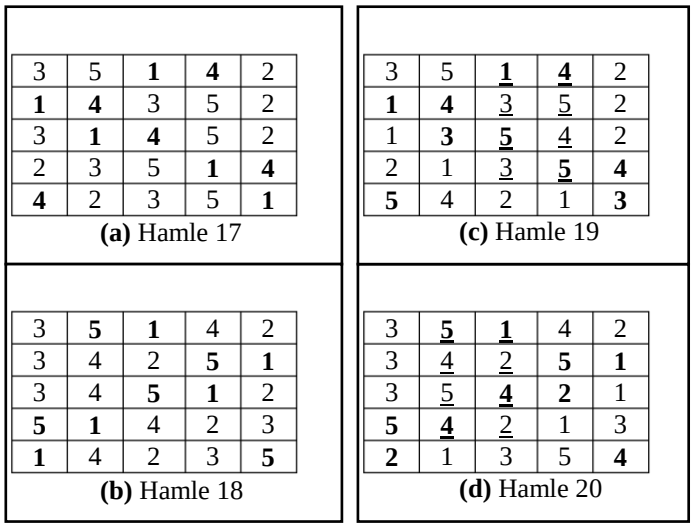
61. *obj_n= fitness_value*; 62.*chr_n (:) = base (:)*;
63. endif
64. if *best_ind_fitness* > *obj_n* 65.break;
66. endif
67. endif
68. endfor
69. if *best_ind_fitness* > *obj_n* 70.*move_location* = 1;
71. else 72.*move_location* = *move_location* + 1;
73. endif
74. if *move_location* == *seq_lenght* 75.*move_location* = 1;
76. endif
77. return *obj_n*, *chr_n (:)* and *move_location*

Hamle 14, Hamle 15 ve Hamle 16: Hamle 1 ve 13 arasındaki benzer ilişkide olduğu gibi, hamle 14, 15 ve 16 sırasıyla hamle 2, 3 ve 4 ile benzer ilişkilere sahiptir. Hamle 13, 14, 15 ve 16'daki tüm hamle kombinasyonlarını içeren bir örnek Şekil 3.5'te verilmiştir. Bu örnekte, görev dizisinin üçüncü konumunun temel konum olduğu ve bloğun uzunluğunun 2 olduğu varsayılır. Hamlelerdeki ilk satırlar, önceden verilen görev dizilerini temsil eder.



Şekil 3.5. Hamle 13, 14, 15 ve 16'daki tüm hamle kombinasyonlarına bir örnek.

Hamle 17, Hamle 18, Hamle 19, Hamle 20: Bu hamleler sırasıyla hamle 5, 6, 7 ve 8'in blok versiyonudur (Ekleme operatörünü kullanırlar). Hamle 17, 18, 19 ve 20'deki tüm hamle kombinasyonlarını içeren bir örnek Şekil 3.6'da verilmiştir.



Şekil 3.6. Hamle 17, 18, 19 ve 20'deki tüm hamle kombinasyonlarına bir örnek.

Hamle 21 ve Hamle 22: Hamle 3 ve hamle 9 ile hamle 4 ve 10 arasındaki ilişki gibi, hamle 21 ve 22 sırasıyla hamle 15 ve 16'nın stokastik blok versiyonlarıdır. Hepsi ikili yer değiştirme operatörünü temel alır.

Hamle 23 ve Hamle 24: Hamle 7 ve hamle 11 ile hamle 8 ve 12 arasındaki ilişki gibi, hamle 23 ve 24 sırasıyla hamle 19 ve 20'nin stokastik blok versiyonlarıdır. Hepsi ekleme operatörünü temel alır.

3.2.4. Parametre ayarlaması

Bu bölüm, yukarıda tanımlanan kombinatoryal optimizasyon problemi üzerindeki SSS algoritmasının performansını değerlendirmek için detaylı bir çalışma sunmaktadır.

Kombinatoryal SSS algoritması çeşitli algoritmaların bir kombinasyonu olduğu için ve ayarlanması gereken birçok parametrenin bulunması nedeniyle; SSS algoritması için kesin bir parametre optimizasyonu gerçekleştirmek çok zordur. SSBM'ler aramalarını bitirip çaprazlama operatörüne tabi tutulduktan sonra, rastgele atanan yeni parametre kümeleri ile çözüm alanını keşfetmeye devam ederler. Önemli olan, kombinatoryal SSS algoritması ilerlerken mümkün olduğunca farklı bireylerin çaprazlama operatörüne tabi tutulmasıdır. Her SSBM'nin parametresi belirli bir aralıktan rastgele seçildiğinden, çaprazlamaya tabi tutulacak iki bireyin her seferinde farklı olması büyük olasılıktır. Bir birey vaktinden önce (prematüre) çalışmasını tamamlasa bile, çaprazlama operatörünü kullanarak daha iyi bir çözüm elde etmek mümkündür çünkü eşleşeceği diğer bireyin çözüm kalitesi olasılığı yüksek olabilir. Öte yandan, her bir tekli arayan arama sırasında değişen birden fazla komşu yapısı kullanır. Bu anlamda SSBM'lerin erkenden yakınsaması (converge etmesi) önemli bir sorun değildir. Daha önce belirtildiği gibi, önerilen algoritmada önemli olan, algoritma ilerlerken ve tekli arayanlar da sürekli olarak komşu yapılarını değiştirirken farklı bireylerin birbirleriyle eşleşerek yeni bireyler oluşturmasıdır. Bu özellik itibariyle, aslında, önerilen algoritmaya uygun olarak yalnızca bir parametrenin (*MAX_ITERATION*) ayarlanması gerekir. Bir parametre kalibrasyonu gerçekleştirilirken, literatürdeki benzer kombinatoryal optimizasyon problemlerinin çözümünde kullanılan SSBM'lerin parametre setleri öncelikle incelenmiştir. Parametre aralığı kümelerinin ortalama değerleri, literatürdeki parametre değerlerine yakın bir şekilde ayarlanmıştır. Örneğin, SA algoritmasının başlangıç sıcaklığının üst sınırı (*INIT_TEMPT_MAX*) 22'ye, alt sınırı (*INIT_TEMPT_MIN*) 10'a ayarlanmıştır. Söndürme faktörü (Sıcaklık sönümleme oranı) Wang, C. ve ark. (2015)'de olduğu gibi kullanılmış ve minimum sıcaklık 1 olarak sabitlenmiştir. Bu parametre değerleri açısından, SA algoritması; D'nin SA'nın değerlendirdiği nokta sayısı olduğu $D = SA_{INNER} \times (1 + (\ln T_{MIN} - \ln INIT_{TEMPT}) / \ln QUENCH)$ denklemine göre en az 113.8110 (algoritmaya minimum parametre seviyeleri atanmışsa) ve en fazla 606.7602 (eğer

parametrelerin maksimum seviyeleri atanmışsa) farklı noktayı araştırır. Bu, üzerinde komşu yapısı bulunan herhangi bir SSBM'nin, SA kullanarak en az 113 ve en fazla 607 hamle yapacağı anlamına gelir. Diğer SSBM'lerin parametreleri de benzer bir mantıkla ve literatürdeki kullanımlarına uygun bir şekilde atanmıştır. Diğer SSBM'ler, kendilerine atanan her yeni komşu ile benzer şekilde en az 100 ve en fazla 600 hamle gerçekleştirebilmektedirler. Genel olarak, kombinatoryal SSS algoritması, bir parametre dışında ciddi bir parametre optimizasyonu gerektirmemektedir. Yapılan ön deneyler sonucunda *MAX_ITERATION* 25000 olarak ayarlandığında veri setleri için iyi çözümlerin elde edilebileceği görülmüştür. Yapılan ön deneyler sonucunda elde edilen ve kombinatoryal SSS algoritmasında kullanılan parametreler Çizelge 3.4'te verilmiştir.

Çizelge 3.3. Kontrol parametrelerinin sınırları ve değerleri

Parameter	Value	Parameter	Value
GD_ITERATION_MIN	50	*GREEEDY_MAX*	600
GD_ITERATION_MAX	75	*RANDOM_MIN*	100
GD_INNER_MIN	2	*RANDOM_MAX*	600
GD_INNER_MAX	8	*INIT_TEMPT_MIN*	10
TA_MAIN_MIN	50	*INIT_TEMPT_MAX*	22
TA_MAIN_MAX	75	*SA_INNER_MIN*	2
TA_INNER_MIN	2	*SA_INNER_MAX*	8
TA_INNER_MAX	8	*QUENCH*	0.96
THRESHOLDF_MIN	0.5	*T_MIN*	1
THRESHOLDF_MAX	0.99	*NA*	5
GREEEDY_MIN	100	*MAX_ITERATION*	25000

4. BULGULAR

Bu bölümde, genel olarak tarafımızdan geliştirilen matematiksel modeller ve meta-sezgisel algoritmanın performansının test edilmesi üzerine karşılaştırmalar yapılmıştır. İlk olarak Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi (DMPÇP) için şimdiye kadar geliştirilmiş en iyi matematiksel modelin ve bizim önerdiğimiz matematiksel modellerin; aynı yazılım ve aynı özellikte bilgisayar kullanılarak, 600 saniye kısıtı altında, Çizelge 4.1.'de yer alan küçük boyutlu örneklerin çözümlerinden elde edilen sonuçları karşılaştırılmıştır. Böylelikle önermiş olduğumuz matematiksel modellerin performansı test edilmiştir. Aynı şekilde sunmuş olduğumuz meta – sezgisel algoritmanın performansı ise Çizelge 4.1.'de yer alan küçük ve büyük boyutlu örnekler üzerinde yapılan testler sonucunda tespit edilmiştir. Elde ettiğimiz sonuçlar, söz konusu veri setleri için şimdiye kadar geliştirilen tüm sezgisel ve meta – sezgisel algoritmaların en iyi sonuçları ile karşılaştırılmıştır.

Çizelge 4.1. Küçük ve büyük boyutlu örnek setleri

		Değerler	
Örnek Faktörü	Sembol	Küçük	Büyük
İş sayısı	N	8, 12, 16, 20, 24	100, 200, 500
Makine sayısı	M	2, 3, 4, 5	5, 10, 20

Fabrika sayısı	*F*	2, 3, 4	4, 6, 8
Ürün sayısı	*P*	2, 3, 4	30, 40, 50

4.1. Matematiksel Modellere İlişkin Bulgular

Akış tipi problemleri ile ilgili matematiksel dili kullanarak problemleri ayrıntılı bir şekilde tanımlayan birçok akademik çalışma bulunmaktadır. Örneğin; Stafford, ve ark. (2005), Tseng ve Stafford (2008), Ching-Jong ve Li-Man (2008) ve Naderi ve Ruiz (2010) bunlardan sadece birkaçıdır. Hatami ve ark. (2013) tarafından önerilen karışık tamsayılı doğrusal programlama (KTDP) modeli ise; Naderi ve Ruiz'in (2010) dağıtılmış permütasyon akış tipi çizelgeleme problemi (DPÇP) için önerdiği ve diğer modellerden daha iyi performans gösterdiğini ispatladıkları beşinci matematiksel modelden esinlenerek geliştirilmiştir. Bildiğimiz kadarıyla DMPÇP için literatürde geliştirilmiş başka bir matematiksel model bulunmamaktadır.

