T.C
SELÇUK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BASKI DEVRE KARTI DELİK DELME MAKİNELERİ İÇİN GENETİK ALGORİTMALAR YARDIMI İLE GÜZERGÂH BELİRLEME

MURAT ALBAYRAK

YÜKSEK LİSANS TEZİ

ELEKTRONİK ve BİLGİSAYAR SİSTEMLERİ
EĞİTİMİ ANABİLİM DALI

KONYA, 2008

T.C
SELÇUK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BASKI DEVRE KARTI DELİK DELME MAKİNELERİ İÇİN GENETİK ALGORİTMALAR YARDIMI İLE GÜZERGÂH BELİRLEME

Murat ALBAYRAK

YÜKSEK LİSANS TEZİ

ELEKTRONİK VE BİLGİSAYAR SİSTEMLERİ
EĞİTİMİ ANABİLİM DALI

Bu tez 03.07.2008 tarihinde aşağıdaki jüri tarafından oybirliği ile kabul edilmiştir.

.....	.....	.....
Prof. Dr.	Prof. Dr.	Yrd. Doç. Dr.
Novruz ALLAHVERDİ	Ahmet ARSLAN	Mehmet ÇUNKAŞ
(Danışman)	(Üye)	(Üye)

ÖZET

Yüksek Lisans Tezi

BASKI DEVRE KARTI DELİK DELME MAKİNELERİ İÇİN
GENETİK ALGORİTMALAR YARDIMI
İLE GÜZERGÂH BELİRLEME

Murat ALBAYRAK

Selçuk Üniversitesi Fen Bilimleri Enstitüsü
Elektronik ve Bilgisayar Sistemleri Eğitimi Anabilim Dalı

Danışman: Prof. Dr. Novruz ALLAHVERDİ

2008, 180 Sayfa

Jüri : Prof. Dr. Novruz ALLAHVERDİ
Prof. Dr. Ahmet ARSLAN
Yrd. Doç. Dr. Mehmet ÇUNKAŞ

Bu tez, elektronikte kullanılan baskı devre kartlarının makineler tarafından en kısa sürede delinmesi için en kısa mesafeye sahip delme işlemi güzergâhını bulmak amacıyla hazırlanmıştır. Tanımlanan problem bir optimizasyon işlemi olup literatürde Gezgin Satıcı Problemi (GSP) olarak bilinmektedir ve delme makinesinin en kısa turu dolaşması ile elektronik sanayindeki seri üretim yapan işletmelerde verim artışı sağlanması hedeflenmektedir. Bu amaçla, Genetik Algoritmalar (GA) birçok farklı operatörü ile birlikte, sezgisel algoritmalar, adaptif denetimler, yerel arama metotları, paralelizm ve tur bölme teknikleri ile melez bir yapıda uygulanmıştır. Çalışmada kullanılan tüm metotların reel ve etkileşimli performansları analiz edilerek en uygun melez parametreler ve operatörler belirlenmeye çalışılmış, belirlenen parametreler ile literatürdeki 63 farklı GSP probleminde 10 adet deneme yapılmıştır.

Araştırmalarda kullanılan Basit Genetik Algoritmaların en iyi hata oranının yaklaşık %3.60988, ortalama hata oranının ise yaklaşık %5.35182 olduğu ve çok fazla çalışma süresi talep ettiği görülmüştür. Tez çalışmasında yeni geliştirilen Paralel Genetik Yerel Arama Algoritmasının GSP optimizasyonunda en iyi hata oranını yaklaşık %0.01795 seviyesine, ortalama hata oranını ise yaklaşık %0.04192 seviyesine düşürdüğü görülmüştür. Sonuç olarak GSP optimizasyonu için yeni geliştirilen Paralel Genetik Yerel Arama Algoritmasının oldukça başarılı olduğu gözlenmiştir.

Anahtar Kelimeler: Genetik Algoritmalar, Paralel Genetik Yerel Arama Algoritması, Sezgisel Algoritmalar, Tur Bölme, Adaptif Denetim, Gezgin Satıcı Problemi, Optimizasyon

ABSTRACT

MS Thesis

DETERMINATION OF ROUTE BY MEANS OF
GENETIC ALGORITHMS
FOR PRINTED CIRCUIT BOARD DRILLER MACHINES

Murat ALBAYRAK

Selçuk University
Graduate School of Natural Applied Sciences
Department of Electronics and Computer Systems Education

Supervisor: Prof. Dr. Novruz ALLAHVERDİ

2008, 180 Page

Jury : Prof. Dr. Novruz ALLAHVERDİ
Prof. Dr. Ahmet ARSLAN
Assist. Prof. Dr. Mehmet ÇUNKAŞ

This thesis has been prepared to find the drilling route having the shortest distance for drilling the printed circuit boards by machines in the shortest time. The defined problem is an optimization process and is known as Traveling Salesman Problem (TSP) in the literature. It aims to increase the efficiency by the help of the drilling machine's feature of following the shortest tour in the businesses which make mass production in the electronic industry. For this aim, Genetic Algorithms (GA), besides many different operators, have been applied in a hybrid way with heuristic algorithms, adaptive controls, local search methods, parallelism and tour partition methods. The most appropriate hybrid parameters and operators have been tried to be determined by analyzing the sole and interactive performances of all the methods used in the research. 10 trials have been made on 63 different TSP problems in the literature with the determined parameters.

It has been seen that, the best error rate of Simple Genetic Algorithms used in the researches is approximately 3.60988%, average error rate is approximately 5.35182% and that, for these error rates it requires so much running time. It has also been seen that, the Parallel Genetic Local Search Algorithms, which has just been developed in the thesis study, has reduced the best error rate to 0.01795% and average error rate to approximately 0.04192% in the TSP optimization. As a result, the newly developed Parallel Genetic Local Search Algorithm for the TSP optimization is highly successful.

Keywords: Genetic Algorithms, Parallel Genetic Local Search Algorithms, Heuristic Algorithms, Tour Partition, Adaptive Controls, Traveling Salesman Problem, Optimization

TEŞEKKÜR

Hazırlanan bu çalışmada yardımlarını esirgemeyen danışman hocam Sayın Prof. Dr. Novruz ALLAHVERDİ'ye, Yrd. Doç. Dr. Mehmet ÇUNKAŞ hocama, Chungnam Ulusal Üniversitesinden Prof. Dr. Dongil SEO'ya, AT&T Laboratuarlarından David S. JOHNSON'a, sabrı ve tezimdeki yardımlarından dolayı sevgili eşim Oya ALBAYRAK'a ve her şeyim olan aileme teşekkürlerimi sunarım.

Ayrıca tez çalışması sırasındaki analizler sürecinde gerekli bilgisayar donanımı imkânlarını sağlayan aynı zamanda görev yerim olan Konya Hüyük Selki Kasabası Merkez İlköğretim Okulu idaresine (Sayın Ramazan ALTUN ve Ramazan GÜNDOĞDU) gösterdikleri sabır ve anlayıştan dolayı minnettarım.

Bu çalışma S.Ü. BAP koordinatörlüğü tarafından 07201054 nolu proje ile desteklenmiştir. Kendilerine teşekkür ederim.

İÇİNDEKİLER

ÖZET iii
ABSTRACT iv
TEŞEKKÜR v
İÇİNDEKİLER vi
ŞEKİLLER LİSTESİ x
TABLOLAR LİSTESİ xiii
SİMGELER xv
1. GİRİŞ 1
1.1. Tezin Amacı ve Önemi 2
1.2. Tezin Organizasyonu 4
2. GEZGİN SATICI PROBLEMİ 5
2.1. Kesin Çözüm Algoritmaları 7
2.2. Yaklaşık Algoritmalar 8
2.2.1. Tur Oluşturma Algoritmaları 8
2.2.2. Tur Geliştirme Algoritmaları 10
2.2.3. Meta-Sezgisel Algoritmalar 10
2.2.3.1. Tavlama Benzetimi 11
2.2.3.2. Tabu Arama 13
2.2.3.3. Karınca Kolonisi 14
2.2.3.4. Genetik Algoritmalar 16
3. GENETİK ALGORİTMALAR 18
3.1. Genetik Algoritmanın Tarihçesi 18
3.2. Genetik Algoritmanın Tanımı 19
3.3. Genetik Algoritmanın Bileşenleri 21
3.3.1. Kodlama 21
3.3.1.1. İkili Kodlama 22
3.3.1.2. Permutasyon Kodlama 23
3.3.1.3. Değer Kodlama 23
3.3.1.4. Ağaç Kodlama 24
3.3.2. Sunum 25

3.3.2.1. Yol Sunumu 25
3.3.2.2. Bitişik Sunum 25
3.3.2.3. Sıra Sunum 25
3.3.3. Başlangıç Popülasyonu 26
3.3.4. Seçim 27
3.3.4.1. Ağırlıklı Seçim 27
3.3.4.2. Sıralama Seçimi 27
3.3.4.3. Rasgele Seçim 28
3.3.4.4. Turnuva Seçim 28
3.3.4.5. Rulet Tekerleği Seçimi 28
3.3.5. Eşleştirme 29
3.3.5.1. Yukarıdan Aşağıya Doğru Eşleştirme 30
3.3.5.2. Rasgele Eşleştirme 30
3.3.5.3. Ağırlıklı Rasgele Eşleştirme 30
3.3.5.4. En İyi – En Kötü Eşleştirmesi 31
3.3.5.5. İmparator Eşleştirmesi 31
3.3.4. Çaprazlama 32
3.3.5. Mutasyon 33
3.3.6. Seçkinlik 34
3.3.7. Durdurma Kriterleri 35
3.4. Genetik Algoritmanın Çeşitleri 36
3.4.1. Basit Genetik Algoritma 36
3.4.2. Paralel Genetik Algoritma 36
3.4.3. Mikro Genetik Algoritma 37
3.4.4. Melez Genetik Algoritma 37
3.4.5. Genetik Yerel Arama Algoritması 38
4. GENETİK YEREL ARAMA ALGORİTMASI VE BİLEŞENLERİ 42
4.1. Tur Oluşturma Metotları 43
4.1.1. En Yakın Komşu Tur Oluşturma Metodu 43
4.1.2. Çift Bitişli En Yakın Komşu Tur Oluşturma Metodu 45
4.1.3. En Yakın Parça Tur Oluşturma Metodu 45
4.1.4. En Yakını Ekleme Tur Oluşturma Metodu 47

4.1.5. En Uzağı Ekleme Tur Oluşturma Metodu 48
4.1.6. Rasgele Ekleme Tur Oluşturma Metodu 49
4.2. Çaprazlama Metotları 50
4.2.1. Tek Noktalı Çaprazlama 51
4.2.2. İki Noktalı Çaprazlama 52
4.2.3. Çok Noktalı Çaprazlama 53
4.2.4. Kısmi Haritalanmış Çaprazlama 54
4.2.5. Çevrim Çaprazlama 55
4.2.6. Sıralı Çaprazlama 57
4.2.7. Değiştirilmiş Sıralı Çaprazlama 58
4.2.8. Sıra Temelli Çaprazlama 60
4.2.9. Pozisyon Temelli Çaprazlama 60
4.2.10. Dönüşümlü Kenar Çaprazlama 61
4.2.11. Yeniden Kenar Birleşimi Çaprazlama 63
4.2.12. Sezgisel Çaprazlama 68
4.2.13. Sezgisel Çaprazlama–2 70
4.2.14. Açgözlü Alt Tur Çaprazlama 70
4.2.15. Mesafe Korumalı Çaprazlama 71
4.3. Mutasyon Metotları 72
4.3.1. Değiş-Tokuş Mutasyonu 72
4.3.2. Yerdeğiştirme Mutasyonu 73
4.3.3. Tersleme Mutasyonu 73
4.3.4. Ekleme Mutasyonu 74
4.3.5. Basit Tersleme Mutasyonu 74
4.3.6. Karıştırma Mutasyonu 75
4.5.7. Çift Köprü Mutasyonu 75
4.3.8. Açgözlü Değiş-Tokuş Mutasyonu 76
4.3.9. Inver-Over Operatörü 77
4.3.10. Tersleme-En Yakın Komşu Mutasyonu 78
4.3.11. Açgözlü Alt Tur Mutasyonu 79
4.4. Adaptif Stratejiler 82
4.4.1. Adaptif Strateji–1 82

4.4.2. Adaptif Strateji–2 84
4.5. Yerel Arama Metotları 85
4.5.1. 2-Opt Metodu 85
4.5.2. 3-Opt Metodu 87
4.5.3. 2.5-Opt Metodu 89
4.5.4. Lin-Kernighan Metodu 90
4.5.5. Yerel Aramayı Hızlandırma Teknikleri 94
4.5.5.1. Sabit Yarıçaplı En Yakın Komşu Arama 95
4.5.5.2. Bitlere Bakmama 96
4.5.5.3. Aday Listeleri 96
4.6. Geometrik Bölme Metotları 99
4.7. Paralel Genetik Algoritmanın Yapısı ve Bileşenleri 102
4.8. Diğer Kullanılan Metotlar 105
5. GENETİK ALGORİTMA VE BİLEŞENLERİ İÇİN UYGUN PARAMETRELERİ BELİRLEME 107
5.1. Geliştirilen Yazılım Özellikleri 109
5.2. Genetik Algoritmada Popülasyon Büyüklüğünün Etkisi 111
5.3. Tur Oluşturma Metotlarının Çözüm Kalitesi 112
5.4. Tur Oluşturma Metotlarının Genetik Algoritma Üstündeki Etkisi 113
5.5. Çaprazlama Oranının Etkisi 116
5.6. Mutasyon Oranının Etkisi 118
5.7. Adaptif Kontrol Parametresi Belirlemenin Etkisi 119
5.8. Çaprazlama ve Mutasyon Operatörlerinin Etkisi 122
5.9. Seçim ve Eşleştirme Metotlarının Etkisi 124
5.10. Yerel Arama Metotlarının Etkisi 126
5.11. Aday Listesi Metotları ve Büyüklüklerinin Etkisi 128
5.12. Genetik Algoritma ve Tur Bölme Metodu İle Optimizasyon 130
5.13. Paralel Genetik Algoritmanın Etkisi ve Parametrelerin Belirlemesi 134
6. SONUÇLAR VE ÖNERİLER 138
KAYNAKLAR 146
EKLER 155

ŞEKİLLER LİSTESİ

Şekil 2.1. En Yakın Komşu Algoritmasının Açgözlü Davranışı 9
Şekil 2.2. Genel Tavlama Benzetimi Algoritması Adımları 12
Şekil 2.3. Genel Tabu Arama Algoritması Adımları 14
Şekil 2.4. Karınca Çevrim Algoritmasının Temel Adımları 15
Şekil 2.5. Basit Genetik Algoritma Adımları 16
Şekil 3.1. Basit Genetik Algoritma Akış Diyagramı 20
Şekil 3.2. Çözüm Uzayında Bir Bireyin Komşusu ile Hamming Mesafesi 23
Şekil 3.3. Ağaç Kodlama 24
Şekil 3.4. Basit Tek Noktalı Çaprazlama Metodu 32
Şekil 3.5. Yer Değiştirme Mutasyon İşlemi 33
Şekil 3.6. P_M Değerinin Çözüm Sürecinde Etkisi 34
Şekil 3.7. Genetik Yerel Arama Algoritması 39
Şekil 3.8. Farklı Bir Genetik Yerel Arama Algoritması 40
Şekil 4.1. En Yakın Komşunun Lin318 Problemindeki Açgözlü Davranışı 44
Şekil 4.2. En Yakın Parça ile Tur Oluşumu 47
Şekil 4.3. En Yakın Ekleme ile Tur Oluşumu 48
Şekil 4.4. En Uzak Ekleme ile Tur Oluşumu 49
Şekil 4.5. MOX ve OX Operatörlerinin KroA100 Problemindeki Performansları 59
Şekil 4.6. EX için Eklenmeye Çalışılan Şehrin Yapacağı Kenarlar 63
Şekil 4.7. ERX için Seçilen Ebeveynler 64
Şekil 4.8. ERX'de Ebeveynlerden {1} Numaralı Şehir için Oluşabilecek Kenarlar 64
Şekil 4.9. ERX'de Ebeveynlerden {4} Numaralı Şehir için Oluşabilecek Kenarlar 65
Şekil 4.10. ERX-5 (Edge–5) İçin Kenar Seçimi 67
Şekil 4.11. ERX ve Türevlerinin Çözümdeki Etkileri 68
Şekil 4.12. HX için Bireyler ve Kenar Uzunlukları 69
Şekil 4.13. Ardışık Olmayan Çift Köprü Mutasyonu 76
Şekil 4.14. Açgözlü Değiş-Tokuş Mutasyon Operatörü 77
Şekil 4.15. Inver-Over Operatörünün Algoritması 78
Şekil 4.16. Inver-RNN'nin Inver-Over'dan Farklı Olan Kod Bloğu 79
Şekil 4.17. Açgözlü Alt Tur Mutasyon Operatörü Algoritması 81

Şekil 4.18. Inver-RNN'nin GST Mutasyon Operatörü Üzerindeki Etkisi 81
Şekil 4.19. Adaptif Kontrol Parametresi Belirleme: Strateji–1 83
Şekil 4.20. Adaptif Kontrol Parametresi Belirleme: Strateji–2 84
Şekil 4.21. Basit 2-Opt Algoritması 86
Şekil 4.22. Bir 2-Opt Hareketi 86
Şekil 4.23. Basit 3-Opt Algoritması 88
Şekil 4.24. İki Mümkün 3-Opt Hareketi 89
Şekil 4.25. LK için 3-Opt Hareketi 91
Şekil 4.26. LK için x_i, y_i, x_{i+1} ve y_{i+1}'in Seçim Şartları 91
Şekil 4.27. LK için Mümkün Olmayan Ardışık Değişim 92
Şekil 4.28. Lin-Kernighan Algoritması 93
Şekil 4.29. Sabit Yarıçaplı En Yakın Komşu Arama 95
Şekil 4.30. CL_{NN} Aday Listesi 97
Şekil 4.31. Hızlandırılmış 2-Opt Yerel Arama Metodu 98
Şekil 4.32. Karp Tur Bölme Metodu 100
Şekil 4.33. Tur Bölme Metodu–1 101
Şekil 4.34. Tur Bölme Metodu–2 102
Şekil 4.35. PGA Bağlantı Topolojileri 105
Şekil 5.1. Geliştirilen Programın Basit Akış Diyagramı 107
Şekil 5.2. Popülasyonunun Büyüklüğünün Etkisi 111
Şekil 5.3. Başlangıç Popülasyonunun Zaman Üstündeki Etkisi 112
Şekil 5.4. Tur Oluşturma Metotlarının Hata Oranları 113
Şekil 5.5. Başlangıç Metotlarının Genetik Algoritma Üzerindeki Etkisi 114
Şekil 5.6. Başlangıç Metotlarının Karıştırmalı Analiz Ortalamaları 115
Şekil 5.7. Çaprazlama Oranlarının GSP Optimizasyonundaki Etkisi 117
Şekil 5.8. Mutasyon Oranlarının GSP Optimizasyonundaki Etkisi 118
Şekil 5.9. Mutasyon Oranlarının GSP Optimizasyonundaki Hızı 119
Şekil 5.10. GSP Optimizasyonunda Adaptif ve Sabit Mutasyon Oranlarının Etkisi 121
Şekil 5.11. Adaptif Mutasyon Oranının Zamana Bağlı Değişimi 121
Şekil 5.12. Çaprazlama Operatörlerinin Ortalama Hata Değerleri 123
Şekil 5.13. Mutasyon Operatörlerinin Ortalama Hata Değerleri 123
Şekil 5.14. Yerel Arama Metotlarının Hata Değerleri 126

Şekil 5.15. Genetik Yerel Arama Hata ve Süre Ortalamaları 127
Şekil 5.16. Aday Listesi Metotlarının Hata Değerleri..... 128
Şekil 5.17. Alt Problemlerin BGA ile Araştırılması 131
Şekil 5.18. Tur Bölme ile Oluşan Alt Turlar..... 132
Şekil 5.19. U2319 için Tur Bölme ile Oluşan Sonuç Turları..... 133
Şekil 5.20. PGA Parametre Değişimlerinin Ortalama Hata Oranları..... 135
Şekil 5.21. PGA-BGA için Mesafe-Jenerasyon Grafiği 136
Şekil 5.22. Pcb1173 Problemi için GA ile Bulunan Güzergâh..... 137
Şekil 5.23. Pcb1173 Problemi için PGA ile Bulunan Güzergâh..... 137
Şekil 6.1. BGA, PGA, GYA ve PGYA Hata Değerleri 139
Şekil 6.2. Algoritmaların Çalışma Süreleri..... 140
Şekil 6.3. BGA ve GTB+BGA Metotları Hata ve Süre Değerleri..... 142

TABLOLAR LİSTESİ

Tablo 2.1. Tavlama Benzetiminde GSP Parametre Dönüşümleri 11
Tablo 3.1. Çeşitli GYA Performanslarının Karşılaştırılması 38
Tablo 4.1. En Yakın Parçanın Performansı 46
Tablo 4.2. OX ve MOX Operatörlerinin Çeşitli Algoritmalardaki Sonuçları 59
Tablo 4.3. ERX Kenar Haritası 65
Tablo 4.4. Yerel Aramayı Hızlandırma Tekniklerinin Etkisi 94
Tablo 4.5. GSP'deki Şehir Sayısı ile Çözüm Uzayı Büyüklüğü İlişkisi 99
Tablo 5.1. Kullanılan Bilgisayar Konfigürasyonları 108
Tablo 5.2. Adaptif Mutasyon Oranı Etkisi 120
Tablo 5.3. Seçim-Eşleştirme Kombinasyonu Hata ve Süre Ortalamaları 125
Tablo 5.4. U2319 Probleminde Tur Bölme Katsayısı Analizi 130
Tablo 6.1. Analizlerde Kullanılan Parametre Değerleri 138
Tablo 6.2. Genetik Algoritma ve Geometrik Tur Bölme Sonuçları 141
Tablo 6.3. Çeşitli GSP Metotlarının Ortalama Hata Değerleri 144
Tablo 6.4. Çeşitli GSP Metotlarının Ortalama Hata Değerleri – 2 145
Ek-Tablo 1. Popülasyon Büyüklüğünün GA Çözümündeki Etkisi 156
Ek-Tablo 2. Tur Oluşturma Metotlarının Hata Değerleri 157
Ek-Tablo 3. Tur Oluşturma Metotları ile Başlayan BGA'nın Hata Değerleri 158
Ek-Tablo 4. Tur Oluşturma Metotlarının Karıştırılmasının Etkisi 159
Ek-Tablo 5. Sabit Çaprazlama Oranının Çözümdeki Etkisi 160
Ek-Tablo 6. Sabit Mutasyon Oranının Çözümdeki Etkisi 161
Ek-Tablo 7. Berlin52 için Çaprazlama-Mutasyon Çifti Ortalama Hata Değerleri .. 162
Ek-Tablo 8. Eil76 için Çaprazlama-Mutasyon Çifti Ortalama Hata Değerleri 163
Ek-Tablo 9. KroB200 için Çaprazlama-Mutasyon Çifti Ortalama Hata Değerleri . 164
Ek-Tablo 10. Berlin52&KroA150 için Seçim-Eşleştirme Çifti Ortalama Hataları . 165
Ek-Tablo 11. Ts225&Pcb443 için Seçim-Eşleştirme Çifti Ortalama Hataları 166
Ek-Tablo 12. Yerel Arama Metotlarının Çözümdeki Etkisi 167
Ek-Tablo 13. Genetik Algoritmada Yerel Arama Metotlarının Çözümdeki Etkisi . 168
Ek-Tablo 14. Komşu Listesi Büyüklüğünün Çözümdeki Etkisi 169
Ek-Tablo 15. Komşu Listesi Büyüklüğünün Çözümdeki Etkisi 170

Ek-Tablo 16. Aday Listesi Uygulanış Biçiminin Çözümdeki Etkisi 171
Ek-Tablo 17. Pcb1173 TSP Problemi için PGA Parametre Analizi 172
Ek-Tablo 18. Standart Genetik Algoritma Sonuç Değerleri – 1 173
Ek-Tablo 19. Standart Genetik Algoritma Sonuç Değerleri – 2 174
Ek-Tablo 20. Paralel Genetik Algoritma Sonuç Değerleri – 1 175
Ek-Tablo 21. Paralel Genetik Algoritma Sonuç Değerleri – 2 176
Ek-Tablo 22. Genetik Yerel Arama Algoritması Sonuç Değerleri – 1 177
Ek-Tablo 23. Genetik Yerel Arama Algoritması Sonuç Değerleri – 2 178
Ek-Tablo 24. Paralel Genetik Yerel Arama Algoritması Sonuç Değerleri – 1 179
Ek-Tablo 25. Paralel Genetik Yerel Arama Algoritması Sonuç Değerleri – 2 180

SİMGELER

C_i : *i*. Şehir
n : Toplam Şehir Sayısı
X_i : *i*. Şehrin X Koordinatı
Y_i : *i*. Şehrin Y Koordinatı
T : Tur
S : Başlangıç Çözümü
$f(S)$: *S* Çözümünün Toplam Mesafesi
d : Ağırlık Matrisi
p_i : *i*. Bireyin Seçilme İhtimali
Kp_i : *i*. Bireyin Kümülatif Seçilme İhtimali
N : Popülasyon Sayısı
$P_Ç$: Çaprazlama İhtimali
Ptr : Çaprazlama Deseni
R_i : Çaprazlama Noktaları
P_M : Mutasyon İhtimali
ϑ : Yerel (Lokal) Optimum
m : Göç Eden Birey Sayısı
ξ : Göç Aralığı
ω_S : Göç Edecek Birey Seçimi
ω_R : Göç Yerleşim Politikası
ε : Hata
$\varepsilon_{İYİ}$: Bulunan En İyi Hata Değeri
ε_{ORT} : Bulunan Ortalama Hata Değeri
$\varepsilon_{KÖTÜ}$: Bulunan En Kötü Hata Değeri
t : Süre
t_{ORT} : Ortalama Süre
ms : Milisaniye
s : Saniye
β : Bölme Oranı
∞ : Sonsuz Tam Sayı

Kısaltmalar

BGA : Basit Genetik Algoritma
CL : Aday Listeleri (*Candidate Lists*)
CLK : Zincirlenmiş (*Chained*) Lin-Kernighan
CL_{MAX} : Maksimum Aday Listesi Büyüklüğü
CL_{NN} : En Yakın Komşular Aday Listesi (*Nearest Neighbors*)
CL_{QN} : Çeyrek Daire Komşuları Aday Listesi (*Quadrant Neighbors*)
CX : Çevrim Çaprazlama (*Cycle Crossover*)
DB : Çift Köprü Mutasyonu (*Double-Bridge Mutation*)
DENN : Çift Bitişli En Yakın Komşu (*Double Ended Nearest Neighbor*)
DISP : Yer Değiştirme Mutasyonu (*Displacement Mutation*)
DLB : Bitlere Bakmama (*Don't Look Bits*)
DPX : Mesafe Korumalı Çaprazlama (*Distance Preserving Crossover*)
ERX : Yeniden Kenar Birleşimi Çaprazlama (*Edge Recombination Crossover*)
ETŞ : Ebeveyn Tabanlı Çaprazlama Şablonu
EX : Dönüşümlü Kenar Çaprazlama (*Alternate Edge Crossover*)
EXC : Değiş-Tokuş Mutasyonu (*Exchange Mutation*)
FI : En Uzağı Ekleme (*Farthest Insertion*)
FRNNS : Sabit Yarıçaplı En Yakın Komşu Arama (*Fixed Radius Nearest Neighbors Search*)
GA : Genetik Algoritma
GSM : Açgözlü Değiş-Tokuş Mutasyonu (*Greedy Swap Mutation*)
GSP : Gezgin Satıcı Problemi
GST : Açgözlü Alt Tur Mutasyonu (*Greedy Sub Tour Mutation*)
GSX : Açgözlü Alt Tur Çaprazlama (*Greedy Sub Tour Crossover*)
GTB : Geometrik Tur Bölme
GYA : Genetik Yerel Arama
HM : Hamming Mesafesi
HX : Sezgisel Çaprazlama (*Heuristic Crossover*)
INS : Ekleme Mutasyonu (*Insertion Mutation*)
INV : Tersleme Mutasyonu (*Inversion Mutation*)

İYA : İterasyonel Yerel Arama
KK : Karınca Kolonisi
KONF : Kullanılan Sistem Konfigürasyonu
LK : Lin-Kernighan
LKH : Lin-Kernighan Helsgaun
MA : Memetik Algoritma
μGA : Mikro Genetik Algoritma
MGA : Melez Genetik Algoritma
MGTB : Melez Genetik Tavlama Benzetimi
MMAS : Max-Min Karınca Sistemi (*Max-Min Ant System*)
MOX : Değiştirilmiş Sıralı Çaprazlama (*Modified Order Crossover*)
NF : En Yakın Parça (*Nearest Fragment*)
NI : En Yakını Ekleme (*Nearest Insertion*)
NN : En Yakın Komşu (*Nearest Neighbor*)
OPT : Optimizasyon
OX : Sıralı Çaprazlama (*Order Crossover*)
PBX : Pozisyon Temelli Çaprazlama (*Position Based Crossover*)
PGA : Paralel Genetik Algoritma
PGYA : Paralel Genetik Yerel Arama
PMX : Kısmi Haritalanmış Çaprazlama (*Partially Mapped Crossover*)
PTB : Paralel Tavlama Benzetimi
RI : Rasgele Ekleme (*Random Insertion*)
RND : Rasgele (*Random*)
RŞ : Rasgele Çaprazlama Şablonu
SCM : Karıştırma Mutasyonu (*Scramble Mutation*)
S-GSP : Simetrik Gezgin Satıcı Problemi
SIM : Basit Tersleme Mutasyonu (*Simple Inversion Mutation*)
SŞ : Statik Çaprazlama Şablonu
TA : Tabu Arama
TB : Tavlama Benzetimi
TSPLIB : Gezgin Satıcı Problemi Kütüphanesi

1. GİRİŞ

Günümüzdeki problemler eskiye nazaran çok daha karmaşık ve büyük boyutlu hale gelmektedir. Bu yüzden problemlerin çözümü aşamasında insanların yerini, iyi programlanmış süper bilgisayarlar almaktadır. Bunun temel sebebi, bilgisayarların birçok işlemi insanlara göre saniyede milyarlarca kez daha hızlı yapabilme kapasiteleri ve hata yapmadan sonraki işlemlerini devam ettirebilmeleridir. Özellikle bazı büyük ve karmaşık yapıda olan problem alanlarında, tam çözümlerin süper bilgisayar ağları kullanılarak dahi çözümlenmesi 15-20 yıl gibi oldukça uzun zaman alabilmektedir. Tam çözümleri bulabilmek için harcanan sürenin çok fazla olmasından dolayı, daha kısa sürede ve optimuma çok yakın çözümler kabul edilebilir duruma gelmektedir. Dolayısıyla karmaşık ve büyük problemlere bilgisayarlarla çözüm aramak, bu çözümü arayacak yazılımlar ve algoritmalar geliştirmek gün geçtikçe önem kazanmaktadır. Bu çözümlerin zaman içerisinde araştırılarak iyileştirilmesi sürecine ise kısaca optimizasyon denilmektedir.

Literatürde yer alan Gezgin Satıcı Problemi (*Traveling Salesman Problem*: TSP) kısaca, gezgin satıcının belirlenen şehirleri, her şehre bir defa uğramak ve sonunda başlangıç şehrine dönüp toplamda en kısa mesafeyi kat etmesi için gereken güzergâhın (turun) tayini problemidir. Gezgin Satıcı Problemi (GSP) çözümü oldukça zor kombinasyonel optimizasyon problemi olup, günlük hayatta birçok önemli uygulama alanında kullanılmaktadır. Baskı devre kartlarının delinmesi, araçların güzergâhının belirlenmesi, dağıtım problemleri, bilgisayar ağı bağlantıları, robot kontrolü ve iş planlama bunlardan sadece birkaçıdır.

Günümüzde GSP optimizasyonunda, düşük hata değerlerine sahip çözümler kabul edilmektedir ve optimizasyon sürecinin performansı iki parametre dikkate alınarak değerlendirilir. Bu parametreler çözüm süresi ve hata değeridir. GSP için tam çözüme yakın sonuçlar üretebilmek amacıyla kullanılan birçok başarılı yapay zekâ tekniği ve metot mevcuttur. Literatürde sıkça kullanılan yapay zekâ teknikleri Tabu Arama, Tavlama Benzetimi, Karınca Kolonisi ve Genetik Algoritmalardır.

GSP'de minimum tur mesafesini bulmak amacıyla kullanılan birçok algoritmadan birisi olan Genetik Algoritma (GA), biyolojik bir sistemin çevresine adaptasyonunu temel alan, evrim temellerine dayanan bir algoritmadır. GA doğal seçim mekaniklerine ve doğal genetiklerin en uygunlarının hayatta kalması prensibine dayanır. GA, popülasyondaki ebeveynlerden alınan genetik bilgilerle oluşturulan nesillerin popülasyondaki kötü bireylerin yerine geçip, sürekli iyileştirme için çalışan bir algoritmadır. GA verilen problemi optimize ederken yapısında bulunan birçok bileşenin kullanılması ile optimal çözüme veya yerel optimum çözümlere yaklaşır. Popülasyondaki genetik bilginin nasıl başlayacağı (başlangıç popülasyonu), genetik bilginin hangi ebeveynlerden alınacağı (seçim) ve bu bilgilerin yavruya hangi düzende aktarılacağı (çaprazlama ve mutasyon) gibi işlemler GA operatörleri tarafından gerçekleştirilir.

Holland tarafından 1975'te ortaya konan GA, GSP gibi zor kombinasyonel optimizasyon problemlerinin küçük boyutlu olanlarında optimal çözümler bulabilirken, büyük boyutlu olanlarında optimum değeri bulmakta diğer çeşitli metotlarla birleştirilerek geliştirilmiş olan genetik metotlara göre oldukça yetersiz kalmaktadır (Hu ve Gu 2007). Bu yüzden GA'ya, GSP çözümündeki performansı yükseltecek çeşitli metot ve stratejilerin adapte edilmesi gerekmektedir. GA'nın, başlangıç metodu oluşturma teknikleri, güçlü çaprazlama ve mutasyon operatörleri, problemi bölerek küçük parçalar halinde araştırma teknikleri, sezgisel metotlar, adaptif stratejiler, tur geliştirme metotları ve melez, mikro veya paralel GA modelleri ile birleştirilmesi sonucu oluşan yeni GA modeli ile GSP için oldukça etkili sonuçlar elde edilebilir. Bu metotlardan bazılarının veya kombinasyonlarının kullanılmasıyla oluşturulmuş GA modelleri için literatürde oldukça başarılı örnekler bulunmaktadır (Nagata 2006, Seo ve Moon 2002, Merz ve Freisleben 1997, Ulder ve ark. 1991, Katayama ve Narihisa 1999, Schleuter 1997, 1989).

1.1. Tezin Amacı ve Önemi

Dünyamızda teknolojik ürünler ve üretimleri alanında gün geçtikçe rekabet artmaktadır. Üretim yapan firmaların dünya pazarında yerlerini alabilmeleri için sıfır

hata ve ucuz üretim maliyeti şartlarını yerine getirmeleri gerekmektedir. Sektörde, donanımsal ve yazılımsal ihtiyaçlar karmaşık problemler oluşturmakta, çözüm yolları da firmalara oldukça yüksek maliyetli harici çözüm bileşenlerini beraberinde getirmektedir. Bu yüzden makine sanayi ve yazılım sektörü, firmaların isteklerine cevap verebilecek şekilde oldukça hızlı gelişmeler göstermektedir.

Makine parkı maliyetlerinin firmalara yüksek miktarda maddi yük getirmesinin yanında, rekabet için gereken sıfır hata, çalışma zamanının daha iyi kullanılması, makine yıpranmalarının ve arızalarının, hata oranının, sürekli uzmana bağımlılığın, üretim maliyetlerinin ve gereksiz insan gücünün azaltılması gibi özellikleri de kazandırmaktadır. Bazı imalat problemleri, donanımsal güçlendirmeler yerine yazılım ile iyileştirilerek firmaların masrafları azaltılabilir.

Elektronik sanayinde kullanılan “Baskı Devre Kartı Delme Makinesi” (*Printed Circuit Board Driller*) gibi seri imalat yapması beklenen sistemlerde, delinecek deliklerin sıralamasının bir uzman tarafından, bazı basit CAM yazılımları veya basit algoritmalar kullanarak düzenlenmesi seri üretimdeki verimi önemli ölçüde azaltmakta ve bu da sonuç olarak maliyetlere yansımaktadır. Baskı devre kartlarının ideal olarak delinmesi süreci, kartın üzerindeki tüm noktaların sadece bir kez delinmesi, delinmemiş nokta kalmaması ve makinenin ilk başlangıç pozisyonuna geri dönerek yeni bir baskı devre kartının yerleştirilmesine imkân sağlayarak tüm bu işleri minimum sürede halletmesi olarak kısaca özetlenebilir. Dolayısıyla baskı devre kartlarının en kısa yolu takip ederek delinmesi aslında bir GSP optimizasyonudur (GSP’deki gezgin, makinenin matkabı; şehirlerde delinecek olan noktalardır).

