



T.C.
KONYA TEKNİK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ



AYRIK OPTİMİZASYON PROBLEMLERİNİN
ÇÖZÜMÜ İÇİN JAYA ALGORİTMASI
TABANLI YENİ YAKLAŞIMLAR

Murat ASLAN

DOKTORA TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Aralık-2020
KONYA
Her Hakkı Saklıdır

TEZ KABUL VE ONAYI

Murat ASLAN tarafından hazırlanan “Ayrık Optimizasyon Problemlerinin Çözümü için Jaya Algoritması Tabanlı Yeni Yaklaşımlar” adlı tez çalışması 23/12/2020 tarihinde aşağıdaki jüri tarafından oy birliği ile Konya Teknik Üniversitesi Lisansüstü Eğitim Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı’nda DOKTORA TEZİ olarak kabul edilmiştir.

Jüri Üyeleri

Başkan

Prof. Dr. Halife KODAZ

Danışman

Doç. Dr. Mesut GÜNDÜZ

Üye

Doç. Dr. Mustafa Servet KIRAN

Üye

Doç. Dr. Hüseyin HAKLI

Üye

Dr. Öğr. Üyesi Ayşe Merve ACILAR

İmza

.....

.....

.....

.....

.....

Yukarıdaki sonucu onaylarım.

Prof. Dr. Saadettin Erhan KESEN
Enstitü Müdürü

TEZ BİLDİRİMİ

Bu tezdeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.



Murat ASLAN

28/12/2020

ÖZET

DOKTORA TEZİ

AYRIK OPTİMİZASYON PROBLEMLERİNİN ÇÖZÜMÜ İÇİN JAYA ALGORİTMASI TABANLI YENİ YAKLAŞIMLAR

Murat ASLAN

**Konya Teknik Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı**

Danışman: Doç. Dr. Mesut GÜNDÜZ

2020, 118 Sayfa

Jüri

**Prof. Dr. Halife KODAZ
Doç. Dr. Mesut GÜNDÜZ
Doç. Dr. Mustafa Servet KIRAN
Doç. Dr. Hüseyin HAKLI
Dr. Öğr. Üyesi Ayşe Merve ACILAR**

Jaya algoritması, kısıtlı ve kısıtsız sürekli optimizasyon problemlerinin çözümü için Rao (2016) tarafından literatüre kazandırılan popülasyon tabanlı, stokastik bir metasezgisel algoritmadır. Bu tez kapsamında ikili ve ayrık tam sayı optimizasyon problemlerinin çözümü için Jaya algoritması tabanlı yeni yaklaşımlar geliştirilmiştir. Temel Jaya algoritması sürekli optimizasyon problemlerinin çözümü için geliştirildiğinden dolayı, karar değişkenleri '0' ya da '1' değerlerini alabilen ikili optimizasyon problemlerinin üzerine doğrudan uygulanamaz. Bu kapsamda temel Jaya algoritmasının konum güncelleme mekanizmasında bazı değişiklikler yapılmış olup, ikili optimizasyon problemlerinin çözümü için Jaya algoritması tabanlı yeni yaklaşımlar geliştirilmiştir. Geliştirilen ilk yaklaşım JayaX olarak adlandırılan; temelini Jaya algoritması ve 'özel veya' (XOR) lojik fonksiyonundan alan yaklaşımdır. Diğer yaklaşım ise JayaX algoritmasının yerel arama yönünü iyileştirmek için JayaX-LSM olarak adlandırılan, önerilen JayaX algoritmasının ve LSM olarak adlandırılan yerel arama modülünün birlikte kullanılması ile geliştirilen yaklaşımdır. İkili optimizasyon problemlerinin çözümü için önerilen algoritmaların performans analizi için deney aşamasında: Kapasitesiz tesis yerleştirme problemi (KTYP), CEC 2015 nümerik fonksiyonları ve rüzgâr türbini yerleştirme problemi kullanılmıştır. İlk deneysel analiz KTYP problemi için yapılmıştır. Önerilen algoritmaların performansını analiz etmek ve doğrulamak için deneylerde 15 farklı KTYP kullanılmıştır. Önerilen algoritmalar yakın zamanda literatüre kazandırılmış PSO, ABC, TSA, DE ve GA tabanlı başarılı ikili optimizasyon algoritmalarının sonuçları ile karşılaştırılmıştır. Elde edilen deneysel sonuçlara göre, önerilen algoritmalar, KTYP'yi çözme konusunda, karşılaştırılan diğer algoritmalar ile eşit ya da daha iyi sonuçlar elde etmiştir. İkinci deneysel analiz ise 15 kıyas probleminden oluşan CEC 2015 nümerik veri seti üzerine olmuştur. Bu analizde, JayaX-LSM algoritması SabDE, BQIGSA, GBABC, BHTPSO-QI, BLDE ve SBHS algoritmalarının sonuçları ile karşılaştırılmıştır ve elde edilen deneysel sonuçlar dikkate alındığında JayaX-LSM algoritması, karşılaştırılan algoritmalar ile rekabetçi ya da daha iyi çözümler elde etmiştir. Bu bölümde yapılan son deneysel analiz ise rüzgâr türbini yerleştirme problemi için yapılmıştır. Deneylerde 10×10 ve 20×20 'lik olmak üzere iki farklı ızgara yapısı kullanılmıştır. Deneysel sonuçlara göre, JayaX-LSM algoritması karşılaştırmalarda kullanılan GA tabanlı ikili yöntemler, BIWO, BPSO-TVAC, EA, NGHS, DGHS ve binAAA algoritmalarına benzer ya da daha iyi çözümler üretmiştir.

Geliştirilen bir diğer yöntem ise ayrık tam sayı optimizasyon problemlerinin çözümü için Jaya algoritması tabanlı DJaya olarak adlandırılan Ayrık Jaya algoritmasıdır. Temel Jaya algoritmasının

ayırıklaştırma işlemi için güncelleme mekanizmasında takas, öteleme ve simetri olarak adlandırılan komşuluk operatörleri kullanılmıştır. DJaya’da başlangıç popülasyonu oluşturulurken (N-1) tane aday çözüm rastgele permütasyon ile oluşturulurken, ilk aday çözüm en yakın komşu turu sezgiseli ile oluşturulmaktadır. Ayrıca DJaya’nın elde ettiği çözümlerin kalitesinin artırılması amacıyla 2-opt yerel arama algoritması da kullanılmıştır. DJaya algoritmasının başarısını ve etkinliğini göstermek amacıyla, deneylerde 14 farklı gezgin satıcı problemi (GSP) kullanılmıştır. Elde edilen deneysel sonuçlar yakın zamanda literatüre kazandırılan başarılı algoritmaların sonuçları ile karşılaştırılmıştır. Deneysel bulgulara göre, DJaya algoritması gezgin satıcı probleminin çözümü için karşılaştırılan algoritmalarından daha başarılı ya da rekabetçi çözümler elde etmiştir.

Anahtar Kelimeler: ayırık optimizasyon, gezgin satıcı problemi, ikili optimizasyon, Jaya algoritması, kapasitesiz tesis yerleştirme problemi, komşuluk operatörleri, rüzgâr türbini yerleştirme problemi, sürü zekâsı



ABSTRACT

PhD THESIS

JAYA ALGORITHM BASED NEW APPROACHES FOR SOLVING DISCRETE OPTIMIZATION PROBLEMS

Murat ASLAN

**Konya Technical University
Institute of Graduate Studies
Department of Computer Engineering**

Advisor: Assoc. Prof. Dr. Mesut GÜNDÜZ

2020, 118 Pages

Jury

**Prof. Dr. Halife KODAZ
Assoc. Prof. Dr. Mesut GÜNDÜZ
Assoc. Prof. Dr. Mustafa Servet KIRAN
Assoc. Prof. Dr. Hüseyin HAKLI
Asst. Prof. Dr. Ayşe Merve ACILAR**

Jaya algorithm is a stochastic population-based heuristic optimization algorithm proposed by Rao (2016) for solving constrained and unconstrained continuous optimization problems. In this study, Jaya algorithm based new approaches have been proposed for solving binary and discrete integer optimization problems. Because of the basic Jaya algorithm proposed for solving continuous optimization problems, it cannot be directly applied to binary optimization problems, because the solution space is discretely structured for this type of optimization problems and the decision variables of the binary optimization problems can be element of set [0,1]. In this context, some modifications have been made in update mechanism of the basic Jaya algorithm, and Jaya algorithm based new approaches have been proposed for solving binary optimization problems. The first proposed approach is JayaX; It is based on Jaya algorithm and the 'exclusive or' (XOR) logic function. The second approach is proposed in order to improve the local search capability of the JayaX algorithm, and it is called JayaX-LSM. JayaX-LSM is based on JayaX algorithm and local search module (LSM). For the performance analysis of the proposed algorithms to solve binary optimization problems, in the experimental phase: Uncapacitated facility location problem (UFLP), CEC 2015 numerical functions and wind turbine placement problem are used. The first experimental analysis is made for UFLP problems. In order to examine and approve the performance of the proposed algorithms, 15 different UFLPs are used in the experiments. The proposed algorithms have been compared with the state-of-art binary optimization algorithms based on PSO, ABC, TSA, DE and GA which have been recently proposed into the literature. According to the experimental results, the proposed algorithms obtained equal or better performance than the compared algorithms in solving UFLP. The second experimental analysis is performed on the CEC 2015 numerical benchmark set consisting of 15 problems. In this analysis, the JayaX-LSM algorithm is compared with the experimental results of the SabDE, BQIGSA, GBABC, BHPSO-QI, BLDE and SBHS algorithms, and considering the experiments, the JayaX-LSM algorithm obtained competitive or better solutions with the compared algorithms. The last experimental analysis in this section is made for the wind turbine placement problem. Two different grid structures, 10×10 and 20×20 are used in the experiments. According to the experimental results, the JayaX-LSM algorithm get equal or better solutions than GA-based binary methods, BIWO, BPSO-TVAC, EA, NGHS, DGHS and binAAA algorithms used in comparisons.

Another proposed method is a Jaya based discrete algorithm called as DJaya for the solution of discrete integer optimization problems. Swap, shift and symmetry transformation operators are being used for discretization process of the update mechanism of basic Jaya. While creating the initial population in

DJaya, (N-1) candidate solution is created by random permutation, and one candidate solution is created with the nearest neighbor tour heuristic. Furthermore, 2-opt local search algorithm has been used for improve the quality of solutions in DJaya. In order to examine and confirm the performance of the proposed algorithms, 14 different travelling salesman problems (TSP) are used in the experiments. The proposed algorithm has been compared with the state-of-art optimization algorithms. According to the experimental result, the DJaya algorithm has obtained better or competitive solutions than the compared algorithms for solving the traveling salesman problem.

Keywords: binary optimization, discrete optimization, Jaya algorithm, swarm intelligence, transformation operators, traveling salesman problem, uncapaciated facility location problem, wind turbine placement problem.



ÖNSÖZ

Doktora tez çalışmam boyunca üzerimde emeği olan, gösterdiği katkı ve yardımlarıyla tez çalışmasına yön veren değerli danışman hocam Sayın Doç. Dr. Mesut GÜNDÜZ'e, tez çalışması boyunca yapıcı fikirleri ile tez çalışmasına yön veren doktora tez izleme kurulu üyelerinden Sayın Doç. Dr. Mustafa Servet KIRAN hocama ve tez izleme kurulları boyunca tez çalışmasına yapıcı eleştirileri ile katkı sunan Sayın Dr. Öğr. Üyesi Ayşe Merve ACILAR hocama teşekkür etmeyi bir borç bilirim.

Çalışmalarım boyunca maddi ve manevi desteklerini benden asla esirgemeyen, hayatım boyunca bana ve yaptığım çalışmalara inanan, üzerimde büyük hakları olan babam Salahattin ASLAN'a ve annem Hayriye ASLAN'a teşekkür etmeyi bir borç bilirim. Bununla birlikte, her zaman yanımda olan ve çalışmalarımı destekleyen kardeşlerimin her birine en içten teşekkürlerimi sunarım. Ayrıca, her zaman ilgi, anlayış ve desteğiyle yanımda olan eşim Fatma ASLAN'a teşekkür ederim. Tez çalışmasını sevgili kardeşim Gülistan ASLAN'a ithaf ediyorum.

Murat ASLAN
KONYA-2020

İÇİNDEKİLER

ÖZET	iv
ABSTRACT.....	vi
ÖNSÖZ	viii
İÇİNDEKİLER	ix
KISALTMALAR	xi
1. GİRİŞ	1
2. KAYNAK ARAŞTIRMASI	7
2.1. İkili Optimizasyon Problemlerinin Çözümü için Yapılan Çalışmalar	8
2.1.1. Rüzgâr türbini yerleştirme probleminin çözümü için yapılan çalışmalar	17
2.2. Ayırık Tam Sayı Optimizasyon Problemlerinin Çözümü için Yapılan Çalışmalar	20
2.3. Jaya Algoritması ile İlgili Yapılan Çalışmalar	24
3. MATERYAL VE YÖNTEM.....	28
3.1. Optimizasyon Algoritmaları	28
3.1.1. Temel Jaya algoritması	28
3.1.2. İkili Jaya algoritması (BJA)	30
3.2. İkili Optimizasyon Problemlerinin Çözümü için Önerilen Algoritmalar	31
3.2.1. XOR tabanlı ikili Jaya algoritması (JayaX)	31
3.2.2. Yerel arama modülü eklenmiş ikili Jaya algoritması (JayaX-LSM)	34
3.3. Ayırık Tam Sayı Optimizasyon Problemlerinin Çözümü için Önerilen Jaya Algoritması (DJaya)	39
3.4. Optimizasyon Problemleri	43
3.4.1. Kapasitesiz tesis yerleştirme problemi (KTYP)	43
3.4.2. Sürekli nümerik fonksiyonlar: CEC 2015 veri seti	45
3.4.3. Rüzgâr türbini yerleştirme problemi	47
3.4.4. Gezgin satıcı problemi (GSP)	52
4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA.....	55
4.1. Kapasitesiz Tesis Yerleştirme Probleminin Çözümü için Deneysel Sonuçlar	55
4.1.1. KTYP'nin çözümü için önerilen algoritmanın parametre analizi	55
4.1.2. KTYP'nin çözümü için algoritmaların performans analizi	57
4.2. CEC 2015 Nümerik Fonksiyonlarının Çözümü için Deneysel Sonuçlar	72
4.3. Rüzgâr Türbini Yerleştirme Probleminin Çözümü için Deneysel Sonuçlar	75
4.3.1. Rüzgâr türbini yerleştirme probleminin çözümü için önerilen algoritmanın parametre analizi	77
4.3.2. Rüzgâr türbini yerleştirme probleminin çözümü için algoritmaların performans analizi	81
4.4. Gezgin Satıcı Probleminin Çözümü için Deneysel Sonuçlar	85

4.4.1. GSP'nin çözümü için önerilen algoritmanın parametre analizi.....	86
4.4.2. GSP'nin çözümü için algoritmaların performans analizi	94
5. SONUÇLAR VE ÖNERİLER.....	103
5.1 Sonuçlar	103
5.2 Öneriler	107
KAYNAKLAR	108
ÖZGEÇMİŞ	117



KISALTMALAR

AAA	: Artificial Algae Algorithm
ABC	: Artificial Bee Colony
ACS	: Ant Colony System
ACO	: Ant Colony Optimization
ADP	: Approximate Dynamic Programming
AFSA	: Artificial Fish Swarm Algorithm
ANN	: Artificial Neural Network
BA	: Bat Algorithm
BAAA	: Binary AAA
bAFSA	: Binary AFSA
BBA-LSS	: LSS based Binary BA
BHTPSO-QI	: Binary Hybrid Topology PSO Quadratic Interpolation
binABC	: Binary ABC
binDE	: Binary DE
BinSSA	: Binary Social Spider Algorithm
BIWO	: Binary Invasive Weed Optimization
BH	: Black Hole
BJA	: Binary Jaya
BLDE	: Binary Learning Differential Evolution
BPSO	: Binary PSO
BPSO-TVAC	: TVAC based BPSO
BQIGSA	: Binary Quantum-Inspired GSA
BSSA	: Binary Salp Swarm Algorithm
BTP	: Birim Taahhüt Problemi
CAB	: Civil Aeronautics Board
CABC	: Combinatorial ABC
CEC	: Congress on Evolutionary Computation
CS	: Cuckoo Search
CSGHS	: Hybrid Cuckoo Search and Global Harmony Search
CVRP	: Capacitated Vehicle-Routing Problem
CX2	: Modified Cycle Crossover
DE	: Differential Evolution
DGHS	: Discrete Global Harmony Search
DHJaya	: Hybrid Jaya algorithm with double coding
DisABC	: Dissimilarity-based ABC
DJaya	: Discrete Jaya
DPSO	: Discrete PSO
DSTA	: Discrete STA
DTSA	: Discrete TSA

EA	: Evolutive Algorithm
EO-Jaya	: Elite Opposition-based Jaya
ERX	: Edge Recombination Crossover
ESA	: Evolutionary Simulated Annealing
GA	: Genetic Algorithm
GBABC	: GA based Binary Artificial Bee Colony
GHS	: Global Harmony Search
GNX	: N-Point Crossover
GPU-Jaya	: GPU-Accelerated Parallel Jaya
GSA	: Gravitational Search Algorithm
GSP	: Gezgin Satıcı Problemi
HA	: Hierarchic Approach
ISS	: Improved Scatter Search
JAYALF	: Lévy Fligh and Jaya based hybrid algorithm
JayaX	: XOR based Binary Jaya
JayaX-LSM	: XOR and LSM based Binary Jaya
KP	: Knapsack Problem
KNN	: K-Nearest Neighbor
KTYP	: Kapasitesiz Tesis Yerleştirme Problemi
LogicTSA	: Logic based TSA
LF	: Lévy Fligh
LSM	: Local Search Module
LSS	: Local Search Scheme
MaxFEs	: Maximum function evaluation number
MBPSO	: Modified Binary Particle Swarm Optimization
MDisABC	: Modified dissimilarity-based ABC
MKP	: Multidimensional knapsack problem
MopGA	: Monogamous Pairs Genetic Algorithm
MPGA	: Multi-Population GA
MO-Jaya	: Multi-Objective Jaya
NB	: Net Benefit
NBBA	: Binary BA
NGHS	: Novel Global Harmony Search Algorithm
NSJaya	: Non-dominated Sorting Jaya
OPPP	: Optimal Placement of PMUs
PACO	: Parallel Cooperative Hybrid Algorithm
PACO-3opt	: Parallel Cooperative Hybrid Algorithm with 3opt
PMUs	: Phasor Measurement Units
PSO	: Particle Swarm Optimization
qABC	: quick ABC
SA	: Simulated Annealing

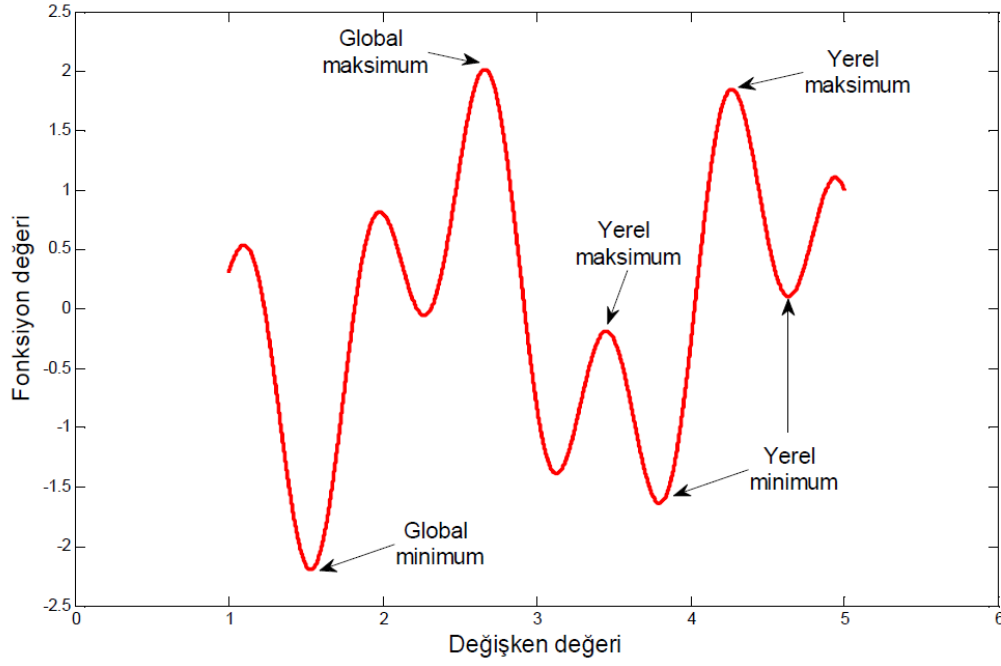
SabDE	: Self-adaptive Binary DE
SAMPE-Jaya	: Elitist-based Self-Adaptive Multi-Population Jaya
SBHS	: Simplified Binary Harmony Search
SCX	: Sequential Constructive Crossover
SFLA	: Shuffled Frog-Leaping Algorithm
SimLogicTSA	: Similarity and Logic based TSA
SimTSA	: Similarity based TSA
SS	: Scatter Search
SSA	: Social Spider Algorithm
ST	: Search Tendency
STA	: State Transition Algorithm
SUKP	: Set-Union Knapsack Problem
SVM	: Support Vector Machine
S-bAFSA	: Simplified bAFSA
TLBO	: Teaching-Learning Based Optimization
TS	: Tabu Search
TSA	: Tree-Seed Algorithm
TSP	: Traveling Salesman Problem
TVAC	: Time Varying Acceleration Coefficients
UFLP	: Uncapacitated Facility Location Problem
WCA	: Wolf Colony Algorithm

1. GİRİŞ

Gerçek dünyada birçok farklı problem ile karşılaşilmektedir ve bu problemler için en uygun çözümün ne olduğu, maliyetin artması ve eldeki kaynakların kısıtlı hale gelmesi durumuna bağlı olarak önem kazanmaktadır. Bu nedenle bir problem için en uygun sonuca ulaşma isteği bir optimizasyon problemi olarak karşımıza çıkmaktadır. Bir problemin optimizasyon problemi olarak tanımlanması demek en az maliyet ile maksimum verim elde etme isteği demektir. Sonuç olarak, birçok gerçek dünya problemi optimizasyon problemi olarak tanımlanabilmekte ve bu problemlerin çözümü için literatürde birbirinden farklı çözüm yöntemleri önerilmektedir. Optimizasyon problemi olarak tanımlanan bir problem, matematiksel olarak modellenabilmektedir. Bu matematiksel model, problemi çözmek için ihtiyaç duyulan amaç fonksiyonunu (objective function) temsil etmektedir. Matematiksel modelin değişkenleri ise problemin karar değişkenlerinden (decision variables) oluşmaktadır. Matematiksel modeli elde edilmiş bir problem için amaç, karar değişkenlerinin alacağı değerlere göre ya genel en büyük (global maksimum) ya da genel en küçük (global minimum) değer elde edilmesidir. Eğer amaç global minimum sonucu elde etmekse optimizasyon problemi minimizasyon problemi olarak tanımlanmakta, değilse optimizasyon problem maksimizasyon problemi olarak tanımlanmaktadır. Yaman (2014), yapmış olduğu çalışmada Denklem 1.1’de verilmiş olan optimizasyon problemi üzerinde global maksimum, yerel (lokal) maksimum, global minimum ve yerel (lokal) minimum noktaları göstermiştir (Yaman, 2014).

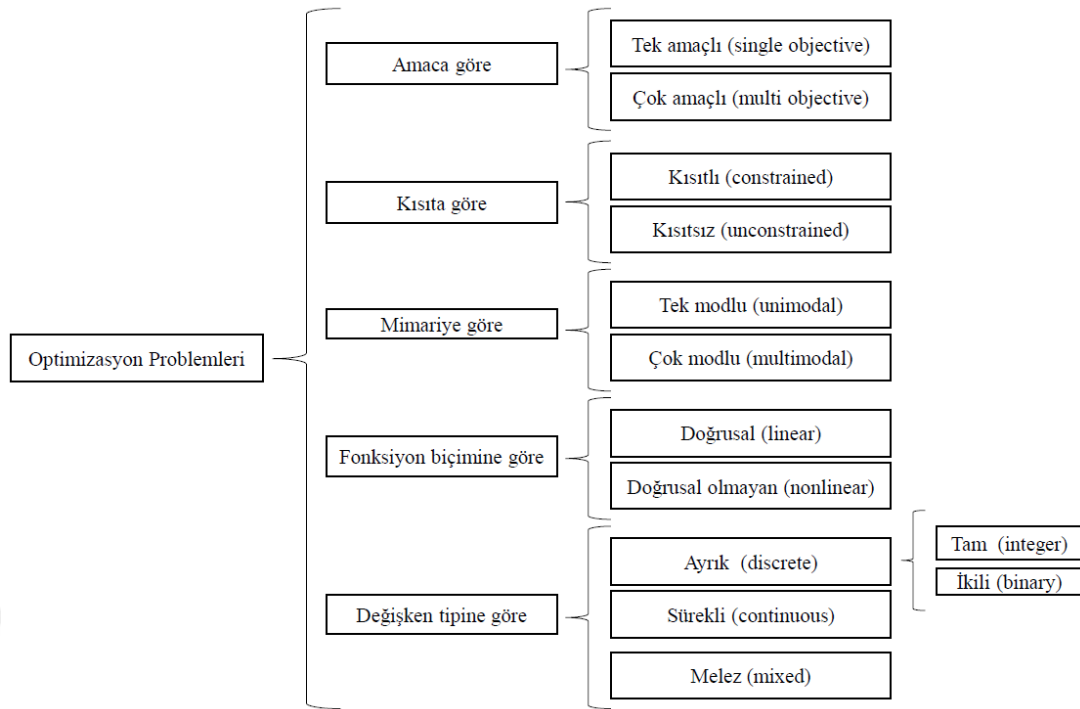
$$f(x) = 1.2 \sin(\pi x) - \cos(2.6\pi x), \quad 1 \leq x < 5 \quad (1.1)$$

Denklem 1.1’de görüldüğü gibi sadece 1 tane karar değişkeni vardır. Burada $f(x)$, optimizasyon problemini, $[1,5]$ tanım aralığı ise karar değişkeni x ’in alabileceği değerler kümesini göstermektedir. Şekil 1.1’de ise farklı x değerleri için $f(x)$ fonksiyonun aldığı değerler gösterilmektedir. Şekil 1.1’de de görüldüğü üzere karar değişkeninin aldığı değerlere göre global maksimum, yerel maksimum, global minimum ve yerel minimum gösterilmektedir. Yerel maksimum ve yerel minimum değerler tanım kümesinin belirli bir aralığındaki en yüksek ve en düşük değerleri göstermektedir.



Şekil 1.1. $f(x)$ fonksiyonunun maksimum ve minimum noktaları (Yaman, 2014)

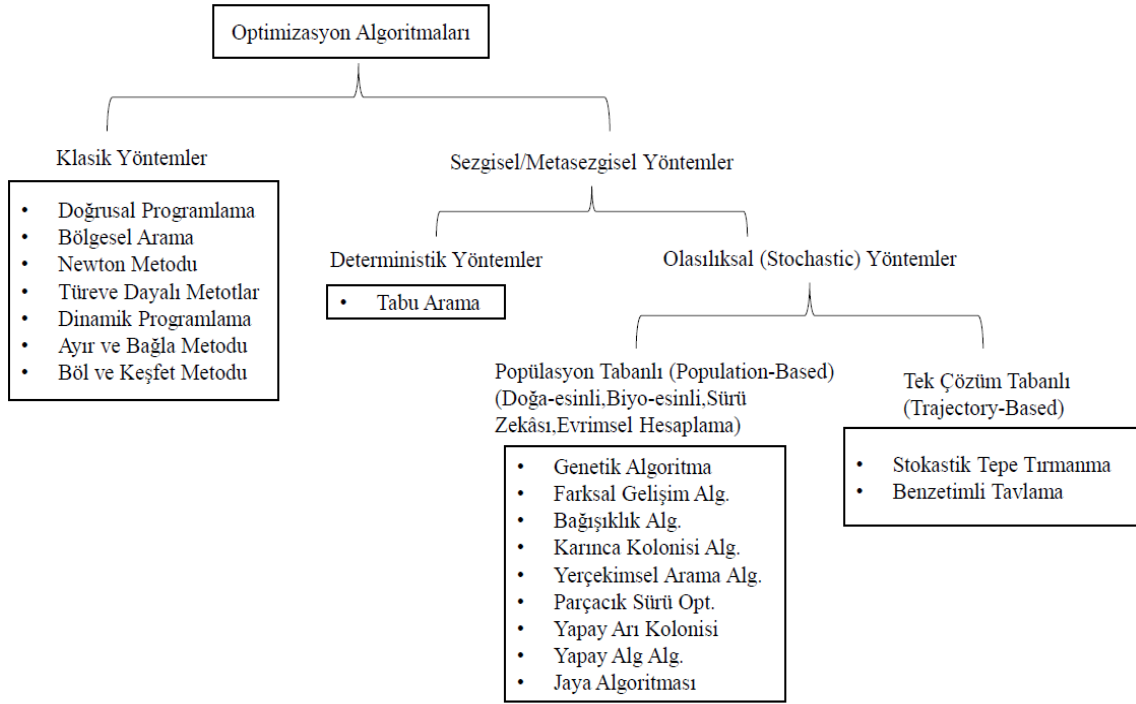
Karar değişkenlerinin sayısının artması ya da problemin boyutunun artması, problemin polinomsal zamanda çözümünü zorlaştırmaktadır. Denklem 1.1’de verilen problem tek boyutlu bir problem olmasına rağmen, tanım aralığında tüm reel değerleri alacağından bu problem için en iyi sonucu verecek değeri bulmak oldukça zor olmaktadır. Rüzgâr Türbini Yerleştirme Problemi (Wind Turbine Placement Problem), Kapasitesiz Tesis Yerleştirme Problemi (KTYP) (Uncapaciated Facility Location Problem - UFLP), Gezgin Satıcı Problemi (GSP) (Traveling Salesman Problem - TSP), 0-1 Sırt Çantası Problemi (SÇP) (Knapsack Problem - KP) gibi problemler üstel zamanda çözümü zor (NP-Zor) olan problemler kategorisine girmektedir. Örneğin KTYP’de amaç en az kurulum ve ulaşım maliyeti ile tüm müşterilere hizmet vermektir. Bazen sadece bir müşteriye hizmet verebilmek için bile yeni bir tesis kurulması gerekebilmektedir. Bu nedenle müşteri sayısının ve tesis sayısının artması ile birlikte problemin çözümü de zorlaşmaktadır. Şekil 1.2’de optimizasyon problemleri yapılarına göre sınıflandırılmıştır. Şekil 1.2 incelendiğinde optimizasyon problemleri; amaca, kısıta, mimariye, fonksiyon biçimine ve değişken tipine göre olmak üzere 5 alt kategoriye ayrılmaktadır (Yang, 2010; Korkmaz, 2019).



Şekil 1.2. Türlerine göre optimizasyon problemleri (Yang, 2010; Korkmaz, 2019)

Her optimizasyon problemi aynı karakteristik yapıda olmadığı için, bu problemlerin çözümü için tek bir yöntem ile en iyi sonuca ulaşılması beklenemez. Problemin doğasına uygun bir yöntem ile problemin çözülmesi gerekmektedir. Optimizasyon problemlerinin çözümü için literatürde birçok farklı yöntem önerilmiştir. Bu yöntemler kesin (exact), tahmini (approximate), yerel arama (local search) ve metasezgisel tabanlı algoritmalarıdır. Optimizasyon problemi olarak tanımlanan bir problemin çözümü için sezgisel algoritmalar mevcut bilgileri kullanarak bütün çözümler arasında en iyi çözüme ulaşmaya çalışmaktadır. NP-Zor tipi karmaşık problemlerin çözümü için lineer ve dinamik programlama tabanlı algoritmalar optimum çözüme ulaşmayı garanti etmezler (Jaballah ve ark., 2014). Bu nedenle, karmaşık problemlerin çözümü için üstel bir zamanda güçlü bir hesaplamaya ihtiyaç vardır (Luo ve Chen, 2014). Metasezgisel algoritmalar eldeki çözümlerin rehberliğinde makul bir sürede kesin çözüme yakın bir çözüme ulaşmayı garanti etmektedirler (Karaboğa, 2011). Metasezgisel yöntemler; doğa esinli, sosyal davranışlara dayanan ya da müzik tabanlı olabilirler. Bunun yanında algoritmalarından bazılarının birlikte kullanılması ile oluşturulmuş hibrit algoritmalar da vardır (Alataş, 2007). Ancak tüm yöntemlerin amacı, eldeki mevcut bilgileri en iyi şekilde kullanarak, kabul edilebilir çözümlere ulaşmaktır. Metasezgisel yöntemlerin en önemli faydalarından birisi gerçek dünya problemlerinin çözümü için öğrenme ve adaptasyon tekniklerini kullanması sayesinde kesin çözüme yakın çözümlere

olan yakınsama hızlarıdır (Lovbjerg, 2002). Literatürde yapılan çalışmalar dikkate alındığında optimizasyon algoritmaları klasik yöntemler ve sezgisel/metasezgisel yöntemler olmak üzere 2 ana alt başlık altında toplanabilir. Şekil 1.3'te optimizasyon algoritmaları yapılarına göre sınıflandırılmıştır (Akay, 2009; Yang, 2010; Kıran, 2014; Korkmaz, 2019).



Şekil 1.3. Yapılarına göre optimizasyon algoritmaları (Akay, 2009; Yang, 2010; Kıran, 2014; Korkmaz, 2019)

Optimizasyon algoritmaları; sürü zekâsına dayalı, popülasyon temelli, fizik, kimya, biyolojiye dayalı, sosyal davranışlara dayalı ya da müzik temelli olabilmektedirler (Alataş, 2007; Aslan, 2017). Literatürde çok sayıda sezgisel tabanlı yöntem önerilmiştir. Bunun en önemli sebebi her problemi çözecek kesin bir yöntemin olmamasıdır. Olasılık tabanlı yöntemler, ilham aldıkları doğa olayını taklit etmektedir. Bundan dolayı da bazı algoritmaların doğası bazı problemlerin çözümü için çok iyi sonuçlar vermektedir. Bununla birlikte bir problem için çok iyi çözüm üreten bir algoritma, başka bir problem için en kötü sonuca ulaşabilmektedir (Korkmaz, 2019). Algoritmaların her problem için bir birinden farklı tepkiler vermesini Wolpert ve Macready (1997) “no free lunch” teorimi ile açıklamıştır (Wolpert ve Macready, 1997). Bu teoreme göre, eğer A yöntemi bazı problemler için B yöntemine kıyasla daha iyi sonuçlar elde etmişse, bu durumda B yönteminin diğer problemler için A yöntemine göre daha iyi sonuçlar elde edebileceğini

belirtilmişlerdir (Yang, 2010; Korkmaz, 2019). Literatürde önerilmiş olan doğa esinli optimizasyon algoritmaları genellikle sürekli optimizasyon problemlerinin çözümü için önerilmişlerdir. Bu algoritmaların bir çoğu araştırmacılar tarafından ayrık optimizasyon problemlerinin çözümü için de uyarlanmaktadır. Optimizasyon problemleri karar değişkenlerinin aldıkları değerlere göre sürekli ya da ayrık problemler olarak tanımlanmaktadır (Gould, 2006). Denklem 1.1’de verilen $f(x)$ fonksiyonu tanım aralığında sonsuz değer alabildiği için sürekli bir problemdir. Ayrık optimizasyon problemleri ise sürekli optimizasyondan farklı olarak tanım aralığındaki tam sayı değerlerini almaktadırlar. Örneğin GSP’de, tüccar şehirleri tek tek gezmektedir. Bundan dolayı GSP’de her şehir bir tam sayı ile ifade edilerek, bu problem ayrık bir optimizasyon problemi olarak modellenmektedir. İkili optimizasyon ise ayrık optimizasyonun bir parçasıdır. İkili optimizasyonda karar değişkenleri ‘0’ ya da ‘1’ değerini almaktadır. ‘0’ değeri var olmama anlamına gelirken, ‘1’ değeri var olma anlamına gelmektedir. Rüzgâr türbini yerleştirme problemi için ‘0’ değeri ilgili alana türbin yerleştirilmeyeceğini, ‘1’ değeri ise türbin yerleştirileceğini göstermektedir. Sürekli optimizasyon algoritmalarında konum güncelleme aşamasında elde edilen aday çözümün tanımlanan sınır değer aralığında olup olmadığı önemlidir. Kombinatoriyal optimizasyon problemlerinde ise aday çözüm tam sayı değerlerinin bir permütasyonuna dayalıdır. Örneğin GSP’de gezgin satıcı her şehre sadece bir kez uğramak zorundadır. Konum güncelleme aşamasında, oluşturulan aday çözümde aynı şehir tekrarlanmamalıdır. Bundan dolayı da eğer konum güncelleme işlemi boyutların yer değiştirmesi olarak yapılmamışsa, oluşturulan aday çözümde tekrar eden şehirler için ayrıca bir kontrol mekanizması kullanılması ve aday çözümün makul bir çözüm olması sağlanmalıdır (Krause ve ark., 2013).

Bu tez çalışması kapsamında ayrık optimizasyon problemlerinin çözümü için Jaya algoritmasına dayalı yeni yöntemler geliştirilmiştir. Jaya algoritması, kısıtlı ve kısıtsız sürekli optimizasyon problemlerinin çözümü için Rao (2016) tarafından literatüre kazandırılan popülasyon tabanlı bir algoritmadır (Rao, 2016). Jaya algoritmasının en önemli özelliklerinden biri parametre bağımsız olmasıdır. Bunun sayesinde optimizasyon problemlerine kolay bir şekilde uyarlanabilmektedir. Jaya, algoritmasının bir diğer önemli özelliği ise, çalışma yapısı belirli bir problemin en iyi çözüme doğru hareket etme eğilimini arttırırken, aynı zamanda en kötü çözümden kaçarak, yerel en kötü çözüme takılmayı önlemek üzerinedir. Bu nedenle hem popülasyondaki en iyi çözümü hem de en kötü çözümü konum güncelleme mekanizmasında kullanmaktadır. Bu da çözüm arama

uzayında daha geniş bir arama yapmayı sağlamaktadır. Jaya algoritması sürekli optimizasyon problemlerinin çözümü için geliştirildiği için ayrık optimizasyon problemleri üzerine doğrudan uygulanamaz. Bundan dolayı da algoritmanın saf haline yenilikler eklenmesi ve ayrık optimizasyon problemlerinin çözümü için uygun hale getirilmesi gerekmektedir. Bu tez çalışması kapsamında ikili Jaya (JayaX), yerel arama modülü eklenmiş ikili Jaya (JayaX-LSM) ve ayrık tam sayılı problemlerinin çözümü için geliştirilmiş olan Ayrık Jaya (DJaya) algoritmaları geliştirilmiştir. İkili optimizasyon problemlerinin çözümü için geliştirilmiş olan Jaya tabanlı yaklaşımlar, KTYP, CEC 2015 numerik optimizasyon problemleri ve rüzgâr türbini yerleştirme problemi üzerine uygulanmıştır. Ayrık tam sayılı problemlerinin çözümü için geliştirilmiş olan Jaya tabanlı yaklaşım ise GSP üzerine uygulanmıştır.

Tez çalışmasının ikinci bölümünde ayrık optimizasyon problemlerinin çözümü için kaynak araştırması yapılmıştır. Tez çalışması ayrık optimizasyon problemlerinin çözümü üzerine yoğunlaştığı için daha çok ikili optimizasyon problemleri ve ayrık tam sayı optimizasyon problemlerinin çözümü için literatürde yapılan çalışmalara ağırlık verilmiştir. Ayrıca tez çalışmasında kullanılan Jaya algoritması ile ilgili literatürde yapılan bazı çalışmalar da bu bölümde verilmiştir.

Çalışmanın üçüncü bölümünde ise hem deneylerde kullanılan problemler detaylandırılmış hem de önerilen Jaya tabanlı yaklaşımlar detaylandırılmıştır. Tez çalışmasında kullanılan problemler KTYP, CEC 2015 nümerik fonksiyonları, rüzgâr türbini yerleştirme problemi ve GSP'dir. Bu problemler ile ilgili tanımlamalar ve problemlerin yapıları detaylandırılmıştır. Bununla birlikte temel Jaya algoritması ve önerilen Jaya algoritması tabanlı yaklaşımların çalışma adımları detaylandırılmıştır.

Çalışmanın dördüncü bölümünde, önerilen algoritmalar için parametre analizleri yapılmıştır. Daha sonra önerilen algoritmaların deneysel sonuçları literatürde bilinen iyi algoritmaların sonuçları ile kıyaslanmıştır. Deneysel sonuçlar yorumlanmış ve değerlendirmeler yapılmıştır.

Çalışmanın son bölümünde ise tez çalışmasından elde edilen sonuçlar ve öneriler sunulmuştur.

2. KAYNAK ARAŞTIRMASI

Optimizasyon algoritmalarının amacı, mevcut problem için bilinen bilgileri kullanarak arama uzayında (tüm çözümler uzayı) mümkün olan en iyi çözüme ulaşmaktır. Optimizasyon problemi, bazı kısıtların aşılmaması şartıyla mevcut bilgilerin ışığında arama uzayında bilinmeyene ulaşma isteği olarak tanımlanabilmektedir (Murty, 2003). Bu tez kapsamında ikili ve ayrık tam sayı tabanlı optimizasyon problemlerinin çözümü için metasezgisel bir algoritma olan popülasyon tabanlı Jaya algoritmasının farklı versiyonları geliştirilmiştir. Optimizasyon problemlerinin çözümünde kullanılan metasezgisel algoritmalar üzerinde çalışılan problem için kesin çözüme ulaşmayı garanti etmezler. Bununla birlikte makul bir sürede arama uzayındaki kesin çözüme yakın bir çözüme ulaşmayı hedeflemektedirler (Karaboğa, 2011). Optimizasyon problemlerinin çözümü için John Holland (1975) tarafından literatüre kazandırılan ilk algoritmalarından biri olan GA, doğası gereği hem sürekli hem de ayrık optimizasyon problemlerin üzerine doğrudan uygulanabilmektedir (Holland, 1975). GA yapısındaki çaprazlama ve mutasyon operatörleri sayesinde, bünyesinde rastsallık taşımaktadır. Bu durum arama uzayında karmaşık problemler için geniş bir arama yapılmasını sağladığı için GA'nın yerel minimum ya da yerel maksimum çözüme takılmadan iyi çözümler elde etmesini sağlamaktadır (Çunkaş, 2006). GA'nın makul bir sürede optimizasyon problemlerini çözmedeki başarısı sayesinde, GA tabanlı bir çok farklı optimizasyon algoritması literatüre kazandırılmıştır. GA'nın çaprazlama ve mutasyon operatörleri bir çok sürekli optimizasyon algoritmasının ayrıklaştırması sürecinde de kullanılmaktadır. Parçacık Sürü Optimizasyonu (Particle Swarm Optimization - PSO) (Eberhart ve Kennedy, 1995), Kurbağa Sıçrama Algoritması (KSA) (Shuffled Frog-Leaping Algorithm - SFLA) (Eusuff ve Lansey, 2003), Yapay Arı Kolonisi (YAK) (Artificial Bee Colony - ABC) (Karaboğa, 2011), Kurt Kolonisi Algoritması (KKA) (The Wolf Colony Algorithm - WCA) (Liu ve ark., 2011), Yer Çekimi Arama Algoritması (Gravitational Search Algorithm - GSA) (Rashedi ve ark., 2009), Yapay Alg Algoritması (YAA) (Artificial Algae Algorithm - AAA) (Uymaz, 2015), Jaya Algoritması (Rao, 2016), Ağaç Tohum Algoritması (ATA) (Tree-Seed Algorithm - TSA) gibi sürekli optimizasyon algoritmaları ayrık optimizasyon problemlerinin çözümü için doğrudan kullanılamamaktadır. Bu tür algoritmaların ayrık optimizasyon problemlerinin üzerine uygulanabilmesi için algoritmaların çalışma yapılarında bazı değişikliklerin yapılması gerekmektedir. Bu kapsamda tez çalışmasında kullanılan Jaya algoritması üzerinde bazı değişiklikler (modifikasyonlar) yapılmıştır.

Gerçek dünya problemlerinde maliyetin artması ve kaynakların kısıtlı olması beraberinde bu problemlerin optimize edilme isteğini daha da ön plana çıkarmıştır. Örneğin fosil-petrol kaynaklarının azalması, bu yakıtların oluşturduğu hava kirliliği, sera gazı etkisi ve iklim değişikliği gibi doğadaki birçok farklı değişiklik yenilenebilir bir enerji kaynağı olan rüzgâr enerjisine olan gereksinimin artmasına neden olmuştur (Bilgen ve ark., 2004; Omer, 2008; Shoaib ve ark., 2019). Rüzgâr enerjisi gibi yenilenebilir bir enerjiden en az maliyet ile maksimum verim elde etme isteği de karşımıza bir optimizasyon problemi olarak ortaya çıkmaktadır. Bir diğer önemli problem de tesis yerleştirme problemidir. Günümüzde müşteri potansiyeli her geçen gün artmaktadır. Bir müşteri bir ürüne en ucuz ve en kısa sürede ulaşmak isterken, hizmet verici ise müşterinin isteğini en az maliyetle karşılamak istemektedir. Hem müşterinin hem de hizmet verenin memnun edilebilmesi için bu problemin en optimum çözümünün elde edilmesi gerekmektedir. Ya da GSP’de, bir gezgin tüm şehirleri en kısa mesafede ziyaret etmek istemektedir. Bu problemler için eldeki kısıtlı imkanlar ile en iyi çözümlere ulaşılması için bu problemlerin optimizasyon problemi olarak tanımlanması ve çözülmesi gerekmektedir.

Tezin amacı ikili ve ayrık tam sayı optimizasyon problemlerinin çözümü için ikili ve ayrık Jaya algoritmalarının geliştirilmesidir. Bu kapsamda kaynak araştırması üç alt başlığa ayrılmıştır. Bölüm 2.1’de ikili optimizasyon problemlerinin çözümü için literatürde yapılan çalışmalar verilmiştir, Bölüm 2.2’de ayrık tam sayılı optimizasyon problemlerinin çözümü için literatürde yapılan çalışmalar verilmiştir ve Bölüm 2.3’te ise Jaya algoritması ile ilgili literatürde yapılan çalışmalar verilmiştir.

2.1. İkili Optimizasyon Problemlerinin Çözümü için Yapılan Çalışmalar

Farklı karakteristik yapıdaki birçok gerçek dünya problemi ikili optimizasyon olarak modellenmekte ve çözülebilmektedir. Bunlardan bazıları; Rüzgâr Türbini Yerleştirme Problemi (Wind Turbine Placement Problem) (Ituarte-Villarreal ve Espiritu, 2011; Gao ve ark., 2015; Rehman ve ark., 2016; Parada ve ark., 2017; Beşkirli ve ark., 2018), Fazör Ölçüm Birimlerinin Optimum Yerleşimi Problemi (Optimal Placement of Phasor Measurement Units (PMUs) - OPPP) (Prakash ve ark., 2017), 0-1 SÇP (Raidl, 1998; Hristakeva ve Shrestha, 2004; Feng ve ark., 2016), Su Pompası Anahtarlama Problemi (Water Pump Switching Problem) (Geem, 2005), Kapasitesiz Tesis Yerleştirme Problemi (Kashan ve ark., 2013; Kiran ve Gunduz, 2013; Korkmaz, 2019), Birim Taahhüt

Problemi (BTP) (Unit Commitment Problem) (Chen ve ark., 2015). İkili optimizasyon problemlerinin çözümü için literatürde yapılan bazı çalışmalar bu bölümde verilmiştir. Ayrıca, rüzgâr türbin probleminin çözümü ile ilgili literatürde yapılan bazı çalışmalar bir bütün olarak bu bölümün bir alt bölümü olarak verilmiştir.

İkili optimizasyon problemlerinin çözümü için literatürde bir çok farklı yöntem önerilmiştir. Bu yöntemler kesin (exact), tahmini (approximate), yerel arama algoritmaları ve metasezgisel yaklaşımlardır. Barcelo ve ark. (1990) yapmış oldukları çalışmada Tesis Yerleştirme Probleminin (TYP) (Facility Location Problem - FLP) çözümü için Lagrangian relaxation sezgisel algoritmasını önermişlerdir (Barcelo ve ark., 1990). Bir başka çalışma ise Holmberg (1999), KTYP'nin çözümü için kesin sonuçlar veren Dal ve Sınır (Branch and Bound) algoritmasını önermiştir (Holmberg, 1999). Bertsimas ve Demir (2002), yapmış oldukları çalışmada Çok Boyutlu SÇP'nin (ÇBSÇP) (Multidimensional Knapsack Problem - MKP) çözümü için Tahmini Dinamik Programlama (Approximate Dynamic Programming - ADP) yaklaşımına dayalı bir yöntem önermişlerdir (Bertsimas ve Demir, 2002).