Birden fazla satıcının bulunduğu Çoklu Gezgin Satıcı Problemi (ÇGSP) ile DMPÇP arasında benzerlik bulunması nedeniyle (Şöyle ki: Çoklu satıcılar ile fabrikalar, gezilmesi gereken noktalar arasındaki mesafeler ile yapılacak iş süreleri arasındaki benzerlik.) literatürde mevcut olan ÇGSP ile ilgili matematiksel modellerden faydalanılarak DMPÇP ile ilgili Hatami ve ark. (2013) tarafından önerilen mevcut matematiksel model yeniden tasarlanmıştır. Böylelikle iki farklı matematiksel model tarafımızdan önerilmiş ve Çizelge 4.1.'de yer alan küçük boyutlu örnekler test edilerek elde edilen sonuçlar Hatami ve ark. (2013) tarafından önerilen mevcut matematiksel modelin (MILP) aynı veri seti için bulduğu sonuçlarla karşılaştırılmıştır.

İzin verilen CPU süresinde, 900 küçük örneği aynı fiziksel koşullarda çözen MILP, Model 1 ve Model 2 algoritmaları sırasıyla % 57.67 (519 / 900), % 65 ve % 65,78 oranında optimum çözüm bulabilmiştir. Çizelge 4.2.'de, kategorize edilen sonuçların karşılaştırma kriterleri şunlardır:

- Her bir algoritmanın 900 örnekten bulduğu optimum çözümlerin; sayısı ve ortalama çözüm zamanı,

- Ortak çözülen örneklerin; hangi algoritma tarafından daha kısa sürede çözüldüğü ve algoritmaların bu ortak çözülen örnekleri ortalama çözme süreleri,
- Algoritmaların 900 örnekten; çözemedikleri örnek sayısı ve çözülemeyen örneklere ait ortalama GAP değerleri,
- Ortak çözülemeyen örneklerden; GAP değeri en düşük olan metot ve bu örneklere ait ortalama GAP değerleri.

Çizelge 4.2. Küçük boyutlu örnekler için matematiksel modellere ait özet tablo

		MILP	Model 1	Model 2
Optimum çözüm	Çözülebilen örnek sayısı	519	585	**592**
	Ortalama çözme zamanı	31,69	28,37	**20,70**
Ortak çözülen örnekler (499)	En kısa sürede çözen metot	194	42	**258**
	Ortalama çözme zamanı	20,69	25,54	**11,68**
Çözülemeyen örnekler	Çözülemeyen örnek sayısı	381	315	**308**
	Ortalama GAP (%)	6,35	7,47	**6,28**
Ortak çözülemeyen örnekler(292)	GAP değeri en düşük olan metot	65	9	**90**
	Ortalama GAP değeri (%)	7,23	7,80	**6,49**

Tabloda en iyi değerler koyu karakterlerle gösterilmiş ve belirtilen 8 test faktörü göz önünde bulundurulduğunda;

- 900 tane örnek içinde MILP, Model 1 ve Model 2 algoritmaları sırasıyla 519, 585 ve 592 tane örneği optimum olarak çözebilmiştir. Model 1 ve Model 2 bir birine çok yakın sayıda örneği çözebilmiş olsa da; algoritmaların çözmüş oldukları örnekleri ortalama çözme zamanlarına baktığımızda; MILP ve Model 1'in yaklaşık değerler aldığı, Model 2'nin ise çözmüş olduğu örnekleri bu

modellerden daha kısa sürede çözdüğü görülmektedir. “Optimum çözüm” kriterinde bulunan “Çözülebilen örnek sayısı” ve “Ortalama çözme zamanı” faktörleri göz önünde bulundurduğumuzda Model 2’nin diğer algoritmalardan daha iyi performans gösterdiğini söyleyebiliriz.

- 3 algoritmanın da küçük boyutlu olan 900 örnekten ortak çözebildikleri 499 tane örneği incelediğimizde; 258 tanesi Model 2 tarafından, 42 tanesi Model 1 tarafından ve 194 tanesi MILP tarafından en kısa sürede çözülmüştür. “Optimum çözüm” kriteri ile karşılaştırma yaptığımızda Model 1’in daha fazla örneği MILP’den daha kısa sürede çözmesi beklenirdi. MILP’in ortak çözülen örneklerden sayı ve ortalama çözüm süresi olarak daha iyi sonuç vermesinin nedeni olarak, MILP tarafından çözülemeyip de Model 1 tarafından çözülen örneklerin Model 1 tarafından çok kısa sürede çözüldüğü ve 3 model tarafından ortak çözülen örnekleri MILP’in Model 1’den daha kısa sürede çözdüğü söylenebilir.
- Çözülemeyen örnekler kriterine ilişkin verileri incelediğimizde ise çözülemeyip de optimum çözüme en yakın değer bulan algoritmanın Model 2 olduğunu görebiliyoruz. MILP’in çözemediği 381 örneğin GAP değerinin, Model 1’in çözemediği 315 tene örnekten küçük olması ise algoritmaların çözemedikleri örneklerde MILP’in optimum çözüme daha çok yaklaştığını söyleyebiliriz.
- Ortak çözülemeyen örnekler kriterinde ise 292 tane örneğin 3 algoritma tarafından da çözülemediği görülmektedir. Yine sonuçlar değerlendirildiğinde ortak çözülen örnekler kriterindeki sonuçlara benzer bir tablo ile karşılaşıyoruz. Çözülemeyen 292 örnekten; GAP değerleri en düşük olan metot ve ortalama GAP değeri faktörü göz ününde bulundurulduğunda, sırasıyla en iyi sonuçların Model 2, MILP ve Model 1 tarafından bulunduğunu görmekteyiz.

Her ne kadar Model 1 ve MILP algoritmaları belirlemiş olduğumuz 8 karşılaştırma faktöründe bir birlerine net bir üstünlük sağlamasa da, **Desrochers ve Laporte (1991)**’in alt eleme kısıtlarını kullanarak geliştirdiğimiz matematiksel modelin (Model 2) diğer modellerden belirlenen süre içinde daha fazla sayıda optimal çözümler

bulduğu, çözülemeyen örneklere ilişkin ortalama GAP değerinin ve ortalama CPU zaman tüketiminin daha az olduğu açıktır.

4.2. Meta – Sezgisel Çözüm Yöntemine ilişkin Bulgular

Bildiğimiz kadarıyla, literatürde DMPÇP'deki tamamlanma süresinin en aza indirilmesi ile ilgili çalışmalar sırasıyla:

- Hatami ve ark. (2013) tarafından önerilen yapıcı sezgisel tarama ve VND'ler,
- Wang ve Wang (2015) tarafından önerilen EDAMA,
- Lin ve Zhang (2016) tarafından önerilen HBBO,
- Lin ve ark. (2017) tarafından önerilen BS-HH,
- Pan ve ark. (2018) tarafından önerilen FFO,
- Ferone ve ark. (2019) tarafından önerilen BR-ILS algoritmasıdır.

Bütün bu yazarlar algoritmalarının performansını ilk makale tarafından oluşturulan örnekleri kullanarak test etmişlerdir. Aynı örnekler, geliştirdiğimiz SSS algoritmasının da verimliliğini test etmek için kullanılmıştır. Bu örnekler, Çizelge 4.1.'de belirtilen dört faktörden oluşmaktadır. İş işleme süreleri ve her bir üründeki işlerin sayısına (h) bağlı olan ürün montaj süreleri rastgele değişkenler kullanılarak oluşturulmuştur: U[1,99] ve U [$1 \times |N_h|$, $99 \times |N_h|$]. Küçük ve büyük boyutlu örnekler için örnek faktörü kombinasyonlarının sayısı sırasıyla 180 ve 81'dir. Küçük boyutlu örneklerde, kombinasyon başına beş çoğaltma yapılırken, büyük boyutlu örneklerde bu sayı 10'dur. Böylece, toplamda 900 küçük boyutlu ve 810 büyük boyutlu iki tane örnek seti üzerinde karşılaştırmalar yapılmıştır. Oluşturulan tüm örnekleri http://soa.iti.es adresinde bulabilirsiniz. Her örnek için, SSS algoritmasını beş defa bağımsız olarak çalıştırdık. Küçük ve büyük boyutlu örnekler için durma süresini (20xNxM) milisaniye olarak belirledik. Algoritma belirlenen süre boyunca çalıştı ve sürenin bittiği anda belirlemiş olduğu tüm çözümler içinden en iyisini nihai sonuç olarak verdi. 500_20_6_40_7 örneği üzerinden durumu açıklayacak olursak; bu büyük boyutlu örnek için SSS algoritması 500x20x20= 200000 milisaniye =200 saniye boyunca çalışmış ve çözüm değeri olarak 25185'i bulmuştur. Şöyle ki, algoritma 25185 değerini 25. saniyede de bulmuş olabilir fakat daha sonra bu sonuçla yapmış olduğu karşılaştırmalar

neticesinde daha iyi bir sonuç bulamaması nedeniyle nihai olarak 25185 değerini vermiştir. Bu çalışma sistemine göre küçük boyutlu örneklerin her biri için bulduğumuz sonuçları ortalama olarak en fazla 1,12 saniyede, büyük boyutlu örnekleri ise ortalama olarak en fazla 62,22 saniyede çözmüş oluyoruz. Çizelge 4.3.'de uygulama detayları hakkında bilgi verilen ve VND algoritmalarından daha iyi performans sergileyen diğer algoritmaların ortalama çözüm sürelerini incelediğimizde; EDEMA'nın 23,16, HBBO'nun 64,19, BS-HH'nin 12,56 ve BR-ILS'nin 60,21 saniye ortalamayla büyük boyutlu örnekleri çözdüğünü görmekteyiz. Her ne kadar elde ettiğimiz çözümlerin tam süresini bilemiyor olup da bire bir karşılaştırma yapamazsak da SSS algoritması için belirlenen ortalama süre üst sınırının 62,22 saniye olduğunu göz önünde bulundurursak; bu algoritmaların tükettiği sürelere yakın süre tükettiğimiz söylenebilir. Benzer şekilde, aynı durum küçük boyutlu örnekler için de geçerli olup, algoritmaların ortalama CPU sürelerinin ihmal edilebilecek kadar küçük olduğu söylenebilir.

Çizelge 4.3. Çözüm yaklaşımlarının uygulama koşullarının karşılaştırılması

Karşılaştırma Kriteri	Çözüm Yklaşımı					
	HBBO	EDEMA	BS-HH	FFO	BR-ILS	SSS
İşlemci	Intel Core i3 - @ 2.50 GHz	Intel Core i5 @ 2.30 GHz	Intel Core i5 @ 2.40 GHz	Intel Core i7 @ 3.40 GHz	Intel Core i5-3470S @2.90GHz	Intel Core i5@ 2.30 GHz
RAM	4 GB	Belirtilmemiş	4 GB	8.00 GB	4 GB	8 GB
Program Dili	Visual C++	Belirtilmemiş	Visual C++	Visual Studio	Java	Matlab

VND'lere ek olarak Hatami ve ark. (2013) ayrıca H11, H12, H21, H22, H31 ve H32 yapıcı yöntemlerini de önerdiler. Yapılan karşılaştırmalar VND'nin yapıcı yöntemlerden daha iyi performans sergilediğini göstermiştir. Diğer yazarlar tarafından önerilen algoritmalar incelendiğinde; HBBO ve EDAMA, VND'ler ile karşılaştırma yapıp daha iyi sonuçlar bulmuştur. BS-HH, FFO ve BR-ILS algoritmaları ise HBBO, EDAMA ve VND ile karşılaştırma yapıp her biri bu üç algoritmadan daha iyi sonuçlar bulmuştur. Fakat daha önce de belirttiğimiz gibi BS-HH, FFO ve BR-ILS algoritmaları arasında henüz bir karşılaştırma yapılmamıştır.