Sonuç olarak hazırlanan tezin iki temel amacı vardır. Birincisi, GA’nın birçok metot ve strateji ile güçlendirilip GSP optimizasyonundaki performansın arttırılması ve bunun nedenlerinin incelenmesidir. İkincisi ise çeşitli metotlarla güçlendirilip geliştirilen GA ile GSP için optimum değere daha yakın değerler elde edip, elektronik sanayinde üretim yapan firmaların, üretim ve makine maliyetlerinin, insan gücü ve üretim sürelerinin azaltılması olarak özetlenebilir.

1.2. Tezin Organizasyonu

Bir GSP optimizasyonu olan baskı devre kartlarının delinme güzergâhının kısaltılmasıyla, seri üretimin desteklemesi için hazırlanan bu tezin 2. bölümünde GSP'nin yapısı, türleri ve optimizasyonu için kullanılan literatürdeki tekniklerden bahsedilecektir.

3. bölümde Genetik Algoritmalar (GA), bileşenleri ve GA'nın alt dalları olan Melez GA, Mikro GA, Paralel GA (PGA) ve Genetik Yerel Arama (GYA) modelleri incelenecektir. 4. bölümde GSP optimizasyonu için geliştirilen GA'da kullanılacak olan çaprazlama ve mutasyon operatörleri, başlangıç turu oluşturma yaklaşımları, adaptif kontroller, yerel arama metotları, yerel aramayı hızlandırma teknikleri, geometrik tur bölme teknikleri ve PGA tanıtılmaktadır. Yine 4. bölümde GA ile GSP optimizasyonu için bu çalışmada geliştirilen ve oldukça etkili bir mutasyon operatörü olan Açgözlü Alt Tur Mutasyonu tanıtılmaktadır. 5. bölümde tüm kullanılan metotların performansları ve optimizasyon sürecindeki uygun parametre değerleri belirlenecektir. 6. ve son bölümde, Genetik Algoritma, Paralel Genetik Algoritma, Genetik Yerel Arama Algoritması ve Paralel Genetik Yerel Arama Algoritması modellerinde literatürde yer alan 63 farklı GSP için analizler yapılmış ve tüm algoritmalardan elde edilen sonuçlar değerlendirilmiştir.

Hazırlanan tezin kapsamının oldukça geniş olmasından dolayı literatür araştırmaları 2., 3. ve 4. bölümler içerisinde yer almaktadır.

2. GEZGİN SATICI PROBLEMİ

Gezgin Satıcı Problemi (GSP), belirlenen şehirler arasında bir satıcının minimum yol maliyeti (uzaklığı) ile dolaşmasını esas alan bir kombinasyonel optimizasyon problemdir. Satıcı her şehre bir kere uğramalı ve uğramadığı şehir kalmadan en kısa mesafeyi dolaşacak şekilde başlangıç şehrine geri dönmelidir. Bu şekilde gezgin turunu tamamlamalıdır. GSP, kombinasyonel optimizasyon problemlerinden en çok çalışılan alandır (Helsgaun 2006). Aynı zamanda NP-Zor (*NP-Hard*, *NP*: *Nondeterministic Polynomial*) tipinde bir problemdir ve polinominal sürede tam olarak çözülemez (Potvin 1996).

Satıcının gezdiği tüm şehirlerin (n) koordinatları (x_i, y_i) bilindiği zaman yapılan turun toplam uzunluğu (f) aşağıdaki gibi hesaplanır.

$$f = \sum_{i=1}^{n-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + \sqrt{(x_n - x_1)^2 + (y_n - y_1)^2} \qquad (2.1)$$

Eğer her bir şehir (C_i) arasındaki mesafenin saklandığı iki boyutlu simetrik bir Uzaklık Matrisi (d) tanımlanır ise turun toplam mesafesi $d(C_i, C_j)$ matrisinden aşağıda belirtildiği gibi hesaplanır ($d(C_i, C_j) = i$ ve j şehirleri arasındaki mesafe).

$$f = d(C_1, C_2) + d(C_2, C_3) + ... + d(C_{n-1}, C_n) + d(C_n, C_1) \qquad (2.2)$$

GSP'de yer alan şehirlerin birbirlerine olan uzaklık bilgilerinin saklandığı uzaklık matrisinin oluşumu sırasında problem sınıfına göre bir yapılandırma gerçekleştirilmelidir. Bu oluşum farklılığı literatürde yer alan farklı GSP tiplerinden kaynaklanmaktadır.

- İki nokta grubu (i, j) arasındaki uzaklıklar birbirine eşit ise bu problem *simetrik*'tir ($d(C_i, C_j) = d(C_j, C_i)$, $1 \le i, j \le n$).

- İki nokta grubu arasındaki uzaklıklar birbirine eşit değil ise bu problem *asimetrik*'tir ($d(C_i, C_j) \neq d(C_j, C_i)$, $1 \leq i, j \leq n$).
- Eğer üçgen eşitsizliği ($d(C_i, C_k) \leq d(C_i, C_j) + d(C_j, C_k)$, $1 \leq i, j, k \leq n$) sağlanıyor ise bu tip problemler *metrik* olarak tanımlanır.
- Eğer $d(C_i, C_j)$ iki nokta arasındaki Öklit (*Euclidean*) mesafesi ise problem *Euclidean* olarak tanımlanır. *Euclidean* problemler hem metrik hem de simetriktir.

Öklit Mesafesi: İki nokta arasındaki cetvelle ölçülebilen ve Pisagor teoreminin uygulanmasıyla kanıtlanabilen mesafedir.

Bu yüzden bir baskı devre delme makinesinin (*PCB-Driller*) çalışma güzergâhı tayini Simetrik GSP (S-GSP) tipte bir problemdir. Burada makinenin deleceği delikler şehirlere karşılık gelmektedir. Satıcının başlangıç noktasına geri dönerek turu tamamlaması da makine matkabının çalışmaya başladığı referans noktasına tekrar dönmesi demektir ki bu da yeni bir baskı devre kartının kolayca yerleştirilmesi için gereken bir şarttır. n şehirli bir GSP problemi için $n!$ tur içerisindeki en kısa tur aranır. S-GSP problemleri için kapalı bir tur döngüsü olduğu için bu ihtimal ½ oranında azalır. Bu yüzden S-GSP'ler için mümkün olan benzersiz tur sayısı $(n-1)!/2$ dir.

Gerçek hayattaki birçok uygulama alanı GSP problemi olarak formüle edilebilir (Helsgaun 2000, Michalewicz ve Fogel 2000).

1. Bilgisayar bağlantıları (Ağ kabloları)
2. Araç güzergâhı belirleme
3. X-Ray kristalografi
4. Robot kontrolü
5. Baskı devre kartlarının delinmesi
6. Baskı devre kartı üretiminde kalıp çizimi
7. Kronolojik sıralama
8. Gaz türbin motor kontrolü

9. Data dizilerinin kümelenmesi
10. İş planlama

Euler tarafından 1759'da tanımlanan satrançtaki Atın Turu (*Knight's Tour*) problemi GSP ile ilgili ilk doküman olmuştur. Atın turu probleminde düğümler satranç tahtası üzerindeki 64 karedir ve atın tek hamlede gidebildiği düğüm komşu düğüm olarak kabul edilmiştir. "Gezgin Satıcı" terimi ilk defa 1923'de bir Alman kitabında kullanılmıştır (Lawler 1985). 1940'larda istatistikçiler tarafından tarımsal uygulamayla bağlantılı olarak incelendi ve matematikçi Merril Flood, RAND kurumunda meslektaşları arasında yaygınlaştırdı. George Dantzig, Ray Fulkerson, and Selmer Johnson's 1954'te 49 U.S.A. şehri ile S-GSP problemi çözdüler (Dantzig ve ark. 1954). 1987'de Groetschel ve Holland dünyadaki 666 tane yer arasındaki optimal turu bulmuşlardır. 2001 yılında 15112 şehirli bir GSP optimal olarak hesaplandı (Applegate ve ark. 2001). Bu hesaplama Rice ve Princeton üniversitelerinde 110 işlemcili bir ağda gerçekleştirildi ve hesaplama zamanı yaklaşık 22.6 yıla eşdeğerdir. GSP üzerinde yıllardır yapılan çalışmalarda birçok farklı algoritma kullanılmış ve sürekli bu algoritmaların geliştirilmesi üzerinde çalışmalar devam etmiştir.

2.1. Kesin Çözüm Algoritmaları

GSP için kesin optimum değeri bulmak için kullanılan algoritmalardır. Bu algoritmalar tüm mümkün turları oluşturup her birisinin uzunluğunu hesaplar. Böylece, ihtimal uzayındaki tüm varyasyonlar denendiğinden sonucun da optimum olması garantidir. Sadece boyutu 10'dan az olan problemlerde kullanılması mantıklıdır. Örneğin 200 şehre sahip bir S-GSP için $1.97x10^{372}$ ihtimal söz konusudur. Günümüzde binlerce şehirli problemlerin çözümü ile ilgili raporlar olmasına rağmen, kesin çözüm algoritmaları ile simetrik problemlerde sadece birkaç yüz problemin çözümüne ulaşılmıştır (Helsgaun 2000) .

En etkili kesin çözüm algoritmaları, *Brach-and-Bound*, *Cutting-Plane* ve *Facet-Finding* algoritmalarıdır ki, bu algoritmalar oldukça kompleks yapılara sahiptir.

Onbinlerce satır program kodundan oluşmaktadırlar. 2392 şehirli bir S-GSP optimizasyonu için oldukça güçlü bir bilgisayar ağı 27 saatten fazla çalışmıştır. 7397 şehirli bir S-GSP optimizasyonu ise çok geniş bir bilgisayar ağında 3-4 yıllık bir işlemci zamanına ihtiyaç duymuştur (Applagete ve ark. 1995).

2.2. Yaklaşık Algoritmalar

Yaklaşık algoritmalar iyi sonuçlar verebilirler, fakat hiçbir zaman optimal sonucu bulmayı garantilemezler. Bu algoritmalar genellikle çok basittirler ve kısa çalışma sürelerine sahiptirler. Bazı algoritmalar optimal çözümden yaklaşık yüzde birkaç sapma göstermektedir. Eğer bu ufak sapmalar kabul edilecek olursa yaklaşık algoritmalar kullanılmalıdır. Yaklaşık algoritmalar 3 temel başlık altında toplanabilir.

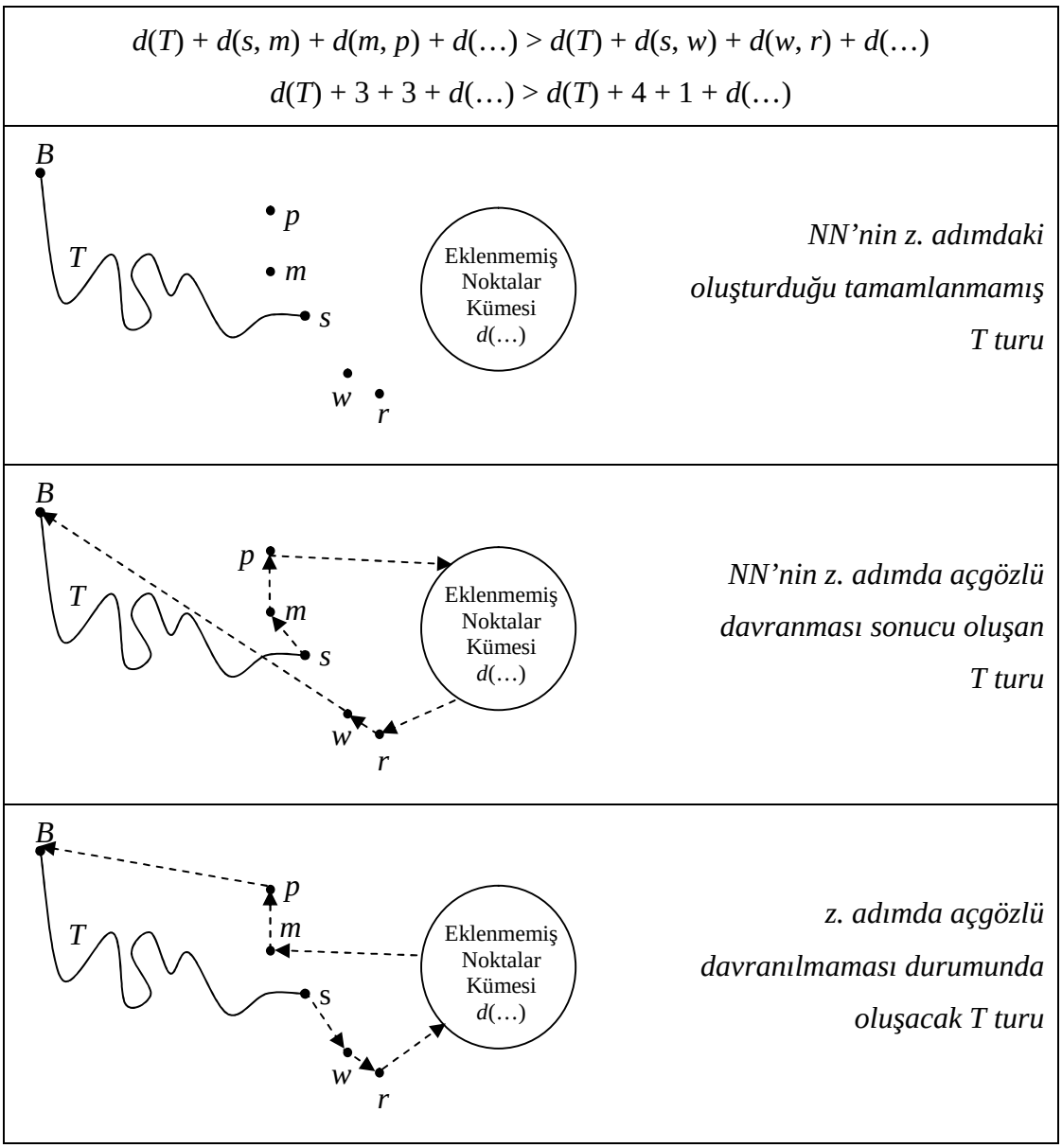
2.2.1. Tur Oluşturma Algoritmaları

Bu algoritmalar tura her adımda bir şehir ekleyerek adım adım turu tamamlarlar. Seçilen algoritmaya göre çok kısa bir sürede ortalama olarak %10-25 arasında optimum değerlere yaklaşırlar (Marinakis ve Migdalas 2005). En basit tur yapılandırma algoritması “En Yakın Komşu” (*Nearest Neighbor*: NN) algoritmasıdır. NN algoritmasında, rasgele bir şehrin boş tura eklenmesiyle başlanır ve her bir adımda son eklenen şehre en yakın şehir tura eklenir. Tüm noktalar eklenip tur tamamlanıncaya kadar bu işleme devam edilir.

NN algoritması da diğer tüm Tur Oluşturma Algoritmaları gibi açgözlü (*greedy*) olduğu için optimal değerden oldukça fazla sapma göstermektedir. Çünkü *z* adımında eklenmiş *s* şehrine en yakın *m* şehrinden sonraki en yakın *p* şehrinin uzunluğu, *s* şehrine en yakın olmayan *w* şehrinden sonraki en yakın *r* şehrinin uzunluğundan fazla olabilir. Açgözlülük diye tabir edilen bu olay Şekil 2.1’de anlatılmaktadır (*B* rasgele seçilmiş başlangıç noktasıdır).

Şekil 2.1’de $d(p, m)=3$, $d(m, s)=3$, $d(s, w)=4$ ve $d(w, r)=1$ olduğu varsayılsın. *T* turuna *z* zamanında eklenen *s* şehrinin en yakın komşusu *m* şehridir. *m* şehri tura

eklendikten sonra, eklenebilecek en yakın komşu *p* noktası olmaktadır. Bu da *z+2* zamanında toplam mesafeyi *d*(*T*) + *d*(*s*, *m*) + *d*(*m*, *p*) + *d*(…) yapmaktadır. *s* noktasının ardından *s*'ye en yakın komşu olmayan *w* şehri eklense idi, ardından da *r* şehri eklenecekti. Bu durumda *z+2* zamanında tamamlanmamış turun toplam mesafesi *d*(*T*) + *d*(*s*, *w*) + *d*(*w*, *r*) + *d*(…) şeklinde oluşacaktı. Aç gözlü davranış örnekteki gibi, sonraki durumları düşünmeden o anda elindeki en iyi gibi gözüken veriler ile çözüm arama işlemidir. Bu yüzden Tur Oluşturma Algoritmaları genellikle çözüm uzayında bir başlangıç metodu olarak kullanılmaktadır.



Şekil 2.1. En Yakın Komşu Algoritmasının Açgözlü Davranışı

2.2.2. Tur Geliştirme Algoritmaları

Turda çeşitli yer değiştirmeler yaparak elde edilen kazanç değerlerine göre turu geliştirmeye çalışan algoritmalardır. Algoritmaya optimize etmesi için verilen turdaki yer değiştirme işlemleri sürekli tekrarlanarak (*iteratif*), yerel minimum çözümler bulununcaya kadar araştırmalara devam edilir. Tur geliştirme algoritmaları "Yerel Arama" (*Local Search*) metotları olarak da bilinir. Temelleri en iyi olmayan komşulukların turdan kaldırılarak, toplam mesafeyi azaltacak (kazanç) şekilde tura yeniden yerleştirilmesidir. Bu yer değiştirme süreci daha iyi komşuluk değişimleri bulunamayıncaya kadar devam edecektir.

Bilinen en genel tur geliştirme algoritmaları turdaki *k*-kenar yer değiştirmeleridir. *k*-kenar metotlarında, *T* turundaki *k* adet kenar turdan kaldırılır ve daha iyi *k* kenar *T* turuna tekrar yerleştirilerek geliştirilmiş yeni bir tur elde edilir. En çok bilinen ve kullanılan tur geliştirme algoritmaları 2-Opt, 3-Opt ve Lin-Kernighan (LK) metotlarıdır. Johnson ve McGeoch (2002) tarafından yapılan analizlerde rasgele üretilen bir GSP'de 2-Opt-JM %4.70 hata (1.41 saniye), 3-Opt-JM %2.28 hata (1.50 saniye) ve LK-JM %2.00 hata (2.06 saniye) değerini vermiştir (JM: Johnson ve McGeoch uygulaması). Lin-Kernighan metodu üzerinde çalışan diğer geliştirilmiş metotlardan en efektif sonucu verenler (Chained Lin-Kernighan: CLK, Lin-Kernighan Helsgaun: LKH) aynı problem üzerinde %0.9 hata (63.91 saniye) ile CLK, %0.69 hata (1840.00 saniye) ile LKH metotlarıdır.

2.2.3. Meta-Sezgisel Algoritmalar

"Meta-Sezgisel, alt sezgisel metotların araştırma sahasının keşfedilip kullanılması için farklı kavramların zekice birleştirilmesini yöneten sürekli oluşum işlemidir. Etkili bir şekilde optimal değere yakın sonuçlar bulmak amacıyla çeşitli öğrenme stratejileri, bilgi yapılandırma için kullanılır." (Osman ve Laporte 1996). Meta-sezgisel algoritmaların bir çok uygulama alanlarında ortaya çıkan zor kombinasyonel optimizasyon problemlerin çözümünde yaygın bir başarısı vardır.

Sezgisel veya yaklaşıma dayalı çözümler veren yöntemler, makul bir sürede en iyi çözüme yakın sonuçlara ulaşılmasını sağlarlar. Sezgisel yöntemlerin en önemlileri, Genetik Algoritmalar, Tavlama Benzetimi, Tabu Arama ve Karınca Kolonisi tabanlı algoritmalardır (Cevre ve ark. 2007).

2.2.3.1. Tavlama Benzetimi

Tavlama Benzetimi (*Simulated Annealing*) algoritması ilk olarak 1953 yılında Metropolis, Rosenbluth ve Teller tarafından bir ısı banyosundaki taneler kümesinin denge dağılımını hesaplamak amacıyla kullanılmıştır (Metropolis ve ark. 1953). Tavlama Benzetimi (TB) "Isıl İşlem" olarak da adlandırılır. TB ilk olarak 1983 yılında optimizasyon problemlerine uygulanmıştır (Kickpatrick ve ark. 1983).

TB, metal malzemelerde katı halde sıcaklık değişmeleri ile bir ya da birbirine bağlı birkaç işlemle amaca uygun özellik değişmelerinin sağlanması olayı şeklinde tanımlanır. Bu tanım gereği ısıl işlem sayesinde, malzemenin belirli bir sıcaklığa kadar ısıtılması (ısıtma), bu sıcaklıkta uygun bir süre tutulması (bekleme) ve belirli bir stratejiye göre sıcaklığın oda sıcaklığına kadar azaltılması (soğutma) sayesinde üç aşamada malzemede özellik değişmeleri sağlanır (Karaboğa 2004).

Kickpatrick arkadaşlarıyla (1983) ve Cenry (1985) TB'nin GSP'ye nasıl uygulanabileceğini göstermişlerdir. Buna göre TB'deki parametrelerin optimizasyon problemleri için dönüşümleri Tablo 2.1'deki gibidir. Ayrıca TB hesaplama devam ettikçe bir komşudan diğer komşuya geçen yerel arama algoritması olarak tanımlanabilir (Johnson ve McGeoch 1997).

Tablo 2.1. Tavlama Benzetiminde GSP Parametre Dönüşümleri

Fiziksel İşlem	Meta-Sezgisel
Enerji	Maliyet Fonksiyonu
Sıcaklık	Kontrol Parametreleri
Donma Noktası	Yaklaşık Çözüm
Sistem Durumu	Uygun Çözüm

TB algoritmasının en genel akış şeması Şekil 2.2'deki gibidir (Johnson ve McGeoch 1997).

1. Bir başlangıç S çözüm kümesi oluşturulur ve şampiyon çözüm $S^* = S$ yapılır.
2. Başlangıç sıcaklığı T belirlenir.
3. Tekrarla ("Donma" olmadığı sürece)
   3.1. $i = 0$
   3.2. Tekrarla // $i = N(t)$
      3.2.1. Şimdiki çözümden rasgele bir S' komşusu seç
      3.2.2. $\Delta = f(S') - f(S)$
      3.2.3. Eğer $\Delta \leq 0$ ise
                  $S = S'$
                  Eğer Uzunluk(S) < Uzunluk(S^*) ise $S^* = S$
      3.2.4. Değilse
            [0-1] arasında rasgele bir r sayısı seç;
            Eğer $r < e^{-\Delta/T}$ ise $S = S'$;
      3.2.5. $i = i + 1$;
      3.3. $i = N(t)$ oluncaya kadar Adım 3.2.'ye git
   3.4. T sıcaklığını düşür
4. Son ("Donma" olmadığı sürece)
5. Sonuç S^*

Tanımlar

$f(x)$	: x çözümü için maliyet
S	: Önceki çözüm
S'	: Sonraki çözüm
S^*	: En Uygun çözüm
$N(t)$	: Aynı sıcaklıktaki bozunum sayısı

Şekil 2.2. Genel Tavlama Benzetimi Algoritması Adımları

Johnson ve McGeoch (1997) tarafından yapılan analizlerde TB ve çeşitli varyasyonları, standart tur geliştirme algoritmaları (2-Opt, 3-Opt ve LK) ile rasgele üretilmiş 100, 316 ve 1000 boyutlu problemlerde analiz edilmiştir. Analize göre 10^3 şehirli bir GSP'de TB %4 hata (3170 saniye), 2-Opt %4.9 hata (0.34 saniye), 3-Opt %3.1 hata (0.41 saniye) ve LK %2 hata (0.77 saniye) değerlerini vermektedir.

Buna göre tur geliştirme algoritmaları, TB algoritması ve varyasyonlarından daha iyi neticeler vermektedir. Ayrıca Ohlidal ve Schwarz tarafından yapılan başka bir analizde, Paralel Tavlama Benzetimi (PTB) algoritması hibrit bir sistem olan Melez Paralel Genetik Tavlama Benzetimi (MGTB) algoritması Berlin52 S-GSP'de karşılaştırılmıştır. Genetik Algoritmalar ile melezlenmiş MGTB 15 farklı denemede ortalama ε_{ORT}=%0.00 hata oranı verirken PTB'nin sadece yerel bir çözüm ($\varepsilon\approx$%6.32) döndürdüğü görülmüştür (Ohlidal ve Schwarz 2004).

2.2.3.2. Tabu Arama

Modern Tabu Arama (TA) formları ilk olarak F. Glover tarafından 1986'da optimizasyon problemlerinin çözümü için geliştirilmiş tekrarlamalı bir araştırma algoritmasıdır (Glover 1989, 1990).

Tabu Arama algoritmasının felsefesi, zeki problem çözme eğiliminden ortaya çıkmıştır. Bundan dolayı TA algoritması, geçmişi hafızasında tutarak yerel minimumdan kolaylıkla çıkar. Her bir hareket yapıldığında tabu listesi denilen listeye yerleştirilir. Tabu listesi, kabul edilen değişimlerin saklanması ve istenmeyen durumun ortadan kaldırılması amacıyla kullanılır.

TA algoritmasının uygulanması genelde çok basittir. Basit bir tabu arama prosedürü, bir başlangıç çözümünden (S^*) başlar, bu çözümü şu andaki durum (S) ve en iyi durum (S^*) olarak hafızaya alır. Şu andaki durum üzerinde çeşitli operatörler kullanılarak komşu çözümler üretilir. Oluşturulan bu komşu çözümler aday çözümlerdir. Aday çözümlerin içerisinden en iyi kabul edilebilir olanı seçilir (tabu listesi ve tabu yıkma kriterlerine bağlı olarak) ve seçilen çözüm (S) tabu listesine eklenir. Yeni bulunan çözüm en iyi çözümden iyi ise, en iyi çözüm şu andaki duruma eşitlenir ($S^* = S$). Tüm bu işlemler durdurma kriterleri sağlanıncaya kadar devam eder. Şekil 2.3'de basit bir Tabu Arama algoritması verilmektedir.

Adım 1.	Başlangıç çözümünü oluştur (S^*) ve şimdiki çözüm (S) olarak ata ($S=S^*$),
Adım 2.	Şimdiki çözüm üzerinde değişim operatörleri kullanarak yeni aday çözümleri oluştur,
Adım 3.	Aday listesinde, Tabu Listesinde olmayan veya Tabu Yıkma Kriterini sağlayan en iyi çözümü ara,
Adım 4.	Adım-3'de bulunan çözüm (S) en iyi çözümden (S^*) daha iyi ise, bulunan çözümü en iyi çözüm olarak ata ($S^*=S$),
Adım 5.	Durdurma kriteri sağlanıncaya kadar Adım 2'ye dön

Şekil 2.3. Genel Tabu Arama Algoritması Adımları

Marinakis ve arkadaşları (2005) tarafından yapılan çalışmada, DIMACS'tan alınmış 6 farklı yerel arama metodu ile kombine edilmiş tabu arama metotlarının, 10 farklı GSP'deki etkisi gösterilmiştir. Tabu aramanın 6 farklı varyasyonundan, en kötü genel ortalama %4.83 ile 2-Opt yerel arama metodunu kullanan TS22, en iyi genel ortalama %1.034 ile *Stem-Cycle* ve *Double-Bridge* kullanılan TSSCDB metodudur. Johnson ve McGeoch (2002) tarafından yapılan analizlerde ise rasgele üretilmiş 10000 şehirli bir problemde Tabu-SC-SC metodunun *Held-Karp* değerinden %1.48 sapma gösterdiği bu değerin 18830 saniye gibi çok uzun bir sürede elde edildiği görülmüştür.

2.2.3.3. Karınca Kolonisi

İlk olarak 1991 yılında Dorigo ve arkadaşları tarafından ortaya konmuştur. Gerçek karınca kolonilerinin davranışlarının matematiksel modellemelerine dayanan bir algoritmadır. Karıncalar, hareket halinde oldukları yola belli bir miktarda fenomen maddesi bırakırlar ki dolaşmak için seçecekleri yolun belirlenmesinde bırakılan fenomen miktarı etkilidir. Belirli bir olasılık dahilinde daha fazla fenomen olan yönler karıncalar tarafından daha az fenomen olan yönlere göre tercih edilirler. Kısa olan yollardan birim zamanda geçen karınca sayısı daha fazla olduğundan yine birim zamanda kısa yolda karıncalar tarafından bırakılan koku miktarı daha fazla olacaktır. Böylece, kısa sürede kolonideki çoğu karınca mesafe bakımından kısa olan

yolu tercih edecektir. Şekil 2.4'de Karınca Çevrim algoritmasının temel adımları verilmektedir (Karaboğa 2004).

Adım 1. Tüm şehirlere belirli miktarda $b_i(t)$ karınca yerleştir. Buna göre karıncaların tabu listesini yenile. Her hattın koku miktarını sıfırla; Sayacı sıfırla;
Adım 2. Tabu listeleri dolana kadar aşağıdaki işlemleri tekrarla
* Her şehirdeki tüm karıncalar için $P_{ij}(t)$ seçilebilirliğine göre hareket etmek için j. şehri seç;
* Karınca k'yı j. şehre hareket ettir ve j. şehri k. Karıncanın tabu listesine dahil et;
Adım 3. Tüm karıncalar için tur uzunluğu hesapla;
Her kenara (t, t+1) zaman aralığında depolanan koku miktarını hesapla;
Adım 4. Her kenar için fenomen değerini hesapla;
Adım 5. Şu ana kadar bulunan en kısa turu hafızaya al;
* Durdurma kriteri sağlanıyorsa Adım 6'ya git;
* Tüm tabu listelerinin boşalt;
* Tüm şehirlere belirli miktarda karınca yerleştir;
* İkinci adıma git;
Adım 6. Dur.

Şekil 2.4. Karınca Çevrim Algoritmasının Temel Adımları

Dorigo ve Gamberdella (1997) yaptıkları çalışmada 100 şehre kadar olan küçük GSP'lerde optimum değeri yakalarken, 198 şehirden 1577 şehre kadar yaptıkları büyük GSP analizlerinde hata oranının %0.68'den %3.27'ye kadar yükseldiği görülmüştür. Ayrıca Rat778 problemi için %2.37'lik bir hatayı 991276. üretilen kuşakta, her bir üretme süresi olarak 0.13 saniye harcayarak elde etmiştir ki bu da oldukça uzun bir çevrim süresidir. Ayrıca Dorigo ve Gambardella (1997) yaptıkları bir diğer çalışmada karınca koloni sistemi ile 3-Opt yerel arama metoduyla birleştirmiş Rat778 için hata oranını %0.36'ya 1317 saniyede düşürmüşlerdir. Ayrıca ACS-3-Opt metodunu Freisleben ve Merz (1997) tarafından geliştirilen Genetik Yerel Arama (GYA) algoritması ile kıyaslamışlar ve tüm problemlerde (D198, Lin318, Att532 ve Rat783) GYA'dan yaklaşık %10 civarında daha kötü sonuçlar elde etmişlerdir. Ayrıca Stützle ve Hoos (1997) tarafından *Max-Min Ant System*

(MMAS) ile 2-Opt yerel arama metodu birleştirildiğinde Lin318 GSP problemi için 43158 mesafe değeri (ε = %2.686) 500 saniyede elde edilmiştir.

2.2.3.4. Genetik Algoritmalar

Genetik Algoritmalar (GA), doğal seçimin mekaniklerine ve doğal genetiklerin en uygun konseptlerinin hayatta kalmasına dayanır (Holland 1975). Doğal evrimi, doğrusal olmayan araştırma alanına ve büyük problemlere en iyi çözümleri bulmak hedefiyle benzetirler. Kısaca GA, seçilen problem için oluşturulmuş bir uygunluk fonksiyonu ile popülasyondaki tüm bireylerin ölçeklendirilip, problemi en iyi ifade eden bireylerin çiftleştirilmesi (çaprazlama ve mutasyon) için birtakım operatörlerden faydalanarak anne-babaya benzeyen ve onların gen özeliklerini taşıyan yeni nesillerin üretilerek global optimuma yaklaşılması işlemidir. Holland tarafından ortaya konan GA, Basit Genetik Algoritma (BGA) olarak isimlendirilir ve Şekil 2.5'deki adımları içerir.

```
Alternatif çözümler içeren başlangıç popülasyonu (POP) oluşturulur;
Popülasyondaki her bir bireyin Uygunluk değeri hesaplanır;
Belirlenen durdurma kriteri sağlanıncaya kadar aşağıdaki işlemler
tekrarlanır;
      Üretme işlemi için popülasyondan birey seç;
      Seçilen bireylerden P_C ihtimalinde yeni nesiller üret;
      Yeni nesilleri P_M ihtimalinde Mutasyona uğrat;
      Yeni nesillerin uygunlukları değerlendirilir
      Yeni nesiller için popülasyondan yer açılır ve yeni iyi
nesiller eklenir
Popülasyondaki en iyi birey sunulur.
```

Şekil 2.5. Basit Genetik Algoritma Adımları

Genetik Algoritmalarda her bir operatörün etkisi, çözümün kalitesini önemli ölçüde etkilemektedir. Seo ve Moon (2002) GA için geliştirdikleri VQX çaprazlama operatörünü ve LK yerel arama algoritmasını birleştirerek (Genetik Yerel Arama) Att532, Dsj1000, D2103, Pcb3038 ve Fnl4461 GSP problemleri üzerinde denemeler

yapmışlardır. Att532, Dsj1000 ve D2103 problemleri için optimum değerleri (%0.00) elde ederken 4461 şehirli Fnl4461 problemi için %0.003 gibi optimuma çok yakın değer elde etmişlerdir. Ayrıca Nagata (2006) tarafından yapılan bir diğer çalışmada 4461 şehirli problemden 33810 şehirli GSP problemlerine kadar 20 farklı problemde EAX çaprazlama operatörü CLK (*Chained Lin-Kernighan*) yerel arama metodu ile uygulanmış ve çok iyi sonuçlar alınmıştır.

3. GENETİK ALGORİTMALAR

3.1. Genetik Algoritmanın Tarihçesi

Genetik Algoritma tekniği, Michigan Üniversitesinde psikoloji ve bilgisayar bilimi uzmanı olan John Holland ve çalışma arkadaşlarının liderliğinde "*Cellular Automata*" ile ilgili çalışmalar sonucunda ortaya çıkmıştır. Makine öğrenmesi konusunda çalışan Holland, Darwin'in evrim kuramından etkilenerek, canlı mekanizmalarda yaşanan genetik süreci bilgisayar ortamında gerçekleştirmeyi düşünmüştür. Tek bir yapının öğrenme yeteneğini geliştirmek yerine böyle yapılarda oluşan bir popülasyonun seçim, çiftleşme, mutasyon, vb. genetik süreçlerden geçerek başarılı (öğrenebilen) yeni bireyler oluşturabildiğini gördü. Araştırmalarını, arama ve optimizasyon işlemleri için, doğal seçme ve genetik evrimden yola çıkarak yapmıştır. İşlem boyunca, biyolojik sistemlerin çevreye uyum sağlayıp daha uygun hallere gelmesi baz alınarak, en iyiyi (optimum) bulma ve makine öğrenme problemlerinde, bilgisayar yazılımı modellenmiştir.

Geliştirdiği yöntemdeki çalışmaların sonucunu içeren kitabın 1975'te yayınlanmasından sonra yöntem Genetik Algoritma (GA) olarak yerleşti. Ancak 1985 yılında Holland'ın öğrencisi olarak doktorasını veren David E. Goldberg adlı inşaat mühendisi, 1989 yılında konusunda bir klasik sayılan kitabını yayınlayana dek GA'nın, pek pratik yararı olmayan bir araştırma konusu olduğu düşünülüyordu. Günümüzde GA, optimizasyon, otomatik programlama ve bilgi sistemleri, mekanik öğrenme, ekonomik ve sosyal sistem modelleri, finans, pazarlama, montaj hattı dengeleme problemi, çizelgeleme problemi, tesis yerleşim problemi, atama problemi, hücresel üretim hattı problemi, sistem güvenliği problemi, taşıma problemi, gezgin satıcı problemi, araç rotalama problemi, minimum yayılan ağaç problemi gibi birçok alanda uygulanmaktadır (Emel ve Taşkın 2002).

3.2. Genetik Algoritmanın Tanımı

Genetik Algoritma, doğal seçim ilkelerine dayanan, ihtimallere göre tekrarlamalı olarak çalışan bir arama ve optimizasyon yöntemidir. GA, doğada en iyinin yaşaması kuralına bağlı olarak sürekli iyileşen çözümleri üretir. Burada "iyi" kelimesinin ne olduğunu belirleyen bir uygunluk fonksiyonu (*fitness function*), probleme göre dikkatli bir şekilde hazırlanmalıdır. Mevcut çözümler uzayından yeni çözümler üretmek için seçim, eşleştirme, çaprazlama, mutasyon gibi operatörleri kullanır. GA'nın bir diğer önemli özelliği de çözüme, küçük bir alt grup çözüm kümesiyle ulaşmaya çalışmasıdır. Yani GA çözüm uzayının tamamını değil, belirli bir kısmını tararlar. Böylece, etkin arama yaparak çok daha kısa bir sürede çözüme ulaşırlar (Goldberg 1989).