Optimizasyon problemlerinin çözümü için kullanılan kesin yöntemler, problem boyutunun az olduğu durumlarda genellikle iyi çözümler bulmaktadırlar. Ancak problem boyutunun ve problem karmaşıklığının arttığı durumlarda hesaplama maliyeti de artmaktadır ve kesin yaklaşımlar, optimum çözümü bulmakta zorlanmaktadırlar. Sonuç olarak büyük boyutlu problemler için makul bir zamanda kabul edilebilir çözümler elde etmek amacıyla daha etkili yöntemlere ihtiyaç duyulmuştur. Bu kapsamda büyük boyutlu problemlerin çözümü için daha iyi çözümler elde etmek amacıyla literatürde yerel arama yöntemleri ve metasezgisel yöntemler önerilmiştir. Sultan ve Fawzan (1999), yapmış oldukları çalışmada KTYP'nin çözümü için Tabu Arama (TA) (Tabu Search - TS) algoritmasına dayalı bir yerel arama algoritması önermişlerdir. Sultan ve Fawzan, yapmış oldukları çalışmada ilk çözümü Net Benefit (NB) sezgiseli adında açgözlü bir algoritma kullanarak oluşturmuşlardır. Başlangıç çözümünü oluşturduktan sonra ise TA algoritmasını mevcut çözüm üzerine uygulayarak, KTYP için daha etkili çözümler bulduklarını söylemişlerdir (Al-Sultan ve Al-Fawzan, 1999).

Ghosh (2009), yapmış olduğu çalışmada TA algoritmasına dayalı komşuluk arama stratejileri kullanarak KTYP'nin çözümü için yeni bir metot önermiştir. Ghosh, çalışmasında mevcut bir çözüm için yeni bir konum oluştururken arama geçmişini dikkate alarak oluşturmaktadır. Böylece her yeni çözüm oluşturulduğunda muhtemel komşular arasında ki en optimum çözüm seçilmektedir. Ghosh, önerdiği yaklaşımın iyi sonuçlar

elde ettiğini belirtmiştir (Ghosh, 2003). Yapılan bir başka çalışma da ise Zhuang ve Galiana (1990), ikili optimizasyon problemi olan BTP'nin çözümü için Benzetimli Tavlama (BT) (Simulated Annealing - SA) algoritmasına dayalı bir yaklaşım geliştirmişlerdir. Önerdikleri algoritmada oluşturdukları başlangıç çözümü arama uzayında kabul edilebilir rastgele bir çözümdür. Daha sonra, önerdikleri BT algoritmasını büyük boyutlu termal BTP üzerine uygulamışlardır (Zhuang ve Galiana, 1990).

Aydin ve Fogarty (2004), yapmış oldukları çalışmada Evrimsel Benzetimli Tavlama (EBT) - (Evolutionary Simulated Annealing - ESA) ve EBT'nin ayrıklaştırılmış versiyonu olan DEBT algoritmalarını önermişlerdir. Önerdikleri algoritmaların başarı performansını ve etkinliğini göstermek için Aydin ve Fogarty, bu algoritmaları iki farklı ikili optimizasyon problemi olan Klasik İş Çizelgeleme Problemi (Classical Job Shop Scheduling Problem) ve KTYP üzerine uygulamışlardır (Aydin ve Fogarty, 2004). Falkenauer ve Bouffouix (1991), yapmış oldukları çalışmada İş Çizelgeleme (Job Shop Scheduling) probleminin çözümü için Genetik Algoritma (GA) tabanlı bir yaklaşım geliştirmişlerdir. Falkenauer ve Bouffouix, çalışmalarında zaman ve kaynak tahsisi gibi kısıtları da dikkate alarak etkili bir yöntem sunduklarını belirtmişlerdir ve önerdikleri çalışma sayesinde iş çizelgeleme probleminin zorluğunun üstesinden gelen bir yaklaşım sunduklarını belirtmişlerdir (Falkenauer ve Bouffouix, 1991).

Khuri ve ark. (1994), çalışmalarında GENESys olarak tanımladıkları, GA tabanlı yeni bir yaklaşım önermişlerdir. Khuri ve ark., önerdikleri GENESys algoritmasının, birçok genel amaçlı GA tabanlı algoritmadan farklı olarak, arama uzayında yeni çözümler oluşturduğunda, oluşturulan bu çözümler eğer kısıtları sağlamıyorsa uygun olmayan çözümleri cezalandıracak şekilde maliyeti hesabı yapan basit bir amaç fonksiyonu tanımlayarak algoritmanın başarısını arttırdıklarını belirtmişlerdir. Önerdikleri GENESys algoritmasını çok boyutlu 0-1 SÇP üzerine uygulamışlardır ve GENESys algoritmasının, çok boyutlu 0-1 SÇP için başarılı sonuçlar bulduğu belirtmişlerdir (Khuri ve ark., 1994).

Jaramillo ve ark. (2002), yapmış oldukları çalışma da tesis yerleştirme problemlerinin çözümü için optimum veya optimuma yakın çözümler elde etmek amacıyla, alternatif bir yöntem olarak GA'nın performansını iyileştirmek için İkili Turnuva Seçimi (Binary Tournament Selection) metodunu ve Kombinasyonlu Çaprazlama (Fusion Crossover) yöntemlerini kullanmışlardır. Jaramillo ve ark., GA üzerinde yaptıkları bu değişikliklerin GA'nın performansını arttırdığını belirtmişlerdir (Jaramillo ve ark., 2002). Bir başka çalışmada Topcuoglu ve ark. (2005), Kapasitesiz tekli gövde yerleştirme tahsisi (Uncapacitated Single Allocation Hub Location - USAHL)

probleminin çözümü için güçlü çözümlere ulaşan değiştirilmiş GA yöntemini önermişlerdir. Daha sonra, önerdikleri GA tabanlı yaklaşımlarını, Sivil Havacılık Kurulunda (Civil Aeronautics Board - CAB) ve Avustralya Post'tan (Australian Post - AP) alınan veri setinin üzerine uygulayarak, elde ettikleri sonuçları aynı veri seti üzerine uygulanmış olan literatürdeki diğer algoritmaların en iyi çözümleri ile karşılaştırarak algoritmanın etkinliğini göstermişlerdir (Topcuoglu ve ark., 2005).

Lim ve ark. (2016), yaptıkları çalışmada 0-1 klasik SÇP'nin çözümü için Tek Eşli GA (Monogamous Pairs Genetic Algorithm - MopGA) yöntemini önermişlerdir. MopGA, sosyal hayatta tek eşlilikteki iki önemli kavram olan: çiftler arasındaki bağıllık ve çiftlerin birbirini aldatma ihtimalindeki düşük olasılık tekniklerini GA'nın yapısında kullanmışlardır. Genellikle birçok GA'da ebeveynler (parents) iterasyonlar ilerledikçe değişirler. Ancak Lim ve ark., önerdikleri çalışmada monogamous (tek eşlilik) ebeveynler olarak adlandırdıkları bazı ebeveynleri, nesiller boyunca kullanmaya devam etmektedirler. Lim ve ark., klasik GA'dan farklı olarak kullandıkları bu yapının algoritmanın başarısını arttırdığını belirtmişlerdir (Lim ve ark., 2016).

Kennedy ve Eberhart (1997), klasik Parçacık Sürü Optimizasyonu (PSO) algoritmasından ilham alarak ikili optimizasyon problemlerinin çözümü için yeni bir yöntem önermişlerdir. Klasik PSO sürekli optimizasyon problemlerinin çözümü için geliştirilmiştir. Bu nedenle temel PSO, ikili optimizasyon problemlerinin üzerine doğrudan uygulanamamaktadır. Kennedy ve Eberhart, sürekli optimizasyon problemleri üzerinde başarı sonuçlar veren PSO'nun kendine has konum güncelleme denkleminin kullanılması durumunda aynı başarının ikili optimizasyon problemleri içinde gösterip göstermeyeceğini gözlemlemek için PSO üzerinde bazı değişiklikler yaparak ikili optimizasyon problemleri üzerine uygulamışlardır. PSO'da parçacıkların hız bilgisi önemli bir bileşendir. Çünkü parçacığın yeni konumu hız bilgisine göre belirlenmektedir. Bu nedenle, Kennedy ve Eberhart, ikili optimizasyonun kendine özgü yapısı nedeniyle parçacığın yeni konumunu belirlemek için sigmoid lojistik transfer fonksiyonunu kullanmışlardır. Ayırıklaştırma süreci şu şekilde gerçekleşmektedir; sigmoid fonksiyonu, parçacığın yeni hız bilgisini sigmoid fonksiyonuna göndermekte ve sigmoid fonksiyonu mevcut hız bilgisine göre '0' ile '1' arasında ondalıklı bir değer ($S(v_i)$) döndürmektedir. Daha sonra $[0,1]$ aralığında rastgele bir değer oluşturulmakta ve üretilen rasgele değer $S(v_i)$ değerinden daha küçük ise, parçacığın ilgili boyutunun yeni değeri '1', aksi durumda ise '0' olmaktadır. Böylece parçacığın yeni konumu ikili optimizasyonun doğasına uygun bir şekilde oluşturulmuş olmaktadır (Kennedy ve Eberhart, 1997).

Başka bir çalışmada, Sevkli ve Guner (2006) klasik PSO algoritmasını konum güncelleme denkleminde herhangi bir değişiklik yapmadan kullanmışlardır. Öncelikle konum güncelleme aşamasında parçacık için yeni konum oluşturulmaktadır. Daha sonra ise parçacığın yeni konumunu ayırklaştırmak için, parçacık *mod* alma fonksiyonuna gönderilmekte ve sürekli değerler ayırklaştırılmaktadır. Sevkli ve Guner, önerdikleri PSO'nun başarı performansını arttırmak için PSO'nun yapısına yerel arama algoritmasını eklemişlerdir. Sevkli ve Guner, önerdikleri algoritmanın başarısını test etmek için, önerdikleri algoritmayı OR-Library kütüphanesinde verilmiş olan KTYP veri seti üzerine uygulamışlardır. Önerdikleri algoritmanın etkinliğini göstermek için ise elde ettikleri sonuçları, aynı veri seti üzerine uygulanmış olan GA ve EBT algoritmalarının sonuçları ile kıyaslamışlardır. Sevkli ve Guner, yaptıkları deneysel çalışmalar dikkate alındığında önerdikleri sürekli PSO'nun ikili optimizasyon problemlerinin çözümü için GA ve EBT algoritmalarına kıyasla daha etkili bir yöntem olduğunu belirtmişlerdir (Sevkli ve Guner, 2006). Guner ve Sevkli (2008), literatüre kazandırmış oldukları başka bir çalışmada ise PSO'nun ayrık bir versiyonu olan Ayrık Parçacık Sürü Optimizasyon (DPSO) algoritmasını önermişlerdir. Önerdikleri DPSO algoritmasını KTYP üzerine uygulamışlardır ve çalışmalarının deneysel sonuçlarına göre önerdikleri algoritmanın başarılı sonuçlar bulduğunu belirtmişlerdir (Guner ve Sevkli, 2008).

Bansal ve Deep (2012), yapmış oldukları çalışmada özellikle 0–1 SÇP'ni ve çok boyutlu SÇP'ni çözmek için Değiştirilmiş İkili Parçacık Sürü Optimizasyon (Modified Binary Particle Swarm Optimization - MBPSO) algoritmasını önermişlerdir. Kennedy ve Eberhart (1997)'in önermiş oldukları algoritmada, parçacığın yeni konumu belirlenirken sigmoid transfer fonksiyonu kullanırken, MBPSO'da ise parçacığın konumunu güncellemek için Bansal ve Deep tarafından yeni bir lojistik transfer fonksiyonu tanımlanmıştır. Bansal ve Deep, önerdikleri transfer fonksiyonunun sigmoid fonksiyonuna kıyasla daha geniş bir arama uzayında arama yapma imkânı sağladığını ve böylece daha iyi çözümlerin elde edildiğini belirtmişlerdir (Bansal ve Deep, 2012).

Mirjalili ve Lewis (2013), transfer fonksiyonlarının performansını karşılaştırmak için altı farklı transfer fonksiyonunu ikili PSO algoritması ile kombine etmişlerdir. Önerdikleri transfer fonksiyonlarının bir kısmı sigmoid tabanlı, bir kısmı ise tanjant hiperbolik tabanlıdır. Sigmoid tabanlı transfer fonksiyonları S Tipi (S-Shaped), diğerleri ise V Tipi (V-Shaped) olarak sınıflandırılmıştır. Mirjalili ve Lewis, transfer fonksiyonları ile bütünleştirdikleri ikili PSO algoritmalarını karşılaştırmak için bu algoritmaları 25 farklı kıyas (benchmark) probleminden oluşan CEC 2005 nümerik veri seti üzerine

uygulamışlardır. Mirjalili ve Lewis'in önerdikleri çalışmanın deneysel sonuçları dikkate alındığında V Tipi transfer fonksiyonlarının ikili PSO'nun çözüm kalitesini önemli ölçüde arttırdığını belirtmişlerdir (Mirjalili ve Lewis, 2013).

ABC algoritması sürekli optimizasyon problemlerinin çözümü için önerilmiş olan Metasezgisel bir yaklaşımdır (Karaboga ve Basturk, 2007). Bu nedenle, temel ABC algoritması ikili optimizasyon problemlerinin üzerine doğrudan uygulanamamaktadır. Kashan ve ark. (2011), yapmış oldukları çalışmada temel ABC üzerinde bazı değişiklikler yaparak DisABC adı verilen ayrık bir ikili ABC optimizasyon algoritması geliştirmişlerdir. DisABC, temel ABC'nin konum güncelleme mekanizması yerine Jaccard benzerlik katsayısını kullanarak popülasyondaki ikili bireylerin birbirine olan benzerlik ve benzersizliklerine göre, güncellenecek bireyin yeni konumunu oluşturmaktadır. Kashan ve ark., önerdikleri DisABC algoritmasını KTYP üzerine uygulayarak, algoritmanın başarısını göstermişlerdir (Kashan ve ark., 2011). Başka bir çalışmada ise, Kiran ve Gündüz (2013), ikili optimizasyon problemlerinin çözümü için binABC olarak adlandırdıkları ABC tabanlı ikili bir optimizasyon algoritması geliştirmişlerdir. binABC algoritmasında, bir arının yeni konumu oluşturulurken ABC algoritmasının orijinal güncelleme denklemi yerine 'Özel VEYA' (XOR) lojik kapısını kullanılarak yeni bir yaklaşım önermişlerdir. Kiran ve Gündüz, önerdikleri binABC algoritmasını, KTY problemleri üzerine uygulamışlardır ve önerdikleri algoritmanın çözüm kalitesini, sağlamlığını ve basitliğini göstermek için binABC algoritmasını aynı şartlar altında BPSO, DisABC ve geliştirilmiş BPSO (IBPSO) ile birlikte KTYP üzerine uygulamışlardır. Yaptıkları çalışmanın deneysel sonuçları dikkate alındığında binABC'nin ikili optimizasyon problemlerinin çözümü için daha efektif bir algoritma olduğunu söylemişlerdir (Kiran ve Gündüz, 2013). Hancer ve ark. (2015), evrimsel temelli benzerlik algoritmasını mevcut bir ikili ABC algoritmasına entegre ederek, özellik seçimi (feature selection) problemleri için MDisABC adlı ikili bir ABC önermişlerdir (Hancer ve ark., 2015). Başka bir çalışmada ise Kiran (2015), ikili optimizasyon problemlerinin çözümü için ABCbin adlı bir yaklaşım önermişlerdir, ABCbin algoritmasında ajanlar algoritmanın arka planında sürekli bir arama uzayında çalışmaktadırlar. Yani konum güncelleme sürekli veriler üzerinde olmaktadır. Ancak aday çözümün uygunluk değerinin hesaplanması aşamasında, aday çözümün ayrıklaştırılması işlemi *mod alma* fonksiyonu ile yapılmaktadır (Kiran, 2015a).

Storn ve Price (1997) tarafından geliştirilen Farksal Gelişim (Differential Evolution - DE) algoritması, sürekli optimizasyon problemlerinin çözmek için önerilmiş

olan stokastik popülasyon tabanlı bir algoritmadır. Kashan ve ark. (2013), özellikle ikili optimizasyon problemlerinin çözümü için Jaccard katsayısını kullanarak benzerlik-benzersizlik yaklaşımını DE algoritması ile birlikte kullanarak DE algoritmasının ikili bir versiyonunu geliştirmişlerdir. Daha sonra Kashan ve ark., geliştirdikleri ikili DE algoritmasının etkinliğini göstermek için önerdikleri algoritmayı KTYP üzerine uygulamışlardır (Kashan ve ark., 2013). Bir başka çalışmada Chen ve ark. (2015), bir önceki iterasyonda elde edilen popülasyondan faydalanarak optimum çözümler elde eden bir ikili öğrenme farksal gelişim (Binary Learning Differential Evolution - BLDE) algoritması önermişlerdir ve önerdikleri BLDE'nin etkinliğini göstermek için BLDE algoritmasını güç sistemlerinde BTP üzerine uygulamışlardır. Banitalebi ve ark. (2016), ikili optimizasyon problemlerinin çözümü için benzersizlik (dissimilarity) metriklerini DE algoritması ile birlikte kullanarak kendinden adaptif ikili DE (Self-adaptive binary DE - SabDE) algoritmasını önermişlerdir. SabDE, başlangıç çözümlerini oluşturmak için adaptif bir mekanizma, parametre değerlerini ayarlamak için ise kaotik bir mekanizma kullanmaktadır. Banitalebi ve ark., önerdikleri algoritmanın başarısını göstermek için önerdikleri SabDE algoritmasını büyük ölçekli kıyas problemleri olan CEC 2015 problemleri üzerine uygulamışlardır (Banitalebi ve ark., 2016) ve SabDE algoritmasının sonuçlarını; Büyük ölçekli 0-1 SÇP'lerin çözümü için geliştirilen basitleştirilmiş ikili harmoni arama algoritması (Simplified Binary Harmony Search - SBHS) (Kong ve ark., 2015), BTP'nin çözümü için önerilen ikili farksal gelişim algoritması (BLDE) (Chen ve ark., 2015), doğrusal olmayan yüksek boyutlu kıyas (benchmarks) fonksiyonları ve 0-1 MKP problemlerinin çözümü için geliştirilen ikili hibrit topoloji PSO karesel enterpolasyon (Binary Hybrid Topology PSO Quadratic Interpolation - BHTPSO-QI) (Beheshti ve ark., 2015), ikili kuantumdan ilham alan GSA (Binary Quantum-Inspired GSA - BQIGSA) (Nezamabadi-pour, 2015), ve ikili optimizasyon problemlerini çözmek için GA'nın çaprazlama ve takas operatörlerine dayalı yeni ikili ABC algoritması (Novel Binary Artificial Bee Colony - GBABC) (Ozturk ve ark., 2015) ile karşılaştırmışlardır.

Zou ve ark. (2011), 0-1 KP problemlerini çözmek için Global Harmoni Arama (Global Harmony Search - GHS) algoritmasına dayalı (A Novel Global Harmony Search algorithm - NGHS) yeni bir yaklaşım önermişlerdir. NGHS algoritması yapısında, konum güncelleme ve düşük olasılıklı genetik mutasyon olarak adlandırılan iki önemli mekanizma kullanmaktadır. Zou ve ark., önerdikleri NGHS algoritmasının 0-1 KP problemlerinin çözümü için iyi sonuçlar bulunduğunu belirtmişlerdir (Zou ve ark., 2011). Bir başka çalışma da ise Xiang ve ark. (2014), 0-1 SÇP'ni çözümü için Ayrık Global

Harmoni Arama (Discrete Global Harmony Search - DGHS) algoritmasını önermişlerdir. Xiang ve ark., DGHS algoritmanın etkinliğini arttırmak için başlangıç popülasyonu açgözlü bir algoritma kullanarak oluşturmuşlardır. Bununla birlikte, ayırık genetik mutasyon operatörü ve iki fazlı onarım (two-phase repair) operatörü kullanmışlardır. Xiang ve ark., yaptıkları değişikliklerin temel GHS algoritmasının performansını fark edilir şekilde arttırdığını belirtmişlerdir (Xiang ve ark., 2014). Feng ve ark. (2016), temel Guguk Kuşu Arama (Cuckoo Search - CS) algoritmasının eksikliklerini gidermek ve GHS algoritmasının avantajları kullanmak için CS algoritması ile GHS algoritmasını hibrit bir şekilde kullanarak CSGHS olarak adlandırdıkları yeni bir algoritma önermişlerdir. Feng ve ark.'nın yapmış oldukları çalışmaya göre CSGHS'nin etkinliğinin GHS ve CS algoritmarının etkin bir şekilde birlikte kullanılmasından kaynaklandığını belirtmişlerdir; GHS algoritmasının çözüm uzayında keşif (exploration) yapmanın etkinliğini ve CS algoritmasının sömürü (exploitation) sürecinin etkinliğini arttırdığını belirtmişlerdir (Feng ve ark., 2016).

Azad ve ark. (2013), KTYP'nin çözümü için S-bAFSA olarak adlandırdıkları Yapay Balık Sürüsü Algoritması (Artificial Fish Swarm Algorithm - AFSA) tabanlı basitleştirilmiş ikili bir AFSA algoritmasını önermişlerdir. S-bAFSA'da aday çözümler çaprazlama ve mutasyon operatörleri tarafından oluşturulmaktadır ve daha iyi çözümler elde etmek için belirli bir iterasyondan sonra popülasyondaki bireyler yeniden oluşturulmaktadır. Bununla birlikte, Azad ve ark., S-bAFSA'da çözümlerin kalitesini arttırmak için yerel bir arama algoritması da kullanmışlardır (Azad ve ark., 2013).

Başka bir çalışmada Prakash ve ark. (2017), PMU yerleştirme probleminin çözümü için optimum çözümler elde etmek amacıyla Jaya algoritmasına dayalı BJA (Binary Jaya) olarak adlandırdıkları yeni bir ikili optimizasyon algoritması önermişlerdir (Prakash ve ark., 2017). Temel Jaya algoritması kısıtlı ve kısıtsız sürekli optimizasyon problemlerin çözümü için (Rao, 2016) tarafından literatüre kazandırılmıştır. Prakash ve ark., tanjant hiperbolik transfer fonksiyonunu kullanarak saf Jaya algoritmasını ayırtmışlardır ve önerdikleri BJA algoritmanın üstünlüğünü ve doğruluğunu göstermek için BJA algoritmasını farklı standart test sistemleri üzerine uygulamışlardır (Prakash ve ark., 2017).

Uymaz ve ark. (2015), sürekli optimizasyon problemlerinin çözümü için alglerin doğal hareketlerini modelleyerek AAA'yı önermişlerdir (Uymaz ve ark., 2015). AAA'nın ikili optimizasyon problemleri üzerine uygulanabilmesi için ayırtılması gerekmektedir. Zhang ve ark. (2016), yapmış oldukları çalışmada AAA'yı bazı transfer

fonksiyonları ile birlikte kullanarak ayırtlaştırmışlardır ve önerdikleri algoritmanın başarısını göstermek için önerdikleri BAAA'yı 0-1 MKP üzerine uygulamışlardır (Zhang ve ark., 2016). Bir başka çalışmada ise Korkmaz ve ark. (2018), ikili optimizasyon problemlerinin çözümü için binAAA olarak adlandırıldıkları AAA'nın yeni bir ikili versiyonunu önermişlerdir. Korkmaz ve ark., önerdikleri binAAA'da yeni bir güncelleme mekanizması kullanmışlardır ve önerdikleri algoritmanın etkinliğini ve başarısını göstermek için binAAA'yı ve literatürden bilinen bazı algoritmaları ile aynı şartlar altında KTYP üzerine uygulamışlardır. Önerdikleri binAAA'nın ikili optimizasyon problemlerinin çözümü için etkin bir yöntem olduğunu belirtmişlerdir (Korkmaz ve ark., 2018)

Ağaç Tohum Algoritması (ATA) (Tree Seed Algorithm - TSA) (Kiran, 2015b) sürekli optimizasyon problemlerinin çözümü için literatüre kazandırılmış olan metasezgisel bir algoritmadır. TSA'nın ikili optimizasyon problemleri üzerine uygulanabilmesi için üzerinde bazı değişiklikler yapılması gerekmektedir. Bu kapsamda Cinar ve Kiran (2018), ikili optimizasyon problemlerini çözmek için TSA algoritmasını lojik kapılar ile birlikte kullanarak LogicTSA, benzerlik ölçüm tekniklerini kullanarak SimTSA ve bu iki tekniğin hibrit kullanılması ile birlikte SimLogicTSA olarak adlandırdıkları çeşitli ikili TSA algoritmalarını önermişlerdir. LogicTSA sadece 'XOR' lojik fonksiyonunu kullanırken, SimTSA ise sadece benzerlik metriklerini kullanmaktadır, ikisinin hibrit versiyonu olan SimLogicTSA ise, [0,1] aralığında rastgele oluşturulan bir değere bağlı olarak ya 'XOR' lojik fonksiyonunu ya da benzerlik metriklerini kullanmaktadır. Cinar ve Kiran, önerdikleri algoritmaların etkinliğini göstermek için önerdikleri algoritmaları ve literatürde bilinen bazı iyi algoritmaları, aynı şartlar altında KTYP üzerine uygulamışlardır ve çalışmalarının deneysel sonuçlarına göre önerdikleri SimLogicTSA algoritmasının KTYP için etkili çözümlere ulaştığını belirtmişlerdir (Cinar ve Kiran, 2018).

Rizk-Allah ve Hassanien (2018), 0-1 SÇP'nin çözümü için Yarasa Algoritması (Bat Algorithm - BA) tabanlı İkili Yarasa Algoritmasını (NBBA) önermişlerdir. Önerdikleri algoritma, ikili yarasa algoritması (BBA) ile yerel arama algoritmasını (LSS) birlikte kullanmaktadır. Yarasa algoritması yarasaların keşif kabiliyetini geliştirirken, LSS ise sömürü yeteneğini geliştirmektedir. Önerdikleri algoritmayı farklı boyutlu 0-1 SÇP'leri üzerine uygulamışlardır. Deneysel sonuçlarına göre BBA-LSS algoritmasının büyük ölçekli SÇP'nin çözümü için iyi sonuçlar verdiğini belirtmişlerdir (Rizk-Allah ve Hassanien, 2018).

Haklı ve Ortaçay (2019), yapmış oldukları çalışmada KTYP'nin çözümü için Dağıtık Arama (Scatter Search -SS) algoritması tabanlı Geliştirilmiş Dağıtık Arama (Improved Scatter Search - ISS) adlı bir yaklaşım geliştirmişlerdir. SS algoritması, ikili arama uzayındaki problemlere doğrudan uygulanabilmektedir. SS'nin keşif ve sömürü yapısını geliştirmek amacıyla, GA'nın farklı çaprazlama ve mutasyon operatörlerini kullanmışlardır. Önerdikleri algoritmanın KTY problemlerinin çözümü için başarılı sonuçlar bulunduğunu belirtmişlerdir (Haklı ve Ortaçay, 2019).

Rizk-Allah ve ark. (2019), ikili optimizasyon problemlerinin çözümü için Salp Sürü Algoritmasına (Salp Swarm Algorithm - SSA) dayalı BSSA olarak adlandırılan ikili SSA algoritmasını önermişlerdir. Önerdikleri algorithmada sürekli değerleri ayırklaştırmak amacıyla değiştirilmiş bir *Arctan* transfer fonksiyonu kullanmışlardır ve BSSA algoritmasını global optimizasyon problemleri üzerine uygulamışlardır (Rizk-Allah ve ark., 2019).

Baş ve Ülker (2020), özellik seçimi probleminin çözümü için Sosyal Örümcek Algoritması (James ve Li, 2015) (Social Spider Algorithm -SSA) tabanlı BinSSA adını verdikleri ikili sosyal örümcek algoritması geliştirmişlerdir. Önerdikleri algorithmada SSA algoritmasının konum güncellenmesi aşamasında transfer fonksiyonlarını kullanmışlardır. Çalışmalarında S Tipi (S-Shaped) ve V Tipi (V-Shaped) olmak üzere iki ayrı tipte sekiz farklı transfer fonksiyonunu kullanmışlardır. Özellik seçim probleminde sınıflandırıcı olarak ise K-en yakın komşu (K-nearest neighbor - KNN) ve destek vektör makineleri (Support Vector Machine - SVM) kullanmışlardır. Önerdikleri BinSSA algoritmasının özellik seçiminde iyi sonuçlar verdiğini belirtmişlerdir (Emine ve Ülker, 2020).

2.1.1. Rüzgâr türbini yerleştirme probleminin çözümü için yapılan çalışmalar

Rüzgâr türbinlerinin yerleştirilmesi probleminde amaç en az maliyet ile rüzgâr çiftliğine yerleştirilen türbinlerden maksimum verim elde etmektir. Her bir rüzgâr türbininin kurulum, bakım ve onarım maliyeti vardır. Bu kurulum, bakım ve onarım maliyetlerinin en aza indirilebilmesi için birçok rüzgâr türbininin rüzgâr çiftliğine birlikte kurulması gerekmektedir (Crespo ve Herna, 1996; Huang, 2009; Mittal, 2010). Rüzgâr çiftliğindeki rüzgâr türbinlerinin sayısının artması ile birlikte, türbinlerin yerleştirildiği konumlara bağlı olarak bir uyandırma etkisi (wake effect) oluşmaktadır. Çünkü bir türbinden geçmiş olan rüzgârın hızı azalmaktadır. Eğer iki rüzgâr türbini birbirine çok

yakın yerleştirilirse uyandırma etkisinden dolayı ikinci türbinden istenilen verim elde edilemeyecektir. Bundan dolayı da bir rüzgâr çiftliğinde, yerleştirilen türbinlerin konumu oldukça önem kazanmaktadır (Crespo ve Herna, 1996; Huang, 2009). Bu kapsamda rüzgâr türbini yerleştirme problemi Mosetti ve ark. (1994), tarafından ikili optimizasyon problemi olarak modellenmiş ve GA tabanlı bir yaklaşımla çözülmüştür. Çalışmalarında Jensen uyandırma modelini kullanmışlardır. Yaptıkları çalışmaya göre Jensen uyandırma modeli ve GA sayesinde, uyandırma etkisini minimize ettiklerini ve türbinlerden elde ettikleri verimi maksimize ettiklerini belirtmişlerdir (Jensen, 1983; Mosetti ve ark., 1994).

Grady ve ark. (2005) yapmış oldukları çalışmada, toplam türbin sayısını ve her bir rüzgâr çiftliğinin toplam alanını göz önünde bulundurarak, rüzgâr türbinlerini en optimum şekilde yerleştirerek maksimum enerji elde etmek için GA tabanlı bir yaklaşım önermiştir (Grady ve ark., 2005). Gao ve ark. (2015), rüzgâr türbini yerleştirme probleminin çözümü için Çok Popülasyonlu GA (Multi-Population Genetic Algorithm - MPGA) yöntemini geliştirmişlerdir. MPGA'yı en az yatırım maliyetiyle maksimum enerjiyi elde etmek amacıyla Hong Kong'un açık denizine uygulamışlardır (Gao ve ark., 2015). Wan ve ark. (2009), yapmış oldukları çalışmada GA ve *Weibull* fonksiyonunu birlikte kullanarak rüzgâr türbini probleminin çözümü için yeni bir model önermişlerdir. *Weibull* Fonksiyonu olası hız dağılımını göstererek, bir türbinden elde edilecek enerjinin tahmin edilmesi sağlamaktadır. Böylece rüzgâr çiftliğinde türbinlerin daha optimum şekilde yerleştirildiğini belirtmişlerdir (Wan ve ark., 2009). Mittal (2010), yapmış olduğu çalışmada büyük rüzgâr çiftliklerindeki rüzgâr türbinlerinin kurulum maliyetini en aza indirmek ve rüzgâr çiftliğinde üretilen birim enerji başına düşen maliyeti azaltmak için GA tabanlı bir yaklaşım önermiştir (Mittal, 2010). Bir başka çalışmada Mittal ve ark. (2016), olasılıksal (probabilistic) GA ve deterministik gradyan tabanlı optimizasyon (deterministic gradient based optimization) yöntemini melez (hibrit) bir şekilde kullanarak, rüzgâr türbinlerinin optimum yerleştirilmesi amacıyla yeni bir yaklaşım önermişlerdir (Mittal ve ark., 2016). Emami ve Noghereh (2010), yapmış oldukları çalışmada rüzgâr türbini yerleştirme probleminin çözümü için GA tabanlı yeni bir kodlama yöntemi ve maliyet, enerji ve verimlilik üzerinde daha fazla kontrol sağlamak için ayarlanabilir katsayılar içeren yeni bir amaç fonksiyonu önermişlerdir (Emami ve Noghereh, 2010). Parada ve ark. (2017), yapmış oldukları çalışmada rüzgâr türbini yerleştirme problemi için en uygun yerleşimi elde etmek amacıyla Gauss uyandırma modelini (Gaussian wake model) kullanarak yeni bir yaklaşım önermişlerdir. Gauss uyandırma modeli, uyandırma etkisi için tek tip bir hız profili düşünen Jensen uyandırma

etkisi modeli yerine hız eksikliğinin üstesinden gelmek için üstel bir fonksiyon kullanmaktadır (Parada ve ark., 2017).

Başka bir çalışmada, Ituarte-Villarreal ve Espiritu (2011), sabit rüzgâr hızı ve tek yönlü rüzgârı dikkate alarak rüzgâr türbinlerinin optimum yerleşimini bulmak için yeni bir viral tabanlı optimizasyon algoritması geliştirmişlerdir (Ituarte-Villarreal ve Espiritu, 2011). Pookpant ve Ongsakul (2013), bir rüzgâr çiftliğindeki türbinlerin yerleşimi problemini çözmek için zamanla değişen hızlanma katsayıları (Time-Varying Acceleration Coefficients - TVAC) ile ikili parçacık sürüsü optimizasyonu (BPSO) algoritmasını kullanarak verimli bir yaklaşım önermişlerdir. BPSO-TVAC'nin etkinliğini göstermek için test alanını 100 eşit kareye bölmüşlerdir ve BPSO-TVAC algoritmasını test alanı üzerine uygulamışlardır (Pookpant ve Ongsakul, 2013). Başka bir çalışmada Moorthy ve Deshmukh (2015), rüzgâr türbini yerleştirme problemini çözmek için PSO algoritmasına dayalı yeni bir yaklaşım geliştirmişlerdir (Moorthy ve Deshmukh, 2015).

Gonzalez ve ark. (2010), rüzgâr türbini problemini çözmek için, uyandırma etkisi eksikliğinden kaynaklanan yıllık enerji üretimindeki düşüş gibi kısıtlamaları göz önünde bulundurarak, evrimsel algoritma (Evolutive Algorithm - EA) tabanlı bir yaklaşım önermişlerdir. Changshui ve ark. (2011), rüzgâr türbini yerleştirme problemini çözmek için submodüler özellikli Jensen uyandırma modelini tembel ağgözlü optimizasyon (Lazy Greedy Optimization) algoritması ile birlikte kullanarak hızlı bir algoritma geliştirmişlerdir (Changshui ve ark., 2011). Rehman ve ark. (2016), rüzgâr türbini yerleştirme probleminin çözümü için sezgisel tabanlı tohum çözümü (Heuristic-based Seed Solution) ile guguk kuşu arama (GS) (Cuckoo Search - CS) algoritmasına dayalı yeni bir yöntem önermişlerdir (Rehman ve ark., 2016). Patel ve ark. (2017), rüzgâr türbini yerleştirilme probleminin çözümü için etkili bir çözüm önermek amacıyla Öğretme-Öğrenmeye Dayalı Optimizasyon (Teaching-Learning Based Optimization - TLBO) algoritmasını önermişlerdir. Çalışmalarına göre, temel TLBO'nun keşif ve sömürü özelliklerine, etkili arama yöntemlerini dâhil etmişlerdir. (Patel ve ark., 2017). Bunlara ek olarak, son yıllarda rüzgâr türbini yerleştirme optimizasyon problemini çözmek için farklı optimizasyon algoritmaları geliştirilmiştir. Bu algoritmalarından bazıları şunlardır; on farklı transfer fonksiyonuna dayalı İkili AAA (BinAAA) (Beşkirli ve ark., 2018), Biyocoğrafik Tabanlı Optimizasyon (Biogeographical-based Optimization) algoritması (Pouralafi-Kheljan ve ark., 2018) ve İkili Yabani Ot Optimizasyonu (Binary Invasive Weed Optimization - BIWO) algoritması (Beşkirli ve ark., 2019).

2.2. Ayırık Tam Sayı Optimizasyon Problemlerinin Çözümü için Yapılan Çalışmalar

Görünüşte çözümü basit gibi görünen pek çok optimizasyon problemini makul bir sürede çözmek zordur. Çünkü problemin boyutsallığının artmasına bağlı olarak, potansiyel çözümlerin sayısı katlanarak artmakta ve her olası kombinasyonu deneyerek sonuca ulaşmak çok karmaşık hale gelmektedir (Ouaarab ve ark., 2014). Gezgin Satıcı Problemi (Shi ve ark., 2007; Gündüz ve ark., 2015; Mahi ve ark., 2015; Zhou ve ark., 2016), Üretim Hücresi Oluşumu Problemi (Manufacturing Cell Formation Problem) (Sayadi ve ark., 2013), Graf Boyama Problemi (Graph Coloring Problem) (Hertz ve de Werra, 1987; Aslan ve Baykan, 2016) gibi birçok gerçek dünya problemi ayırık tamsayı optimizasyon problemi olarak çözülmektedir. GSP, kombinatorial ayırık optimizasyondaki ünlü problemlerden biridir ve NP-Zor tipi optimizasyon problemi kategorisine dahil edilmektedir (Arora, 1998; Ouaarab ve ark., 2014). Çünkü muhtemel çözümlerin sayısı, karar değişkenlerine bağlı olarak katlanarak artmakta ve böyle bir problemi polinomsal zamanda çözecek bir algoritma henüz mevcut değildir. GSP alanında tanımlanan bazı problemler şunlardır; Bilgisayar Kablolaması (Computer Wiring) problemi, Araç Rotalama (Vehicle Routing) problemi, bir Veri Dizisini Kümeleme problemi (Clustering a Data Array), İş Yeri Çizelgeleme (Job-Shop Scheduling) problemi (Lenstra ve Kan, 1975), Baskı Devre Kartlarının Delinmesi problemi (Drilling of Printed Circuit Boards), X-ışını Kristalografisi (X-ray Crystallography) (Ravikumar, 1992; Reinelt, 1994).

GSP problemi: Minimum toplam maliyetle n adet şehri ziyaret etmek isteyen bir satış görevlisinin her şehri sadece bir kez ziyaret edebileceği ve en sonunda başlangıç noktasına dönebileceği en kısa Hamilton döngüsünün bulması olarak tanımlanabilir (Zhong ve ark., 2007). Hamilton döngüsünde; başlangıç düğümünden (şehir) başlayarak her düğüme sadece bir kez uğramak şartı ile başlangıç düğümüne geri dönmeli ve bunu yaparken de geçilen yoldan (kavis) bir daha geçilmemelidir.

GSP'yi çözmek için kesin, yaklaşık, yerel arama ve metasezgisel tabanlı birçok yöntem literatürde önerilmiştir. Fleischmann (1985) yapmış olduğu çalışmada, yol ağlarında GSP probleminin çözümü için Kesme Düzlemi (Cutting Plane) adında bir metot önermiştir (Fleischmann, 1985). Radharamanan ve Choi (1986), GSP ve ulaşım rotalama problemlerini çözmek için dal ve sınır yöntemini ceza tur fonksiyonu ile birlikte kullanarak yeni bir metot geliştirilmişlerdir (Radharamanan ve Choi, 1986). Padberg ve

Rinaldi (1987), 532 şehirden oluşan simetrik GSP'nin çözümü için Dal ve Kesme (Branch-and-Cut) algoritmasına dayalı bir yöntem önermişlerdir (Padberg ve Rinaldi, 1987). Başka bir çalışmada ise, Chentsov ve Korotayeva (1997), ayrık-sürekli rotalama probleminin çözümü için Dinamik Programlama (Dynamic Programming - DP) tabanlı bir yaklaşım geliştirmişlerdir (Chentsov ve Korotayeva, 1997). Barnhart ve ark. (1998), büyük tamsayı programları çözmek için sütun üretimi problemimin çözümü için Dal ve Fiyat (Branch and Price) adlı metodunu önermişlerdir (Barnhart ve ark., 1998). Başka bir çalışmada, Hernández-Pérez ve Salazar-González (2004), hammadde toplama ve teslimat adı verilen bir rotalama probleminin çözümü için Dal ve Kesme algoritması tabanlı bir yaklaşım önermişlerdir (Hernández-Pérez ve Salazar-González, 2004). Ergun ve Orlin (2006), GSP'yi çözmek için çok geniş çaplı komşuluk araştırmasında dinamik programlama metodolojisini önermişlerdir (Ergun ve Orlin, 2006).

Knox (1994), simetrik GSP'yi çözmek için TA algoritmasına dayalı bir yaklaşım sunmuş ve TA'yı, K-opt algoritmasının sonuçları ile karşılaştırmıştır (Knox, 1994). Irnich ve ark. (2006), kapasiteli araç yönlendirme probleminin (Capacitated Vehicle-Routing Problem - CVRP) çözümü için Or-opt, 2-opt ve 3-opt gibi yerel arama algoritmalarını sıralı arama algoritması (Sequential Search Algorithm) ile birlikte kullanmışlardır (Irnich ve ark., 2006). Geng ve ark. (2011), GSP'yi çözmek için SA algoritmasına ve açgözlü arama tekniklerine dayalı etkili bir yerel arama algoritması önermişlerdir ve daha etkili çözümlere ulaşmak için üç farklı mutasyon ve açgözlü arama yönteminin kombinasyonunu ve SA algoritmasını birlikte kullanılmışlardır (Geng ve ark., 2011).

GA doğası gereği sürekli ve ayrık optimizasyon problemlerinin çözümü için kolay uyarlanabilir bir algoritmadır. Bu nedenle GA'nın çaprazlama (crossover) ve mutasyon (mutation) operatörleri birçok sezgisel algoritmanın güncelleme mekanizmasında kullanılmaktadır (Goldberg ve Holland, 1988). Grefenstette ve ark. (1985), GSP'yi çözmek için GA'ya dayalı bir yöntem önermişlerdir. Bir başka çalışmada Ulder ve ark. (1990) GSP'nin çözümü için GA tabanlı bir yerel arama algoritmaları önermişlerdir. Önerdikleri bu algoritmalarda 2-opt ve Kernighan komşuluk operatörlerini kullanmışlardır (Lin ve Kernighan, 1973; Ulder ve ark., 1990). Başka bir çalışmada Potvin (1996), farklı çaprazlama ve mutasyon tekniklerinin kullanarak GA tabanlı farklı yaklaşımlar üzerine bir araştırma sunmuştur (Potvin, 1996). Üçoluk (2002), GSP için özel bir çaprazlama ve mutasyon operatörüne dayanan GA tabanlı bir yaklaşım önermiştir (Üçoluk, 2002). Ahmed (2010), GSP'yi çözmek için Sıralı Yapıcı Çaprazlama (Sequential Constructive Crossover - SCX) adlı çaprazlama operatörüne dayanan GA

tabanlı bir yaklaşım önermiştir. SCX tabanlı yaklaşım, Kenar Rekombinasyon Çaprazlama (Edge Recombination Crossover - ERX) ve genelleştirilmiş N-nokta Çaprazlama (N-Point Crossover - GNX) gibi bazı mevcut çaprazlama operatörlerine dayanan yaklaşımların elde ettikleri sonuçlar ile karşılaştırılmış ve SCX tabanlı yaklaşımın daha iyi sonuçlar bulduğunu belirtmiştir (Ahmed, 2010). Başka bir çalışmada, Hussain ve ark. (2017), GSP'yi çözmek için Değiştirilmiş Çevrim Çaprazlama (Modified Cycle Crossover - CX2) adlı çaprazlama operatörünü GA ile birlikte kullanmışlardır (Hussain ve ark., 2017).

Wang ve ark. (2003), GSP'nin çözümü için PSO tabanlı yeni bir yaklaşım sunmuşlardır. Yaptıkları çalışmada takas (swap) operatörü ve takas dizisini kullanan bazı özel yöntemler geliştirmişlerdir ve bununla birlikte bazı operatörleri yeniden tanımlamışlardır (Wang ve ark., 2003). Pang ve ark. (2004), yapmış oldukları çalışmada GSP'yi çözmek için parçacıkların konumunu ve hızını temsil etmek için entegre bulanık matrislere (integrated fuzzy matrices) sahip modifiye edilmiş ayırık PSO algoritmasını önermişlerdir (Pang ve ark., 2004). Başka bir çalışmada Shi ve ark. (2007), GSP'yi çözmek için PSO tabanlı yeni bir yaklaşım önermişlerdir. Yaptıkları çalışmada, yakınsama hızını arttırmak için belirsiz arama stratejisini (certain searching strategy) ve çaprazlamayı eleme tekniğini (crossover eliminated technique) kullanmışlardır (Shi ve ark., 2007). Chen ve Chien (2011), GSP'yi çözmek için PSO algoritmasını GA, SA ve Karınca Kolonisi Optimizasyon (Ant Colony Optimizayion - ACO) algoritması ile birlikte kullanarak karma bir yaklaşım geliştirmişlerdir. Çalışmalarını TSPLIB'den elde edilen 25 örnek üzerinde test etmişlerdir ve elde ettikleri deneysel sonuçları bazı evrimsel optimizasyon algoritmalarıyla karşılaştırmışlardır (Chen ve Chien, 2011). Başka bir çalışmada Mahi ve ark. (2015), GSP'yi çözmek için PSO, ACO ve bir yerel arama algoritması olan 3-opt algoritmasına dayanan yeni bir hibrit yöntem önermişlerdir. Çalışmalarında, ACO algoritmasında şehir seçim işlemlerinde kullanılan ve şehirler arası feromon ve mesafelerin önemini belirleyen α ve β parametrelerinin optimum değerlerini tespit etmek için PSO'yu kullanmışlardır. Sonlandırma koşulu karşılandıktan sonra ise tur uzunluğunun azaltılması için 3-opt (Lin, 1965) yerel arama algoritmasını uygulamışlardır (Mahi ve ark., 2015).

Dorigo ve Gambardella (1997), simetrik ve asimetrik GSP problemlerinin çözümü için Karınca Kolonisi Sistemini (Ant Colony System - ACS) ve ACS-3-opt yöntemini kullanmışlardır (Dorigo ve Gambardella, 1997). Gülcü ve ark. (2018), GSP'yi çözmek için ACO ve 3-opt algoritmalarını dahil ettikleri PACO-3opt adından paralel işbirlikçi bir

hibrit algoritma önermişlerdir. İlk olarak PACO algoritmasını GSP üzerine uygulamışlardır ve belirli bir iterasyondan sonra mevcut çözümleri iyileştirmek için her koloniye 3-opt algoritmasını uygulayıp, en iyi turu diğer kolonilerle paylaşmışlardır. Çalışmalarının deneysel sonuçlarına göre, önerdikleri PACO-3opt algoritmasının daha verimli ve güvenilir bir performans gösterdiğini belirtmişlerdir (Gülcü ve ark., 2018). Başka bir çalışmada Gündüz ve ark. (2015), GSP'yi çözmek için ACO ve ABC'ye dayalı Hiyerarşik Yaklaşım (Hierarchic Approach - HA) adlı bir yöntem önermişlerdir. Çalışmalarında, yol yapımına dayalı (path construction-based) yöntem olan ACO algoritmasını, yol iyileştirmeye dayalı (path improvement-based) olan ABC algoritmasının başlangıç çözümlerinin daha iyi bir şekilde oluşturulması aşamasında kullanmışlardır (Gündüz ve ark., 2015). Li ve ark. (2011), yapmış oldukları çalışmada açgözlü seçimle daha iyi bir tur elde etmek için takas operatörünü ABC algoritması ile birlikte kullanarak yeni bir ayrık ABC algoritması önermişlerdir (Li ve ark., 2011). Karaboga ve Görkemli (2011), GSP'yi çözmek için Kombinatorial ABC (Combinatorial ABC - CABC) adı verilen yeni bir ABC algoritması önermişlerdir (Karaboga ve Gorkemli, 2011). Bir başka çalışmada ise Görkemli ve Karaboga (2013), CABC'nin hızlı ABC (quick ABC - qABC) adı verilen daha da geliştirilmiş bir versiyonunu önermişlerdir ve yaptıkları iyileştirme sayesinde izci arıların davranışlarının daha ayrıntılı bir şekilde modellendiğini belirtmişlerdir (Görkemli ve Karaboga, 2013). Kıran ve ark. (2013) yapmış oldukları çalışmada, GSP problemini çözmek için komşuluk operatörlerine (neighborhood operators) dayalı ayrık ABC algoritmasını önermişlerdir. Yaptıkları çalışmada, Rasgele Takas (Random Swap - RS), Rasgele Ekleme (Random Insertion - RI), Alt Dizilerin Rastgele Takası (Random Swap of Subsequences- RSS), Alt Dizinin Rastgele Eklenmesi (Random Insertion of Subsequence - RIS), Alt Dizinin Rastgele Tersine Çevrilmesi (Random Reversing of Subsequence - RRS), Alt Dizilerin Rastgele Ters Çevrilmesi ve Takas edilmesi (Random Reversing Swap of Subsequences - RRSS) ve Alt Dizinin Rastgele Tersine Çevirmesi ve Eklenmesi (Random Reversing Insertion of Subsequence - RRIS) komşuluk operatörlerini kullanmışlardır. Ayrıca önerdikleri ayrık ABC algoritmasının etkinliğini artırmak için 2-opt ve 3-opt yerel arama yaklaşımlarını ayrık ABC algoritmasına entegre etmişlerdir. (Kıran ve ark., 2013).