VND, HBBO, EDAMA, BS-HH, FFO, BR-ILS ve SSS algoritmaları deterministtik yaklaşımlar olmamaları nedeniyle bu algoritmalar arasında adil bir

karşılaştırmanın yapılabilmesi için, her örneği birkaç kez çalıştırmak mantıklı olacaktır. Literatürde tekrarlar (replikasyonlar (R)) tarafından sağlanan sonuçları değerlendirmek için farklı stratejiler geliştirilmiştir. Bir yandan, Lin ve Zhang (2016), belirli bir örnek için bağıl yüzdece sapma (RPD) ortalamasını hesaplayarak HBBO sonuçlarını bu şekilde hesapladılar:

$$\mathrm{RPD}_{\mathrm{Average}} = \sum_{i=1}^{R} \left(\frac{Res_{alg}^{i} - Res_{best}}{Res_{best}} \; x \; 100 \right) x \; \frac{1}{R}$$

burada Res_{alg}^{i} herhangi bir algoritmada çalıştırılan örneğin i'ninci defa çalıştırılmasında elde edilen toplam tamamlanma zamanını ifade eder. Diğer taraftan, Wang ve Wang (2015) belirli bir örnek için tüm çalışmalarda sağlanan en iyi çözümü dikkate alarak ve RPD'yi şu şekilde hesaplamıştır:

$$\mathrm{RPD}_{\mathrm{Best}} = \min_{i=1,\ldots.R} \left(\frac{Res_{alg}^{i} - Res_{best}}{Res_{best}} \; x \; 100 \right)$$

İlgili çalışmalarda her örnek için $\mathrm{RPD}_{\mathrm{Average}}$ değeri belirlenirken, HBBO 5 defa, BS-HH 30 defa ve BR-ILS 5 defa çalıştırılmıştır. $\mathrm{RPD}_{\mathrm{Best}}$ değerini hesaplayan algoritmalar ise EDAMA, BR-ILS ve FFO olup tekrar sayıları sırasıyla 20, 5 ve 5'tir. Adil bir karşılaştırma yapmak için SSS algoritması tarafından $\mathrm{RPD}_{\mathrm{Average}}$ ve $\mathrm{RPD}_{\mathrm{Best}}$ değerleri hesaplanırken tekrar sayısı 5 olarak belirlenmiştir.

4.2.1. Küçük boyutlu örneklerin değerlendirilmesi

DMPÇP için önceki çalışmalarda da yapıldığı gibi, küçük boyutlu örnekler üzerinde SSS algoritmasının performansını değerlendirmek için, n ve f ile kategorize edilen özetlenmiş sonuçlar düzenlenmiştir. RPD değeri her örnek için 5 defa hesaplanmış ve $\mathrm{RPD}_{\mathrm{Average}}$ değeri için 5 sonucun ortalaması alınmıştır. $\mathrm{RPD}_{\mathrm{Best}}$ değeri için ise 5 sonuç arasından en iyi olan sonuç dikkate alınmıştır. Hesaplanan $\mathrm{RPD}_{\mathrm{Average}}$ ve $\mathrm{RPD}_{\mathrm{Best}}$ değerleri sırasıyla Tablo 4.4. ve Tablo 4.5.'te görüleceği üzere n ve f faktörleriyle kategorilere ayrılmıştır.

Çizelge:4.4. Küçük boyutlu örnekler için farklı yaklaşımların ortalamalarının karşılaştırması

	RPD						$RPD_{Average}$			
f x n	H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	HBBO	BS-HH	BR-ILS	SSS
2 x 8	1,00	0,76	1,00	0,76	1,02	0,78	0,38	0,38	0,26	**0,00**
2 x 12	0,93	0,87	0,93	0,87	0,93	0,87	0,41	0,41	0,07	**0,00**
2 x 16	0,73	0,55	0,72	0,53	1,09	0,53	0,16	0,16	0,02	**-0,02**
2 x 20	0,53	0,36	0,51	0,37	0,57	0,37	−0,15	−0,17	0,03	**-0,32**
2 x 24	0,54	0,21	0,54	0,21	0,54	0,21	−0,32	−0,34	0,04	**-0,48**
3 x 8	1,09	0,70	1,15	0,76	1,15	0,76	0,40	0,40	0,05	**0,00**
3 x 12	0,44	0,28	0,44	0,28	0,44	0,28	0,12	0,12	**−0,22**	0,00
3 x 16	0,86	0,56	0,91	0,56	0,91	0,56	0,08	0,08	0,08	**0,00**
3 x 20	0,43	0,46	0,43	0,43	0,43	0,43	0,09	0,09	0,07	**-0,02**
3 x 24	0,64	0,33	0,64	0,33	0,64	0,33	−0,10	−0,12	**−0,33**	-0,13
4 x 8	1,08	0,63	0,99	0,63	0,99	0,63	0,25	0,25	**−0,07**	0,00
4 x 12	0,74	0,47	0,74	0,47	0,74	0,56	0,38	0,38	0,02	**0,01**
4 x 16	0,59	0,28	0,59	0,28	0,59	0,28	0,13	0,13	0,20	**0,03**
4 x 20	1,10	0,63	1,10	0,63	1,10	0,63	0,13	0,08	0,05	**-0,01**
4 x 24	0,57	0,26	0,57	0,26	0,57	0,26	−0,01	**−0,03**	0,01	**-0,03**
Ortalama	0,75	0,75	0,75	0,49	0,78	0,50	0,13	0,12	0,00	**-0,06**

Tablo 4.4.'te gösterildiği gibi $RPD_{Average}$ değerini hesaplayan HBBO, BS-HH, BR-ILS ve SSS algoritmaları, tek tekrar sonucu RPD değerini belirleyen Hatami ve arkadaşlarının (2013) VND algoritmasından daha iyi performans sergilemiştir. HBBO ve BS-HH neredeyse aynı kalitede sonuçlar vermiş ve BR-ILS algoritması sıfır ortalama ile HBBO ve BS-HH'den daha verimli sonuçlar bulmuştur. SSS algoritması ise 3x12,

3x24 ve 4x8 kategorileri haricinde diğer tüm kategorilerde en iyi sonuçları bulmuş ve -0,06 $RPD_{Average}$ değeri ile ortalaması negatif olan tek algoritma olmuştur.

Çizelge 4.5. Küçük boyutlu örnekler için farklı yaklaşımların en iyi değerlerinin karşılaştırması

	RPD						RPD_{Best}			
f x n	H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	EDAMA	BR-ILS	FFO	SSS
2 x 8	1,00	0,76	1,00	0,76	1,02	0,78	0,00	0,26	0,00	**0,00**
2 x 12	0,93	0,87	0,93	0,87	0,93	0,87	0,00	0,07	0,00	**0,00**
2 x 16	0,73	0,55	0,72	0,53	1,09	0,53	−0,05	0,02	0,02	**-0,06**
2 x 20	0,53	0,36	0,51	0,37	0,57	0,37	−0,33	0,03	0,09	**-0,40**
2 x 24	0,54	0,21	0,54	0,21	0,54	0,21	−0,48	0,04	0,17	**-0,56**
3 x 8	1,09	0,70	1,15	0,76	1,15	0,76	0,00	0,02	0,00	**0,00**
3 x 12	0,44	0,28	0,44	0,28	0,44	0,28	0,01	**-0,25**	0,00	0,00
3 x 16	0,86	0,56	0,91	0,56	0,91	0,56	−0,01	0,06	0,02	**-0,01**
3 x 20	0,43	0,46	0,43	0,43	0,43	0,43	−0,01	0,03	0,08	**-0,04**
3 x 24	0,64	0,33	0,64	0,33	0,64	0,33	−0,16	**-0,38**	0,15	-0,18
4 x 8	1,08	0,63	0,99	0,63	0,99	0,63	0,00	**−0,14**	0,00	0,00
4 x 12	0,74	0,47	0,74	0,47	0,74	0,56	0,00	0,00	0,00	**0,00**
4 x 16	0,59	0,28	0,59	0,28	0,59	0,28	0,03	0,20	0,02	**-0,01**
4 x 20	1,10	0,63	1,10	0,63	1,10	0,63	-0,01	0,03	0,07	**-0,07**
4 x 24	0,57	0,26	0,57	0,26	0,57	0,26	-0,05	0,00	0,14	**-0,09**
Ortalama	0,75	0,75	0,75	0,49	0,78	0,50	−0,07	0,00	0,05	**-0,10**

Küçük boyutlu örnekler için RPD_{Best} değeri ise EDAMA, FFO ve BR-ILS algoritmaları tarafından hesaplanmıştır. VND algoritmasının tek tekrar sonucunda bulmuş olduğu sonuçlar da bu tabloda yer almıştır. EDAMA, FFO ve BR-ILS algoritmalarının RPD_{Best} değerleri incelendiğinde, bu algoritmaların VND algoritmasından daha iyi performans sergilediği söylenebilir. EDAMA, FFO, BR-ILS

ve SSS algoritmaları ortalama olarak her ne kadar yakın değerler bulmuş olsa da 3x12, 3x24 ve 4x8 kategorileri hariç diğer kategorilerde ortalama -0,10 RPD_{Best} değeriyle en iyi çözümler SSS algoritması tarafından bulunmuştur.

Böylelikle VND, EDAMA, HBBO, BS-HH, FFO ve BR-ILS algoritmalarının sonuçları da göz önünde bulundurularak güncel olan en iyi sonuçların oluşturduğu 900 tane küçük örnek seti içerisinde SSS algoritması Tablo 4.6.'da detayları verilen 40 tane daha iyi sonuç bulmuştur.