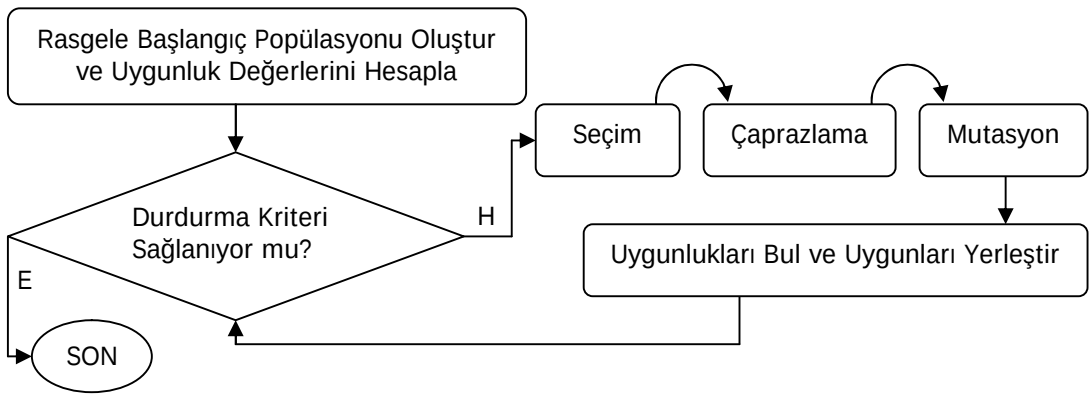
Genetik Algoritmayı diğer meta-sezgisel algoritmalardan ayıran en önemli özelliklerinden biri seçim mekanizmasıdır. Genetik Algoritmada popülasyonda yer alan çözümlerin amaç fonksiyonu tarafından belirlenmiş olan uygunluğu dâhilinde seçilme şansı artmakta fakat bu garanti edilmemektedir. Seçim mekanizması da popülasyonun oluşturulması gibi rasgeledir, ancak bu rasgele seçimde seçilme olasılıklarını çözümlerin uygunluğu belirler. GA'ları diğer metotlardan ayıran noktalar şu şekilde sıralanabilir (Haupt ve Haupt 2004):

- Sürekli veya ayrık parametreleri optimize etmesi,
- Türevsel bilgilere ihtiyacının olmaması,
- Amaç fonksiyonunu geniş bir alanda araması,
- Değişkenlerin büyük bir bölümüyle çalışabilmesi,
- Paralel bilgisayarlar kullanılarak çalışabilmesi,
- Son derece karmaşık maliyet yüzeyinde değişkenleri yerel optimumdan çıkararak optimize etmesi,
- Tek çözüm yerine optimumların bir listesini vermesi,
- Nümerik üretilmiş veriler, deneysel veriler ya da analitik fonksiyonlarla çalışabilmesidir.

Emel ve Taşkın'a (2002) göre GA'nın çalışma adımları kısaca şöyle açıklanabilir:

- Arama uzayındaki tüm mümkün çözümler birey olarak kodlanır.
- Genellikle rastsal bir çözüm kümesi seçilir ve başlangıç popülasyonu olarak kabul edilir.
- Her bir birey için bir uygunluk değeri hesaplanır, bulunan uygunluk değerleri bireylerin çözüm kalitesini gösterir.
- Bir grup birey belirli bir olasılık değerine göre rastsal olarak seçilip çoğalma işlemi gerçekleştirilir.
- Yeni bireylerin uygunluk değerleri hesaplanarak, çaprazlama ve mutasyon işlemlerine tabi tutulur.
- Önceden belirlenen kuşak (jenerasyon) sayısı boyunca yukarıdaki işlemler devam ettirilir.
- İterasyon, belirlenen kuşak sayısına ulaşınca işlem sona erdirilir. Amaç fonksiyonuna göre en uygun olan çözüm seçilir.

Yukarıda tanımlanan aşamalar, mevcut çözümleri değiştirip birleştirmek olarak görülebilir. GA'nın oluşturulmasında ve tekrarlamalı bir şekilde çalışması sırasında bileşenlere ihtiyaç duyulmaktadır. En basit haliyle GA akış diyagramı Şekil 3.1'de gösterildiği gibidir.



Şekil 3.1. Basit Genetik Algoritma Akış Diyagramı

3.3. Genetik Algoritmanın Bileşenleri

Genetik Algoritma, problem ile ilgili optimal bir çözümü ararken birçok işlem gerçekleştirir. Çözüm arama işlemi için Genetik Algoritmanın bazı altyapı kriterlerinin de problem alanına uygun olarak seçilmesi gerekmektedir. Genetik Algoritmalar ile çözüm aranacak bir problemde belirlenmesi gereken bileşenler aşağıdaki gibidir.

- Kodlama
- Sunum
- Başlangıç Popülasyonu
- Seçim
- Eşleştirme
- Çaprazlama Metotları
- Mutasyon Metotları
- Seçkinlik
- Durdurma Kriterleri

Yukarıda belirtilen GA bileşenlerinin her birisi problemin çözümü aşamasında oldukça önemlidir. Fakat bu bileşenler haricinde çözüm için oldukça önemli parametrelerin de belirlenmesi gerekmektedir. Popülasyon sayısı, çaprazlama oranı ve mutasyon oranı gibi parametreler de çözüm için oldukça önemlidir.

3.3.1. Kodlama

Seçilecek olan kodlama planı GA’nın önemli bir kısmını teşkil eder. Çünkü bu plan, bilginin çerçevesini şiddetli bir şekilde sınırlayabilir. Kodlamayla, probleme özgü bir bilginin kromozomsal gösterimi, temsili olarak sağlanır. Kromozom genellikle, problemdeki değişkenlerin belli bir düzende sıralanmasıdır. Kromozomu oluşturmak için sıralanmış her bir değişkene “gen” adı verilir. Buna göre bir gen kendi başına anlamlı genetik bilgiyi taşıyan en küçük genetik yapıdır.

Bu bilgileri en küçük genetik yapı olan gen içerisinde muhafaza ederken dikkat edilmesi gereken husus kodlamanın nasıl yapıldığıdır. Çünkü GA'nın performansına etki eden faktörlerden birisi problemin çözümünü göstermek için seçilen kodlama tipidir (Karaoğlan ve Altıparmak 2005). GA için İkili, Permutasyon, Değer ve Ağaç kodlama şeklinde 4 farklı kodlama kullanılabilir.

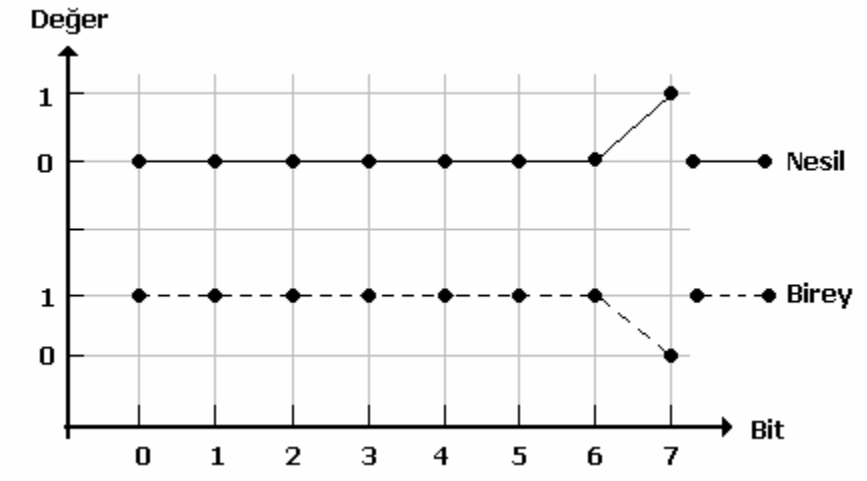
3.3.1.1. İkili Kodlama

Bu yöntem ilk GA uygulamalarında kullanıldığı için hala en çok kullanılan yöntemlerdendir. Burada her kromozom 0 ve 1'lerden oluşan bit dizisidir ve ikili diziyle ifade edilir. Bu dizideki her bit, çözümün bir özelliğini taşır. Dizinin tümü ise bir sayıya karşılık gelir.

Kromozom A: 10111010111001010
Kromozom B: 01101011101011101

İkili kodlama, az eleman sayılı problemlerde bile çok sayıda mümkün kromozom oluşturur. Bu kodlama ile GSP gibi bazı problemlerde çaprazlama ve mutasyon işlemlerinden sonra düzeltmeler yapılmak zorunda kalınabilir.

İkili kodlamanın önemli dezavantajlarından bir tanesi de "*Hamming Mesafesi*"dir. Çözüm uzayında bir bireyden komşu bireyler (Şekil 3.2) üretilmesi için tüm bitlerin değişmesi gerekebilir. Hamming Mesafesi (HM), birbirine benzemeyen bitlerin sayısıdır. Şekil 3.2'deki örnek üzerinde iki birey arasındaki HM=8'dir. Birçok mühendislik probleminde ikili kodlama kullanılarak bireylerin çözümlerinin sunulması hemen hemen imkânsızdır (Gen ve Cheng 2000). Ayrıca GA operatörlerinden sonra fazla çevrim gerektiren kontrol işlemlerinin yapılmasıdır. Bu yüzden GSP için pek tercih edilmemektedir.



Şekil 3.2. Çözüm Uzayında Bir Bireyin Komşusu ile Hamming Mesafesi

3.3.1.2. Permutasyon Kodlama

İş sıralama ve GSP gibi sıralama problemlerinde oldukça yaygın olarak kullanılır. Burada her kromozom bir numaralar dizisinden oluşmaktadır.

Kromozom A: 35214687
Kromozom B: 12345678

Pemutasyon kodlama bazı kaynaklarda "Tamsayı (*Integer*)" kodlama olarak da adlandırılmaktadır. GSP'de dolaşılacak olan şehirlerin kromozomlarda tamsayı olarak kodlanması, ziyaret edilecek bir sonraki şehrin indisini ifade eder.

3.3.1.3. Değer Kodlama

Değer kodlama, reel ve kompleks sayıların yer aldığı problemlerde kullanılır. Reel veya kompleks sayıların kullanıldığı problemlerde ikili kodlamanın kullanılması çok zor olur (Nabiyev 2003). Burada her kromozom, problemin çözümü ile ilgili değerleri içeren gen dizileridir. Örneğin reel sayı, karakter veya bazı kompleks nesneler bile olabilir.

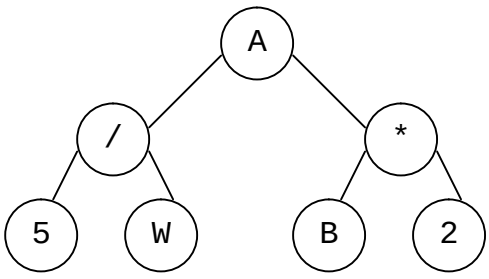
Kromozom A: 3.255, 6.257, 8.256, 1.290

Değer kodlama, ekseriyetle özel problemlerde kullanılır. Yapay Sinir Ağlarının katsayılarının hesaplanması buna örnek olarak verilebilir. Snyder ve Daskin (2006) yaptıkları çalışmada GSP için Rasgele Anahtarlar (*Random-Keys*) kullanmışlar ve sonuçlar değer kodlamanın da GSP için kullanılabilirliğini göstermektedir. GSP'de değer kodlama sunumu için *n* tane şehrin kayan noktalı sayılarla kodlanmasını (*encode*) ve işlenmesini gerekir (Chatterjee ve ark. 1996). Sonra da kodlanmış bilgilerin kodunun (*decode*) çözülmesi gibi bileşenler içerir. Değer kodlamanın dezavantajları, çevrim süresini ve bellek kullanımını (kayan noktalı sayılardan dolayı) arttırmalarıdır.

3.3.1.4. Ağaç Kodlama

Ağaç kodlama genellikle genetik programlamada, programlar ve ifadeler oluşturmak için kullanılır. Ağaç kodlama her kromozom, adından da anlaşılacağı gibi nesneler ve nesneler arası işlemleri içeren bir ağaç yapısından oluşmaktadır (Şekil 3.3).

Ağaç kodlama, program geliştirmek için oldukça uygundur. Bu kodlamada çaprazlama ve mutasyon çok kolay bir şekilde uygulanır. Örneğin televizyonlarda çıkan, verilen bir dizi sayıyı birer kere kullanarak belirlenmiş bir sonucu elde etmede kullanılabilecek bir kodlama şeklidir.



Şekil 3.3. Ağaç Kodlama

3.3.2. Sunum

GSP'de genellikle permutasyon kodlama kullanılmasının yanında, turun kromozom içerisinde hangi düzende sunulacağının da belirlenmesi gerekmektedir. Literatürde bunun için kullanılan farklı yaklaşımlar mevcuttur.

3.3.2.1. Yol Sunumu

Yol sunumu, en doğal ve başarılı sunumlardan birisidir. Her gen bilgisinde bulunan bilgiler şehirlerin indisleridir. Kromozomda bulunan genler (şehir) sırası ile dolaşılır. [1-9-3-5-7-2-6-4-8] şeklindeki bir sunumda $1\rightarrow9\rightarrow3\rightarrow5\rightarrow\ldots\rightarrow4\rightarrow8$ şeklinde bir dolaşım söz konusudur. Klasik operatörlerin kullanımı yol sunumu için uygun olmasa da (illegal tur oluşumlarından dolayı), bu sunum için PMX (Goldberg ve Lingle 1985), OX (Davis 1985), CX (Oliver ve ark. 1987), ERX (Whitley ve ark. 1989), EAX (Nagata ve Kobayashi 1997), VQX (Seo ve Moon 2002) ve NX (Jung ve Moon 2002) gibi birçok operatör geliştirilmiştir.

3.3.2.2. Bitişik Sunum

Bitişik sunumda, kromozomdaki her gen turdaki bir kenarı temsil eder. Eğer turdaki *m.* pozisyondaki değer *k* ise kenar (*m*, *k*)'dır. $1\rightarrow8\rightarrow3\rightarrow5\rightarrow7\rightarrow2\rightarrow6\rightarrow4$ şeklindeki bir tur [8-6-5-1-7-4-2-3] şeklinde sunulur. Bitişik sunumda klasik operatörler illegal tur sonuçları oluşmaktadır. Bitişik Sunum'un en büyük dezavantajı, GSP turunun sadece bir şekilde sunulabilmesidir. Bu sunum için Grefenstette ve arkadaşları (1985) tarafından çeşitli çaprazlama metotları sunulmuştur.

3.3.2.3. Sıra Sunum

Sıra sunumunda tutulan bir Referans Listesi (RL) ile tüm şehirlerin listesi tutulur. Sıra sunumunda yer alan her gen, referans listesindeki kaçıncı gene (şehir) gidileceğini göstermektedir. Gidilecek olan şehir referans listesinden kaldırılır. RL={1,2,3,4,5,6,7,8,9} ve sıra sunumu ile sunulan tur T=[4-5-2-1-3-1-2-2-1]

şeklinde olsun. Turun ilk indisi 4 olduğu için referans listesindeki dördüncü elemana gidilecektir ve referans listesinden {4} kaldırılacaktır. Geriye kalan 8 elemanlı referans listesinden seçilecek olan şehir turun ikinci işaretçisi (5) ile belirlenir. Bu şekilde yapılacak olan dolaşma 4→6→2→1→7→3→8→9→5 şeklinde olacaktır. Sadece klasik operatörlerde kullanılması, illegal turlar oluşturmamasından dolayı avantajlı iken, deneysel performansları çok iyi değildir. Çünkü kesme noktasından itibaren ikinci ebeveynden gelecek olan genler rasgelelik içerecektir.

3.3.3. Başlangıç Popülasyonu

Basit Genetik Algoritmalar (BGA) genellikle rasgele bir başlangıç popülasyonu oluşturarak problem çözümüne hazırlanır. BGA ile GSP optimizasyonu için çeşitli Tur Oluşturma Metotlarıyla oluşturulmuş uygun kalitedeki bir başlangıç popülasyonu rasgele üretilmiş popülasyonlardan daha kısa çevrim süresi ve uygun çözümler vermektedir. Başlangıç popülasyonun kalitesi, sonuç popülasyonunun kalitesini tanımlamada oldukça önemli bir seçimdir. Jayalakshmi ve arkadaşları (2001) tarafından yapılan çalışmada Eil101 ve D198 GSP problemleri için Tur Oluşturma Metotlarıyla üretilmiş başlangıç popülasyonun rasgele üretilmiş bir başlangıç popülasyonundan hem optimuma ulaşmadaki hata değerinde, hem de çevrim süresi bakımından daha iyi sonuçlar verdiği görülmüştür.

Aynı zamanda Tur Oluşturma Metotlarıyla oluşturulan bir başlangıç popülasyonunda, bireylerin benzerlik oranlarının artmasından dolayı çözüm sürecinde yerel optimumlara takılma olasılığının yükseltmesi de önemli bir dezavantajdır. Bu yüzden başlangıç popülasyonunun sadece Tur Oluşturma Metotları ile oluşturulması değil rasgele düzende üretilmiş bireyleri de içermesi BGA'nın başarısını arttırmaktadır.

Popülasyon sayısı da başlangıçta belirlenmesi gereken parametrelerden bir tanesidir. Popülasyon sayısının düşük seçilmesi örnek uzayı yeterince kapsamamasından ve araştırmayı kısıtlamasından dolayı GA'yı çok hızlı bir şekilde yerel optimumlara takılmaya sevk eder. Daha büyük seçilecek popülasyon

büyüklüğü, global optimumu bulmayı kolaylaştırmakta fakat optimizasyon süresini uzatmaktadır (Sastry ve ark. 2005). Araştırmalar, probleme dayanan belli bir sınırdan sonra popülasyon sayısının arttırılmasının faydalı olmadığını göstermektedir.

En iyi popülasyon büyüklüğü seçimi, genellikle uygulamasal olarak belirlenebilir ve kromozom uzunluğu (GSP'deki toplam şehir sayısı) ile alakalıdır. Çoğu araştırmacı küçük boyutlu GSP problemlerinde 20 ile 100 arasındaki popülasyon büyüklüğünü önermektedir.

3.3.4. Seçim

Başlangıç popülasyonu, uygunlukları bulunduktan sonra bir sonraki nesli oluşturmak için çaprazlama ve diğer operatörlere maruz kalacaktır. Popülasyondaki bireylerin en uygun olanları hayatta kalmalı diğerleri yok olmalıdır. Bu noktada çeşitli tekniklere göre seçilen bireyler sanal bir eşleşme havuzu içerisinde genetik operatörlerin uygulanması için toplanmalıdır. Eşleşme havuzunun büyüklüğü parametrik olarak belirlenebilir (Weise 2007). Çaprazlama ve mutasyon operatörleri yeni araştırma noktalarını yönsüz bir şekilde araştırırken, seçim araştırmanın yönünü belirler (Zhang ve Kim 2000). Dolayısıyla seçim işleminde kullanılacak metotlar genetik algoritmanın çözüme gitmesini önemli ölçüde etkilemektedir.

3.3.4.1. Ağırlıklı Seçim

İlk olarak amaç fonksiyonunda, kromozomların uygunluk değerleri hesaplanır. Hesaplanan uygunluk değerleri en küçükten en büyüğe doğru sıralanır. Belirlenen yüzdelik seçim oranına göre en iyi bireyler seçilerek eşleştirme havuzuna alınır.

3.3.4.2. Sıralama Seçimi

Bu seçim metodunda popülasyonda yer alan tüm bireyler uygunluk değerlerine bakılmaksızın eşleşme havuzuna alınır. Bu metodun bir faydası, yerel optimumlardan çıkabilmeyi sağlamasıdır. Dezavantajı ise çözüme daha uzun

zamanda ulaşabilmesidir (Nabiyev 2003). Burada oluşturulan havuz sayısı popülasyondaki birey sayısı kadardır.

3.3.4.3. Rasgele Seçim

Bu seçim stratejisinde herhangi bir kısıtlama ölçüsü yoktur. Popülasyon içerisindeki bireyler rasgele olarak havuza alınır. Uygunluk yönünden kötü olan bireylerin birden fazla seçilerek sonraki nesilde yer alma olasılığının artması dezavantajıdır.

3.3.4.4. Turnuva Seçim

Bu yöntemde bir grup *k* bireyi rasgele olarak seçilir. Popülasyondan rasgele seçilen *k* adet birey daha sonra bir turnuvaya katılır ve en iyi uygunluklu birey seçilir ($2 \leq k \leq$ Popülasyon Sayısı). Turnuva seçiminin avantajı, popülasyonun daha kötü bireylerinin seçilmeyecek olması ve bundan dolayı sonraki neslin genetik yapısına katılmayacak olmasıdır. Ayrıca en iyi bireyler tekrar üretim sürecinde daima baskın olmayacaktır.

3.3.4.5. Rulet Tekerleği Seçimi

Rulet Tekerleği seçim metodunun uygulanması için, ilk önce popülasyondaki bireylerin toplam uygunluk değeri hesaplanır. Kromozomlar, toplam uygunluk değerine bölünerek her bir kromozom için [0–1] arasında değişen seçim ihtimalleri (p_i) aşağıdaki gibi bulunur (3.1).

$$p_i = \frac{f_i}{\sum_{J=1}^{N} f_J} \tag{3.1}$$

$$Kp_i = \sum_{J=1}^{i} p_J \tag{3.2}$$

i : Popülasyondaki i. birey

f_i : i. bireyin uygunluk değeri

N : Popülasyon sayısı

p_i : i. bireyin seçilim ihtimali

Kp_i : i. bireyin kümülatif seçim ihtimali

Daha sonra, her bireyin kümülatif ihtimalleri (Kp_i) hesaplanır (3.2). Popülasyon sayısı kadar [0-1] arasında rasgele sayılar üretilir. Üretilen rasgele sayı, birinci kromozomun kümülatif seçim ihtimalinden küçük ise, birinci kromozom seçilir. Seçim sağlanmıyor ise, ikinci kromozomun veya diğerlerinin kümülatif ihtimalleriyle karşılaştırılarak hangisinden küçükse o kromozom seçilir. Böylece Rulet Tekerleği seçimi gerçekleştirilmiş olur.

Burada, GSP problemleri için dikkat edilmesi gereken önemli bir nokta vardır. Amaç fonksiyonundan alınan küçük uygunluk değerlerine sahip olan bireyler çözüme daha yakın bireyleri ifade etmektedir. Yani uygunluk değeri daha küçük olan bireylerin seçilme ihtimali daha yüksek olmalıdır. Bu yüzden (3.2)'de gösterilen seçilme ihtimali (p_i) üzerinde bir değişiklik yapmak gerekmektedir. Bu işleme "Minimumu Maksimize Etme" denilmektedir. Bu işlem en kolay olarak sonsuz bir sayıdan (∞) uygunluğun çıkartılması veya (-1) ile çarpılmasıyla elde edilebilir (3.3). Böylece minimum değerlere sahip olan bireylerin seçilme olasılığı yükseltilmiş olur.

$$p_i = \frac{\infty - f_i}{\sum_{J=1}^{N} (\infty - f_J)} \tag{3.3}$$

3.3.5. Eşleştirme

Yeni nesillerin üretilmesi işlemi için eşleşme havuzundaki bireyler karşılıklı olarak eşleştirilmelidir. Bu şekilde iki yeni nesil üretilmektedir. Eşleştirme için kullanılan farklı yöntemler mevcuttur.

3.3.5.1. Yukarıdan Aşağıya Doğru Eşleştirme

Eşleştirme işlemi havuzdan ardışık olarak gerçekleştirilir. Böylece $Birey_{2i-1}$ ile $Birey_{2i}$ eşleştirilir. Örneğin $Birey_1$ ile $Birey_2$, $Birey_3$ ile $Birey_4$ eşleştirilir ve genetik operatörlere sunulur (Haupt ve Haupt 2004).

3.3.5.2. Rasgele Eşleştirme

Bireylerin eşleştirilmesinde rasgele üretilen sayılar ve eşleştirme havuzu büyüklüğü (HB) kullanılır. Rasgele sayılar [0-1] aralığında üretilir. Üretilecek her iki sayı eşleştirmeyi belirleyecektir (Haupt ve Haupt 2004).

$Birey_i$ = Üste Yuvarla (HB * (*Rasgele Sayı*))
$Birey_j$ = Üste Yuvarla (HB * (*Rasgele Sayı*))

3.3.5.3. Ağırlıklı Rasgele Eşleştirme

Bu yöntem, eşleştirme havuzundaki bireylerin, amaç fonksiyonunda hesaplanan uygunluk değerlerine göre seçilme ihtimallerini dikkate alır. En düşük uygunluk değerine (en kısa mesafe) sahip birey, eşleştirme ihtimali en yüksek birey olurken, en yüksek uygunluk değerine (en uzun mesafe) sahip birey eşleştirme ihtimali en düşük olarak düşünülür. Üretilen rasgele sayı, hangi kromozomun seçileceğini belirler. Bu yöntem kendi içinde ikiye ayrılır (Haupt ve Haupt 2004).

Sıra Ağırlık: Havuzdaki bireyler uygunluk değerleri bakımından büyükten küçüğe doğru sıralanır ve aşağıdaki denklem kullanılarak bireylerin sıralamasındaki yerlerine göre p_i olasılıkları (3.4) ve kümülatif ihtimalleri (3.3) hesaplanır.

$$p_i = \frac{HB - N + 1}{\sum_{j=1}^{HB} j} \tag{3.4}$$

Rasgele üretilen bir sayı, bireylerin kümülatif ihtimallerinin ardışık sırada toplanmasını geçtiği anda birey seçilmiş olur. GSP problemleri için ilk başta söylenen havuzun sıralama işlemi ters olarak (küçükten büyüğe doğru) yapılmalıdır. Çünkü uygunluk değeri 1234 olan bir birey, 1345 olan bir bireyden daha iyi bir çözümdür.

Değer Ağırlık: Bireylerin uygunluk değerlerine göre her bir bireyin eşleşme ihtimalleri hesaplanır. Eşleşme ihtimallerinin hesaplanmasında uygulukların GSP problemine göre normalize edilmesi gerekmektedir (3.5). Üretilen rasgele sayıların kümülatif eşleşme ihtimallerine gelmesi durumuna göre havuzda yer alan bireylerin eşleştirme indisleri elde edilir.

$$p_i = \frac{\infty - f_i}{\sum_{J=1}^{HB} (\infty - f_J)} \tag{3.5}$$

3.3.5.4. En İyi – En Kötü Eşleştirmesi

Havuzdaki en iyi bireyler ile en kötü bireyler eşleştirilir. Bu yöntemde uygunluk bakımından en kötü olan bireylerin de aslında iyi bir sonuca götürebileceği düşünülür (Türkoğlu 2001). En iyi birinci birey ile en kötü birey, en iyi ikinci birey ile en kötü ikinci birey ve en iyi *n*. birey ile en kötü *n*-1. birey eşleştirilir.

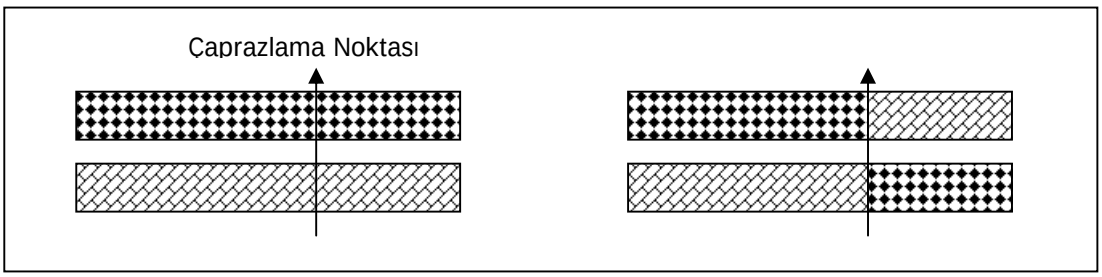
3.3.5.5. İmparator Eşleştirmesi

Havuzdaki uygunluk bakımından en iyi olan birey, geriye kalan tüm bireylerle sırasıyla eşleştirilir. Yeni nesillerin en iyi bireye daha fazla benzer özellikte gelmesi için kullanılır (Türkoğlu 2001).

3.3.4. Çaprazlama

Yeni nesiller, eşleşme havuzundan seçilen iki birey (anne ve baba) tarafından genetik operatörler yardımı ile üretilir. Çaprazlama operatörü, çaprazlama olasılığı ($P_Ç$) dâhilinde seçilen bireylerin karakteristik özelliklerinin taşıyan yeni nesiller üretme işlemi olarak tanımlanabilir.

GA'nın performansını etkileyen önemli parametrelerden biri de çaprazlama operatörüdür (Emel ve Taşkın 2002). Çaprazlamaya tabi tutulan iki bireyden seçilen çaprazlama metoduna göre bir veya iki nesil üretilmektedir.



Şekil 3.4. Basit Tek Noktalı Çaprazlama Metodu

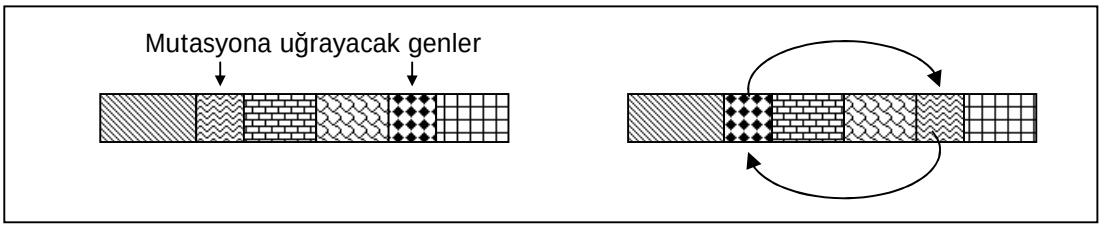
GA'daki en basit çaprazlama operatörü Tek Noktalı çaprazlamadır. Rasgele bir sayı üretilir ve eşler arasında seçilen bölgeler karşılıklı olarak yer değiştirerek yeni nesiller üretilir (Şekil 3.4). Fakat basit operatörler GSP için uygun sonuçlar vermezken daha karmaşık yapıda olan sezgisel yapıyla geliştirilmiş çaprazlama operatörleri daha uygun sonuçlar verebilmektedir. GSP için kullanılan çaprazlama metotları yapı olarak pozisyon, sıra ve kenar temelli olarak gruplandırılabilir.

Çaprazlama işlemi, seçilen bir çaprazlama ihtimaline göre yapılmaktadır. Çaprazlama ihtimali ($P_Ç$) BGA'larda genellikle araştırma süresince sabit kalmaktadır. GSP problemlerinde $P_Ç$ çaprazlama olasılığının ne seçileceği genellikle denemelerle bulunabilir. Çalışmalarda çaprazlama olasılığının yaklaşık [0.5-0.85] arasında seçildiği görülmektedir (Freislenben ve Merz 1996, Ray ve ark. 2005). Fakat araştırma süresinde çözüm kümesi değişeceği için operatör parametrelerinin de

değişmesi performansı arttıracaktır (Karaboğa 2004). Operatör parametrelerinin çözüm sürecinde problemin o andaki karakteristiğine göre kendi kendini ayarlaması adaptif metot olarak tanımlanır. Çaprazlama ve mutasyon oranlarının en uygun değer seçilmesi için araştırma süresince kendini modüle eden bulanık mantık temelli gelişmiş melez adaptif sistemler de mevcuttur (Amraii ve ark. 2007).

3.3.5. Mutasyon

Genetik Algoritmaların diğer bir önemli bileşeni mutasyon operatörleridir. Mutasyon operatörleri ile doğal genetik mutasyon olayının bir benzetişimi yapılmaktadır. Mutasyon, çaprazlama işleminden sonra belirli bir mutasyon ihtimali (P_M) dâhilinde uygulanır. En basit mutasyon operatörü rasgele belirlenen iki gendeki bilginin Yer Değiştirmesidir (Şekil 3.5). Mutasyon operatörü ile algoritma daha geniş bir parametre uzayında arama işlemi yapar. Böylece yeni, görülmemiş ve araştırılmamış çözümlerin bulunmasını sağlar. GSP problemleri için kullanılan birçok mutasyon operatörü vardır. Bazı çalışmalarda tur geliştirme metotları (2-Opt, 3-Opt), mutasyon operatörü olarak veya sonrasında kullanılarak daha iyi sonuçlar alınabilir (Sengoku ve Yoshihara 1998, Liu ve Kang 2003).

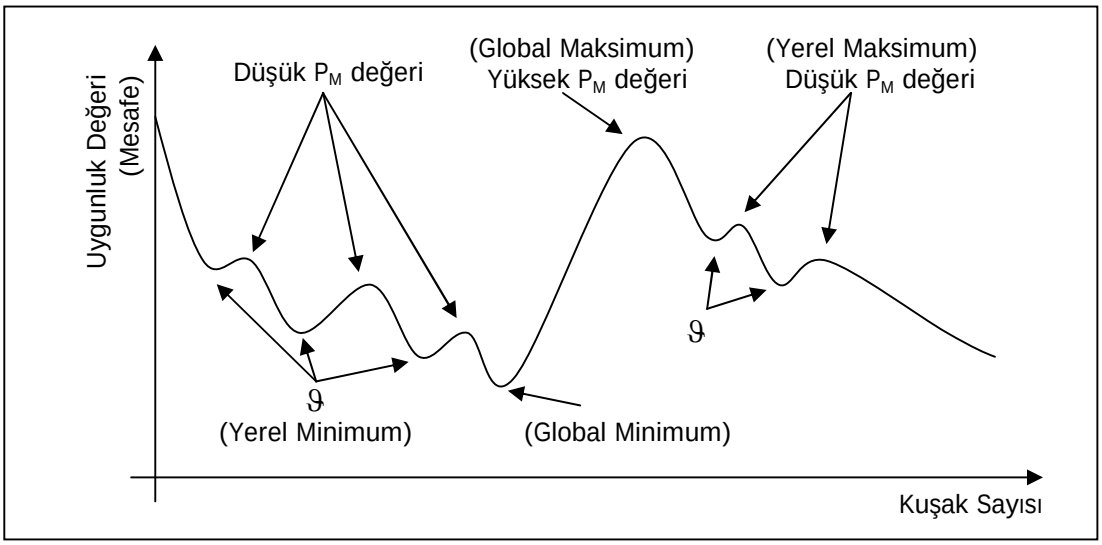


Şekil 3.5. Yer Değiştirme Mutasyon İşlemi

Mutasyon operatörünün temel amacı popülasyona görülmemiş çözümler katmanın yanında yerel optimumlardan (ϑ) kurtulmayı P_M ihtimaline bağlı olarak sağlamasıdır. Kullanılan mutasyon operatörlerin hepsinin amacı da bu doğrultudadır. Eğer ki genetik araştırma için popülasyon sayısı küçük seçilirse P_M oranın yükseltilmesi gerekebilir. Çünkü popülasyondaki birey çeşitliliği az olduğundan,

araştırma ilerledikçe, bireyler birbirlerine oldukça fazla benzeyecekler ve bu yüzden de araştırma sürecinde yerel optimumlara sık düşülecektir.

Mutasyon oranı, genellikle çok küçük bir değer seçilir (0.05-0.2). P_M değerinin büyük seçilmesi durumunda popülasyondaki bireylerin anne-baba'dan aldıkları kalıtımsal özelliklerin büyük bir kısmını kaybetmesi ve böylece optimal çözümlerden uzaklaşmasına sebep olunur. Mutasyon operatörünün büyük seçilme durumu aslında popülasyonun yeniden üretilmesi (*Random Restart*) gibi düşünülebilir. Ayrıca P_M değerinin çok yüksek olması popülasyonun yerel minimumdan bir anda global maksimuma yükseltebilir (Şekil 3.6). Bu da çözüm süresini arttırmakta veya optimal çözümlere ulaşılamamasına neden olmaktadır.



Şekil 3.6. P_M Değerinin Çözüm Sürecinde Etkisi

Bu yüzden GSP'de en iyi bireyin korunması (*Elitism*) şartının sağlanmasının ardından mutasyon oranının adaptif bir sistemle yükseltilmesi en makul olan çözümdür.

3.3.6. Seçkinlik

Genetik operatörlerle üretilecek olan yeni popülasyonda en iyi birey kaybedilecektir. Seçkinlik (*Elitism*), Genetik Algoritmanın performansını oldukça

hızlı bir şekilde arttırmaktadır (Nabiyev 2003). Seçkinlik ilk başta en iyi kromozomu yeni popülasyona kopyalar. Sadece, bir bireyin yeni popülasyonda yer alması gibi belirlenecek yüzdelik veya adetsel *k* bireyin de sonraki nesle aktarılması sağlanabilir. Popülasyondaki en iyi nesillerin kaybını önleyen seçkinlik GA'nın performansını oldukça hızlı bir şekilde attırmaktadır.