Chunhua ve ark. (2012), GSP problemini çözmek için Durum Geçiş Algoritması (State Transition Algorithm - STA) tabanlı bir yöntem geliştirmişlerdir (Chunhua ve ark., 2012). Başka bir çalışmada ise Zhou ve ark. (2016), kısıtsız tamsayı optimizasyon problemlerini çözmek için Ayrık Durum Geçiş Algoritmasını (Discrete State Transition

Algorithm - DTSA) önermişlerdir ve önerdikleri algorithmda aday çözümleri oluşturmak için takas, öteleme ve simetri (swap, shift and symmetry) adı verilen komşuluk operatörlerini kullanılmışlardır (Zhou ve ark., 2016). Ouaarab ve ark. (2014), GSP'yi çözmek için Guguk Kuşu Arama (CS) algoritmasının ayrı bir versiyonunu önermişlerdir ve geliştirdikleri algorithmda CS algoritmasının zayıf yönlerinin üstesinden gelmek için algoritmanın yerel arama mekanizmasını iyileştirmişlerdir (Ouaarab ve ark., 2014). Başka bir çalışmada ise Hatamlou (2018), GSP'yi çözmek için Kara Delik (Black Hole - BH) algoritmasına dayalı yeni bir yaklaşım sunmuştur. Önerdiği BH algoritmasının başarı performansını göstermek amacıyla, BH algoritmasını farklı GSP problemleri üzerine uygulamış ve elde ettiği deneysel sonuçları literatürdeki benzer algoritmalar ile kıyaslamıştır (Hatamlou, 2018). Bir başka çalışmada ise Cinar ve ark. (2019) permütasyon tabanlı optimizasyon problemlerini çözmek için takas, öteleme ve simetri komşuluk operatörlerini temel TSA'nın konum güncelleme mekanizmasına ekleyerek Ayrık TSA (DTSA) algoritmasını önermişlerdir. Önerdikleri DTSA'da başlangıç popülasyonunu oluştururken, popülasyondaki ilk bireyi en yakın komşu turu (nearest neighbor tour) sezgisel algoritması ile oluşturmuşlardır. Popülasyondaki diğer bireyleri ise rastgele bir permütasyon ile oluşturmuşlardır. Durdurma kriteri sağlandığında ise 2-opt yerel arama algoritması çalıştırılmakta ve bulunan tur uzunluğundan daha kısa bir yol olup olmadığı araştırılmaktadır. Böylece DTSA'nın etkinliğinin arttığını belirtmişlerdir (Cinar ve ark., 2019).

2.3. Jaya Algoritması ile İlgili Yapılan Çalışmalar

Jaya algoritması, kısıtlı ve kısıtsız sürekli optimizasyon problemlerinin çözümü için Rao (2016) tarafından literatüre kazandırılan popülasyon tabanlı bir algoritmadır. Jaya algoritmasının en önemli özelliklerinden biri parametre bağımsız olmasıdır. Bu özelliği sayesinde optimizasyon problemlerine kolay bir şekilde uyarlanabilmektedir. Jaya, algoritmasının bir diğer önemli özelliği ise, çalışma yapısı belirli bir problemin en iyi çözüme doğru hareket etme eğilimini arttırırken, aynı zamanda en kötü çözümden kaçarak, yerel en kötü ya da yerel en iyi çözüme takılmayı önlemek üzerinedir. Bu nedenle hem popülasyondaki en iyi çözümü hem de en kötü çözümü konum güncelleme denkleminde kullanmaktadır. Bu da arama uzayında daha geniş bir araştırma yapmayı sağlamaktadır. Rao (2016), tarafından literatüre kazandırılan Jaya algoritması birçok araştırmacıya ilham kaynağı olmuş ve birçok farklı problemlerin çözümü için

uyarlanmıştır. Hem tek amaçlı ve hem de çok amaçlı gerçek dünya optimizasyon problemlerini çözmek için literatürde birçok farklı Jaya tabanlı yaklaşım önerilmiştir (Rao, 2019). Bu kısımda Jaya algoritması ile yapılan çalışmalar detaylandırılmıştır.

Prakash ve ark. (2017) yapmış oldukları çalışmada fazör ölçüm birimlerinin optimum yerleşimi probleminin çözümü için Jaya tabanlı İkili Jaya (Binary Jaya - BJA) algoritmasını önermişlerdir. Yapmış oldukları çalışmada sürekli değerleri ayrıklaştırmak için tanjant hiperbolik lojistik transfer fonksiyonunu kullanmışlardır. Önerdikleri BJA algoritmasının üstünlüğünü ve doğruluğunu göstermek için, BJA'yı farklı test sistemleri üzerine uygulamışlardır (Prakash ve ark., 2017). Başka bir çalışmada Rao ve ark. (2016) yüzey taşlama işlemi optimizasyonu (surface grinding process optimization) probleminin çözümü için Jaya tabanlı bir yaklaşım kullanmışlardır (Rao ve ark., 2016b). Benzer şekilde, Rao ve Waghmare (2017) yapmış oldukları çalışmada, Jaya algoritmasını karmaşık kısıtlı tasarım optimizasyon problemlerine (complex constrained design optimization problems) uygulamışlardır (Rao ve Waghmare, 2017). Wang ve Huang (2018), fotovoltaik hücre modellerinin parametre tahmini için Elit Muhalefet Tabanlı bir Jaya algoritması (Elite Opposition-based Jaya - EO-Jaya) önermişlerdir ve yaptıkları deneysel çalışmaların sonucuna göre, EO-Jaya'nın fotovoltaik hücre modelleri için iyi sonuçlar elde ettiğini belirtmişlerdir (Wang ve Huang, 2018). Rao ve ark. (2016a), yapmış oldukları çalışmada plazma ark işleme (plasma arc machining - PAM), elektro-deşarj işleme (electro-discharge machining - EDM) ve mikro elektro-deşarj işleme (micro electro-discharge machining - μ -EDM) süreçlerinin çok amaçlı optimizasyon yönlerini çözmek için Çok Amaçlı Jaya (multi-Objective - MO-Jaya) algoritmasını önermişlerdir (Rao ve ark., 2016a). Bir başka çalışmada ise Rao ve ark. (2017), modern işleme süreçleri optimizasyonunu (modern machining processes optimization) probleminin çözümü için çok amaçlı bir Jaya algoritması önermişlerdir (Rao ve ark., 2017).

Rao ve More (2017), termal cihazların optimum tasarımı için kendinden adaptif Jaya (self-adaptive Jaya) algoritması geliştirmişlerdir ve önerdikleri algoritmanın, çalışmalarında karşılaştırdıkları diğer optimizasyon tekniklerinden daha iyi sonuçlar verdiğini belirtmişlerdir (Rao ve More, 2017). Rao ve Saroj (2017), kabuk ve borulu ısı değiştirici (shell-and-tube heat exchanger - STH) tasarım problemlerinin çözümü için Jaya tabanlı Elitist Jaya algoritması önermişlerdir (Rao ve Saroj, 2017). Huang ve ark. (2018), kısmi gölgeleme koşulları altında PV sistemlerinin maksimum güç noktası izleme (maximum power point tracking - MPPT) problemini verimli bir şekilde çözmek için Jaya algoritmasına dayalı bir yöntem önermişlerdir (Huang ve ark., 2018).

Wang ve ark. (2018a), toprak su tutma eğrisi modelinin (soil water retention curve model) parametre tahmini için Jaya algoritmasını kullanmışlardır (Wang ve ark., 2018a). Başka bir çalışmada ise Wang ve ark. (2018b) Li-ion pil modeli parametrelerini verimli bir şekilde tahmin etmek için GPU Hızlandırmalı Paralel Jaya algoritmasını (GPU-Accelerated Parallel Jaya - GPU-Jaya) önermişlerdir. GPU-Jaya algoritmasının, pil modeli parametrelerini doğru bir şekilde tahmin etmesinin yanında, aynı zamanda hem giriş seviyesi hem de profesyonel seviyede GPU kullanımı sırasında çalışma süresini büyük ölçüde azalttığını belirtmişlerdir (Wang ve ark., 2018b). Uzlu (2019), yapmış olduğu çalışmada Türkiye'nin gelecekteki enerji talebini tahmin etmek için Yapay Sinir Ağları (YSA) (Artificial Neural Network - ANN) ve Jaya algoritmasına dayalı yeni bir yaklaşım önermiştir (Uzlu, 2019). Rao ve Saroj (2019), farklı nümerik ve mühendislik optimizasyon problemlerini çözmek için popülasyonu alt popülasyonlara ayıran Çoklu Popülasyonlu Elitist Tabanlı Kendinden Adaptif Jaya (Elitist-based Self-Adaptive Multi-Population - SAMPE) algoritması önermişlerdir. Önerdikleri algoritmanın etkinliğini göstermek için SAMPE-Jaya algoritmasını CEC 2015 problemleri üzerinde test etmişlerdir (Rao ve Saroj, 2019).

Singh ve Chaudhary (2020), tarımsal harman çarkı (flywheel of the agricultural thresher) optimizasyon probleminin çözümü için NSJaya adı verilen, bastırılmamış sıralama (non-dominated sorting) ve Jaya algoritmasına dayalı bir yaklaşım önermişlerdir. NSJaya algoritması, optimum Pareto yüzeyini elde etmek için kalabalıklaşma mesafesini (crowding distance) ve bastırılmamış sıralama yaklaşımını kullanmaktadır (Singh ve Chaudhary, 2020). Kumawat ve ark. (2020), yapmış oldukları çalışmada dağıtılmış enerji kaynaklarının optimum tahsisi (optimal allocation of distributed energy resources) probleminin çözümü için Jaya tabanlı bir yaklaşım kullanmışlardır (Kumawat ve ark., 2020). Başka bir çalışmada ise Wu ve He (2020), Set-Union SÇP'nin (Set-Union Knapsack Problem - SUKP) çözümü için Çift Kodlamalı Hibrit Jaya (Hybrid Jaya Algorithm with Double Coding - DHJaya) algoritmasını önermişlerdir (Wu ve He, 2020). Warid (2020), optimum güç akışı probleminin (optimal power flow problem) çözümü için AMTPG-Jaya algoritmasını önermiştir. AMTPG-Jaya algoritmasının etkinliğini göstermek için AMTPG-Jaya algoritmasını iyi bilinen iki standart güç sistemine uygulamıştır (Warid, 2020). More ve Rao (2020), plaka kanatlı ısı değiştiricinin (plate-fin heat exchanger) tasarım optimizasyonu probleminin çözümü için değiştirilmiş bir Jaya algoritması önermişlerdir (More ve Rao, 2020). Başka bir çalışmada ise Dede ve ark. (2020), çaprazlı kubbe yapılarının (braced dome structures)

optimizasyonu probleminin çözümü için Jaya algoritması tabanlı bir yaklaşım önermişlerdir (Dede ve ark., 2020).

Ingle ve Jatoth (2020), doğrusal olmayan kablosuz iletişim kanalları probleminin (non-linear channel equalization problem) çözümü için Jaya algoritması ve Lévy uçuşu (Lévy Fligh - LF) algoritmalarına dayalı JAYALF olarak adlandırılan bir yaklaşım sunmuşlardır. Global arama için Jaya algoritmasının arama gücünü, yerel arama için ise LF algoritmasının arama gücünü kullanmışlardır. Ingle ve Jatoth, önerdikleri JAYALF algoritmasının etkinliğini doğrulamak için, simülasyon amacıyla JAYALF algoritmasını iki farklı doğrusal olmayan kablosuz iletişim kanalları üzerine uygulamışlardır (Ingle ve Jatoth, 2020).



3. MATERYAL VE YÖNTEM

Tez çalışmasının bu bölümünde temel Jaya algoritması, BJA algoritması, ikili optimizasyon problemlerinin çözümü için önerilen JayaX ve JayaX-LSM algoritmaları, ayrık tamsayı optimizasyon problemlerinin çözümü için önerilen DJaya algoritması detaylandırılmıştır. Bununla birlikte deneylerde kullanılan problemler ve veri setleri de çalışmanın bu aşamasında verilmiştir. Deneylerde kullanılan problemler; KTYP, CEC 2015 nümerik fonksiyonları, rüzgâr türbini yerleştirme problemi ve GSP'dir.

3.1. Optimizasyon Algoritmaları

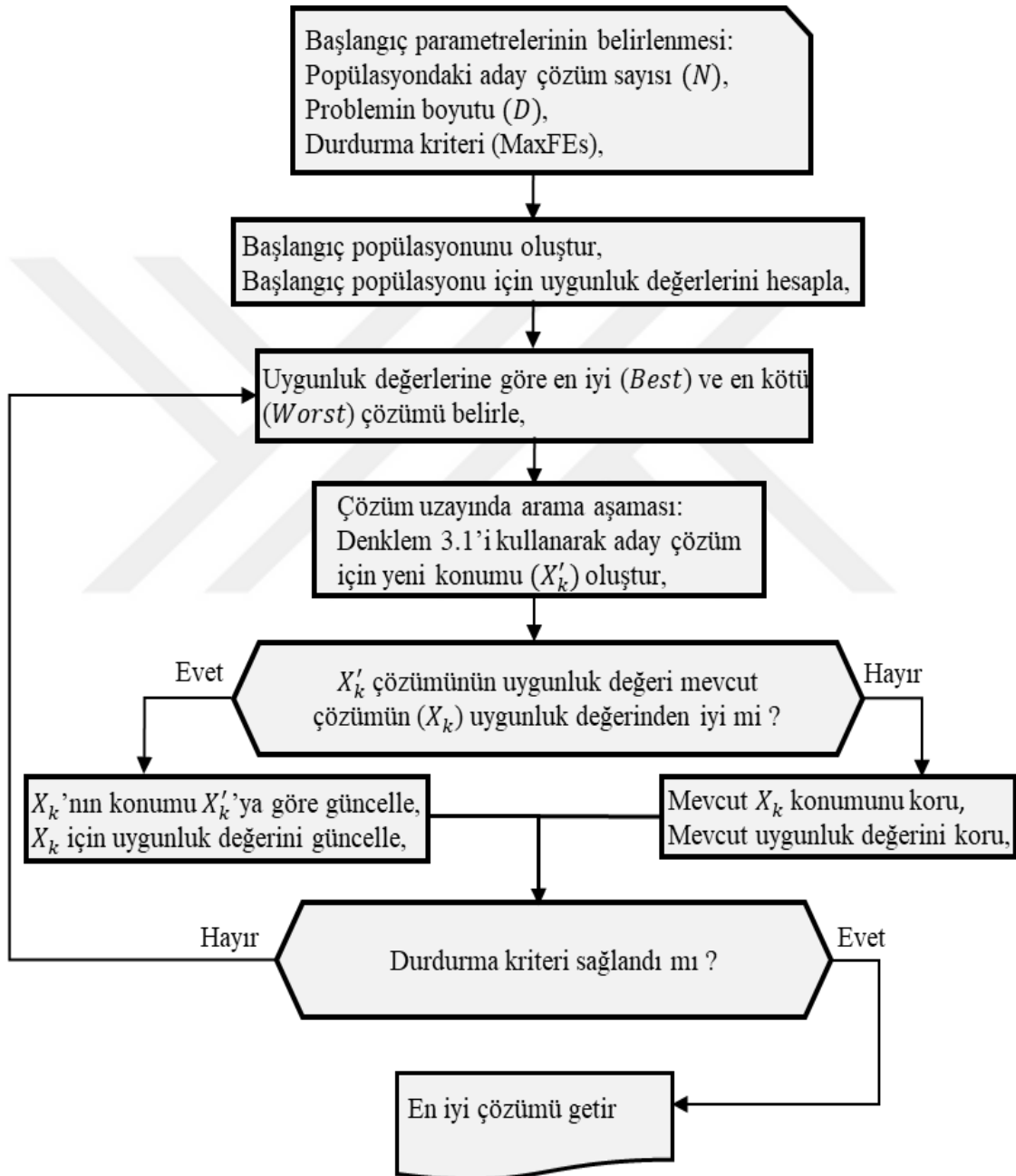
3.1.1. Temel Jaya algoritması

Jaya algoritması, kısıtlı ve kısıtsız sürekli optimizasyon problemlerinin çözümü için Rao (2016) tarafından literatüre kazandırılan popülasyon tabanlı, stokastik bir metasezgisel algoritmadır. Jaya algoritmasının en önemli özelliklerinden biri parametre bağımsız olmasıdır. Algoritmanın parametre bağımsız olma özelliği sayesinde optimizasyon problemlerine kolay bir şekilde uyarlanabilmektedir. Jaya algoritmasının bir diğer önemli özelliği ise çalışma yapısıdır; konum güncelleme işlemi sırasında belirli bir problemin en iyi çözüme doğru hareket etme eğilimini arttırırken, aynı zamanda en kötü çözümden kaçarak yerel minimum ya da yerel maksimum çözüme takılmayı önlemek üzerinedir. Bu nedenle hem popülasyondaki en iyi uygunluk değerine sahip çözümü hem de en kötü uygunluk değerine sahip çözümü konum güncelleme aşamasında kullanmaktadır. Aday çözümlerin oluşturulması aşamasında hem en iyi uygunluk değerine sahip konumun hem de en kötü uygunluk değerine sahip konumun kullanılması, arama uzayında keşif ve sömürü mekanizmaları arasında bir denge sağlamaktadır. Bu da çözüm uzayında daha geniş bir arama yapma olanağı sunmaktadır. Jaya algoritmasının konum güncelleme denklemi Denklem 3.1'de verilmiştir;

$$X'_{k,j} = X_{k,j} + r_1 * (Best_j - |X_{k,j}|) - r_2 * (Worst_j - |X_{k,j}|) \quad (3.1)$$

Denklem 3.1'de verilmiş olan: $k = (1, 2, \dots, N)$ üzerinde işlem yapılan aday çözümün indisini gösterirken, $j = (1, 2, \dots, D)$ üzerinde işlem yapılan problemin ilgili boyutunu, $Best$ popülasyondaki en iyi uygunluk değerine sahip çözümü, $Worst$ ise

popülasyondaki en kötü uygunluk değerine sahip çözümü temsil etmektedir. Son olarak r_1 ve r_2 değerleri ise $[0,1]$ arasında oluşturulan rastgele ondalıklı bir değeri göstermektedir. Durdurma kriteri ise maksimum fonksiyon değerlendirme sayısı ($MaxFEs$) olarak belirlenmiştir. Temel Jaya algoritmasının akış diyagramı Şekil 3.1’de verilmiştir.



Şekil 3.1. Temel Jaya algoritmasının akış diyagramı

3.1.2. İkili Jaya algoritması (BJA)

Jaya algoritmasının ilk ikili versiyonu olan BJA algoritması Prakash ve ark. (2017) tarafından fazör ölçüm birimlerinin optimum yerleşimi probleminin çözümü için geliştirilmiştir. BJA algoritması, temel Jaya algoritmasının konum güncelleme mekanizmasına göre aday çözüm için oluşturulan sürekli değerleri tanjant hiperbolik ($\tanh$) transfer fonksiyonunu kullanarak ayırklaştırmaktadır (Prakash ve ark., 2017). Ancak BJA algoritması, fazör ölçüm birimlerinin optimum yerleşimi probleminin çözümü için kabul edilebilir çözümler elde etmesine rağmen ikili optimizasyon problemlerini çözmek için genel bir çözüm sağlamamaktadır ve performansı, bu tez çalışmasında kullanılan ikili optimizasyon problemlerinin çözümü için zayıf kalmaktadır. BJA algoritması, Şekil 3.1’de verilen temel Jaya algoritmasının tüm adımlarını kullanmaktadır. Buna ek olarak, sürekli değerleri ikili değerlere dönüştürmek için tanjant hiperbolik transfer fonksiyonu kullanmaktadır. Ayırklaştırma sürecinde; BJA algoritması öncelikle temel Jaya algoritmasındaki Denklem 3.1’i kullanmaktadır. Daha sonra ise Denklem 3.2’yi kullanarak sürekli değerleri ayırklaştırmaktadır. Son olarak da Denklem 3.3’ü kullanarak aday çözümün ilgili boyutu için yeni konumu belirlemektedir.

$$\tanh(|X_{k,j}|) = \frac{e^{(|X_{k,j}|) - 1}}{e^{(|X_{k,j}|) + 1}} \quad (3.2)$$

$X_{k,j}$ aday çözümün ilgili boyutunu temsil etmektedir. Algoritma öncelikle temel Jaya’daki Denklem 3.1’i kullanmaktadır. Daha sonra elde edilen değer tanjant hiperbolik transfer fonksiyonuna gönderilmektedir. Tanjant hiperbolik transfer fonksiyonu da $X_{k,j}$ ’nin mevcut değerini ‘0’ ya da ‘1’ olacak şekilde ikili bir ayırık değere dönüştürmektedir. $X_{k,j}$ ’nin ‘0’ ya da ‘1’ değerine dönüştürme işlemi Denklem 3.3’te verilmiştir.

$$X_{k,j} = \begin{cases} 1 & \text{Eğer } \text{rand}() < \tanh(|X_{k,j}|) \\ 0 & \text{Değilse,} \end{cases} \quad (3.3)$$

Denklem 3.3’te verilen $\text{rand}()$ ifadesi $[0,1]$ arasında rastgele oluşturulan ondalıklı bir değeri temsil etmektedir. Denklem 3.3 incelendiğinde, tanjant hiperbolik

fonksiyonu sonucunda $X_{k,j}$ 'nin aldığı değer, $[0,1]$ arasında rastgele oluşturulan sürekli değerden küçükse $X_{k,j}$ 'nin yeni konumu '1' değilse $X_{k,j}$ 'nin yeni konumu '0' olmaktadır.

3.2. İkili Optimizasyon Problemlerinin Çözümü için Önerilen Algoritmalar

Bu çalışmada ikili optimizasyon problemlerinin çözümü için Jaya algoritması tabanlı yeni yaklaşımlar geliştirilmiştir. Geliştirilen ilk yaklaşım JayaX olarak adlandırılan; temelini Jaya algoritması ve 'özel veya' (XOR) lojik fonksiyonundan alan yaklaşımdır. Geliştirilen diğer yaklaşım ise JayaX algoritmasının yerel arama yönünü iyileştirmek için JayaX-LSM olarak adlandırılan, geliştirilen ilk algoritma olan JayaX algoritması ve Yerel Arama Modülü (Local Search Module - LSM) olarak adlandırılan yöntemin birlikte kullanılması ile geliştirilen yaklaşımdır.

3.2.1. XOR tabanlı ikili Jaya algoritması (JayaX)

Bu çalışmada ikili optimizasyon problemlerinin çözümü için 'XOR' lojik fonksiyonu ve Jaya algoritması tabanlı bir yaklaşımın geliştirilmesi amaçlanmaktadır. Cinar ve Kiran (2018), yapmış oldukları çalışmada temel TSA'yı ikili TSA'ya dönüştürmek için lojik kapıları ve benzerlik ölçüm tekniklerini (similarity measurement techniques) birlikte kullanmışlardır. Yapılan bu çalışmada da Cinar ve Kiran (2018) yapmış oldukları çalışmadan ilham alınarak, ikili optimizasyon problemlerinin çözümü için 'XOR' mantıksal kapısı ve Jaya algoritması birlikte kullanılmıştır. Temel lojik kapılar 'VE' (AND), 'VEYA' (OR), 'DEĞİL' (NOT) ve 'ÖZEL VEYA' (XOR) kapıları olmak üzere dört adettir. İkili optimizasyonda, aday çözümün her boyutu sadece '1' ya da '0' değerini alabilmektedir. Bu nedenle, algoritmalarda kullanılan mantıksal (lojik) kapı oldukça önemlidir. Şekil 3.2'de lojik kapılar için doğruluk tablosu verilmiştir.

Karar değişkenleri		Lojik kapılar		
a	b	$a + b$ (OR)	$a \& b$ (AND)	$a \oplus b$ (XOR)
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0
		%25=0	%75=0	%50=0
		%75=1	%25=1	%50=1

Şekil 3.2. Lojik kapılar için doğruluk tablosu ve '0' ve '1' değerleri için yüzdelik dilimler

Şekil 3.2’de verilen + sembolü ‘OR’ lojik kapısını , $\oplus$ sembolü ‘XOR’ lojik kapısını ve & sembolü ‘AND’ lojik kapısını temsil etmektedir. Şekil 3.2’de verilmiş olan a ve b sembolleri ise konum güncelleme işlemi için kullanılan iki aday çözümün karşılıklı boyutlarını göstermektedir. Aday çözümün konum güncellenmesi sırasında ‘OR’ kapısı kullanılırsa sadece tek bir durumda ilgili boyutun değeri ‘0’ olmaktadır, o da a ve b değerlerinin ‘0’ olması durumudur. Eğer güncelleme işlemi için ‘AND’ kapısı kullanılırsa aday çözümün ilgili konumu sadece tek bir durumda ‘1’ değerini almaktadır. Bu nedenle konum güncelleme işlemi için seçilecek lojik kapısı oldukça önemlidir. Şekil 3.2 incelendiğinde konum güncelleme işlemi için ‘OR’ kapı kullanılırsa, aday çözümün ilgili boyutu %75 olasılıkla ‘1’ olacaktır. Güncelleme işlemi için ‘AND’ kapısı kullanılırsa aday çözümün ilgili boyutu %25 olasılıkla ‘1’ olacaktır. Sonuç olarak, Şekil 3.2’de verilen doğruluk tablosu kontrol edildiğinde, konum güncelleme işlemi için ‘XOR’ kapısı seçildiğinde ‘0’ ve ‘1’ değerlerinin seçim olasılığının eşit olduğu görülecektir (Kiran ve Gunduz, 2013).

Jaya algoritmasının konum güncelleme aşamasında kullanılan Denklem 3.1 sürekli optimizasyon problemlerinin çözümü için geliştirilmiş olması nedeniyle, ikili optimizasyon problemlerinin çözümü için güncellenmesi gerekmektedir. Bu kapsamda Denklem 3.1 yeniden tasarlanmış ve ikili optimizasyon problemlerinin çözümüne uygun olması amacıyla ‘XOR’ tabanlı bir denklem önerilmektedir. Bu çalışmada konum güncelleme aşamasında Denklem 3.4 kullanılmaktadır.

$$X'_{k,j} = \{ X_{k,j} \oplus (Best_j \oplus Worst_j) \} \quad (3.4)$$

Denklem 3.4’te X'_k aday çözümü, X_k konum güncelleme aşamasında kullanılacak olan popülasyondaki k . çözümü, $Best$ popülasyondaki en iyi uygunluk değerine sahip çözümü, $Worst$ ise popülasyondaki en kötü uygunluk değerine sahip çözümü göstermektedir. Denklem 3.4’de verilen X_k , $Best$ ve $Worst$ çözümleri temel Jaya algoritmasının konum güncelleme aşamasında da kullanılan çözümlerdir. Denklem 3.4 incelendiğinde anlaşılmaktadır ki temel Jaya algoritmasının ana iskeleti korunmuştur ve böylece Jaya algoritmasının sağladığı avantajlar ikili optimizasyon problemlerinin çözümünde de kullanılabilir. Sonuç olarak, önerilen JayaX algoritmasında, hem temel Jaya algoritmasının çalışma yapısına sadık kalınmış hem de ayrık ikili optimizasyon problemlerinin çözümü için ‘XOR’ mantıksal kapısı eklenmiştir.

Temel Jaya algoritmasında yapılan bir diğer yenilik ise konum güncelleme aşamasında her boyutta güncelleme işleminin yapılmamasıdır. Temel Jaya algoritmasında konum güncelleme aşamasında X_k çözümünün her boyutunda güncelleme işlemi yapılmaktadır. Bu da rastsallığı arttırdığı için arama uzayında büyük bir pertürbasyona sebep olmakta ve bazen en iyi çözümden uzaklaşmaya neden olabilmektedir. Temel Jaya algoritmasının bu eksikliğini üstesinden gelebilmek için algoritmaya yeni bir mekanizma eklenmiştir. Bu mekanizma sayesinde problemin boyutuna bağlı olarak, her bir yinelemede güncellenecek boyut sayısı ve ilgili boyutlar belirlenmektedir. Temel Jaya algoritmasına eklenen bu mekanizma sayesinde JayaX algoritmasında, arama uzayında çözüm araştırması sırasında yerel arama ile global arama arasında denge sağlanmış olmaktadır. Bu çalışma kapsamında önerilen algoritmaya eklenmiş olan yeni mekanizmanın sözde kodu Şekil 3.3'te verilmiştir.

Adım 1: Konum güncelleme aşamasında seçilecek boyut sayısını belirle

Olasılık (Probability - P) parametresini $[0,1]$ arasında olacak şekilde belirle,
 Problemin boyutsallığına (D) bağlı olarak güncelleme işlemi için boyut sayısını (d) belirle,
 Güncellenecek karar değişkenlerinin sayısını belirle ($d = \text{round}(D * P)$),

Adım 2: Konum güncelleme aşamasında aday çözümün güncellenecek boyutlarını belirle

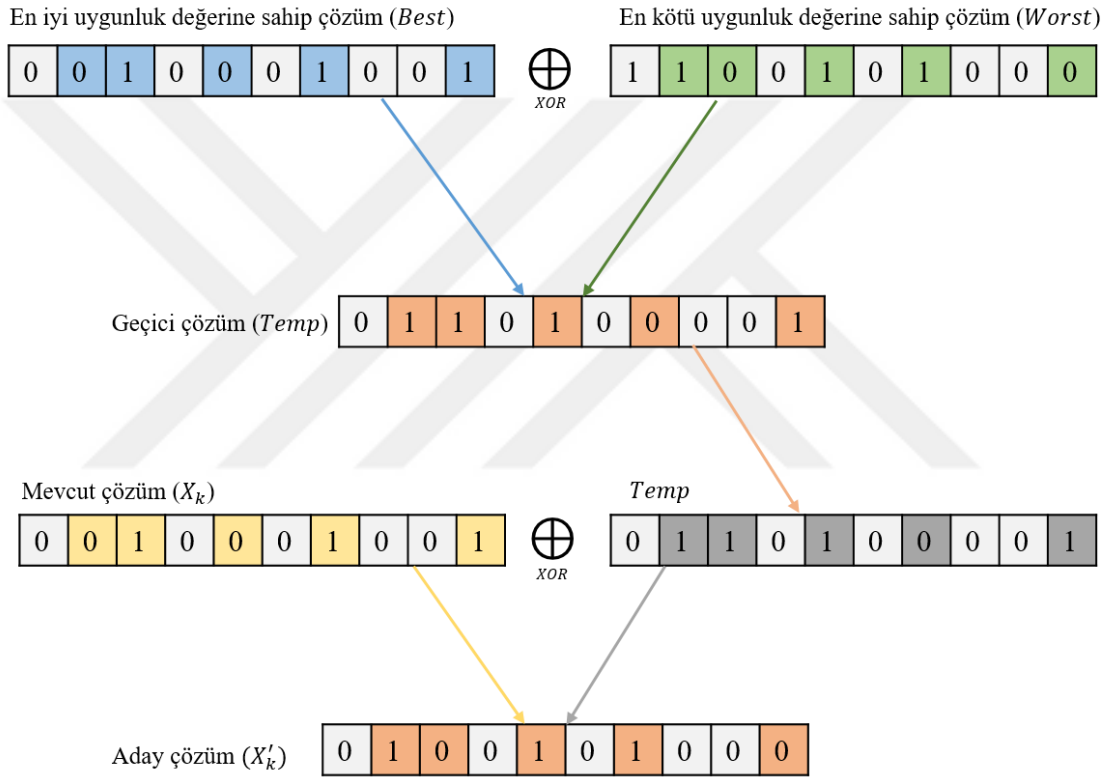
d Adet karar değişkenlerini rastgele seç,

Adım 3: Güncellenecek boyutları getir

Şekil 3.3. Konum güncelleme aşamasında aday çözümün güncellenecek boyutlarının belirlenmesi

Temel Jaya algoritması üzerinde yapılan değişikliklerinin ve eklenen yeniliklerin daha iyi anlaşılabilmesi için, yapılan değişiklikler aşağıda verilen örnek üzerinde gösterilmiştir. Bu örnekte kullanılan değerler rastgele oluşturulmuştur. Örnekte seçilen problemin boyutu $D = 10$ olarak seçilmiştir. Olasılık parametresi olan $P = 0.50$ olarak belirlenmiştir. Aday çözümün konum güncellemesi aşamasında güncellenecek boyut sayısı da $d = D * P$ formülasyonuna göre $d = 5$ olmaktadır. Sonuç olarak bir yinelemede aday çözümün 5 boyutunda güncelleme işlemi gerçekleştirilmektedir. Konum güncelleme aşamasında güncellenecek bu 5 boyut, her yinelemede rastgele seçilmektedir ve bu örnek için konum güncelleme aşamasında aday çözümün 2, 3, 5, 7 ve 10'uncu boyutları güncellenecektir. Aday çözüm Şekil 3.4'te verilen aşamalara göre oluşturulmaktadır. Konum güncelleme aşamasında kullanılacak olan 2, 3, 5, 7 ve

10'uncu boyutlar renkli olarak işaretlenmiştir. Denklem 3.4'e göre öncelikle en iyi uygunluk değerine sahip çözüm olan *Best* çözümü ile en kötü uygunluk değerine sahip çözüm olan *Worst* çözümlerine 'XOR' mantıksal operatörü uygulanmaktadır. 'XOR' işlemi sonrasında oluşan çözüm geçici (*Temp*) çözüm olarak hafızada tutulmaktadır. *Best* ve *Worst* çözümlere 'XOR' işleminin uygulanmasından sonra oluşan *Temp* çözümü Şekil 3.4'te verilmiştir. İkinci aşamada ise mevcut çözüm olan X_k çözümü ile *Temp* çözümünün 2, 3, 5, 7 ve 10'uncu boyutlarına 'XOR' işlemi uygulanmaktadır. İkinci aşama da tamamlandıktan sonra X'_k aday çözümü oluşmaktadır.



Şekil 3.4. JayaX algoritmasında önerilen konum güncelleme kuralına göre aday çözümün oluşturulması

3.2.2. Yerel arama modülü eklenmiş ikili Jaya algoritması (JayaX-LSM)

Önerilen JayaX-LSM algoritması, Bölüm 3.2.1'de verilmiş olan JayaX algoritmasının tüm adımlarını kullanmaktadır. Bununla birlikte önerilen JayaX algoritmasının etkinliğini artırmak amacıyla JayaX algoritmasına bir yerel arama algoritması da dâhil edilmiştir. İkili optimizasyon problemlerinin çözümü için geliştirilen bu algoritmaların performans analizi deneysel sonuçlar bölümünde değerlendirilmiştir.

Yerel arama modülü (Local Search Module - LSM) aşağıda verilmiş olan bölümde detaylandırılmıştır.

3.2.2.1. Yerel arama modülü (LSM)

Önerilen algorithma kullanılan yerel arama modülü, basit bir komşuluk arama yaklaşımıdır. LSM mevcut bir çözümle başlar ve mevcut çözümün yakınlarında mevcut çözümden daha iyi bir çözüm olup olmadığını araştırmaktadır. Kashan ve ark. (2011) yapmış oldukları çalışmada KTYP'nin çözümü için önerdikleri DisABC algoritmasına basit ama etkili bir yerel arama algoritmasını eklemişlerdir. Kashan ve ark. önerdikleri algorithma yerel arama yapıp yapılmayacağı P_{local} parametresine göre belirlenmektedir. Yani önerdikleri algorithma her yinelemede yerel arama işlemi yapılmamaktadır. Böylece, P_{local} parametresi sayesinde algoritmanın yerel minimum çözüme takılmasının önüne geçtiklerini belirtmişlerdir (Kashan ve ark., 2011). Bir başka çalışmada Zhang ve ark. (2016), ikili AAA'nın (BAAA) performansını artırmak için *EliteLocalSearch* adlı açgözlü bir yerel arama algoritması kullanmışlardır. Önerdikleri algorithma yerel arama algoritmasını her yinelemede kullanmışlardır. Eğer yerel arama işlemli sıklıkla yapılırsa, bir noktadan sonra çözümün yerel minimuma takılma ihtimali yüksektir ve keşif ile sömürü arasındaki denge bozulabilmektedir. Yapılan bu çalışmada da *EliteLocalSearch* yerel arama algoritmasının adımları kullanılmıştır. Ayrıca bu algorithma Kashan ve ark. (2011) çalışmalarında kullandıkları P_{local} kontrol parametresi de dâhil edilmiştir. P_{local} parametresi, önerilen algoritmanın her iterasyonda (yinelemede) yerel arama algoritmasının kullanmasını engellemektedir ve bu sayede daha az sayıda fonksiyon değerlendirme sayısı (FEs) harcanmaktadır. Sonuç olarak LSM mekanizması sayesinde keşif ve sömürü mekanizmalarının etkinliğinin artırılması amaçlanmaktadır. LSM'nin sözde kodu Şekil 3.5'te verilmiştir.

```

IF  $rand() < P_{local}$  //  $rand()$ , [0,1] arasında oluşturulan rastgele bir değer,
     $Best =$  En iyi uygunluk değerine sahip çözüm
    FOR  $j = 1$  to  $D$  //  $D$  problemin boyutunu gösterir.
        FOR  $jj = 1$  to  $D$ 
            IF ( $Best^j \neq Best^{jj}$ )
                //  $Best$  çözümünün  $j$ . ve  $jj$ . boyutlarının değerlerini karşılıklı olarak değiştir,
                 $x = swap(Best, j, jj)$ ; //  $x$  geçici çözümü temsil etmektedir.
                 $x$  çözümü için uygunluk (fitness) değerini hesapla,
                IF ( $x$  çözümünün uygunluk değeri  $Best$  çözümünün uygunluk değerinden iyi ise)
                    // En iyi uygunluk değerine sahip çözümün ( $Best$ ) konumunu güncelle,
                     $Best = x$ ;
                END IF
                // Fonksiyon değerlendirme sayısını güncelle,
                 $FES = FES + 1$ ;
            END IF
        END FOR
    END FOR
END IF

```

Şekil 3.5. JayaX algoritmasında kullanılan yerel arama modülünün çalışma adımları

JayaX algoritmasına eklenen bu yerel arama modülü sayesinde, önerilen algoritmanın daha geniş bir arama uzayında çözüm araştırması yapılması amaçlanmaktadır. Şekil 3.6'da JayaX-LSM algoritmasının sözde kodu verilmiştir. JayaX algoritması için ise ayrıca çalışma adımları verilmemiştir. Şekil 3.6'da JayaX-LSM algoritması için verilmiş olan çalışma adımlarından Adım 5 çıkarıldığında, geriye kalan diğer adımlar JayaX algoritmasının çalışma adımlarını göstermektedir.

Adım 1: Algoritmanın başlatılması

Problemin boyutunu belirle (D),
 Popülasyon boyutunu belirle (N),
 Durdurma kriterini belirle ($MaxFES$),
 Olasılık parametresinin değerini belirle (P), // $[0,1]$ arasında olacak şekilde bir değer belirle
 P_{local} parametresinin değerini $[0,1]$ arasında olacak şekilde belirle,

Adım 2: Başlangıç popülasyonunun oluşturulması

N adet aday çözümü D -boyutlu arama uzayında problemin tanım aralığında olacak şekilde rastgele oluştur,

N adet aday çözümün uygunluk değerini amaç fonksiyonuna (objective function) göre hesapla,

En iyi ($Best$) ve en kötü ($Worst$) uygunluk değerine sahip çözümleri belirle,

Fonksiyon değerlendirme sayısını belirle ($FES = N$),

Adım 3: Çözüm uzayında arama işleminin başlatılması

WHILE $FES \leq MaxFES$

FOR Her bir aday çözüm için (X_k)

$d = round(D * P)$; // güncellenecek boyut sayısı belirle,

Problemin boyutları ($j = 1, 2, \dots, D$) arasından d adet farklı boyutu rastgele belirle,

// X'_k çözümünün yeni konumu oluşturulurken X_k çözümünün d adet boyutunu güncelle

Denklemler 3.4'e göre X'_k için yeni konumu oluştur,

X'_k çözümü için uygunluk değerini hesapla,

IF X'_k çözümünün uygunluk değeri X_k çözümünün uygunluk değerinden daha iyi ise

$X_k = X'_k$; // X_k çözümünün mevcut konumunu güncelle,

$Fitness(X_k) = Fitness(X'_k)$; // X_k çözümünün uygunluk değerini güncelle,

END IF

$FES = FES + 1$;

END FOR

Adım 4: En iyi ($Best$) ve en kötü ($Worst$) uygunluk değerine sahip çözümleri belirle

En iyi ($Best$) uygunluk değerine sahip çözümü bul,

En kötü ($Worst$) uygunluk değerine sahip çözümü bul,

Adım 5: En iyi ($Best$) uygunluk değerine sahip çözüm için yerel arama işlemini gerçekleştir

IF $rand() < P_{local}$ // $rand()$ $[0,1]$ arasında rastgele oluşturulan bir değer,

$Best$ çözümünün konumu Şekil 3.5'te verilen LSM algoritmasına göre güncelle,

END IF

Adım 6: Sonlandırma kriterini kontrol et

IF Sonlandırma kriteri sağlanmamışsa

Adım 3'e git,

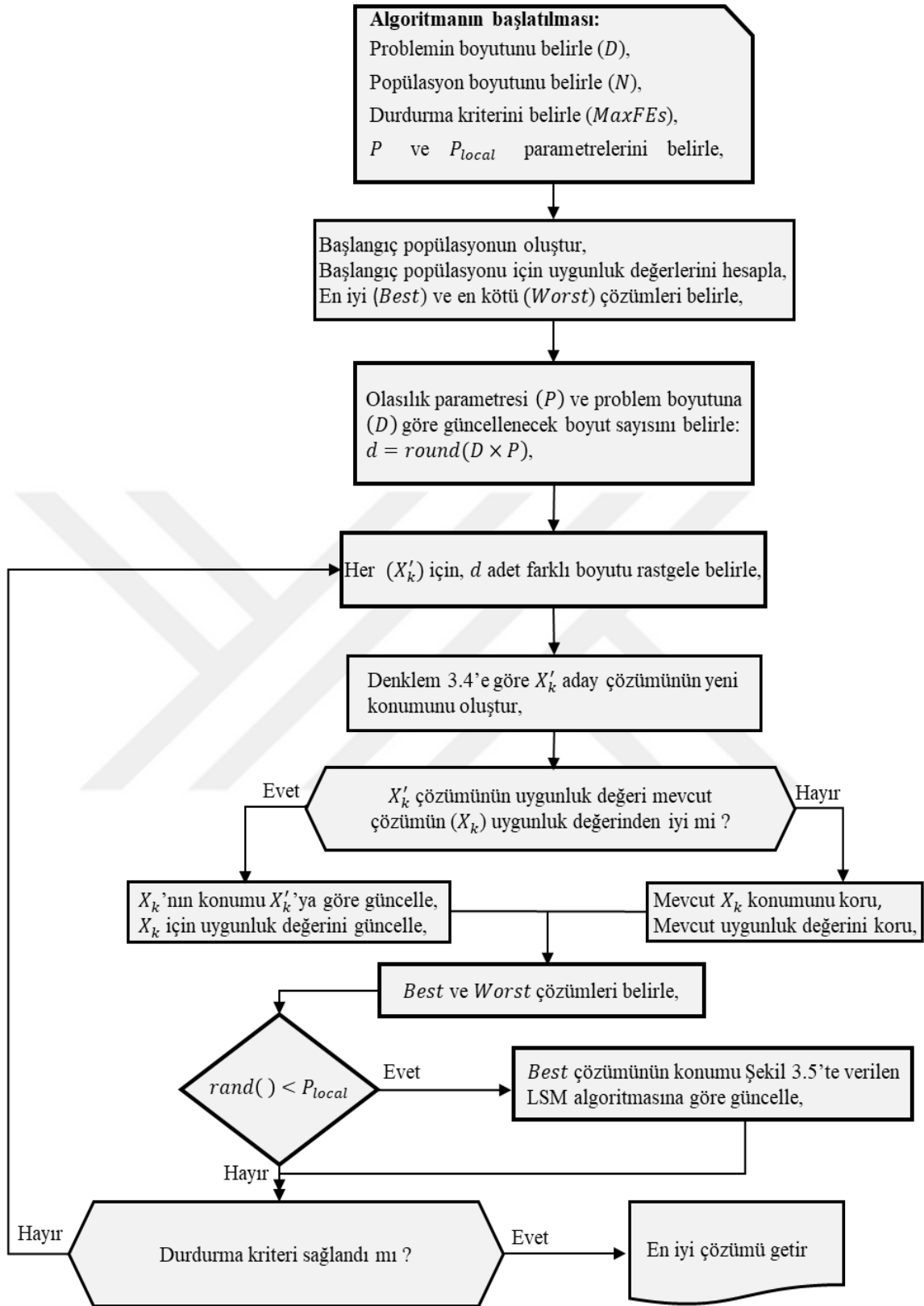
ELSE

En iyi ($Best$) uygunluk değerine sahip çözümü getir.

END WHILE

Şekil 3.6. JayaX-LSM algoritmasının çalışma adımları

Şekil 3.7'de ise JayaX-LSM algoritmasına ait akış diyagramı verilmiştir.



Şekil 3.7. JayaX-LSM algoritmasının akış diyagramı

3.3. Ayırık Tam Sayı Optimizasyon Problemlerinin Çözümü için Önerilen Jaya Algoritması (DJaya)

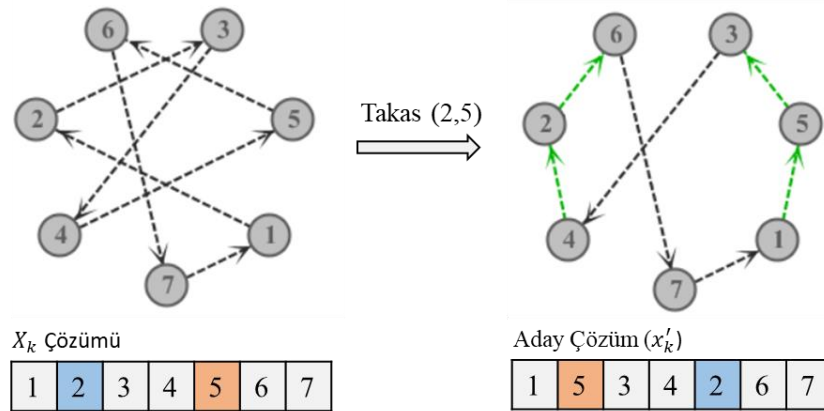
Tez çalışmasının bu aşamasında ayırık tam sayı optimizasyon problemlerinin çözümü için Jaya algoritması tabanlı DJaya olarak adlandırılan ayırık Jaya algoritması önerilmiştir. Jaya algoritmasının konum güncelleme aşamasında kullanılan Denklem 3.1 sürekli optimizasyon problemlerinin çözümü için geliştirildiğinden dolayı, ayırık tam sayı optimizasyon problemlerinin çözümü için değiştirilmesi gerekmektedir. Bu kapsamda Jaya algoritmasının konum güncelleme denklemi yeniden tanımlanmıştır. Önerilen DJaya algoritmasında, Jaya algoritmasının ana iskeleti korunmuştur. Temel Jaya algoritmasında konum güncelleme aşamasında X_k , $Best$ ve $Worst$ çözümleri kullanılmaktadır. Aynı şekilde DJaya algoritmasında da aday çözümün konum güncellemesi aşamasında X_k , $Best$ ve $Worst$ çözümleri kullanılmaktadır. Kiran (2015), TSA yönteminin keşif kabiliyetini geliştirmek için temel ağaç tohum algoritmasına arama eğilimi (Search Tendency - ST) parametresini dâhil etmiştir. Önerilen DJaya yöntemine de, çözüm uzayında keşif ve sömürü kabiliyeti arasında bir denge kurulması amacıyla $ST1$ ve $ST2$ olarak adlandırılan iki arama eğilimi parametresi dâhil edilmiştir. X'_k aday çözümünün oluşturulması aşamasında, X_k , $Best$ ve $Worst$ çözümlerinden biri $ST1$ ve $ST2$ parametrelerine göre seçilmektedir. $ST1$ ve $ST2$ parametreleri $[0, 1]$ aralığında bir değer olarak belirlenmektedir. $ST1$ ve $ST2$ parametrelerinin değerlerini belirlemek amacıyla deneysel sonuçlar bölümünde parametre analizi yapılmıştır.

Ayırık ağaç tohum algoritması, durum geçiş algoritması ve ayırık durum geçiş algoritmaları ayırık tam sayı problemlerinin çözümü için algoritmik yapılarında aday çözümler oluşturma işlemi sırasında takas (swap), öteleme (shift) ve simetri (symmetry) komşuluk operatörlerini kullanılmışlardır (Chunhua ve ark., 2012; Zhou ve ark., 2016; Cinar ve ark., 2019). Yapılan bu çalışmada da Jaya algoritmasının ayırıklaştırılması sürecinde bu komşuluk operatörleri kullanılmıştır. Öte yandan, bu algoritmalarından farklı olarak rulet tekerleği seçimi (Roulette Wheel Selection) (Potvin, 1996) önerilen algoritmaya dâhil edilmiştir. Rulet tekerleği seçimi kullanılmasının sebebi hem zayıf çözümlere hem de güçlü çözümlere seçilme şansı vermektir. Çünkü bazen zayıf çözümler arama uzayında daha iyi çözümlere ulaşmamıza ve yerel minimum çözüme takılmamızı önlemektedir. Önerilen algorithmda takas, öteleme ve simetri komşuluk operatörlerinin konum güncelleme aşamasında rulet tekerleğine göre seçilme durumları eşit bir seçim

olasılığı ile başlamaktadır. Ancak iterasyonlar ilerledikçe komşuluk operatörlerinin seçilme olasılığı aday çözüm için daha iyi bir konum oluşturup, oluşturmama durumuna göre değişmektedir.