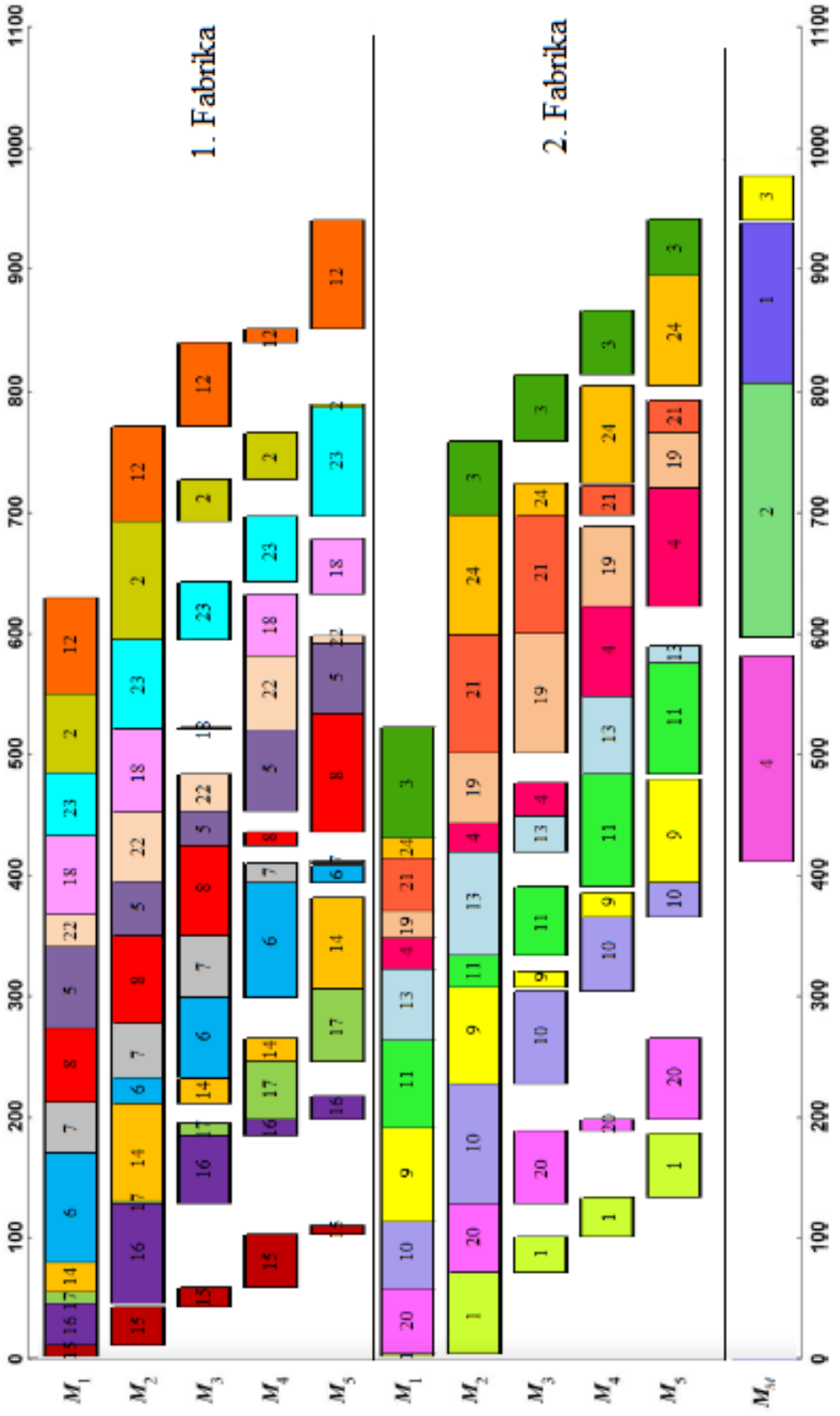
Çizelge 4.6. Küçük örnekler için yeni en iyi çözümler

Örnek	En İyi Bilinen değer	SSS	RPD
(16_2_2_2_3)	1058 (Hatami ve ark. 2013)	1056	-0,19
(16_3_2_2_5)	702 (Hatami ve ark. 2013)	701	-0,14
(16_3_2_3_1)	1041 (Hatami ve ark. 2013)	1038	-0,29
(16_3_2_3_2)	563 (Hatami ve ark. 2013)	560	-0,53
(16_3_2_4_2)	626 (Hatami ve ark. 2013)	625	-0,16
(16_5_2_2_4)	1389 (Hatami ve ark. 2013)	1385	-0,29
(16_5_2_3_5)	1330 (Ferone ve ark. 2019)	1326	-0,3
(16_5_2_4_4)	835 (Ferone ve ark. 2019)	825	-1,2
(20_2_2_3_2)	1206 (Hatami ve ark. 2013)	1198	-0,66
(20_2_3_2_2)	1627 (Hatami ve ark. 2013)	1626	-0,06
(20_2_4_2_2)	365 (Lin ve ark. 2017)	359	-1,64
(20_3_2_3_2)	1115 (Ferone ve ark. 2019)	1100	-1,35
(20_3_2_3_3)	895 (Hatami ve ark. 2013)	881	-1,56
(20_3_4_3_2)	1369 (Hatami ve ark. 2013)	1363	-0,44
(20_4_2_2_2)	860 (Ferone ve ark. 2019)	831	-3,37
(20_4_2_3_2)	820 (Ferone ve ark. 2019)	814	-0,73
(20_4_2_4_2)	1277 (Ferone ve ark. 2019)	1266	-0,86
(20_4_3_2_1)	616 (Hatami ve ark. 2013)	611	-0,81
(20_4_3_3_5)	1129 (Lin ve ark. 2017)	1117	-1,06
(20_5_2_2_2)	1250 (Ferone ve ark. 2019)	1248	-0,16
(20_5_2_3_2)	1171 (Ferone ve ark. 2019)	1165	-0,51

(20_5_4_2_1)	1002 (Ferone ve ark. 2019)	996	-0,6
(24_2_2_2_2)	947 (Hatami ve ark. 2013)	944	-0,32
(24_2_2_3_2)	1509 (Lin ve ark. 2017)	1491	-1,19
(24_2_4_2_1)	1442 (Hatami ve ark. 2013)	1441	-0,07
(24_2_4_2_5)	556 (Lin ve ark. 2017)	553	-0,54
(24_3_2_2_5)	1674 (Lin ve Zhang 2016)	1671	-0,18
(24_3_2_3_2)	1398 (Lin ve Zhang 2016)	1391	-0,5
(24_3_2_4_2)	751 (Lin ve Zhang 2016)	734	-2,26
(24_3_3_2_4)	578 (Lin ve Zhang 2016)	576	-0,35
(24_3_3_3_1)	1429 (Hatami ve ark. 2013)	1428	-0,07
(24_3_3_4_1)	916 (Ferone ve ark. 2019)	915	-0,11
(24_3_3_4_2)	1037 (Ferone ve ark. 2019)	1031	-0,58
(24_3_4_2_3)	1753 (Hatami ve ark. 2013)	1741	-0,68
(24_3_4_3_3)	1482 (Hatami ve ark. 2013)	1478	-0,27
(24_4_4_3_5)	1219 (Hatami ve ark. 2013)	1218	-0,08

Küçük boyutlu örnek setinde yer alan 24_5_2_4_2 örneği için çözüm değeri sırasıyla;

- Hatami ve ark. (2013) tarafından 1054,
- Lin ve Zhang (2016) tarafından 1032,
- Lin ve ark. (2017) tarafından 1031,
- Ferone ve ark. (2019) tarafından 1022 olarak bulunmuştur. SSS algoritması tarafından bulunan çözüm değeri ise 980 olup ilgili çözüme karşılık gelen Gantt şeması Şekil 4.1.'de gösterilmiştir.



Şekil 5.1. SSS algoritması tarafından elde edilen I_24_5_2_4_2 örneğine ait yeni en iyi çözümün Gantt şeması,

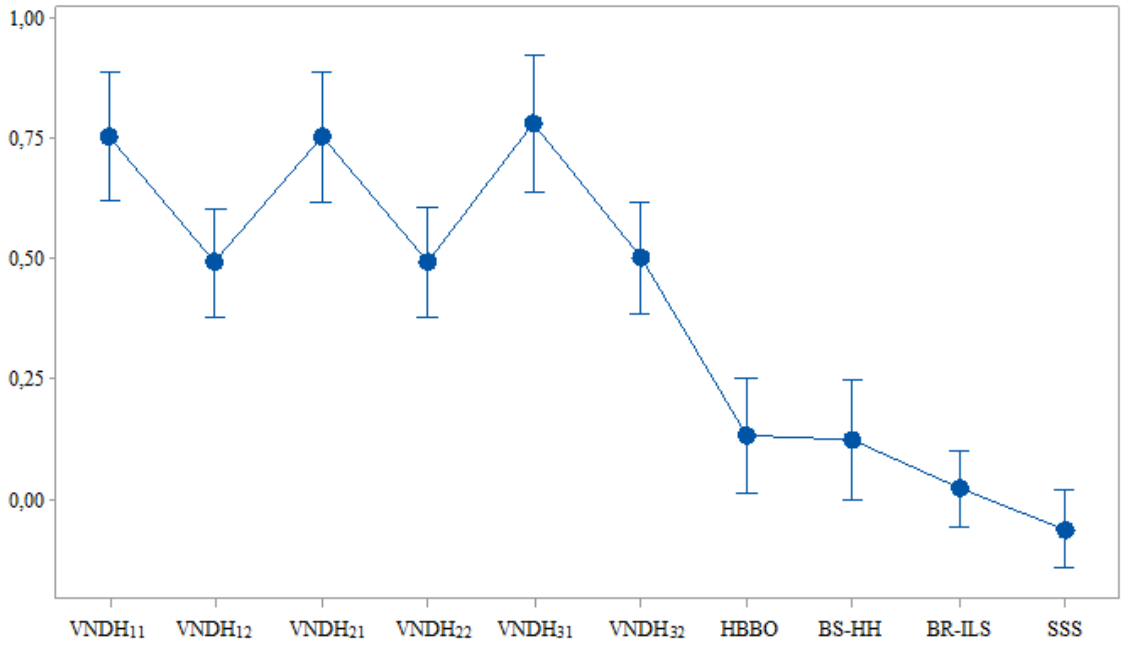
Çizelge 4.4.'te ve Çizelge 4.5.'te gözlemlenen farklılıkların istatistiksel olarak anlamlı olup olmadığını incelemek için, farklı algoritmalar tarafından sağlanan ortalama RPD değerleri üzerinde varyans analizi (ANOVA) yapılmıştır. Algoritmalara ait ortalamaların gösterildiği grafik ve % 95 güven seviyesinde Tukey'in dürüst anlamlı fark (honest significant difference) aralıkları;

- $RPD_{Average}$ endeksi dikkate alınarak VND, HBBO, BR-ILS ve SSS algoritmaları için Şekil 4.1. ve Çizelge 4.7.'de,
- RPD_{Best} indeksi dikkate alınarak VND, EDAMA, BR-ILS, FFO ve SSS algoritmaları için Şekil 4.2. ve Çizelge 4.8.'de gösterilmektedir.

Çizelge 4.7. Küçük örneklerde $RPD_{Average}$ indeksine için Tukey'in %95 güven seviyesindeki anlamlılık karşılaştırması

Algoritma	Ortalama	Gruplar		
H_{31}	0,7807	A		
H_{11}	0,7513	A		
H_{21}	0,7507	A		
H_{32}	0,4987		B	
H_{22}	0,4913		B	
H_{12}	0,4900		B	
HBBO	0,1300			C
BS-HH	0,1213			C
BR-ILS	0,0187			C
SSS	-0,0647			C

Çizelge 4.7. de görüldüğü gibi %95 güven seviyesinde: anlamlılık derecelerine göre farklılaşmayan algoritmalar aynı harflerle, farklılaşan gruplar ise farklı harflerle gösterilmiştir. C harfi ile gösterilen HBBO, BS-HH BR-ILS ve SSS algoritmaları anlamlı olarak birbirlerinden farklılaşmamış fakat VND algoritmalardan daha iyi performans göstererek farklılaşmışlardır.