3.3.7. Durdurma Kriterleri

Genellikle GA'da durdurma kriteri popülasyonun değişmemesine bağlıdır. Bu durdurma kriteri GSP problemleri çözümünde de kullanılabildiği gibi yeterli kriter olmayabilir. Çünkü problem yerel optimuma takılmış ve mutasyon operatörünün ihtimali (P_M) dahilinde henüz bu yerel noktadan çıkamamış olabilir. Bu yüzden GA'larda farklı durdurma kriterleri kullanılır (Hulin 1997).

Jenerasyon Sayısı : Parametrik olarak kullanıcı tarafından belirlenen bir jenerasyon (kuşak) sayısına ulaşıldığı zaman GA araştırması sonlandırılır.

Zaman Sınırı : Parametrik olarak belirlenen bir maksimum çalışma süresi ile GA'ların çalışma zamanı kontrol edilebilir.

Uygunluk Eşiği : Belirlenen bir uygunluk değeri elde edilinceye kadar Genetik Algoritma durdurulmaz. Eğer araştırma yerel optimuma düşmüş ve mutasyon operatörü popülasyonu bu noktadan çıkartamıyor veya girilen uygunluk değeri elde edilemeyecek derecede ise Uygunluk Eşiği durdurma kriterinde sonsuz döngü oluşabilir. Ayrıca rasgele oluşturulmuş veya optimum değeri bilinmeyen GSP problemleri için kullanılması imkansızdır.

Yakınsama : Popülasyondaki en iyi olan veya tüm bireylerin uygunluk değerlerinin değişmemesi durumuna göre değişim gözlenmemesi bir sayaçta sayılır. Belirlenecek bir jenerasyon sayısı sonrasında GA'ların çalışması sonlandırılır. Popülasyondaki değişimin sabit hale gelmesi (*stabilizasyon*) olarak da adlandırılabilir.

3.4. Genetik Algoritmanın Çeşitleri

Genetik Algoritmalar (GA), çeşitli problem tiplerine uygulanması sırasında yetersiz kaldığında, erken yakınsama (*premature convergence*) problemi olduğunda veya çevrim süresi için çok fazla zaman harcandığında geliştirilmesi gerekmektedir.

3.4.1. Basit Genetik Algoritma

Holland'ın orijinal GA'sı, "Basit (*Simple*)" veya "Temel (*Basic*)" Genetik Algoritma (BGA, TGA) olarak bilinir. Basit genetik operatörlerden ve süreçlerden oluşan (Şekil 3.1) algoritmalardır. Basit Genetik Algoritma (BGA), GSP gibi *NP-Zor* sınıfındaki optimizasyon problemleri için küçük boyutlu problemlerde optimal çözümleri bulabilirken, büyük boyutlu GSP'lerde optimum değeri bulmakta diğer genetik modellere göre oldukça yetersiz kalmaktadır (Hu ve Gu 2007). Grefenstette ve arkadaşları (1985) HX çaprazlama metodunu kullanarak 50, 100 ve 200 şehirli GSP problemlerinde sırasıyla %25, %16 ve %27 gibi yüksek hata değerleri almışlardır. Bu hata değerlerinin yüksekliği BGA'ların GSP için verimli olmadıklarını açık bir şekilde göstermektedir.

3.4.2. Paralel Genetik Algoritma

İki veya daha fazla Genetik Algoritmanın paralel çalışması esasına dayanır. Özellikle karmaşık ve büyük optimizasyon problemlerinin çözümünde kullanılır. Popülasyon, alt popülasyonlara bölünerek, her popülasyon farklı bilgisayar ağında çalıştırılır. Önceden belirlenen stratejilere göre GA'lar birbirleri arasında haberleşirler. Schleuter (1989), Mulhenbein (1991) ve Schleuter (1991) tarafından Paralel Genetik Algoritmalar GSP bazında incelenmiş ve PGA önerilmiştir. Schleuter ilk olarak 1989'da tanıttığı ASPARAGOS (*Asynchronous Parallel Genetic Optimization Strategy*) metodunu 1997'de geliştirmiştir. Çeşitli tur geliştirme metotlarını kullanarak 7 farklı S-GSP için (D198=%0.00, Lin318=%0.00, Pcb442=%0.01, Att532=%0.00, Rat783=%0.00, Fl1577=%0.22, Fl3795=%0.22) oldukça iyi sonuçlar elde etmiştir.

3.4.3. Mikro Genetik Algoritma

Genetik Algoritmanın popülasyon tabanlı bir algoritma olmasından dolayı çözüme ulaşması, popülasyondaki bireylerin optimal çözüme tekrarlamalı olarak geliştirilip yaklaştırılmaları gerekmektedir. Bu yaklaşma işlemi oldukça uzun zaman alabilmektedir. Süreyi kısaltmak için az popülasyonla çalışabilen ve iyi performansa sahip Genetik Algoritmaların geliştirilmesi gerekmektedir (Karaboğa 2004). Mikro (*Micro*) Genetik Algoritmalarda (μGA) popülasyon 5-6 bireyden oluşacak şekilde hazırlanır ve belirli sürelerde popülasyondaki en iyi birey haricindeki bireyler tekrar rasgele üretilerek araştırmaya devam edilir (Krishnakumar 1989). Aynı durum yerel arama metotlarında "*Random Restart*" olarak da bilinmektedir. Literatürde μGA'nın genellikle matematiksel fonksiyon optimizasyonlarında kullanıldığı görülmektedir. Ayrıca yerel optimizasyon metotları da kullanarak 1 nüfuslu yüksek performanslı Genetik Algoritmalara biyolojiksel olarak "*parthenogenetic*" algoritma denilebilir (Johnson ve McGeoch 1997).

3.4.4. Melez Genetik Algoritma

Çeşitli sezgisel algoritmaların Basit Genetik Algoritmaya alt işlemci olarak eklenmesi ile oluşturulan algoritmalara Melez (*Hybrid*) Genetik Algoritma (MGA) denir. MGA'lar bir veya daha fazla komşuluk araması, elitist stratejiler, çeşitli optimizasyon algoritmaları (Yapay Sinir Ağları, Tabu Arama, Karınca Kolonisi vs.), yerel aramalar, ekstra seçim ve üretme operatörleri içerebilirler (Chainate ve ark. 2007). Chainate ve arkadaşları (2007) BGA'lar ile MGA'ları 17, 35 ve 76 şehir sayısına sahip 3 farklı GSP üzerinde karşılaştırmış ve BGA'ların performansının arttırılması için MGA'ları önermişlerdir.

3.4.5. Genetik Yerel Arama Algoritması

Genetik Algoritmalara güçlü yerel arama tekniklerinin eklenmesiyle GA'ların gücü arttırılır. Bu yeni algoritmalara Genetik Yerel Arama (GYA) Algoritması denir (Ibaraki 1997, Merz ve Freisleben 1997, Yagiura ve İbaraki 1996, Katayama ve Narihisa 1999).

Ulder ve arkadaşları (1991) yaptıkları çalışmada GYA'nın performansını 8 ayrı GSP üzerinde, farklı algoritmalarla ile test etmişlerdir. Sonuçta GYA'nın güçlü bir yerel arama metodu ile birleştirilmesinden optimuma oldukça yakın sonuçlar alınabileceğini göstermişlerdir (Tablo 3.1).

GYA algoritmalarında tüm bireyler yerel minimumu temsil eder. Bu yüzden bir genetik operatör uygulandıktan sonra yerel araştırma prosedürü operatörden çıkan bireylere uygulanmalıdır. GA ile yerel prosedürlerin birleştirilmesi Memetik Algoritma (*Memetic Algorithm*) olarak da bilinmektedir. Merz ve Freisleben (1997) tarafından önerilen GYA algoritması Şekil 3.7'deki gibidir.

Tablo 3.1. Çeşitli GYA Performanslarının Karşılaştırılması

Problem	*t* (saniye)	Simulated Annealing (2-Opt)	Multi Start-LS (2-Opt)	Multi Start-LS (LK)	Genetic Local S. (2-Opt)	Genetic Local S. (LK)
Gro48	6	1.89	1.35	**0.00**	0.19	**0.00**
Tom57	10	1.94	1.34	**0.00**	0.50	**0.00**
Eur100	60	2.59	3.23	**0.00**	1.15	**0.00**
Gro120	86	2.94	4.57	0.08	1.42	**0.05**
Lin318	1600	2.37	6.35	0.37	2.02	**0.13**
Gro442	4100	2.60	9.29	0.27	3.02	**0.19**
Gro532	8600	2.77	8.34	0.37	2.99	**0.17**
Gro666	17000	2.19	8.67	1.18	3.45	**0.36**

```
POP başlangıç popülasyonunu sezgisel metotlarla oluştur;
Popülasyondaki her i elemanına Yerel-Arama uygula (i ∈ POP);
Popülasyonda yakınsama oluncaya kadar aşağıdaki işlemleri tekrarla;
Başla
      For i := 1 to #Çaprazlamalar do
      Başla
            Popülasyondan rasgele i_a ve i_b bireylerini eşleştir
            (i_a, i_b ∈ POP);
            i_c := DPX(i_a, i_b);
            i_c := Yerel-Arama(i_c);
            i_c neslini POP popülasyonuna ekle;
      Son;
      For i := 1 to #Mutasyonlar do
      Başla
            Popülasyondan rasgele i_m bireyi seç (i_m ∈ POP);
            i_m := Mutasyon(i_m);
            i_m := Yerel-Arama(i_m);
            i_m neslini POP popülasyonuna ekle;
      Son;
      S* := Seçim(POP);
Son;
```

Şekil 3.7. Genetik Yerel Arama Algoritması

Merz ve Freisleben (1997) yaptıkları çalışmada D198, Lin318, Pcb442, Att532, Rat783, Fl1577 ve Fl3795 gibi popüler GSP'ler üzerinde çalışmışlardır. GYA ile belirtilen GSP problemlerinde elde ettikleri en iyi neticeler sırasıyla %0.00 (163 saniye) , %0.00 (168 saniye), %0.00 (178 saniye), %0.00 (290 saniye), %0.00 (424 saniye), %0.20 (12762 saniye) ve %0.18 (17824 saniye) olarak görülmüştür.

Aslında İterasyonel Yerel Arama (İYA) metotları veya Çok Başlangıçlı Yerel Arama (*Multi Start Local Search*) metotları GYA'larda popülasyon sayısının 1 seçilmesinin özel bir durumudur (Yagiura ve İbaraki 1996). GA'ya eklenen yerel arama metodu bazı araştırmacı tarafından MGA grubunda sayılırlar (Misevicius ve ark. 2004, Chainate ve ark. 2007). Şekil 3.7'de tanımlanan algoritma haricinde

Misevicius ve arkadaşları tarafından tanımlanan algoritma da aslında GYA algoritması olarak kabul edilebilir (Şekil 3.8).

```
POP popülasyonu üretilir
S* ← En iyi birey (POP)
Durdurma kriteri sağlanıncaya kadar devam et;
Başla
      Üretilecek yavru sayısı kadar devam et;
      Başla
            s′ ve s′′ bireyleri seçilir;   (s′, s′′ ∈ POP)
            s′′′← Çaprazlama (s′, s′′);
            Yerel Geliştirme (s′′′);
            POP ← POP ∪ {s′′′};
            Eğer f(s′′′) < f(s*) ise s* ← s′′′;
            POP popülasyonundaki birkaç bireye mutasyon uygula;
      Son;
      İçinde en iyi bireyler kalacak şekilde POP popülasyonunu
      güncelle;
Son;
```

Şekil 3.8. Farklı Bir Genetik Yerel Arama Algoritması

Yagiura ve İbaraki (1996) yaptıkları çalışmada GYA için aşağıdaki sonuçları belirmişler ve yüksek kalitede çözümler için GYA veya Tavlama Benzetimi (*Simulated Annealing*) önermişlerdir.

- GYA, GA ve İYA'dan daha yüksek kaliteli çözümler elde edebilir, özellikle uzun hesaplama zamanı kabul edildiği zaman.
- GYA, çaprazlama ve mutasyon türlerine oldukça duyarsızdır. Çaprazlama ve mutasyonun arasından çaprazlama metotları biraz daha etkili gibi görünmektedir. Öte yandan yalnızca mutasyonlarla birlikte olan GYA'nın uygulaması daha kolaydır. Çünkü mutasyon, yerel aramada komşuları kullanarak gerçekleştirilir ve bu yüzden de çok fazladan efora gerek yoktur.
- Çözüm kalitesi kritik bir şekilde komşunun türüne bağlıdır

GYA metotlarında BGA'nın yapısını fazla değiştirmeden kullanılan literatürde farklı kombinasyonlarda mevcuttur (Freisleben ve Merz 1996, Liu ve Kang 2003, Ghoseiri ve Sarhadi 2007). Ibaraki (1997) neslin mutasyon operatöründen çıkmasından sonra (bir sonraki seçim işleminden önce) Freisleben ve Merz (1996, 1997), Ghoseiri ve Sarhadi (2007) ve Matayoshi ve arkadaşları (2004) çaprazlamadan sonra ve mutasyondan sonra yerel arama prosedürleri uygulamışlardır.

4. GENETİK YEREL ARAMA ALGORİTMASI VE BİLEŞENLERİ

Önceden de belirtildiği gibi Genetik Algoritma (GA), GSP gibi zor optimizasyon problemlerinin çözümünde yetersiz kalabilmektedir. Bu yüzden GA'nın farklı sezgisel veya melez metotlar ile güçlendirilmesi gerekmektedir. Yapılan literatür araştırmaları sonucunda GA'nın en iyi, yerel arama metotlarıyla birleştirilmesi sonucunda optimuma yakın değerler verdiği görülmüştür. Bu birleştirme işlemi ile oluşan GA'lara Genetik Yerel Arama (GYA) Algoritmaları denir (Merz ve Freisleben 1997, Yagiura ve İbaraki 1996, Liu ve Kang 2003, Ghoseiri ve Sarhadi 2007). Aynı zamanda GYA algoritmaları Memetik Algoritma (MA) olarak da tanınırlar.

GSP için uygulanan GYA'nın bazı bileşenleri BGA'lara göre farklılık göstermektedir. Bu yüzden GYA algoritması için bu bileşenler haricinde ekstra bileşenlerin tanımlanması çözüm sürecinde performansı arttıracaktır. GYA için eklenebilecek olan bileşenler;

- Tur Oluşturma Metotları ile daha kaliteli başlangıç popülasyonlarının oluşturulması,
- GSP'ye özgü güçlü çaprazlama operatörlerinin kullanılması,
- GSP'ye özgü güçlü mutasyon operatörlerinin kullanılması,
- Yerel optimuma takılma durumunu takip eden bir modülün eklenmesi ve
- GYA Algoritmasının asıl temel operatörü olan yerel arama prosedürlerinin farklı biçimlerde eklenmesi şeklindedir.

Bu bileşenler, GA'nın GSP optimizasyonundaki başarısını oldukça büyük oranda arttırmaktadır. Ayrıca tekrar üreme operatörlerinin başarısını yükseltmek ve GYA'nın çalışma süresini kısaltmak için çeşitli adaptif süreçler kullanılabilir. GSP'lerde şehir sayısı arttıkça çözüm uzayının da üstel bir şekilde artması, problemin çözülebilirliğini ve optimum değerlere ulaşmayı oldukça zorlaştırdığı için çeşitli tur bölme teknikleri de kullanılabilir.

4.1. Tur Oluşturma Metotları

GA'lar gibi diğer popülasyon tabanlı algoritmaların hepsinin temelindeki bileşenlerden birisi başlangıç popülasyonunun oluşturulmasıdır. Basit algoritmaların genelinde başlangıç popülasyonu rasgele metotlar ile oluşturulabildiği gibi çeşitli sezgisel yaklaşımların kullanılması ile de oluşturulabilir.

Sezgisel metotlar çoğunlukla rasgelelikten uzak, sadece o anda elinde bulunan veriler içerisinden en iyi olanı değerlendirip, geriye dönmeden kullanan yaklaşımlardır. Bu davranışlarından dolayı sezgisel metotlar Açgözlü Algoritmalar (*Greedy Algorithms*) olarak tanımlanırlar. Sezgisel metotlar optimuma doğru çok kısa bir zamanda çok fazla yaklaşım gösterirler. Fakat açgözlü olmalarından dolayı global optimuma ulaşamazlar. Evrimsel algoritmalar ise çözüm uzayında rasgele sunulmuş bir başlangıç popülasyonu içerisinde tekrarlamalı olarak popülasyonu iyileştirmeye çalışır. Fakat basit evrimsel algoritmalar açgözlü metotlar ile oluşturulmuş bir popülasyonun uygunluk değerleri ortalamasına gelebilmek için sezgisel metotlardan daha fazla süre harcayabilirler ve açgözlü algoritmaların oluşturduğu uygunluğa gelinceye kadar yerel optimumlarda tıkanabilirler.

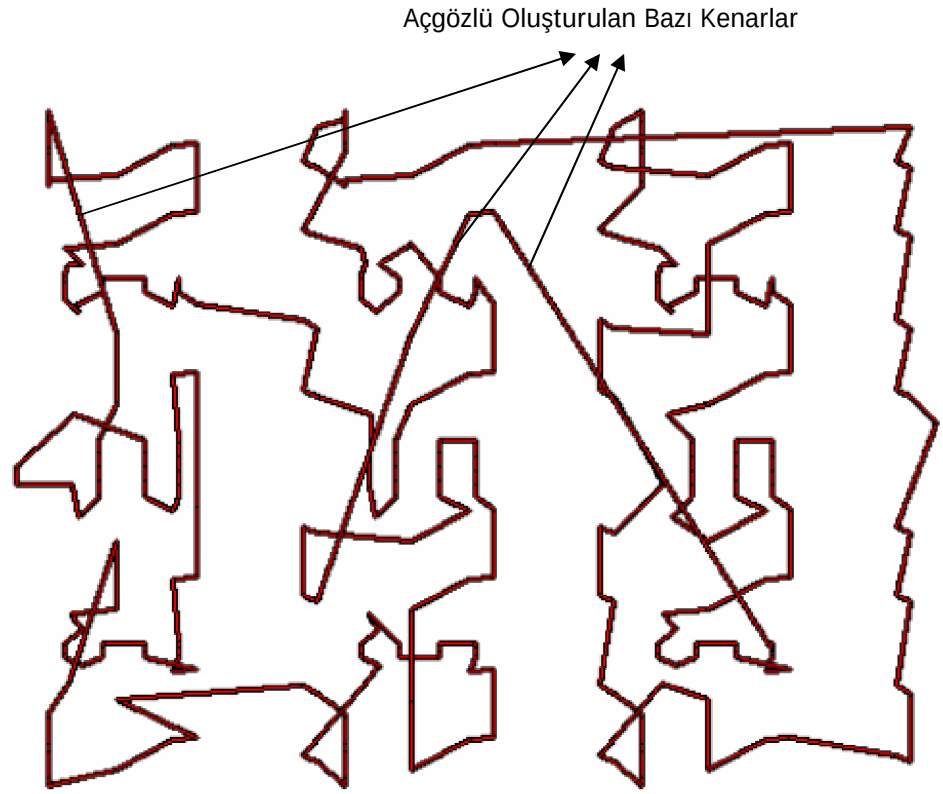
Yukarıda belirtilen unsurlardan dolayı açgözlü Tur Oluşturma Metotları GA'nın veya diğer algoritmaların başlangıç popülasyonunu oluşturmasında oldukça sık bir şekilde kullanılır. Ayrıca bu metotların kullanılması GA'ların ulaşacağı optimum değeri ve çevrim süresini de etkilemektedir. GSP'de kullanılan birçok Tur Oluşturma Metodu mevcuttur ve bu metotlar, çalışma süreleri ve üretilen turun kalitesine göre değerlendirilirler (Johnson ve McGeoch 1997).

4.1.1. En Yakın Komşu Tur Oluşturma Metodu

En Yakın Komşu (*Nearest Neighbor*) GSP için en basit ve yaygın olarak kullanılan algoritmadır. Algoritma ilk başta rasgele bir başlangıç noktasını turun ilk noktası olarak ekler. Daha sonra algoritma tura son eklenen noktaya en yakın olan ve ziyaret edilmemiş noktayı ekler. Bu şekilde tüm ziyaret edilmemiş şehirler tura

ekleninceye kadar devam edilir (Rosenkrantz ve ark. 1974). En Yakın Komşu (NN) algoritması *n* şehirli bir GSP problemi için $\Theta(n^2)$ çalışma süresine ihtiyacı vardır.

Eğer ki her şehir başlangıç pozisyonuna geçecek şekilde tüm şehirler incelenirse NN'nin verebileceği en iyi tek sonuç elde edilir. Bu metot, Tekrarlayan En Yakın Komşu (*Repetitive Nearest Neighbor* (RNN)) olarak bilinir. RNN $\Theta(n^3)$ çalışma süresine ihtiyaç duyar. NN'de diğer tüm Tur Oluşturma Metotları gibi açgözlü algoritmalardır. Bu yüzden optimum sonuçtan oldukça uzak değerler vermektedirler. Çünkü tura ilk başta eklenen şehirler kısa kenarlar oluştururken, turun sonlarına doğru eklenen kenarlar eklenebilecek mümkün kısa kenarlardan daha büyüktür (Şekil 4.1). Johnson ve McGeoch (2002) rasgele dağılımlı 10000 şehirli bir GSP probleminde NN'nin 0.28 saniyede sonuç ürettiğini ve %24.79 hata verdiğini göstermişlerdir. Marinakis ve arkadaşlarının (2005) DIMACS'tan aldıkları ortalamada NN'nin %24.85 hata ürettiğini göstermişlerdir.



Şekil 4.1. En Yakın Komşunun Lin318 Problemindeki Açgözlü Davranışı

4.1.2. Çift Bitişli En Yakın Komşu Tur Oluşturma Metodu

Çift Bitişli En Yakın Komşu (*Double Ended Nearest Neighbor*), NN algoritmasının bir türevidir. İlk başta rasgele seçilen bir *a* şehri *T* turunun başlangıç noktasına eklenerek başlanır. Ardından en yakın veya rasgele seçilen ikinci şehir olan *b*, *T* turunun bitiş noktasına eklenir. Daha sonraki davranış NN algoritmasının çift yönlü çalışması şeklindedir. Bulunan *a'* ve *b'* şehirleri *T* turuna çift yönlü olarak eklenir. Bu işlem tekrarlamalı olarak tüm şehirler ekleninceye kadar devam edilir. Çift Bitişli En Yakın Komşu (DENN) pratikte neredeyse NN kadar hızlı çalışabilir (Johnson ve McGeoch 2002).

4.1.3. En Yakın Parça Tur Oluşturma Metodu

NN algoritmasının temel dezavantajı, birkaç şehrin oluşum sırasında dikkat edilmeden eklenmesiyle, turun toplam uzunluğunun yüksek çıkmasıdır. Bu şekildeki tur oluşturma da GSP'de yüksek hatalara yol açar. NN'nin bu hatasının üstesinden gelebilmek için En Yakın Parça (*Nearest Fragment*) algoritması Ray ve arkadaşları (2005) tarafından NN ile üretilmiş bir başlangıç popülasyonuna sahip BGA için iç çevrim operatörü olarak önerilmektedir. Fakat En Yakın Parçanın (NF) bir operatör olarak kullanılması çevrim süresi olarak gereksiz maliyet getirmekte ve performansı düşürmektedir. Bu yüzden bu metodun sadece başlangıç popülasyonu üretilmesinde NN üzerine bir eklenti olarak uygulanması daha uygundur.

NF metodunda, *n* şehirli bir *T* turu *frag* sayısınca rasgele parçalara bölünür. Burada belirtilen "*frag*" kullanıcı tabanlı bir parametredir. Tur parçalara (*fragment*) bölündükten sonra ilk parça rasgele seçilir. İlk parçanın bitiş noktası diğer kullanılmamış parçaların başlangıç veya bitiş noktaları ile birleştirilmesi durumundaki maliyetlerden en az uzamayı verecek olan parça seçilir. Eğer seçilen parçanın bitiş noktası turdaki parçanın son noktasına en az uzamayı verdiyse seçilen parça ters çevrilerek, aksi halde olduğu gibi eklenir. Bu işlem tüm parçalar ekleninceye kadar devam eder.

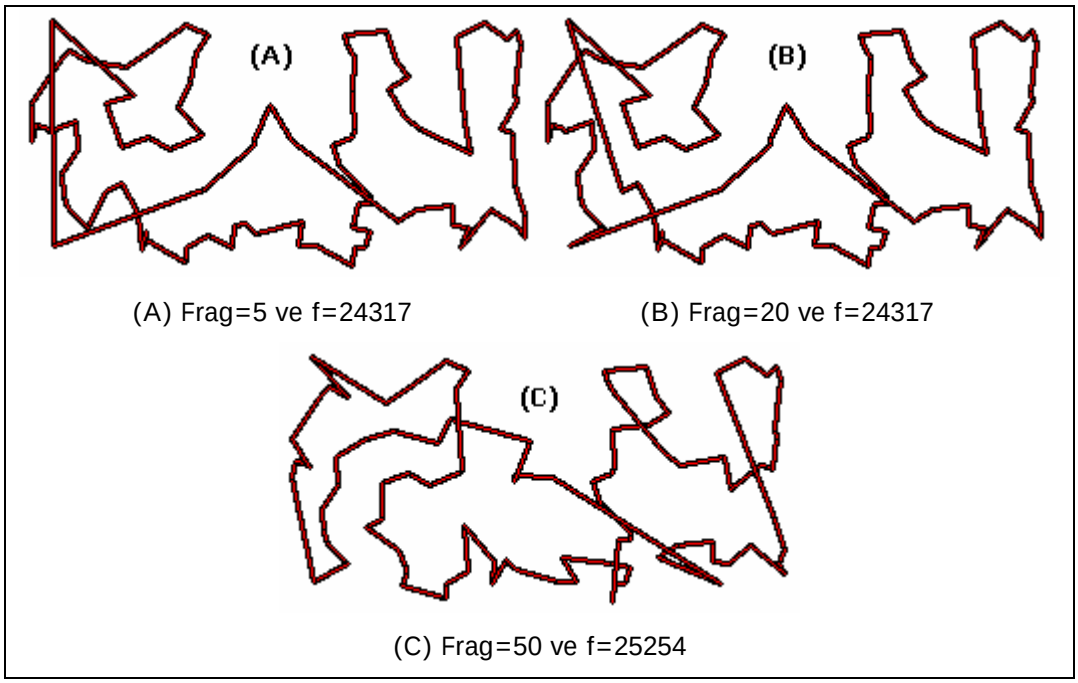
Ray ve arkadaşları (2005) yaptıkları çalışmada NF'yi reel olarak ölçmemişlerdir. Fakat NF kullandıkları bir BGA'da kullanmadıkları BGA'lardan daha iyi netice almışlardır (bkz. Şekil 4.5 FRAG_GA). Turun parçalara ayrılma miktarını da şehir sayısı (*n*) ile bağlantılı olarak *frag* ≈ *n* / 8 olarak vermişlerdir.

Ray ve arkadaşları (2005) NF için reel olarak deneysel sonuçlar ortaya koymadıkları için bu metot geliştirilen yazılımda 6 farklı GSP problemi üzerinde denenmiştir (Tablo 4.1).

Tablo 4.1. En Yakın Parçanın Performansı

Problem	KroA100	KroB150	KroA200	KroB200	Lin318	Pcb442
NF-Frag	12	18	25	25	39	55
Optimum Mesafe	21282	26130	29368	29437	42029	50778
En İyi Mesafe Hata (ε_{iyi})	24698 %16.05	31490 %20.51	34355 %16.98	35389 %20.22	49201 %17.06	59274 %16.73
En Kötü Mesafe Hata ($\varepsilon_{KÖTÜ}$)	32054 %50.62	39933 %52.82	43899 %49.48	44758 %52.05	60944 %45.00	71310 %40.43

NF metodu yapılan deneysel sonuçlara göre ortalamada "en iyi" de %17.925, "en kötü" de %48.4 hata vermektedir. Fakat önerilen *frag* ≈ *n* / *k* parçalama sabitinde *k* değerinin şehir sayısı arttıkça arttırılması gerektiğini görülmüştür. Pcb442 problemi için *k*=22 seçildiği zaman parçalama oranı (*frag*) 20 olmaktadır ve en iyi hatanın %15.69 (58747) ve en kötü hatanın %33.00 (67533) değerlerine düştüğü görülmüştür. Parçalama yine 20 olarak alındığında KroB200 probleminde ise en iyi hatanın %19.81 (35268) ve en kötü hatanın %51.26 (44525) değerlerine indiği görülmüştür. Şekil 4.2'de KroA100 problemi için NF'de seçilen parçalama sayılarına göre elde edilen en iyi güzergahlar verilmiştir.



Şekil 4.2. En Yakın Parça ile Tur Oluşumu

4.1.4. En Yakını Ekleme Tur Oluşturma Metodu

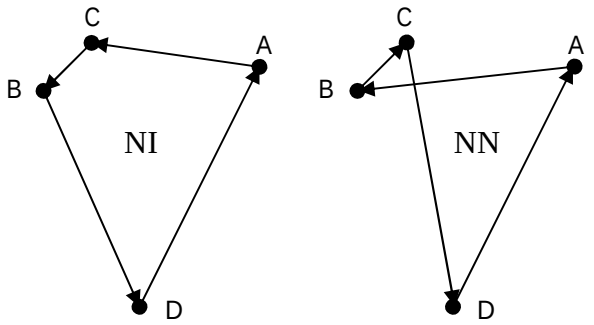
Rasgele bir *i* şehri seçilip *T* turuna eklenerek başlanır. Daha sonra turda yer alan bir *j* ($j \in T$) şehri için, turda yer almayan ve kenar maliyeti en düşük olan *k* şehri bulunur ($k \notin T$). Bulunan *k* noktası için en uygun ($\Delta_{MIN} = d(l, k) + d(k, m) - d(l, m)$) iki şehir arası yer bulunur {*l, m*} ve daha sonra *k* şehri bulunan şehirlerin arasına {*l, k, m*} yerleştirilir. Bu süreç tüm şehirler ziyaret edilinceye kadar, yani tura eklenmeyen şehir kalmayıncaya kadar devam eder. En Yakını Ekleme (*Nearest Insertion*) algoritması *n* şehirli bir GSP problemi için $\Theta(n^2)$ çalışma süresine ihtiyacı vardır.

Johnson ve McGeoch (2002) rasgele dağılımlı 10000 şehirli GSP probleminde En Yakını Eklemenin (NI) *Held-Karp* değerine göre %26.50 hata verdiğini göstermiştir. Yaklaşık NN kadar hata değeri vermesinin yanında NI daha yavaş çalışır.

Şekil 4.3'deki gibi bir ağırlık matrisi üzerinden NI algoritmasını örnekleyelim. İlk başta *A* şehri *T* turana eklenerek başlansın. *A* şehrine en yakın şehir olan *B*, *T*

turuna en düşük maliyeti getirdiği için *B* şehri tura eklenir. *AB*'den başlanarak tur uzunluğunu en az arttıran kenar olan {*BC*} kenarı bulunur. *C* şehri *B* ve *A* şehri arasına eklenir ($T=\{A\rightarrow C\rightarrow B\rightarrow A\}$). Ardından eklenmeyen son şehir olan *D* en uygun yere {*BA*} eklenir ve sonuç turu $T=\{A\rightarrow C\rightarrow B\rightarrow D\rightarrow A\}$ şeklinde geliştir ve toplam tur maliyeti 32'dir. Aynı işlem NN algoritması için *A* şehri başlangıç şehri olarak seçilerek gerçekleştirilirse $T=\{A\rightarrow B\rightarrow C\rightarrow D\rightarrow A\}$ turu bulunur ve toplam maliyeti 40'dır.

	A	*B*	*C*	*D*
A	∞	5	10	13
B	5	∞	2	7
C	10	2	∞	20
D	13	7	20	∞

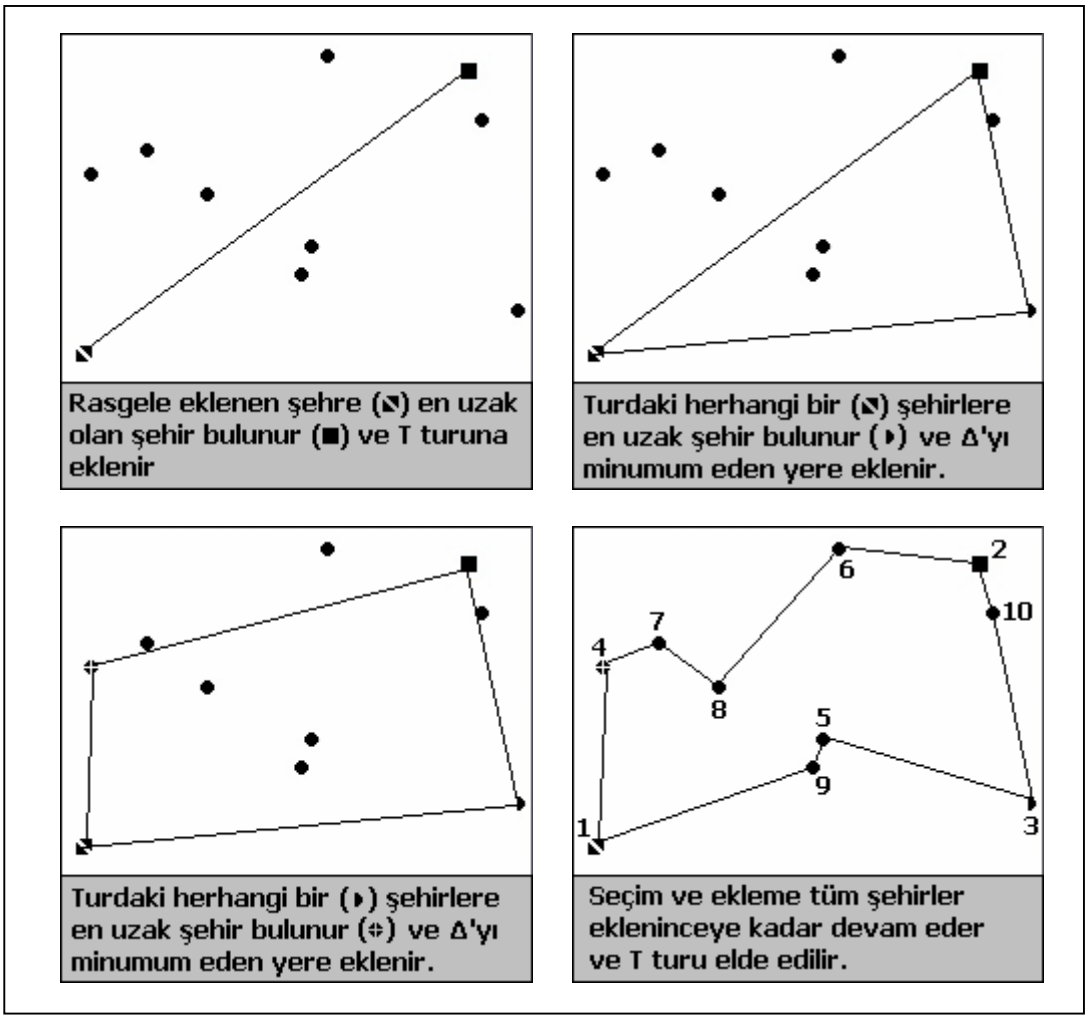


Şekil 4.3. En Yakın Ekleme ile Tur Oluşumu

4.1.5. En Uzağı Ekleme Tur Oluşturma Metodu

En Uzağı Ekleme (*Farthest Insertion*) metoduna, rasgele bir *i* şehrinin *T* turuna eklenmesi ile başlanır. Daha sonra eklenmemiş şehirlerin arasından maksimum artışı sağlayan bir *r* şehri ($r \notin T$) bulunur ve *T* turu $i\rightarrow r\rightarrow i$ olacak şekilde eklenir. Bu iki adımdan sonra üçüncü adım olan seçim adımı başlar. *T* turu içerisindeki herhangi bir *j* şehrine ($j \in T$) en uzak, turda yer almayan bir *r* şehri ($r \notin T$) bulunur. Seçim aşamasından sonra bulunan *r* şehrinin *T* turunda nereye eklenmesine karar verilmelidir. *T* turunda yer alan kenarlar içerisinden $\Delta = d(i, r) + d(r, j) - d(i, j)$ değerini minimize eden [*i*, *j*] kenarı bulunur ($i, j \in T$) ve seçim aşamasında bulunan *r* şehri *i* ve *j* şehirlerinin arasına ($i\rightarrow r\rightarrow j$) eklenir. Seçim ve ekleme işlemine tüm şehirler ekleninceye kadar devam edilir (Şekil 4.4).

En Uzağı Ekleme (FI) metodu $\Theta(n^2)$ çalışma süresine ihtiyaç duyar. Johnson ve McGeoch (2002) rasgele dağılımlı 10000 şehirli bir GSP probleminde FI'nin 2.59 saniyede sonuç ürettiğini ve %13.35 hata verdiğini göstermişlerdir. Marinakis ve arkadaşları (2005) 6 farklı GSP probleminin DIMACS'tan aldıkları sonuçlarında, FI'nin ortalamada optimumdan %16.64 fazla sonuç üretildiğini göstermişlerdir.