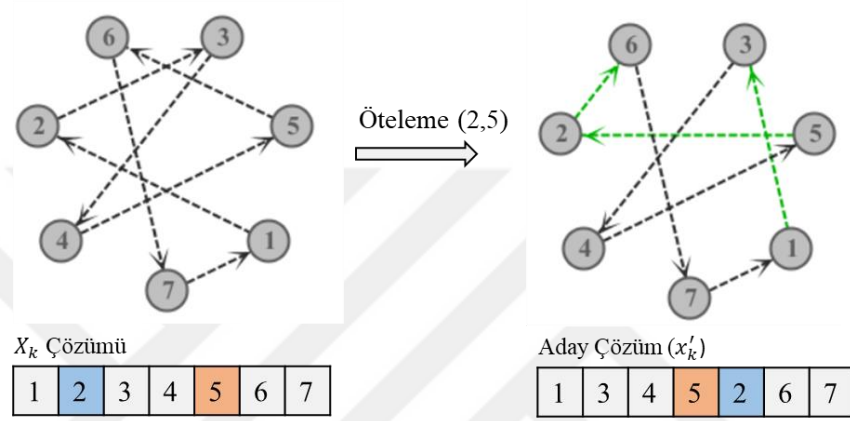
Komşuluk operatörlerinin sekiz farklı kombinasyonu olan; takas, öteleme, simetri, eşit seçilmeye sahip *takas+öteleme* (*swap+shift*), eşit seçilmeye sahip *takas+simetri* (*swap+symmetry*), eşit seçilmeye sahip *öteleme+simetri* (*shift+symmetry*), Kombine 1 (takas, öteleme ve simetri operatörlerinin eşit seçimine dayanır) ve Kombine 2 (takas, öteleme ve simetri operatörünün rulet tekerleği seçimine dayanır), sırasıyla konum güncelleme mekanizmasında kullanılmıştır ve bu sekiz farklı kombinasyonlu komşuluk operatörleri için deneysel bulgular aşamasında parametre analizi yapılmış olup önerilen algorithmada Kombine 2 yönteminin konum güncelleme mekanizmasında kullanılmasına karar verilmiştir. Algoritmayı sonlandırma koşulu sağlandıktan sonra, en iyi çözümün toplam tur uzunluğunu azaltmak amacıyla en iyi uygunluk değerine sahip çözüme (*Best*) 2-opt (Croes, 1958) yerel arama yöntemi uygulanmıştır. Aday çözümün (X'_k) üretilmesi aşamasında kullanılan komşuluk operatörleri aşağıdaki gibi tanımlanmaktadır:

Takas operatörü: 1 ile problemin boyutu (D) arasında iki farklı boyut (nokta) belirlenir. GSP’de, D şehir sayısını göstermektedir. İki farklı nokta belirlendikten sonra takas operatörü kullanılarak, aday çözümün bu iki boyutunun değeri birbiri ile değiştirilir. Şekil 3.8’de takas operatörü kullanılarak güncelleme işleminin nasıl yapıldığı açıklanmıştır. Verilen problemde şehir sayısı 7 olarak seçilmiştir ve takas edilecek boyutlar 2. ve 5. boyutlar olarak belirlenmiştir.



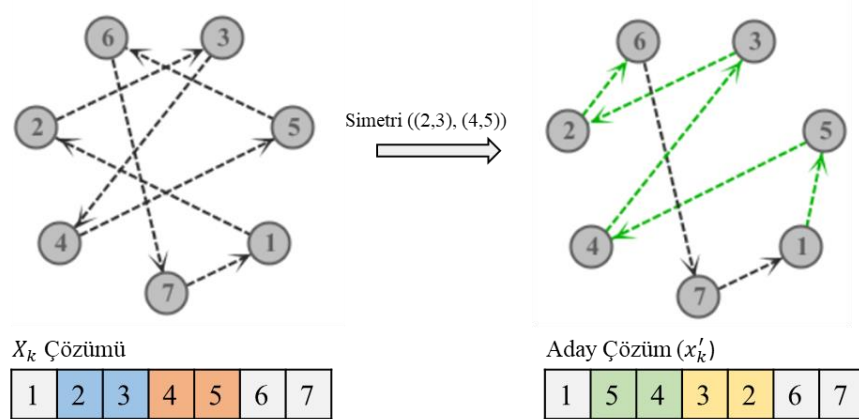
Şekil 3.8. GSP için kullanılan takas operatörü

Öteleme operatörü: 1 ile problemin boyutu (D) ($[1, D]$) arasında olacak şekilde x ve y şeklinde iki farklı boyut belirlenir. Daha sonra, x . konumunun bilgisi hafızaya kaydedilir ve $[x, y]$ aralığındaki konumlar aday çözüm vektörü üzerinde sola kaydırılır. En sonunda da y . boyutun konum bilgisi x . boyut için hafızaya alınmış olan konum bilgisi ile güncellenir. Şekil 3.9’da öteleme operatörü kullanılarak, güncelleme işleminin nasıl yapıldığı açıklanmıştır. Verilen problemde şehir sayısı 7 olarak seçilmiştir ve öteleme işlemi için seçilen x ve y konumları, sırasıyla 2 ve 5’tir.



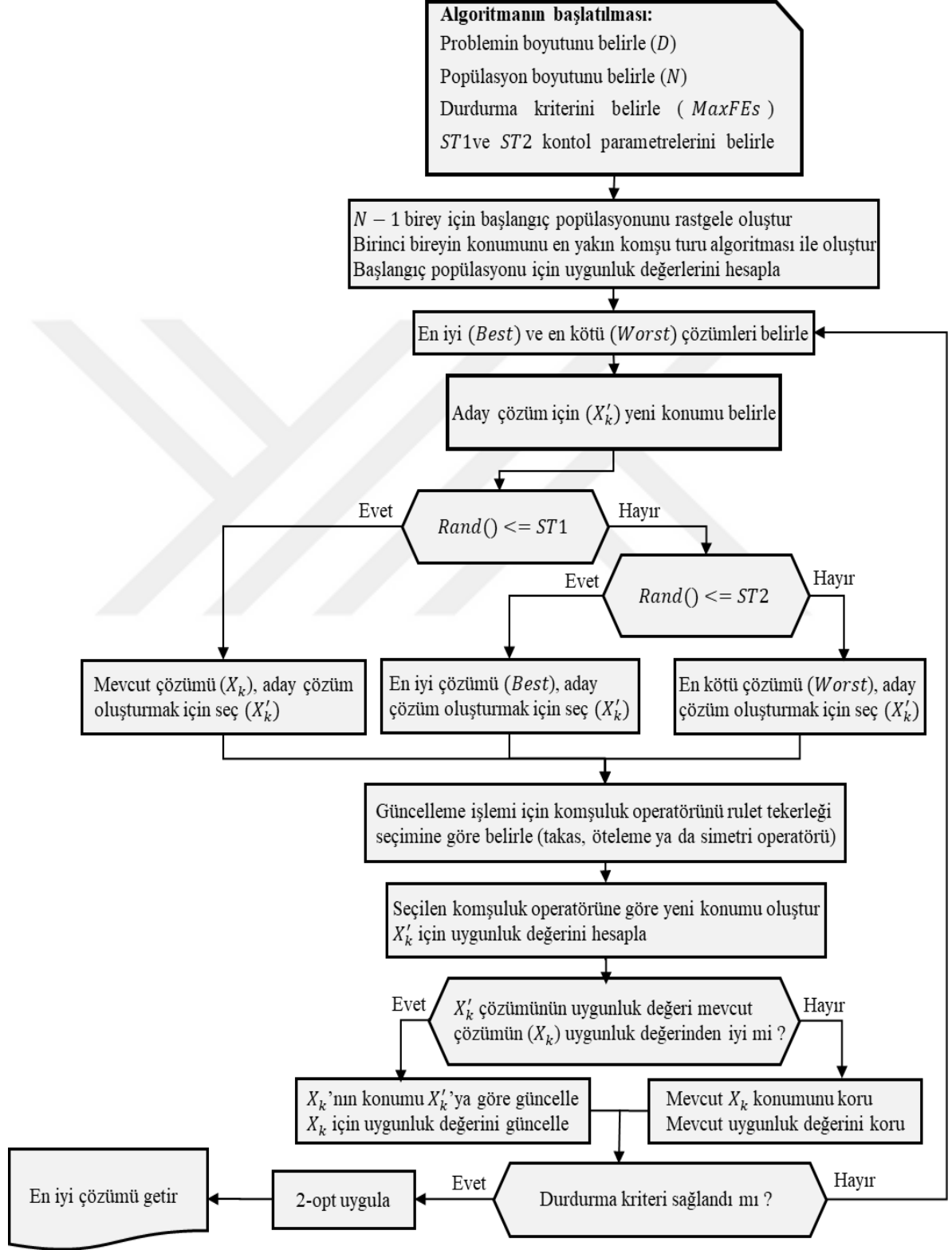
Şekil 3.9. GSP için kullanılan öteleme operatörü

Simetri operatörü: $[1, D]$ arasında olacak şekilde iki nokta belirlenir. Sonra bu noktalara göre ardışık konumlarda bulunan iki dizi rastgele belirlenir. Daha sonra, belirlenen bu dizilerin konum bilgisi kendi içine ters çevrilir ve son olarak bu dizilerin konumları birbiri ile takas edilir. Simetri operatörü X'_k çözümünü oluştururken X_k çözümü üzerinde büyük değişiklikler gerçekleştirmektedir. Şekil 3.10’da simetri operatörü kullanılarak, güncelleme işleminin nasıl yapıldığı açıklanmıştır.



Şekil 3.10. GSP için kullanılan simetri operatörü

Ayrık tam sayı problemlerinin çözümü için komşuluk operatörlerine ve Jaya algoritmasına dayalı önerilen ayrık Jaya (DJaya) algoritmasının akış diyagramı Şekil 3.11’de verilmiştir.



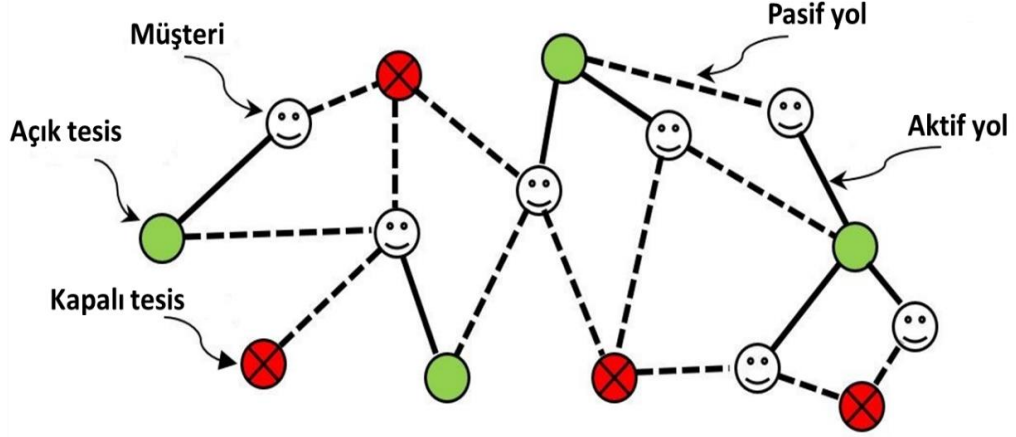
Şekil 3.11. Ayrık Jaya (DJaya) algoritmasının akış diyagramı

3.4. Optimizasyon Problemleri

Tez çalışması kapsamında önerilen algoritmaların performans analizi için deney aşamasında; KTYP, CEC 2015 nümerik fonksiyonları, rüzgâr türbini yerleştirme problemi ve GSP kullanılmıştır. Kapasitesiz tesis yerleştirme, CEC 2015 nümerik fonksiyonları ve rüzgâr türbini yerleştirme problemleri ikili optimizasyon problemleri kategorisinde çözülmüşlerdir. GSP ise ayrık tam sayı optimizasyon problemi kategorisinde çözülmüştür.

3.4.1. Kapasitesiz tesis yerleştirme problemi (KTYP)

Bu çalışmada, KTYP'nin çözümü için OR-Library (Beasley, 1990) kütüphanesinden alınan veri seti kullanılmıştır. KTYP'de hizmet alan müşteriler ve hizmet veren tesisler vardır. Tesislerin her biri farklı yerlerde konumlanmaktadır ve sabit kurulum maliyetlerine sahiptirler. Müşterilerin her biri farklı konumlarda yer alabilmektedir ve tesisler arasındaki mesafeyle orantılı olarak hizmete ulaşma maliyetleri vardır. Müşterilerin tüm taleplerinin, ulaştırma maliyeti en düşük olan tesisler tarafından karşılanması gerekmektedir. Problemin toplam maliyeti, açık tesislerin kurulum maliyetleri ile müşteriler ve tesisler arasında bulunan hizmet ulaştırma maliyetlerinin toplamına eşittir. KTYP'de temel amaç, minimum toplam maliyetle açık ve kapalı tesisleri belirlemektir. Açık veya kapalı toplam tesis sayısı n olmak üzere, bu problem için 2^n farklı çözüm vardır. Ancak KTYP'de en az bir tane tesisin açık olması şartı vardır. Bu nedenle olası çözümler kümesi $(2^n - 1)$ tane çözümdür. Bu açıklamalardan da görüldüğü gibi n tesis sayısı, problemin karmaşıklığını doğrudan etkilemektedir. KTYP'de tesis sayısının artması, problemin karmaşıklığını da arttırmakta ve polinomsal zamanda problemin çözümünü zorlaştırmaktadır. Bu nedenle KTYP NP-Zor sınıfı problemler kategorisine girmektedir (Krarup ve Pruzan, 1983; Monabbati ve Kakhki, 2015). KTY probleminin polinomsal zaman da çözülebilmesi için tesislerin kurulum maliyetlerinin sıfır olarak kabul edilmesi gerekmektedir. Böylece bir müşteri herhangi bir tesisden herhangi bir anda hizmet alabilecektir. KTY probleminin daha iyi anlaşılabilmesi için basit bir gösterimi Şekil 3.12'de verilmiştir.



Şekil 3.12. Kapasitesiz tesis yerleştirme probleminin basit bir gösterimi

Şekil 3.12’de *yeşil* çemberler açık tesisleri, *kırmızı* çemberler kapalı tesisleri ve *beyaz* gülümseyen yüzler ise müşterileri göstermektedir. Müşteri ile tesis arasındaki pasif bağlantı anlamına gelen *kesikli* çizgiler, müşterinin ilgili tesisi kullanmadığı anlamına gelmektedir. Müşteri ile tesis arasındaki aktif bağlantı anlamına gelen *düz* çizgiler ise müşterinin ilgili tesisi kullandığı göstermektedir. KTYP’de amaç fonksiyonu şu şekilde hesaplanır (Ghosh, 2003):

$$\begin{aligned}
 I &= \{i_1, i_2, \dots, i_m\} && , \text{Açık ve kapalı tesisler kümesi} \\
 J &= \{j_1, j_2, \dots, j_n\} && , \text{Potansiyel müşteri kümesi} \\
 F &= (f_1, f_2, \dots, f_m) && , \text{Her bir tesis için sabit kurum maliyeti} \\
 C &= \begin{bmatrix} c_{11} & c_{1,2} & \dots & c_{1,n} \\ \dots & & & \\ c_{m1} & c_{m,2} & \dots & c_{m,n} \end{bmatrix} && , \text{Müşteriye hizmet ulaştırma maliyeti} \\
 &&& \text{matrisi}
 \end{aligned}$$

$$\arg \min \{ \delta(S) = \sum_{i \in S} f_i + \sum_{j \in J} \min \{ c_{i,j} \mid i \in S \} \} \quad (3.5)$$

Denklem 3.5’te verilen S , I kümesinin boş olmayan alt kümesini göstermektedir. KTYP için amaç maliyetin en az olduğu S kümesine ulaşmaktır. S kümesinde bulunan herhangi bir tesis açık ise ‘1’, kapalı ise ‘0’ değerini almaktadır. Müşteriye hizmet ulaştırma maliyeti matrisini gösteren C matrisindeki m sembolü, I kümesinde bulunan açık ya da kapalı tesisi temsil etmektedir. Tesisin açık olması durumunda $m = 1$, değilse $m = 0$ değerini alacaktır.

Bu çalışmada KTYP'nin çözümü için OR-Library (Beasley, 1990) kütüphanesinden alınmış olan 15 farklı problemten oluşan bir veri seti kullanılmıştır. Deneylerde kullanılan veri seti ile ilgili açıklamalar Çizelge 3.1'de verilmiştir. Çizelge 3.1'in ilk sütunu ilgili problemin adını, ikinci sütunu problemin boyutunu, üçüncü sütunu probleminin optimum maliyetini ve dördüncü sütunu problemin tipi hakkındaki bilgileri vermektedir. Cap71-74 arasındaki problemler küçük boyutlu problemlerdir ve bu problemler için tesis sayısı 16, müşteri sayısı ise 50'dir. Cap101-104 arasındaki problemler orta boyutlu problemlerdir ve bu problemler için tesis sayısı 25, müşteri sayısı ise 50'dir. Cap131-134 arasındaki problemler büyük boyutlu problemlerdir ve bu problemler için tesis sayısı 50, müşteri sayısı ise 50'dir. CapA, CapB ve CapC problemleri ise çok büyük boyutlu problemler sınıfına girmektedir ve bu problemler için tesis sayısı 100, müşteri sayısı ise 1000'dir.

Çizelge 3.1. KTYP'nin çözümü için deneylerde kullanılan veri seti

Problemin adı	Problemin boyutu	Optimum maliyet	Problemin tipi
Cap71	16 × 50	932615.750	Küçük boyutlu
Cap72	16 × 50	97779.400	
Cap73	16 × 50	1010641.450	
Cap74	16 × 50	1034976.975	
Cap101	25 × 50	796648.438	Orta boyutlu
Cap102	25 × 50	854704.200	
Cap103	25 × 50	893782.113	
Cap104	25 × 50	928941.750	
Cap131	50 × 50	793439.563	Büyük boyutlu
Cap132	50 × 50	851495.325	
Cap133	50 × 50	893076.713	
Cap134	50 × 50	928941.750	
CapA	100 × 1000	17156454.478	Çok büyük boyutlu
CapB	100 × 1000	12979071.58	
CapC	100 × 1000	11505594.33	

3.4.2. Sürekli nümerik fonksiyonlar: CEC 2015 veri seti

İkili optimizasyon problemlerinin performans analizi için sürekli nümerik problemler de kullanılabilir. Sürekli fonksiyonların ikili optimizasyon algoritmaları ile çözülebilmesi için her bir nümerik değer için bir ikili kodlama vektörü

kullanılmaktadır. Yani her bir nümerik değer, ikili şekilde temsil edilmektedir. Son yıllarda ikili optimizasyon problemlerinin çözümü için önerilen bir çok ikili optimizasyon algoritmasının performans karşılaştırmalarında CEC (IEEE Congress on Evolutionary Computation) veri setleri kullanılmaktadır (Banitalebi ve ark., 2016). Yapılan bu çalışmada da önerilen yaklaşımın performans analizi için Chen ve ark. (2014), literatüre kazandırmış oldukları CEC 2015 tek amaçlı kısıtlı hesaplamaya dayalı güçlü nümerik optimizasyon problemleri (Bound Constrained Single-Objective Computationally Expensive Numerical Optimization Problems) veri seti kullanılmıştır (Chen ve ark., 2014).

Çizelge 3.2. CEC 2015 nümerik kıyas fonksiyonları veri seti (Korkmaz, 2019)

Grup	F. No	Fonksiyon adı	Ana fonksiyonlar	Optimum
Tek Modlu Fonksiyonlar	f_1	Rotated Bent Cigar	Bent Cigar	100
	f_2	Rotated Discus	Discus	200
	f_3	Shifted and Rotated Weierstrass	Weierstrass	300
	f_4	Shifted and Rotated Schwefel's	Schwefel's	400
	f_5	Shifted and Rotated Katsuura	Katsuura	500
Kolay Çok Modlu Fonksiyonlar	f_6	Shifted and Rotated HappyCat	HappyCat	600
	f_7	Shifted and Rotated HGBat	HGBat	700
	f_8	Shifted and Rotated Expanded Griewank's plus Rosenbrock's	Griewank's Rosenbrock's	800
	f_9	Shifted and Rotated Expanded Scaffer's F6	Expanded Scaffer's F6	900
	f_{10}	Hibrit Fonksiyon 1 (N = 3)	Schwefel's Rastrigin's High Conditioned Elliptic Griewank's	1000
Hibrit Fonksiyonlar	f_{11}	Hibrit Fonksiyon 2 (N = 4)	Weierstrass Rosenbrock's Scaffer's F6 Katsuura HappyCat Griewank's	1100
	f_{12}	Hibrit Fonksiyon 3 (N = 5)	Rosenbrock's Schwefel's Ackley's Rosenbrock's High Conditioned Elliptic	1200
	f_{13}	Birleşik Fonksiyon 1 (N = 5)	Bent Cigar Discus High Conditioned Elliptic Schwefel's	1300
Birleşik Fonksiyonlar	f_{14}	Birleşik Fonksiyon 2 (N = 3)	Rastrigin's High Conditioned Elliptic HGBat	1400
	f_{15}	Birleşik Fonksiyon 3 (N = 5)	Rastrigin's Schwefel's Weierstrass High Conditioned Elliptic	1500

CEC 2015 veri seti 15 farklı nümerik fonksiyondan oluşmaktadır ve bu fonksiyonlar dört ana gruba ayrılmaktadır. Birinci grup; tek modlu fonksiyonlar (unimodal functions), ikinci grup; kolay çok modlu fonksiyonlar (simple multimodal functions), üçüncü grup; hibrit fonksiyonlar (hybrid functions) ve son olarak dördüncü grup; birleşik fonksiyonlar (composition functions) olarak ayrılmaktadır. Çizelge 3.2’de verilmiş olan CEC 2015 problemler için karar değişkenleri $[-100, 100]$ aralığındaki değerleri alabilmektedir. Çizelge 3.2’in ilk sütunu ilgili problemin dâhil olduğu problemin sınıfını, ikinci sütun fonksiyon numarasını, üçüncü sütun probleminin adını, dördüncü sütun oluşturulan yeni fonksiyonun bağlı olduğu ana fonksiyonları, son sütun ise ilgili fonksiyon için optimum değeri göstermektedir. Çizelge 3.2’de verilen problemler için daha fazla ayrıntıya Chen ve ark. (2014) çalışmasından bakılabilir.

Sürekli nümerik fonksiyonlarda karar değişkenleri sürekli değerlerden oluştuğundan dolayı, bu problemlerin ikili optimizasyon algoritmaları ile çözülebilmesi için her bir nümerik değerin bir ikili kodlama vektörüne dönüştürülmesi gerekmektedir. Yani her bir nümerik değer, ikili şekilde temsil edilmelidir. Herhangi bir nümerik bir değere karşılık gelen ikili vektörün, nümerik değere dönüştürülmesi işlemi Denklem 3.6’da verilmiştir.

$$NV^j = AltS^j + \frac{ÜstS^j - AltS^j * OndalıkD^j}{2^m - 1} \quad (3.6)$$

Denklemde 3.6’ da verilen, NV^j ifadesi nümerik vektörün (NV) j . boyutundaki sürekli değeri, $AltS^j$ ifadesi j . boyutun alabileceği alt sınır değerini, $ÜstS^j$ ifadesi j . boyutun alabileceği üst sınır değerini, $OndalıkD^j$ j . boyut için ikili sayının ondalık sayı sistemindeki tamsayı değerini ve $2^m - 1$ ifadesi, m boyutlu ikili vektörün bit uzunluğuna bağlı olarak ikili vektörün alabileceği maksimum tam sayı değerini göstermektedir.

3.4.3. Rüzgâr türbini yerleştirme problemi

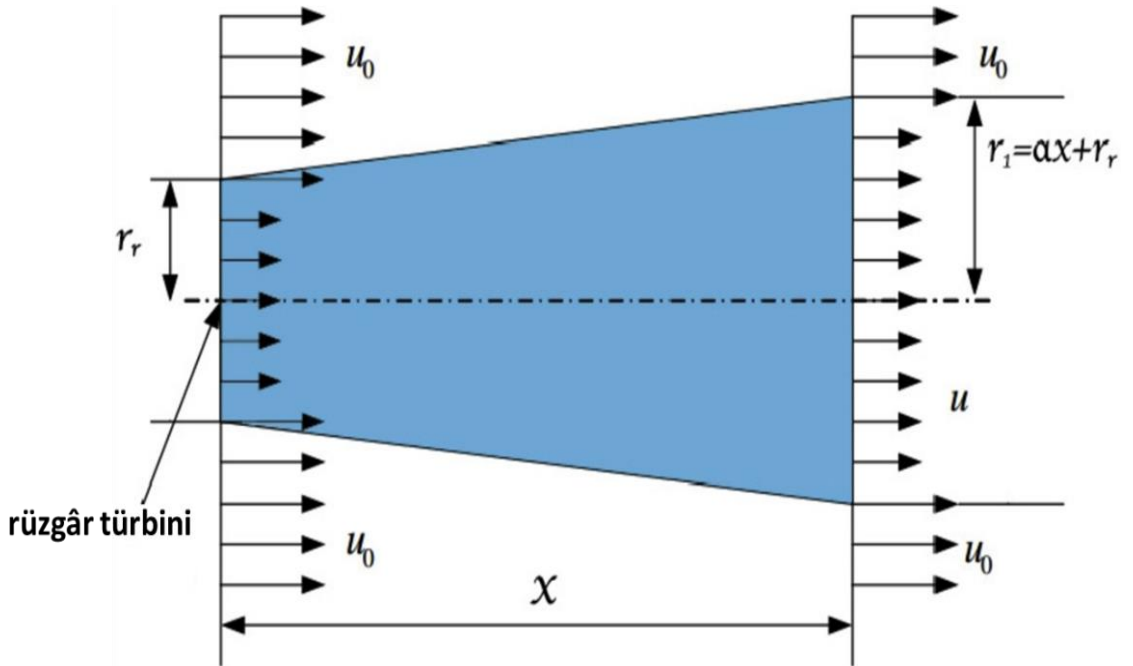
Rüzgâr türbinleri, rüzgârdan elde edilen enerjiyi elektrik enerjisine dönüştürmektedirler. Bu dönüştürme işlemi, kanatların üzerindeki rüzgârın aerodinamik kuvvetine bağlı olarak dönen bir direğe bağlanan kanatlar tarafından gerçekleştirilir (Mittal, 2010). Daha fazla enerji elde etmek ve bunu sağlarken bakım, onarım ve kurulum maliyeti gibi maliyeleri azaltmak amacıyla birçok rüzgâr türbininin, bir rüzgâr çiftliğinde

birlikte yerleştirilmesi gerekmektedir. Böylece her rüzgâr türbini için elektrik şebekesi ile ilgili maliyetler, yönetim maliyetleri gibi sabit maliyetler azaltılmış olmaktadır (Crespo ve Herna, 1996). Eğer bir rüzgâr çiftliğinde bulunan türbin sayısı artırılırsa türbinler arası uyandırma etkisinden dolayı türbin yerleştirme problemleri ortaya çıkacaktır ve türbinlerin kötü konumlandırılmasından kaynaklı rüzgâr türbinlerinden beklenen verim alınamayacaktır (Crespo ve Herna, 1996; Beşkirli ve ark., 2018). Sonuç olarak rüzgâr türbinlerinin bir rüzgâr çiftliğine uygun bir şekilde yerleştirilmesi, uyandırma etkisinin olumsuz özelliklerini azaltabilir ve rüzgâr türbinlerinden elde edilen verimi yükseltebilir (Gao ve ark., 2015).

Rüzgâr türbini yerleştirme probleminin çözümü için kullanılan ilk metasezgisel algoritma, Mosetti ve ark. (1994) tarafından önerilen GA tabanlı bir yaklaşımdır. Yaptıkları çalışmada türbinlerin yerleştirilmesi için $2\text{ km} \times 2\text{ km}$ bir alan kullanmışlardır. Türbinler arası uyandırma etkisinin en aza indirilebilmesi için iki türbin arasındaki mesafenin 5 türbin çapına (5 diameter - 5D) eşit veya daha yüksek olması gerekmektedir. Bir türbin çapı 40 metreye eşit olduğundan dolayı uygun bir yerleştirme için iki komşu türbin arası mesafe en az 200 metre olmalıdır. Eğer $2\text{ km} \times 2\text{ km}$ alan, 10×10 hücreli bir ızgaraya (grid) bölünürse, iki komşu hücre merkezi arasındaki mesafe 200 m'ye eşit olacağı için rüzgâr türbini yerleştirme işleminde kullanılmak üzere $n = 100$ olası konum oluşacaktır. Bu durumda 10×10 hücreli bir ızgara alanda toplamda 2^{100} farklı çözüm seçeneği olacaktır. Fakat eğer test alanı, Beşkirli ve ark. (2018) yapmış oldukları çalışmada olduğu gibi 20×20 hücreli bir ızgaraya bölünürse, $n = 400$ olacağı için muhtemel çözümlerin sayısı 2^{400} olacaktır (Beşkirli ve ark., 2018). Bu veriler ışığında anlaşılmaktadır ki türbin yerleştirme işlemi için olası konumların sayısının artmasına bağlı olarak problemin çözüm karmaşıklığı da katlanarak artacaktır (Parada ve ark., 2017) ve bundan dolayı rüzgâr çiftliği yerleşim optimizasyonu problemi bir NP-Zor sınıfı problemi olarak kabul edilmektedir (Rehman ve ark., 2016). Bu nedenle bu tip problemlerin çözümü için kesin algoritmaların makul bir sürede maksimum performansı göstermesi zordur ve üstel zamanda karmaşıklığı artan rüzgâr türbini yerleştirme problemi gibi zor problemlerin çözümü için etkili metodolojilere ihtiyaç vardır.

3.4.3.1. Uyandırma etkisi modelinin matematiksel formülü (Wake model)

Rüzgâr türbinlerinin itme katsayısı eğrisi (thrust coefficient curve), rotor çapı (rotor diameter), devreye girme hızı (cut-in speed), nominal hız (nominal speed), kesme hızı (cut-out speed), nominal güç (nominal power), güç eğrisi (power curve) ve gövde yüksekliği (hub height) gibi özellikler rüzgâr türbinlerinden elde edilecek verimi etkilemektedir. Rüzgârdan enerji elde etme sürecinde, rüzgâr bir türbinden çıkarken bir uyandırma etkisi oluşturmaktadır. Böylelikle rüzgâr, türbinin içinden geçtikten sonra türbülans meydana gelmekte ve bu türbülans komşu türbinleri doğrudan etkilemekte ve bu türbinlerden elde edilen verim azalmaktadır (Pookpunt ve Ongsakul, 2013). Jensen (1983) tarafından önerilen uyandırma etkisi modeli (wake model) (Jensen, 1983) basit, pratik ve optimizasyon algoritmalarına kolay uyarlanabilme özelliği sayesinde; (Mosetti ve ark., 1994), (Grady ve ark., 2005), (González ve ark., 2010), (Pookpunt ve Ongsakul, 2013), (Beşkirli ve ark., 2018) ve (Beşkirli ve ark., 2019) yapmış oldukları çalışmalarda kullanılmıştır. Bu nedenle, yapılan bu çalışmada da uyandırma etkisi modeli olarak Jensen uyandırma etkisi modeli kullanılmıştır. Jensen modeline göre, rüzgâr yukarı yöndeki rüzgâr türbininden geçtiğinde bir türbülans meydana gelmektedir ve bu türbülans, mevcut rüzgâr hızını düşürdüğü için aşağı yöndeki rüzgâr türbinini doğrudan etkilemektedir. Jensen uyandırma etkisi modeli Şekil 3.13’de verilmiştir.



Şekil 3.13. Jensen uyandırma etkisi modeli (Beşkirli ve ark., 2018)

Şekil 3.13’de görüldüğü gibi rüzgâr yukarı yöndeki türbinden geçtikten sonra rüzgâr hızında bir salınım gerçekleşmekte ve mevcut rüzgâr hızı azalmaktadır. Rüzgârın tekrar eski hızına ulaşabilmesi için en az x mesafesi kadar bir mesafeye ihtiyaç duyulmaktadır. Eğer aşağı yöndeki türbin belirli x mesafesi kadar bir mesafeden sonra yerleştirilirse, bu türbinden de maksimum verim alınabilecektir. Jensen uyandırma etkisi modeli Denklem 3.7’de verilmiştir.

$$u = u_0 = \left[1 - 2a \left(1 + \frac{\alpha x}{r_r} \right)^{-2} \right] \quad (3.7)$$

Denklem 3.7’de verilen, x yukarı yöndeki ve aşağı yöndeki rüzgâr türbinleri arasındaki mesafeyi, a eksensel indüksiyon faktörünü (axial induction factor), u aşağı yöndeki rüzgâr hızını (downstream wind speed), r_r rotor yarıçapını (rotor radius), u_0 serbest rüzgâr hızını ve α sürüklenme sabitini (entrainment constant) göstermektedir. Yukarı yönlü rotor yarıçapı ve eksensel indüksiyon faktörü, aşağı yönlü rotor yarıçapını (r_1) etkilemektedir. İtme katsayısı (thrust coefficient - C_T), bir rüzgâr türbininin rotoru üzerinde rüzgâr tarafından uygulanan itme veya eksensel indüksiyon faktörünün ve rotor düzleminden doğrudan karşılaşılan rüzgârın etkisinin oranını göstermektedir (Pookpunt ve Ongsakul, 2013). Aşağı yönlü rotor yarıçapı ve rüzgâr türbininin itme katsayısı aşağıdaki gibi hesaplanmaktadır:

$$r_1 = r_r \sqrt{\frac{1-a}{1-2a}}, \quad C_T = 4a(1-a) \quad (3.8)$$

Denklem 3.8’de verilen, α sürüklenme sabiti, uyandırma etkisine bağlı olarak büyümektedir. Denklem 3.9’da verilen z rüzgâr türbininin göbek yüksekliğini ve z_0 ise rüzgâr türbininin yerleştirildiği yüzey pürüzlülüğünü göstermektedir.

$$\alpha = \frac{0.5}{\ln(z/z_0)} \quad (3.9)$$

Uyandırma etkisinin kinetik enerji açığı, tüm türbinlerin enerji açıklarının toplamı olarak düşünülürse, N adet rüzgâr türbini için aşağı yönünde ortaya çıkan hız Denklem 3.10 ile hesaplanabilir.

$$\left(1 - \frac{u}{u_0}\right)^2 = \sum_{i=1}^N \left(1 - \frac{u_i}{u_0}\right)^2 \quad (3.10)$$

Bir rüzgâr çiftliğine yerleştirilmiş olan tüm türbinlerinden elde edilen toplam enerji (P_{total}) üretimi Denklem 3.11'e göre hesaplanmaktadır.

$$P_{total} = \sum_{i=1}^N 0.3 \times u^3 \quad (3.11)$$

Rüzgâr türbini yerleştirme problemi için toplam maliyet ($Cost$) Denklem 3.12'de verilmiştir. Denklem 3.12'ye göre, bir rüzgâr türbininin sabit kurulum maliyetinin 1 olduğu ve bir rüzgâr çiftliğine birden fazla rüzgâr türbini kurulması halinde azami maliyetin $1/3$ oranında azalacağı varsayılmaktadır. Maliyetin geri kalanı sabit $2/3$ oranında belirlenmektedir (Mosetti ve ark., 1994).

$$Cost = N \left[\frac{2}{3} + \frac{1}{3} e^{-0.00174N^2} \right] \quad (3.12)$$

Rüzgâr türbini probleminde kullanılan amaç fonksiyonu Denklem 3.13'te verilmiştir. Denklem 3.13'te amaç fonksiyonu, bir rüzgâr çiftliğinden elde edilen toplam enerji (P_{total}) ve Denklem 3.12'de verilen maliyet fonksiyonuna bağlı olarak hesaplanmaktadır. Rüzgâr türbinlerinin toplam maliyetin bu türbinlerden üretilen toplam enerjiye bölünmesi ile amaç fonksiyonu hesaplanmaktadır. Denklem 3.13'ten de anlaşıldığı üzere en az maliyet sağlayacak bir türbin yerleştirme işlemi ile maksimum enerji elde amacı vardır.

$$Amaç \text{ fonksiyonu} = \frac{Cost}{P_{total}} \quad (3.13)$$

Bir rüzgâr çiftliğine yerleştirilmiş olan türbinlerden maksimum verim alabilmek için rüzgâr türbinlerinin optimum şekilde konumlandırılmış olması gerekir. *Verimlilik*, Moorthy ve Deshmukh (2015) çalışmalarında kullanmış oldukları Denklem 3.14'te göre hesaplanmaktadır (Moorthy ve Deshmukh, 2015).

$$Verimlilik = \frac{P_{total}}{N \times 0.3 \times u_0^3} \quad (3.14)$$

Bu çalışmada rüzgâr türbini yerleştirme probleminin çözümü için Mosetti ve ark. (1994), yapmış oldukları çalışmada kullanmış oldukları $2 \text{ km} \times 2 \text{ km}$ test alanı kullanılmıştır. Test alanı 10×10 (100 muhtemel nokta) ve 20×20 (400 muhtemel nokta) olmak üzere iki farklı ızgara alanına bölünmüştür. Elde edilen sonuçlar literatürde bilinen iyi algoritmaların sonuçları ile karşılaştırılmıştır.

3.4.4. Gezgin satıcı problemi (GSP)

GSP: Minimum toplam maliyetle n adet şehri ziyaret etmek isteyen bir satış görevlisinin her şehri sadece bir kez ziyaret edebileceği ve en sonunda başlangıç noktasına dönebileceği en kısa Hamilton döngüsünün bulunması olarak tanımlanabilir (Zhong ve ark., 2007). Hamilton döngüsünde; başlangıç düğümünden (şehir) başlayarak her düğüme sadece bir kez uğramak şartı ile başlangıç düğümüne geri dönmeli ve bunu yaparken de geçilen yoldan (kavis) bir daha geçilmemelidir. n adet şehirli bir problem için, başlangıç şehri belli olmak şartı ile satış görevlisi için $(n - 1)!$ farklı yol söz konusudur. Bu durumda satış görevlisinin en kısa yol ile turunu tamamlanabilmesi için hangi yollardan geçmesi gerektiğinin tespiti oldukça zor olacaktır. GSP, $G = (V, E)$ olmak üzere ağırlıklı bir graf (weighted graph) olarak temsil edilmektedir. Burada G ilgili grafı, V düğümler (şehirler) kümesini, E kavisler (yollar) kümesini göstermektedir. $t \neq z$ olmak üzere, iki şehir arasındaki mesafe (d_{tz}) Öklid (Euclidean) mesafesine göre hesaplanmaktadır ve d_{tz} genellikle Denklem 3.15'teki gibi hesaplanmaktadır (Hatamlou, 2018):

$$d_{tz} = \sqrt{(x_t - x_z)^2 + (y_t - y_z)^2} \quad (3.15)$$

Çınar (2019) çalışmasında GSP'nin amaç fonksiyonu aşağıdaki denklemlerde detaylandırılmıştır (Çınar, 2019).

$$\text{Minimize: } \sum_{t \neq z}^D d_{tz} b_{tz} \quad (3.16)$$

$$\sum_{z=1}^D b_{tz} = 1 \quad t = 1, 2, \dots, D \quad (3.17)$$

$$\sum_{t=1}^D b_{tz} = 1 \quad z = 1, 2, \dots, D \quad (3.18)$$

$$\sum_{t, z \in S}^D b_{tz} \leq |S| - 1 \quad S \subset V, \quad 2 \leq S \leq D - 2 \quad (3.19)$$

$$b_{tz} \in \{0, 1\}, t, z = 1, 2, \dots, D, t \neq z \quad (3.20)$$

GSP'de amaç en kısa tur uzunluğuna sahip şehir sıralamasını elde etmektir. Bu nedenle GSP bir minimizasyon problemidir. Denklem 3.16'da verilen, D gezilmesi gereken toplam şehir sayısını göstermektedir. t ve z gezilen ya da gezilecek farklı iki şehri göstermektedir. b_{tz} matrisi, gezgin satıcının ilgili şehirleri gezip gezmediğine ait bilgiyi tutan matristir. d_{tz} matrisi ise gezilecek iki şehir arasındaki maliyeti gösteren mesafe matrisidir. Denklem 3.17 ve Denklem 3.18'de verilen kısıtlar sayesinde gezgin satıcının her şehre sadece bir kez uğrayabilmesine izin verilmektedir. Denklem 3.19'da verilen kısıtlar ise alt turların oluşmasını engelleyerek, gezgin satıcının ayrıldığı şehre ikinci kez uğramasını engellemektedir. Denklem 3.20'de ise t ve z şehirleri birbirinden farklı olmak şartı ile gezilecek şehirler için ikili kısıt şartı verilmiştir.

Bu çalışmada GSP'nin çözümü için TSPLIB kütüphanesinden alınmış 14 farklı simetrik GSP kullanılmıştır (Reinelt, 1991). TSPLIB kütüphanesinde bulunan GSP'lerin optimum değerleri bilinmektedir. Deneylerde kullanılan veri seti ile ilgili açıklamalar Çizelge 3.3'te verilmiştir. Veri setindeki bu problemlerin etiketinde genellikle problemdeki şehir sayısı kullanılmaktadır. Örneğin, Çizelge 3.3'te verilmiş olan *BERLIN52* problemi için gezgin satıcının uğraması gereken şehir sayısı 52'dir. Çizelge 3.3'ün ilk sütunu ilgili problemin adı, ikinci sütunu problemin boyutu, üçüncü sütunu problemin optimum maliyeti (en kısa yol uzunluğu) hakkındaki bilgileri vermektedir.

Çizelge 3.3. GSP'nin çözümü için deneylerde kullanılan veri seti

Problemin adı	Problemin boyutu	En kısa yol uzunluğu
EIL51	51	428.87
EIL76	76	545.38
EIL101	101	642.31
KROA100	100	21282 (21285.44)
KROB100	100	22141
KROC100	100	20749
KROD100	100	21294
KROE100	100	22068
OLIVER30	30	423.74
BERLIN52	52	7542 (7544.37)
TSP225	225	3859
ST70	70	677.11
PR76	76	108159.44
CH150	150	6532.10

Önerilen algoritmanın başarısını göstermek ve doğrulamak amacıyla deneyler için hem küçük boyutlu hem de çok büyük boyutlu problemler seçilmiştir. Örneğin, Çizelge 3.3'te verilen problemler dikkate alındığında en az şehirden oluşan problem *OLIVER30* problemidir ve bu problem için gezgin satıcının uğraması gereken şehir sayısı 30'dur. En fazla şehirden oluşan problem ise *TSP225* problemidir ve bu problem için gezgin satıcının uğraması gereken şehir sayısı 225'tir. Böylece önerilen algoritmanın başarı performansı daha iyi analiz edilebilir.

4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA

Çalışmanın bu bölümünde, önerilen algoritmalar ayrık optimizasyon problemleri üzerine uygulanmıştır. Daha sonra elde edilen deneysel sonuçlar analiz edilmiş ve literatürdeki başarılı algoritmaların sonuçları ile karşılaştırılmıştır. Tez çalışması kapsamında önerilen algoritmalar MATLAB ortamında kodlanmıştır. Deneylerde kullanılan kişisel bilgisayarın işlemcisi Intel® Core™ i7-6700HQ 2.60GHz, Ram belleği 24GB ve işletim sistemi Windows 10 Pro 64-bit'tir.

4.1. Kapasitesiz Tesis Yerleştirme Probleminin Çözümü için Deneysel Sonuçlar

Önerilen algoritmanın performansını analiz etmek ve doğrulamak için deneylerde 15 farklı KTY problemi kullanılmıştır. Önerilen algoritmanın etkinliğini göstermek amacıyla, önerilen algoritma karşılaştırılan diğer algoritmalar ile aynı şartlarda çalıştırılmıştır. Önerilen algoritma ve diğer algoritmalar birbirinden bağımsız rastgele 30 kez çalıştırılmıştır. Elde edilen sonuçlar, 30 çalıştırma (run) sonucunun ortalaması, standart sapması (Std. Sap.), *gap* ve isabet sayısı (*hit*) kriterlerine göre rapor edilmiştir. *Gap*, sonuçların ortalamasının beklenen sonuca olan uzaklığını ifade etmektedir ve *gap* değeri sıfıra yaklaştıkça algoritmanın başarılı olduğu sonucuna varılmaktadır. *Hit*, isabet sayısını ifade eder ve ilgili problem için önerilen algoritmanın optimum çözüme kaç kez ulaştığını gösterir. Önerilen algoritma için parametre analizi gerçekleştirildikten sonra, önerilen algoritma diğer algoritmaların deneysel sonuçları ile karşılaştırılmıştır.

4.1.1. KTYP'nin çözümü için önerilen algoritmanın parametre analizi

Önerilen algoritmanın kontrol parametrelerinin belirlenmesi amacıyla, öncelikle önerilen algoritmanın parametreleri analiz edilmiştir. Parametre analizi için popülasyon boyutu (N) 10'dan 100'e kadar seçilmektedir, ve bir denemede güncellenecek boyutların sayısı (P parametresi) 0.1'den 1'e kadar seçilmektedir. P parametresinin 1 olarak seçilmesi durumunda aday çözümün konum güncellemesi aşamasında, temel Jaya algoritmasında olduğu gibi tüm boyutlar tek seferde güncellenmektedir. P parametresinin 0.1 olarak seçilmesi durumunda ise aday çözümün toplam boyut sayısının %10'u kadar boyutun güncelleneceği anlamına gelmektedir. Yerel arama algoritması olan LSM algoritmasının kontrol parametresi olan P_{local} parametresi, Kashan ve ark. (2011)

çalışmalarında kullandıkları gibi $P_{local} = 0.01$ olarak belirlenmiştir. Sonlandırma koşulu ise maksimum fonksiyon değerlendirme sayısıdır (MaxFEs) ve tüm analizlerde, karşılaştırılan algoritmaların kullandıkları sonlandırma koşulu olan $8E + 5$ olarak belirlenmiştir. Önerilen algoritmanın parametre analizi aşamasında elde edilen deneysel sonuçlar Çizelge 4.1’de verilmiştir. Çizelge 4.1’de verilen sonuçlar her bir problem için toplam çalıştırma sayısına bağlı olarak optimum sonuca ulaşma başarı oranını göstermektedir. Deneylerde kullanılan KTY problemlerinin sayısı 15’tir ve her bir problem için algoritma rastgele 30 kez çalıştırılmıştır. Sonuç olarak algoritma her parametre değeri için toplamda 450 kez çalıştırılmıştır. Her parametre değeri için 450 çalıştırma sonunda, optimum sonuca ulaşma sayısına bağlı olarak algoritmanın başarıları belirlenmektedir. Başarı oranı, Denklem 4.1’e göre hesaplanmaktadır.

$$Başarı_oranı = (TBS)/(PS \times \mathcal{CS}) \quad (4.1)$$

Denklem 4.1’de verilen, TBS parametresi 450 çalışmanın içindeki toplam başarılı isabet sayısı (*hit*) sayısını, PS parametresi deneylerde kullanılan problem sayısını ve $\mathcal{CS}$ toplam çalıştırma (*run*) sayısını ifade etmektedir. Deneylerde kullanılan $PS = 15$ ’tir, $\mathcal{CS}$ ise her bir problem için 30 çalıştırmadır.

Çizelge 4.1 incelendiğinde P değeri 1’e yaklaştıkça algoritmanın performansı genellikle azalmaktadır. Temel Jaya algoritmasında aday çözümün her boyutunda güncelleme işlemi yapılmaktadır. Ancak önerilen algoritmaya eklenen olasılık parametresi olan P parametresi sayesinde her boyutta güncelleme işleminin yapılmasının algoritmanın performansını olumsuz yönde etkilediği gözlemlenmiştir. Önerilen algoritma, en başarısız sonuçlara popülasyon boyutu 10 ve P parametresi 1 olarak seçildiğinde ulaşmıştır. Aynı şekilde popülasyon boyutu 100 olarak seçildiğinde de algoritmanın performansı düşmektedir. Önerilen algoritma genellikle, P değeri 0.4~0.6 aralığında ve popülasyon boyutu da 40~70 aralığında seçildiğinde başarılı sonuçlar elde etmiştir. Önerilen algoritma, en başarılı sonuçlara $P = 0.5$ ve popülasyon boyutu 50 olarak seçildiğinde ulaşmıştır. Ancak literatürde önerilen diğer algoritmalar, çalışmalarında popülasyon boyutunu 40 olarak belirlemişlerdir. Çizelge 4.1 incelendiğinde, önerilen algoritmanın popülasyon boyutu 40 olarak seçildiğinde de çok başarılı sonuçlar elde ettiği görülmektedir. Algoritmaların karşılaştırılması sırasında adil bir karşılaştırma yapılması amacıyla, deneylerde popülasyon boyutu 40, $P = 0.5$ ve

sonlandırma kriteri ise $8E + 5$ olarak seçilmiştir. Durdurma kriterinin popülasyon boyutuna bölünmesi ile iterasyon sayısı hesaplanmaktadır ve bu durumda iterasyon sayısı 2000 olmaktadır.