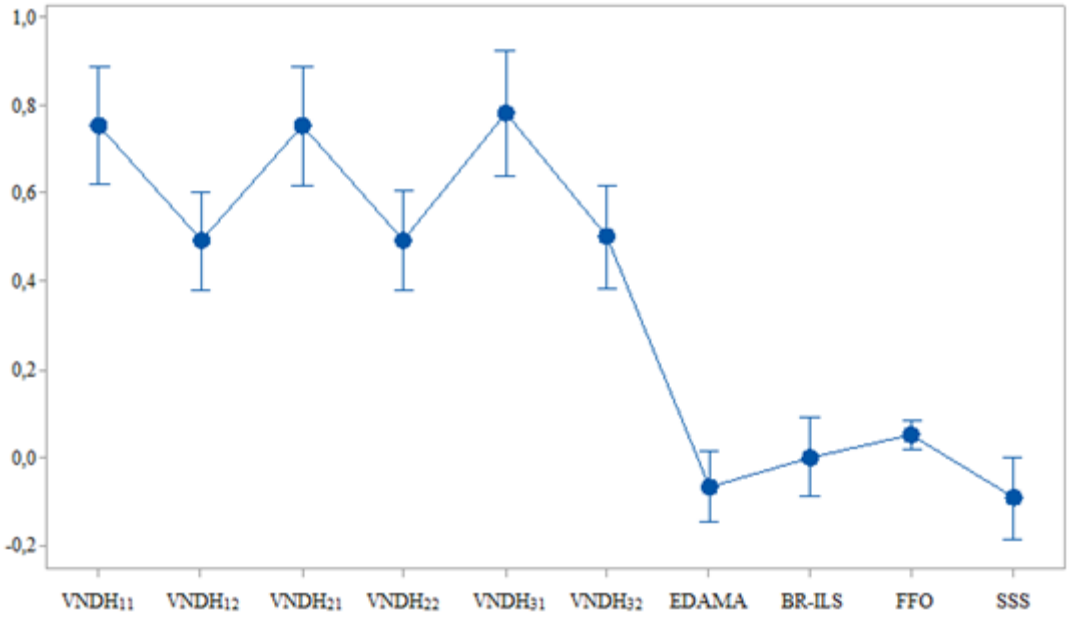
Şekil 4.2. Küçük örnekler için% 95 güven aralığına sahip $RPD_{Average}$ grafiği.

%95 güven seviyesinde VND, HBBO, BS-HH, BR-ILS ve SSS algoritmalarının ortalamalarının karşılaştırıldığı Şekil 4.2.'de görüldüğü gibi HBBO, BS-HH, BR-ILS ve SSS algoritmaları VND algoritmalarından daha iyi performans göstermiştir. Ortalama olarak -0,06 RPD değerine sahip olan SSS algoritmasının diğer algoritmalardan daha kaliteli sonuçlar bulduğu anlaşılmaktadır.

Çizelge 4.8. Küçük örneklerde RPD_{Best} indeksine için Tukey'in %95 güven seviyesindeki anlamlılık karşılaştırması

Algoritma	Ortalama	Gruplar		
H_{31}	0,7807	A		
H_{11}	0,7513	A		
H_{21}	0,7507	A		
H_{32}	0,4987		B	
H_{22}	0,4913		B	
H_{12}	0,4900		B	
FFO	0,0507			C
BR-ILS	-0,0007			C
EDAMA	-0,0707			C
SSS	-0,0947			C

Küçük örnekler için ANOVA analizi sonucunda elde ettiğimiz RPD_{Best} değerlerinin karşılaştırmasına yönelik tabloyu incelediğimizde; FFO. BR-ILS, EDAMA ve SSS algoritmalarının anlamlı olarak birbirinden farklılaşmadığını ve RDP değeri verilmiş olan VND algoritmalardan daha iyi sonuçlar bulduğunu görmekteyiz.



Şekil 4.3. Küçük örnekler için% 95 güven aralığına sahip RPD_{Best} grafiği.

Şekil 4.3.'te yer alan RPD_{Best} indeksine göre ise SSS algoritması ve EDAMA arasında önemli bir fark yoktur, ancak SSS algoritması ortalama olarak daha düşük RPD bildirmektedir. Diğer taraftan, SSS algoritmasının her örnek için 5 defa, EDAMA'nın ise 20 defa çalıştırılıp bulmuş olduğu en iyi sonucu tabloya yansıttığını göz önünde bulunduracak olursak, SSS algoritması için de 20 tekrar yapmamız durumunda daha iyi sonuçlar bulabileceğimizi söyleyebiliriz.

4.2.2. Büyük boyutlu örneklerin değerlendirilmesi

Büyük boyutlu örnekler için ortalama RPD değerleri ve en iyi RDP değerlerine ilişkin bilgiler sırasıyla Çizelge 4.9. ve Çizelge 4.11.' de özetlenmiştir. Örnekler; fabrika sayısı (f), ürün sayısı (t) ve iş sayısı (n) olarak sınıflandırılmıştır. $RPD_{Average}$

indeksi kapsamında HBBO, BS-HH, BR-ILS ve SSS algoritmaları arasında karşılaştırma yapılmıştır. Aynı şekilde RPD_{Best} indeksi kapsamında ise EDAMA, BR-ILS ve SSS algoritmaları arasında karşılaştırma yapılmıştır. HBBO, BS-HH ve BR-ILS algoritmaların kendi bulmuş oldukları sonuçları VND algoritmaları ile karşılaştırmaları nedeniyle her ne kadar RPD değeri ($RPD_{Average}$ veya RPD_{Best} değeri hesaplanmamıştır, sadece tek tekrar yapılarak RPD değeri hesaplanmıştır.) hesaplanmış olsa da . $RPD_{Average}$ ve RPD_{Best} indekslerine göre yapmış olduğumuz karşılaştırmada VND algoritmalarının sonuçlarına da yer verilmiştir.

Çizelge 4.9. Büyük boyutlu örnekler için farklı yaklaşımların ortalamalarının karşılaştırması

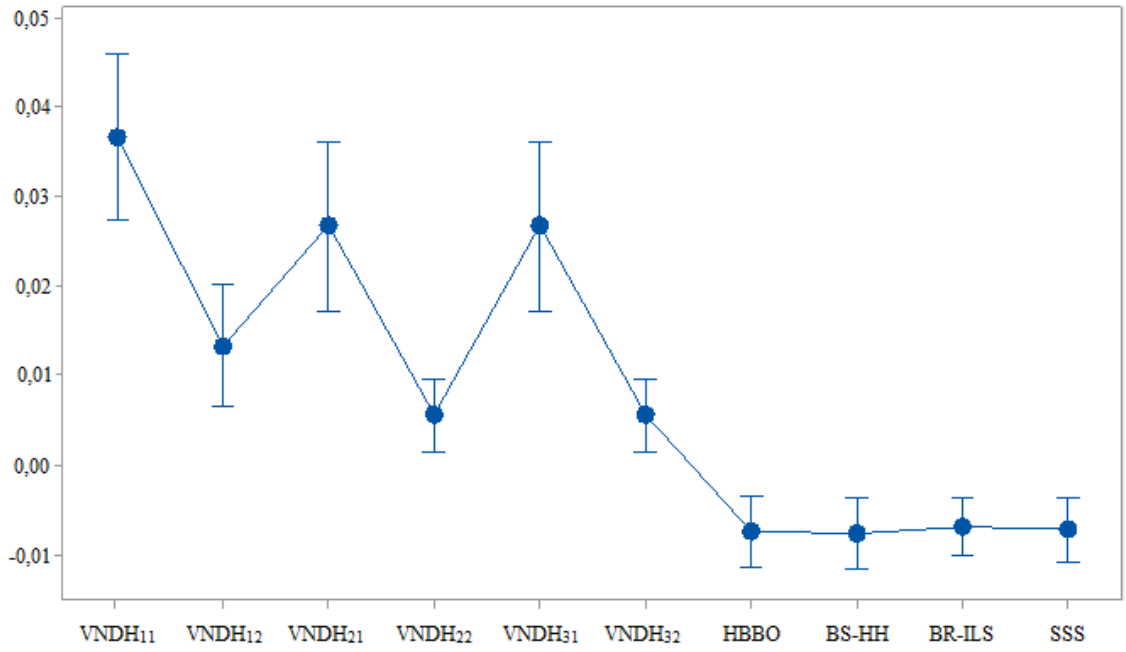
		RPD						$RPD_{Average}$			
		H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	HBBO	BS-HH	BR-ILS	SSS
Fabrikalar	4	0,06	0,03	0,05	0,01	0,05	0,01	−0,013	−0,014	−0,015	-0,014
(f)	6	0,03	0,01	0,02	0,00	0,02	0,00	−0,005	−0,004	−0,003	-0,004
	8	0,02	0,00	0,01	0,00	0,01	0,00	−0,005	−0,005	−0,003	-0,005
Ürünler	30	0,03	0,01	0,04	0,01	0,04	0,01	−0,012	−0,013	−0,010	-0,010
	40	0,04	0,02	0,02	0,01	0,02	0,01	−0,006	−0,007	−0,008	-0,011
(t)	50	0,04	0,01	0,02	0,00	0,02	0,00	−0,004	−0,003	−0,003	-0,001
İşler	100	0,05	0,02	0,03	0,01	0,03	0,01	−0,003	−0,003	−0,010	-0,008
	200	0,03	0,01	0,02	0,00	0,02	0,00	−0,003	−0,004	−0,006	-0,002
(n)	500	0,03	0,01	0,03	0,01	0,03	0,01	−0,017	−0,016	−0,005	-0,011
Ortalama		0,04	0,01	0,03	0,01	0,03	0,01	−0,007	−0,008	−0,007	-0,007

Büyük boyutlu örnekler için HBBO, BS-HH, BR-ILS ve SSS algoritmaları ortalama RPD değeri olarak VND algoritmalarından daha iyi sonuçlar bulmuştur. HBBO, BS-HH, BR-ILS ve SSS algoritmaları kendi içlerinde çok yakın çözümler bulmuş olsa da BS-HH algoritmasının -0,008 RPD değeri ile ortalama olarak en iyi sonuçları verdiğini söyleyebiliriz.

Çizelge 4.10. Büyük örneklerde $RPD_{Average}$ indeksine için Tukey'in %95 güven seviyesindeki anlamlılık karşılaştırması

Algoritma	Ortalama	Gruplar			
H_{11}	0,03667	A			
H_{31}	0,02667	A			
H_{21}	0,02667	A			
H_{12}	0,01333		B		
H_{32}	0,00556		B	C	
H_{22}	0,00556		B	C	
BR-ILS	-0,00700			C	D
SSS	-0,00733				D
HBBO	-0,00756				D
BS-HH	-0,00767				D

Çizelge 4.10.'u incelediğimiz de gözümüze ilk çarpan hususlardan biri Küçük örneklerde olduğu gibi; BR-ILS, SSS, HBBO ve BS-HH algoritmaları ile VND algoritmaları arasındaki ortalama farkının fazla olmaması nedeniyle, anlamlılık guruplarının sayısı 3'ten 4'e çıkmıştır. Şöyle ki, Küçük örneklerde D grubunda yer alan BR-ILS algoritması, B grubunda yer alan VND_{H32} ve VND_{H22} algoritmalarından %95 güven aralığında anlamlılık olarak farklılaşamadığı için B ve D grubunun arasında, VND_{H32}, VND_{H22} ve BR-ILS'den meydana gelen C grubu oluşmuştur.