Şekil 4.4. En Uzak Ekleme ile Tur Oluşumu

4.1.6. Rasgele Ekleme Tur Oluşturma Metodu

Rasgele Ekleme (*Random Insertion*), FI metoduna oldukça benzer. Rasgele bir i şehrinin T turuna eklenmesi ile başlanır. Daha sonra eklenmemiş şehirlerin arasından rasgele bir r şehri ($r \notin T$) seçilir ve T turu $i \rightarrow r \rightarrow i$ olacak şekilde eklenir.

Bu iki adımdan sonra üçüncü adım olan seçim adımı başlar. Seçim aşaması Rasgele Ekleme (RI) ve FI'nin farkıdır. RI, T turu r şehrini seçerken maksimum uzaklık kriterine dayanmadan rasgele bir seçim yapar ($r \notin T$). Seçim aşamasında rasgele bulunan r şehrinin T turunda nereye eklenmesine karar verilmelidir. T turunda yer alan kenarlar içerisinden $\Delta = d(i, r) + d(r, j) - d(i, j)$ değerini minimize eden $[i, j]$ kenarı bulunur ($i, j \in T$) ve seçim aşamasında bulunan r şehri i ve j'nin arasına ($i \rightarrow r \rightarrow j$) eklenerek seçim ve ekleme işlemine tüm şehirler ekleninceye kadar devam edilir.

RI metodu $\Theta(n^2)$ çalışma süresine ihtiyaç duyar. Marinakis ve arkadaşları (2005) 1000 şehirden fazla büyüklüğe sahip 6 farklı GSP probleminin DIMACS'tan aldıkları sonuçlarında RI'nin ortalamada optimumdan %17.47 fazla sonuç üretildiğini göstermişlerdir. Johnson ve McGeoch (2002) analizlerinde, rasgele dağılımlı 10000 şehirli bir GSP probleminde RI'nin %15.00 hata verdiğini göstermişlerdir.

Tur oluşturma işlemlerinde sıkça kullanılan diğer yaklaşımlar (Marinakis ve arkadaşları 2005):

- Greedy (ε_{ORT} = %18.23)
- Boruvka (ε_{ORT} = %17.76)
- Quick-Boruvka (ε_{ORT} = %19.6)
- Savings (ε_{ORT} = %11.26) algoritmalarıdır.

4.2. Çaprazlama Metotları

Genetik Algoritmalar, en iyinin yaşamasını ve iyi olmayan nesillerin sonraki popülasyonda yer almamasını konu alan bir yapay zekâ dalıdır. Bu noktada iyi nesillerin varlıklarını sonraki kuşaklara aktarabilme görevi GA'da çaprazlama operatörleri tarafından gerçekleştirilir. Günümüzde GA'da GSP için kullanılan birçok çaprazlama operatörü vardır. Her bir çaprazlama operatörünün GSP'deki etkisi ve performansı farklıdır. Bu yüzden geliştirilen uygulama yazılımında birçok çaprazlama operatörü kullanılmıştır.

4.2.1. Tek Noktalı Çaprazlama

Tek Noktalı Çaprazlama (*One Point Crossover*) metodu oldukça basit bir çaprazlama metodudur. Her iki ebeveyn için rasgele seçilen bir çaprazlama noktasından kromozomun sonuna kadar olan dizilerin karşılıklı yer değiştirmesi ile iki nesil üretilmesi prensibine dayanır. Fakat bu tip bir çaprazlama metodunda ile GSP için illegal turlar oluşabilmektedir. Bu illegallik, bir şehre birden fazla kere uğranması ve uğranmayan şehir kalmasıdır. Bu problem çeşitli düzeltme algoritmaları ile düzeltilebilmektedir. Mitchell (2005) tarafından Statik, Ebeveyn Tabanlı ve Rasgele olmak üzere 3 farklı düzeltme algoritması sunulmaktadır.

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

Adım 1: Rasgele sayı üretilir ($R1$=4)

Adım 2: Şehirler karşılıklı yer değiştirilerek iki yavru oluşturulur.

$$\text{Yavru–1} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow \mathbf{1} \rightarrow \mathbf{8} \rightarrow \mathbf{7} \rightarrow \mathbf{6} \rightarrow \mathbf{9} \rightarrow \mathbf{3}\ \}$$

$$\text{Yavru–2} = \{\ 4 \rightarrow 5 \rightarrow 2 \rightarrow \mathbf{4} \rightarrow \mathbf{5} \rightarrow \mathbf{6} \rightarrow \mathbf{7} \rightarrow \mathbf{8} \rightarrow \mathbf{9}\ \}$$

Adım 3: Yavrular düzeltme algoritmasına girer (Statik Şablon: SŞ).

Adım 3.1: Yavrularda [1-$R1$] arasındaki tekrarlanan şehirler boşaltılır.

$$\text{Yavru–1} = \{\ \mathbf{x} \rightarrow 2 \rightarrow \mathbf{x} \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

$$\text{Yavru–2} = \{\ \mathbf{x} \rightarrow \mathbf{x} \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

Adım 3.2: Yavrulardaki kullanılan şehirler statik şablonlarından çıkartılır.

$$S\text{Ş}_{\text{YAVRU–1}} := \{\ \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow 4 \rightarrow 5 \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x}\ \}$$

$$S\text{Ş}_{\text{YAVRU–2}} := \{\ \mathbf{1} \rightarrow \mathbf{x} \rightarrow 3 \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x}\ \}$$

Adım 3.3: Yavrularda boş olan genlere (x) kendi statik şablonlarındaki şehirler sırasıyla yerleştirilir.

$$\text{Yavru–1} = \{\ 4 \rightarrow 2 \rightarrow 5 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

$$\text{Yavru–2} = \{\ 1 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

Davis tarafından 1985 yılında tanımlanan *Modified Crossover* operatörü aslında Tek Noktalı Çaprazlama metodunda oluşan hataların düzeltilerek çaprazlamaya devam edilmesidir. İlk etapta rasgele bir *R*1 çaprazlama noktası seçilir. Anne'den yavruya [1-*R*1] arası şehirler kopyalanır. Yavrunun boş kalan genleri Baba'dan gelen şehirlerle doldurulurken kullanılmamış şehirlerin eklenmesi yapılır. Bu durumda *Modified Crossover* operatörü bahsedilen Tek Noktalı Çaprazlama operatörü ve "Ebeveyn Tabanlı Şablon" düzeltme algoritmasının birleşimidir.

4.2.2. İki Noktalı Çaprazlama

Tek Noktalı Çaprazlama metoduna oldukça benzeyen basit bir çaprazlama metodudur. Tek Noktalı Çaprazlama metodunda [*R*1-*n*] şeklinde bir çaprazlama durumu gerçekleşirken, İki Noktalı Çaprazlama (*Two Point Crossover*) metodunda ise çaprazlama işlemi [*R*1-*R*2] şeklinde olmaktadır. Her iki ebeveyn için rasgele seçilen iki çaprazlama noktası arasında kalan dizilerin karşılıklı yer değiştirmesi ile iki nesil üretilmesi prensibine dayanır. İki Noktalı Çaprazlama metodu ile de tekrarlanan şehirler oluşabilmektedir. Bundan dolayı çaprazlama metodunun uygulanmasının ardından bir düzeltme prosedürü uygulanmalıdır.

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$
$$\text{Baba} = \{\ 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

Adım 1: Rasgele iki sayı üretilir (*R*1=3 ve *R*2=7)

Adım 2: Şehirler karşılıklı yer değiştirilerek iki yavru oluşturulur.

$$\text{Yavru–1} = \{\ 1 \rightarrow 2 \rightarrow \mathbf{2} \rightarrow \mathbf{1} \rightarrow \mathbf{8} \rightarrow \mathbf{7} \rightarrow \mathbf{6} \rightarrow 8 \rightarrow 9\ \}$$
$$\text{Yavru–2} = \{\ 4 \rightarrow 5 \rightarrow \mathbf{3} \rightarrow \mathbf{4} \rightarrow \mathbf{5} \rightarrow \mathbf{6} \rightarrow \mathbf{7} \rightarrow 9 \rightarrow 3\ \}$$

Adım 3: Tekrarlanan şehirler düzeltilir (Ebeveyn Tabanlı Şablon: ETŞ).

Adım 3.1: Yavrularda [1-*R*1)-(*R*2-*n*] arasındaki tekrarlanan şehirler boşaltılır.

$$\text{Yavru–1} = \{\ \mathbf{x} \rightarrow \mathbf{x} \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow \mathbf{x} \rightarrow 9\ \}$$
$$\text{Yavru–2} = \{\ \mathbf{x} \rightarrow \mathbf{x} \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow \mathbf{x}\ \}$$

Adım 3.2: Yavruların ziyaret edilen şehirleri kendi ebeveyn tabanlı şablonlarından çıkartılır.

$$ETŞ_{YAVRU-1} = \{ \mathbf{x} \rightarrow \mathbf{x} \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \}$$

$$ETŞ_{YAVRU-2} = \{ \mathbf{x} \rightarrow \mathbf{x} \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \rightarrow \mathbf{x} \}$$

Adım 3.3: Yavrularda boş olan genlere (*x*) kendi ebeveyn tabanlı şablonlarındaki şehirler sırasıyla yerleştirilir.

$$\text{Yavru–1} = \{ \mathbf{3} \rightarrow \mathbf{4} \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow \mathbf{5} \rightarrow 9 \}$$

$$\text{Yavru–2} = \{ \mathbf{2} \rightarrow \mathbf{1} \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow \mathbf{8} \}$$

4.2.3. Çok Noktalı Çaprazlama

Çok Noktalı Çaprazlama (*Multi Point Crossover*) metodunda ilk başta çaprazlamayı yönlendiren *n* elemanlı bir *Ptr* deseni oluşturulmaktadır. *Ptr* deseni rasgele üretilen {0,1} rakamlarından oluşturmaktadır. Yani çaprazlama, *Ptr* desenine göre ebeveyndeki genlerin karşılıklı yer değiştirerek iki yavru oluşturması prensibine dayanmaktadır. Üretilen desende {1} olan genler karşılıklı yer değiştirirken, {0} olan genler durumlarını korumaktadır. Bu çaprazlama metodunda da illegal turlar oluşabilmektedir. Bu yüzden çaprazlamanın ardından bir gen düzeltme prosedürü uygulanmalıdır. Çok noktalı çaprazlamada gen değişimi sayısı ne kadar fazla olursa yavruların ebeveynlerinin kalıtsal özelliklerini taşıması da o denli azalmaktadır.

$$\text{Anne} = \{ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \}$$

$$\text{Baba} = \{ 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3 \}$$

Adım 1: Rasgele *Ptr* deseni oluşturulur.

$$Ptr = \{ 0 - 1 - 0 - 1 - 1 - 0 - 0 - 1 - 0 \}$$

Adım 2: Şehirler *Ptr* desenine göre yer değiştirilerek iki yavru oluşturulur.

$$\text{Yavru–1} = \{ 1 \rightarrow \mathbf{5} \rightarrow 3 \rightarrow \mathbf{1} \rightarrow \mathbf{8} \rightarrow 6 \rightarrow 7 \rightarrow \mathbf{9} \rightarrow 9 \}$$

$$\text{Yavru–2} = \{ 4 \rightarrow \mathbf{2} \rightarrow 2 \rightarrow \mathbf{4} \rightarrow \mathbf{5} \rightarrow 7 \rightarrow 6 \rightarrow \mathbf{8} \rightarrow 3 \}$$

Adım 3: Yavrular düzeltme algoritmasına alınır (Rasgele Şablon: RŞ).

Adım 3.1: Yavrularda *Ptr*[*i*]=0 olan ve tekrarlanan şehirler boşaltılır.

$$\text{Yavru–1} = \{\ \mathbf{x} \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 8 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow \mathbf{x}\ \}$$

$$\text{Yavru–2} = \{\ \mathbf{x} \rightarrow 2 \rightarrow \mathbf{x} \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 6 \rightarrow 8 \rightarrow 3\ \}$$

Adım 3.2: Yavrudaki şehirler kendi "Rasgele Şablonlarından" çıkartılır.

$$\text{RŞ}_{\text{YAVRU–1}} = \{\ x \rightarrow 2 \rightarrow x \rightarrow 4 \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow x\ \}$$

$$\text{RŞ}_{\text{YAVRU–2}} = \{\ 1 \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow 9\ \}$$

Adım 3.3: Yavrularda boş olan genlere (*x*) kendi rasgele şablonlarındaki şehirler rasgele yerleştirilir.

$$\text{Yavru–1} = \{\ \mathbf{4} \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 8 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow \mathbf{2}\ \}$$

$$\text{Yavru–2} = \{\ \mathbf{1} \rightarrow 2 \rightarrow \mathbf{9} \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 6 \rightarrow 8 \rightarrow 3\ \}$$

4.2.4. Kısmi Haritalanmış Çaprazlama

Kısmi Haritalanmış Çaprazlama (*Partially Mapped Crossover*) ilk etapta her iki eşten rasgele iki çaprazlama noktası seçer (*R*1 ve *R*2) ve bu iki rasgele sayı arasında kalan diziler yavrulara aktarılır. İkinci adımda her iki ebeveynden seçilen alt dizilerin karşılıklı haritaları çıkarılır. Yavruların boş olan genleri ebeveynden gelen şehirlerle doldurulur. Bu doldurma işleminde, eğer eklenecek olan şehir haritada yer alıyorsa, haritadaki karşılığı eklenir (Goldberg ve Lingle 1985). Kısmi Haritalanmış Çaprazlama (PMX) metodunda dikkat edilmesi gereken bir özellik, 3. adımda haritaya göre yerleştirme yaparken haritanın derinlemesine incelenmesi gerekliliğidir.

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

Adım 1: Rasgele iki sayı üretilir (*R*1=4 ve *R*2=7). Üretilen sayılar arasında kalan dizilerdeki şehirler yavrulara aynen aktarılır.

$$\text{Yavru–1} = \{\ x \rightarrow x \rightarrow x \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow x \rightarrow x\ \}$$

$$\text{Yavru–2} = \{\ x \rightarrow x \rightarrow x \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow x \rightarrow x\ \}$$

Adım 2: [*R*1-*R*2] arasındaki şehir dizisinin karşılıklı haritası çıkarılır.

Harita: 1↔4, 8↔5, 7↔6, 6↔7

Adım 3: Yavrulardaki boş genler [*R*1-*R*2] dizisini aldığı ebeveyn ile değil, diğer ebeveyn ile haritaya göre doldurulur (Yavru–1→Anne, Yavru–2→Baba).

$$\text{Yavru–1} = \{\ 1{>}4 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 8{>}5 \rightarrow 9\ \}$$

$$\text{Yavru–2} = \{\ 4{>}1 \rightarrow 5{>}8 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow 3\ \}$$

Adım 6. Sonuç olarak iki ebeveynden üretilen iki yavru aşağıdaki gibidir.

$$\text{Yavru–1} = \{\ 4 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 9\ \}$$

$$\text{Yavru–2} = \{\ 1 \rightarrow 8 \rightarrow 2 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 9 \rightarrow 3\ \}$$

4.2.5. Çevrim Çaprazlama

Çevrim Çaprazlama (*Cycle Crossover*) pozisyon temelli bir çaprazlama operatörüdür. Çevrim Çaprazlamaya (CX), iki ebeveyn için rasgele bir başlangıç noktasının (*R*1) seçilmesiyle başlanır. Anne'den yavruya eklenen *R*1 geninden sonra Baba'dan *R*1 geninde yer alan *Y* şehri, Anne'deki bulunduğu pozisyondaki gibi yavruya eklenir. Bu işlem eklenen bir noktanın Baba'daki karşılığına ($Y_{J.}$) varıncaya kadar devam eder. Yavruda boş kalan genler Baba'daki kullanılmayan genler ile doldurulur. Kısacası CX metodu çaprazlama sürecinde her şehir ve o şehrin pozisyonu eşlerin yalnızca birinden gelir. Bu işlem eklenmiş bir şehrin *J.* adımda tekrar eklenmeye çalışılmasına kadar devam eder (Oliver ve ark. 1987).

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 4 \rightarrow 1 \rightarrow 2 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3 \rightarrow 5\ \}$$

Adım 1: Rasgele başlangıç noktası seçilir (*R*1=1). Yavru–1 için Anne'den *R*1 pozisyonunda yer alan şehir *R*1 pozisyonuna yerleştirilir.

$$\text{Yavru–1} = \{\ \mathbf{1} \rightarrow \text{x} \rightarrow \text{x} \rightarrow \text{x} \rightarrow \text{x} \rightarrow \text{x} \rightarrow \text{x} \rightarrow \text{x} \rightarrow \text{x}\ \}$$

Adım 2: Anne'den *R*1 pozisyonundan eklenen {1} şehri ile karşılıklı olan Baba'daki şehir {4}, Anne'deki olduğu pozisyona eklenir ve *R*1 son eklenen şehir sıra numarasına set edilir.

$$R1 \leftarrow \text{Anne_Gen_Sıra_No[Baba[}R1\text{]]}$$

$$\text{Yavru–1} = \{\ 1 \rightarrow x \rightarrow x \rightarrow \mathbf{4} \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow x\ \}$$

Adım 3: Anne'den *R*1 pozisyonundan eklenen {4} şehri ile karşılıklı olan Baba'daki şehir {8}, Anne'deki olduğu pozisyona eklenir ve *R*1 son eklenen şehir sıra numarasına set edilir.

$$\text{Yavru–1} = \{\ 1 \rightarrow x \rightarrow x \rightarrow 4 \rightarrow x \rightarrow x \rightarrow x \rightarrow \mathbf{8} \rightarrow x\ \}$$

Adım 4: Anne'den *R*1 pozisyonundan eklenen {8} şehri ile karşılıklı olan Baba'daki şehir {3}, Anne'deki olduğu pozisyona eklenir ve *R*1 son eklenen şehir sıra numarasına set edilir.

$$\text{Yavru–1} = \{\ 1 \rightarrow x \rightarrow \mathbf{3} \rightarrow 4 \rightarrow x \rightarrow x \rightarrow x \rightarrow 8 \rightarrow x\ \}$$

Adım 5: Anne'den *R*1 pozisyonundan eklenen {3} şehri ile karşılıklı olan Baba'daki şehir {2}, Anne'deki olduğu pozisyona eklenir ve *R*1 son eklenen şehir sıra numarasına set edilir.

$$\text{Yavru–1} = \{\ 1 \rightarrow \mathbf{2} \rightarrow 3 \rightarrow 4 \rightarrow x \rightarrow x \rightarrow x \rightarrow 8 \rightarrow x\ \}$$

Adım 6: Anne'den *R*1 pozisyonundan eklenen {2} şehri ile karşılıklı olan Baba'daki şehir {1}, Anne'deki olduğu pozisyona eklenir. {1} şehri zaten eklenmiş olduğu için Yavru–1 için çaprazlama Baba'nın kullanılmayan şehirlerin yerleştirilmesi ile son bulur.

$$\text{Yavru–1} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{7} \rightarrow \mathbf{6} \rightarrow \mathbf{9} \rightarrow 8 \rightarrow \mathbf{5}\ \}$$

Aynı çaprazlama işlemi bir kez daha tekrar ettirilerek Yavru–2 üretilir. Yavru–2 üretilirken ilk şehir Baba'dan eklenecek ve Anne'deki karşılık şehirler kontrol edilerek Baba'dan şehir ekleme işlemine devam edilecektir. Adım-6'daki durum oluşunca boş kalan genler Anne tarafından doldurulacaktır.

Çaprazlama işlemi Yavru–2 için yapılırsa aşağıdaki sonuç alınacaktır.

$$\text{Yavru–2} = \{\ 4 \rightarrow 1 \rightarrow 2 \rightarrow 8 \rightarrow \mathbf{5} \rightarrow \mathbf{6} \rightarrow \mathbf{7} \rightarrow 3 \rightarrow \mathbf{9}\ \}$$

Brady tarafından 1985'de tanıtılan çaprazlama operatörü aslında CX'in kısıtlandırılmış bir versiyonudur. Brady'ye göre, genlerin alt dizilerinin her iki ebeveynin ardışık pozisyonlarını doldurması gerekmektedir.

4.2.6. Sıralı Çaprazlama

Sıralı Çaprazlama (*Order Crossover*), çaprazlamaya tabii tutulan iki ebeveyn için ilk adımda iki tane rasgele çaprazlama noktası ($R1$ ve $R2$) oluşturur. Bu noktalar arasındaki kalan şehir dizileri oldukları gibi yavrulara aktarılır (Yavru–1[$R1$-$R2$]←Anne[$R1$-$R2$] ve Yavru–2[$R1$-$R2$]←Baba[$R1$-$R2$]). Büyük olan çaprazlama noktasından ($R2$>$R1$) başlamak üzere her iki ebeveyn kaydırılarak elde edilen *Ao* ve *Bo* turları yavrulara $R2$ noktasından $R1$ noktasına varıncaya kadar yerleştirilir (Yavru-1'e *Bo* ve Yavru-2'ye *Ao* kullanılmayan genleri uygulanır) (Davis 1985).

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

Adım 1: Rasgele iki sayı üretilir. ($R1$=4 ve $R2$=7)

Adım 2: Üretilen sayılara arasında kalan şehirler yavrulara aktarılır.

$$\text{Yavru–1} = \{\ x \rightarrow x \rightarrow x \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow x \rightarrow x\ \}$$

$$\text{Yavru–2} = \{\ x \rightarrow x \rightarrow x \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow x \rightarrow x\ \}$$

Adım 3: $R2$'den n'ye, 1'den $R1$'e kadar olacak şekilde *Ao* ve *Bo* turları üretilir.

$$Ao = \{\ 8 \rightarrow 9 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7\ \}$$

$$Bo = \{\ 9 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6\ \}$$

Adım 4: *Ao* ve *Bo* kaydırılmış turlarındaki kullanılmayan şehirler ($R2$+1) noktasından başlamak üzere sırayla yavrulara yerleştirilir.

Yavru–1 = { **2** → **1** → **8** → 4 → 5 → 6 → 7 → **9** → **3** }

Yavru–2 = { **3** → **4** → **5** → 1 → 8 → 7 → 6 → **9** → **2** }

Sıralı Çaprazlama (OX) operatörünün bir türevi olan MPX (*Maximal Preservative Crossover*) Schleuter (1989) tarafından sunulmuştur.

4.2.7. Değiştirilmiş Sıralı Çaprazlama

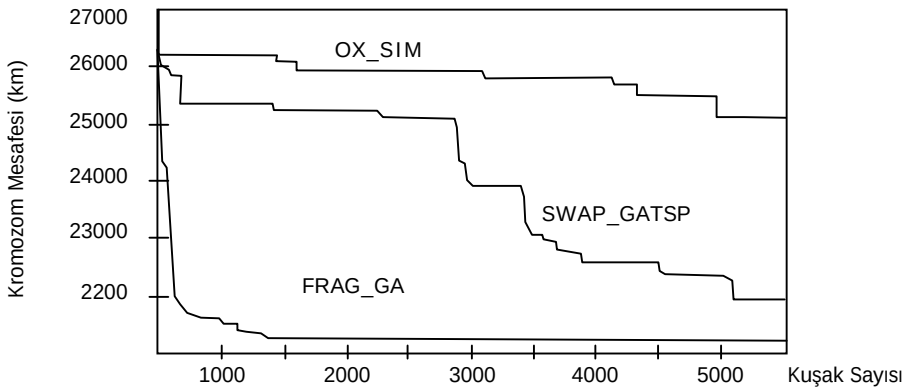
Değiştirilmiş Sıralı Çaprazlama (*Modified Order Crossover*) aslında OX çaprazlama metodunda belirlenen *R*1 ve *R*2 rasgele sayılarının gen sayısı ile ilintili olarak seçilmesi ile ilgili yapılan bir düzenlemesidir. Değiştirilmiş Sıralı Çaprazlamada (MOX) tekrarlanacak olan adımlar aynı OX metodunda olduğu gibidir. *R*1 ve *R*2 sayılarının seçimi aşağıda verilen kurallara göre yapılmaktadır (Ray ve ark. 2004, 2005).

- Rasgele belirlenecek olan alt dizi uzunluğu (α) GSP'deki şehir sayısıyla ilgilidir ve $\alpha = Rnd(n/7 - n/9) + n/9$ alt dizi uzunluğu için idealdir. Bu durumda rasgele belirlenen alt dizi uzunluğu ($n/9 \le \alpha \le n/7$) aralığında olacaktır.
- Çaprazlama işlemi için uygulanacak alt dizi uzunluğunun (y) en az 2 olması gerekmektedir. Fakat küçük boyutlu GSP problemlerinde rasgele belirlenen alt dizi uzunluğu 1 olarak hesaplanabilmektedir. Çaprazlamada uygulanacak olan alt dizi uzunluğunun belirlenmesinde bu problem $y = max\{2, \alpha\}$ eşitliği uygulanarak engellenir.
- OX çaprazlama için gereken rasgele çaprazlama noktaları olan *R*1 ve *R*2 uygulanacak alt dizi uzunluğuna göre belirlenir. *R*1 çaprazlama noktası *R*2'den küçüktür. $R1 = Rnd(n - y)$ ve $R2 = R1 + y$ şeklinde hesaplanarak OX için gereken çaprazlama noktaları parametreleri problem boyutuna göre elde edilmiş olur.

Tablo 4.2. OX ve MOX Operatörlerinin Çeşitli Algoritmalardaki Sonuçları

		En İyi Bulunan Değerler		
Problem	Optimum Değer	FRAG_GA	SWAP_GATSP	OX_SIM
Grtschels24	1272	**1272**	**1272**	**1272**
KroA100	21282	**21282**	21504	22400
D198	15780	**15834**	15992	16720
Ts225	126643	**126730**	127012	135800
Pcb442	50778	**51104**	52620	53402
Rat783	8806	**9007**	9732	10810

Ray ve arkadaşları (2005) yaptıkları çalışmalarda GA'ya çeşitli sezgisel yaklaşımlar (başlangıç popülasyonu NN ile üretilmekte, her üretim işleminde önce bireylere NF operatörü uygulanmaktadır) eklenmesiyle ve MOX operatörünün kullanılmasıyla oluşan sisteme FRAG_GA adını vermektedirler. Yine Ray ve arkadaşlarının (2004) yaptığı diğer bir çalışmada, SWAP_GATSP adını verdikleri GA'yı çeşitli operatörler ile birlikte (*Knowledge Based Multiple Inversion, Knowledge Based Swapping*) kullanılmıştır. OX_SIM ise BGA'da OX çaprazlama operatörü ile SIM mutasyon operatörünün kullanılması ile elde edilen sonuçları göstermektedir. Sonuç olarak Ray ve arkadaşlarının (2004, 2005) Tablo 4.2'de sundukları değerlerden, sezgisel yaklaşımlarla beraber kullanılan MOX operatörünün daha efektif olduğu görülmektedir. MOX ve OX operatörlerinin kullanıldığı GA modelleri arasındaki performans farkı Şekil 4.5'de daha net olarak görülebilmektedir.



Şekil 4.5. MOX ve OX Operatörlerinin KroA100 Problemindeki Performansları

4.2.8. Sıra Temelli Çaprazlama

Sıra Temelli Çaprazlama (*Order Based Crossover*) metodu da pozisyon tabanlıdır. Sıra Temelli Çaprazlamada (OBX), ebeveynlerin birinden rasgele seçilen *M* gen diğer ebeveyndeki aynı genlerin pozisyonlarına seçilen sırada yerleştirilerek Yavru–1 elde edilir. Aynı işlem eşler değiştirilerek tekrarlandığında Yavru-2'de elde edilmiş olacaktır (Syswerda 1990).

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$
$$\text{Baba} = \{\ 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

Adım 1: Anne'den rasgele *M* adet gen seçilir $\{\ 2 \rightarrow 4 \rightarrow 6 \rightarrow 7\ \}$.

Adım 2: Seçilen genler $\{\ 2 \rightarrow 4 \rightarrow 6 \rightarrow 7\ \}$ Baba'nın aynı genlerine seçim sırasına göre eklenerek Yavru–1 elde edilir.

$$\text{Yavru–1} = \{\ x \rightarrow 5 \rightarrow x \rightarrow 1 \rightarrow 8 \rightarrow x \rightarrow x \rightarrow 9 \rightarrow 3\ \}$$
$$\text{Yavru–1} = \{\ \mathbf{2} \rightarrow 5 \rightarrow \mathbf{4} \rightarrow 1 \rightarrow 8 \rightarrow \mathbf{6} \rightarrow \mathbf{7} \rightarrow 9 \rightarrow 3\ \}$$

Adım 3: Aynı seçilen genler Baba temel alınarak tekrarlandığında Yavru–2 elde edilir.

$$\text{Seçilen Genler} : \{\ 4 \rightarrow 2 \rightarrow 7 \rightarrow 6\ \}$$
$$\text{Yavru–2} = \{\ 1 \rightarrow x \rightarrow 3 \rightarrow x \rightarrow 5 \rightarrow x \rightarrow x \rightarrow 8 \rightarrow 9\ \}$$
$$\text{Yavru–2} = \{\ 1 \rightarrow \mathbf{4} \rightarrow 3 \rightarrow \mathbf{2} \rightarrow 5 \rightarrow \mathbf{7} \rightarrow \mathbf{6} \rightarrow 8 \rightarrow 9\ \}$$

4.2.9. Pozisyon Temelli Çaprazlama

Pozisyon Temelli Çaprazlama (*Position Based Crossover*), OX operatörüne oldukça benzemektedir. Pozisyon Temelli Çaprazlamada (PBX), ebeveynlerin birinden rasgele seçilen *M* genli bir alt dizi aynı pozisyonda olmak üzere yavruya eklenir. Yavrunun boş kalan genleri diğer ebeveynden tekrarlanmayan şehirlerle doldurulur. Aynı işlem ebeveynleri değiştirerek tekrarlandığında Yavru–2 elde edilmiş olur (Syswerda 1990).

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6 \rightarrow 9 \rightarrow 3\ \}$$

Adım 1: Anne'den {*M*} genli rasgele bir alt dizi seçilir.

$$\text{Anne} = \{\ 1 \rightarrow 2 \rightarrow \mathbf{3} \rightarrow 4 \rightarrow \mathbf{5} \rightarrow 6 \rightarrow 7 \rightarrow \mathbf{8} \rightarrow 9\ \}$$

Adım 2: Anneden rasgele seçilen {*M*} gen Yavru-1'e aynen eklenir.

$$\text{Yavru–1} = \{\ x \rightarrow x \rightarrow 3 \rightarrow x \rightarrow 5 \rightarrow x \rightarrow x \rightarrow 8 \rightarrow x\ \}$$

Adım 3: Baba'dan gelen şehirler Yavru-1'nin boş kalan genlerine yerleşir. Yerleştirme işleminde Yavru-1'de kullanılmayan şehirlere dikkat edilir.

$$\text{Yavru–1} = \{\ \mathbf{4} \rightarrow \mathbf{2} \rightarrow 3 \rightarrow \mathbf{1} \rightarrow 5 \rightarrow \mathbf{7} \rightarrow \mathbf{6} \rightarrow 8 \rightarrow \mathbf{9}\ \}$$

Adım 4: Aynı işlem eşler yer değiştirerek tekrarlandığında Yavru–2 üretilir. Aynı pozisyondaki genlerin Baba'dan seçilerek Yavru–2 üretilirse aşağıdaki gibi olacaktır.

$$\text{Yavru–2} = \{\ 1 \rightarrow 3 \rightarrow \mathbf{2} \rightarrow 4 \rightarrow \mathbf{8} \rightarrow 5 \rightarrow 6 \rightarrow \mathbf{9} \rightarrow 7\ \}$$

4.2.10. Dönüşümlü Kenar Çaprazlama

Dönüşümlü Kenar Çaprazlama (*Alternate Edges Crossover*) kenar temelli bir operatördür. Dönüşümlü Kenar Çaprazlamaya (EX), ebeveynlerden birisinden rasgele bir (*I*, *J*) kenarı seçilerek başlanır. Ardından diğer ebeveynden (*J*, *K*) kenarı seçilir. Bu kenar seçim işlemi eşler arasında ardışık bir sırada devam eder. Seçilen her kenar yavruya eklenmektedir. Eklenen bir gene rastlandığı zaman yeni gen eklenmemiş olanlardan rasgele seçilmektedir (Grefenstette ve ark. 1985).

$$\text{Anne} = \{\ 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 4 \rightarrow 2 \rightarrow 8 \rightarrow 7 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

Adım 1: Anne'den rasgele bir (*I*, *J*) kenarı seçilir ve yavrunun aynı pozisyonuna eklenmektedir. { (*I*, *J*) = (4, 2) }.

Yavru–1 = { x → x → x → x → 4 → 2 → x → x → x }

Adım 2: (*I*, *J*) = (4, 2) olarak eklendikten sonra Baba'dan (*J*, *K*) kenarı (2, 3) eklenir.

Yavru–1 = { x → x → x → x → 4 → 2 → **3** → x → x }

Adım 3: (*J*, *K*) = (2, 3) olarak eklendikten sonra Anne'den (*K*, *I*) kenarı (3, 5) eklenir.

Yavru–1 = { x → x → x → x → 4 → 2 → 3 → **5** → x }

Adım 4: (*K*, *I*) = (3, 5) olarak eklendikten sonra Baba'dan (*I*, *J*) kenarı (5, 7) eklenir.

Yavru–1 = { x → x → x → x → 4 → 2 → 3 → 5 → **7** }

Adım 5: (*I*, *J*) = (5, 7) olarak eklendikten sonra Anne'den (*J*, *K*) kenarı (7, 9) eklenir.

Yavru–1 = { **9** → x → x → x → 4 → 2 → 3 → 5 → 7 }

Adım 6: (*J*, *K*) = (7, 9) olarak eklendikten sonra Baba'dan (*K*, *I*) kenarı (9, 1) eklenir.

Yavru–1 = { 9 → **1** → x → x → 4 → 2 → 3 → 5 → 7 }

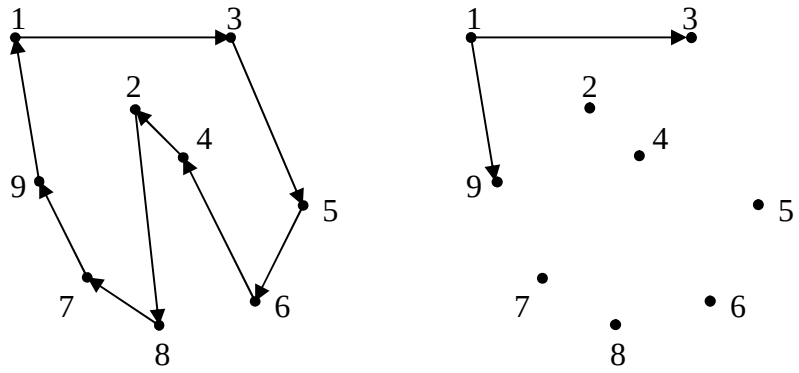
Adım 7: (*K*, *I*) = (9, 1) olarak eklendikten sonra Anne'den (*I*, *J*) kenarı (1, 3) eklenmek için seçilir. Fakat {3} şehri zaten yavruda yer aldığı için eklenmemiş rasgele bir şehir seçilir (*J* = 6).

Yavru–1 = { 9 → 1 → **6** → x → 4 → 2 → 3 → 5 → 7 }

Adım 8: (*I*, *J*) = (1, 6) olarak eklendikten sonra Baba'dan (*J*, *K*) kenarı (6, 5) eklenmek için seçilir. Fakat {5} şehri zaten yavruda yer aldığı için eklenmemiş rasgele bir şehir seçilir (*K*=5). Yavru-2'yi üretmek için aynı prosedürler tekrarlanır.

Yavru–1 = { 9 → 1 → 6 → 5 → 4 → 2 → 3 → 5 → 7 }

S-GSP problemleri için Dönüşümlü Kenar Çaprazlama (EX) çaprazlama operatöründe her şehir iki farklı kenarda yer almaktadır (Şekil 4.6). Çaprazlama sırasında eğer tekrarlanan gen olursa diğer alternatif kontrol edilmelidir. Örneğin Adım 7'de oluşan tekrarlama işlemi için Anne'den $(I, J)_1 = (1, 3)$ kenarı denendikten sonra eklenemediği için $(I, J)_2 = (1, 9)$ kenarının da eklenebilirliği kontrol edilmelidir. Eğer $(I, J)_2$ kenarı eklenebilir ise eklenmelidir. Adım 8'de ise Baba'dan $(J, K)_1=(6, 5)$ kenarı eklenemiyorsa $(J, K)_2 = (6, 3)$ kenarının eklenebilirliği kontrol edilmelidir. Eğer $(J, K)_2$ kenarı da eklenemiyorsa o zaman rasgele bir kenar seçilmelidir $(J, K) = (6, x)$.