Çizelge 4.1. Önerilen algoritmanın kontrol parametrelerinin başarı performansına etkileri

P	Popülasyon boyutu (N)									
	10	20	30	40	50	60	70	80	90	100
0.10	0.8378	0.9089	0.9156	0.9111	0.9067	0.9000	0.9044	0.9222	0.9044	0.9089
0.15	0.8444	0.9311	0.9267	0.9133	0.9133	0.9222	0.8978	0.9178	0.9022	0.9089
0.20	0.8156	0.9356	0.9400	0.9200	0.9267	0.9022	0.9111	0.9000	0.9000	0.9067
0.25	0.8200	0.9200	0.9400	0.9467	0.9267	0.9244	0.9133	0.9067	0.8933	0.8933
0.30	0.7956	0.9044	0.9400	0.9511	0.9333	0.9244	0.9089	0.9044	0.8867	0.8844
0.35	0.7667	0.9111	0.9489	0.9578	0.9378	0.9267	0.9178	0.9111	0.8933	0.8689
0.40	0.7489	0.9000	0.9444	0.9622	0.9622	0.9311	0.9244	0.9044	0.8911	0.8511
0.45	0.7311	0.8756	0.9444	0.9556	0.9444	0.9333	0.9200	0.9133	0.8778	0.8556
0.50	0.7378	0.8756	0.9289	0.9533	0.9667	0.9533	0.9244	0.9133	0.8933	0.8511
0.55	0.7533	0.8578	0.9089	0.9311	0.9444	0.9444	0.9378	0.9222	0.9178	0.8822
0.60	0.6956	0.8467	0.8978	0.9400	0.9222	0.9489	0.9600	0.9289	0.9289	0.8933
0.65	0.6600	0.8244	0.8911	0.9311	0.9400	0.9378	0.9578	0.9578	0.9311	0.9178
0.70	0.6600	0.8133	0.8711	0.9133	0.9156	0.9356	0.9378	0.9578	0.9356	0.9311
0.75	0.6356	0.7644	0.8378	0.8822	0.9044	0.9089	0.9311	0.9356	0.9356	0.9289
0.80	0.5956	0.7578	0.8067	0.8667	0.9000	0.9111	0.9178	0.9222	0.9111	0.9356
0.85	0.5222	0.7400	0.7844	0.8378	0.8644	0.8644	0.9067	0.9067	0.9244	0.9267
0.90	0.4200	0.6711	0.7289	0.7578	0.8089	0.8556	0.8711	0.8756	0.8756	0.8978
0.95	0.1933	0.4889	0.6244	0.6867	0.7444	0.7844	0.7711	0.8267	0.8289	0.8467
1	0.0667	0.1044	0.1600	0.2156	0.2844	0.2933	0.3333	0.3467	0.3244	0.3622

4.1.2. KTYP'nin çözümü için algoritmaların performans analizi

Parametre analizinde elde edilen sonuçlar değerlendirildiğinde, JayaX-LSM algoritmasının 15 KTY probleminden 13'ü için optimum çözümlere ulaştığı ve geriye kalan 2 problem içinde elde ettiği performansın kabul edilebilir seviyede olduğu görülmektedir. Bu bölümde ise önerilen algoritmaların performans analizinin yapılması amacıyla önerilen algoritmalar, sürü zekâsı veya evrimsel hesaplama dayalı algoritmaların önde gelen üyeleriyle karşılaştırılmıştır.

Önerilen JayaX algoritmasının iki sürümü vardır: bunlar, yerel arama modülüne sahip olmayan JayaX ve yerel arama modülünü kullanan JayaX-LSM'dir. Bu iki algoritma, farklı genetik algoritma tabanlı yaklaşımlarla, ikili yapay alg algoritmalarıyla (BAAA-Tanh, BAAA-Sig, binAAA), ikili parçacık sürüsü optimizasyonu (BPSO), ikili yapay arı kolonisi algoritmalarıyla (binABC, DisABC ve ABCbin), ikili ağaç tohum algoritması (SimLogicTSA) ve ikili farksal gelişim algoritmaları (DisDE ve binDE) gibi yakın zamanda literatüre kazandırılmış algoritmalarla karşılaştırılmıştır. İlk deneysel karşılaştırma, farklı çaprazlama operatörleri (tek noktalı, iki noktalı ve uniform çaprazlama) kullanan GA tabanlı yaklaşımlarla, AAA'nın farklı ikili versiyonlarıyla ve BJA algoritmasıyla yapılmıştır. İlk deneysel karşılaştırmada kullanılan algoritmaların her biri için 30 bağımsız çalıştırma (run) yapılmıştır. Elde edilen sonuçlar için ortalama, standart sapma (Std. Sap.), *Hit* ve *Gap* değerleri kaydedilmiştir. *Hit* (isabet sayısı) 30 çalıştırmadaki başarı sayısını ve *Gap* ise 30 çalıştırmanın sonunda elde edilen ortalama maliyet ile optimum maliyet arasındaki uzaklığı göstermektedir ve aşağıdaki gibi hesaplanmaktadır:

$$Gap = \frac{f(ortalama) - f(optimum)}{f(optimum)} \times 100 \quad (4.2)$$

Denklem 4.2'de verilen, $f(optimum)$ değeri ilgili problem için optimum maliyeti, $f(ortalama)$ ise ilgili problem için 30 bağımsız çalıştırmanın sonunda elde edilen ortalama maliyeti göstermektedir.

Çizelge 4.2, 4.3, 4.4 ve 4.5'te verilmiş olan algoritmalar: tek noktalı (single point) çaprazlamaya dayalı GA (GA-SP), iki noktalı çaprazlamaya (two point) dayalı GA (GA-TP), uniform çaprazlamaya dayalı GA (GA-UP), BAAA-Tanh, BAAA-Sig ve binAAA algoritmalarına ait sonuçlar doğrudan Korkmaz ve ark. (2018)'nin yapmış oldukları çalışmadan alınmıştır (Korkmaz ve ark., 2018). Prakash ve ark. (2017)'nin önermiş oldukları BJA algoritması ise bu çalışma kapsamında kodlanmış olup, önerilen JayaX ve JayaX-LSM algoritmaları ile aynı şartlar altında çalıştırılmıştır. Cap71, Cap72, Cap73 ve Cap74 problemleri için karşılaştırılan algoritmaların elde ettiği sonuçlar Çizelge 4.2'de verilmiştir. Cap71-74 problemleri 16 tesis ve 50 müşteriden oluşan, nispeten kolay olarak nitelendirilen problemler kategorisine dahil edildiği için genellikle bütün algoritmalar 30 çalıştırmanın tamamında optimum sonuca ulaşmışlardır. Ancak, GA-SP, GA-TP, GA-UP ve BJA algoritmaları başarılı sonuçlara ulaşmasına rağmen

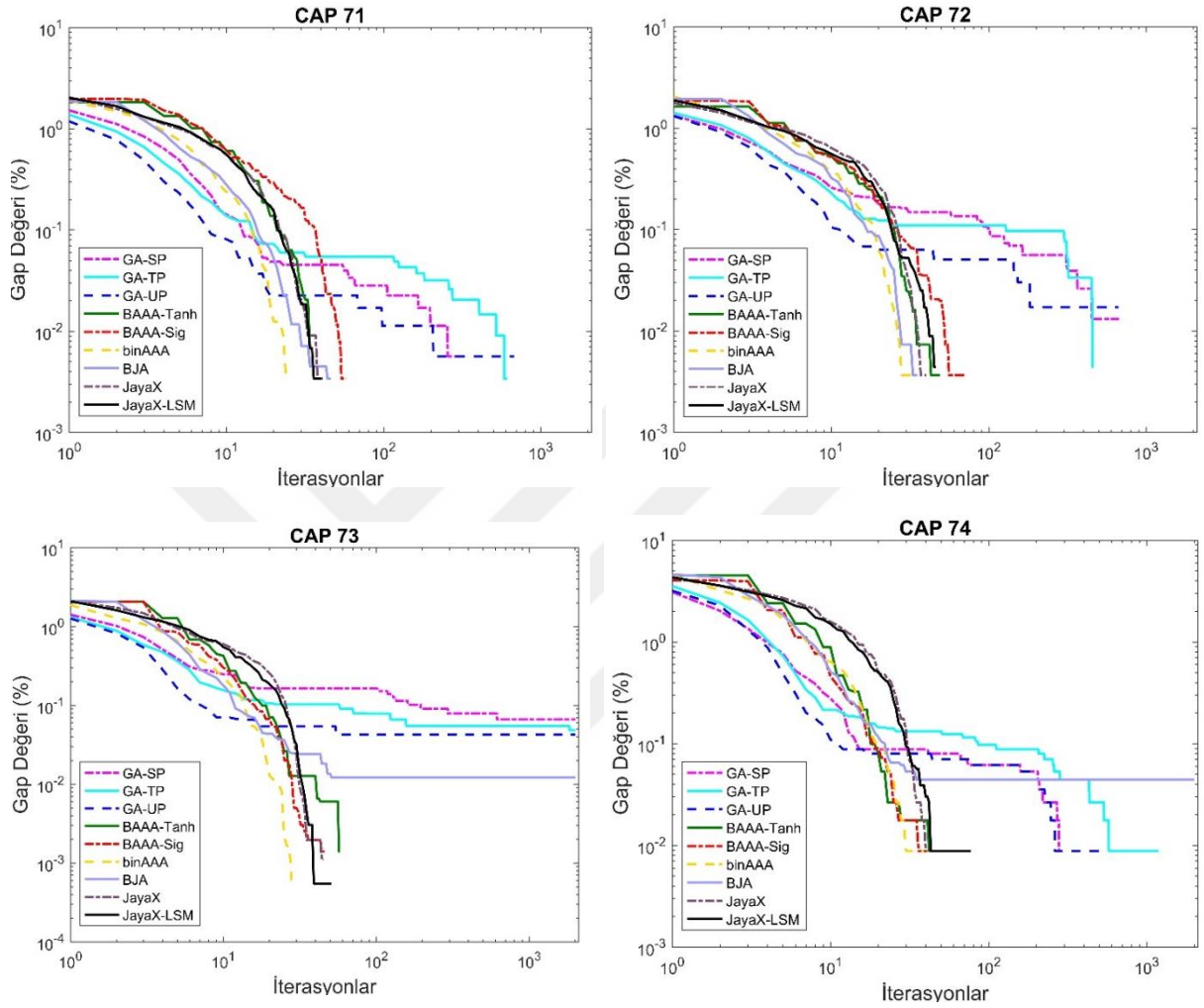
Cap71, Cap72, Cap73 ve Cap74 problemleri için 30 çalıştırmanın tamamında optimum sonuçlara ulaşamamıştır.

Çizelge 4.2. JayaX ve JayaX-LSM'nin Cap71, Cap72, Cap73 ve Cap74 problemleri için yakın zamanda önerilen algoritmalarla karşılaştırılması

Algoritma	Kriter	Cap71	Cap72	Cap73	Cap74
GA-SP	Ortalama	932615.750	977799.400	1011314.476	1034976.975
	Gap	0.00000	0.00000	0.06659	0.00000
	Hit	30	30	19	30
	Std. Sap.	0.000	0.000	899.650	0.000
GA-TP	Ortalama	932615.750	977799.400	1011130.923	1034976.975
	Gap	0.00000	0.00000	0.04843	0.00000
	Hit	30	30	22	30
	Std. Sap.	0.000	0.000	825.576	0.000
GA-UP	Ortalama	932615.750	977799.400	1011069.739	1034976.975
	Gap	0.00000	0.00000	0.04238	0.00000
	Hit	30	30	23	30
	Std. Sap.	0.000	0.000	789.612	0.000
BAAA-Tanh	Ortalama	932615.750	977799.400	1010641.450	1034976.975
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
BAAA-Sig	Ortalama	932615.750	977799.400	1010641.450	1034976.975
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
binAAA	Ortalama	932615.750	977799.400	1010641.450	1034976.975
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
BJA	Ortalama	932615.750	977799.400	1010763.818	1035433.658
	Gap	0.00000	0.00000	0.01211	0.04412
	Hit	30	30	28	25
	Std. Sap.	0.000	0.000	465.688	1038.632
JayaX	Ortalama	932615.750	977799.400	1010641.450	1034976.975
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
JayaX-LSM	Ortalama	932615.750	977799.400	1010641.450	1034976.975
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000

Cap71, Cap72, Cap73 ve Cap74 problemleri için algoritmaların iterasyonlar boyunca optimum sonuca yakınsama grafikleri Şekil 4.1'de verilmiştir. Şekil 4.1 incelendiğinde optimum sonuçlara en hızlı yakınsayan algoritma binAAA'dır. BAAA-Tanh, BAAA-Sig, binAAA, JayaX ve JayaX-LSM algoritmaları ortalama 100

iterasyonda her dört Cap problemi içinde optimum sonuçlara ulaşabilmışlerdir. GA-SP, GA-TP, GA-UP ve BJA algoritmaları ise diğer beş algoritmaya kıyasla optimum sonuçlara daha yavaş yakınsamıştır.



Şekil 4.1. Cap71, Cap72, Cap73 ve Cap74 problemleri için algoritmaların yakınsama grafikleri

Cap101, Cap102, Cap103 ve Cap104 problemleri için karşılaştırılan algoritmaların elde ettiği sonuçlar Çizgelge 4.3'te verilmiştir. Cap101-104 problemleri 25 tesis ve 50 müşteriden oluşan, orta boyutlu olarak nitelendirilen problemler kategorisine dahil edildiği halde, algoritmaların çoğu 30 çalıştırmanın bir kısmı için optimum sonuca ulaşmıştır. Çizgelge 4.3 incelendiğinde, BAAA-Sig, binAAA, JayaX ve JayaX-LSM algoritmaları Cap101-104 problemleri için 30 çalıştırmanın tamamında optimum sonuçlara ulaşmıştır. BAAA-Tanh algoritması ise Cap101 hariç diğer orta boyutlu problemler için optimum sonuçlara ulaşmıştır. Ancak, GA-SP, GA-TP, GA-UP

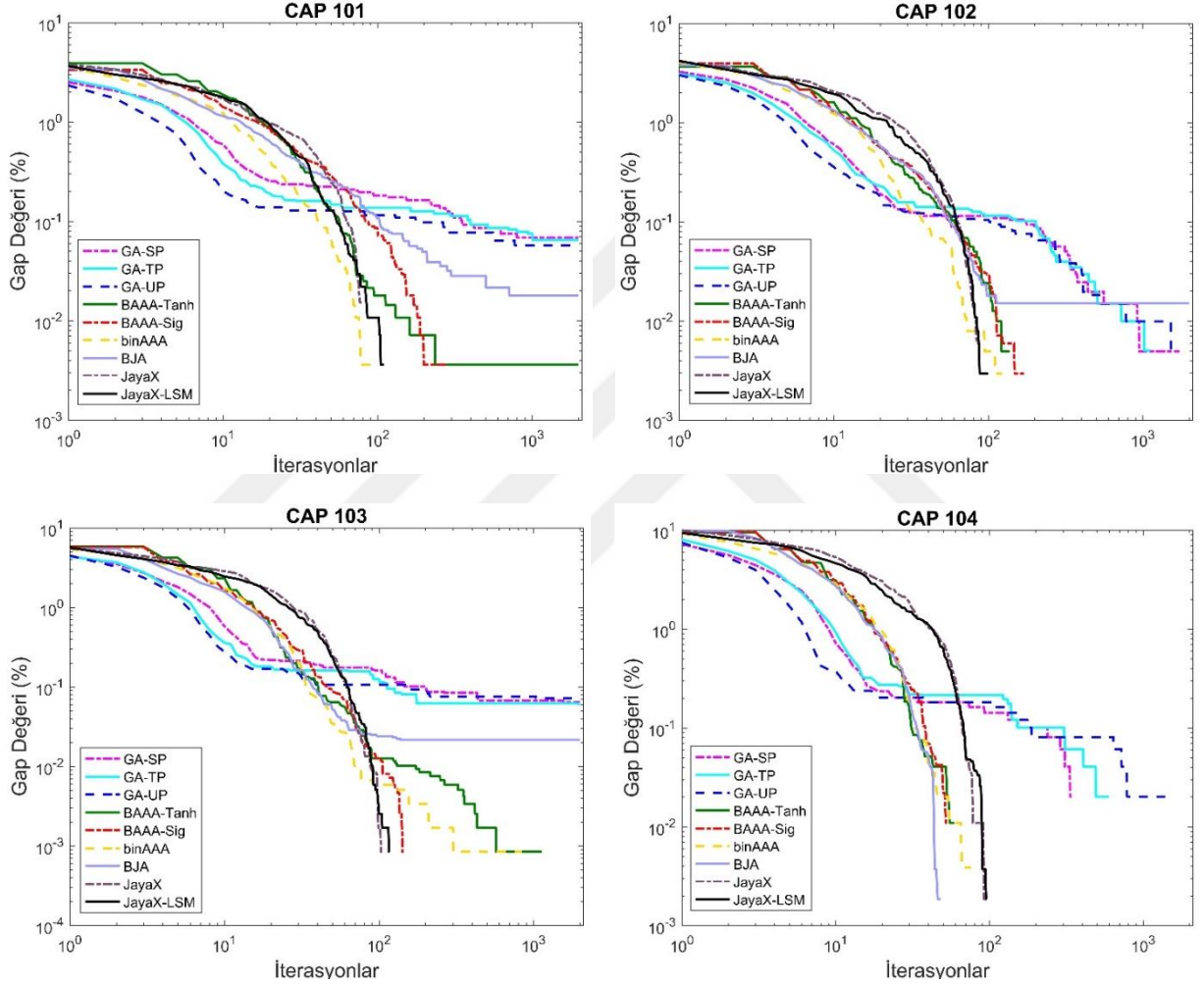
ve BJA algoritmalarının Cap101-104 problemleri için elde ettiği sonuçlar diğer algoritmalara nazaran zayıf kalmıştır.

Çizelge 4.3. JayaX ve JayaX-LSM'nin Cap101, Cap102, Cap103 ve Cap104 problemleri için yakın zamanda önerilen algoritmalarla karşılaştırılması

Algoritma	Kriter	Cap101	Cap102	Cap103	Cap104
GA-SP	Ortalama	797193.286	854704.200	894351.782	928941.750
	Gap	0.06839	0.00000	0.06374	0.00000
	Hit	11	30	6	30
	Std. Sap.	421.655	0.000	505.036	0.000
GA-TP	Ortalama	797164.610	854704.200	894329.179	928941.750
	Gap	0.06479	0.00000	0.06121	0.00000
	Hit	12	30	10	30
	Std. Sap.	428.658	0.000	540.160	0.000
GA-UP	Ortalama	797107.258	854704.200	894427.382	928941.750
	Gap	0.05759	0.00000	0.07220	0.00000
	Hit	14	30	9	30
	Std. Sap.	436.524	0.000	522.784	0.000
BAAA-Tanh	Ortalama	796677.114	854704.200	893782.113	928941.750
	Gap	0.00360	0.00000	0.00000	0.00000
	Hit	29	30	30	30
	Std. Sap.	157.066	0.000	0.000	0.000
BAAA-Sig	Ortalama	796648.438	854704.200	893782.113	928941.750
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
binAAA	Ortalama	796648.438	854704.200	893782.113	928941.750
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
BJA	Ortalama	796791.819	854833.133	893974.557	928941.750
	Gap	0.01800	0.01509	0.02153	0.00000
	Hit	25	27	22	30
	Std. Sap.	326.091	393.441	382.124	0.000
JayaX	Ortalama	796648.438	854704.200	893782.113	928941.750
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
JayaX-LSM	Ortalama	796648.438	854704.200	893782.113	928941.750
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000

Cap101, Cap102, Cap103 ve Cap104 problemleri için algoritmaların iterasyonlar boyunca optimum sonuca yakınsama grafikleri Şekil 4.2'de verilmiştir. Şekil 4.2 incelendiğinde Cap101 problemi için en hızlı yakınsamayı binAAA, Cap102 problemi için en hızlı yakınsamayı JayaX-LSM, Cap103 problemi için en hızlı yakınsamayı JayaX ve JayaX-LSM ve Cap104 problemi için ise en hızlı yakınsamayı BJA göstermiştir. Şekil 4.2 incelendiğinde, BAAA-Sig, binAAA, JayaX ve JayaX-LSM algoritmaları Cap101-104 problemleri için ortalama 100 iterasyonda optimum sonuçlara

ulaşılabilmişlerdir. BJA algoritmasının, Cap104 problemi için optimum sonuca 100 iterasyon civarında ulaşmasına rağmen diğer problemler için yakınsaması yavaş olmuştur. BAAA-Tanh, GA-SP, GA-TP ve GA-UP algoritmalarının ise bazı orta boyutlu Cap problemleri için optimum sonuçlara ulaşabilmesi için 1000 civarında iterasyona ihtiyacı olmuştur, hatta bu algoritmalar bazı Cap problemler için 2000 iterasyonda dahi optimum sonuçlara ulaşamamışlardır.



Şekil 4.2. Cap101, Cap102, Cap103 ve Cap104 problemleri için algoritmaların yakınsama grafikleri

Cap131, Cap132, Cap133 ve Cap134 problemleri için karşılaştırılan algoritmaların elde ettiği sonuçlar Çizgelge 4.4'te verilmiştir. Cap131-134 problemleri 50 tesis ve 50 müşteriden oluşan, büyük boyutlu olarak nitelendirilen problemler kategorisine dahil edildiği için, karşılaştırmalarda kullanılan algoritmaların çoğu genellikle 30 çalıştırmanın bir kısmı için optimum sonuçlara ulaşmıştır. Çizgelge 4.4 incelendiğinde, BAAA-Sig, JayaX ve JayaX-LSM algoritmaları

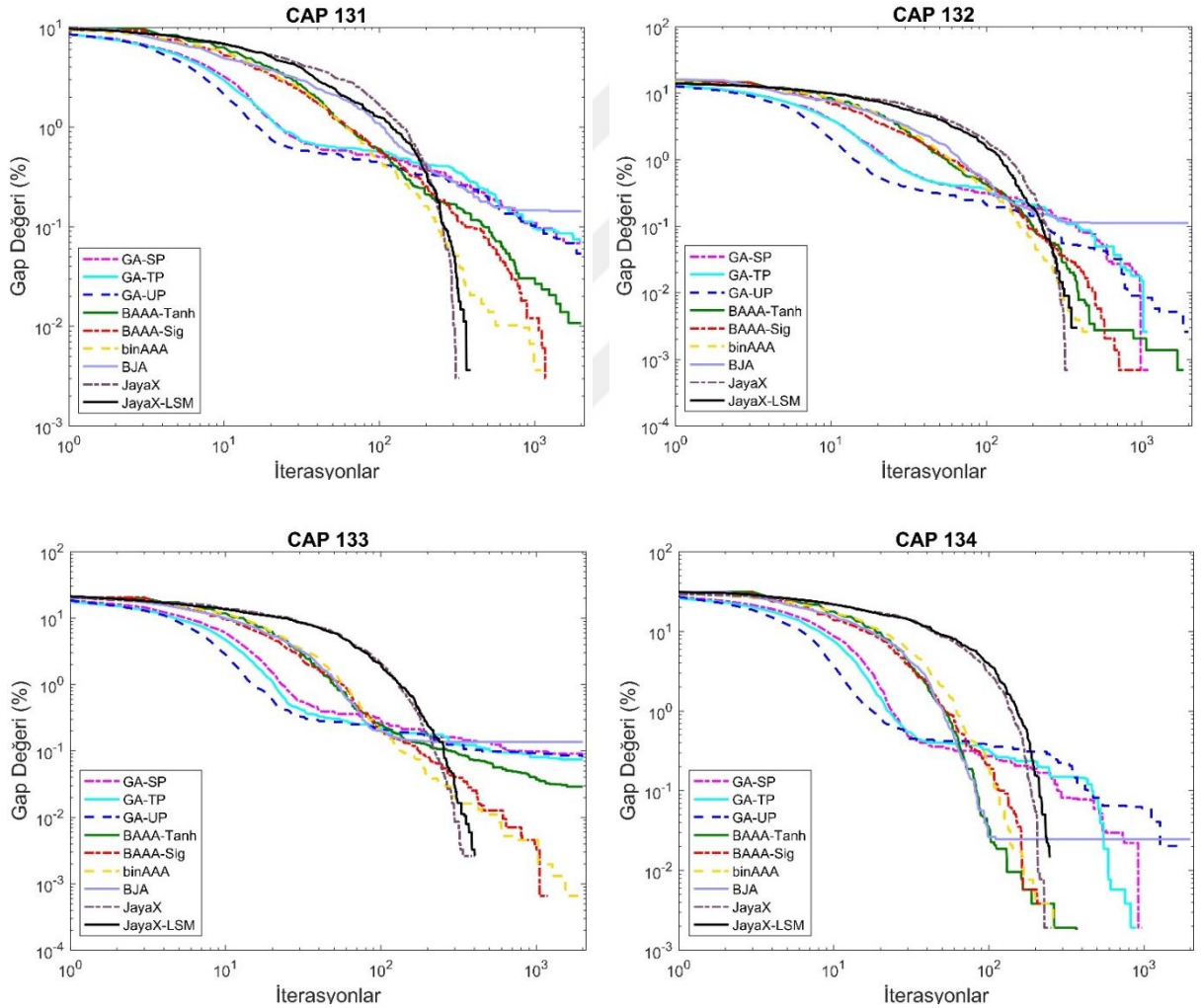
Cap131-134 problemleri için 30 çalıştırmanın tamamında optimum sonuçlara ulaşmıştır. binAAA algoritması ise Cap133 problemi hariç diğer büyük boyutlu problemler için optimum sonuçlara ulaşmıştır. BJA algoritması ise bu dört problem için 30 çalıştırmanın sadece bir kısmı için optimum sonuca ulaşmıştır. Bununla birlikte, BAAA-Tanh, GA-SP, GA-TP, GA-UP algoritmaları Cap131-134 problemleri için BJA algoritmasından daha iyi sonuçlar elde etmesine rağmen binAAA, BAAA-Sig, JayaX ve JayaX-LSM algoritmalarına nazaran zayıf kalmıştır.

Çizelge 4.4. JayaX ve JayaX-LSM'nin Cap131, Cap132, Cap133 ve Cap134 problemleri için yakın zamanda önerilen algoritmalarla karşılaştırılması

Algoritma	Kriter	Cap131	Cap132	Cap133	Cap134
GA-SP	Ortalama	793980.104	851495.325	893891.911	928941.750
	Gap	0.06813	0.00000	0.09128	0.00000
	Hit	16	30	10	30
	Std. Sap.	720.877	0.000	685.076	0.000
GA-TP	Ortalama	794012.905	851495.325	893740.954	928941.750
	Gap	0.07226	0.00000	0.07438	0.00000
	Hit	14	30	12	30
	Std. Sap.	690.560	0.000	655.920	0.000
GA-UP	Ortalama	793865.023	851517.200	893808.891	928941.750
	Gap	0.05362	0.00257	0.08198	0.00000
	Hit	15	29	9	30
	Std. Sap.	433.467	119.817	628.654	0.000
BAAA-Tanh	Ortalama	793525.591	851495.325	893333.515	928941.750
	Gap	0.01084	0.00000	0.02875	0.00000
	Hit	27	30	16	30
	Std. Sap.	262.498	0.000	324.451	0.000
BAAA-Sig	Ortalama	793439.563	851495.325	893076.713	928941.750
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
binAAA	Ortalama	793439.563	851495.325	893082.539	928941.750
	Gap	0.00000	0.00000	0.00065	0.00000
	Hit	30	30	29	30
	Std. Sap.	0.000	0.000	31.914	0.000
BJA	Ortalama	794573.409	852450.297	894293.350	929170.160
	Gap	0.14290	0.11215	0.13623	0.02459
	Hit	11	9	6	21
	Std. Sap.	1266.076	1159.249	1689.417	502.235
JayaX	Ortalama	793439.563	851495.325	893076.713	928941.750
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000
JayaX-LSM	Ortalama	793439.563	851495.325	893076.713	928941.750
	Gap	0.00000	0.00000	0.00000	0.00000
	Hit	30	30	30	30
	Std. Sap.	0.000	0.000	0.000	0.000

Cap131, Cap132, Cap133 ve Cap134 problemleri için algoritmaların iterasyonlar boyunca optimum sonuca yakınsama grafikleri Şekil 4.3'de verilmiştir. Şekil 4.3

incelendiğinde Cap131, Cap132 ve Cap133 problemleri için en hızlı yakınsamayı JayaX ve JayaX-LSM algoritmaları göstermiştir. Cap 134 problemi için ise GA-SP, GA-TP ve GA-UP ve BJA algoritmaları hariç diğer algoritmalar hızlı bir şekilde optimum sonuca yakınsamışlardır. Şekil 4.3 incelendiğinde, JayaX ve JayaX-LSM algoritmaları Cap131-134 problemleri için ortalama 500 iterasyon civarında optimum sonuçlara ulaşabilmışlardır. İkinci en hızlı yakınsamayı ise BAAA-Sig algoritması göstermiştir. Diğer algoritmaların ise Cap131-134 problemler için optimum sonuçlara yakınsaması daha yavaş olmuştur, hatta bu algoritmalar bazı problemler için 2000 iterasyonda dahi optimum sonuçlara ulaşamamışlardır.



Şekil 4.3. Cap131, Cap132, Cap133 ve Cap134 problemleri için algoritmaların yakınsama grafikleri

CapA, CapB ve CapC problemleri için karşılaştırılan algoritmaların elde ettiği sonuçlar Çizelge 4.5'te verilmiştir. CapA, CapB ve CapC problemleri 100 tesis ve 1000

müşteriden oluşan, çok büyük boyutlu olarak nitelendirilen problemler kategorisine dahil edildiği için, karşılaştırmalarda kullanılan algoritmaların çoğunun 30 çalıştırmanın tamamında optimum sonuçlara ulaşması oldukça güçtür.

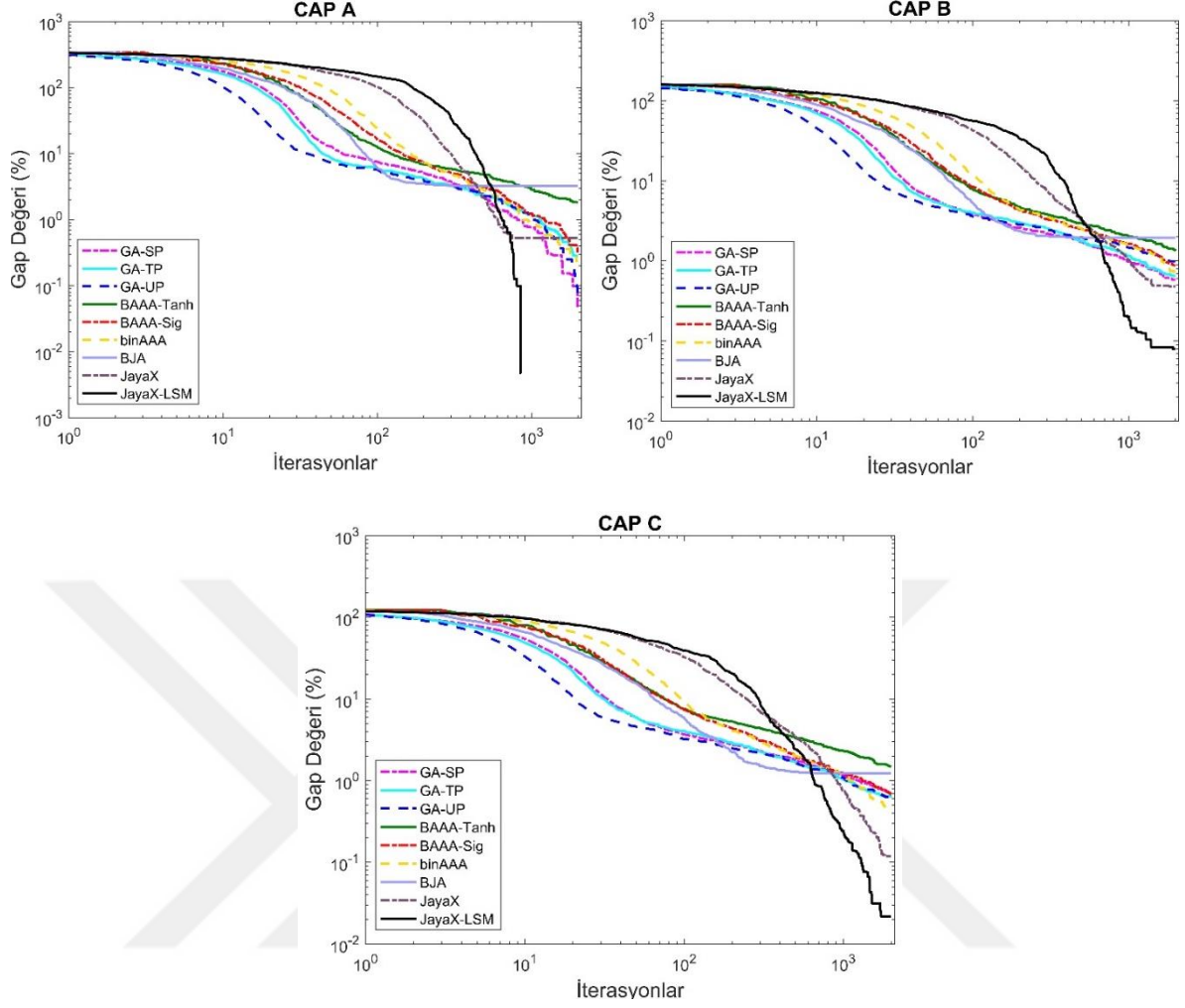
Çizelge 4.5. JayaX ve JayaX-LSM'nin CapA, CapB ve CapC problemleri için yakın zamanda önerilen algoritmalarla karşılaştırılması

Algoritma	Kriter	CapA	CapB	CapC
GA-SP	Ortalama	17164354.456	13054858.045	11586692.969
	Gap	0.04605	0.58391	0.70486
	Hit	24	9	2
	Std. Sap.	22451.206	66658.649	51848.248
GA-TP	Ortalama	17205089.145	13063527.186	11577797.524
	Gap	0.28348	0.65071	0.62755
	Hit	24	11	0
	Std. Sap.	139690.216	89122.485	46346.052
GA-UP	Ortalama	17166811.915	13107633.077	11578600.532
	Gap	0.06037	0.99053	0.63453
	Hit	24	3	0
	Std. Sap.	35181.974	79714.021	57031.219
BAAA-Tanh	Ortalama	17471223.794	13153617.764	11676427.752
	Gap	1.83470	1.34483	1.48479
	Hit	3	0	0
	Std. Sap.	225123.921	73978.543	101438.607
BAAA-Sig	Ortalama	17210900.533	13093705.559	11583462.068
	Gap	0.31735	0.88322	0.67678
	Hit	16	1	1
	Std. Sap.	90743.456	62168.803	45788.678
binAAA	Ortalama	17196684.161	13066644.044	11555763.878
	Gap	0.23449	0.67472	0.43604
	Hit	21	3	3
	Std. Sap.	78259.572	57572.956	43963.880
BJA	Ortalama	17714111.072	13230727.167	11646233.385
	Gap	3.25042	1.93893	1.22235
	Hit	1	0	0
	Std. Sap.	376679.686	126103.466	92452.923
JayaX	Ortalama	17246690.954	13041453.028	11519301.843
	Gap	0.52596	0.48063	0.11914
	Hit	19	15	8
	Std. Sap.	151110.150	89306.162	30219.354
JayaX-LSM	Ortalama	17156454.478	12989379.443	11508089.967
	Gap	0.00000	0.07942	0.02169
	Hit	30	26	17
	Std. Sap.	0.000	27033.019	5455.945

Çizelge 4.5 incelendiğinde, CapA problemi hariç olmak üzere karşılaştırmalarda kullanılan algoritmalar CapB ve CapC problemlerinde, 30 çalıştırmanın bir kısmı için optimum sonuçlara ulaşmışlardır ve *gap* değerleri dikkate alındığında en başarılı

algoritmanın JayaX-LSM olduğu görülmektedir. Bununla birlikte, JayaX-LSM algoritması, CapA problemi için 30 çalıştırmanın tamamında, CapB problemi için 26 kez ve CapC problemi için 17 kez olmak optimum sonuçlara ulaşabilmiştir. Çizelge 4.5'e göre CapA, CapB ve CapC problemlerinin çözümü için ikinci en iyi algoritmanın JayaX algoritması olduğu görülmektedir. JayaX algoritması CapA problemi için 19 kez, CapB problemi için 15 kez ve CapC problemi için 8 kez olmak üzere optimum çözüme ulaşmıştır. JayaX algoritmasının elde ettiği bu sonuçlar bile yakın zamanda önerilen binAAA, BJA, BAAA-Tanh ve BAAA-Sig algoritmalarına kıyasla oldukça başarılı sonuçlardır. Bu da gösteriyor ki önerilen JayaX algoritması, LSM mekanizmasını kullanılsa bile oldukça başarılı bir algoritmadır. Küçük boyutlu problemler için nispeten kötü sonuçlar elde eden GA-SP, GA-TP, GA-UP algoritmaları, CapA, CapB ve CapC problemleri için başarılı sonuçlar elde etmiştir. Cap problemleri için genellikle iyi sonuçlara ulaşan binAAA algoritması ise *gap* değerleri dikkate alındığında CapA, CapB ve CapC problemleri için başarılı sonuçlar elde etmesine rağmen, diğer Cap problemlerinde elde ettikleri başarıyı çok büyük boyutlu olan CapA, CapB ve CapC problemleri için gösterememiştir. CapA, CapB ve CapC problemlerinin çözümü için en zayıf kalan algoritmalar ise BAAA-Tanh ve BJA algoritmalarıdır. Çizelge 4.2, 4.3, 4.4 ve 4.5, bir bütün olarak değerlendirildiğinde, Cap problemlerinin çözümü için JayaX ve JayaX-LSM algoritmalarının literatürde yakın zamanda önerilmiş olan ve bu çalışma kapsamında karşılaştırmalarda kullanılan algoritmalara göre oldukça başarılı sonuçlara ulaştığı anlaşılmaktadır.

CapA, CapB ve CapC problemleri için algoritmaların iterasyonlar boyunca optimum sonuca yakınsama grafikleri Şekil 4.4'de verilmiştir. Şekil 4.4 incelendiğinde CapA, CapB ve CapC problemleri için en hızlı yakınsamayı JayaX-LSM algoritması göstermiştir. JayaX-LSM algoritması ortalama 1000 iterasyonda CapA problemi için optimum sonuçlara ulaşmıştır. CapB ve CapC problemleri için algoritmaların elde ettiği sonuçlar dikkate alındığında, ikinci en iyi yakınsamayı JayaX algoritması göstermiştir. CapA problemi için ise ikinci en iyi yakınsamayı GA-SP ve GA-UP algoritmaları göstermiştir. Diğer algoritmaların ise CapA, CapB ve CapC problemleri için optimum sonuçlara yakınsaması daha yavaş olmuştur.



Şekil 4.4. CapA, CapB ve CapC problemleri için algoritmaların yakınsama grafikleri

Çizelge 4.6’da ise, JayaX ve JayaX-LSM algoritmaları BPSO algoritması ile karşılaştırılmıştır. BPSO’nun sonuçları doğrudan Kiran ve Gunduz (2013) yapmış oldukları çalışmadan alınmıştır. Çizelge 4.6’dan da anlaşılacağı üzere, JayaX ve JayaX-LSM, BPSO’dan daha başarılı çözümler bulmaktadır. JayaX-LSM algoritması 15 Cap probleminin 13 tanesi için, JayaX algoritması ise 12 tanesi için 30 çalıştırmanın tamamında optimum sonuçlara ulaşmışlardır. BPSO algoritması ise sadece küçük boyutlu Cap71 ve Cap72 problemleri için optimum sonuçlara ulaşabilmiştir. Çizelge 4.6’da verilen sonuçlar dikkate alındığında önerilen algoritmaların BPSO algoritmasına kıyasla daha etkin ve güvenilir olduğu anlaşılmaktadır.

Çizelge 4.6. JayaX ve JayaX-LSM algoritmalarının BPSO algoritması ile karşılaştırılması

Problem	BPSO		JayaX			JayaX-LSM		
	Gap	Std. Sap.	Gap	Std. Sap.	Hit	Gap	Std. Sap.	Hit
Cap71	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap72	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap73	0.0242	634.62	0.0000	0.00	30	0.0000	0.00	30
Cap74	0.0088	500.27	0.0000	0.00	30	0.0000	0.00	30
Cap101	0.0462	566.44	0.0000	0.00	30	0.0000	0.00	30
Cap102	0.0148	386.76	0.0000	0.00	30	0.0000	0.00	30
Cap103	0.0422	485.26	0.0000	0.00	30	0.0000	0.00	30
Cap104	0.0810	1951.81	0.0000	0.00	30	0.0000	0.00	30
Cap131	0.1317	1207.63	0.0000	0.00	30	0.0000	0.00	30
Cap132	0.0914	1196.19	0.0000	0.00	30	0.0000	0.00	30
Cap133	0.1115	821.28	0.0000	0.00	30	0.0000	0.00	30
Cap134	0.1346	2285.42	0.0000	0.00	30	0.0000	0.00	30
CapA	2.1785	374302.81	0.5260	151110.15	19	0.0000	0.00	30
CapB	1.9490	176206.07	0.4806	89306.16	15	0.0794	27033.02	26
CapC	1.4870	92977.85	0.1191	30219.35	8	0.0217	5455.94	17

Bir diğer kıyaslama ise ikili optimizasyon problemlerinin çözümü için literatüre kazandırılmış olan ikili ABC algoritmaları ile yapılmıştır. Karşılaştırmalar da kullanılan algoritmalar için deneysel sonuçlar Çizelge 4.7’de verilmiştir. İkili ABC algoritmalarından binABC ve DisABC algoritmalarının sonuçları doğrudan Kiran ve Gunduz (2013) yapmış oldukları çalışmadan, ve ABCbin algoritmasının sonuçları ise doğrudan Kiran (2015b) yapmış olduğu çalışmadan alınmıştır. İkili ABC algoritmaları ile bu çalışma kapsamında önerilen algoritmalarımız, küçük, orta ve büyük ölçekli problemlerde kabul edilebilir bir performans göstermişlerdir. Ancak Çizelge 4.7 incelendiğinde önerilen algoritmalar, binABC, DisABC ve ABCbin algoritmalarına kıyasla KYYP’nin çözümü için daha etkili bir performans ortaya koymuştur. Deneysel sonuçlar dikkate alındığında önerilen JayaX ve JayaX-LSM algoritmaları, ABC tabanlı ikili optimizasyon algoritmalarına kıyasla daha etkin, daha sağlam ve daha verimli algoritmalarlardır.

Çizelge 4.7. JayaX ve JayaX-LSM algoritmalarının ABC tabanlı binABC, DisABC ve ABCbin algoritmaları ile karşılaştırılması

Problem	binABC		DisABC		ABCbin		JayaX		Hit	JayaX-LSM		
	Gap	Std. Sap.	Gap	Std. Sap.	Gap	Std. Sap.	Gap	Std. Sap.		Gap	Std. Sap.	Hit
Cap71	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap72	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap73	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap74	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap101	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap102	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap103	0.0000	0.00	0.0000	0.00	0.0051	85.67	0.0000	0.00	30	0.0000	0.00	30
Cap104	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
Cap131	0.0000	0.00	0.6196	2337.64	0.1967	1065.73	0.0000	0.00	30	0.0000	0.00	30
Cap132	0.0000	0.00	0.0945	813.37	0.0199	213.28	0.0000	0.00	30	0.0000	0.00	30
Cap133	0.1215	200.24	0.0309	359.03	0.0747	561.34	0.0000	0.00	30	0.0000	0.00	30
Cap134	0.0000	0.00	0.0000	0.00	0.0000	0.00	0.0000	0.00	30	0.0000	0.00	30
CapA	2.9622	236833.50	0.1522	74782.61	3.1723	268685.20	0.5260	151110.15	19	0.0000	0.00	30
CapB	2.5081	91430.13	3.3027	109738.50	2.8154	88452.80	0.4806	89306.16	15	0.0794	27033.02	26
CapC	2.5800	82312.70	4.6968	95778.78	2.0374	78162.20	0.1191	30219.35	8	0.0217	5455.94	17

Çizelge 4.8’de, JayaX ve JayaX-LSM algoritmaları ikili TSA (Cinar ve Kiran, 2018) ve ikili DE algoritmaları (Kashan ve ark., 2013) ile karşılaştırılmıştır. SimLogicTSA, DisDE/rand ve binDE algoritmalarının deneysel sonuçları doğrudan Cinar ve Kiran (2018)’in yapmış oldukları çalışmadan alınmıştır. SimLogicTSA, küçük, orta ve büyük boyutlu problemlerde önerilen JayaX ve JayaX-LSM algoritmalarıyla karşılaştırılabilir bir performans göstermiştir. CapA, CapB ve CapC gibi çok büyük boyutlu problemlerde ise, JayaX-LSM karşılaştırılan tüm algoritmalarından daha etkin ve daha güvenilir sonuçlar elde etmiştir. Deneysel sonuçlar dikkate alındığında SimLogicTSA algoritması CapA problemi için 30 çalıştırmanın tamamında, DisDE/rand algoritması 29 kez, JayaX algoritması 19 kez ve binDE algoritması 8 kez optimum çözümlere ulaşmışlardır. Bu nedenle CapA problemi için SimLogicTSA algoritması DisDE/rand, JayaX ve binDE algoritmasına kıyasla daha başarılıdır. Ancak SimLogicTSA algoritması CapB ve CapC problemleri için aynı başarıyı gösterememiştir. CapB ve CapC problemi için SimLogicTSA ve binDE algoritmaları 30 çalıştırmanın hiçbirinde optimum çözüme ulaşmazken, DisDE/rand algoritması sırasıyla 18 kez ve 8 kez, JayaX algoritması ise sırasıyla 15 kez ve 8 kez optimum çözümlere ulaşabilmiştir. Sonuç olarak JayaX algoritması ve DisDE/rand algoritması CapB ve CapC problemlerinin çözümü için SimLogicTSA ve binDE algoritmalarına kıyasla daha başarılı bir performans sergilemişlerdir.

Çizelge 4.8. JayaX ve JayaX-LSM algoritmalarının SimLogicTSA, DisDE/rand ve binDE algoritmaları ile karşılaştırılması

Problem	DisDE/rand		binDE		SimLogicTSA			JayaX			JayaX-LSM		
	Gap	Hit	Gap	Hit	Gap	Std. Sap.	Hit	Gap	Std. Sap.	Hit	Gap	Std. Sap.	Hit
Cap71	0.0000	30	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap72	0.0000	30	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap73	0.0000	30	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap74	0.0000	30	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap101	0.0036	29	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap102	0.0049	29	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap103	0.0055	27	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap104	0.0000	30	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap131	0.0036	29	0.0036	29	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap132	0.0000	30	0.0050	29	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap133	0.0138	25	0.0138	24	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
Cap134	0.0000	30	0.0000	30	0.0000	0.00	30	0.0000	0.00	30	0.0000	0.00	30
CapA	0.0370	29	1.3000	8	0.0000	0.00	30	0.5260	151110.15	19	0.0000	0.00	30
CapB	0.1890	18	1.5200	0	0.3176	47993.68	0	0.4806	89306.16	15	0.0794	27033.02	26
CapC	0.0909	8	1.5500	0	0.4120	48379.53	0	0.1191	30219.35	8	0.0217	5455.94	17

4.2. CEC 2015 Nümerik Fonksiyonlarının Çözümü için Deneysel Sonuçlar

Önerilen algoritmanın başarısını göstermek amacıyla yapılan bir diğer deneysel çalışma ise 15 nümerik fonksiyondan oluşan CEC 2015 veri seti üzerine olmuştur. Geliştirilen JayaX ve JayaX-LSM algoritmalarının performans analizi Bölüm 4.1’de KYT problemlerinin üzerine uygulanarak yapılmıştı. Bölüm 4.1’deki deneysel sonuçlar dikkate alındığında bünyesinde bir yerel arama algoritması bulunduran JayaX-LSM algoritmasının, JayaX algoritmasının geliştirilmiş bir versiyonu olduğu görülmektedir. Bundan dolayı bu bölümde ve bundan sonraki bölümde yapılan deneysel çalışmalarda sadece JayaX-LSM algoritması kullanılmıştır. JayaX-LSM algoritması, ikili optimizasyon problemlerinin çözümü için Banitalebi ve ark. (2016) tarafından literatüre kazandırılmış olan kendinden adaptif ikili farksal gelişim algoritması (SabDE), Kong ve ark., (2015) tarafından önerilen basitleştirilmiş ikili harmoni arama algoritması (SBHS), Chen ve ark., (2015) tarafından önerilen ikili farksal gelişim algoritması (BLDE), Beheshti ve ark., (2015) tarafından önerilen ikili hibrit topoloji parçacık sürü optimizasyonu tabanlı karesel enterpolasyon algoritması (BHTPSO-QI), Nezamabadi-pour (2015) tarafından geliştirilen ikili kuantumdan ilham alan yer çekimi arama algoritması (BQIGSA) ve Ozturk ve ark., (2015) tarafından önerilen GA tabanlı yeni ikili ABC algoritması (GBABC) (Ozturk ve ark., 2015) ile karşılaştırılmıştır. SabDE, BQIGSA, GBABC, BHTPSO-QI, BLDE ve SBHS algoritmalarının sonuçları doğrudan Banitalebi ve ark. (2016) yapmış oldukları çalışmadan alınmıştır. Adil bir karşılaştırma yapılabilmesi amacıyla önerilen algoritmanın ortak parametreleri Banitalebi ve ark. (2016) yapmış oldukları çalışma ile aynı olacak şekilde belirlenmiştir. CEC 2015 problemlerinin çözümü için algoritmaların ortak parametreleri Çizelge 4.9’da verilmiştir. Önerilen algoritmanın kendi kontrol parametreleri olan $P_{local} = 0.01$ ve $P = 0.50$ olarak belirlenmiştir.