Şekil 4.4. Büyük örnekler için% 95 güven aralığına sahip $RPD_{Average}$ grafiği.

Büyük örnekler için $RPD_{Average}$ değerlerinin gösterildiği Şekil 4.4. incelendiğinde, değerler birbirine çok yakın gözükse de BS-HH algoritmasının en iyi sonuçları verdiği göze çarpmaktadır. VND algoritması ise RPD değerleri olarak yine HBBO, BS-HH, BR-ILS ve SSS algoritmalarından daha kötü sonuçlar bulmuştur.

Çizelge 4.11. Büyük boyutlu örnekler için farklı yaklaşımların en iyi değerlerinin karşılaştırması

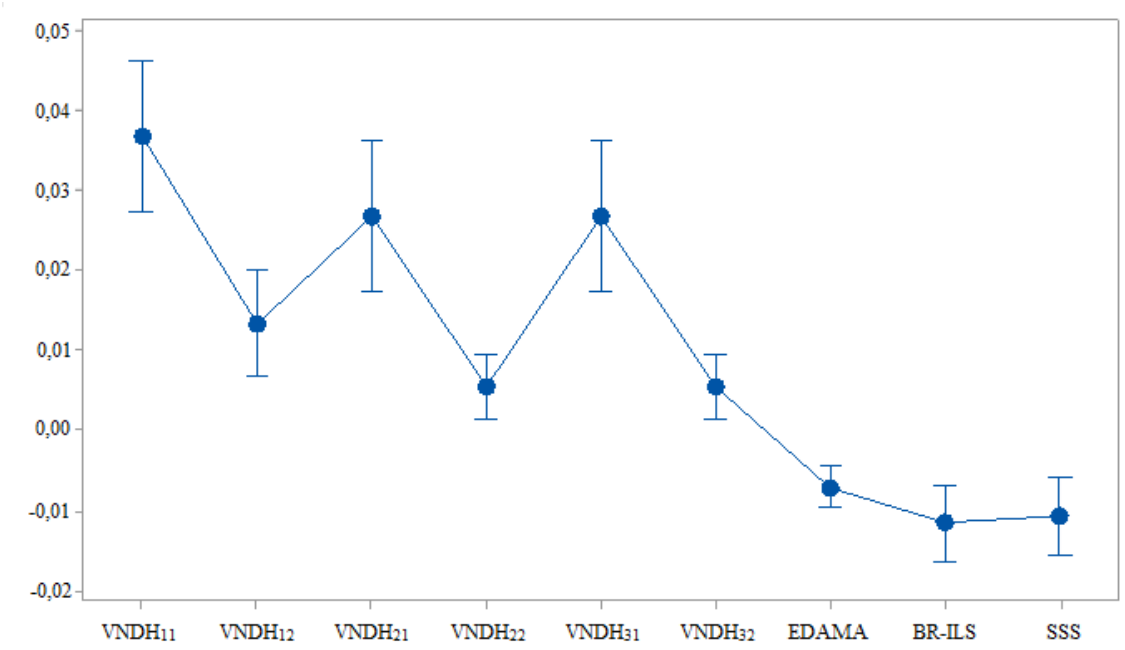
		RPD						RPD_{Best}		
		H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	EDAMA	BR-ILS	SSS
Fabrikalar	4	0,06	0,03	0,05	0,01	0,05	0,01	−0,013	−0,023	-0,021
(f)	6	0,03	0,01	0,02	0,00	0,02	0,00	−0,004	−0,007	-0,007
	8	0,02	0,00	0,01	0,00	0,01	0,00	−0,004	−0,005	-0,004
Ürünler	30	0,03	0,01	0,04	0,01	0,04	0,01	−0,011	−0,017	-0,016
	40	0,04	0,02	0,02	0,01	0,02	0,01	−0,008	−0,012	-0,011
(t)	50	0,04	0,01	0,02	0,00	0,02	0,00	−0,003	−0,006	-0,005
İşler	100	0,05	0,02	0,03	0,01	0,03	0,01	−0,008	−0,011	-0,009
	200	0,03	0,01	0,02	0,00	0,02	0,00	−0,006	−0,007	-0,005
(n)	500	0,03	0,01	0,03	0,01	0,03	0,01	−0,007	−0,017	-0,018
Ortalama		0,04	0,01	0,03	0,01	0,03	0,01	−0,007	−0,012	-0,011

RPD_{Best} indeksine göre algoritmaların ürettiği sonuçların özetlendiği tabloyu incelediğimizde; BR-ILS ve SSS algoritmalarının $RPD_{Average}$ değerini hesaplarken ortalamalarını aldıkları 5 tekrar sonucundan en iyilerini tabloya yansıtmaları nedeniyle RPD_{Best} değerlerinin $RPD_{Average}$ değerlerinden daha iyi olduğunu görmekteyiz.

Çizelge 4.12. Büyük örneklerde RPD_{Best} indeksine için Tukey'in %95 güven seviyesindeki anlamlılık karşılaştırması

Algoritma	Ortalama	Gruplar		
H_{11}	0,03667	A		
H_{31}	0,02667	A		
H_{21}	0,02667	A		
H_{12}	0,01333		B	
H_{32}	0,00556		B	C
H_{22}	0,00556		B	C
EDAMA	-0,00711			C D
SSS	-0,01067			D
BR-ILS	-0,01167			D

Çizelge 4.12. incelendiğinde büyük örneklerin $RPD_{Average}$ değerlerinin %95 güven seviyesinde algoritmaları karşılaştırmasındaki benzer bir durum burada da mevcuttur. D grubunda yer alan EDAMA algoritmasının RPD_{Best} değeri ile B grubunda yer alan VND algoritmalarının arasında anlamlı bir faklılaşma olmamıştır. Böylelikle sadece SSS ve BR-ILS algoritmaları VND algoritmalarından anlamlı olarak faklılaşmıştır.



Şekil 4.5. Büyük örnekler için% 95 güven aralığına sahip RPD_{Best} grafiği.

Şekil 4.5.'te Grafikte görüleceği üzere, EDAMA, BR-ILS ve SSS, algoritmaları VND algoritmalarından daha iyi sonuçlar rapor etmekte olup kendi içlerinde benzer performans göstermişlerdir. Yaptığı 5 tekrar içinden en iyilerini derleyerek tablodaki sonuçları bulan BR-ILS ve SSS algoritması, 20 tekrar içinden en iyilerini tabloya yansıtan EDAMA'dan daha iyi performans göstermiştir.

Sonuç olarak, büyük boyutlu örnekler için SSS algoritması diğer algoritmalardan tarafından bulunmuş en iyi değerlerden daha iyi sonuçlar bulamasa da; $RPD_{Average}$ ve RPD_{Best} değerleri göz önünde bulundurulduğunda diğer algoritmalar tarafından üretilen çözümlere benzer kalitede çözümler bulmuştur.

5. TARTIŞMA ve SONUÇ

Bu bölümde, bu tezde izlenen genel araştırma hatlarını ve genel sonuçları ele aldık. Bu tez, dağıtılmış imalat ve montaj çizelgeleme problemlerini bir bütün olarak çözmeye odaklanmıştır. Aşağıda bu çalışma hakkında bazı genel bilgiler ve bu tezin ana katkıları özetlenmiştir. Daha sonra gelecekteki araştırmalar için çeşitli olası çalışmalar belirtilmiştir.

Üretim sistemleri son zamanlarda bir takım önemli engellerle karşılaşmış ve bu zorlukları aşmak için rekabetçi üretim sistemlerinin yeniden dizayn edilmesi gerekmiştir. Dağıtık imalat ve montaj sistemleri bu engellere çözüm bulmak adına kullanılan iki önemli alternatiftir. Bu tezde, her iki alternatifin de bir üretim probleminde bulunduğu Dağıtılmış Montaj Hattı Permütasyon Akışı Çizelgeleme Problemi (DMPÇP) incelenmiştir. DMPÇP, ilk aşamada dağıtılmış üretim sistemlerinin bulunduğu, ikinci aşamada ise nihai ürünleri üretmek için işlerin tek bir montaj makinesinde, tanımlanmış bir montaj programı aracılığıyla birleştirildiği karmaşık yapılı bir optimizasyon problemidir. Bu tezdeki amaç fonksiyonu üretilen ürünlerin montaj aşamasındaki üretim süresinin en aza indirilmesidir. Problemi çözmek için tarafımızdan iki farklı matematiksel model ve bir meta sezgisel yöntem önerilmiştir. Önermiş olduğumuz Model 1 ve Model 2 isimli matematiksel modelleri test etmek için Visual Studio ve CPLEX yazılımlarını kullandık. Bu tür problemler NP-Zor olmaları nedeniyle büyük boyutlu örnekler kesin çözüm yöntemleri ile çözülemezler. Bu nedenle, bunları çözmek için sezgisel bir yaklaşım geliştirdik.

Önerdiğimiz meta sezgisel algoritma, orijinal olarak sürekli kısıtlı ve kısıtsız optimizasyon problemlerini çözmek için geliştirilen SSS algoritmasının ele aldığımız kombinatoryal optimizasyon problemini çözebilmek için değiştirilmiş halidir. Şöyle ki algoritmik çerçeve aynı kalmıştır, ancak SSS algoritmasının ele aldığımız kombinatoryal optimizasyon problemini daha etkin çözebilmesi için SSS algoritması için yeni komşuluk yapıları tanımlanmıştır. Elde edilen sonuçlar, kombinatoryal SSS algoritmasının tatmin edici bir performansa sahip olduğunu ve kombinatoryal optimizasyon problemlerini çözmede oldukça yetenekli olduğunu göstermiştir.

Karışık tamsayılı doğrusal programlama (KTDP) modellerini ve meta-sezgisel SSS algoritmasını test etmek için, farklı dört örnek faktörüne (iş, makine, ürün ve fabrika sayısı) dayalı iki örnek seti kullanılmıştır. KTDP modelleri, 600 saniye zaman kısıtı altında, küçük örnek setinde yer alan örnekler üzerinde test edilmiştir. Önerdiğimiz 'Model 2' matematiksel modeli tüm küçük örnekler için optimum veya en iyi çözümü bulamasa da; modelin, Hatami ve ark. (2013) tarafından önerilmiş olan MILP matematiksel modelinden (DMPÇP'nin çözümüne yönelik olarak şimdiye kadar literatürde bilinen en etkin matematiksel yöntem) daha verimli çalıştığı gösterilmiştir.

Meta sezgisel yöntem ise küçük ve büyük örnek setlerinde test edilmiştir. Önerilen algoritmaların karşılaştırılabilmeleri için her örneğe ait RPD değerleri hesaplanmış ve algoritmaların, örneklerin çözümlerine ilişkin sonuçlarının istatistiksel anlamlılığını değerlendirmek için de varyans analizi yapılmıştır. DMPÇP'nin çözümüne yönelik olarak şu ana kadar gerçekleştirilen tüm çalışmaların (Hatami ve ark. (2013), Wang ve Wang (2015), Lin ve Zhang (2016), Lin ve ark. (2017), Pan ve ark. (2018) ve Ferone ve ark. (2019)) sonuçları göz önünde bulundurularak; küçük boyutlu örnek setinde yapılan tüm çalışmalardan daha iyi sonuçlar bularak daha küçük RPD değerleri elde ettiğimiz ve test sırasında kullanılan 900 tane örnek setine yönelik var olan en iyi çözümlerin 40 tanesinden daha iyi çözümler elde ettiğimiz görülmüştür. Büyük boyutlu örneklerde ise diğer algoritmalara benzer çözümler bulduğumuz görülmüştür.