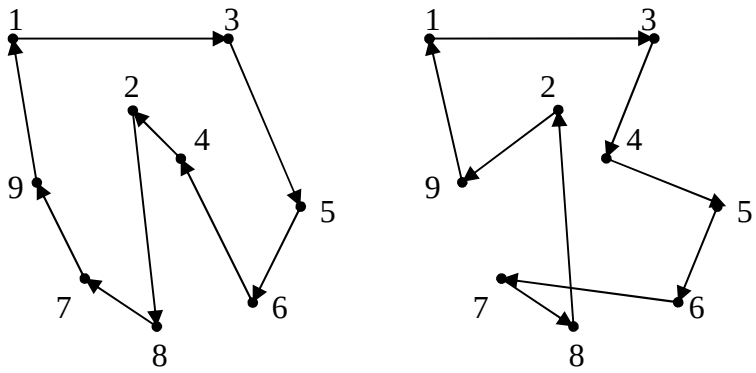
Şekil 4.6. EX için Eklenmeye Çalışılan Şehrin Yapacağı Kenarlar

4.2.11. Yeniden Kenar Birleşimi Çaprazlama

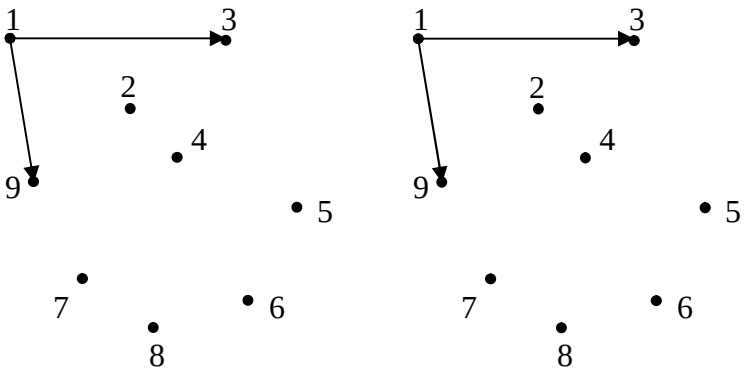
Yeniden Kenar Birleşimi Çaprazlaması (*Edge Recombination Crossover*) kenar temelli çaprazlamalardandır. Çaprazlamaya tutulan her iki ebeveynin kenar haritalarının çıkartılması ile çaprazlama işlemine hazırlanılır. Her iki ebeveyn (Şekil 4.7) dikkate alındığında herhangi bir noktadan oluşan en az 2 (Şekil 4.8) en fazla 4 (Şekil 4.9) kenar oluşabilmektedir. Yeniden Kenar Birleşimi Çaprazlamaya (ERX) kenar haritasındaki en az kenar sayısına sahip olan bir şehir ile başlanır ve bu şehir yavruya eklenir. Eklenen şehrin haritasında bulunan kenarı tamamlayacak olan şehirlerin hangisinin ekleneceği haritadaki kenar sayısı büyüklüklerine bakarak seçilir. Bu seçimde daha az kenar oluşturabilen şehirler tercih edilir. Bu işleme tüm şehirler eklenınceye kadar devam edilir. Eğer eklenen şehrin haritasında yer alan

elemanların hepsi eklenmiş ise eklenmemiş bir şehir rasgele seçilerek çaprazlamaya devam edilir (Whitley 1989).

Kenar haritasına *H* diyecek olursak: *H* dizisi bellekte (*n**4*2+*n*) byte yer kaplar. Burada bellek tahsisi için her şehrin oluşturabileceği maksimum 4 kenar olmasından ve her şehir 2 byte ile ifade edilmesinden dolayı *n**4*2 byte'lık bir alana ve her şehrin kaç kenar oluşturduğunun sayısını tutan *n* byte'lık bir alana ihtiyaç duyulmaktadır. Ayrıca kenar temelli çaprazlama metotları, harita oluşturma işlemleri ve kontrol yaklaşımlarının hesaplamalarından dolayı daha fazla zaman ve bellek ihtiyacı doğurmaktadır.

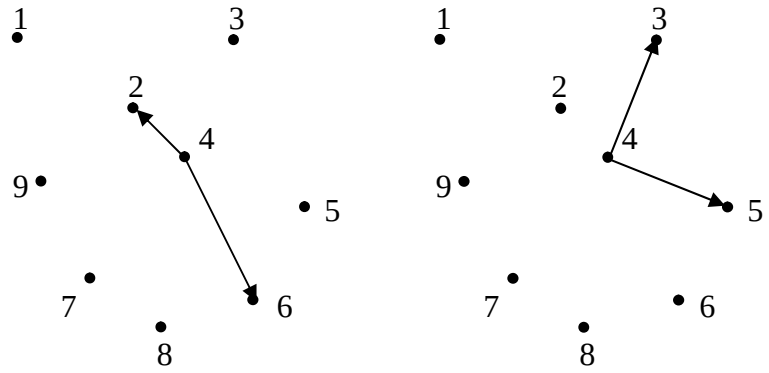


Şekil 4.7. ERX için Seçilen Ebeveynler



Şekil 4.8. ERX'de Ebeveynlerden {1} Numaralı Şehir için Oluşabilecek Kenarlar

(1 numaralı şehir için oluşabilecek kenar 2 ve bu kenarlar {(1,9), (1,3)} şeklindedir)



Şekil 4.9. ERX'de Ebeveynlerden {4} Numaralı Şehir için Oluşabilecek Kenarlar

(4 numaralı şehir için 4 farklı kenar oluşturulur {(4,2), (4,6), (4,3), (4,5)})

$$\text{Anne} = \{\ 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 4 \rightarrow 2 \rightarrow 8 \rightarrow 7 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 8 \rightarrow 9\ \}$$

Adım 1: Çaprazlama işlemine başlanmadan önce her iki ebeveyn dikkate alınarak her şehrin oluşturduğu kenar haritaları çıkartılır (Tablo 4.3)

Tablo 4.3. ERX Kenar Haritası

Şehir	Kenar Oluşturan Şehirler			
1	3	9	4	
2	4	8	3	
3	1	5	2	6
4	6	2	1	
5	3	6	7	
6	5	4	3	
7	8	9	5	
8	2	7	9	
9	7	1	8	

Adım 2: Başlangıç şehri olarak {1} şehrini rasgele seçtiğimizi varsayalım. Bu şehir yavruya eklendikten sonra tüm haritadan "Kenar Oluşturan Şehirler"

bölümünden eklenen şehir {1} çıkartılacaktır. Ardından {1} numaralı şehrin kenar oluşturan şehirler listesinde yer alan şehirlerin en az uzunlukta şehir tutan listeye yönlenilerek çaprazlama işlemine devam edilecektir. {1} şehri yavruya eklenip "Kenar Oluşturan Şehirler" tablosundan çıkartılınca tablodaki şehirlerin kenar oluşturabilecek şehir sayıları: 1.Şehir=0, 2. Şehir=3, 3. Şehir=3, 4. Şehir=2, 5. Şehir=3, 6. Şehir=3, 7.Şehir=3, 8. Şehir=3 ve 9. Şehir=2 şeklinde olacaktır. {1} numaralı şehrin listesinde yer alan {3, 9, 4} şehirlerinden en az liste uzunluğuna sahip olan şehir seçilir (9 veya 4 arasında rasgele seçim yapılır).

$$\text{Yavru–1} = \{\ 1 \rightarrow 9 \rightarrow 7 \rightarrow 8 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 5 \rightarrow 3\ \}$$

Adım 3: Bu şekilde çaprazlama işlemine devam edilir. Eğer eklenen noktanın listesinde yer alan tüm şehirlerin liste uzunlukları sıfır ise tüm o şehirler yavruya aktarılmış demektir ki eklenecek olan şehir eklenmemiş olanlar arasından rasgele seçilerek tüm şehir ekleninceye kadar çaprazlamaya devam edilir.

Adım 4: Aynı işlem başlangıç noktası değiştirilerek tekrarlanır ve Yavru–2 elde edilir.

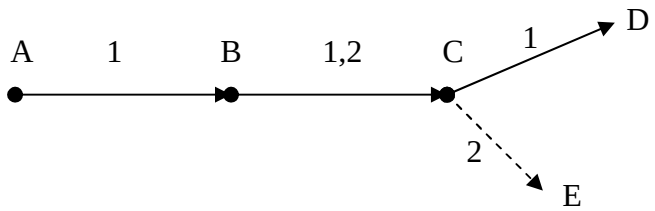
ERX operatörü için literatürde faklı geliştirmeler mevcuttur. Standart ERX metodundan daha efektif iki yeni ERX metodu Mathias ve Whitley tarafından (1992) öne sürülmüştür. Birinci metotta (Edge–2) kenar haritası oluşturulurken iki kere gelen (tekrarlanan) kenarlar (-) işareti ile işaretlenip, sonraki şehir seçilirken bunlara öncelik tanınmasını temel almaktadır.

Mathias ve Whitley (1992) tarafından tanımlanan ikinci metotta (Edge–3) ise çaprazlama sırasında hataların azaltılarak ebeveynlerin genetik özelliklerini yavrulara daha fazla aktarmasını sağlamaktadır. Edge–3 operatörü bir hata ile karşılaştığı zaman, hemen rasgele bir şehir seçimi işlemine gitmez. Yavruya aktarılmış olan şehirlerin diğer başlangıç noktasındaki şehirle oluşturacağı kenarlar incelenir ve eğer diğer taraftaki şehirden oluşturulabilecek kenar mevcut ise yavruya kopyalanmış alt tur parçası ters döndürülerek devam çaprazlama işlemine devam edilir. Yavrudaki

parça turun her iki tarafındaki (başlangıç ve bitiş) şehirlerin “Kenar Oluşturan Şehirler” listesinde şehir bulunmadığı zaman rasgele bir şehir seçilir.

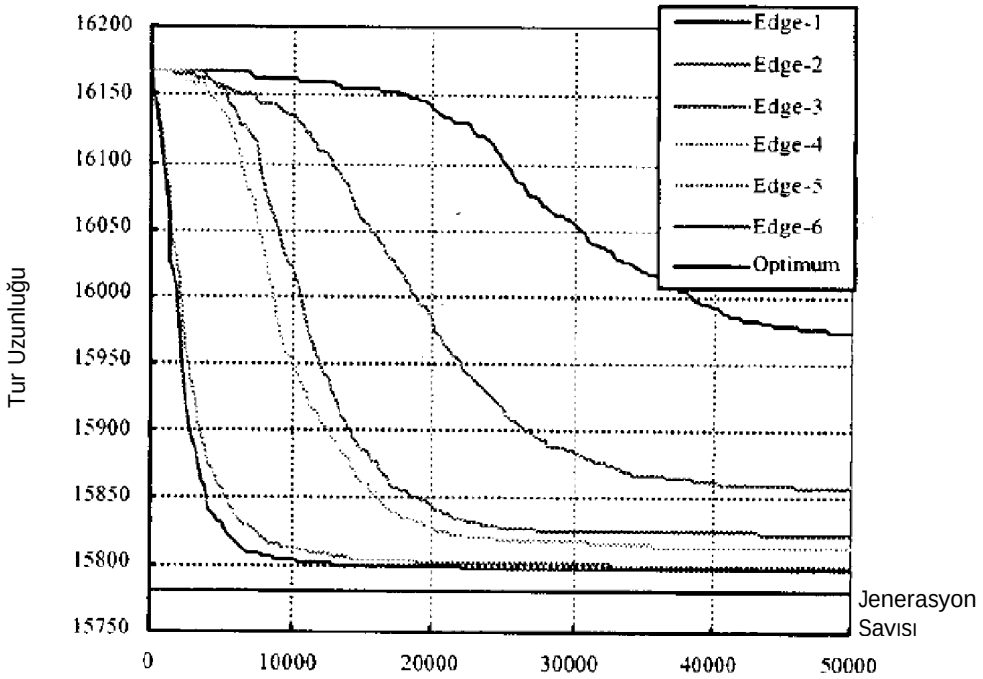
Dzubera ve Whitley (1994) tarafından tanımlanan Edge–4 operatörü hata durumunda yavrudaki parçalı turun bir parçasının döndürülmesini temel alır. Döndürülecek olan alt parçanın başlangıç kısmı şu şekilde belirlenir. Eklenecek olan şehrin komşu listesindekiler haricinde kalan tüm şehirler yavrudaki şehirlerde incelenir ve en az “Kenar Oluşturan Şehirler” liste uzunluğuna sahip olan şehirden son noktaya kadar olan parçalı alt tur ters çevrilir. Eğer bu işlemler sonucunda hala şehir eklenemiyorsa, rasgele bir şehir seçilerek işleme önceki gibi devam edilir.

Nguyen ve arkadaşları (2000) tarafından da iki yeni ERX metodu (Edge–5 ve Edge–6) tanımlanmış ayrıca yukarıda bahsedilen diğer ERX metotları ile karşılaştırılmıştır. Edge–5 metodunun çalışma prensibi ortak kenarların hangi ebeveynden geldiği temeline dayanmaktadır. Eklenmiş şehirden hangi şehre gidileceği, en son hangi ebeveynden kenar oluşturulmuş ise listede bulunan ve kenar oluşturan o şehre gidilecektir. Bu seçimi bir örnek üzerinde açıklayacak olursak:



Şekil 4.10. ERX-5 (Edge–5) İçin Kenar Seçimi

C şu andaki şehir ve parçalı tur {A,B,C}’den oluşmaktadır (Şekil 4.10). {AB} kenarı Anne’den (1) ve {BC} kenarı ise ortak kenardır (hem Anne’de (1) hem de Baba’da (2) mevcuttur). Şimdi {C} şehrinin kenar listesinde {D,E} şehirleri vardır. Her ikisinin de eşit sayıda kalan kenarları var. {CD} ve {CE} kenarları Anne (1) ve Baba’ya (2) aittir. Burada {CD} tercih edilecektir. Çünkü önceden tercih edilmiş ortak olmayan kenar {AB} Anne’den geldiği için öncelik Anne’ye aittir.



Şekil 4.11. ERX ve Türevlerinin Çözümdeki Etkileri

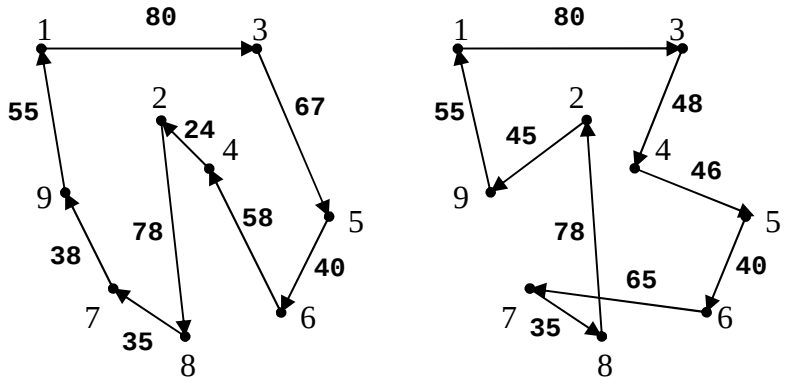
Edge–6 metodu Edge–5'e dayanmaktadır. Hata sayısını azaltmak için bir diğer yöntemidir. Edge–6 parçalı turun bir ucunun diğer ucuna kenarı olup olmadığını kontrol eder. Eğer öyle bir kenar varsa geriye eklenmemiş kenarı kalan ilk şehri bulmak için parçalı turu inceler. Eğer eklenmemiş bir şehir varsa parçalı turun iki ucunu birbirine bağlar ve eklenmemiş kenarı olan şehri ve komşusunu birbirinden ayırır. Bu komşulukta ebeveynlerde ortak olmayan kenar tercih edilir.

Nguyen ve arkadaşları (2000) tarafından tüm ERX çaprazlama metotları 3 farklı GSP problemine uygulanmış ve en etkili metodun Edge–6 olduğu görülmüştür. Şekil 4.11'deki grafik D198 problemine uygulanmış farklı ERX metotlarının çözümdeki etkilerini göstermektedir.

4.2.12. Sezgisel Çaprazlama

Sezgisel Çaprazlama (*Heuristic Crossover*) operatörü de kenar temelli bir operatördür. Sezgisel Çaprazlama (HX) için ilk başta ebeveynlerin birinden rasgele bir başlangıç noktası seçilir. Seçilen başlangıç noktasından ayrılarak oluşan her iki

ebeveyndeki kenarların uzunlukları kontrol edilerek en kısa kenarı oluşturan şehir seçilir. Eğer seçilen şehir zaten yavruya eklenmiş durumda ise eklenecek olan şehir rasgele seçilir. Bu işlem tüm şehirler ekleninceye kadar devam eder. Şekil 4.12'de anne ve babanın turları ve kenarlar üzerinde uzunlukları verilmiştir (Grefenstette ve ark. 1985, Grefenstette 1987).



Şekil 4.12. HX için Bireyler ve Kenar Uzunlukları

$$\text{Anne} = \{\ 1 \rightarrow 3 \rightarrow 5 \rightarrow 6 \rightarrow 4 \rightarrow 2 \rightarrow 8 \rightarrow 7 \rightarrow 9\ \}$$

$$\text{Baba} = \{\ 1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 2 \rightarrow 9\ \}$$

Adım 1: Rasgele bir başlangıç noktası seçilir ve yavruya eklenir ($R1$=3).

Adım 2: Anne ve Baba'dan yavruya son eklenen şehir {3} ile birlikte oluşan kenarların {(1,3), (3,5), (3,4)} uzunlukları kontrol edilir ve en kısa kenarı oluşturan şehir {(3,**4**)=48} seçilir.

Adım 3: Seçilen şehir yavruda yer almadığı için yavruya eklenir ve 2. adıma geri dönülür. Eğer seçilen şehir yavruda yer alıyorsa eklenecek olan şehir eklenmemiş şehirler içinden rasgele seçilir.

Adım 4: Tüm şehirler ekleninceye kadar Adım 2 ve Adım 3 tekrarlanır. Yavru-2'yi üretmek için aynı prosedürler tekrarlanır.

$$\text{Yavru–1} = \{\ 3 \rightarrow 4 \rightarrow 2 \rightarrow 9 \rightarrow 7 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 1\ \}$$

4.2.13. Sezgisel Çaprazlama–2

Sezgisel Çaprazlama–2 (HX–2), HX metoduna göre ufak bir modifikasyon içermektedir. İlk başta ebeveynlerin birinden rasgele bir başlangıç noktası (*R*1) seçilir. Seçilen başlangıç noktasından ayrılarak oluşan her iki ebeveyndeki kenarların uzunlukları kontrol edilerek en kısa kenarı oluşturan şehir seçilir. Eğer seçilen şehir zaten yavruya eklenmiş durumda ise diğer ebeveynin başlangıç noktasından oluşturduğu kenar seçilir. Eğer diğer ebeveynden de gelen şehir yavruya eklenmiş durumda ise o zaman eklenecek olan şehir rasgele seçilir. Bu işlem tüm şehirler ekleninceye kadar devam eder (Suh ve Gucht 1987, Liepins ve ark. 1987).

HX ve HX–2 dışında ayrıca başka bir modifikasyon daha vardır. HX–2 rasgele eklenecek olan şehrin büyüklüğü parametresel olarak belirlenen en kısa kenarları içeren havuzdan rasgele seçilmesini önermektedir (Jog ve ark. 1989).

4.2.14. Açgözlü Alt Tur Çaprazlama

Açgözlü Alt Tur Çaprazlamaya (*Greedy Sub Tour Crossover*), rasgele bir şehir (*R*) numarasının her iki ebeveynde de aynı olmak üzere seçilmesiyle başlanır. Bu rasgele seçilen şehir yavruya eklendikten sonra çaprazlamaya devam edilir. Rasgele seçilen *R* şehrinin Anne'deki pozisyonundan geriye, Baba'daki pozisyonundan ileriye doğru şehirlerin seçimi ve yavruya eklemesi yapılır. Her iki bireyden de aynı anda eklenecek şehir olmadığında, ziyaret edilmemiş şehirler yavruya rasgele sırada yerleştirilerek Açgözlü Alt Tur Çaprazlaması (GSX) sonlandırılmış olur (Sengoku ve Yoshihara 1998).

$$\text{Anne} = \{\ 4 \rightarrow 8 \rightarrow 2 \rightarrow 1 \rightarrow \mathbf{3} \rightarrow 6 \rightarrow 7 \rightarrow 5 \rightarrow 9\ \}$$
$$\text{Baba} = \{\ 2 \rightarrow \mathbf{3} \rightarrow 4 \rightarrow 7 \rightarrow 8 \rightarrow 6 \rightarrow 5 \rightarrow 1 \rightarrow 9\ \}$$

Adım 1: Rasgele bir şehir numarası seçilir $R = 3$ ve bu şehir yavruya eklenir.

$$\text{Yavru–1} = \{\ x \rightarrow x \rightarrow x \rightarrow \mathbf{3} \rightarrow x \rightarrow x \rightarrow x \rightarrow x \rightarrow x\ \}$$

Adım 2: Anne'de *R* şehrinin bulunduğu şehirden bir önceki şehir yavrudaki *R* şehrinin bulunduğu yerden öncesine, Baba'daki *R* şehrinin bulunduğu şehirden bir sonraki şehir yavrudaki *R* şehrinin bulunduğu yerin sonrasına olacak şekilde tekrarlamalı bir şekilde eklene işlemine devam edilir.

$$\text{Yavru–1} = \{ x \rightarrow x \rightarrow \mathbf{1} \rightarrow 3 \rightarrow \mathbf{4} \rightarrow x \rightarrow x \rightarrow x \rightarrow x \}$$

$$\text{Yavru–1} = \{ x \rightarrow \mathbf{2} \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow \mathbf{7} \rightarrow x \rightarrow x \rightarrow x \}$$

$$\text{Yavru–1} = \{ \mathbf{8} \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow x \rightarrow x \rightarrow x \}$$

Adım 3: Her iki ebeveynden eklenebilirlik durumu "*False*" olduğu zaman eklenilmeyen şehirler rasgele bir sırada yavrudaki boşluklara yerleştirilir.

$$\text{Yavru–1} = \{ 8 \rightarrow 2 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow \mathbf{6} \rightarrow \mathbf{5} \rightarrow \mathbf{9} \}$$

Adım 4: Yavru-2'yi üretme işlemi de Adım-1'den başlamak üzere yeni bir çaprazlama yapılmasından ibarettir.

4.2.15. Mesafe Korumalı Çaprazlama

Mesafe Korumalı Çaprazlama (*Distance Preserving Crossover*) işlemine tabii tutulacak olan anne ve babanın benzer alt turlarından parçalar oluşturulması bunların Açgözlü Birleştirme (*Greedy Reconnection* (GR)) prosedürüyle birleştirilmesini temel almaktadır. Mesafe Korumalı Çaprazlamada (DPX) ilk başta Anne ve Baba'nın benzer alt turlarından bir *Parçalar* dizisi (*fragments*) oluşturulur. Daha sonra bu dizideki parçalar alt turlar en yakın komşuluk kuralına göre birleştirilir. Buradaki en yakın komşuluk kuralına göre birleştirme işlemi GR prosedürüne karşılık gelmektedir (Freisleben ve Merz 1996).

$$\text{Anne} = \{ \boxed{5 \rightarrow 3 \rightarrow 9} \rightarrow \boxed{1 \rightarrow 2} \rightarrow \boxed{8} \rightarrow \boxed{4 \rightarrow 6} \rightarrow \boxed{7} \}$$

$$\text{Baba} = \{ \boxed{1 \rightarrow 2} \rightarrow \boxed{5 \rightarrow 3 \rightarrow 9} \rightarrow \boxed{8} \rightarrow \boxed{6 \rightarrow 4} \rightarrow \boxed{7} \}$$

Adım 1: Anne ve Baba'dan gelen komşuluklara göre alt tur parçaları oluşturulur.

$$\textit{Parçalar} = \{ \boxed{5 \rightarrow 3 \rightarrow 9} \rightarrow \boxed{1 \rightarrow 2} \rightarrow \boxed{8} \rightarrow \boxed{4 \rightarrow 6} \rightarrow \boxed{7} \}$$

Adım 2: Rasgele bir başlangıç şehri seçilir (*R*=6). Seçilen şehir (6) ve bu şehrin bulunduğu parçanın diğer bitiş noktası olan (4) *Parçalar'*da kullanılmayan diğer alt turlardan birisi ile en kısa bağlantıyı yapabilecek şekilde yavruya eklenir (bağlantı için gerekiyor ise alt turlar döndürülmelidir).

$$\text{Yavru–1} = \{ \boxed{6 \rightarrow 4} \rightarrow \boxed{5 \rightarrow 3 \rightarrow 9} \rightarrow x \rightarrow x \rightarrow x \rightarrow x \}$$

Adım 3: *Parçalar*'da yer alan tüm alt turlar kullanılıncaya kadar Adım 2 işlemine devam edilir.

$$\text{Yavru–1} = \{ 6 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 9 \rightarrow 7 \rightarrow 2 \rightarrow 1 \rightarrow 8 \}$$

Adım 4: Yavru-2'de bu çaprazlama işlemi tekrarlanarak elde edilir.

4.3. Mutasyon Metotları

GA'ların bir diğer önemli bileşeni olan mutasyon operatörleri, optimizasyon sürecinde popülasyonun düştüğü yerel optimumdan kurtararak çözüm uzayındaki görülmemiş örneklerden optimizasyona devam edilmesini amaçlamaktadır. Bu amaçla GA'larda kullanılan birçok mutasyon operatörü mevcuttur.

4.3.1. Değiş-Tokuş Mutasyonu

Değiş-Tokuş Mutasyonu (*Exchange Mutation*) oldukça yaygın olarak bilinen mutasyon operatörüdür. *Swap Mutation* olarak da bilinir. Değiş-Tokuş Mutasyon (EXC) operatöründe bireyden rasgele pozisyonlardan iki nokta (*R*1 ve *R*2) seçilir. Seçilen bu noktalar karşılıklı olacak şekilde yer değiştirir (Banzhaf 1990).

EXC operatörü sadece iki yer değiştirme haricinde bir *k* parametresi ile *k-Swap* operatörü olarak da kullanılabilir. Aynı zamanda bu yer değiştirme işlemlerinde mutasyon içerisine eklenecek olan bir kontrol yaklaşımı ile (toplam kazanç: mesafe analizi) daha iyi neticeler alınabilir (*Greedy Swap*).

$$\text{Birey} = \{ 3 \rightarrow 8 \rightarrow 1 \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow 6 \rightarrow 2 \rightarrow 5 \}$$

Adım 1: Rasgele iki nokta seçilir ($R1$=3 ve $R2$=7).

Adım 2: Bu seçilen noktalar karşılıklı yer değiştirir.

$$\text{Birey} = \{\ 3 \rightarrow 8 \rightarrow \mathbf{6} \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow \mathbf{1} \rightarrow 2 \rightarrow 5\ \}$$

4.3.2. Yerdeğiştirme Mutasyonu

Yerdeğiştirme Mutasyonuna (*Displacement Mutation*) uğrayacak olan bireyden ilk etapta bir alt tur seçilir. Yerdeğiştirme Mutasyonunda (DISP) alt tur seçimi için iki rasgele nokta üretilmelidir ($R1$ ve $R2$). Belirlenen alt tur bireyden çıkartılır ve geriye kalan kısmi bireyin herhangi bir pozisyonuna (rasgele belirlenen $R3$ ile) alt tur eklenir (Michalewicz 1992).

$$\text{Birey} = \{\ 3 \rightarrow 8 \rightarrow 1 \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow 6 \rightarrow 2 \rightarrow 5\ \}$$

Adım 1: Rasgele iki nokta belirlenerek alt tur seçilir ($R1$=3 ve $R2$=7).

Adım 2: Seçilen alt tur {1, 4, 9, 7, 6} bireyden çıkartılır. Böylece alt tur çıkartılmış birey elde edilir. Yeni bireyin dizi uzunluğu $DU = (n - |R1 - R2| - 1)$ formülü ile bulur.

$$\text{Birey} = \{\ 3 \rightarrow 8 \rightarrow 2 \rightarrow 5\ \}$$

Adım 3: [1-DU] arasında rasgele bir $R3$ sayısı belirlenir. Bu sayı kaldırılan [$R1$-$R2$] alt turunun ekleneceği pozisyonun başlangıç noktasını belirlemektedir. Örneğin $R3$=2 olduğu zaman mutasyona uğrayan birey aşağıdaki gibi olacaktır.

$$\text{Birey} = \{\ 3 \rightarrow \mathbf{1} \rightarrow \mathbf{4} \rightarrow \mathbf{9} \rightarrow \mathbf{7} \rightarrow \mathbf{6} \rightarrow 8 \rightarrow 2 \rightarrow 5\ \}$$

4.3.3. Tersleme Mutasyonu

Tersleme Mutasyonu (*Inversion Mutation*), Yerdeğiştirme operatörüne (DISP) oldukça benzemektedir. Tek farkı alt turun terslenerek eklenmesidir. Tersleme Mutasyonu (INV) için DISP operatöründe kullanılan aynı birey ve $R1$, $R2$, $R3$ rasgele değerleri için mutasyona uğramış birey aşağıdaki örnekteki gibi olacaktır (Fogel 1990).

Birey = { 3 → **6**→ **7**→ **9**→ **4**→ **1**→ 8→ 2 → 5 }

4.3.4. Ekleme Mutasyonu

Ekleme Mutasyonuna (*Insertion Mutation*) uğrayacak bireyden rasgele bir nokta (*R*1) seçilir ve bu noktadaki şehir turdan kaldırılır. Kaldırılan bu şehir, kalan turdaki rasgele seçilen diğer bir noktaya (*R*2) eklenir. Ekleme Mutasyon (INS) operatörü, DISP mutasyon operatörünün alt tur uzunluğunun 1 seçilmesinin özel bir durumudur (Fogel 1988).

Birey = { 3 → 8 → 1 → 4 → 9 → 7 → 6 → 2 → 5 }

Adım 1: Kaldırılmak üzere rasgele bir nokta seçilir (*R*1=3 için Şehir={1}).

Birey = { 3 → 8 → 4 → 9 → 7 → 6 → 2 → 5 }

Adım 2: Kaldırılan şehir (1) rasgele seçilen başka bir noktaya eklenir (*R*2=6).

Birey = { 3 → 8 → 4 → 9 → 7 → 1 → 6 → 2 → 5 }

4.3.5. Basit Tersleme Mutasyonu

Basit Tersleme Mutasyon (*Simple Inversion Mutation*) operatörüne girecek olan bireyden rasgele iki kesme noktası (*R*1 ve *R*2) seçilir. Basit Tersleme Mutasyonu (SIM), Tersleme Mutasyonun yerdeğiştirme özelliğinin olmamasından ibarettir. SIM operatörü, seçilen kesme noktaları arasında kalan alt turu kendi içerisinde tersine çevirir (Holland 1975).

Birey = { 3 → 8 → 1 → 4 → 9 → 7 → 6 → 2 → 5 }

Adım 1: İki kesme noktası seçilir (*R*1=3 ve *R*2=7)

Adım 2: [*R*1-*R*2] aralığındaki alt tur tersine çevrilir.

Birey = { 3 → 8 → **6** → **7** → **9** → **4** → **1** → 2 → 5 }

4.3.6. Karıştırma Mutasyonu

Karıştırma Mutasyona (*Scramble Mutation*) uğrayacak bireyden rasgele bir alt tur seçilir (*R*1 ve *R*2 noktaları ile). Belirlenen alt tur içerisinde kalan şehirler rasgele aynı noktalar içerisinde karıştırılır (Syswerda 1990). Karıştırma Mutasyon (SCM) operatöründe *R*1 ve *R*2 kesme noktalarını farkının yüksek olması bireyde oldukça fazla bir bozunum gerçekleştirecektir. Bu yüzden mutlak farkın küçük seçilmesi idealdir.

$$\text{Birey} = \{\ 3 \rightarrow 8 \rightarrow 1 \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow 6 \rightarrow 2 \rightarrow 5\ \}$$

Adım 1: Alt tur seçilir (*R*1=3 ve *R*2=7).

Adım 2: Alt turdaki kromozomlar rasgele karıştırılır.

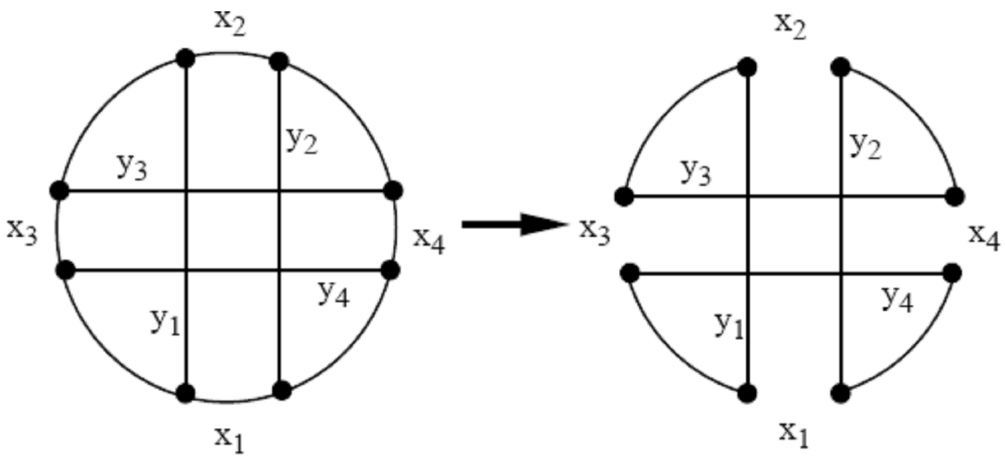
$$\text{Birey} = \{\ 3 \rightarrow 8 \rightarrow \mathbf{7} \rightarrow \mathbf{9} \rightarrow \mathbf{1} \rightarrow \mathbf{6} \rightarrow \mathbf{4} \rightarrow 2 \rightarrow 5\ \}$$

4.5.7. Çift Köprü Mutasyonu

Çift Köprü (*Double Bridge*) genellikle yerel arama metotları ile birlikte kullanılan ve tura bozunum (*perturbation*) sağlayarak yerel optimumdan bireyi kurtarmak için kullanılmaktadır. GA'da kullanılan mutasyon operatörlerinin de amacı aynı doğrultuda olmasından ve kullanılan bazı sezgisel yaklaşımlardan ve yerel arama metotları ile melez bir sistem oluşturulmasından dolayı popülasyondaki bireylerin yerel optimuma takılmasını engellemek için Çift Köprü Mutasyonu (DB) kullanılmaktadır.

Verilen turdaki rasgele seçilen $\{ x_1, x_2, x_3, x_4 \}$ kenarları turdan silinir ve bu kenarların silinmesi ile elde edilen alt turlar $\{ y_1, y_2, y_3, y_4 \}$ kenarları ile bağlanır (Şekil 4.13). Burada rasgele seçilen kenarlar turun eş uzunluklu alt turlarının başlangıç-bitiş aralığından seçilmelidir (Martin ve ark. 1991). Literatürde DB mutasyon işleminde kenarların belirlenmesinde sadece rasgele seçim metodu yoktur. *Close*, *Geometric* ve *Random-Walk* seçim metotları da mevcuttur. Fakat en yaygın

olarak kullanılan Rasgele (*Random*) seçim metodudur. Applegate ve arkadaşları (2003) bu 4 metodun analizini yapmışlardır.



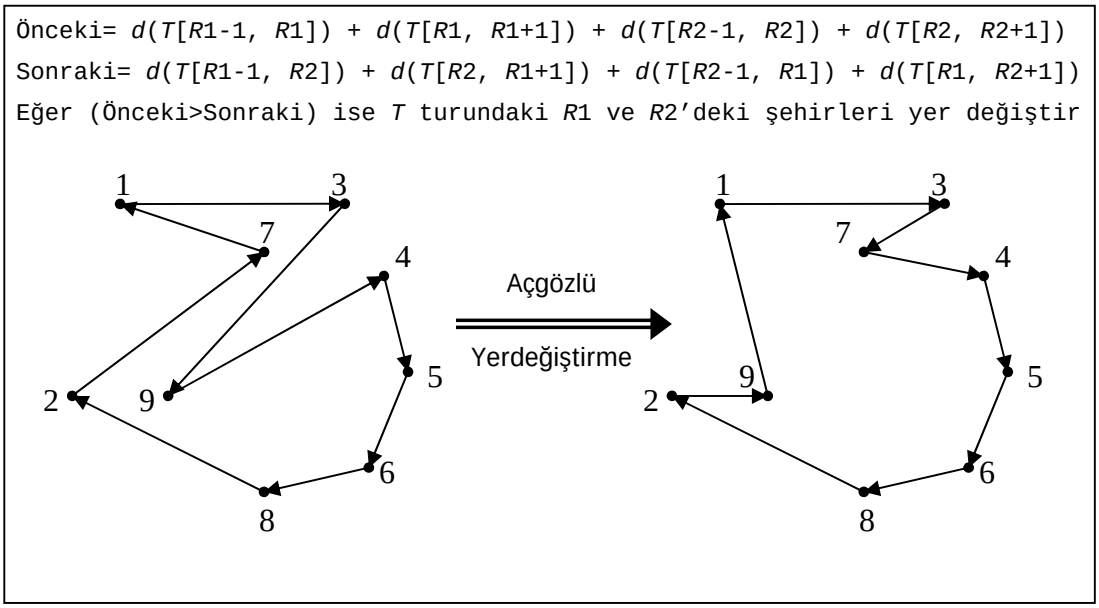
Şekil 4.13. Ardışık Olmayan Çift Köprü Mutasyonu

Bireye tek bir DB metodunun uygulanması Basit Genetik Algoritma (BGA) için yerel optimumdan çıkmaya yetebilmekte iken yerel arama metotları kullanıldığında yetersiz kalabilmektedir. Bu yüzden bu işlem *k* kuvvetine bağlı olarak *k* kere uygulanabilir. Ayrıca Johnson (1990) Lin-Kernighan (LK) yerel arama metodu ile *Random*-DB mutasyonunu birleştirerek *Iterated Lin-Kernighan* (ILK) metodunu geliştirmiştir.