Çizelge 4.9. CEC 2015 fonksiyonlarının çözümü için algoritmaların ortak parametreleri

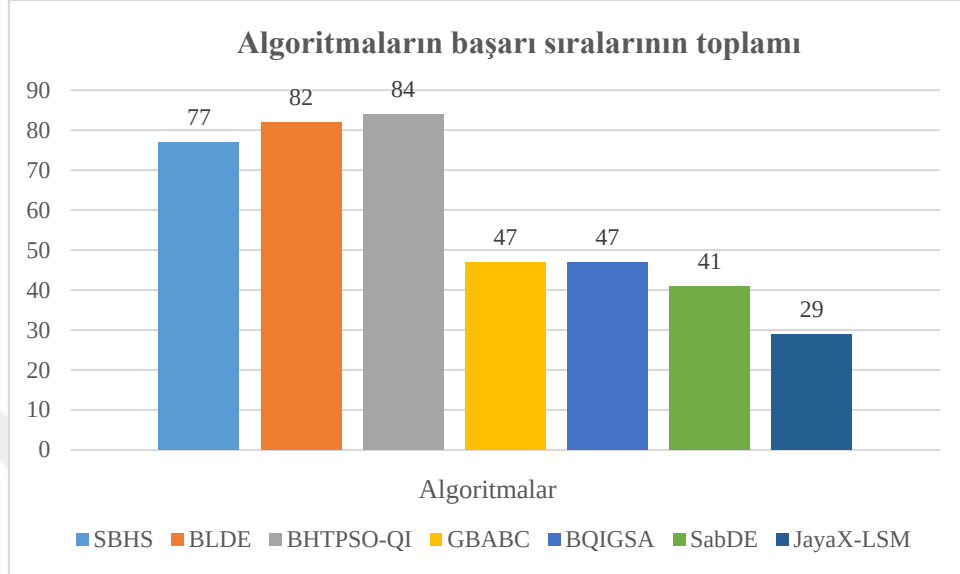
Parametre	Açıklama	Değeri
N	Popülasyon boyutu	50
D	Problemin boyutu	10
m	Her bir boyut için toplam bit uzunluğu	50
$D * m$	Aday çözümün boyutu	500
MaxFEs	Sonlandırma kriteri	$D \times 10000$
ÜstS	Problemin tanım aralığındaki üst sınırı	100
AltS	Problemin tanım aralığındaki alt sınırı	-100
Run	Algoritmanın çalıştırılma sayısı	30

CEC 2015 nümerik optimizasyon problemlerinin çözümü için JayaX-LSM, SabDE, BQIGSA, GBABC, BHTPSO-QI, BLDE ve SBHS algoritmalarının elde ettiği sonuçlar Çizelge 4.10’da verilmiştir. Çizelge 4.10’un ilk sütunu ilgili problemin adını, ikinci sütunu karşılaştırma kriterlerini, diğer sütunlarda ise karşılaştırılan algoritmaların bulunduğu sonuçları göstermektedir. Deneylerde her bir problem için 30 bağımsız çalıştırma yapılmıştır ve elde edilen sonuçlar ortalama, standart sapma (Std. Sap.) ve sıra (*rank*) değerlerine göre değerlendirilmektedir. Algoritmaların problemler üzerindeki başarısının karşılaştırılması amacıyla her bir problem için *sıra* işlemi yapılmıştır. *sıra* işlemi şu şekilde gerçekleşmektedir: öncelikle problemi çözmek için kullanılan algoritmaların sayısı belirlenir, bu sayı *sıra* işlemi için kullanılacak en büyük sırayı ifade etmektedir. Daha sonra ilgili problem için bulunan sonuçlar sıralanır. En iyi sonucu bulan algoritmaya 1. sıra, en kötü sonucu bulan algoritmaya ise sonuncu sıra verilir. Bütün problemler için *sıra* işlemi gerçekleştirilir. En sonunda her algoritmanın, her bir problem için başarı sıraları toplanır ve *sıra* numaralarının toplamı en az olan algoritma, en başarılı algoritma olarak işaretlenir.

Çizelge 4.10. CEC 2015 nümerik fonksiyonlarının çözümü için algoritmaların performans karşılaştırması

f	Kriter	SBHS	BLDE	BHTPSO-QI	GBABC	BQIGSA	SabDE	JayaX-LSM
f_1	Ortalama	3.700E+08	1.087E+11	7.510E+08	2.729E+07	8.419E+07	3.093E+08	9.728E+07
	Std. Sap.	2.271E+08	2.351E+11	2.203E+09	2.267E+07	7.354E+07	1.168E+08	9.588E+07
	Sıra	5	7	6	1	2	4	3
f_2	Ortalama	4.493E+09	4.747E+09	5.132E+09	2.864E+09	7.834E+09	2.541E+09	1.968E+04
	Std. Sap.	1.790E+06	3.209E+08	4.685E+08	2.374E+09	6.527E+09	5.008E+09	1.075E+04
	Sıra	4	5	6	3	7	2	1
f_3	Ortalama	3.203E+02	3.201E+02	3.203E+02	3.202E+02	3.202E+02	3.200E+02	3.065E+02
	Std. Sap.	7.013E-02	1.093E-02	7.426E-02	2.641E+02	2.641E+02	2.044E-02	1.247E+00
	Sıra	5	3	5	4	4	2	1
f_4	Ortalama	4.478E+02	4.378E+02	4.478E+02	4.358E+02	4.389E+02	4.116E+02	6.696E+02
	Std. Sap.	5.171E+00	7.754E+00	1.122E+01	3.599E+02	3.621E+02	7.606E+00	1.481E+02
	Sıra	5	3	5	2	4	1	6
f_5	Ortalama	2.592E+03	1.934E+03	1.804E+03	1.108E+03	1.542E+03	9.330E+02	5.007E+02
	Std. Sap.	1.959E+02	3.088E+02	3.926E+02	9.736E+02	1.275E+03	9.464E+01	3.774E-01
	Sıra	7	6	5	3	4	2	1
f_6	Ortalama	5.312E+04	2.484E+06	6.799E+06	7.442E+06	5.582E+05	4.625E+04	6.006E+02
	Std. Sap.	1.512E+04	7.369E+06	1.186E+07	1.321E+07	6.055E+05	4.076E+04	2.362E-01
	Sıra	3	5	6	7	4	2	1
f_7	Ortalama	8.368E+02	8.472E+02	8.510E+02	7.589E+02	7.392E+02	7.752E+02	7.019E+02
	Std. Sap.	7.981E+00	1.289E+01	1.694E+01	4.668E+02	4.324E+02	4.155E+03	2.760E+00
	Sıra	5	6	7	3	2	4	1
f_8	Ortalama	3.344E+08	7.104E+09	5.407E+07	3.949E+07	2.948E+06	2.395E+07	8.075E+02
	Std. Sap.	1.810E+09	5.651E+09	8.312E+07	2.442E+08	2.469E+06	5.432E+07	7.075E+00
	Sıra	6	7	5	4	2	3	1
f_9	Ortalama	1.189E+03	1.190E+03	1.209E+03	1.017E+03	1.048E+03	1.177E+03	9.034E+02
	Std. Sap.	5.210E+00	1.736E+01	2.050E+01	8.397E+02	8.643E+02	4.102E+01	3.154E-01
	Sıra	5	6	7	2	3	4	1
f_{10}	Ortalama	9.038E+08	1.326E+10	1.320E+09	9.909E+05	4.426E+04	2.416E+07	1.489E+05
	Std. Sap.	6.596E+08	1.724E+10	2.073E+09	3.659E+06	4.477E+04	8.862E+07	4.007E+05
	Sıra	5	7	6	3	1	4	2
f_{11}	Ortalama	1.737E+03	1.599E+03	1.632E+03	1.159E+03	1.172E+03	1.114E+03	1.106E+03
	Std. Sap.	1.661E+01	3.313E+01	3.371E+01	9.557E+02	9.669E+02	1.131E+01	1.708E+00
	Sıra	7	5	6	3	4	2	1
f_{12}	Ortalama	1.445E+03	1.440E+03	1.437E+03	1.264E+03	1.255E+03	1.224E+03	1.293E+03
	Std. Sap.	5.683E-01	3.247E+00	7.552E+00	1.044E+03	1.035E+03	1.302E+00	6.664E+01
	Sıra	7	6	5	3	2	1	4
f_{13}	Ortalama	1.464E+10	2.882E+10	5.365E+08	1.446E+03	1.452E+03	2.815E+09	1.620E+03
	Std. Sap.	9.749E+09	2.073E+10	2.294E+09	1.193E+03	1.197E+03	4.158E+09	3.812E+00
	Sıra	6	7	4	1	2	5	3
f_{14}	Ortalama	2.736E+03	4.267E+03	5.005E+03	2.162E+03	3.356E+03	1.727E+03	1.602E+03
	Std. Sap.	2.220E+02	2.569E+03	2.703E+03	2.091E+03	2.869E+03	4.411E+02	4.683E+00
	Sıra	4	6	7	3	5	2	1
f_{15}	Ortalama	1.700E+03	1.700E+03	1.734E+03	2.012E+03	1.530E+03	1.700E+03	1.640E+03
	Std. Sap.	1.552E-04	2.622E-05	1.851E+02	1.659E+03	1.262E+03	2.177E-05	1.856E+02
	Sıra	3	3	4	5	1	3	2

CEC 2015 nümerik optimizasyon problemlerinin çözümü için JayaX-LSM, SabDE, BQIGSA, GBABC, BHTPSO-QI, BLDE ve SBHS algoritmalarının elde ettiği sonuçların *sıra* işlemine göre başarısı Şekil 4.5'te verilmiştir.



Şekil 4.5. CEC 2015 nümerik optimizasyon problemlerinin çözümü için algoritmaların *sıra* mekanizmasına göre başarı sıralarının toplamı

Şekil 4.5'te verilen sonuçlar değerlendirildiğinde, *sıra* mekanizmasına göre sıra sayılarının toplamı en az olan algoritma JayaX-LSM algoritmasıdır. Bu da JayaX-LSM algoritmasının CEC 2015 nümerik problemlerinin çözümü için, karşılaştırılan diğer algoritmalarla kıyasla daha iyi sonuçlar bulduğunu göstermektedir. En başarılı ikinci algoritma ise *sıra* mekanizmasına göre sıra sayılarının toplamı 41 olan SabDE algoritmasıdır. En kötü sonuçlara ise sıra sayılarının toplamı 84 olan BHTPSO-QI algoritması ulaşmıştır.

4.3. Rüzgâr Türbini Yerleştirme Probleminin Çözümü için Deneysel Sonuçlar

Bu bölümde önerilen JayaX-LSM algoritması rüzgâr türbini yerleştirme probleminin üzerine uygulanmıştır. Yapılan bu çalışmada, Mosetti ve ark. (1994) rüzgâr türbini yerleştirme probleminin çözümü için kullandıkları $2\text{ km} \times 2\text{ km}$ 'lik bir test alanı kullanılmıştır. Aynı test alanını, Grady ve ark. (2005), González ve ark. (2010), Pookpunt ve Ongsakul (2013), Beşkirli ve ark. (2018) yapmış oldukları çalışmalarda kullanılmışlardır. Yapılan bu çalışmada, $2\text{ km} \times 2\text{ km}$ 'lik test alanı Beşkirli ve ark. (2018) yapmış oldukları çalışmada olduğu gibi 10×10 (100 muhtemel türbin

yerleştirme lokasyonu) ve 20×20 (400 muhtemel türbin yerleştirme lokasyonu) olmak üzere iki farklı ızgara alanına bölünmüştür. Birinci tür problemde, test alanı 10×10 'luk ızgara alanına bölünmüştür. Bu durumda test alanı 100 eşit kareye bölünmüş olmaktadır ve türbinler arası uyandırma etkisinin en aza indirilebilmesi için iki türbin arasındaki mesafenin en az $5D$ (200 m) olması şartı sağlanmış olmaktadır. 10×10 'luk ızgara alanı için muhtemel farklı çözümlerin sayısı 2^{100} olduğundan dolayı, ilk tür problem büyük ölçekli bir problem olarak sınıflandırılabilir. İkinci tür problemde ise, test alanı 20×20 'lik bir ızgara alanına bölünmüştür. Bu durumda test alanı 400 eşit kareye bölünmüş olmaktadır ve iki komşu hücre merkezi arasındaki mesafe 100 m indiği için türbinler arası uyandırma etkisinin kontrolü zorlaşmaktadır. 20×20 'lik ızgara alanı için, muhtemel farklı çözümlerin sayısı 2^{400} olduğu için, ikinci tür problem çok büyük ölçekli bir problem olarak sınıflandırılabilir. Rüzgâr türbini yerleştirme probleminde amaç, rüzgâr çiftliğine yerleştirilen türbinlerden maksimum enerji (P_{total}) elde etmektir ve bunu yaparken de kurulum, bakım, onarım ve elektrik şebekesi gibi maliyetlerin minimize etmektir. Rüzgâr türbini yerleştirme problemi için karşılaştırılan algoritmalarında kullandığı probleme özgü parametreler Çizelge 4.11'de verilmiştir.

Çizelge 4.11. Rüzgâr türbini yerleştirme probleminin parametreleri (Mittal ve ark., 2016)

Parametre	Açıklama	Değeri
-	Rüzgâr çiftliğinin alanı	$2\text{ km} \times 2\text{ km}$
-	Rotor çapı	40 m
z	Gövde yüksekliği	60 m
C_T	İtme katsayısı	0.88
z_0	Yüzey pürüzlülüğü	0.3 m
u_0	Serbest rüzgâr hızı	12 m/s
α	Sürüklenme sabitini	0.0944

Önerilen algoritma için parametre analizi Bölüm 4.3.1'de yapılmıştır. Yapılan parametre analizine göre; önerilen algoritmanın popülasyon boyutu $N = 60$, yerel aramanın gerçekleştirilmesi şartı $P_{local} = 0.005$ ve güncellenecek boyut sayısını belirleyen P parametresi 0.50 olarak seçilmiştir. Sonlandırma kriteri ise diğer algoritmaların sonuçları ile adil bir karşılaştırma yapılması amacıyla $MaxFEs = 6 \times 10^5$ olarak seçilmiştir. Ayrıca ikili optimizasyon problemlerinin çözümü için literatürde önerilmiş olan global harmoni arama algoritmasına dayalı iki farklı ikili optimizasyon algoritması olan; Zou ve ark.(2011) tarafından önerilen NGHS ve Xiang ve ark. (2014) tarafından önerilen DGHS algoritmaları rüzgâr türbini problemi üzerine uygulanmıştır.

NGHS ve DGHS algoritmaları için de popülasyon boyutu önerilen algoritma ile aynı olacak şekilde seçilmiştir. NGHS ve DGHS algoritmalarının kendine özgü parametreleri ise önerilen çalışmalarda verilen şekliyle alınmıştır. NGHS'nin kendine özgü parametreleri: genetik mutasyon olasılığı parametresi p_m 'dir ve p_m problemin boyutuna göre $p_m = 2/D$ olacak şekilde belirlenmektedir. DGHS'nin kendine özgü parametreleri ise: maksimum bellek değerlendirme oranı $HCMR_{max} = 0.95$, minimum bellek değerlendirme oranı $HCMR_{min} = 0.3$ ve eğim ayarlama oranı (pitch adjusting rate) $PAR = 0.75$ olarak belirlenmiştir. Bütün algoritmalar 30 kez çalıştırılmıştır.

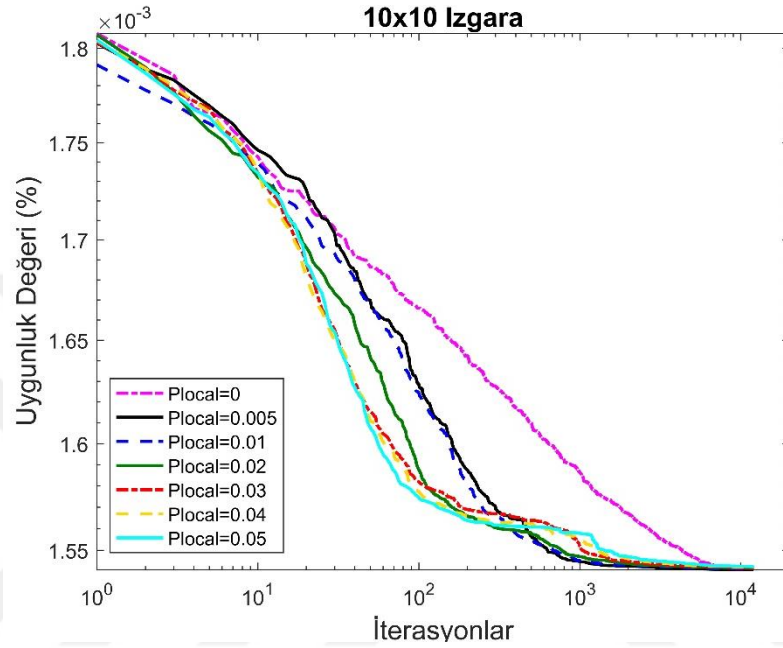
4.3.1. Rüzgâr türbini yerleştirme probleminin çözümü için önerilen algoritmanın parametre analizi

Önerilen algoritmanın kendine özgü parametreleri P_{local} ve P parametreleridir ve bu parametrelerin etkinliğini göstermek amacıyla bu bölümde P_{local} , N ve P parametreleri analiz edilmiştir. Tüm deneysel analizler için, $MaxFES$ sayısı 6×10^5 olarak belirlenmiştir ve önerilen algoritma, optimum algoritmik parametre değerlerine karar vermek için rastgele 30 kez çalıştırılmıştır. Deneysel analiz sonucunda ulaşılan uygunluk değerleri ($Cost/P_{total}$), 30 çalıştırmanın sonucuna göre *En iyi*, *En kötü*, *Ortalama* ve *standart sapma* (Std. Sap.) olarak kaydedilmektedir. Parametre analizinin ilk aşamasında P_{local} parametresi analiz edilmiştir. Bu analiz için $N = 50$ ve $P = 0.50$ olarak seçilmiştir. Analizler hem 10×10 'luk ızgara için hem de 20×20 'lik ızgara için yapılmıştır. 10×10 'luk ızgara alanı üzerinde farklı P_{local} değeri için önerilen algoritmanın bulduğu sonuçlar Çizelge 4.12'de verilmiştir. P_{local} değeri 0 ile 0.05 arasında olacak şekilde analiz edilmiştir. $P_{local} = 0$ olarak seçilmesi LSM'nin gerçekleşmediği anlamına gelmektedir.

Çizelge 4.12. Farklı P_{local} değerleri için 10×10 'luk ızgara yapısı üzerinde elde edilen uygunluk değerlerinin karşılaştırılması

	En iyi	En kötü	Ortalama	Std. Sap.
$P_{local} = 0$	0.0015408	0.0015454	0.0015411	9.61E-07
$P_{local} = 0.005$	0.0015408	0.0015421	0.0015411	3.51E-07
$P_{local} = 0.01$	0.0015408	0.0015421	0.0015412	3.52E-07
$P_{local} = 0.02$	0.0015408	0.0015430	0.0015417	7.35E-07
$P_{local} = 0.03$	0.0015408	0.0015430	0.0015418	6.14E-07
$P_{local} = 0.04$	0.0015414	0.0015430	0.0015421	5.54E-07
$P_{local} = 0.05$	0.0015414	0.0015442	0.0015425	6.84E-07

Çizelge 4.12 incelendiğinde LSM mekanizması kapalı olduğu halde 10×10 'luk ızgara için önerilen algoritma *en iyi* sonuçlara ulaşabilmiştir. Ancak P_{local} değeri büyüdükçe algoritmanın bulduğu *ortalama* sonuçlar kötüye gitmiştir. 10×10 'luk ızgara için farklı P_{local} değerleri için önerilen algoritmanın 30 çalıştırmanın sonucuna göre elde ettiği *ortalama* değerlerin yakınsama grafiği Şekil 4.6'da verilmiştir.



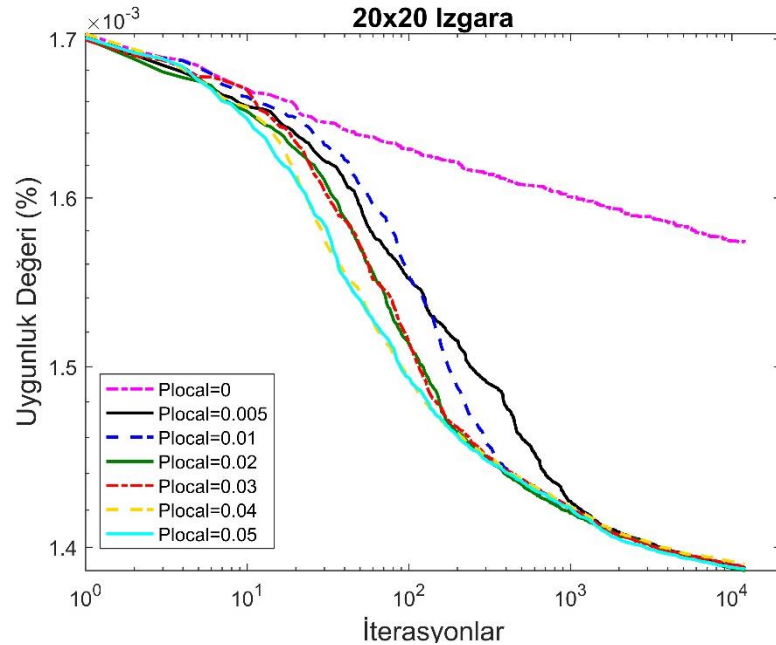
Şekil 4.6. Farklı P_{local} değerleri için 10×10 'luk ızgara yapısı üzerinde yakınsama grafiği

10×10 'luk ızgara için Şekil 4.6'da verilen yakınsama grafiği incelendiğinde, algoritmalar *ortalama* değerlerine genellikle 1000 iterasyon civarında ulaşmışlardır. Ancak $P_{local} = 0.005$ olarak seçildiğinde, önerilen algoritma diğer P_{local} değerlerine göre az da olsa daha hızlı yakınsamıştır. 10×10 ızgara yapısında karar değişkenlerinin sayısı 100'dür. Bu nedenle P_{local} parametresinin, algoritmanın performansına etkisi fark edilememektedir. Ancak problemin boyutu arttıkça yerel arama algoritmasının performansının da artacağını göstermek amacıyla önerilen algoritma farklı P_{local} değerleri için 20×20 'lik ızgara yapısı içinde analiz edilmiştir. 20×20 'lik ızgara yapısı için karar değişkenlerinin sayısı 400'dür. Bu durumda problemin karmaşıklığı 10×10 'luk ızgara yapısına kıyasla artmaktadır. 20×20 'lik ızgara alanı üzerinde farklı P_{local} değeri için önerilen algoritmanın bulduğu sonuçlar Çizelge 4.13'te verilmiştir.

Çizelge 4.13. Farklı P_{local} değerleri için 20×20 'lik ızgara yapısı üzerinde elde edilen uygunluk değerlerinin karşılaştırılması

	En iyi	En kötü	Ortalama	Std. Sap.
$P_{local} = 0$	0.0015535	0.0015903	0.0015731	8.94E-06
$P_{local} = 0.005$	0.0013734	0.0014090	0.0013890	7.76E-06
$P_{local} = 0.01$	0.0013757	0.0014156	0.0013885	9.80E-06
$P_{local} = 0.02$	0.0013751	0.0014097	0.0013876	7.95E-06
$P_{local} = 0.03$	0.0013751	0.0014064	0.0013894	7.36E-06
$P_{local} = 0.04$	0.0013767	0.0014078	0.0013911	7.38E-06
$P_{local} = 0.05$	0.0013765	0.0014048	0.0013885	7.75E-06

Çizelge 4.13 incelendiğinde LSM mekanizması kapalı olduğu zaman ($P_{local} = 0$), 20×20 'lik ızgara yapısı üzerinde önerilen algoritmanın bulunduğu *en iyi*, *en kötü* ve *ortalama* sonuçlar diğer P_{local} değerlerinin sonucuna göre oldukça kötüdür. Elde edilen sonuçlardan anlaşılmaktadır ki LSM mekanizması önerilen algoritmanın başarı performansını doğrudan etkilemektedir. Çizelge 4.13'e göre önerilen algoritma en iyi sonuçlara $P_{local} = 0.005$ olarak seçildiğinde ulaşmıştır. Bu nedenle P_{local} parametresi diğer deneysel analizlerde 0.005 olarak seçilmiştir. 20×20 'lik ızgara yapısı üzerinde farklı P_{local} değerleri için önerilen algoritma için 30 çalıştırmanın sonucuna göre elde ettiği *ortalama* değerlerin yakınsama grafiği Şekil 4.7'de verilmiştir.



Şekil 4.7. Farklı P_{local} değerleri için 20×20 'lik ızgara üzerinde yakınsama grafiği

20×20 'lik ızgara için Şekil 4.7'de verilen yakınsama grafiği incelendiğinde, algoritmalar *ortalama* değerlerine genellikle 10^4 iterasyon civarında ulaşmışlardır.

Önerilen algoritma $P_{local} = 0$ hariç olmak üzere, diğer P_{local} değerleri için genellikle iyi sonuçlar elde etmiştir. Ancak $P_{local} = 0$ olarak seçildiğinde, önerilen algoritma oldukça kötü bir performans sergilemiştir. Sonuç olarak, Şekil 4.7'e göre LSM yapısı algoritmanın başarısını doğrudan etkilemektedir.

İkinci parametre analizi ise, popülasyon büyüklüğünü (N) belirlemek için yapılmıştır. Bu amaçla 10×10 'luk ızgara yapısı kullanılmıştır, $P_{local} = 0.005$ ve P parametresi 0.50 olarak belirlenmiştir. Popülasyon boyutu ise 10, 20, 30, 40, 50, 60, 70, 80, 90 ve 100 olacak şekilde deneylerde kullanılmıştır. N parametresinin belirlenmesi amacıyla yapılan deneysel analiz sonuçları Çizelge 4.14'te verilmiştir. Çizelge 4.14 incelendiğinde her N değeri için önerilen algoritma *en iyi* sonuçlara ulaşabilmiştir. Bu da göstermektedir ki rüzgâr türbini yerleştirme probleminin çözümü için popülasyon büyüklüğünün çözüm kalitesine etkisi çok azdır. Bununla birlikte $N = 60$ olarak seçildiğinde önerilen algoritmanın bulduğu *ortalama* sonuçlar ile *en kötü* sonuçlar diğer N değerlerine kıyasla biraz daha iyidir.

Çizelge 4.14. Farklı N değerleri için 10×10 'luk ızgara yapısı üzerinde elde edilen uygunluk değerlerinin karşılaştırılması

N	En iyi	En kötü	Ortalama	Std. Sap.
10	0.0015408	0.0015474	0.0015435	1.98E-06
20	0.0015408	0.0015474	0.0015424	1.79E-06
30	0.0015408	0.0015474	0.0015421	1.28E-06
40	0.0015408	0.0015442	0.0015416	9.69E-07
50	0.0015408	0.0015430	0.0015413	6.11E-07
60	0.0015408	0.0015421	0.0015410	3.98E-07
70	0.0015408	0.0015430	0.0015413	7.36E-07
80	0.0015408	0.0015430	0.0015414	5.63E-07
90	0.0015408	0.0015442	0.0015417	7.28E-07
100	0.0015408	0.0015457	0.0015422	1.15E-06

Son parametre analizi ise P parametresi için yapılmıştır. Bu analiz için, 10×10 'luk ızgara yapısı kullanılmıştır. Diğer kontrol parametreleri ise Çizelge 4.12, 4.13 ve 4.14'te verilen parametre analizlerine göre belirlenmiştir ve bu parametrelerin değerleri $N = 60$ ve $P_{local} = 0.005$ 'tir. P parametresinin belirlenmesi aşamasında, P parametresinin değeri 0.10 ile 1 arasında olacak şekilde ayarlanmıştır. P 'nin 1 olarak seçilmesi durumunda, temel Jaya algoritmasında olduğu gibi aday çözümün tüm boyutlarında güncelleme işlemi yapılmaktadır. Farklı P değerleri için önerilen algoritmanın bulduğu uygunluk değerleri Çizelge 4.15'te verilmiştir. Çizelge 4.15

incelendiğinde $P = 1$ değeri için önerilen algoritmanın bulduğu *en iyi*, *en kötü* ve *ortalama* uygunluk değerleri diğer P değerlerine kıyasla daha kötüdür. Sonuç olarak, aynı anda tüm boyutlarda güncelleme işleminin yapılması algoritmanın performansını düşürmüştür. Çizelge 4.15'te verilen analiz sonuçları değerlendirildiğinde $P = 0.50$ olarak seçildiğinde önerilen algoritmanın bulduğu uygunluk değerleri, diğer P değerlerine kıyasla daha iyidir.

Çizelge 4.15. Farklı P değerleri için 10×10 'luk ızgara yapısı üzerinde elde edilen uygunluk değerlerinin karşılaştırılması

P	En iyi	En kötü	Ortalama	Std. Sap.
$P = 0.10$	0.0015408	0.0015421	0.0015412	3.79E-07
$P = 0.20$	0.0015408	0.0015474	0.0015430	1.68E-06
$P = 0.30$	0.0015408	0.0015457	0.0015424	1.39E-06
$P = 0.40$	0.0015408	0.0015442	0.0015415	9.07E-07
$P = 0.50$	0.0015408	0.0015421	0.0015410	3.98E-07
$P = 0.60$	0.0015408	0.0015442	0.0015418	1.00E-06
$P = 0.70$	0.0015408	0.0015457	0.0015422	1.37E-06
$P = 0.80$	0.0015408	0.0015442	0.0015418	1.02E-06
$P = 0.90$	0.0015408	0.0015474	0.0015434	2.12E-06
$P = 1$	0.0015442	0.0016049	0.0015587	1.33E-05

4.3.2. Rüzgâr türbini yerleştirme probleminin çözümü için algoritmaların performans analizi

Önerilen JayaX-LSM algoritmasının etkinliğini göstermek amacıyla, önerilen algoritma yakın zamanda literatüre kazandırılmış olan algoritmaların deneysel sonuçları ile karşılaştırılmıştır. İlk deneysel karşılaştırma Pookpunt ve Ongsakul (2013), Beşkirli ve ark. (2018), Mosetti ve ark. (1994), Grady ve ark. (2005), González ve ark. (2010) ve Beşkirli ve ark. (2019) yapmış oldukları çalışmalar ile 10×10 'luk ızgara yapısı üzerinde yapılmıştır. Önerilen algoritma ile karşılaştırılan algoritmaların buldukları deneysel sonuçlar Çizelge 4.16'da verilmiştir. Çizelge 4.16'nın ilk sütunu karşılaştırmalarda kullanılan algoritmaları, ikinci sütunu algoritmaların bulduğu uygunluk değerlerini ($Cost/P_{total}$), üçüncü sütunu algoritmaların bulduğu toplam enerjiyi (P_{total}), dördüncü sütunu ızgara alanına yerleştirilen türbin sayısını (NT), beşinci sütunu türbin başına düşen ortalama enerjiyi (P_{total}/NT) ve son sütunu da rüzgâr türbini yerleştirme sonuçlarına göre algoritmaların verimliliğini göstermektedir. Çizelge 4.16'dan da görüldüğü üzere, önerilen algoritma 10×10 'luk ızgara yapısı için uygunluk değeri dikkate alındığında, diğer yaklaşımlardan daha iyi bir performans göstermiştir. Çizelge 4.16 incelendiğinde,

algoritmaların elde ettikleri uygunluk değerleri, toplam enerji, ortalama enerji ve verimlilik gibi kısıtlar dikkate alındığında önerilen algoritma; BPSO-TVAC (Pookpunt ve Ongsakul, 2013), GA (Mosetti ve ark., 1994), GA (Grady ve ark., 2005), EA (González ve ark., 2010) ve DGHS algoritmalarından daha iyi, NGHS algoritması ile ise aynı sonuçları elde etmiştir. Toplam enerji ve rüzgâr çiftliğinde bulunan türbin sayısına (NT) bağlı olarak hesaplanan verimliliğin yüksek olabilmesi için algoritmaların iyi bir yerleştirme işlemi gerçekleştirmesi gerekmektedir. Çizelge 4.16 incelendiğinde verimliliği en yüksek olan iki algoritma JayaX-LSM ve NGHS algoritmalarıdır. Bu algoritmalar 0.0015408’lik bir uygunluk değeri ile rüzgâr çiftliğine 30 tane türbin yerleştirmişlerdir. binAAA, BIWO ve DGHS algoritmaları ise yaklaşık olarak 0.0015414’lük bir uygunluk değeri ile rüzgâr çiftliğine 31 tane türbin yerleştirmişlerdir. Çizelge 4.16 incelendiğinde binAAA, BIWO ve DGHS algoritmalarının ürettiği toplam enerji önerilen algoritma ve NGHS algoritmasına kıyasla daha yüksektir. Bunun sebebi ise JayaX-LSM ve NGHS algoritmalarına kıyasla rüzgâr çiftliğine 1 tane fazla türbin yerleştirmeleridir. Ancak uygunluk değerleri dikkate alındığında binAAA, BIWO ve DGHS algoritmaları, türbin yerleştirme işlemi için daha fazla maliyet harcamışlardır ve türbin başına elde ettikleri ortalama enerji ve verimlilikleri JayaX-LSM ve NGHS algoritmalarından daha düşüktür. Sonuç olarak, iyi bir performans için toplam enerjinin yüksek olması tek başına yeterli değildir, aynı zamanda türbin başına düşen ortalama enerjinin ve verimliliğin yüksek olması gerekmektedir.

Çizelge 4.16. Algoritmaların 10×10 ’luk ızgara yapısı üzerinde buldukları sonuçların karşılaştırılması

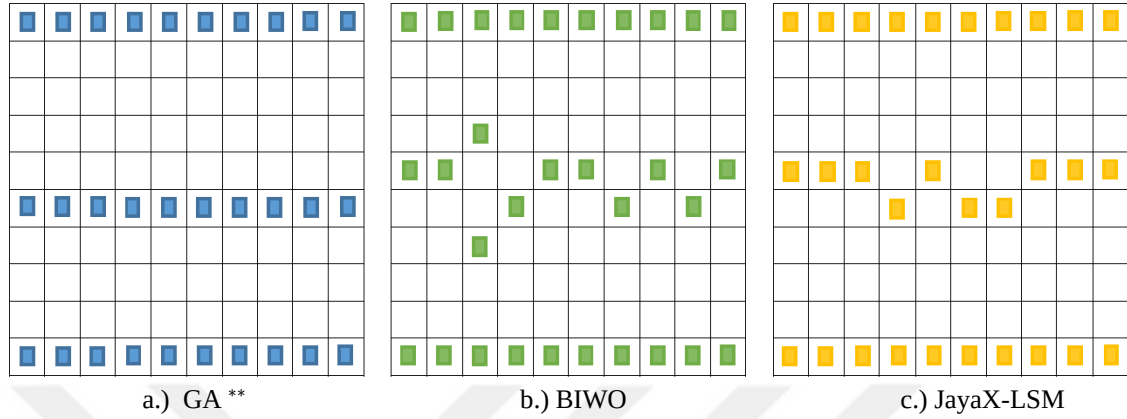
Yöntem	Uygunluk değeri ($Cost/P_{total}$)	Toplam Enerji (P_{total})	Türbin Sayısı (NT)	Ortalama Enerji (P_{total}/NT)	Verimlilik (%)
JayaX-LSM	0.0015408	14336	30	478	92.183
GA *	0.0016197	12352	26	475	91.643
GA **	0.0015436	14310	30	477	92.014
EA	0.0015436	14310	30	477	92.014
BPSO-TVAC	0.0015436	14310	30	477	92.014
binAAA	0.0015414	14667	31	473	91.267
BIWO	0.0015413	14667	31	473	91.267
NGHS	0.0015408	14336	30	478	92.183
DGHS	0.0015414	14667	31	473	91.267

* Mosetti ve ark. (1994) önermiş oldukları GA tabanlı yaklaşım

** Grady ve ark. (2005) önermiş oldukları GA tabanlı yaklaşım

Grady ve ark. (2005) önermiş oldukları GA tabanlı yaklaşım, Beşkirli ve ark. (2019) önermiş oldukları BIWO algoritması ve önerilen algoritmanın 10×10 ’luk bir

ızgara yapısı için elde ettikleri yerleştirme işleminin simülasyonu Şekil 4.8’de verilmiştir. Şekil 4.8’de verilen 10×10 ızgaranın her bir hücresinin uzunluğu 200 m ’dir ve ızgara üzerinde yerleştirilen kutucuklar (■) rüzgâr türbinlerini temsil etmektedir.



Şekil 4.8. 10×10 ’luk bir ızgara yapısı için algoritmaların elde ettikleri türbin yerleştirmeleri

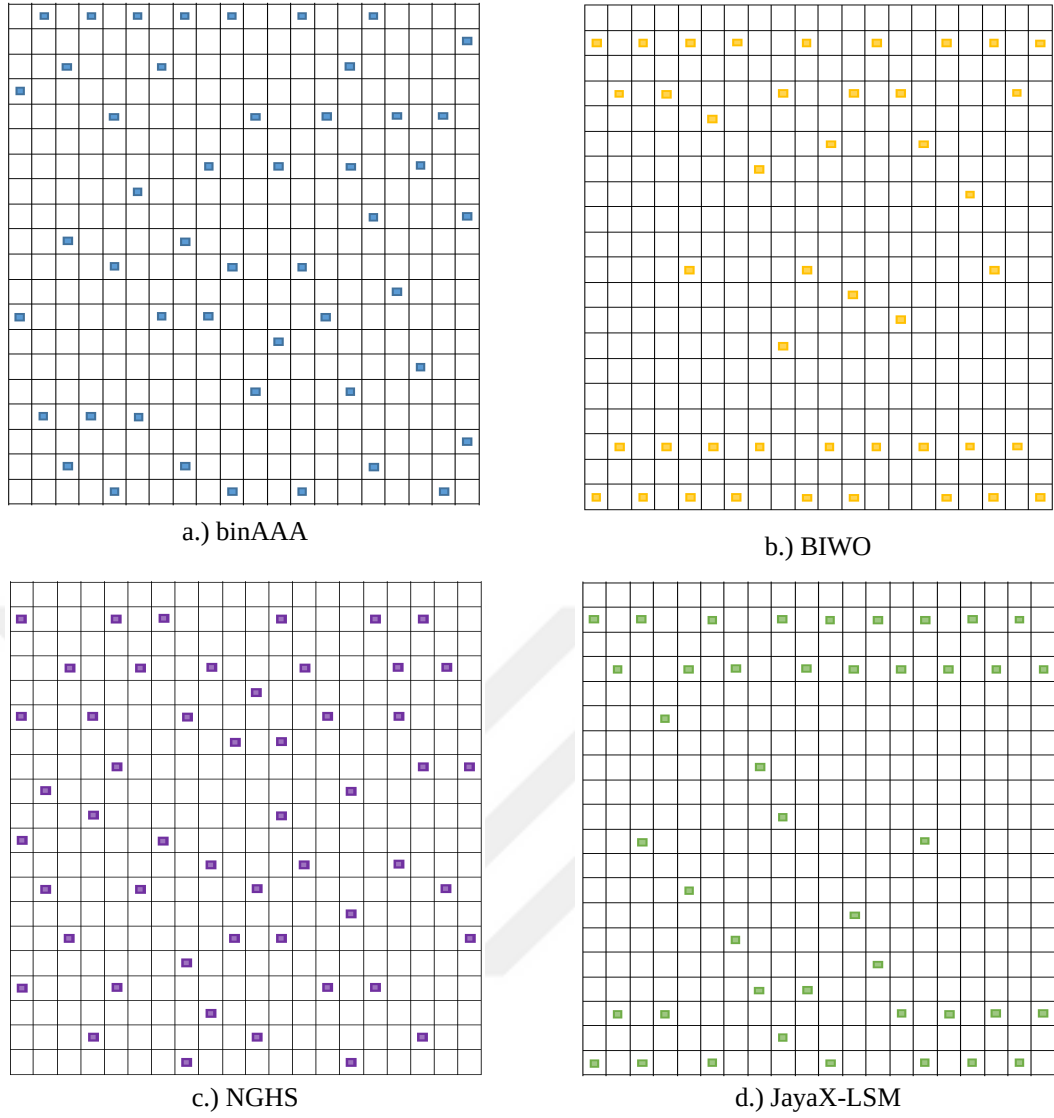
20×20 ’lik bir ızgara yapısı için algoritmaların elde ettikleri uygunluk değerleri, toplam enerji, ortalama enerji, verimlilik ve rüzgâr çiftliğine yerleştirilen türbin sayısı gibi deneysel sonuçlar Çizelge 4.17’de verilmiştir. 10×10 ’luk ızgara yapısına kıyasla, 20×20 ’lik ızgara yapısı üzerine literatürde yapılan çalışma sayısı azdır. Beşkirli ve ark. (2018) önermiş oldukları binAAA ve Beşkirli ve ark. (2019) önermiş oldukları BIWO algoritmaları 20×20 ’lik bir ızgara yapısını kullanarak rüzgâr türbini yerleştirme işlemini yapmışlardır. Yapılan deneysel karşılaştırmalarda bu iki algoritma kullanılmıştır. Bununla birlikte ikili optimizasyon problemlerinin çözümü için Zou ve ark.(2011) tarafından önerilen NGHS, Xiang ve ark. (2014) tarafından önerilen DGHS algoritmaları ile önerilen JayaX-LSM algoritması, 20×20 ’lik ızgara yapısı üzerine uygulanarak rüzgâr türbini yerleştirme işlemi yapılmıştır. 20×20 ’lik ızgara yapısı için 400 muhtemel türbine yerleştirme lokasyonu vardır ve bu ızgara türünde komşu iki türbin merkezi arasındaki mesafe 100 m ’dir. Uygun bir yerleştirmede uyandırma etkinin olumsuzluklarından kaçınmak için iki türbin arası mesafe en az 200 m olmalıdır. Sonuç olarak, aday çözüm oluşturulurken çakışmaların engellenmesi gereksiniminden dolayı 20×20 ’lik ızgara yapısı için uygun bir yerleştirme işlemi yapmak daha zordur. Çizelge 4.17’de JayaX-LSM, binAAA, BIWO, NGHS ve DGHS algoritmalarının 20×20 ’lik ızgara yapısı için buldukları sonuçlar verilmiştir. Uygunluk değerleri, toplam enerji, ortalama enerji ve verimlilik gibi kısıtlar dikkate alındığında önerilen JayaX-LSM algoritması diğer algoritmalara kıyasla oldukça iyi sonuçlar elde etmiştir. Çizelge 4.17

incelendiğinde binAAA, BIWO, NGHS ve DGHS algoritmalarının elde ettikleri toplam enerji sırasıyla, 23422 kW, 22382 kW, 23417 kW ve 23617 kW'dir. Önerilen algoritmanın bulduğu toplam enerji ise 21324 kW'dir. Deneysel sonuçlara göre binAAA, BIWO, NGHS ve DGHS algoritmaları, JayaX-LSM algoritmasına kıyasla daha iyi bir toplam enerji üretmiştir. Bunun sebebi bu algoritmaların, JayaX-LSM algoritmasına kıyasla rüzgâr çiftliğine daha fazla türbin yerleştirmeleridir. Ancak uygunluk değerleri dikkate alındığında binAAA, BIWO, NGHS ve DGHS algoritmaları, türbin yerleştirme işlemi için daha fazla maliyet harcamışlardır ve türbin başına elde ettikleri ortalama enerji ve verimlilikleri JayaX-LSM algoritmasından daha düşüktür. JayaX-LSM algoritmasına göre türbin başına düşen ortalama enerji 496 kW ve algoritmanın türbin yerleştirme verimliliği %95.661'dir.

Çizelge 4.17. Algoritmaların 10×10 'luk ızgara yapısı üzerinde buldukları sonuçların karşılaştırılması

Yöntem	Uygunluk değeri ($Cost/P_{total}$)	Toplam Enerji (P_{total})	Türbin Sayısı (NT)	Ortalama Enerji (P_{total}/NT)	Verimlilik (%)
JayaX-LSM	0.0013713	21324	43	496	95.661
binAAA	0.0014054	23422	49	478	92.207
BIWO	0.0014149	22382	47	476	91.862
NGHS	0.0014057	23417	49	478	92.187
DGHS	0.0014745	23617	52	454	87.611

Uygunluk değerleri dikkate alındığında, JayaX-LSM algoritması; binAAA, BIWO, NGHS ve DGHS algoritmalarına kıyasla sırasıyla %2.50, %3.18, %2.51 ve %7.53'lik daha iyi bir performans göstermiştir. Algoritmaların verimliliği dikkate alındığında ise, JayaX-LSM algoritması; binAAA, BIWO, NGHS ve DGHS algoritmalarına kıyasla sırasıyla %3.46, %3.80, %3.48 ve %8.05'lik daha iyi bir performans göstermiştir. Beşkirli ve ark. (2018) önermiş oldukları binAAA algoritması, Beşkirli ve ark. (2019) önermiş oldukları BIWO algoritması, NGHS algoritması ve önerilen JayaX-LSM algoritmasının 20×20 'lik bir ızgara yapısı için elde ettikleri yerleştirme işleminin simülasyonu Şekil 4.9'da verilmiştir. Şekil 4.9'da verilen 20×20 'lik ızgaranın komşu iki hücre merkezi arası mesafe 100 m'tir ve ızgara üzerinde yerleştirilen kutucuklar (■) rüzgâr türbinlerini temsil etmektedir.



Şekil 4.9. 20×20 'lik bir ızgara yapısı için algoritmaların elde ettikleri türbin yerleştirmeleri

4.4. Gezgin Satıcı Probleminin Çözümü için Deneysel Sonuçlar

Bu çalışmada, GSP'nin çözümü için önerilen Ayrık Jaya (DJaya) algoritması TSPLIB kütüphanesinden alınmış 14 farklı simetrik GSP üzerine uygulanmıştır (Reinelt, 1991). Deneylerde kullanılan gezgin satıcı problemleri ve bu problemlerin optimum değerleri ile ilgili detaylı bilgiler Çizelge 3.3'te verilmiştir. Deneylerde ve karşılaştırmalarda kullanılan GSP problemleri için Öklid mesafesi kullanılmıştır. Karşılaştırmalarda kullanılan tüm yöntemler 20 kez (run) çalıştırılmıştır ve deneysel sonuçlar *ortalama*, *standart sapma* (Std. Sap.) ve *göreceli hata* (relative error - RE) olarak sunulmuştur. Denklem 4.3'te verilen RE değeri, DJaya tarafından bulunan 20 çalışmanın

ortalama maliyeti ile optimum maliyet arasındaki farkı ifade etmektedir. RE değeri sıfıra yaklaştıkça, önerilen algoritmanın optimum sonuca yaklaştığı sonucuna varılmaktadır.

$$RE = \frac{f(ortalama) - f(optimum)}{f(optimum)} \times 100 \quad (4.3)$$

Denklem 4.3’de verilen, $f(optimum)$ değeri ilgili problem için optimum çözümü, $f(ortalama)$ ise ilgili problem için 20 bağımsız çalıştırmanın sonunda elde edilen ortalama sonucu ifade etmektedir. Önerilen DJaya’nın kontrol parametrelerinin değerlerine karar vermek amacıyla 225 şehirden oluşan *TSP225* problemi üzerinde bir parametre analizi gerçekleştirilmiştir. Parametre analizi aşamasında *TSP225* probleminin kullanılmasının sebebi, bu problemin büyük boyutlu bir problem olmasıdır. Böylece önerilen algoritmanın *TSP225* problemi gibi zor problemler için daha iyi sonuçlar vereceği düşünülmüştür. Gezgin satıcı probleminin çözümü için önerilen algoritmanın parametre analizi Bölüm 4.4.1’de verilmiştir.

4.4.1. GSP’nin çözümü için önerilen algoritmanın parametre analizi

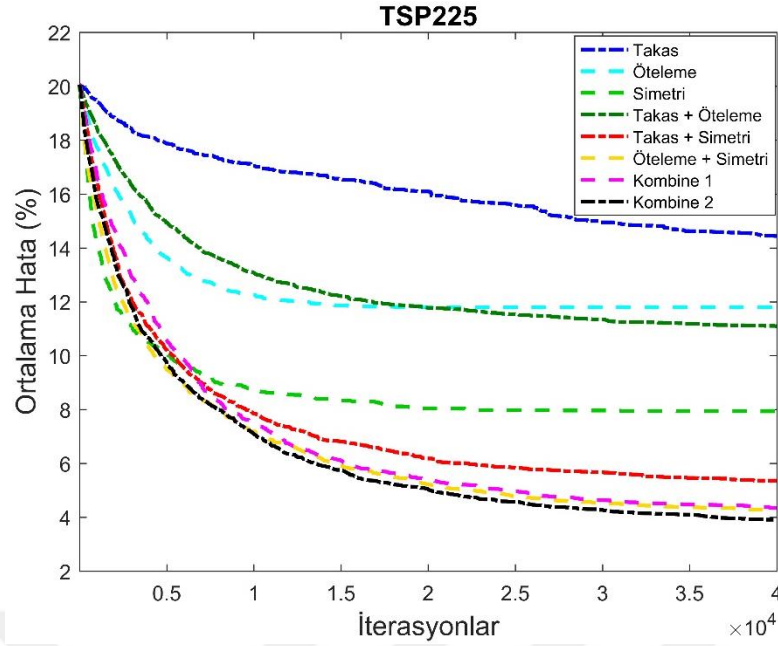
Bu çalışma kapsamında önerilen DJaya algoritması yakın zamanda literatüre kazandırılmış olan başarılı algoritmaların elde ettiği deneysel sonuçlar ile karşılaştırılmadan önce, kontrol parametrelerinin değerlerine karar vermek amacıyla bir parametre analizi gerçekleştirilmiştir. Parametre analizinde; komşuluk operatörlerinin sekiz farklı kombinasyonu olan; takas, öteleme, simetri, eşit seçilmeye dayalı *takas+öteleme* (*swap+shift*), eşit seçilmeye dayalı *takas+simetri* (*swap+symmetry*), eşit seçilmeye dayalı *öteleme+simetri* (*shift+symmetry*), Kombine 1 (takas, öteleme ve simetri operatörlerinin eşit seçimine dayalı) ve Kombine 2 (takas, öteleme ve simetri operatörleri rulet tekerleği seçimine göre belirlenir), sırasıyla konum güncelleme mekanizmasında kullanılmıştır. Popülasyon boyutu (10, 20, ... , 100) olacak şekilde, *ST1* ve *ST2* parametreleri de (0.1, 0.2, ... , 1) olacak şekilde analiz edilmiştir. Sonlandırma kriteri (MaxFEs) ise tüm analizlerde 8×10^5 olarak belirlenmiştir. İlk parametre analizi, komşuluk operatörleri üzerinde gerçekleştirilmiştir. Önerilen algoritmanın 8 farklı komşuluk operatörü için bulduğu *ortalama* sonuçlar Çizelge 4.18’de verilmiştir. Ayrıca 2-opt yerel arama algoritmasının, DJaya’nın performansına etkisini gözlemlemek

amacıyla elde edilen sonuçlar 2-opt olmaksızın ve 2-opt'lu olmak üzere ayrı ayrı verilmiştir.