Gelecekteki araştırmalarda ise paralel ve dağıtılmış hesaplama (parallel and distributing computing) için uygun bir yapıya sahip olduğu için SSS algoritmasının paralel ve dağıtılmış uygulamaları yapılabilir.

KAYNAKLAR

Akpinar S., Bayhan G.M. 2011. A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. ***Engineering Applications of Artificial Intelligence,*** **24:** 449–457

Allahverdi, A., Al-Anzi, F.S., 2006. Evolutionary heuristics and an algorithm for the two stage assembly scheduling problem to minimize makespan with setup times..***International Journal of Production Research,*** **44** (22): 4713– 4735.

Allahverdi, A., Al-Anzi, F.S., 2009. The two-stage assembly scheduling problem to minimize total completion time with setup times. ***Computers & Operations Research,*** **36** (10): 2740–2747.

Allahverdi, A., Aydilek, A., Aydilek, H., 2016. Minimizing the number of tardy jobs on a two-stage assembly flowshop. ***Journal of Industrial and Production Engineering,*** **33** (6): 391–403.

Allahverdi, A., Aydilek, H., 2015. The two stage assembly flowshop scheduling problem to minimize total tardiness. ***Journal of Intelligent Manufacturing,*** **26** (2): 225–237.

Baykasoğlu, A., & Akpinar, Ş. 2017. Weighted Superposition Attraction (WSA): A swarm intelligence algorithm for optimization problems–Part 1: Unconstrained optimization. ***Applied Soft Computing***, **56:** 520-540.

Baykasoğlu A., Hamzadayı A., Akpinar S., 2019. Single Seekers Society (SSS): Bringing together heuristic optimization algorithms for solving complex problems. ***Knowledge-Based Systems,*** **165:** 53-76

Bektas, T., 2006. The multiple traveling salesman problem: an overview of formulations and solution procedures. ***Omega,*** **34:** 209 – 219

Ching-Jong, L., and L. Li-Man. 2008. Improved MILP Models for Two-machine Flowshop with Batch Processing Machines. ***Mathematical and Computer Modelling,*** **48** (7–8): 1254–1264.

Desrochers M., Laporte G., 1991. Improvements and extensions to the Miller- Tucker-Zemlin subtour elimination constraints. ***Operations Research Letters,*** **10** (01): 27-36

Du, X., Ji, M., Li, Z., Liu, B., 2016. Scheduling of stochastic distributed assembly flowshop under complex constraints. ***2016 IEEE Symposium Series on Computational Intelligence (SSCI)***, Athens, Greece. 1–7.

Dueck, G. 1993. New optimization heuristics: The great deluge algorithm and the record- to-record travel. ***Journal of Computational Physics***, **104** (1): 86-92.

Dueck, G., & Scheuer, T. 1990. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. ***Journal of Computational Physics***, **90** (1): 161-175.

Fernandez-Viagas, V., Framinan, J.M., 2015. A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. ***International Journal of Production Research,*** **53** (4): 1111–1123.

Ferone D., Hatami S., Gonz´alez- Neira E. M., Juan A. A., Festa P., 2019. A biased-randomized iterated local search for the distributed assembly permutation flow-shop problem. ***Internatıonal Transactıons in Operational Research,*** **27** (03): 1–24

Gonzalez-Neira, E.M., Ferone, D., Hatami, S., Juan, A.A., 2017. A biased-randomized simheuristic for the distributed assembly permutation flow-shop problem with stochastic processing times. ***Simulation Modelling Practice and Theory,*** **79:** 23–36.

Hariri, A., Potts, C., 1997. A branch and bound algorithm for the two-stage assembly scheduling problem. ***European Journal of Operational Research,*** **103** (3): 547–556.

Hatami, S., Ebrahimnejad, S., Tavakkoli-Moghaddam, R., Maboudian, Y., 2010. Two meta-heuristics for three-stage assembly flowshop schedulingwith sequence-dependent setup times. ***International Journal of Advanced Manufacturing Technology,*** **50** (9–12): 1153–1164.

Hatami, S., Ruiz, R., Andr´es-Romano, C., 2013. The distributed assembly permutation flowshop scheduling problem. ***International Journal of Production Research,*** **51** (17): 5292–5308.

Hatami, S., Ruiz, R., Andr´es-Romano, C., 2015. Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. ***International Journal of Production Economics,*** **169:** 76– 88.

Ji, M., Yang, Y., Duan, W., Wang, S., Liu, B., 2016. Scheduling of no-wait stochastic distributed assembly flowshop by hybrid PSO. ***IEEE Congress on Evolutionary Computation (CEC), Vancouver, BC, Canada,*** pp. 2649–2654.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. ***Science,*** **220:** 671-680.

Komaki, G., Kayvanfar, V., 2015. Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. ***Journal of Computational Science,*** **8:** 109–120.

Komaki, G., Sheikh, S.,Malakooti, B., 2019. Flow shop scheduling problems with assembly operations: a review and new trends. ***International Journal of Production Research,*** **57** (10): 2926–2955.

Komaki, G., Teymourian, E., Kayvanfar, V., Booyavi, Z., 2017. Improved discrete cuckoo optimization algorithm for the three-stage assembly flowshop scheduling problem. ***Computers & Industrial Engineering,*** **105:** 158–173.

Kulkarni R.V., Bhave P.R., 1985. Integer programming formulations of vehicle routing problems. ***European Journal of Operational Research,*** **20:** 58-67

Lee, C.-Y., Cheng, T., Lin, B., 1993. Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. ***Management Science,*** **39** (5): 616–625.

Liao, C.-J., Lee, C.-H., Lee, H.-C., 2015. An efficient heuristic for a two-stage assembly scheduling problem with batch setup times to minimize makespan. ***Computers & Industrial Engineering,*** **88** (313): 317–325.

Lin, S.-W., Ying, K.-C., 2016. Minimizing makespan for solving the distributed no-wait flowshop scheduling problem. ***Computers & Industrial Engineering,*** **99:** 202–209.

Lin J., Wang Z.-J., Li X., 2017. A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem. ***Swarm and Evolutionary Computation,*** **36:** 124–135

Lin, J., Zhang, S., 2016. An effective hybrid biogeography-based optimization algorithm for the distributed assembly permutation flow-shop scheduling problem. ***Computers and Industrial Engineering,*** **97:** 128–136.

Mahdavi, I., Shirazi, B., Cho, N., Sahebjamnia, N., and Ghobadi, S. 2008. Modeling an e-based real-time quality control information system in distributed manufacturing shops. ***Computers in Industry***, **59** (8): 759–766.

Mladenovi´c, N., Alkandari, A., Pei, J., Todosijevi´c, R., Pardalos, P.M., 2020. Less is more approach: basic variable neighborhood search for the obnoxious p-median problem. ***International Transactions in Operational Research,*** **27** (1): 480–493.

Mozdgir, A.Fatemi Ghomi, S., Jolai, F., Navaei, J., 2013. Two-stage assembly flow-shop scheduling problem with nonidentical assembly machines considering setup times. ***International Journal of Production Research,*** **51** (12): 3625– 3642.

Naderi, B., Ruiz, R., 2010. The distributed permutation flowshop scheduling problem. ***Computers & Operations Research,*** **37** (4): 754–768.

Naderi, B., Ruiz, R., 2014. A scatter search algorithm for the distributed permutation flowshop scheduling problem. ***European Journal of Operational Research,*** **239** (2): 323–334.

Onwubolu, G., & Davendra, D. 2006. Scheduling flow shops using differential evolution algorithm. ***European Journal of Operational Research***, **171** (2): 674-692.

Pan Y.-R., Chen Q.-D., Pan Q.-K., 2018. An Effective Fruit Fly Optimization for the Distributed Assembly Flowshop Scheduling Problem. ***Proceedings of the 37th Chinese Control Conference***, Wuhan, China. 8374-8378.

Peklenik, J. 1992. Fms: A complex object of control. ***In The Eight International IFIP WG5.3 PROLAMAT Conference,*** Tokyo, Japan. 1–25,

Potts, C.N., Sevast'Janov, S., Strusevich, V.A., Van Wassenhove, L.N., Zwaneveld, C.M., 1995. The two-stage assembly scheduling problem: complexity and approximation. ***Operations Research,*** **43** (2): 346–355.

Sang, H.-Y., Pan, Q.-K., Li, J.-Q., Wang, P., Han, Y.-Y., Gao, K.-Z., Duan, P., 2019. Effective invasive weed optimization algorithms for distributed assembly permutation flow-shop problem with total flowtime criterion. ***Swarm and Evolutionary Computation,*** **44:** 64–73.

Shoaardebili, N., Fattahi, P., 2015. Multi-objective meta-heuristics to solve three-stage assembly flow shop scheduling problem with machine availability constraints. ***International Journal of Production Research,*** **53** (3): 944–968.

Stafford, E. F., F. T. Tseng, and J. N. D. Gupta. 2005. "Comparative Evaluation of MILP Flowshop Models." ***Journal of the Operational Research Society,*** **56** (1): 88–101.

Sun, X.,Morizawa, K.,Nagasawa, H., 2003. Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling. ***European Journal of Operational Research,*** **146** (3): 498–516.

Tseng, F. T., E. F. Stafford. 2008. “New MILP Models for the Permutation Flowshop Problem.” ***Journal of the Operational Research Society,*** **59** (10): 1373–1386.

Wang, K., Huang, Y., Qin, H., 2016. A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown. ***Journal of the Operational Research Society,*** **67** (1): 68–82.

Wang, S.-Y., Wang, L., 2015. An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem. ***IEEE Transactions on Systems, Man, and Cybernetics: Systems.*** **46** (1): 139–149.

Xiong, F., Xing, K., 2014. Meta-heuristics for the distributed two-stage assembly scheduling problem with bi-criteria of makespan and mean completion time. ***International Journal of Production Research,*** **52** (9): 2743–2766.