4.3.8. Açgözlü Değiş-Tokuş Mutasyonu

Açgözlü Değiş-Tokuş Mutasyonu (*Greedy Swap Mutation*), Değiş-Tokuş mutasyon operatörünün açgözlü davranışından ibarettir. Açgözlü Değiş-Tokuş Mutasyonunda (GSM) turdan seçilen iki kesme noktasındaki şehir (*R*1 ve *R*2) yer değiştirmeden önce değişimdeki kazanç dikkate alınır (Louis ve Tang 1999). Bir *T* turu için bu değişimin aşağıdaki şarta uyması gereklidir ($d(T[R1, R2])$ fonksiyonu, turda *R*1 ve *R*2 pozisyonlarındaki şehirlerin arasındaki uzaklığı hesaplamaktadır). Ayrıca Hu ve Gu (2007) GSM operatöründe ufak modifikasyonlar yapmışlardır.

GSM'ye eklenen bir *k* parametresi ile mutasyon işleminin kuvveti (tekrarlama sayısı) belirlenebilir. Şekil 4.14'de GSM'ye ait bir örnek bulunmaktadır.



Şekil 4.14. Açgözlü Değiş-Tokuş Mutasyon Operatörü

4.3.9. Inver-Over Operatörü

Evrimsel algoritmalar için Guo ve Michalewicz (1998) tarafından tek operatör olarak geliştirilmiştir. İsmini, bünyesinde çaprazlama ve mutasyon işlemlerini yapmasından dolayı Inver-Over (*Inversion-Crossover*) olarak almıştır. Operatör, bünyesindeki çaprazlama ve mutasyon operatörlerinden hangisine daha yatkın olacağını *p* ihtimaline göre ayarlar. *p* parametresi 0.5 değerinin üzerine çıktıkça popülasyondaki başka bir bireyden bilgi aktarımı olmadığından mutasyon ihtimali, altına indikçe de (popülasyondaki diğer bireylere bağlılığından) çaprazlama ihtimali artacaktır. Inver-Over operatörünün basit algoritması Şekil 4.15'deki gibidir.

```
Rasgele bir POP popülasyonu oluşturulur;
(Sonlanma kriteri sağlanıncaya kadar);
{    Her S_İ bireyi için (S_İ ∈ POP)
     {    S' = S_İ;
          S' bireyinden rasgele bir {c} şehri seçilir;
          Döngü
          {
            Eğer (RND( ) ≤ p) ise S' de kalan şehirlerden c' şehri
            seçilir;
                    Değilse
                    {
                    POP popülasyonundan rasgele bir birey seçilir;
                    Seçilen bireyde c şehrinden "sonraki" gelen
                        şehir c' olarak seçilir;
                    }
          Eğer (S' de c şehrinden "Önceki" veya "Sonraki" şehir c')
            ise Döngüden çık;
          S' deki c' şehri c şehrinden sonraki şehir olacak şekilde
            parçayı ters çevir;
          c = c';     }
     Eğer ( ∑Mesafe(S') ≤  ∑Mesafe(S_İ) ) ise S_İ =S';
}        }
```

Şekil 4.15. Inver-Over Operatörünün Algoritması

Algoritmada yer alan *RND( )* fonksiyonu [0,1] arasından üretilen bir kayan noktalı sayıdır. Guo ve Michalewicz yaptıkları evrimsel algoritma çalışmasında popülasyon sayısını 100 ve parametresel olarak belirlenen *p* olasılığını yaklaşık 0.02 olarak seçmiş ve oldukça etkili sonuçlar almışlardır.

4.3.10. Tersleme-En Yakın Komşu Mutasyonu

Tersleme-Rasgele En Yakın Komşu (*Inver-RNN*), Guo ve Michalewicz (1998) tarafından önerilen Inver-Over operatörü üzerinde bir modifikasyon içermektedir. Bu tez çalışmasında geliştirilen ve Inver-RNN adını verdiğim operatör temelde Inver-Over operatöründen Şekil 4.16'daki sahte kod bloğu ile ayrılmaktadır.

```
Eğer (RND( ) ≤ p) ise S' de kalan şehirlerden c' şehri seçilir
      Değilse
      c şehrinin k adet en yakın komşusu içerisinden rasgele c'
         şehri seç;
```

Şekil 4.16. Inver-RNN'nin Inver-Over'dan Farklı Olan Kod Bloğu

Inver-Over operatörü *c*' şehrini seçerken popülasyondan yer alan diğer bireylerden etkilendiği için aynı zamanda çaprazlama (*crossOVER*) temellidir. Yukarıda verilen kod bloğunda Inver-Over'ın popülasyonda yer alan diğer bireylerden bilgi aktarımı özelliği kaldırılmıştır. Çaprazlama etkisinin kaldırılması ile *p* parametresi, açgözlü bir metot olan Inver-RNN'nin yerel optimuma takılmasını engellemek amacıyla kullanılan bir parametre haline dönüşmektedir. Ayrıca yeni eklenen *k* parametresi, *c* şehrinin sıralanmış en yakın komşularının maksimum liste uzunluğunu göstermektedir. Sonuç olarak, popülasyondan gen aktarımı (çaprazlama) etkisi kaldırıldığından ve rasgele bir bozunum yapısı içermesinden dolayı Inver-RNN tam bir mutasyon operatörü haline gelmektedir. Diğer mutasyon operatörlerinden farklı olan Inver-RNN, kullanıcı tarafından belirlenen *p* ve *k* parametrelerine bağlı olarak çalışmaktadır.

4.3.11. Açgözlü Alt Tur Mutasyonu

Açgözlü Alt Tur Mutasyonu (*Greedy Sub Tour Mutation*) operatörü de bu tez çalışmasında geliştirilen bir diğer mutasyon operatörüdür. Açgözlü Alt Tur Mutasyonu (GST), Açgözlü Yerdeğiştirme (GS) operatörü gibi mutasyon işlemini aç gözlü bir şekilde yapmaktadır. Açgözlü tüm metotlar diğer operatörlere göre daha fazla zaman harcamasına karşın rasgelelikten uzak oldukları için optimum değerlere daha yakın sonuçlar vermektedir. GST, GS mutasyon operatörü gibi tek bir şehir bazında değil turda yer alan *M* uzunluktaki bir alt turun, tura yeniden yerleşimini esas almaktadır. *M* uzunluklu bir alt turun yer değiştirmesi yapıldığından, alt tur içerisinde kalan sıralamanın bozulmadan daha kısa bir turu bulma ihtimalini arttırmaktadır.

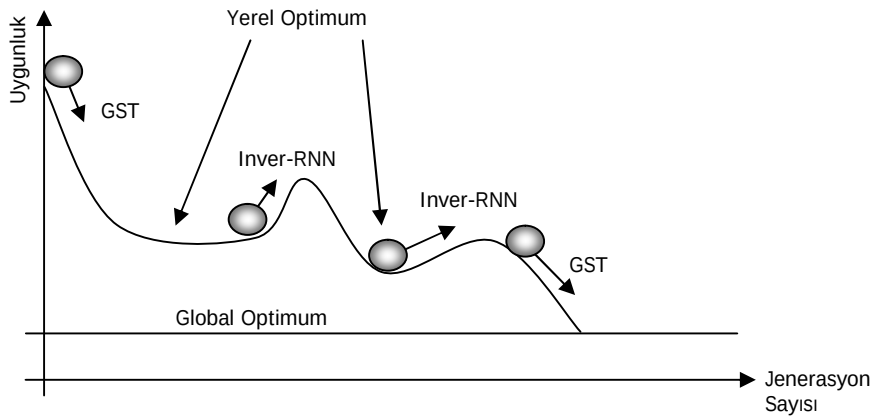
GST operatörünün çalışma mantığı çok karmaşık değildir. Bireyden rasgele seçilen *R*1 ve *R*2 noktaları arasında kalan alt tur, turdan çıkartılır ve [*R*1-…-*R*2] aralıklı alt tur bireyde yerleştirilebilecek noktalarda analiz edilir. Bu analiz yapılırken iki yerleşim koşulu da kontrol edilir. [*R*1-*R*2] alt turunun yerleştirilme maliyeti veya [*R*2-*R*1] alt turunun yerleştirilme maliyetinin en düşük olması durumuna göre alt tur parçalanmış tura bağlanır. Alt turun çift yönlü bir analizi yapıldığından [*R*2-*R*1] birleşmesi kısmi olarak SIM (*Simple Inversion*) ve INV (*Inversion*) mutasyonlarının açgözlü bir şekilde birleşimine benzemektedir.

GST operatörü bir *M* parametresi kontrolünde çalışmaktadır. Bu parametre seçilecek olan alt turun maksimum uzunluğunu belirlemek için kullanılır. Her mutasyon işleminde [2-*M*] arasındaki bir uzunlukta rasgele alt tur seçilir. Ayrıca parametrik belirlenen *I* olasılığı kadar GST operatörü yerel optimumdan çıkmak için Inver-RNN mutasyon metoduna başvurmaktadır. Inver-RNN operatörü için kullanılan *p* ve *k* parametrelerinin seçimi yine yerel optimuma takılmakta etkilidir. Dolayısıyla GST operatörü { *p*, *k*, *M*, *I* } parametrelerine bağımlı olarak çalışmaktadır. GST operatörünün algoritması Şekil 4.17'deki gibidir.

GST operatörünün tek dezavantajı açgözlü bir metot olmasından dolayı yerel optimuma takılma olasılığının yüksek oluşudur. İşte bu sıkıntıyı aşabilmek için de yine bu tez çalışmasında geliştirilen ve oldukça etkili olan Inver-RNN operatörüne *I* olasılığıyla bağlanarak yerel optimumdan çıkış Inver-RNN operatörünün *p* ve *k* parametrelerinin etkisinde sağlanmaktadır (Şekil 4.18).

```
Eğer ( RND( ) > I ) ise
{     Rasgele {M} aralıklı R1 ve R2 sayısı üret;
      Bireydeki [R1-R2] alt turunu kaldır;
      Mesafe = +∞;
      Bireyde kalan turun her { i, i+1 } aralığı için
      {     Eğer ( ( i → [R1-R2] → i+1) Mesafesi < Mesafe )
      {     Yon = TRUE; Yerleşim = i;
            Mesafe = ( i → [R1-R2] → i+1) Mesafesi;
      }
            Eğer ( ( i → [R2-R1] → i+1) Mesafesi < Mesafe )
      {     Yon = FALSE; Yerleşim = i;
            Mesafe = ( i → [R2-R1] → i+1) Mesafesi;
      }   }
      Eğer Yon=TRUE ise T'ye i. İndisten itibaren [R1-R2]'yi ekle
      Değilse T turuna i. İndisten itibaren [R2-R1]'i ekle;
}
Değilse Inver-RNN operatörüne dallan;
```

Şekil 4.17. Açgözlü Alt Tur Mutasyon Operatörü Algoritması



Şekil 4.18. Inver-RNN'nin GST Mutasyon Operatörü Üzerindeki Etkisi

4.4. Adaptif Stratejiler

Bu güne kadar yapılan birçok araştırmada GA operatörleri için uygun kontrol parametreleri bulunmaya çalışılmıştır. Basit GA'lar herhangi bir problem üzerinde araştırma yaparken bu parametreler araştırma süresince sabit kalmaktadır. Bununla beraber bu parametre değerlerinin problemden probleme değişiklik göstereceği kesindir. Ayrıca bir problemde çözüm kalitesinin zamana bağlı olarak değişmesi kontrol parametrelerinin de değişmesi gereksinimine yol açmaktadır. Kullanılacak parametrelerin zamana bağlı otomatik olarak değiştirilmesi hem zaman kazandıracak hem de performansın artmasını sağlayacaktır (Karaboğa 2004).

Genetik Algoritmalar ile verilen bir probleme çözüm araştırması yapılırken çaprazlama ve mutasyon oranlarının belirlenmesi optimum değere ulaşmayı ve çalışma süresini önemli ölçüde etkilemektedir. Ayrıca problem çözmedeki kullanılan popülasyon sayısı da çözüme ulaşmada oldukça önemli ve etkili parametrelerden birisidir (Emel ve Taşkın 2002). Bu bağlamda kontrol parametrelerinin çözüm kalitesine göre zamana bağlı olarak değiştirilmesi, çaprazlama ve mutasyon ihtimalleri için değerlendirilmelidir.

Çaprazlama ve mutasyon oranları GA'ların çözüm kalitesini oldukça etkileyen parametrelerdir. Geliştirilen uygulama yazılımında çaprazlama ve mutasyon oranlarının belirlenmesinde iki faklı adaptif strateji kullanılmıştır.

4.4.1. Adaptif Strateji–1

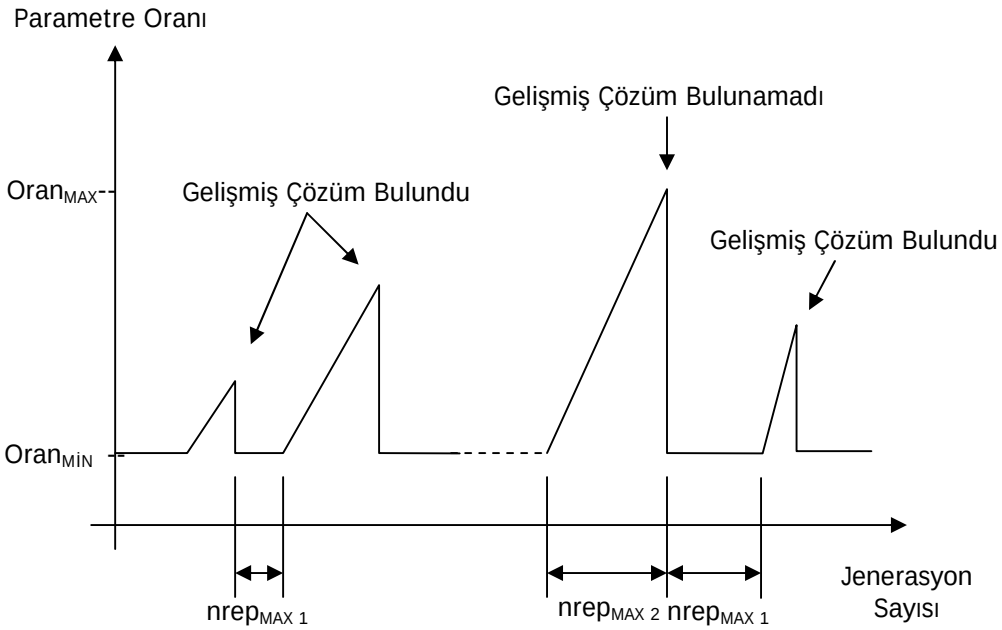
Bu stratejide maksimum ve minimum oranlar kullanıcı tarafından belirlenir. Ayrıca oranlar artmaya başlamadan önce en iyi bireyin çözüm performansının kaç jenerasyon boyunca sabit kalacağının belirleneceği bir eşik değeri ($nrep_{MAX1}$) ve maksimum orana ulaşmak için kaç jenerasyon harcanacağının belirlendiği diğer bir eşik değeri ($nrep_{MAX2}$) parametrik olarak belirlenir. Optimizasyon başlangıcında oranlar minimuma set edilir ve araştırma süresince en iyi çözüm performansına göre

eşik değerlerine bağlı kalınarak maksimuma doğru gidilir. En iyi bireyde gelişme görüldüğünde ise oran yeniden minimuma indirilerek işleme devam edilir.

Kontrol Parametreleri (KP) olan çaprazlama ve mutasyon oranlarının (*KP.Oranı*) adaptif olarak hesaplanması aşağıdaki gibi (4.1) gerçekleştirilir (Karaboğa 2004). Bu strateji hem mutasyon hem de çaprazlama metodunu kontrol parametrelerine uygulanabilmektedir. Strateji-1 için KP.Oranı'nın değişiminin zamana göre oluşturabileceği bir grafik Şekil 4.19'daki gibidir.

$$KP.Oranı = \frac{nrep - nrep_{MAX1}}{nrep_{MAX2}} \left(KP.Oran_{MAX} - KP.Oran_{MIN}\right) + KP.Oran_{MIN} \qquad (4.1)$$

$$(nrep - nrep_{MAX1}) \leq nrep_{MAX2}$$



Şekil 4.19. Adaptif Kontrol Parametresi Belirleme: Strateji–1

4.4.2. Adaptif Strateji–2

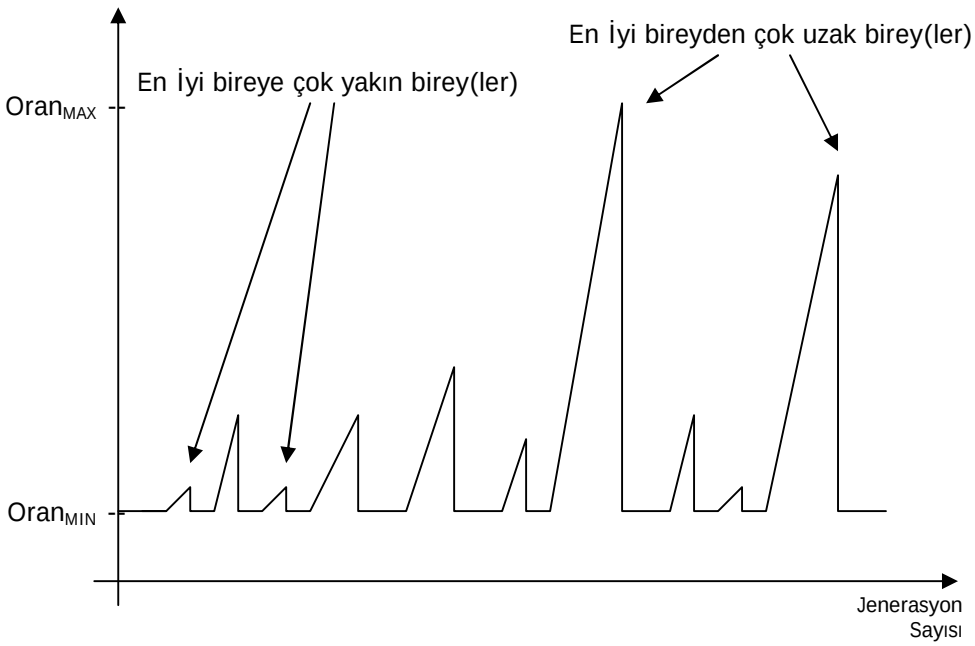
Bu strateji, bu tez çalışmasında önerilmiştir. Temel prensibi kötü çözümlerin iyi çözümlerden daha çok geliştirilmeye ihtiyacının olmasıdır. Adaptif kontrol parametresi oranı belirleme işleminde sadece en iyi bireyin çözüm kalitesi değil operatöre tabii tutulacak bireylerin çözüm kaliteleri de değerlendirilmektedir. Bu bağlamda operatöre tabi tutulacak bireylerin en iyi bireye olan uzaklıkları, uygulanacak operatör oranının şiddetinin belirlenmesinde rol oynamalıdır. Kontrol parametreleri olan çaprazlama ve mutasyon oranlarının adaptif olarak hesaplanması aşağıdaki gibi (4.2) gerçekleştirilir. Strateji-2 için KP.Oranı'nın değişiminin zamana göre oluşturabileceği bir grafik Şekil 4.20'deki gibidir

$$KP.Oranı = \frac{\left(UYG_{ORT} - UYG_{İYİ}\right)}{UYG_{ORT}}\left(KP.Oran_{MAX} - KP.Oran_{MIN}\right) + KP.Oran_{MIN} \qquad (4.2)$$

Çaprazlama İçin $UYG_{ORT} = (UYG_{ANNE} + UYG_{BABA}) / 2$

Mutasyon İçin $UYG_{ORT} = (UYG_{YAVRU} + \varepsilon)$

$\varepsilon = 0.00000000000\ldots0001$



Şekil 4.20. Adaptif Kontrol Parametresi Belirleme: Strateji–2

4.5. Yerel Arama Metotları

GSP problemleri için kullanılan birçok yerel arama metotları ve bunların varyasyonları mevcuttur. Literatürde yerel arama metotları, Tur Geliştirme Metotları olarak da bilinmektedir. Günümüzde GSP problemleri için en ideal neticelerin elde edilmesinde yerel arama operatörlerinin kullanılmasının gerekliliği (Johnson ve McGeoch 1997, Marinakis ve ark. 2005) açıkça görülmektedir. Ayrıca tur üzerinde yapılacak olan yerel aramaların tekrarlamalı bir şekilde yapılması gerektiği (Johnson ve McGeoch 1997, 2002) deneysel sonuçlardan anlaşılmaktadır. Yerel arama metotlarından en genel olarak bilinen ve kullanılanları 2-Opt (Croes 1958), 2.5-Opt (Bentley, 1992), 3-Opt (Lin 1965), Or-Opt (Or 1976, Babin ve ark. 2007), Lin-Kernighan (Lin ve Kernighan, 1973) metotlarıdır. Opt olarak yapılan kısaltma "Optimizasyon" anlamına gelmektedir. Bu metotlardan 2, 2.5, 3 ve Or-Opt metotları *k*-Opt metotları olarak da adlandırılırlar. Bunların haricinde 4-Opt ve 5-Opt metotları da mevcuttur fakat araştırma süresi $\Theta(n^4)$ *ve* $\Theta(n^5)$ olduğu için çeşitli iyileştirmeler yapılmadan kullanılması oldukça uzun zaman almaktadır.

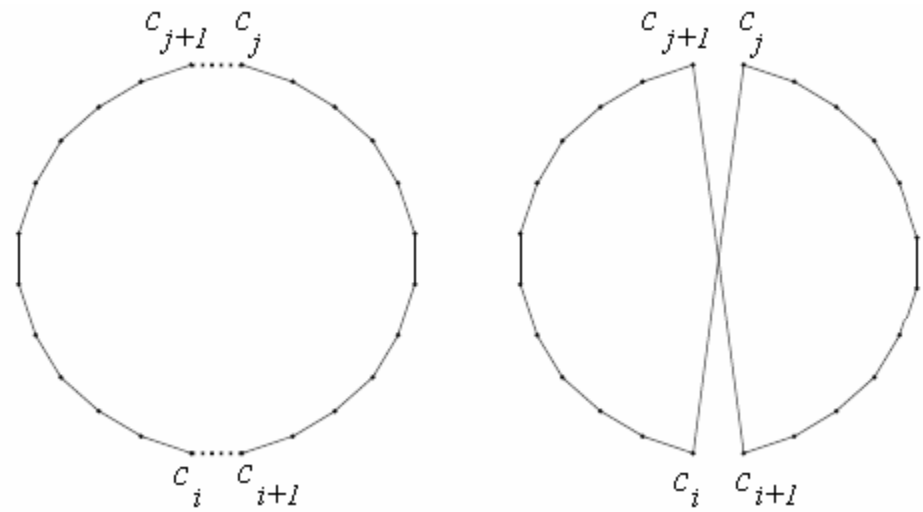
4.5.1. 2-Opt Metodu

İlk defa 1958 yılında öne sürülen 2-Opt metodu verilen turdan iki kenar seçilerek silinir, böylece mevcut tur iki parçaya ayrılır. Silinen kenarlar turda en az uzamaya sebep olacak şehirler ile Şekil 4.21'de verilen algoritmadaki gibi birleştirilir (Croes 1958). Standart bir 2-Opt metodu $\Theta(n^2)$ çalışma zamanına ihtiyaç duyar. GA ile birlikte kullanıldığı (Ghoseiri ve Sarhadi 2007, White ve Yen 2004) birçok çalışma mevcuttur. Her bir şehir için CL_{MAX} uzunluklu bir komşuluk listesinin (*Candidate List*: CL) eklenmesi ile 2-Opt-J metodunun doğruluğu %6.23 civarına gelmektedir. Aynı zamanda operatörün çalışma süresi CL_{MAX} parametresine bağlı olarak $\Theta(n.\log n + n.CL_{MAX})$ olarak azalmaktadır. CL diğer yerel arama operatörleri için de kullanılabilmektedir. Johnson'a göre CL_{MAX} değeri yaklaşık 20 olarak alınması ideal olarak gözükmektedir. $CL_{MAX}=5$ olarak alındığında oldukça hızlı sonuçlar alınmakta fakat asıl amaç olan tur mesafesinde optimum değere ulaşmak zorlaşmaktadır. Ayrıca Bentley (1992) tarafından önerilen Bitlere Bakmama (*Don't*

Look Bit) modülü ve *kd-tree* şeklindeki problem sunumu ile de hem operatörün çalışma süresi azaltılmakta hem de Pcb1173 GSP problemi için %6.20 optimuma yaklaşıldığı (Marinakis ve ark. 2005) görülmektedir. Örnek bir 2-Opt hareketi Şekil 4.22'de gösterilmektedir.

```
Uzunluğu L olan S = { c1, c2, c3, ..., cn, c1} olan tur seçilir;
i = 1;
j = i + 2;
( i ≤  |n| - 2) olduğu sürece dön
{
      S* = { c1, c2, ..., ci, cj, cj-1, ...,ci+1, cj+1, cj+2, ..., c1 }
            turu kurulur;
      L1 = S* turunun uzunluğu hesaplanır;
      Eğer (L1 < L) ise
      {  S = S*;    i = 1;    j = i + 2;
      } değilse {
      j = j + 1;
      Eğer j > |n| ise {  i = i + 1;     j = i + 2;  }
} // döngü sonu
```

Şekil 4.21. Basit 2-Opt Algoritması



Şekil 4.22. Bir 2-Opt Hareketi

4.5.2. 3-Opt Metodu

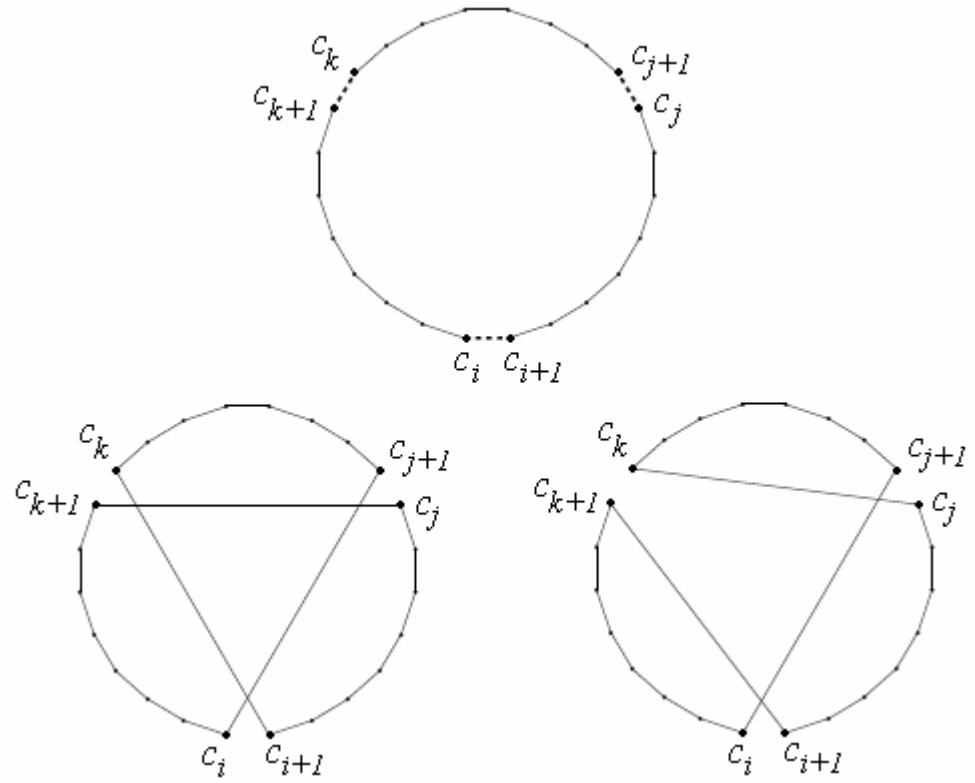
2-Opt metoduna oldukça benzeyen 3-Opt metodu turda yer alan 3 kenar silinir ve tur 3 parçaya bölünür. Parçalanmış olan turdaki kenarlar farklı kombinasyonlarda birleştirilerek en kısa mesafeyi veren birleştirme işleminden turdaki tüm kenarlar deneninceye kadar optimizasyona devam edilir. 2-Opt metodundan optimuma daha yakın sonuçlar vermesine rağmen $\Theta(n^3)$ çalışma süresi standart bir 3-Opt algoritmasının dezavantajıdır.

Çeşitli hızlandırma teknikleriyle birleştirilen 3-Opt metodunun varyasyonları olan 3-Opt-J ve 3-Opt-B metotlarının Pcb1173 problemine uygulanmasından optimum değere sırasıyla %3.09 ve %4.01 yaklaşım gözlenmektedir (Marinakis ve ark. 2005). Genetik algoritmalarda 3-Opt yerel arama metodunun kullanarak yapılmış birçok çalışma mevcuttur (Liu ve Kang 2003). Ayrıca Stützle ve Hoos tarafından (1999) yapılan analizlerde tekrarlamalı bir 3-Opt (I3-Opt) işleminin I2-Opt ve ILK arasında zaman harcadığını göstermektedir. Şekil 4.23'de standart bir 3-Opt arama algoritması verilmektedir.

Sezgisel algoritmalarda olduğu gibi tüm *k*-Opt algoritmalarında da yerel optimuma takılma ihtimali yüksektir. Bu yüzden geliştirilen turda bir bozunum (*perturbation*) yapılarak araştırmaya devam edilmelidir. *k*-Opt metotlarının bir bozunum sonrası araştırmaya tekrar başlaması *Ik*-Opt *(Iterated k-opt)* olarak bilinmektedir. Şekil 4.24'de ise mümkün 3-Opt hareketlerine bir örnek verilmiştir.

```
S ={ c1, c2, ..., cn } turu seçilir
For i = 1 to n do
{
    For k = 1 to n-3 do
    {
      For j = k+1 to n-1 do
      {
      Eğer d(ck, cj+1)+ d(c1, cj) ≤  d(c1, cj+1) + d(ck, cj) ise
         {
            d = d(ck, cj+1) + d(c1, cj);
            α = TRUE;
         } değilse
         {
            d = d(c1, cj+1) + d(ck, cj);
            α = FALSE;
         }
      Eğer (d + d(ck+1, cn) < d(c1, cn) + d(ck, ck+1) + d(cj, cj+1)) ise
         {
            Eğer α = TRUE ise
            {c1, c2, ..., tn} = {cj+2, ..., cn, ck+1, ..., cj, c1, ...,
                                 ck, cj+1}
            Değilse // α = FALSE
            {c1, c2, ..., cn} = {cj+2, ..., cn, ck+1, ..., cj, ck, ...,
                                 c1, cj+1 };
         }
      } // For j
    } // For k
      {c1, c2, ..., cn} = {cn,c1, c2, ..., tn-1 };
} // For i
```

Şekil 4.23. Basit 3-Opt Algoritması



Şekil 4.24. İki Mümkün 3-Opt Hareketi

4.5.3. 2.5-Opt Metodu

2-Opt ile 3-Opt arasında çalışan bir operatör olarak geliştirilmiştir (Bentley 1992). 2-Opt'a göre çalışma süresi bakımından %30-40 civarında daha iyi sonuç vermesinin yanında toplam tur maliyetinde rasgele oluşturulmuş (*Random Euclidean*) GSP'ler için %0.5 TSPLIB problemleri için yaklaşık %1 daha avantajlıdır (Johnson ve McGeoch 1997). Bentley (1992) tarafından geliştirilen 2.5-Opt operatörü Pcb1173 problemi için optimum mesafede ortalama %4.57'lik bir hata payına sahiptir. 2.5-Opt metodu biraz daha fazla efor harcayarak bulunabilecek 3-Opt hareketinin yapılmasına benzer. 2-Opt hareketi gelişim gösteremediği zaman turdaki bir şehir silinir ve turda başka bir yere eklenir.

4.5.4. Lin-Kernighan Metodu

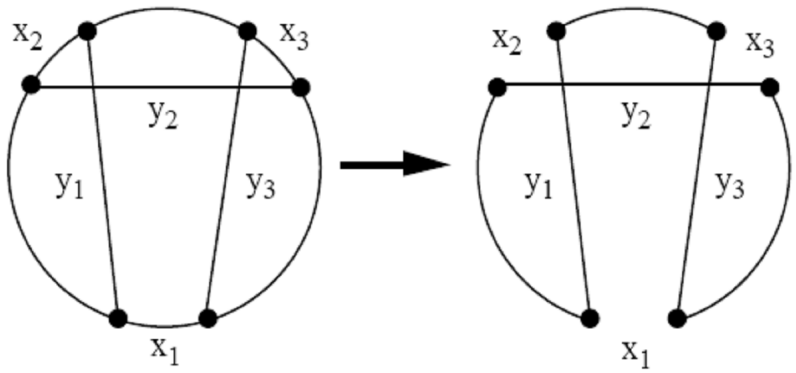
k-Opt değişim işlemleri şehir sayısı arttıkça üstel bir şekilde artmaktadır. Basit bir uygulamada (*n* şehir) *k*-değişimlerinin test edilmesi $\Theta(n^k)$ zaman karmaşasına sahiptir. *k*-değişimlerinin önemli bir üst sınırı yoktur. Genellikle birçok çalışmada *k*=2 ve *k*=3 olarak seçilmektedir. *k*=2 seçilmesi 2-Opt, *k=3* seçilmesi de 3-Opt metotlarına karşılık gelmektedir. *k* bağlantıları sayısı arttıkça turun optimal tur olma ihtimali artmaktadır.

k değerinin önceden belirlenmesi gerekmesi bir engeldir. *k*'nın çalışma süresi ve çözüm kalitesi arasında en iyi performansı vermesi için ne seçilmesi gerektiğini bilmek zordur. Bu engeli ortadan kaldırmak için Lin ve Kernighan 1973'de güçlü bir *Değişken*-Opt algoritması öne sürmüşlerdir. Lin-Kernighan (LK) algoritması, her iterasyonda *k*'nın değerini belirleyerek çalışma esnasında *k*'nın değerini değiştirmektedir. Dolayısıyla LK algoritması çalışma süresince değişken bir *k*-Opt hareketidir. Her bir iterasyon adımında algoritma *k*'nın yüksek değerleri için *k* kenarlarının değişiminin daha kısa bir tur oluşturup oluşturmadığını inceler. *k* kenarlarının değişmesini düşünürsek, bir dizi test *k*+1 kenar değişiminin göz önünde bulundurulup bulundurulmamasına karar vermek için yapılır. Bu da değişim duruncaya kadar devam eder.

Algoritma her adımda potansiyel değişimlerin büyüyen kümesini (*k*=2 ile başlayarak) düşünür. Bu değişimler işlemin herhangi bir safhasında uygun bir tur oluşturacak şekilde seçilir. Eğer algoritma daha kısa bir tur bulursa gerçek tur yeni turla değiştirilir. Algoritma bir turu diğer turla değiştiren değişimlere göre belirlenir. Uygun tur verildiği zaman, hiçbir değişiklik değişim sağlamayıncaya kadar algoritma sürekli şimdiki turun uzunluğunu azaltan değişimler yapar. Bu işlem rasgele bir şekilde oluşturulan ilk turlarda birçok kez tekrarlanabilir. Algoritma aşağıda daha ayrıntılı bir şekilde anlatılmıştır.

T şimdiki tur olsun. Algoritma her iterasyonda iki kenar kümesi bulmaya çalışır ($X = \{ x_1, \ldots, x_k \}$ ve $Y = \{ y_1, \ldots, y_k \}$). Eğer *X* kenarları *T* turundan silinirse

ve yerine *Y* kenarları konulursa daha iyi bir tur ortaya çıkar. Bu kenar değişimine *k*-Opt hareketi denilir. Şekil 4.25'de basit bir 3-Opt hareketi gösterilmektedir.

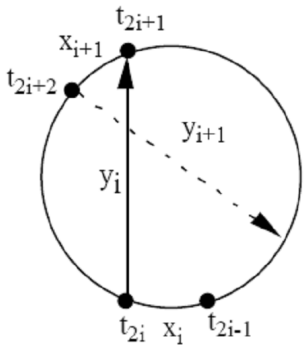


Şekil 4.25. LK için 3-Opt Hareketi

X ve *Y* kümeleri element element oluşturulur. Başlangıçta *X* ve *Y* kümeleri boştur. Algoritmanın *i* adımında bir çift kenar x_i ve y_i, *X* ve *Y* kümelerine sırasıyla eklenir. Etkili bir algoritmaya ulaşmak için aşağıdaki kritere uyan tek kenar *X* ve *Y* kümelerine girebilir.