Çizelge 4.18. TSP225 probleminin çözümünde komşuluk operatörlerinin performans analizi

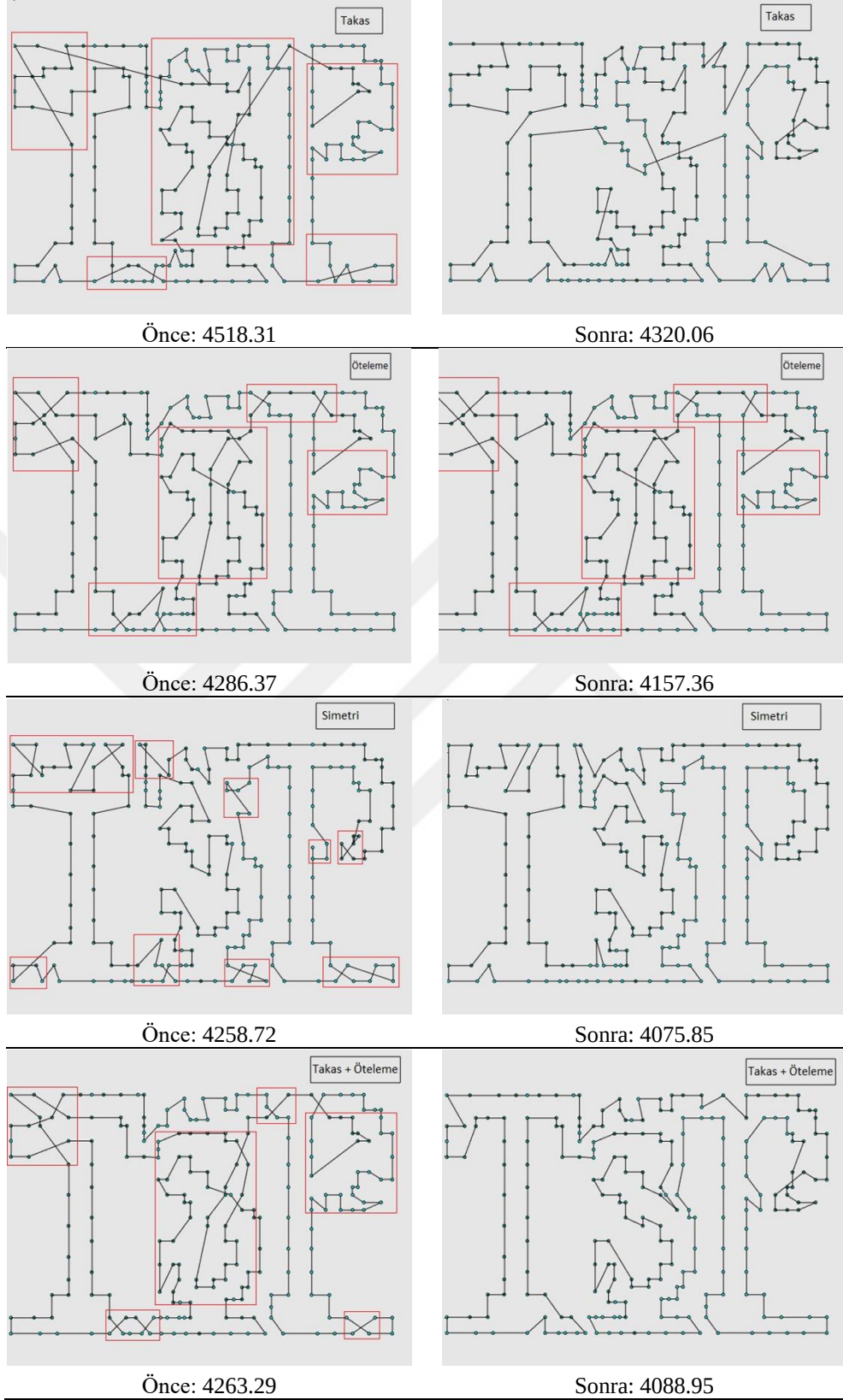
Operatör	DJaya (ortalama)	DJaya + 2-opt (ortalama)	Std. Sap.	RE (%)
Takas	4416.20	4253.34	29.24	10.22
Öteleme	4313.97	4127.88	36.87	6.97
Simetri	4165.15	4048.98	38.58	4.92
Takas + Öteleme	4287.21	4119.37	33.52	6.75
Takas + Simetri	4065.91	4045.20	33.44	4.83
Öteleme + Simetri	4024.10	4002.56	34.42	3.72
Kombine 1	4026.68	4014.73	28.31	4.04
Kombine 2	4009.96	3995.88	42.91	3.55

Çizelge 4.18 incelendiğinde en iyi sonuçların rulet tekerleği seçimini kullanan Kombine 2 komşuluk operatörü olduğu görülmektedir. En iyi ikinci komşuluk operatörü ise *Öteleme* + *Simetri* komşuluk operatörüdür. Ayrıca Çizelge 4.18'te de görüldüğü üzere 2-opt yerel arama algoritması, DJaya algoritmasının performansını arttırmıştır. TSP225 probleminin çözümünde komşuluk operatörlerinin performans analizinde en iyi sonuçları Kombine 2 mekanizması gösterdiği için diğer parametre analizlerinde ve deneysel çalışmalarda Kombine 2 mekanizması kullanılmıştır. Sekiz farklı komşuluk operatörü için DJaya algoritmasının yakınsama grafiği Şekil 4.10'da verilmiştir. Şekil 4.10 incelendiğinde TSP225 problemi için en hızlı yakınsamayı Kombine 2 operatörü göstermiştir. En iyi ikinci hızlı yakınsamayı ise *Öteleme* + *Simetri* operatörü göstermiştir. En kötü yakınsamayı ise takas operatörü göstermiştir.

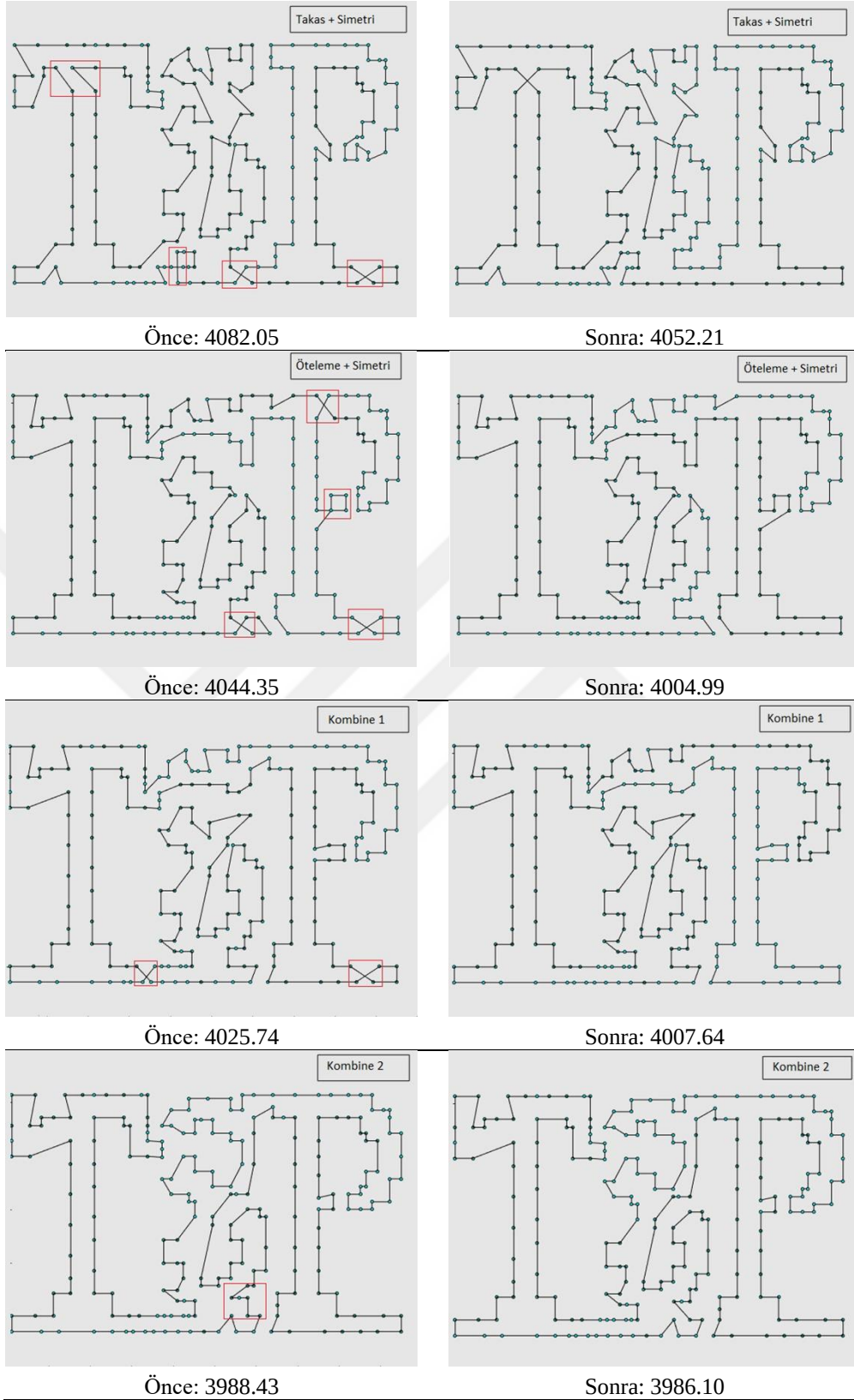


Şekil 4.10. Komşuluk operatörleri için ortalama hata için yakınsama grafiği

Şekil 4.11 ve Şekil 4.12’de verilen grafikte, 2-opt uygulanmadan önce ve 2-opt uygulandıktan sonra *TSP225* üzerinde önerilen algoritmanın 8 farklı komşuluk operatörü için bulduğu en kısa rota sonuçları verilmiştir. Şekil 4.11 ve Şekil 4.12’de verilen 2-opt uygulanmadan önceki rota durumunu göstermek için şekillerin üzerinde kırmızı çizgiler kullanılmıştır, bu kırmızı çizgiler 2-opt uygulandıktan sonra bu rotalarda güncellenme işlemi yapıldığı anlamına gelmektedir. Şekil 4.11 ve Şekil 4.12 incelendiğinde, *TSP225* için Kombine 2 komşuluk operatörü kullanılması durumunda, 2-opt yerel arama algoritması DJaya’nın bulmuş olduğu rotada çok ufak değişiklikler meydana getirmiştir. Çünkü 2-opt uygulanmadan önce, DJaya algoritmasının bulmuş olduğu rota uzunluğunu 3988.43 iken, 2-opt uygulandıktan sonra elde edilen rota uzunluğu 3986.10’dur. Ancak Şekil 4.11 ve Şekil 4.12’den de anlaşılacağı üzere diğer 7 komşuluk operatörü için 2-opt algoritması, tur uzunluğu fark edilir şekilde azaltmıştır. 2-opt yerel arama algoritması en kayda değer değişikliği ise güncelleme mekanizmasında takas operatörü kullanıldığı zaman göstermiştir. Çünkü 2-opt uygulanmadan önce, DJaya algoritmasının bulmuş olduğu rota uzunluğu takas operatörü için 4518.31 iken, 2-opt uygulandıktan sonra elde edilen rota uzunluğu 4320.06 olmuştur.



Şekil 4.11. Takas, Öteleme, Simetri ve *Takas + Öteleme* komşuluk operatörleri için 2-opt'tan önce ve 2-opt'tan sonra TSP225 problemi için tur uzunlukları



Şekil 4.12. *Takas + Simetri*, *Öteleme + Simetri*, *Kombine 1* ve *Kombine 2* komşuluk operatörleri için 2-opt'tan önce ve 2-opt'tan sonra TSP225 problemi için tur uzunlukları

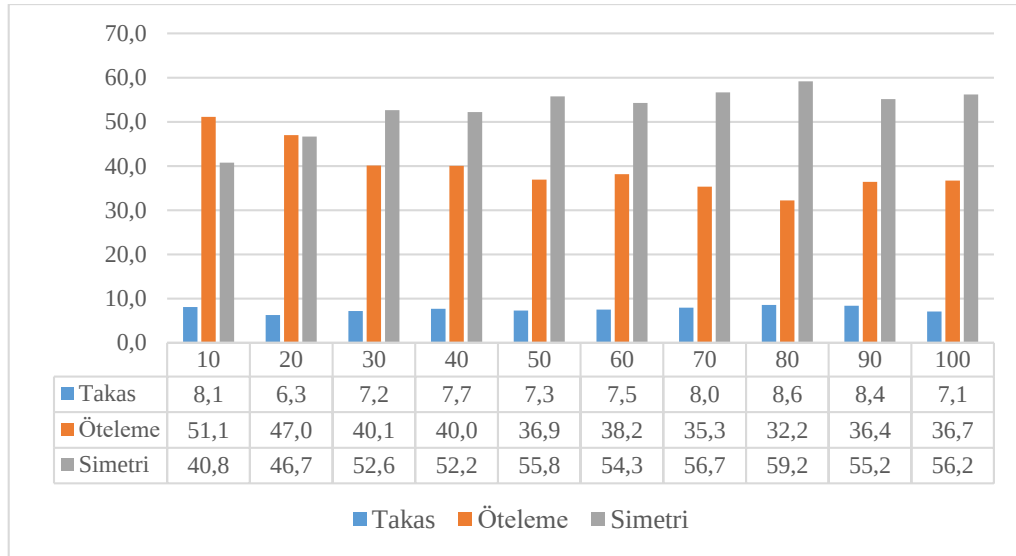
Sonuç olarak Şekil 4.11 ve Şekil 4.12’de de görüldüğü üzere, 2-opt algoritması takas, öteleme, simetri ve *takas + öteleme* komşuluk operatörleri için tur uzunluğunu fark edilir bir seviyede azaltmıştır. Bununla birlikte, Kombine 1, Kombine 2, *öteleme + simetri*, *takas + simetri* komşuluk operatörlerin bulduğu tur uzunlukları diğer komşuluk operatörlerine kıyasla daha iyi oldukları için 2-opt algoritması tur uzunluğunu çok az değiştirebilmiştir.

İkinci deneysel analiz ise, popülasyon boyutunun (N) belirlenmesi amacıyla gerçekleştirilmiştir. Bu amaçla, N , 10’dan 100’e, $ST1 = 0.5$, $ST2 = 0.5$ olarak belirlenmiştir ve konum güncelleme mekanizması için Kombine 2 komşuluk operatörü kullanılmıştır. Popülasyon boyutunun belirlenmesi amacıyla yapılan parametre analizi için elde edilen bulgular Çizelge 4.19’da verilmiştir.

Çizelge 4.19. Popülasyon boyutunun belirlenmesi amacıyla yapılan parametre analizi

N	Ortalama	Std. Sap.	RE (%)
10	4001.25	44.61	3.69
20	3995.88	42.91	3.55
30	4012.55	42.04	3.98
40	4014.62	42.47	4.03
50	4011.95	27.08	3.96
60	4028.86	38.37	4.40
70	4005.20	34.23	3.79
80	4027.70	36.79	4.37
90	4017.53	40.41	4.11
100	4019.96	30.16	4.17

Çizelge 4.19 incelendiğinde, $N = 20$ olarak seçildiğinde, DJaya algoritmasının bulduğu ortalama sonuçlar, diğer popülasyon boyutlarına kıyasla biraz daha iyidir. Bu nedenle $ST1$ ve $ST2$ parametrelerinin değerlerinin belirlenmesi aşaması için $N = 20$ olarak seçilmiştir. DJaya’da aday çözümün yeni konumu belirlenirken Kombine 2 komşuluk operatörü kullanılmaktadır. Kombine 2, takas, öteleme ve simetri operatörlerini rulet tekerleği seçimine göre kullanmaktadır. Her bir popülasyon boyutu için komşuluk operatörlerinin seçilme oranı Şekil 4.13’te verilmiştir.



Şekil. 4.13. Farklı N değerleri için rulet tekerleği seçimine göre komşuluk operatörlerinin seçilme oranı

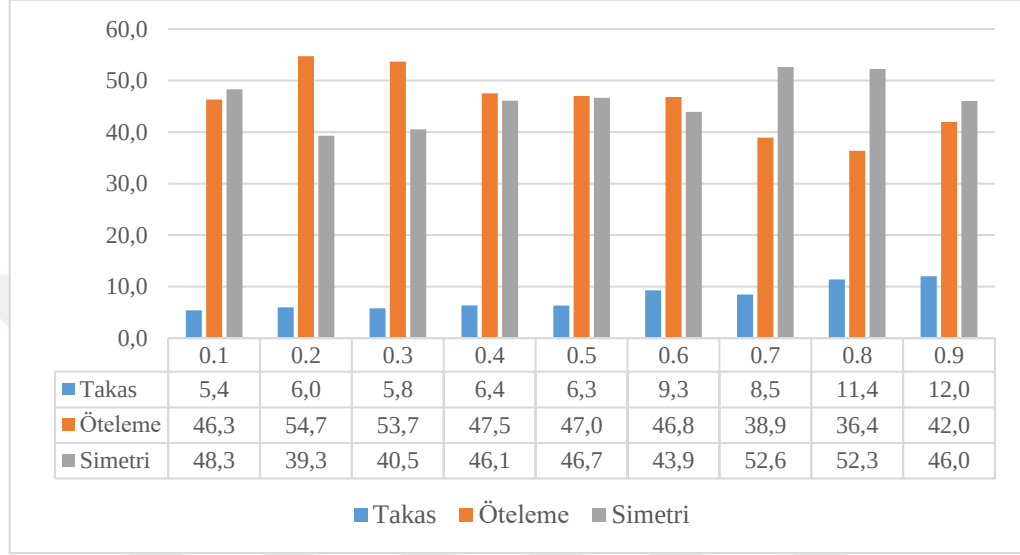
Şekil 4.13'ten de anlaşıldığı üzere farklı popülasyon boyutları için *simetri* operatörünün, aday çözüm oluşturma başarısı diğer komşuluk operatörlerine kıyasla daha iyidir. Aday çözümün oluşturulması aşamasında en az kullanılan komşuluk operatörü ise takas operatörü olmuştur. Çizelge 4.19'da verilen sonuçlar dikkate alındığında, önerilen algoritma $N = 20$ olarak seçildiğinde, diğer popülasyon boyutlarına kıyasla daha iyi sonuçlar elde etmiştir. Şekil 4.13'e göre $N = 20$ için, öteleme ve simetri operatörlerinin seçim yüzdeleri birbirine yakındır. Takas operatörünün seçilme yüzdesi ise %6.3'tür. Elde edilen deneysel sonuçlar dikkate alındığında, takas operatörünün aday çözüm oluşturma aşamasında genellikle iyi konumlar oluşturmadığı görülmektedir.

Çizelge 4.20. $ST1$ parametresinin belirlenmesi amacıyla yapılan parametre analizi

$ST1$	Ortalama	Std. Sap.	RE (%)
0.1	4013.95	31.63	4.02
0.2	4005.28	47.28	3.79
0.3	4008.81	33.25	3.88
0.4	4007.12	40.48	3.84
0.5	3995.88	42.91	3.55
0.6	4012.18	29.87	3.97
0.7	3996.47	38.73	3.56
0.8	4007.05	34.75	3.84
0.9	4014.88	45.75	4.04

Yapılan bir diğer parametre analizi ise $ST1$ parametresinin belirlenmesi amacıyla yapılmıştır ve elde edilen deneysel bulgular Çizelge 4.20'de verilmiştir. Deneysel

çalışmalarda $ST1$ parametresi (0.1, 0.2, ..., 0.9) olarak seçilmiştir. Diğer kontrol parametreleri popülasyon boyutu ve $ST2$ değeri sırasıyla 20 ve 0.5 olarak seçilmiştir. Çizelge 4.20’de de görüldüğü gibi, $ST1 = 0.5$ olarak seçildiğinde, önerilen algoritma diğer $ST1$ değerlerine kıyasla daha iyi sonuçlar elde etmiştir. Farklı $ST1$ değerleri için komşuluk operatörlerinin seçilme oranı Şekil 4.14’te verilmiştir.



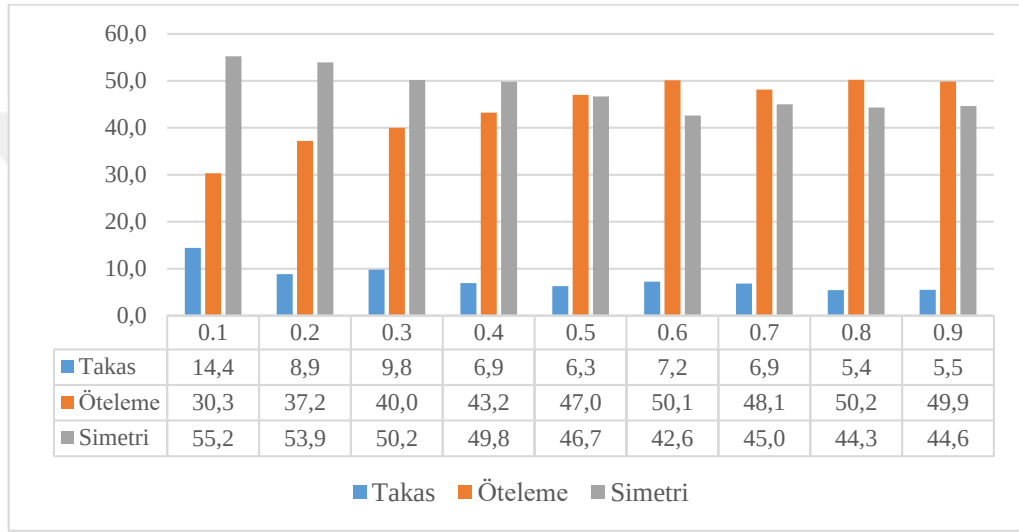
Şekil. 4.14. Farklı $ST1$ değerleri için rulet tekerleği seçimine göre komşuluk operatörlerinin seçilme oranı

Şekil 4.14’ten de anlaşıldığı üzere farklı $ST1$ değerleri için simetri ve öteleme operatörlerinin, aday çözüm oluşturma başarısı takas operatörüne kıyasla daha iyidir. Aday çözüm oluşturma aşamasında en az kullanılan komşuluk operatörü ise takas operatörü olmuştur.

Çizelge 4.21. $ST2$ parametresinin belirlenmesi amacıyla yapılan parametre analizi

$ST2$	Ortalama	Std. Sap.	RE (%)
0.1	4031.82	35.79	4.48
0.2	4011.32	33.97	3.95
0.3	4005.27	31.56	3.79
0.4	4004.29	38.54	3.77
0.5	3995.88	42.91	3.55
0.6	4012.13	29.31	3.97
0.7	4008.60	39.95	3.88
0.8	4014.67	55.42	4.03
0.9	4003.50	39.65	3.74

Yapılan son parametre analizi ise $ST2$ parametresinin belirlenmesi amacıyla yapılmıştır ve elde edilen deneysel sonuçlar Çizelge 4.21’de verilmiştir. Deneysel çalışmalarda $ST2$ parametresi (0.1, 0.2, ..., 0.9) olarak seçilmiştir. Diğer kontrol parametreleri ise Çizelge 4.19 ve Çizelge 4.20’ye göre belirlenmiş olup, $N = 20$ ve $ST1 = 0.5$ olarak seçilmiştir. Çizelge 4.21’de de görüldüğü üzere, $ST2 = 0.5$ olarak seçildiğinde, önerilen algoritma, diğer $ST2$ değerlerine kıyasla daha iyi sonuçlar elde etmiştir. Farklı $ST2$ değerleri için komşuluk operatörlerinin seçilme oranı Şekil 4.15’de verilmiştir.



Şekil. 4.15. Farklı $ST2$ değerleri için rulet tekerleği seçimine göre komşuluk operatörlerinin seçilme oranı

Şekil 4.15 incelendiğinde, farklı $ST2$ değerleri için simetri ve öteleme operatörlerinin, aday çözüm oluşturma başarısı takas operatörüne kıyasla daha iyidir.

4.4.2. GSP’nin çözümü için algoritmaların performans analizi

Önerilen DJaya algoritmasını etkinliğini ve başarısını göstermek için, önerilen algoritma yakın zamanda literatüre kazandırılmış olan algoritmaların deneysel sonuçları ile karşılaştırılmıştır. Adil bir karşılaştırma yapılması amacıyla, önerilen algoritma, karşılaştırmalarda kullanılan algoritmalar ile aynı şartlarda çalıştırılmıştır. Bu nedenle, her deneysel çalışma için yapılan analizler ayrı başlıklar altında sunulmuştur.

4.4.2.1. DJaya algoritmasının SA, ACO, STA ve DTSA ile karşılaştırılması

İlk deneysel karşılaştırma, aday çözüm oluşturma aşamasında komşuluk operatörlerini kullanan Chunhua ve ark. (2012) çalışması ve Cinar ve ark. (2019) çalışması ile yapılmıştır. Chunhua ve ark. (2012) yapmış oldukları çalışmada önerdikleri STA algoritmasını SA ve ACO algoritmaları ile karşılaştırmışlardır ve deneylerde *BERLIN52*, *ATT48* ve *ULYSSES16* problemlerini kullanmışlardır. Cinar ve ark (2019) çalışmalarında, *ATT48* ve *ULYSSES16* problemleri için koordinat sistemi hesaplamasının hatalı yapılmasından dolayı, Chunhua ve ark. (2012) çalışmalarında vermiş oldukları STA, SA ve ACO algoritmaları ile karşılaştırma yapmamışlardır. Dolayısıyla yapılan bu çalışmada, önerilen algoritma *BERLIN52* problemi üzerine uygulanmış ve elde edilen deneysel bulgular STA, SA, ACO ve DTSA algoritmaları ile karşılaştırılmıştır. STA, SA ve ACO'nun sonuçları doğrudan Chunhua ve ark. (2012)'nin çalışmasında, DTSA'nın sonuçları ise doğrudan Cinar ve ark. (2019) çalışmasından alınmıştır. Adil bir karşılaştırma yapılması amacıyla sonlandırma kriteri karşılaştırılan algoritmalar ile aynı olacak şekilde seçilmiş olup $MaxFES = 4000$ 'dir. Önerilen algoritma ile karşılaştırılan algoritmaların buldukları deneysel sonuçlar Çizelge 4.22'de verilmiştir.

Çizelge 4.22. DJaya algoritmasının ACO, SA, STA, DTSA ile karşılaştırılması

Problem	Yöntem	En iyi	En kötü	Ortalama	Std. Sap.
BERLIN52	SA	8186.40	9585.80	8983.80	380.10
	ACO	8240.40	9151.30	8777.60	267.11
	STA	7544.40	8630.50	8247.20	273.45
	DTSA	7542.00	7929.00	7689.17	108.40
	DJaya	7542.00	7831.00	7668.35	112.50

Çizelge 4.22 incelendiğinde DJaya algoritmasının *BERLIN52* problemi üzerinde STA, SA, ACO ve DTSA algoritmalarından daha iyi bir performansa sahip olduğu anlaşılmaktadır. DTSA ve DJaya algoritmalarının ikisi de *BERLIN52* problemi için *en iyi* sonuçlara ulaşmışlardır. Ancak *en kötü* ve *ortalama* sonuçlar dikkate alındığında DJaya algoritmasının, DTSA algoritmasına kıyasla daha iyi sonuçlara ulaştığı söylenebilir.

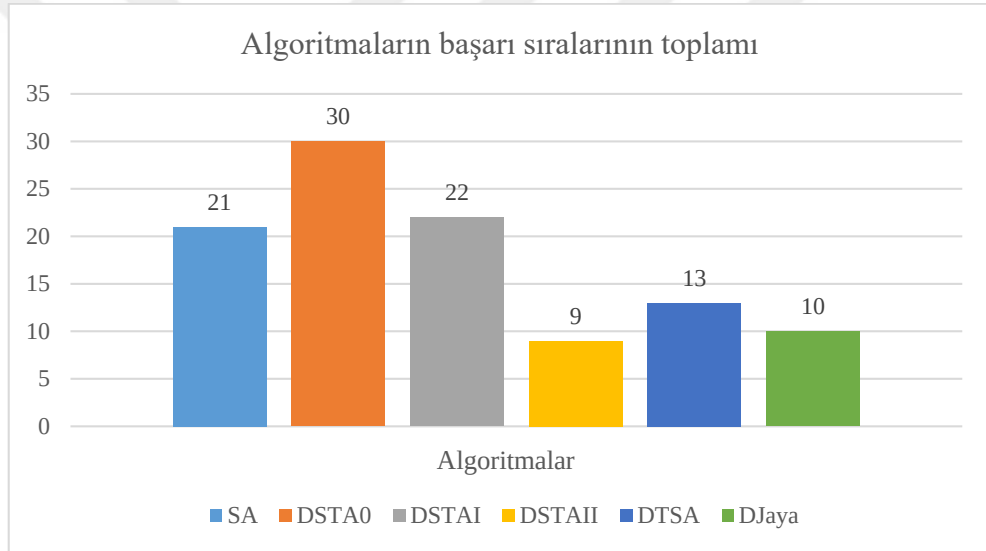
4.4.2.2. DJaya algoritmasının SA, DSTA0, DSTAI, DSTAI ve DTSA algoritmalarının deneysel sonuçları ile karşılaştırılması

Diğer bir deneysel karşılaştırma, Zhou ve ark. (2016) önermiş oldukları DSTA0, DSTAI ve DSTAI algoritmaları ve Cinar ve ark. (2019) önermiş oldukları DTSA algoritması ile yapılmıştır. Zhou ve ark. (2016) yapmış oldukları çalışmada önerdikleri algoritmaları *KROA100*, *KROB100*, *KROC100*, *KROD100* ve *KROE100* gezgin satıcı problemlerinin üzerine uygulamışlardır ve deneysel sonuçlarını SA algoritmasının sonuçları ile karşılaştırmışlardır. SA, DSTA0, DSTAI ve DSTAI algoritmalarının sonuçları doğrudan Zhou ve ark. (2016) önermiş oldukları çalışmadan alınmıştır, DTSA'nın sonuçları ise Cinar ve ark. (2019) önermiş oldukları çalışmadan alınmıştır. Adil bir karşılaştırma yapılabilmesi için sonlandırma kriteri karşılaştırılan algoritmalar ile aynı olacak şekilde seçilmiş olup $MaxFES = 9 \times 10^4$ 'tür. Önerilen algoritma ve karşılaştırılan algoritmaların buldukları deneysel sonuçlar Çizelge 4.23'de verilmiştir.

Çizelge 4.23. DJaya algoritmasının SA, DSTA0, DSTAI, DSTAI ve DTSA ile karşılaştırılması

Problem	Yöntem	Ortalama	Std. Sap.	RE (%)	Sıra
KROA100	SA	22635.00	778.72	6.36	4
	DSTA0	23213.00	906.11	9.07	6
	DSTAI	22835.00	715.85	7.30	5
	DSTAI	21767.00	221.64	2.28	3
	DTSA	21506.78	260.55	1.06	1
	DJaya	21705.98	290.22	1.99	2
KROB100	SA	23657.00	445.78	6.85	4
	DSTA0	23794.00	517.05	7.47	6
	DSTAI	23734.00	507.38	7.19	5
	DSTAI	22880.00	302.14	3.34	1
	DTSA	23139.26	181.74	4.51	3
	DJaya	22973.73	234.79	3.76	2
KROC100	SA	22223.00	522.20	7.10	5
	DSTA0	22877.00	709.87	10.26	6
	DSTAI	21891.00	536.88	5.50	4
	DSTAI	21378.00	246.34	3.03	1
	DTSA	21817.08	217.77	5.15	3
	DJaya	21702.02	186.32	4.59	2
KROD100	SA	22911.00	483.01	7.59	4
	DSTA0	23043.00	565.80	8.21	6
	DSTAI	22665.00	592.53	6.44	3
	DSTAI	21991.00	315.32	3.27	1
	DTSA	22972.26	390.50	7.88	5
	DJaya	22631.25	487.62	6.28	2
KROE100	SA	23125.00	389.42	4.79	4
	DSTA0	23738.00	450.82	7.57	6
	DSTAI	23371.00	678.69	5.90	5
	DSTAI	22637.00	166.82	2.58	3
	DTSA	22547.00	121.96	2.17	1
	DJaya	22582.47	252.07	2.33	2

Çizelge 4.23 incelendiğinde, *KROA100* ve *KROE100* problemlerinin çözümü için en iyi çözücü algoritma DTSA olmasına rağmen DJaya algoritmasının bulunduğu sonuçlarda DTSA algoritmasına oldukça yakındır. *KROB100*, *KROC100* ve *KROD100* problemleri için ise en iyi sonuçlara DSTAII algoritması ulaşmıştır. Ancak bu problemler için en iyi ikinci çözücü algoritma DJaya algoritması olmuştur. Çizelge 4.23'te verilen deneysel sonuçların daha iyi analiz edilebilmesi amacıyla bu sonuçlara *sıra* mekanizması uygulanmıştır. *Sıra* mekanizmasına göre *KROA100*, *KROB100*, *KROC100*, *KROD100* ve *KROE100* problemleri için algoritmaların çözüm sıralaması Çizelge 4.23'te verilmiştir. Şekil 4.16'da ise *sıra* mekanizmasına göre algoritmaların başarı sıralarının toplamı verilmiştir. Böylece genel olarak hangi algoritmanın daha güvenilir sonuçlar elde ettiği daha iyi anlaşılmaktadır.

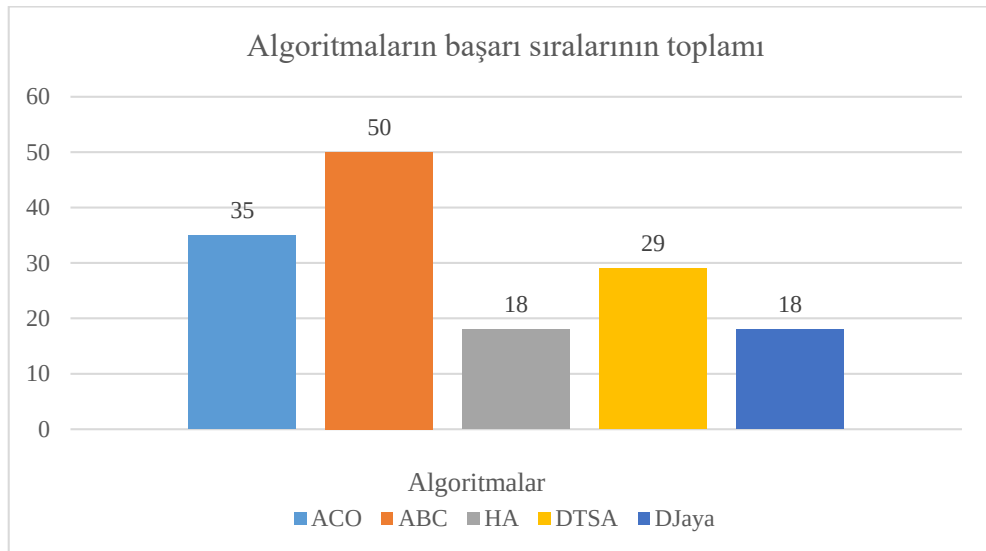


Şekil 4.16. SA, DSTA versiyonları, DTSA ve DJaya algoritmalarının *sıra* mekanizmasına göre başarı sıralarının toplamı

Şekil 4.16'da da görüldüğü üzere, DSTAII ve DJaya algoritmalarının genel olarak, karşılaştırılan diğer algoritmalara kıyasla daha iyi ve daha etkili çözümler elde ettiği söylenebilir. DSTAII'nin toplam sıra değeri 9 ve DJaya'nın toplam sıra değeri ise 10'dur. Sonuç olarak *KRO** problemlerinin çözümü için en iyi çözücü algoritma DSTAII algoritması ve en iyi ikinci çözücü algoritma ise DJaya algoritması olmuştur.

4.4.2.3. DJaya algoritmasının ACO, ABC, HA ve DTSA algoritmalarının deneysel sonuçları ile karşılaştırılması

Diğer bir deneysel karşılaştırma ise Gündüz ve ark. (2015) önerdikleri çalışma ve Cinar ve ark. (2019) önerdikleri çalışma ile yapılmıştır. Gündüz ve ark. (2015) GSP'nin çözümü için ACO ve ABC algoritmalarına dayalı HA algoritmasını önermişlerdir ve önerdikleri HA algoritmasını 10 farklı GSP problemi olan *EIL51*, *EIL76*, *EIL101*, *KROA100*, *OLIVER30*, *BERLIN52*, *ST70*, *PR76*, *CH150* ve *TSP225* problemleri üzerine uygulamışlardır ve deneysel sonuçlarını ACO ve ABC algoritmaları ile karşılaştırmışlardır. Yapılan bu çalışmada DJaya algoritması, ABC, ACO, HA ve DTSA algoritmaları ile kıyaslanmıştır. ABC, ACO ve HA algoritmalarının sonuçları doğrudan Gündüz ve ark. (2015) çalışmasından alınmıştır. DTSA'nın sonuçları ise doğrudan Cinar ve ark. (2019) çalışmasından alınmıştır. Adil bir karşılaştırma yapılabilmesi için sonlandırma kriteri karşılaştırılan algoritmalar ile aynı olacak şekilde seçilmiş olup $MaxFES = D \times 500$ 'tür. D ilgili problemdeki şehir sayısına bağlı olarak değişmektedir. Önerilen algoritma ile karşılaştırılan algoritmaların buldukları deneysel sonuçlar Çizelge 4.24'de verilmiştir. Ayrıca algoritmaların etkinliğini göstermek amacıyla, bu algoritmaların bulduğu sonuçların üzerine *sıra* mekanizması uygulanmıştır ve algoritmaların toplam başarı sıraları Şekil 4.17'de verilmiştir.



Şekil 4.17. ACO, ABC, HA, DTSA ve DJaya algoritmalarının *sıra* mekanizmasına göre başarı sıralarının toplamı

Çizelge 4.24. DJaya algoritmasının ABC, ACO, HA ve DTSA ile karşılaştırılması

Problem	Yöntem	Ortalama	Std. Sap.	RE (%)	Sıra
OLIVER30	ACO	424.68	1.41	0.22	2
	ABC	462.55	12.47	9.16	5
	HA	423.74	0.00	0.00	1
	DTSA	428.50	4.21	1.12	4
	DJaya	426.88	2.74	0.74	3
EIL51	ACO	457.86	4.07	6.76	4
	ABC	590.49	15.79	37.69	5
	HA	443.39	5.25	3.39	2
	DTSA	443.93	4.04	3.51	3
	DJaya	440.18	4.95	2.64	1
BERLIN52	ACO	7659.31	38.70	1.52	4
	ABC	10390.26	439.69	37.72	5
	HA	7544.37	0.00	0.00	1
	DTSA	7545.83	21.00	0.02	2
	DJaya	7580.30	80.60	0.48	3
ST70	ACO	709.16	8.27	4.73	4
	ABC	1230.49	41.79	81.73	5
	HA	700.58	7.51	3.47	1
	DTSA	708.65	6.77	4.66	3
	DJaya	702.30	9.56	3.72	2
EIL76	ACO	561.98	3.50	3.04	2
	ABC	931.44	24.86	70.78	5
	HA	557.98	4.10	2.31	1
	DTSA	578.58	3.93	6.09	4
	DJaya	573.17	6.33	5.10	3
PR76	ACO	116321.22	885.79	7.55	4
	ABC	205119.61	7379.16	89.65	5
	HA	115072.29	742.90	6.39	3
	DTSA	114930.03	1545.64	6.26	2
	DJaya	113258.29	1711.93	4.71	1
KROA100	ACO	22880.12	235.18	7.49	4
	ABC	53840.03	2198.36	152.94	5
	HA	22435.31	231.34	5.40	3
	DTSA	21728.40	358.13	2.08	1
	DJaya	21735.31	331.33	2.13	2
EIL101	ACO	693.42	6.80	7.96	4
	ABC	1315.95	35.28	104.88	5
	HA	683.39	6.56	6.40	2
	DTSA	689.91	4.47	7.41	3
	DJaya	677.37	4.87	5.46	1
CH150	ACO	6702.87	20.73	2.61	4
	ABC	21617.48	453.71	230.93	5
	HA	6677.12	19.30	2.22	2
	DTSA	6748.99	32.63	3.32	3
	DJaya	6638.63	52.79	1.63	1
TSP225	ACO	4176.08	28.34	8.22	3
	ABC	17955.12	387.35	365.28	5
	HA	4157.85	26.27	7.74	2
	DTSA	4230.45	58.76	9.93	4
	DJaya	4095.02	42.54	6.12	1

Çizelge 4.24'te de görüldüğü gibi DJaya algoritması; *EIL51*, *EIL101*, *PR76*, *TSP225* ve *CH150* problemlerinin çözümü için en iyi birinci çözücü algoritma olmuştur. HA algoritması ise *EIL76*, *OLIVER30*, *BERLIN52*, *ST70* problemleri için diğer algoritmalarla kıyasla en kısa rotayı bulmuştur. Geriye kalan *KROA100* probleminin

çözümü için ise en iyi rotayı DTSA algoritması bulmuştur. Bu sonuçlar ışığında DJaya'nın ACO, ABC ve DTSA algoritmalarına kıyasla daha iyi bir performans gösterdiği anlaşılmaktadır. Çizelge 4.24'e göre karşılaştırmalarda kullanılan algoritmaların elde ettiği göreceli hatalar dikkate alındığında, en başarısız sonuçlara ulaşan algoritma ABC algoritması olmuştur. Şekil 4.17'deki sıra mekanizmasına göre elde edilen toplam sıra değerleri dikkate alındığında, HA ve DJaya algoritmalarının toplam sıra değerleri birbirine eşittir. Hem HA hem de DJaya algoritmaları, diğer algoritmalarla kıyasla etkili performanslar gösterilmişlerdir. Bununla birlikte, deneylerde kullanılan toplam problem sayısı dikkate alınırsa DJaya algoritması, HA algoritmasına göre biraz daha iyi bir performans göstermiştir. Bununla birlikte DJaya ve HA'nın elde ettiği başarıya en yakın diğer çözücü algoritma ise DTSA algoritması olmuştur. Şekil 4.17'deki toplam sıra değerleri dikkate alındığında en kötü sonuçlara ulaşan algoritma ise ABC algoritması olmuştur.

4.4.2.4. DJaya algoritmasının ACO, PSO, GA, BH ve DTSA algoritmalarının deneysel sonuçları ile karşılaştırılması

Bu bölümde yapılan son deneysel karşılaştırma ise Hatamlou (2018) çalışması ve Cinar ve ark. (2019) yapmış oldukları çalışma ile yapılmıştır. Hatamlou (2018), GSP'yi çözmek için kara delik algoritmasına (BH) dayalı bir yöntem sunmuştur ve deneylerde *ULYSSES22*, *BAYG29*, *ATT48*, *GR96*, *EIL51*, *EIL76*, *EIL101*, *BERLIN52* ve *ST70* olmak üzere 9 farklı problem kullanmıştır. Cinar ve ark. (2019), yaptıkları çalışmada *ULYSSES22*, *BAYG29*, *ATT48* ve *GR96* problemlerini farklı koordinat sistemi hesaplaması nedeniyle kullanmamışlardır. Bu nedenle yapılan bu çalışmada da Cinar ve ark. (2019)'nın yapmış oldukları çalışma da olduğu gibi *EIL51*, *EIL76*, *EIL101*, *BERLIN52* ve *ST70* problemleri deneylerde kullanılmış olup, elde edilen sonuçlar BH, ACO, PSO, GA ve DTSA algoritmaları ile karşılaştırılmıştır. BH, ACO, PSO ve GA algoritmalarının sonuçları doğrudan Hatamlou (2018) çalışmasından alınmıştır, DTSA'nın sonuçları ise doğrudan Cinar ve ark. (2019) çalışmasından alınmıştır. Adil bir karşılaştırma yapılabilmesi için sonlandırma kriteri karşılaştırılan algoritmalar ile aynı olacak şekilde seçilmiş olup $MaxFES = 2 \times 10^4$, popülasyon boyutu $N = 100$ ve toplam çalıştırma sayısı her bir problem için 5 bağımsız çalıştırma olarak belirlenmiştir.

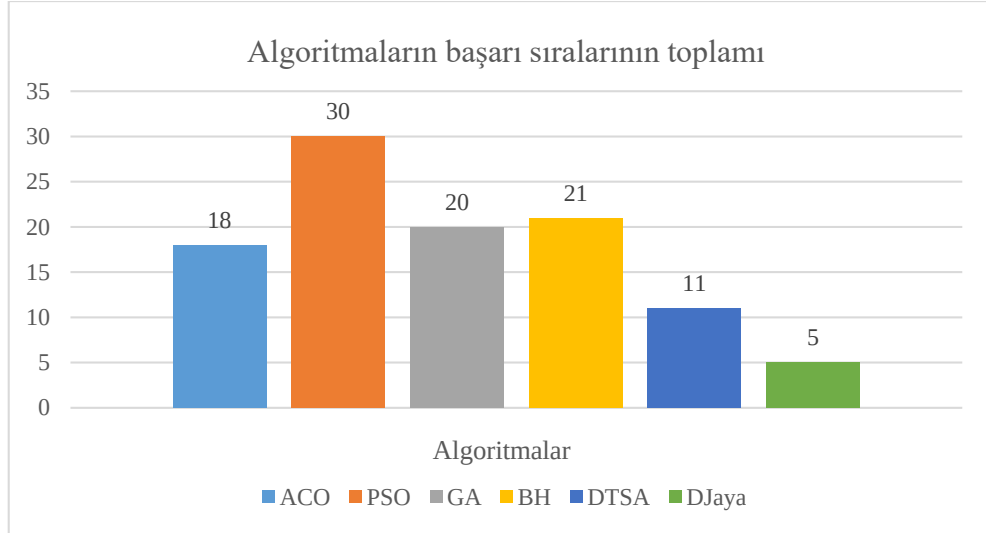
BH, ACO, PSO, GA, DTSA ve DJaya algoritmaları için deneysel sonuçlar Çizelge 4.25'te verilmiştir. Çizelge 4.25'te de görüldüğü gibi, DJaya algoritması

deneylerde kullanılan her 5 problem için de etkili ve iyi sonuçlar elde etmiştir. En iyi çözücü ikinci algoritma ise DTSA algoritması olmuştur. Çizelge 4.25'e göre karşılaştırmalarda kullanılan algoritmaların elde ettiği ortalama sonuçlar dikkate alındığında, en kötü sonuçlara ulaşan algoritma PSO algoritmasıdır. Algoritmaların elde ettiği ortalama sonuçlar dikkate alındığında, çözüm kalitesi ve sağlamlık açısından DJaya algoritması ACO, PSO, GA, BH ve DTSA algoritmalarından daha iyi bir performans göstermiştir. Bununla birlikte DTSA algoritması da genel olarak ACO, PSO, GA ve BH algoritmalarından daha iyi bir performans göstermiştir.

Çizelge 4.25. DJaya algoritmasının ACO, PSO, GA, BH ve DTSA ile karşılaştırılması

Problem	Yöntem	Ortalama	Std. Sap.	Sıra
EIL51	ACO	461.0175	6.2974	5
	PSO	574.8022	107.2371	6
	GA	453.4773	9.4157	2
	BH	458.9252	38.6365	4
	DTSA	456.5184	8.9247	3
	DJaya	440.4394	3.1055	1
EIL76	ACO	594.1442	40.2152	3
	PSO	975.6397	152.4061	6
	GA	652.0593	122.0972	4
	BH	659.1021	152.1754	5
	DTSA	588.0623	5.7296	2
	DJaya	574.4803	5.6710	1
EIL101	ACO	763.9207	59.9684	3
	PSO	1499.9911	319.7468	6
	GA	838.8307	9.9642	4
	BH	897.3813	210.1446	5
	DTSA	689.8384	7.2994	2
	DJaya	686.8843	6.0664	1
BERLIN52	ACO	8522.9017	1152.2000	4
	PSO	11089.5286	2067.9323	6
	GA	9288.4483	1301.2108	5
	BH	8455.8304	508.9871	3
	DTSA	7761.6000	62.8594	2
	DJaya	7627.0000	120.3869	1
ST70	ACO	757.7540	59.6079	3
	PSO	1321.8137	269.2793	6
	GA	1158.8458	52.1734	5
	BH	797.5745	125.2272	4
	DTSA	710.4037	2.7956	2
	DJaya	707.2151	15.3049	1

Şekil 4.18'de deneysel karşılaştırmalarda kullanılan optimizasyon algoritmalarının *sıra* mekanizmasına göre elde ettiği başarı sıralarının toplamı verilmiştir.