EKLER

EK 1 — Özetlenmiş Kodlar

1.1- Parametre seçimi

```
function CHAR_ASSIGN (ind, Algorithms' parameters )
1. if character_matrix (ind, 1) == 1
2.          character_matrix (ind, 2) = round (unifrnd (GD_ITERATION_MIN, GD_ITERATION_MAX));
3.          character_matrix (ind, 3) = round (unifrnd (GD_INNER_MIN, GD_INNER_MAX));
4.          character_matrix (ind , 4) = F (ind); //Initial water level for great deluge algorithm
5.          character_matrix (ind, 5) = (F (ind) - OPTIMUM) / character_matrix (ind, 2); // Seviye azaltma
faktörü
6. endif
7. if character_matrix (ind ,1) == 2
8.          character_matrix (ind, 2) = round (unifrnd (TA_MAIN_MIN, TA_MAIN_MAX));
9.          character_matrix (ind, 3) = round (unifrnd (TA_INNER_MIN, TA_INNER_MAX));
10.     character_matrix (ind, 4) = F (ind); //Initial threshold value
11.     character_matrix (ind, 5) = unifrnd (THRESHOLDF_MIN, THRESHOLDF_MAX);
12. endif
13. if character_matrix (ind,1) == 3
14.         character_matrix (ind, 2) = round (unifrnd (GREEEDY_MIN, GREEEDY_MAX));
15. endif
16. if character_matrix (ind ,1) == 4
17.     character_matrix (ind, 2) = round (unifrnd (RANDOM_MIN, RANDOM_MAX));
18. endif
19. if character_matrix (ind,1) == 5
20.     character_matrix (ind, 2) = round (unifrnd (INIT_TEMPT_MIN, INIT_TEMPT_MAX));
21.     character_matrix (ind, 3) = round (unifrnd (SA_INNER_MIN, SA_INNER_MAX));
22.     character_matrix (ind, 4)= unifrnd (QUENCH_MIN, QUENCH_MAX);
23. endif
```

1.2- Great deluge algoritmasının temel adımları

Run great deluge algorithm ()

```
1. Call Function GD (chromosome, F (i), character_matrix (i, 4), character_matrix (i, 6), move_position_matrix
(character_matrix (i, 6), 1), BF (i));
2. if not_improving == 1
3. parameter_count (i, 2) = parameter_count (i, 2) + 1;
4. endif
5. if parameter_count (i, 2) == character_matrix (i, 3)
6.          parameter_count (i, 1) = parameter_count (i,1) + 1;
7.          character_matrix (i, 4) = character_matrix (i, 4) - character_matrix (i, 5);
8.          parameter_count (i, 2) = 0;
9. endif
10. if parameter_count (i, 1) == character_matrix (i, 2)
11.     crossover_decision (i, 1) = 1;
12.     parameter_count (i, 1) = 0;
13.     parameter_count (i, 2) = 0;
14. endif
```

function GD (*chr_c*, *obj_c*, *LEVEL*, *MOVE*, *move_location*, *best_ind_fitness*)
1. Run MOVE_x (*chr_c*, *obj_c*, *best_ind_fitness*, *move_location*)
2. *move_position_matrix (character_matrix* (*i*, 6), 1) = returned *move_position* after running the MOVE_x function
3. if *obj_n* < *obj_c*
4. *obj_c* = *obj_n*;
5. *chr_c* = *chr_n*;
6. *not_improving* = 0;
7. else
8. if *obj_n* <= *LEVEL*
9. *obj_c* = *obj_n*;
10. *chr_c* = *chr_n*;
11. *not_improving* = 0;
12. else
13. *not_improving* = 1;
14. endif
15. endif
16. *new_chromosome* = *chr_c*;
17. *f_value* = *obj_c*;
18. return *new_chromosome*, *f_value* and *not_improving*

1.3- The threshold accepting algoritmasının temel adımları

Run threshold accepting algorithm ()

1. Call Function TA (*chromosome*, *F*(*i*), *character_matrix* (*i*, 4), *character_matrix* (*i*, 6), *move_position_matrix* (*character_matrix* (*i*, 6), 1), *BF* (*i*));
2. *parameter_count* (*i*, 2) = *parameter_count* (*i*, 2) + 1;
3. if *parameter_count* (*i*, 2) == *character_matrix* (*i*, 3)
4. *parameter_count* (*i*, 1) = *parameter_count* (*i*, 1) + 1;
5. *character_matrix* (*i*, 4) = *character_matrix* (*i*, 4) * *character_matrix* (*i*, 5);
6. *parameter_count* (*i*, 2) = 0;
7. endif
8. if *parameter_count* (*i*, 1) == *character_matrix* (*i*, 2)
9. *crossover_decision* (*i*, 1) = 1;
10. *parameter_count* (*i*, 1) = 0;
11. *parameter_count* (*i*, 2) = 0;
12. endif

function TA (*chr_c*, *obj_c*, *THRESHOLD*, *MOVE*, *move_location*, *best_ind_fitness*)
1. Run MOVE_x (*chr_c*, *obj_c*, *best_ind_fitness*, *move_location*)
2. *move_position_matrix (character_matrix* (*i*, 6), 1) = returned *move_position* after running the MOVE_x function
3. Δ = *obj_n* - *obj_c*;
4. if Δ <= *THRESHOLD*
5. *obj_c* = *obj_n*;
6. *chr_c* = *chr_n*;
7. endif
8. *new_chromosome* = *chr_c*;
9. *f_value* = *obj_c*;
10. return *new_chromosome* and *f_value*

1.4- The greedy search algoritmasının temel adımları

Run greedy search algorithm ()

1. Call Function GREEDYS (*chromosome*, *F*(*i*), *character_matrix* (*i*, 6), *move_position_matrix* (*character_matrix* (*i*, 6), 1), *BF* (*i*));
2. *parameter_count* (*i*, 1)= *parameter_count* (*i*, 1) + 1;
3. if *parameter_count* (*i*, 1) == *character_matrix* (*i*, 2)
4. *crossover_decision* (*i*, 1) = 1;
5. *parameter_count* (*i*, 1) = 0;
6. endif

function GREEDYS (*chr_c*, *obj_c*, *MOVE*, *move_location*, *best_ind_fitness*)
1. Run MOVE_x (*chr_c*, *obj_c*, *best_ind_fitness*, *move_location*)
2. *move_position_matrix (character_matrix* (*i*, 6), 1) = returned *move_position* after running the MOVE_x function
3. if *obj_n* < *obj_c*
4. *obj_c* = *obj_n*;
5. *chr_c* = *chr_n*;
6. endif
7. *new_chromosome* = *chr_c*;
8. *f_value* = *obj_c*;
9. return *new_chromosome* and *f_value*

1.5- The random search algoritmasının temel adımları

Run random search algorithm ()

1. Call Function RANDOMS (*chromosome*, *F* (*i*), *character_matrix* (*i*, 6), *move_position_matrix* (*character_matrix* (*i*, 6), 1), *BF* (*i*));
2. *parameter_count* (*i*, 1) = *parameter_count* (*i*, 1) + 1;
3. if *parameter_count* (*i*, 1) == *character_matrix* (*i*, 2)
4. *crossover_decision* (*i*, 1) = 1;
5. *parameter_count* (*i*, 1) = 0;
6. endif

function RANDOMS (*chr_c*, *obj_c*, *MOVE*, *move_location*, *best_ind_fitness*)
1. Run MOVE_x (*chr_c*, *obj_c*, *best_ind_fitness*, *move_location*)
2. *move_position_matrix (character_matrix* (*i*, 6), 1) = returned *move_position* after running the MOVE_x function
3. *new_chromosome* = *chr_n*;
4. *f_value* = *obj_n*;
5. return *new_chromosome* and *f_value*

1.6- The simulated annealing algoritmasının temel adımları

Run simulated annealing algorithm ()

1. Call Function SA (*chromosome*, *F* (*i*), *character_matrix* (*i*, 2), *character_matrix* (*i*, 6), *move_position_matrix* (*character_matrix* (*i*, 6), 1), *BF* (*i*));
2. *parameter_count* (*i*, 1) = *parameter_count* (*i*, 1) + 1;
3. if *parameter_count* (*i*, 1) == *character_matrix* (*i*, 3)
4. *character_matrix* (*i*, 2) = *character_matrix* (*i*, 2) * *character_matrix* (*i*, 4);
5. *parameter_count* (*i*, 1) = 0;
6. endif
7. if *character_matrix* (*i*, 2) <= *T_MIN*
8. *crossover_decision* (*i*, 1) = 1;
9. *parameter_count* (*i*, 1) = 0;
10. endif

function SA (*chr_c*, *obj_c*, *TEMPERATURE*, *MOVE*, *move_location*, *best_ind_fitness*)
1. Run MOVE_x (*chr_c*, *obj_c*, *best_ind_fitness*, *move_location*)
2. *move_position_matrix (character_matrix* (*i*, 6), 1) = returned *move_position* after running the MOVE_x function
3. if *obj_n* < *obj_c*
4. *obj_c* = *obj_n*;
5. *chr_c* = *chr_n*;
6. else
7. Δ = *obj_n* - *obj_c*;
8. *Accept_prob* = $e^{(-\Delta/\mathrm{TEMPERATURE})}$;
9. if rand() < *Accept_prob*
10. *obj_c* = *obj_n*;
11. *chr_c* = *chr_n*;
12. endif
13. endif
14. *new_chromosome* = *chr_c*;
15. *f_value* = *obj_c*;
16. return *new_chromosome* and *f_value*

ÖZ GEÇMİŞ

1990 Van doğumlu olan Mehmet Ali ARVAS; ilk ve orta öğrenimini Van'da tamamladı. Gazi Üniversitesi Mühendislik Fakültesi, Endüstri Mühendisliği Bölümü'nden 2013 yılında mezun oldu. Üniversite eğitimi esnasında 3 ay "Work and Travel" Programı kapsamında ABD'de bulundu.

2014 - 2015 yılında Yedek Subay olarak askerlik görevini yerine getirdi. 17.03.2015 tarihinden itibaren KOSGEB Van Müdürlüğünde KOBİ Uzmanı olarak çalışmaktadır.

Eylül 2017'de Yüzüncü Yıl Üniversitesi Fen Bilimleri Enstitüsü İstatistik Anabilim Dalında Yüksek Lisans eğitimine başladı.

Evli ve bir kız çocuk babası olan Mehmet Ali ARVAS, iyi seviyede İngilizce bilmektedir.

T.C
VAN YÜZÜNCÜ YIL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
LİSANSÜSTÜ TEZ ORİJİNALLİK RAPORU

Tarih: 05/05/2020...

Tez Başlığı / Konusu: Dağıtılmış Montaj Hattı Permütasyon Akış Tipi Çizelgeleme Problemi İçin Yeni Çözüm Yöntemleri

Yukarıda başlığı/konusu belirlenen tez çalışmamın Kapak sayfası, Giriş, Ana bölümler ve Sonuç bölümlerinden oluşan toplam 61 sayfalık kısmına ilişkin, 05/05/2010 tarihinde şahsım/tez danışmanım tarafından Turnitin intihal tespit programından aşağıda belirtilen filtreleme uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 3 (Üç) dür.

Uygulanan filtreler aşağıda verilmiştir:
- Kabul ve onay sayfası hariç,
- Teşekkür hariç,
- İçindekiler hariç,
- Simge ve kısaltmalar hariç,
- Gereç ve yöntemler hariç,
- Kaynakça hariç,
- Alıntılar hariç,
- Tezden çıkan yayınlar hariç,
- 7 kelimeden daha az örtüşme içeren metin kısımları hariç (Limit inatch size to 7 words)

Van Yüzüncü Yıl Üniversitesi Lisansüstü Tez Orijinallik Raporu Alınması ve Kullanılmasına İlişkin Yönergeyi inceledim ve bu yönergede belirtilen azami benzerlik oranlarına göre tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini bilgilerinize arz ederim.

05.05.2020
Tarih ve İmza

Adı Soyadı: Mehmet Ali Arvas
Öğrenci No: 1791002114
Anabilim Dalı: İstatistik
Programı: İstatistik
Statüsü: Y. Lisans ☑ Doktora ☐

DANIŞMAN ONAYI
UYGUNDUR

Dr. Öğr. Üyesi Alper HAMZADAYI

ENSTİTÜ ONAYI
UYGUNDUR