Şekil 4.18. ACO, PSO, GA, BH, DTSA ve DJaya algoritmalarının *sıra* mekanizmasına göre başarı sıralarının toplamı

Şekil 4.18’de verilmiş olan toplam sıra değerleri dikkate alındığında, DJaya algoritmasının toplam sıra değeri 5’tir ve DJaya algoritması, karşılaştırmalarda kullanılan diğer algoritmalara kıyasla daha etkili bir performans gösterilmiştir. Karşılaştırmalarda kullanılan diğer algoritmaların sonuçları dikkate alındığında en iyi ikinci çözücü algoritma DTSA algoritması olmuştur ve DTSA algoritmasının sıra değeri 11’dir. Şekil 4.18’e göre karşılaştırmalarda kullanılan algoritmalar arasından en başarısız algoritma ise toplam sıra değeri 30 olan PSO algoritması olmuştur.

5. SONUÇLAR VE ÖNERİLER

5.1 Sonuçlar

Optimizasyon algoritmalarının amacı, herhangi bir optimizasyon problemi için bilinen verileri kullanarak arama uzayında (tüm çözümler uzayı), belirli şartlar altında mümkün olan en iyi çözüme ulaşmaktır. Optimizasyon problemlerinin çözümünde kullanılan metasezgisel algoritmalar üzerinde çalışılan problem için kesin çözüme ulaşmayı garanti etmezler. Bununla birlikte makul bir sürede arama uzayındaki kesin çözüme yakın bir sonuca ulaşmayı hedeflemektedirler. Yakın zamanda literatüre kazandırılmış birçok farklı optimizasyon algoritması vardır. Jaya algoritması, kısıtlı ve kısıtsız sürekli optimizasyon problemlerinin çözümü için Rao (2016) tarafından literatüre kazandırılan popülasyon tabanlı, stokastik bir metasezgisel algoritmadır. Jaya algoritmasının en önemli özelliklerinden biri parametre bağımsız olmasıdır. Algoritmanın parametre bağımsız olması özelliği sayesinde optimizasyon problemlerine kolay bir şekilde uyarlanabilmektedir. Jaya, algoritmasının bir diğer önemli özelliği ise konum güncelleme mekanizmasında en iyi uygunluk değerine sahip çözümü ve en kötü uygunluk değerine sahip çözümü kullanmasıdır. Bu sayede, mevcut çözümün en iyi çözüme doğru hareket etme eğilimi artırılırken, aynı zamanda en kötü çözümden uzaklaşarak, yerel minimum ya da yerel maksimum çözüme takılmayı önlemektedir. Bu tez kapsamında ikili ve ayrık tam sayı optimizasyon problemlerinin çözümü için nispeten yeni bir metasezgisel algoritma olan Jaya algoritması tabanlı yeni yaklaşımlar geliştirilmiştir.

Temel Jaya algoritması sürekli optimizasyon problemlerinin çözümü için geliştirildiğinden dolayı, karar değişkenleri '0' ya da '1' değerlerini alabilen ikili optimizasyon problemlerinin üzerine doğrudan uygulanamaz. Bu nedenle temel Jaya algoritmasının konum güncelleme mekanizmasında bazı değişiklikler yapılmıştır. Bu kapsamda geliştirilen ilk yaklaşım JayaX olarak adlandırılan; temelini Jaya algoritması ve 'özel veya' (XOR) lojik fonksiyonundan alan yaklaşımdır. Konum güncelleme mekanizması için 'XOR' kapısı seçilmesinin en önemli sebebi, 'XOR' için karar değişkenlerinin '0' veya '1' değerlerini alma olasılığının birbirine eşit olmasıdır. Temel Jaya algoritmasında yapılan bir diğer yenilik ise konum güncelleme aşamasında her boyutta güncelleme işleminin yapılmamasıdır. Temel Jaya algoritmasında konum güncelleme aşamasında X_k çözümünün her boyutunda güncelleme işlemi yapılmaktadır. Bu da rastsalılığı arttırdığı için arama uzayında büyük bir pertürbasyona sebep olmakta ve

bazen en iyi çözümden uzaklaşmaya neden olabilmektedir. Temel Jaya algoritmasının bu eksikliğin üstesinden gelebilmek için algoritmaya yeni bir mekanizma eklenmiştir ve bu mekanizmanın içine olasılık (Probability - P) parametresi eklenmiştir. P parametresi $[0,1]$ arasında olacak şekilde bir değer almaktadır. Önerilen algoritmaya eklenen bu mekanizma sayesinde problemin boyutuna bağlı olarak ($d = D * P$), d kadar boyut her bir yinelemede P parametresine göre belirlenmekte ve güncelleme işlemi yapılmaktadır. Sonuç olarak önerilen algoritmaya eklenen bu mekanizma ile arama uzayında çözüm araştırması sırasında yerel arama ile global arama arasında bir denge sağlanmaya çalışılmıştır. Geliştirilen diğer yaklaşım ise JayaX algoritmasının yerel arama yönünü iyileştirmek için JayaX-LSM olarak adlandırılan, geliştirilen ilk algoritma olan JayaX algoritmasını ve LSM olarak adlandırılan yerel arama modülünün birlikte kullanılması ile geliştirilen yaklaşımdır. LSM algoritması, basit bir komşuluk arama yaklaşımıdır. LSM mevcut bir çözümle başlar ve mevcut çözümün yakınlarında mevcut çözümden daha iyi bir çözüm olup olmadığını araştırmaktadır. Önerilen algoritmada yerel arama yapılıp yapılmayacağı ise P_{local} parametresine göre belirlenmektedir. Yani önerilen algoritmada her yinelemede yerel arama işlemi yapılmamaktadır. Böylece, P_{local} parametresi sayesinde algoritmanın yerel minimum çözüme takılması engellenmeye çalışılmaktadır. İkili optimizasyon problemlerinin çözümü için önerilen algoritmaların performans analizi için deney aşamasında; KTYP, CEC 2015 nümerik fonksiyonları ve rüzgâr türbini yerleştirme problemi kullanılmıştır. İlk deneysel analiz KTYP problemi için yapılmıştır. Önerilen algoritmanın performansını analiz etmek ve doğrulamak için deneylerde 15 farklı KTY problemi kullanılmıştır. Önerilen algoritma yakın zamanda literatüre kazandırılmış PSO, ABC, TSA, DE, GA tabanlı başarılı ikili optimizasyon algoritmalarının sonuçları ile karşılaştırılmıştır. KTYP için deneysel sonuçlar incelendiğinde hem JayaX hem de JayaX-LSM algoritmaları KTYP için oldukça başarılı sonuçlar elde etmiştir. JayaX algoritması CapA, CapB ve CapC problemleri hariç olmak üzere diğer 12 problem için 30 çalıştırmanın tamamında optimum çözümlere ulaşmıştır. JayaX-LSM algoritması ise CapA problemi için de 30 çalıştırmanın tamamında optimum çözümlere ulaşmış olmak ile birlikte CapB ve CapC problemleri için de *gap* değerleri dikkate alındığında optimum sonuçlara en yakın çözümleri elde etmiştir. Sonuç olarak hem JayaX hem de JayaX-LSM algoritmaları KTYP'nin çözümü konusunda karşılaştırılan PSO, ABC, TSA, DE, GA ve AAA tabanlı başarılı ikili optimizasyon algoritmalarından daha iyi ya da rekabetçi çözümler elde etmiştir. Deneysel sonuçlardan da anlaşıldığı üzere JayaX-LSM algoritması JayaX algoritmasından daha iyi sonuçlar

elde etmiştir. Bu nedenle diğer ikili optimizasyon problemlerinin çözümü için deneylerde JayaX-LSM algoritması kullanılmıştır. İkinci deneysel analiz ise 15 kıyas probleminden oluşan CEC 2015 nümerik veri seti üzerine olmuştur. Bu analizde, JayaX-LSM algoritması SabDE, BQIGSA, GBABC, BHTPSO-QI, BLDE ve SBHS algoritmalarının sonuçları ile karşılatırmıştır ve elde edilen deneysel sonuçlar dikkate alındığında JayaX-LSM algoritması, karşılaştırmalarda kullanılan diğer algoritmalar ile rekabetçi ya da daha iyi çözümler elde etmiştir. Bu bölümde yapılan son deneysel analiz ise rüzgâr türbini yerleştirme problemi için yapılmıştır. Fosil-petrol kaynaklarının azalması, bu yakıtların oluşturduğu hava kirliliği, sera gazı etkisi ve iklim değişikliği gibi doğadaki birçok değişiklik yenilenebilir enerji kaynaklarına olan gereksinimin artmasına sebep olmuştur. Bundan dolayı yenilenebilir bir enerji kaynağı olan rüzgârdan maksimum verimle enerji elde etmek oldukça önem kazanmıştır. Deneylerde 10×10 ve 20×20 'lik olmak üzere iki farklı ızgara yapısı kullanılmıştır. 10×10 'luk ızgara alanı için, test alanı 100 eşit kareye bölünmekte ve muhtemel farklı çözümlerin sayısı 2^{100} olduğu için, ilk tür problem büyük ölçekli bir problem olarak sınıflandırılabilir. İkinci tür problemde ise, test alanı 20×20 'lik ızgara alanına bölünmüştür. Bu durumda test alanı 400 eşit kareye bölünmekte ve muhtemel farklı çözümlerin sayısı 2^{400} olduğu için, ikinci tür problem çok büyük ölçekli bir problem olarak sınıflandırılabilir. 10×10 'luk ızgara yapısı için türbin yerleştirme işleminde kullanılmak üzere 100 farklı konum vardır. Her bir karenin alanının küçültüldüğü 20×20 'lik ızgara yapısında ise 400 farklı konum vardır. Ancak türbinler arası uyandırma etkisinin en aza indirilebilmesi için komşu iki türbin arası mesafenin en az 5D (200 m) olması gerekmektedir. Bundan dolayı 20×20 'lik ızgara tipinde yerleştirme işlemini yapmak 10×10 'luk ızgara yapısına kıyasla daha zordur. Ama 20×20 'lik ızgara tipinde daha iyi bir yerleştirme ile rüzgâr türbini başına düşen enerji arttırılabilmektedir. Deneysel sonuçlar incelendiğinde 20×20 'lik ızgara yapısı üzerinde 10×10 'luk ızgara yapısına kıyasla daha küçük bir uygunluk değeri ile daha iyi bir yerleştirme işlemi yapılmıştır. Böylece rüzgâr çiftliğinden elde edilen verim yükseltilmiştir. Literatür ile yapılan karşılaştırmalarda ise JayaX-LSM algoritması GA tabanlı ikili yöntemler, BIWO, BPSO-TVAC, EA, NGHS, DGHS ve binAAA algoritmaları ile kıyaslanmıştır. Elde edilen sonuçlar değerlendirildiğinde JayaX-LSM algoritması karşılaştırılan algoritmalara benzer ya da daha iyi çözümler üretmiştir.

İkinci geliştirilen yöntem ise ayırık tam sayı optimizasyon problemlerinin çözümü için Jaya algoritması tabanlı DJaya olarak adlandırılan Ayırık Jaya algoritmasıdır. Temel

Jaya algoritması sürekli optimizasyon problemlerinin çözümü için geliştirildiğinden dolayı, ayrık tam sayı optimizasyon problemleri üzerine doğrudan uygulanamamaktadır. Bundan dolayı, ayrık tam sayı optimizasyon problemlerinin çözümü için temel Jaya algoritmasının konum güncelleme mekanizması yeniden tanımlanmıştır. Ayrıklaştırma işlemi için takas, öteleme ve simetri olarak adlandırılan komşuluk operatörleri kullanılmıştır. Deneysel analizlerde bu komşuluk operatörlerinin sekiz farklı kombinasyonu için analiz yapılmıştır. Yapılan deneysel analizlerden sonra, DJaya algoritmasının konum güncelleme mekanizmasında takas, öteleme ve simetri komşuluk operatörlerinin rulet tekerleği seçimine dayanan Kombine 2 mekanizması kullanmıştır. DJaya’da başlangıç popülasyonu oluşturulurken $(N - 1)$ tane aday çözüm rastgele permütasyon ile oluşturulurken, ilk aday çözüm ise en yakın komşu turu sezgiseli kullanılarak oluşturulmaktadır. Kiran (2015), TSA yönteminin keşif kabiliyetini geliştirmek için temel ağaç tohum algoritmasına Arama Eğilimi (Search Tendency - ST) parametresini dâhil etmiştir. Önerilen DJaya algoritmasına da çözüm uzayında keşif ve sömürü kabiliyeti arasında bir denge kurmak amacıyla $ST1$ ve $ST2$ olarak adlandırılan iki arama eğilimi parametresi dâhil edilmiştir. X'_k Aday çözümünün oluşturulması aşamasında, X_k , $Best$ ve $Worst$ çözümlerinden biri $ST1$ ve $ST2$ parametrelerine göre belirlenmektedir. Ayrıca DJaya’nın elde ettiği çözümlerin kalitesini arttırmak amacıyla 2-opt yerel arama algoritması da kullanılmıştır. DJaya algoritmasının başarısını ve etkinliğini göstermek amacıyla, deneylerde 14 farklı simetrik GSP ($EIL51$, $EIL76$, $EIL101$, $KROA100$, $KROB100$, $KROC100$, $KROD100$, $KROE100$, $OLIVER30$, $BERLIN52$, $TSP225$, $ST70$, $PR76$ ve $CH150$) kullanılmıştır. Elde edilen deneysel sonuçlar yakın zamanda literatüre kazandırılan başarılı algoritmaların sonuçları ile karşılaştırılmıştır. Deneysel bulgulara göre, DJaya algoritması GSP’nin çözümü için karşılaştırılan algoritmalarından daha başarılı ya da rekabetçi çözümler elde etmiştir.

5.2 Öneriler

Bu tez çalışması kapsamında ikili ve ayrık tam sayı optimizasyon problemlerinin çözümü için Jaya algoritması tabanlı yeni yaklaşımlar geliştirilmiştir. Tez çalışması kapsamında önerilen algoritmaların performans analizi için ise deney aşamasında; KTYP, CEC 2015 nümerik fonksiyonları, rüzgâr türbini yerleştirme problemi ve GSP kullanılmıştır. Daha sonraki çalışmalarda 0-1 SÇP, özellik seçimi problemi, iş akışı çizelgeleme problemi, graf boyama problemi gibi optimizasyon problemlerinin çözümü için önerilen algoritmalar kullanılabilir. Ayrıca mevcut problemlerin çözümü için de önerilen algoritmaların daha da geliştirilmiş versiyonları önerilebilir.

Temel Jaya algoritması kısıtlı veya kısıtsız sürekli optimizasyon problemlerinin çözümü için geliştirilmiş olan popülasyon tabanlı bir optimizasyon algoritmasıdır. Temel Jaya algoritmasında aday çözüm için yeni konum oluşturulurken her boyutta güncelleme işlemi yapılmaktadır. Bu da arama uzayında büyük bir pertürbasyona sebep olmakta ve bazen en iyi çözümden uzaklaşmaya neden olabilmektedir. İkili optimizasyon problemlerinin çözümü için bu çalışma kapsamında önerilen algoritmalarda, bu eksikliğin üstesinden gelebilmek için yeni bir mekanizma önerilmiştir. Bu mekanizma sayesinde problemin boyutuna bağlı olarak, her bir yinelemede güncellenecek boyut sayısı ve ilgili boyutlar belirlenmektedir. Bu çalışma kapsamında önerilen P olasılık parametresi, algoritmanın başında $[0,1]$ aralığında sabit bir oran olarak belirlenmekte ve tüm iterasyonlar boyunca sabit olarak kalmaktadır. Daha sonraki çalışmalarda P parametresinin değerinin belirlenmesi için farklı yaklaşımlar geliştirilebilir. Böylece P parametresinin etkinliği artırılabilir. Bununla birlikte, önerilen P parametresi, temel Jaya algoritmasına da uyarlanarak, kısıtlı veya kısıtsız sürekli optimizasyon problemlerinin çözümü için de bir performans analizi yapılabilir.

Ayrık tam sayı optimizasyon problemlerinin çözümü için önerilen DJaya algoritmasında, konum güncelleme aşaması için aday çözümün belirlenmesi sırasında $ST1$, $ST2$ arama eğilimi parametreleri kullanılmaktadır. Daha sonraki çalışmalarda bu parametreler için optimum değerlerin belirlenmesi amacıyla farklı yaklaşımlar geliştirilebilir. Ayrıca, DJaya'nın konum güncelleme mekanizmasında takas, öteleme ve simetri komşuluk operatörlerinin rulet tekerleği seçimine dayanan Kombine 2 mekanizması kullanılmaktadır. Daha sonraki çalışmalarda bu komşuluk operatörlerinin seçimi işlemi için farklı yaklaşımlar geliştirilebilir.

KAYNAKLAR

- Ahmed, Z. H., 2010, Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator, *International Journal of Biometrics & Bioinformatics (IJBB)*, 3 (6), 96.
- Akay, B., 2009, Nümerik optimizasyon problemlerinde yapay arı kolonisi algoritmasının performans analizi, Doktora Tezi, *Erciyes Üniversitesi Fen Bilimleri Enstitüsü*.
- Al-Sultan, K. S. ve Al-Fawzan, M. A., 1999, A tabu search approach to the uncapacitated facility location problem, *Annals of Operations Research*, 86, 91-103.
- Alataş, B., 2007, Kaotik Haritalı Parçacık Sürü Optimizasyon Algoritmaları Geliştirme, *Fırat Üniversitesi, Fen Bilimleri Enstitüsü*, Elazığ, 157.
- Arora, S., 1998, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems, *Journal of the ACM (JACM)*, 45 (5), 753-782.
- Aslan, M. ve Baykan, N. A., 2016, A performance comparison of graph coloring algorithms, *International Journal of Intelligent Systems and Applications in Engineering*, 1-7.
- Aslan, M., 2017, Graflarda düğüm boyama problemi için kurbağa sıçrama algoritması tabanlı bir yaklaşım, Yüksek Lisans Tezi, *Selçuk Üniversitesi Fen Bilimleri Enstitüsü*.
- Aydin, M. E. ve Fogarty, T. C., 2004, A distributed evolutionary simulated annealing algorithm for combinatorial optimisation problems, *Journal of Heuristics*, 10 (3), 269-292.
- Azad, M., Kalam, A., Rocha, A. M. A. ve Fernandes, E. M. d. G., 2013, A simplified binary artificial fish swarm algorithm for uncapacitated facility location problems, *World Congress on Engineering 2013, WCE 2013*, 31-36.
- Banitalebi, A., Aziz, M. I. A. ve Aziz, Z. A., 2016, A self-adaptive binary differential evolution algorithm for large scale binary optimization problems, *Information Sciences*, 367, 487-511.
- Bansal, J. C. ve Deep, K., 2012, A modified binary particle swarm optimization for knapsack problems, *Applied Mathematics and Computation*, 218 (22), 11042-11061.
- Barcelo, J., Hallefjord, Å., Fernandez, E. ve Jörnsten, K., 1990, Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing, *OR Spectrum*, 12 (2), 79-88.
- Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. ve Vance, P. H., 1998, Branch-and-price: Column generation for solving huge integer programs, *Operations Research*, 46 (3), 316-329.
- Beasley, J. E., 1990, Or-Library - Distributing Test Problems by Electronic Mail, *Journal of the Operational Research Society*, 41 (11), 1069-1072.
- Beheshti, Z., Shamsuddin, S. M. ve Hasan, S., 2015, Memetic binary particle swarm optimization for discrete optimization problems, *Information Sciences*, 299, 58-84.
- Bertsimas, D. ve Demir, R., 2002, An approximate dynamic programming approach to multidimensional knapsack problems, *Management Science*, 48 (4), 550-565.
- Beşkirli, M., Koç, İ., Haklı, H. ve Kodaz, H., 2018, A new optimization algorithm for solving wind turbine placement problem: Binary artificial algae algorithm, *Renewable Energy*, 121, 301-308.

- Beşkirli, M., Koc, I. ve Kodaz, H., 2019, Optimal placement of wind turbines using novel binary invasive weed optimization, *Tehnički vjesnik*, 26 (1), 56-63.
- Bilgen, S., Kaygusuz, K. ve Sari, A., 2004, Renewable energy for a clean and sustainable future, *Energy sources*, 26 (12), 1119-1129.
- Changshui, Z., Guangdong, H. ve Jun, W., 2011, A fast algorithm based on the submodular property for optimization of wind turbine positioning, *Renewable Energy*, 36 (11), 2951-2958.
- Chen, Q., Liu, B., Zhang, Q., Liang, J., Suganthan, P. ve Qu, B., 2014, Problem definitions and evaluation criteria for CEC 2015 special session on bound constrained single-objective computationally expensive numerical optimization, *Technical Report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University*.
- Chen, S.-M. ve Chien, C.-Y., 2011, Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques, *Expert Systems with Applications*, 38 (12), 14439-14450.
- Chen, Y., Xie, W. ve Zou, X., 2015, A binary differential evolution algorithm learning from explored solutions, *Neurocomputing*, 149, 1038-1047.
- Chentsov, A. ve Korotayeva, L., 1997, The dynamic programming method in the generalized traveling salesman problem, *Mathematical and Computer Modelling*, 25 (1), 93-105.
- Chunhua, Y., Xiaolin, T., Xiaojun, Z. ve Weihua, G., 2012, State transition algorithm for traveling salesman problem, *Proceedings of the 31st Chinese Control Conference*, 2481-2485.
- Cinar, A. C. ve Kiran, M. S., 2018, Similarity and logic gate-based tree-seed algorithms for binary optimization, *Computers & Industrial Engineering*, 115, 631-646.
- Cinar, A. C., Korkmaz, S. ve Kiran, M. S., 2019, A discrete tree-seed algorithm for solving symmetric traveling salesman problem, *Engineering Science and Technology, an International Journal*.
- Crespo, A. ve Herna, J., 1996, Turbulence characteristics in wind-turbine wakes, *Journal of Wind Engineering and Industrial Aerodynamics*, 61 (1), 71-85.
- Croes, G. A., 1958, A method for solving traveling-salesman problems, *Operations Research*, 6 (6), 791-812.
- Çınar, A. C., 2019, Kısıtlı ve ayrık optimizasyon problemlerinin çözümü için ağaç-tohum algoritmasının uyarlanması ve analizi, Doktora Tezi, *Konya Teknik Üniversitesi Lisansüstü Eğitim Enstitüsü*.
- Çunkaş, M., 2006, Genetik Algoritmalar ve Uygulamaları Ders Notları, <https://docplayer.biz.tr/14536665-Yrd-doc-dr-mehmet-cunkas.html>: [Ziyaret Tarihi: 15 Ağustos 2020].
- Dede, T., Grzywiński, M. ve Rao, R. V., 2020, Jaya: A New Meta-heuristic Algorithm for the Optimization of Braced Dome Structures, In: *Advanced Engineering Optimization Through Intelligent Techniques*, Eds: Springer, p. 13-20.
- Dorigo, M. ve Gambardella, L. M., 1997, Ant colony system: a cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation*, 1 (1), 53-66.
- Eberhart, R. C. ve Kennedy, J., 1995, A new optimizer using particle swarm theory, *Proceedings of the sixth international symposium on micro machine and human science*, 39-43.

- Emami, A. ve Noghreh, P., 2010, New approach on optimization in placement of wind turbines within wind farm by genetic algorithms, *Renewable Energy*, 35 (7), 1559-1564.
- Emine, B. ve Ülker, E., 2020, An efficient binary social spider algorithm for feature selection problem, *Expert Systems with Applications*, 146, 113185.
- Ergun, Ö. ve Orlin, J. B., 2006, A dynamic programming methodology in very large scale neighborhood search applied to the traveling salesman problem, *Discrete Optimization*, 3 (1), 78-85.
- Eusuff, M. M. ve Lansey, K. E., 2003, Optimization of water distribution network design using the shuffled frog leaping algorithm, *Journal of Water Resources Planning and Management*, 129 (3), 210-225.
- Falkenauer, E. ve Bouffouix, S., 1991, A genetic algorithm for job shop, *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, 824-829.
- Feng, Y., Wang, G.-G. ve Gao, X.-Z., 2016, A novel hybrid cuckoo search algorithm with global harmony search for 0-1 knapsack problems, *International Journal of Computational Intelligence Systems*, 9 (6), 1174-1190.
- Fleischmann, B., 1985, A cutting plane procedure for the travelling salesman problem on road networks, *European Journal of Operational Research*, 21 (3), 307-317.
- Gao, X., Yang, H., Lin, L. ve Koo, P., 2015, Wind turbine layout optimization using multi-population genetic algorithm and a case study in Hong Kong offshore, *Journal of Wind Engineering and Industrial Aerodynamics*, 139, 89-99.
- Geem, Z. W., 2005, Harmony search in water pump switching problem, *International Conference on Natural Computation*, 751-760.
- Geng, X., Chen, Z., Yang, W., Shi, D. ve Zhao, K., 2011, Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search, *Applied Soft Computing*, 11 (4), 3680-3689.
- Ghosh, D., 2003, Neighborhood search heuristics for the uncapacitated facility location problem, *European Journal of Operational Research*, 150 (1), 150-162.
- Goldberg, D. E. ve Holland, J. H., 1988, Genetic algorithms and machine learning.
- González, J. S., Rodriguez, A. G. G., Mora, J. C., Santos, J. R. ve Payan, M. B., 2010, Optimization of wind farm turbines layout using an evolutive algorithm, *Renewable Energy*, 35 (8), 1671-1681.
- Gould, N., 2006, An introduction to algorithms for continuous optimization, Oxford University Computing Laboratory Notes.
- Görkemli, B. ve Karaboga, D., 2013, Quick combinatorial artificial bee colony-qCABC-optimization algorithm for TSP, In, Eds, p.
- Grady, S., Hussaini, M. ve Abdullah, M. M., 2005, Placement of wind turbines using genetic algorithms, *Renewable Energy*, 30 (2), 259-270.
- Guner, A. R. ve Sevkli, M., 2008, A discrete particle swarm optimization algorithm for uncapacitated facility location problem, *Journal of Artificial Evolution and Applications*, 2008.
- Gülcü, Ş., Mahi, M., Baykan, Ö. K. ve Kodaz, H., 2018, A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem, *Soft Computing*, 22 (5), 1669-1685.
- Gündüz, M., Kiran, M. S. ve Özceylan, E., 2015, A hierarchic approach based on swarm intelligence to solve the traveling salesman problem, *Turkish Journal of Electrical Engineering & Computer Sciences*, 23 (1), 103-117.

- Hakli, H. ve Ortacay, Z., 2019, An improved scatter search algorithm for the uncapacitated facility location problem, *Computers & Industrial Engineering*, 135, 855-867.
- Hancer, E., Xue, B., Karaboga, D. ve Zhang, M., 2015, A binary ABC algorithm based on advanced similarity scheme for feature selection, *Applied Soft Computing*, 36 (Supplement C), 334-348.
- Hatamlou, A., 2018, Solving travelling salesman problem using black hole algorithm, *Soft Computing*, 22 (24), 8167-8175.
- Hernández-Pérez, H. ve Salazar-González, J.-J., 2004, A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery, *Discrete Applied Mathematics*, 145 (1), 126-139.
- Hertz, A. ve de Werra, D., 1987, Using tabu search techniques for graph coloring, *Computing*, 39 (4), 345-351.
- Holland, J. H., 1975, Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, U Michigan Press, p.
- Holmberg, K., 1999, Exact solution methods for uncapacitated location problems with convex transportation costs, *European Journal of Operational Research*, 114 (1), 127-140.
- Hristakeva, M. ve Shrestha, D., 2004, Solving the 0-1 knapsack problem with genetic algorithms, *Midwest Instruction and Computing Symposium*.
- Huang, C., Wang, L., Yeung, R. S.-C., Zhang, Z., Chung, H. S.-H. ve Bensoussan, A., 2018, A prediction model-guided Jaya algorithm for the PV system maximum power point tracking, *IEEE Transactions on Sustainable Energy*, 9 (1), 45-55.
- Huang, H.-S., 2009, Efficient hybrid distributed genetic algorithms for wind turbine positioning in large wind farms, *2009 IEEE International Symposium on Industrial Electronics*, 2196-2201.
- Hussain, A., Muhammad, Y. S., Nauman Sajid, M., Hussain, I., Mohamd Shoukry, A. ve Gani, S., 2017, Genetic algorithm for traveling salesman problem with modified cycle crossover operator, *Computational intelligence and neuroscience*, 2017.
- Ingle, K. K. ve Jatoth, R. K., 2020, An efficient JAYA algorithm with lévy flight for non-linear channel equalization, *Expert Systems with Applications*, 145, 112970.
- Irnich, S., Funke, B. ve Grünert, T., 2006, Sequential search and its application to vehicle-routing problems, *Computers & Operations Research*, 33 (8), 2405-2429.
- Ituarte-Villarreal, C. M. ve Espiritu, J. F., 2011, Optimization of wind turbine placement using a viral based optimization algorithm, *Procedia Computer Science*, 6, 469-474.
- Jaballah, S., Rouis, K., Abdallah, F. B. ve Tahar, J. B. H., 2014, An improved shuffled frog leaping algorithm with a fast search strategy for optimization problems, *Intelligent computer communication and processing (ICCP), 2014 IEEE international conference on*, 23-27.
- James, J. ve Li, V. O., 2015, A social spider algorithm for global optimization, *Applied Soft Computing*, 30, 614-627.
- Jaramillo, J. H., Bhadury, J. ve Batta, R., 2002, On the use of genetic algorithms to solve location problems, *Computers & Operations Research*, 29 (6), 761-779.
- Jensen, N. O., 1983, A note on wind generator interaction, Risø National Laboratory, p.

- Karaboga, D. ve Basturk, B., 2007, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, *Journal of global optimization*, 39 (3), 459-471.
- Karaboga, D. ve Gorkemli, B., 2011, A combinatorial artificial bee colony algorithm for traveling salesman problem, *2011 International Symposium on Innovations in Intelligent Systems and Applications*, 50-53.
- Karaboga, D., 2011, Yapay Zekâ Optimizasyon Algoritmaları, Nobel Yayın Dağıtım: 244.
- Kashan, M. H., Nahavandi, N. ve Kashan, A. H., 2011, DisABC: a new artificial bee colony algorithm for binary optimization, *Applied Soft Computing*.
- Kashan, M. H., Kashan, A. H. ve Nahavandi, N., 2013, A novel differential evolution algorithm for binary optimization, *Computational Optimization and Applications*, 55 (2), 481-513.
- Kennedy, J. ve Eberhart, R. C., 1997, A discrete binary version of the particle swarm algorithm, *Systems, Man, and Cybernetics*, 1997. *Computational Cybernetics and Simulation*, 1997 IEEE International Conference on, 4104-4108.
- Khuri, S., Bäck, T. ve Heitkötter, J., 1994, The zero/one multiple knapsack problem and genetic algorithms, *Proceedings of the 1994 ACM symposium on Applied computing*, 188-193.
- Kiran, M. S. ve Gunduz, M., 2013, XOR-based artificial bee colony algorithm for binary optimization, *Turkish Journal of Electrical Engineering & Computer Sciences*, 21 (Sup. 2), 2307-2328.
- Kiran, M. S., 2015a, The continuous artificial bee colony algorithm for binary optimization, *Applied Soft Computing*, 33 (Supplement C), 15-23.
- Kiran, M. S., 2015b, TSA: Tree-seed algorithm for continuous optimization, *Expert Systems with Applications*, 42 (19), 6686-6698.
- Kıran, M. S., İşcan, H. ve Gündüz, M., 2013, The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem, *Neural Computing and Applications*, 23 (1), 9-21.
- Kıran, M. S., 2014, Optimizasyon Problemlerinin Çözümü İçin Yapay Arı Kolonisi Algoritması Tabanlı Yeni Yaklaşımlar, Doktora Tezi, *Selçuk Üniversitesi Fen Bilimleri Enstitüsü*.
- Knox, J., 1994, Tabu search performance on the symmetric traveling salesman problem, *Computers & Operations Research*, 21 (8), 867-876.
- Kong, X., Gao, L., Ouyang, H. ve Li, S., 2015, A simplified binary harmony search algorithm for large scale 0–1 knapsack problems, *Expert Systems with Applications*, 42 (12), 5337-5355.
- Korkmaz, S., Babalik, A. ve Kiran, M. S., 2018, An artificial algae algorithm for solving binary optimization problems, *International Journal of Machine Learning and Cybernetics*, 9 (7), 1233-1247.
- Korkmaz, S., 2019, İkili optimizasyon problemlerinin çözümü için yapay alg algoritması tabanlı yeni yaklaşımlar, Doktora Tezi, *Konya Teknik Üniversitesi Lisansüstü Eğitim Enstitüsü*.
- Krarup, J. ve Pruzan, P. M., 1983, The Simple Plant Location Problem - Survey and Synthesis, *European Journal of Operational Research*, 12 (1), 36-81.
- Krause, J., Cordeiro, J., Parpinelli, R. S. ve Lopes, H. S., 2013, A survey of swarm algorithms applied to discrete optimization problems, In: *Swarm Intelligence and Bio-Inspired Computation*, Eds: Elsevier, p. 169-191.

- Kumawat, M., Gupta, N., Jain, N. ve Bansal, R., 2020, Jaya Algorithm Based Optimal Allocation of Distributed Energy Resources, In: *Intelligent Computing Techniques for Smart Energy Systems*, Eds: Springer, p. 805-814.
- Lenstra, J. K. ve Kan, A. R., 1975, Some simple applications of the travelling salesman problem, *Journal of the Operational Research Society*, 26 (4), 717-733.
- Li, L., Cheng, Y., Tan, L. ve Niu, B., 2011, A discrete artificial bee colony algorithm for TSP problem, *International Conference on Intelligent Computing*, 566-573.
- Lim, T. Y., Al-Betar, M. A. ve Khader, A. T., 2016, Taming the 0/1 knapsack problem with monogamous pairs genetic algorithm, *Expert Systems with Applications*, 54, 241-250.
- Lin, S., 1965, Computer solutions of the traveling salesman problem, *Bell System Technical Journal*, 44 (10), 2245-2269.
- Lin, S. ve Kernighan, B. W., 1973, An effective heuristic algorithm for the traveling-salesman problem, *Operations Research*, 21 (2), 498-516.
- Liu, C., Yan, X., Liu, C. ve Wu, H., 2011, The wolf colony algorithm and its application, *Chinese Journal of Electronics*, 20 (2), 212-216.
- Lovbjerg, M., 2002, "Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality", PhD dissertation, Department of Computer Science, University of Aarhus, Datalogisk Institut.
- Luo, J. ve Chen, M.-R., 2014, Improved shuffled frog leaping algorithm and its multi-phase model for multi-depot vehicle routing problem, *Expert Systems with Applications*, 41 (5), 2535-2545.
- Mahi, M., Baykan, Ö. K. ve Kodaz, H., 2015, A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem, *Applied Soft Computing*, 30, 484-490.
- Mirjalili, S. ve Lewis, A., 2013, S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization, *Swarm and Evolutionary Computation*, 9 (Supplement C), 1-14.
- Mittal, A., 2010, Optimization of the layout of large wind farms using genetic algorithm, MS thesis, *Department of Mechanical and Aerospace Engineering, Case Western Reserve University*.
- Mittal, P., Kulkarni, K. ve Mitra, K., 2016, A novel hybrid optimization methodology to optimize the total number and placement of wind turbines, *Renewable Energy*, 86, 133-147.
- Monabbati, E. ve Kakhki, H. T., 2015, On a class of subadditive duals for the uncapacitated facility location problem, *Applied Mathematics and Computation*, 251, 118-131.
- Moorthy, C. B. ve Deshmukh, M., 2015, A new approach to optimise placement of wind turbines using particle swarm optimisation, *International Journal of Sustainable Energy*, 34 (6), 396-405.
- More, K. C. ve Rao, R. V., 2020, Design Optimization of Plate-Fin Heat Exchanger by Using Modified Jaya Algorithm, In: *Advanced Engineering Optimization Through Intelligent Techniques*, Eds: Springer, p. 165-172.
- Mosetti, G., Poloni, C. ve Diviacco, B., 1994, Optimization of wind turbine positioning in large windfarms by means of a genetic algorithm, *Journal of Wind Engineering and Industrial Aerodynamics*, 51 (1), 105-116.
- Murty, K. G., 2003, Optimization Models For Decision Making, vol. 1, Chapter 1, 1-18, http://www-personal.umich.edu/~murty/books/opti_model/junior-1.pdf: [Erişim Tarihi: 18 Ağustos 2020].

- Nezamabadi-pour, H., 2015, A quantum-inspired gravitational search algorithm for binary encoded optimization problems, *Engineering Applications of Artificial Intelligence*, 40, 62-75.
- Omer, A. M., 2008, Energy, environment and sustainable development, *Renewable and Sustainable Energy Reviews*, 12 (9), 2265-2300.
- Ouaarab, A., Ahiod, B. ve Yang, X.-S., 2014, Discrete cuckoo search algorithm for the travelling salesman problem, *Neural Computing and Applications*, 24 (7-8), 1659-1669.
- Ozturk, C., Hancer, E. ve Karaboga, D., 2015, A novel binary artificial bee colony algorithm based on genetic operators, *Information Sciences*, 297, 154-170.
- Padberg, M. ve Rinaldi, G., 1987, Optimization of a 532-city symmetric traveling salesman problem by branch and cut, *Operations Research Letters*, 6 (1), 1-7.
- Pang, W., Wang, K.-p., Zhou, C.-g. ve Dong, L.-j., 2004, Fuzzy discrete particle swarm optimization for solving traveling salesman problem, *The Fourth International Conference on Computer and Information Technology*, 2004. CIT'04., 796-800.
- Parada, L., Herrera, C., Flores, P. ve Parada, V., 2017, Wind farm layout optimization using a Gaussian-based wake model, *Renewable Energy*, 107, 531-541.
- Patel, J., Savsani, V., Patel, V. ve Patel, R., 2017, Layout optimization of a wind farm to maximize the power output using enhanced teaching learning based optimization technique, *Journal of Cleaner Production*, 158, 81-94.
- Pookpant, S. ve Ongsakul, W., 2013, Optimal placement of wind turbines within wind farm using binary particle swarm optimization with time-varying acceleration coefficients, *Renewable Energy*, 55, 266-276.
- Potvin, J.-Y., 1996, Genetic algorithms for the traveling salesman problem, *Annals of Operations Research*, 63 (3), 337-370.
- Pouraltafi-Kheljan, S., Azimi, A., Mohammadi-ivatloo, B. ve Rasouli, M., 2018, Optimal design of wind farm layout using a biogeographical based optimization algorithm, *Journal of Cleaner Production*, 201, 1111-1124.
- Prakash, T., Singh, V., Singh, S. ve Mohanty, S., 2017, Binary Jaya Algorithm Based Optimal Placement of Phasor Measurement Units for Power System Observability.
- Radharamanan, R. ve Choi, L., 1986, A branch and bound algorithm for the travelling salesman and the transportation routing problems, *Computers & Industrial Engineering*, 11 (1-4), 236-240.
- Raidl, G. R., 1998, An improved genetic algorithm for the multiconstrained 0-1 knapsack problem, *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, 207-211.
- Rao, R., 2016, Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems, *International Journal of Industrial Engineering Computations*, 7 (1), 19-34.
- Rao, R. ve More, K., 2017, Design optimization and analysis of selected thermal devices using self-adaptive Jaya algorithm, *Energy Conversion and Management*, 140, 24-35.
- Rao, R. V., Rai, D., Ramkumar, J. ve Balic, J., 2016a, A new multi-objective Jaya algorithm for optimization of modern machining processes, *Advances in Production Engineering & Management*, 11 (4), 271.
- Rao, R. V., Rai, D. P. ve Balic, J., 2016b, Surface grinding process optimization using Jaya algorithm, In: *Computational Intelligence in Data Mining—Volume 2*, Eds: Springer, p. 487-495.

- Rao, R. V., Rai, D. P. ve Balic, J., 2017, A multi-objective algorithm for optimization of modern machining processes, *Engineering Applications of Artificial Intelligence*, 61, 103-125.
- Rao, R. V. ve Saroj, A., 2017, Constrained economic optimization of shell-and-tube heat exchangers using elitist-Jaya algorithm, *Energy*, 128, 785-800.
- Rao, R. V. ve Waghmare, G., 2017, A new optimization algorithm for solving complex constrained design optimization problems, *Engineering Optimization*, 49 (1), 60-83.
- Rao, R. V., 2019, *Jaya: An Advanced Optimization Algorithm and its Engineering Applications*, Switzerland, Springer International Publishing, p. 334.
- Rao, R. V. ve Saroj, A., 2019, An elitism-based self-adaptive multi-population Jaya algorithm and its applications, *Soft Computing*, 23, 4383-4406.
- Rashedi, E., Nezamabadi-Pour, H. ve Saryazdi, S., 2009, GSA: a gravitational search algorithm, *Information Sciences*, 179 (13), 2232-2248.
- Ravikumar, C., 1992, Parallel techniques for solving large scale travelling salesperson problems, *Microprocessors and Microsystems*, 16 (3), 149-158.
- Rehman, S., Ali, S. ve Khan, S. A., 2016, Wind farm layout design using cuckoo search algorithms, *Applied Artificial Intelligence*, 30 (10), 899-922.
- Reinelt, G., 1991, TSPLIB—A traveling salesman problem library, *ORSA Journal on Computing*, 3 (4), 376-384.
- Reinelt, G., 1994, *The traveling salesman: computational solutions for TSP applications*, Springer-Verlag, p.
- Rizk-Allah, R. M. ve Hassanien, A. E., 2018, New binary bat algorithm for solving 0-1 knapsack problem, *Complex & Intelligent Systems*, 4 (1), 31-53.
- Rizk-Allah, R. M., Hassanien, A. E., Elhoseny, M. ve Gunasekaran, M., 2019, A new binary salp swarm algorithm: development and application for optimization tasks, *Neural Computing and Applications*, 31 (5), 1641-1663.
- Sayadi, M. K., Hafezalkotob, A. ve Naini, S. G. J., 2013, Firefly-inspired algorithm for discrete optimization problems: an application to manufacturing cell formation, *Journal of Manufacturing Systems*, 32 (1), 78-84.
- Sevcli, M. ve Guner, A., 2006, A continuous particle swarm optimization algorithm for uncapacitated facility location problem, *Ant colony optimization and swarm intelligence*, 316-323.
- Shi, X. H., Liang, Y. C., Lee, H. P., Lu, C. ve Wang, Q., 2007, Particle swarm optimization-based algorithms for TSP and generalized TSP, *Information processing letters*, 103 (5), 169-176.
- Shoaib, M., Siddiqui, I., Rehman, S., Khan, S. ve Alhems, L. M., 2019, Assessment of wind energy potential using wind energy conversion system, *Journal of Cleaner Production*, 216, 346-360.
- Singh, P. ve Chaudhary, H., 2020, Optimal shape synthesis of a metallic flywheel using non-dominated sorting Jaya algorithm, *Soft Computing*, 24, 6623-6634.
- Topcuoglu, H., Corut, F., Ermis, M. ve Yilmaz, G., 2005, Solving the uncapacitated hub location problem using genetic algorithms, *Computers & Operations Research*, 32 (4), 967-984.
- Ulder, N. L., Aarts, E. H., Bandelt, H.-J., Van Laarhoven, P. J. ve Pesch, E., 1990, Genetic local search algorithms for the traveling salesman problem, *International Conference on Parallel Problem Solving from Nature*, 109-116.
- Uymaz, S. A., 2015, Yeni bir biyolojik ilhamlı metasezgisel optimizasyon metodu: Yapay alg algoritması, *Selçuk Üniversitesi, Fen Bilimleri Enstitüsü*, Konya, 111.

- Uymaz, S. A., Tezel, G. ve Yel, E., 2015, Artificial algae algorithm (AAA) for nonlinear global optimization, *Applied Soft Computing*, 31, 153-171.
- Uzlu, E., 2019, Application of Jaya algorithm-trained artificial neural networks for prediction of energy use in the nation of Turkey, *Energy Sources, Part B: Economics, Planning, and Policy*, 14 (5), 183-200.
- Üçoluk, G., 2002, Genetic algorithm solution of the TSP avoiding special crossover and mutation, *Intelligent Automation & Soft Computing*, 8 (3), 265-272.
- Wan, C., Wang, J., Yang, G., Li, X. ve Zhang, X., 2009, Optimal micro-siting of wind turbines by genetic algorithms based on improved wind and turbine models, *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, 5092-5096.
- Wang, K.-P., Huang, L., Zhou, C.-G. ve Pang, W., 2003, Particle swarm optimization for traveling salesman problem, *Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no. 03ex693)*, 1583-1585.
- Wang, L. ve Huang, C., 2018, A novel Elite Opposition-based Jaya algorithm for parameter estimation of photovoltaic cell models, *Optik-International Journal for Light and Electron Optics*, 155, 351-356.
- Wang, L., Huang, C. ve Huang, L., 2018a, Parameter estimation of the soil water retention curve model with Jaya algorithm, *Computers and Electronics in Agriculture*, 151, 349-353.
- Wang, L., Zhang, Z., Huang, C. ve Tsui, K. L., 2018b, A GPU-accelerated parallel Jaya algorithm for efficiently estimating Li-ion battery model parameters, *Applied Soft Computing*, 65, 12-20.
- Warid, W., 2020, Optimal power flow using the AMTPG-Jaya algorithm, *Applied Soft Computing*, 106252.
- Wolpert, D. H. ve Macready, W. G., 1997, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, 1 (1), 67-82.
- Wu, C. ve He, Y., 2020, Solving the set-union knapsack problem by a novel hybrid Jaya algorithm, *Soft Computing*, 24 (3), 1883-1902.
- Xiang, W.-l., An, M.-q., Li, Y.-z., He, R.-c. ve Zhang, J.-f., 2014, A novel discrete global-best harmony search algorithm for solving 0-1 knapsack problems, *Discrete Dynamics in Nature and Society*, 2014.
- Yaman, F., 2014, Optimizasyon problemlerinin çözümünde hesaplama maliyetinin azaltılması, Doktora Tezi, *Ankara Üniversitesi*, Ankara Üniversitesi Fen Bilimleri Enstitüsü.
- Yang, X.-S., 2010, Engineering optimization: an introduction with metaheuristic applications, John Wiley & Sons, p.
- Zhang, X., Wu, C., Li, J., Wang, X., Yang, Z., Lee, J.-M. ve Jung, K.-H., 2016, Binary artificial algae algorithm for multidimensional knapsack problems, *Applied Soft Computing*, 43, 583-595.
- Zhong, W.-h., Zhang, J. ve Chen, W.-n., 2007, A novel discrete particle swarm optimization to solve traveling salesman problem, *2007 IEEE Congress on Evolutionary Computation*, 3283-3287.
- Zhou, X., Gao, D. Y., Yang, C. ve Gui, W., 2016, Discrete state transition algorithm for unconstrained integer optimization problems, *Neurocomputing*, 173, 864-874.
- Zhuang, F. ve Galiana, F., 1990, Unit commitment by simulated annealing, *IEEE Transactions on Power Systems*, 5 (1), 311-318.
- Zou, D., Gao, L., Li, S. ve Wu, J., 2011, Solving 0–1 knapsack problem by a novel global harmony search algorithm, *Applied Soft Computing*, 11 (2), 1556-1564.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Murat ASLAN
Uyruğu : T.C
Doğum Yeri ve Tarihi : Nusaybin / 18.03.1988
Telefon :
Faks :
e-mail : mrtasln@hotmail.com

EĞİTİM

Derece	Adı, İlçe, İl	Bitirme Yılı
Lise	: Nusaybin (Y.D.A) Süper Lisesi, Nusaybin, Mardin	2006
Üniversite	: Selçuk Üni., Bilgisayar Müh., Selçuklu, Konya	2011
Yüksek Lisans	: Selçuk Üni., Bilgisayar Müh. ABD, Selçuklu, Konya	2017
Doktora	: Konya Teknik Üni., Bilgisayar Müh. ABD, Selçuklu, Konya	-

İŞ DENEYİMLERİ

Yıl	Kurum	Görevi
2013 - 2014	Şırnak Üniversitesi Mühendislik Fakültesi	Araştırma Görevlisi
2014 - 2018	Selçuk Üniversitesi Mühendislik Fakültesi	Araştırma Görevlisi
2018 -	Şırnak Üniversitesi Mühendislik Fakültesi	Araştırma Görevlisi

UZMANLIK ALANI

Ayrık tam ve ikili optimizasyon problemleri, graf teorisi, sürü zekası veya evrimsel hesaplama algoritmaları, yapay zeka.

YABANCI DİLLER

İngilizce

YAYINLAR

Aslan, M., Gunduz, M. ve Kiran, M. S., 2019, JayaX: Jaya algorithm with xor operator for binary optimization, Applied Soft Computing, 82, 105576. doi:10.1016/j.asoc.2019.105576. (SCI-Expanded) (Doktora tezinden üretilmiştir).

Aslan, M., Gunduz, M. ve Kiran, M. S., 2020, A Jaya-based approach to wind turbine placement problem, *Energy Sources, Part A: Recovery, Utilization, And Environmental Effects*, 1-20. doi: 10.1080/15567036.2020.1805528. (SCI-Expanded) (Doktora tezinden üretilmiştir).

Atay, Y., Aslan, M. ve Kodaz, H., 2018, A swarm intelligence-based hybrid approach for identifying network modules, *Journal of computational science*, 28, 265-280. doi: 10.1016/j.jocs.2017.10.011. (SCI-Expanded)

Aslan, M., Beskirli, M., Kodaz, H. ve Kıran, M. S., 2018, An improved tree seed algorithm for optimization problems, *Int J Mach Learn Comput*, 8 (1), 20-25. doi: 10.18178/ijmlc.2018.8.1.657.

Aslan, M. ve Baykan, N. A., 2016, A performance comparison of graph coloring algorithms, *International Journal of Intelligent Systems and Applications in Engineering*, 1-7. doi: 10.18201/ijisae.273053.