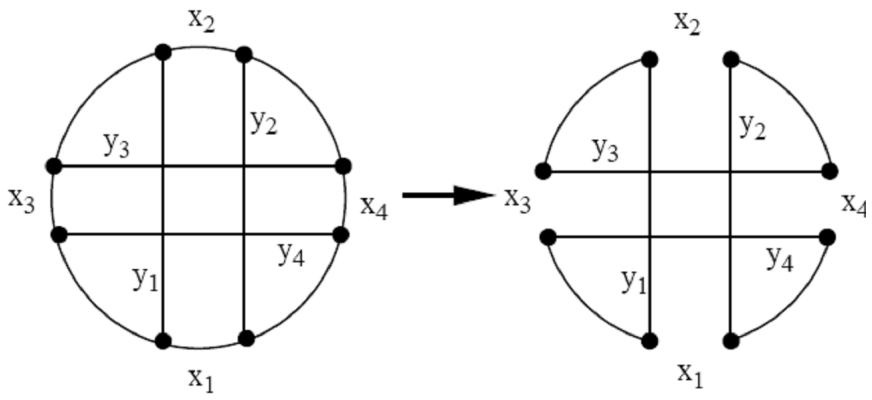
(1) Ardışık Değişim Kriteri

x_i ve y_i ortak bitiş noktaları olmalıdır ayrıca y_i ve x_{i+1}'in de ortak noktaları olmalıdır. Şekil 4.26'daki gibi, eğer t_1 x_1'in iki uç noktasından birisi ise, genellikle $x_i=(t_{2i-1}, t_{2i})$, $y_i=(t_{2i}, t_{2i+1})$ ve $x_{i+1}=(t_{2i+1}, t_{2i+2})$ elde edilir ($i \geq 1$ için).



Şekil 4.26. LK için x_i, y_i, x_{i+1} ve y_{i+1}'in Seçim Şartları

Görüldüğü gibi (x_1, y_1, x_2, y_2, x_3, y_3, …, x_k, y_k) dizisi komşu kenarlar zinciri oluşturur. Gerekli (fakat yeterli olmayan) durum, X kenarlarının Y kenarları ile yer değiştirmesi sonucunda turda kapalı zincir oluşturur ($y_k = (t_{2k}, t_1)$). Genelde, bir turun gelişimi etkilenen kenarların uygun bir şekilde numaralandırılması ile ardışık değişim olarak elde edilir. Ama her zaman bu şekilde olmaz. Şekil 4.27'de ardışık değişimin mümkün olmadığı durum gösterilmektedir.



Şekil 4.27. LK için Mümkün Olmayan Ardışık Değişim (k=4)

(2) Mümkünlük Kriteri

$x_i = \{t_{2i-1}, t_{2i}\}$ seçilmesi gerekmektedir ki, t_{2i} t_1'e eklenirse sonuç bir tur olsun. Bu uygunluk kriteri $i \geq 3$ için kullanılır ve kapalı bir turu garanti eder.

(3) Pozitif Kazanç Kriteri

y_i'nin her zaman seçilmesi gerekir ki, önerilen değişim kümelerindeki kazanç (G_i) pozitif olsun. x_i ve y_i'nin değiştirilmesinin kazancının $g_i = d(x_i) - d(y_i)$ olduğunu düşünürsek, $G_i = g_1 + g_2 + \ldots + g_i$ 'dir. Durdurma kriteri algoritmanın verimliliğinde önemli bir etkendir. Kısıtlandırıcı kriter, tüm *Gi* kazançlarının pozitif olmasıdır.

(4) Komşu Olmama Kriteri

Sonuçta, X ve Y kümelerinin kesişmemeleri gerekir. Bu kodlamayı basitleştirir, çalışma süresini azaltır ve etkili bir sonlandırma kriteri verir.

Algoritmanın temel adımları Şekil 4.28'deki gibidir.

Adım–1. Rasgele bir T turu üretilir,

Adım–2. t_1 şehri seçilir ve i=1 yapılır

Adım–3. $x_1 = (t_1, t_2)$ kenarı turdan seçilir ($x_1 = (t_1, t_2) \in T$)

Adım–4. T turunda olmayan $y_1 = (t_2, t_3)$ kenarı pozitif kazancın büyüklüğüne göre seçilir
($g_1 = d(x_1) - d(y_1) > 0$ (veya en büyük pozitif olan))
($y_1 = (t_2, t_3) \notin T$).
Eğer bu mümkün değil ise Adım 12'ye gidilir.

Adım–5. $i = i + 1$

Adım–6. $x_i = (t_{2i-1}, t_{2i})$ turdan seçilir ($x_i = (t_{2i-1}, t_{2i}) \in T$), şöyle ki:
a) Eğer t_{2i} t_1'e bağlanırsa, sonuç turu T^1 ve
b) x_i y_j' deki kenarlardan biri değildir ($x_i \neq y_j$, $j<i$)
Eğer T^1 turu T turundan daha iyi ise $T=T^1$ ve Adım 2'ye git.

Adım–7. T turunda olmayan y_i kenarı seçilir
($y_i = (t_{2i}, t_{2i+1}) \notin T$), şöyle ki:
a) $G_i = d(x_i) - d(y_i) > 0$,
b) y_i x_j'deki kenarlardan biri değildir
($y_i \neq x_j$, $j \leq i$) , ve
c) x_{i+1} mevcutsa.
Eğer böyle bir y_i var ise, Adım 5'e git.

Adım–8. Eğer y_2 için denenmemiş alternatif var ise,
i=2 yap ve Adım 7'ye git

Adım–9. Eğer x_2 için denenmemiş alternatif var ise,
i=2 yap ve Adım 6'ya git

Adım–10. Eğer y_1 için denenmemiş alternatif var ise,
i=1 yap ve Adım 4'e git

Adım–11. Eğer x_1 için denenmemiş alternatif var ise,
i=1 yap ve Adım 3'e git

Adım–12. Eğer t_1 için denenmemiş alternatif var ise,
Adım 2'ye git

Adım–13. Dur

Şekil 4.28. Lin-Kernighan Algoritması

4.5.5. Yerel Aramayı Hızlandırma Teknikleri

Yerel arama metotları optimum sonuçlara yaklaşmada diğer sezgisel metotlara göre daha iyidir, fakat çevrim sürelerinin fazla oluşu dezavantajlarıdır. Şehir sayısı n olan bir problem için 2-Opt metodu $\Theta(n^2)$, 3-Opt metodu $\Theta(n^3)$, LK metodu için $\Theta(n^k)$ zaman karmaşası mevcuttur. Şehir sayısını artması araştırma süresini üstel olarak arttırmaktadır. Yerel arama metotlarını çevrim süreleri bakımından hızlandırmak için çeşitli yöntemler kullanılmaktadır. Bunlardan en yaygın olarak bilinenleri:

- Sabit Yarıçaplı En Yakın Komşu Arama
- Bitlere Bakmama
- Aday Listeleri

Bu metotların her birisi arama süresini oldukça önemli ölçüde azaltmaktadır. Aşağıdaki tabloda hızlandırma tekniklerinin çeşitli problemler üzerindeki etkisi Stützle (2008) tarafından gösterilmektedir.

Tablo 4.4. Yerel Aramayı Hızlandırma Tekniklerinin Etkisi

Problem	2-Opt (STANDART)		2-Opt (FRNNS + CL)		2-Opt (FRNNS + CL + DLB)		3-Opt (FRNNS + CL + DLB)	
	ε_{ORT}	t_{ORT}	ε_{ORT}	t_{ORT}	ε_{ORT}	t_{ORT}	ε_{ORT}	t_{ORT}
KroA100	8.9	1.6	6.4	0.5	6.6	0.4	2.4	4.3
D198	5.7	6.4	4.2	1.2	4.3	0.8	1.4	30.1
Lin318	10.6	22.1	7.5	2.1	7.9	1.5	3.4	65.5
Pcb442	12.7	55.7	7.1	2.9	7.6	2.2	3.8	63.4
Rat783	13.0	239.7	7.5	7.5	8.0	5.8	4.2	213.8
Pr1002	12.8	419.5	8.4	13.2	9.2	9.7	4.6	357.6
Pcb1173	14.5	603.1	8.5	16.7	9.3	12.4	5.2	372.3
D1291	16.8	770.3	10.1	16.9	11.1	12.4	5.5	377.6
Fl1577	13.6	1251.1	7.9	25.8	9.0	19.2	4.0	506.8
Pr2392	15.0	2962.8	8.8	65.5	10.1	49.1	5.3	878.1

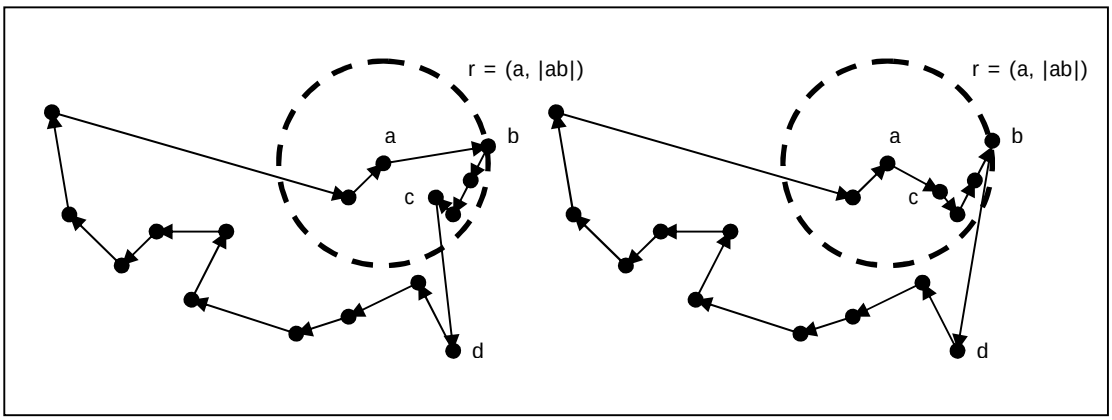
(FRNNS: Sabit Yarıçaplı En Yakın Komşu Arama, CL: Aday Listeleri, DLB: Bitlere Bakmama)

Tablo 4.4'deki çözümlenen 10 GSP probleminin zamansal verilerine dikkat edilecek olursak önemli ölçüde azalma gözükmektedir. Örneğin Pr1002 problemi için standart bir 2-Opt metodunun çalışma süresi FRNNS + CL + DLB hızlandırma teknikleri sayesinde yaklaşık 43 kat hızlandırılabilir. Tüm zamansal verilerin ortalamaları dikkate alındığında hızlandırma teknikleri yaklaşık 4-60 kat arasında performans artışı sağlamaktadır (problem büyüklüğüne göre).

Hızlandırma teknikleri yerel arama metotları içerisine kodlama sırasında eklenerek GSP çözümünde çevrim süresi bakımından iyileştirme sağlarken, en iyi çözümü bulmayı garanti etmezler.

4.5.5.1. Sabit Yarıçaplı En Yakın Komşu Arama

Sabit Yarıçaplı En Yakın Komşu Arama (*Fixed Radius Nearest Neighbor Search*) ile sadece (yerel arama metotları içerisine eklenen kodlama) belirli bir yarıçap içerisinde kalan şehirlerin yer değiştirmesi incelenir. Örneğin 2-Opt metodu için verilen bir c_i şehri için turdaki her iki komşusu (c_j) dikkate alınarak araştırma yarıçapı belirlenir. c_i çevresindeki tüm c_k şehirleri $d(c_i, c_k) < d(c_i, c_j)$ koşulunu sağlayacak şekilde araştırılır ve ilk gelişen 2-Opt hareketi kabul edilir. Bu olay Şekil 4.29'daki gibidir. Böylece Sabit Yarıçaplı En Yakın Komşu Arama (FRNNS) ile uzay yarıçap ile sınırlandırılıp araştırma süresi azaltılır (Bentley 1992).



Şekil 4.29. Sabit Yarıçaplı En Yakın Komşu Arama

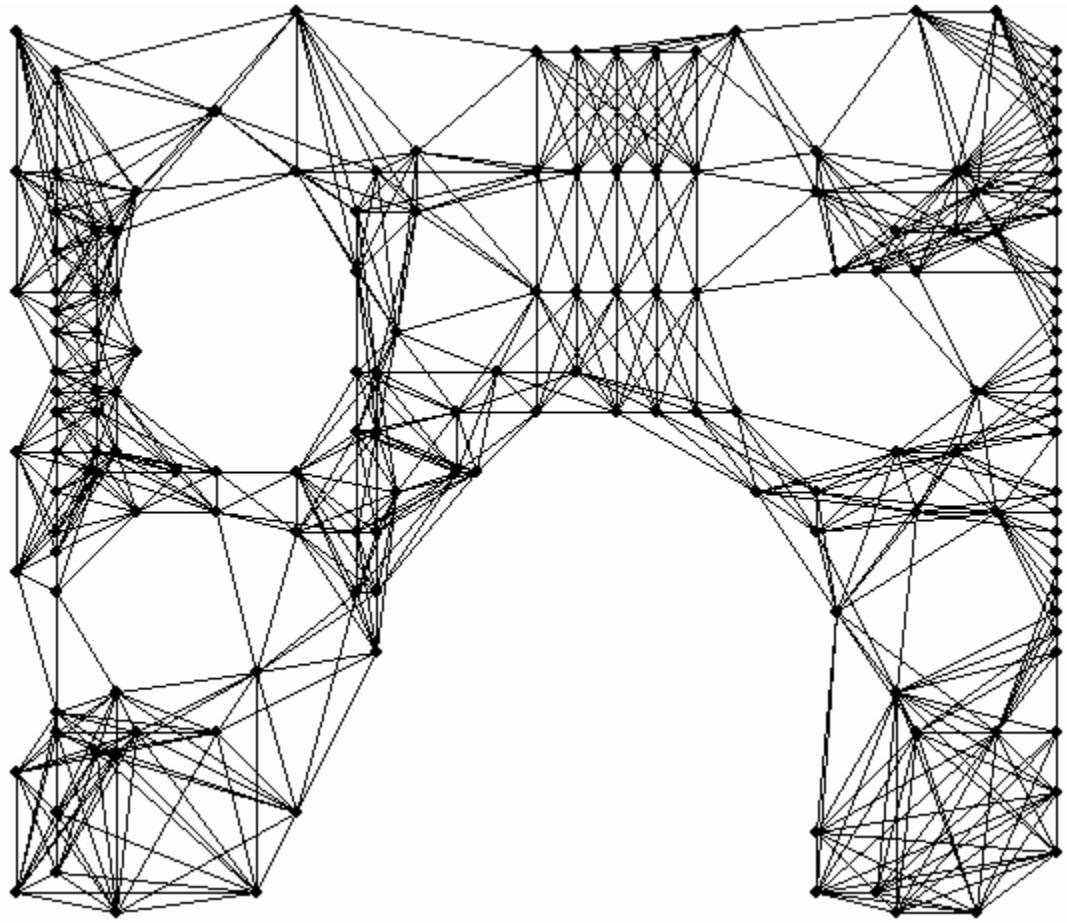
4.5.5.2. Bitlere Bakmama

Bitlere Bakmama (*Don't Look Bits*) fikri Bentley (1992) tarafından yerel aramalarda başarısız aramaların tekrarlanmasından kaçınmak için düşünülmüştür. Bu metodun ana fikri verilen c_i şehri için herhangi bir gelişme kaydedilmediyse ve c_i'in komşuları değişmediyse bir sonraki araştırmada c_i'in gelişim gösterme ihtimali düşüktür. Bu yüzden bir sonraki araştırmada c_i, gelişim hareketi için üye olarak düşünülmez. Bentley, bu fikri her şehir için bir kontrol bayrağı (boolean tipinde bir dizi) ile gerçekleştirmektedir. Yerel aramaya ilk başlandığında tüm bayraklar kapalı (*False*) olarak alınmakta ve yerel arama süresince her şehre ait olan bayraklar değişimin olmamasının gözlenmesi ile açılmakta veya kapanmaktadır. c_i=c olduğunda ve c şehri bir gelişim gösteremediğinde, c'ye ait olan bayrak açılır (*True*). Bu şekilde c_i şehri yerel arama süresinde tekrar incelenmeye tabii tutulacağında bayrağının durumuna göre işleme devam edilir.

4.5.5.3. Aday Listeleri

Yerel arama prosedürlerinde şehir sayısının artmasının çözüm süresini kullanılan tekniğe göre yüksek ölçüde arttırmaktadır. Bu süreç içerisinde turun gelişmesi için sonuç vermeyecek birçok deneme yapılmaktadır. Bir şehri diğer şehir ile bağlayacak kısa yolların toplam uzunluğu azaltması uzun birleşmelere göre daha kuvvetli bir ihtimaldir. Bu yüzden her şehir için en yakın komşuların (aday) tutulduğu listeler (aday listeleri) ile yerel arama süresince herhangi bir c şehri için tüm şehirler (n) yerine dizideki yakın olan komşular taranacaktır. Aday listelerinin (CL) büyüklüğü (CL_{MAX}) kullanıcı tarafından belirlenir. Bu şekilde çalışma süresi önemli ölçüde azalmaktadır. Gereken en yakın komşuluk dizisinin hesaplanması için $\Theta(n^2.\log CL_{MAX})$ süresine ihtiyaç vardır. Sık olarak kullanılan En Yakın Komşular (*Nearest Neighbors*: CL_{NN}) ve Çeyrek Daire Komşuları (*Quadrant Neighbors*: CL_{QN}) aday listeleri mevcuttur. CL_{NN} aday listelerinde herhangi bir c şehri için tüm şehirler en yakından en uzağa olmak üzere sıralanır ve kullanıcı tarafından belirlenen CL_{MAX} sayısınca c şehrinin aday listesini oluşturur. CL_{QN} aday listelerinde ise c şehri orijin olarak düşünülür (x=0, y=0) ve her 4 geometrik bölgede bulunan en yakın

komşulardan birer tanesi aday listesine eklenir. Şekil 4.30'da, U159 GSP probleminde yer alan 159 şehrin her birisine ait en yakın CL_{MAX}=10 olan aday listesi gösterilmektedir.



Şekil 4.30. CL_{NN} Aday Listesi (U159 Problemi, $CL_{NN\text{-}MAX}$ =10)

Nearest Neighbors ve *Quadrant Neighbors* komşulukları dışında *Delunay Triangulations Lists* ve α-*Value Based Lists* metotları da mevcuttur. Şekil 4.31'de 2-Opt yerel arama prosedürüne "*Bitlere Bakmama*", "*Aday Listesi*" ve "*Sabit Yarıçaplı En Yakın Komşu Araması*" metotlarının eklenmesi ile oluşan hızlandırılmış 2-Opt versiyonunun kod bloğu verilmiştir.

```
Tüm şehirler için DLB'yi sıfırla (FALSE)
Gelişme sürdüğü sürece (Gelişim=TRUE)
FOR i := 1 TO n DO
{
   Gelişim := FALSE;          C1 := T[i]
   Eğer DLB[C1]=TRUE ise GOTO 200;
   SC1       := Turda i şehrinden sonraki şehir;
   Yarıçap  := d(C1,SC1);
   FOR h := 1 TO CLMAX DO
   {  C2 := Aday_Listesi[C1, h];
      Eğer (Yarıçap > d(C1, C2)) ise
      { SC2 := Turda C2 şehrinden sonra gelen şehir;
         Kazanç := - Yarıçap + d(C1, C2) + d(SC1, SC2) - d(C2, SC2);
         Eğer (Kazanç < 0) ise { Gelişim := TRUE; GOTO 100; }
      } Değilse h döngüsünden çık;
   }

   ÖC1 := Turdaki C1 şehrinden bir önceki şehir;
   Yarıçap := d(ÖC1, C1);
   FOR h := 1 TO CLMAX DO
   {  C2 := Aday_Listesi[C1, h];
      Eğer (Yarıçap > d(C1, C2)) ise
      { ÖC2 := Turdaki C2 şehrinden önceki şehir;
         Kazanç := -Yarıçap + d(C1, C2) + d(ÖC1, ÖC2) – d(ÖC2,C2);
         Eğer (Kazanç < 0) ise { Gelişim := TRUE; GOTO 100; }
      } Değilse h döngüsünden çık;
   }

   Eğer (Gelişim=TRUE) ise
   {   100: 2-OPT yerdeğişimi yap;
   } değilse
   { DLB[C1]:= TRUE; }
   200: }
```

Şekil 4.31. Hızlandırılmış 2-Opt Yerel Arama Metodu

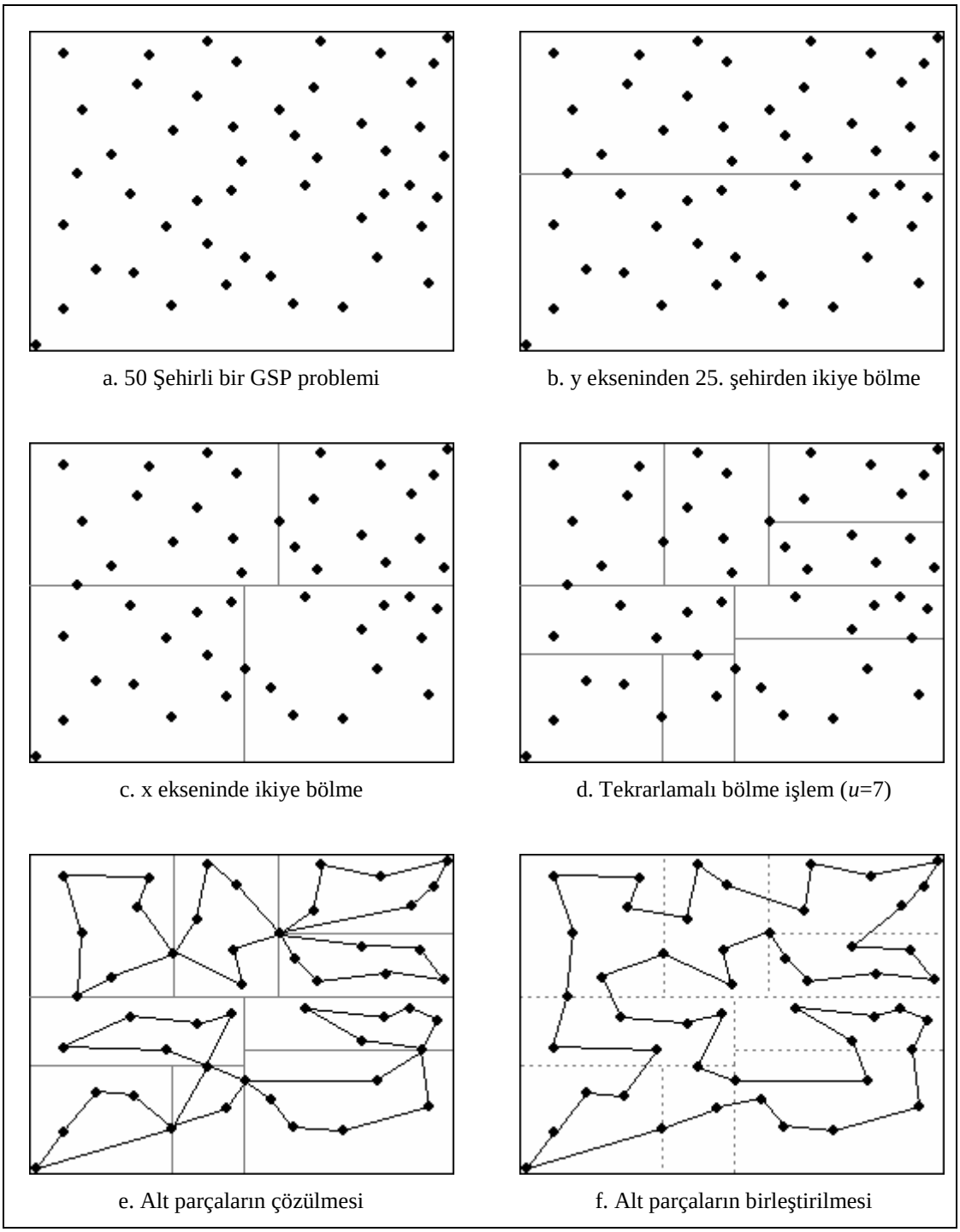
4.6. Geometrik Bölme Metotları

S-GSP problemleri için (n-1)!/2 ihtimal söz konusudur. Burada dikkat edersek şehir sayısının artması çözüm uzayı sayısını çok büyük ölçüde arttırmaktadır. Yani şehir sayısının artması, çözüm kalitesini hızlı bir şekilde azaltmaktadır (Valenzuela ve Jones 2001). Tablo 4.5'de GSP'deki şehir sayısının çözüm uzayının büyüklüğü üzerindeki etkisi gösterilmektedir.

Tablo 4.5. GSP'deki Şehir Sayısı ile Çözüm Uzayı Büyüklüğü İlişkisi

Şehir Sayısı	**Çözüm Uzayı (n-1)!/2**
5	12
10	181440
100	4.666×10^{155}
500	1.220×10^{1131}
1000	2.012×10^{2564}
5000	4.229×10^{16321}

Şehir sayının artması ile olası çözüm kümesinin ivmeli artışı, optimum değerlere ulaşmayı engellemekte ve problem çözümündeki hesaplama süresini afaki değerlere taşımaktadır. Bu sorunun önlenmesi için Genetik Algoritmaların problemi parçalara ayırarak (Karp metodu ile) arama yöntemi Valenzuela ve Jones (2001) tarafından önerilmektedir. Karp (1977) tarafından ortaya konan tur bölme metodunda, R alanına sahip bir GSP probleminde bulunan n şehir, kullanıcı tarafından belirlenen u adet şehir içererek alt parçalara bölünür. Bölümlenen alt parçaların her birisi dikdörtgendir ve komşu dikdörtgen içerisindeki şehirlerden birisi ortaktır. GSP problemini bölme işlemi, yatay ve dikeyde alt problem gruplarının sayısı u oluncaya kadar tekrarlamalı bir şekilde devam eder. İlk başta GSP problemi, -üst sınır noktasından başlamak üzere- y ekseninden [n/2]. noktasından ikiye bölünerek parçalamaya başlanır ve problem üst ve alt dikdörtgenlere ayrılır. Ardından -sol sınır noktasından başlamak üzere- x ekseninden [n/2]. noktasından ikiye bölünür. Bu işlem Şekil 4.32'deki gibi tekrarlanarak devam eder ve elde edilen alt turlar tanımlanan bir açgözlü birleştirme algoritması ile birleştirilir.

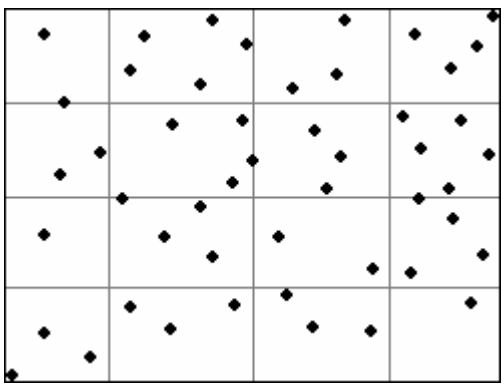


Şekil 4.32. Karp Tur Bölme Metodu

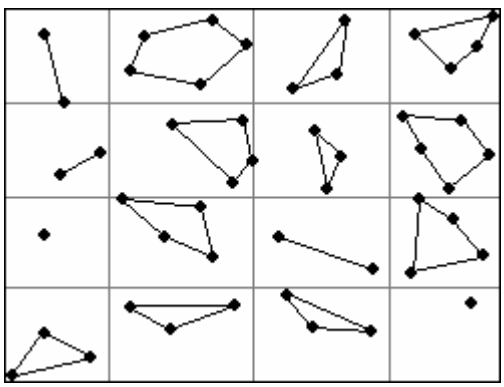
Hazırlanan tez çalışmasında yeni geliştirilen iki farklı bölme tekniği denenmiştir. Parçalama işlemlerinin her ikisinde de Karp metodundaki *u* sabiti yerine kullanıcı tabanlı parçalama (bölme) sayısı girilmektedir. Yeni tanımlanan bölme

işlemlerinde, GSP probleminde yer alan şehir sayısının yoğunluğuna göre değil bölgesel dağılım yoğunluğuna göre yapılmaktadır.

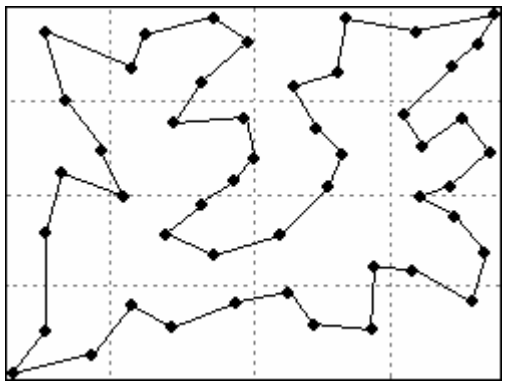
Şekil 4.33'da gösterilen birinci metotta, kullanıcının girdiği bölgesel bölme sabiti (β) oranında problem yatayda ve dikeyde eşit desenlere parçalanmaktadır ve her alt problem parçası GA ile ayrı ayrı optimize edilip 3-Opt yerel arama tekniği ile birleştirilmektedir. Birinci metoda göre β sabiti ile parçalanan GSP probleminde β^2 alt problem oluşmaktadır.



a. Problemin 16'ya bölünmesi (β=4)



b. Alt parçaların GA ile optimize edilmesi

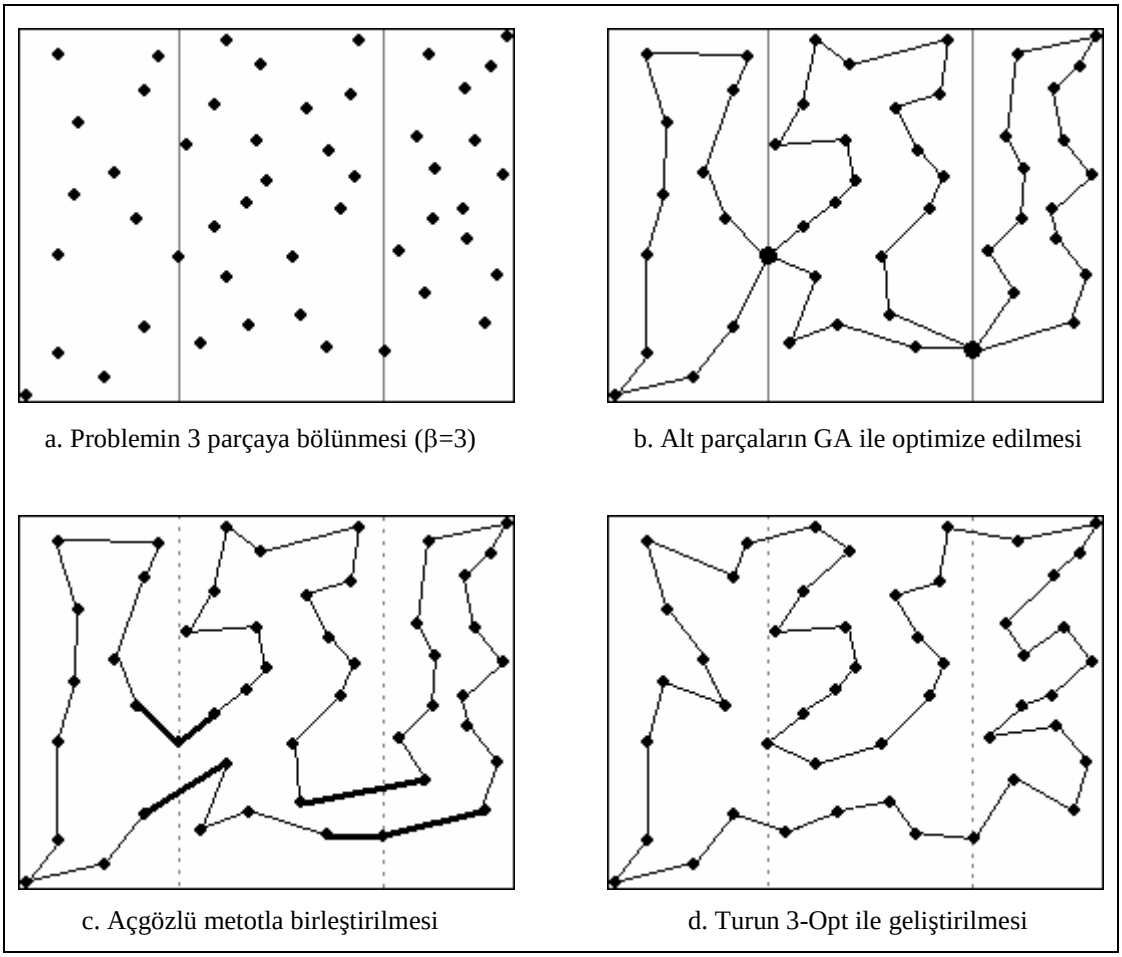


c. Alt parçaların 3-Opt ile birleştirilmesi ve optimizasyonu

Şekil 4.33. Tur Bölme Metodu–1

Şekil 4.34'de gösterilen ikinci metotta ise bölgesel bölme sabiti (β) oranında sadece yatayda bölme işlemi ve iki ortak nokta tanımlaması yapılmış, her parça birinci metottaki gibi GA ile optimize edilip açgözlü birleştirme metodu ile

birleştirilmekte ve 3-Opt metodu ile tur geliştirilerek optimizasyon süreci tamamlanmaktadır.



a. Problemin 3 parçaya bölünmesi (β=3)

b. Alt parçaların GA ile optimize edilmesi

c. Açgözlü metotla birleştirilmesi

d. Turun 3-Opt ile geliştirilmesi

Şekil 4.34. Tur Bölme Metodu–2

4.7. Paralel Genetik Algoritmanın Yapısı ve Bileşenleri

Fazla çözüm süresi gerektiren büyük ve karmaşık optimizasyon problemlerinde kullanılan bir teknik olan paralelizm, paralel bilgisayarların bir problem çözümü için belirlenen stratejilerde iletişim kurarak çözüm üzerinde çalıştırılmalarıdır. GA'da ise popülasyonun alt popülasyonlara bölünerek farklı bilgisayarlar üzerinde optimizasyon işleminin yapılması ile oluşan yeni sisteme Paralel Genetik Algoritma (PGA) adı verilmektedir. PGA'larda bilgisayarlar, işin belli bir kısmını üstlenebileceği gibi, belirli zaman dilimlerinde veya gelişimlerde

sistemler arası gelişen örneklerin paylaşılarak (göç) optimizasyona kaldıkları yerden devam edilebilir. PGA'nın arama kalitesini ve performansını etkileyen birçok parametre mevcuttur. Bunlar BGA parametreleri, alt popülasyon sayısı ve büyüklüğü, topoloji ve göç politikası olarak söylenebilir (Kuvat ve ark. 2007).

PGA'lar alt popülasyonların büyüklüğü ve sayısına göre Kalın Taneli (*Coarse Grain*) veya İnce Taneli (*Fine Grain*) olarak iki sınıf oluştururlar. Kalın Taneli algoritmalarda az sayıda alt popülasyon, çok sayıda birey vardır. İşlem sürelerinin çoğu alt kümelerin oluşturulmasında harcanmaktadır ve bireyler göç politikasına göre alt popülasyonlar arasında yer değiştirmektedirler. Her bir alt küme farklı bir işlemcide çalışan GA ile bağdaştırılır. Göç şekli, bireylerin işlemciler arasında (*Island Model*) sunulmasını veya komşu işlemciler arasında yer değiştirmesini (*Stepping-Stone Model*) içerir. Kalın Taneli algoritmalara Dağınık GA'da denilmektedir (Oliveira ve ark. 2004). İnce Taneli modeller, popülasyonu daha çok sayıda, daha küçük alt gruplara ayırır. Alt popülasyonlar arası iletişim, göç operatörü veya alt popülasyonların üst üste bindirilmesi (*overlapping*) ile gerçekleştirilir.

Alt popülasyonların GA'lar ile çözümlenmesi sonucu meydana gelecek olan göçün diğer alt popülasyon tarafından beklenmesi *Senkron PGA*, alt popülasyonların göç edecek olan bireyleri beklemeden optimizasyon işlemine devam etmesi *Asenkron PGA* modeli olarak tanımlanır. Ayrıca alt popülasyonlar üzerinde çalışacak olan GA'ların kontrol parametrelerinin farklı olması ile *Heterojen*, aynı olması ile *Homojen* yapılar elde edilmektedir.

PGA'ların önemli bir bileşeni olan göç politikası birkaç faktöre bağlıdır (Alba ve Troya 2002);

- Göç eden birey sayısı (*migration rate*) (m)
- Göç aralığı (*migration frequency*) (ξ)
- Göç edecek bireylerin seçimi (*selecting migrants*) (ω_S)
- Göç yerleşim politikası (*migration replacement policy*) (ω_R)
- Haberleşme şeklidir

Göç işlemi kısaca, alt popülasyondaki bireyin başka bir alt popülasyon ile paylaşılmasını, bunun sonucu olarak da genel bir iyileştirme elde edilmesini sağlar (Rebaudengo ve Reorda 1993). Bu şekilde de alt popülasyonun yerel optimumlara takılması engellenmiş olur (Kuvat ve ark. 2007).

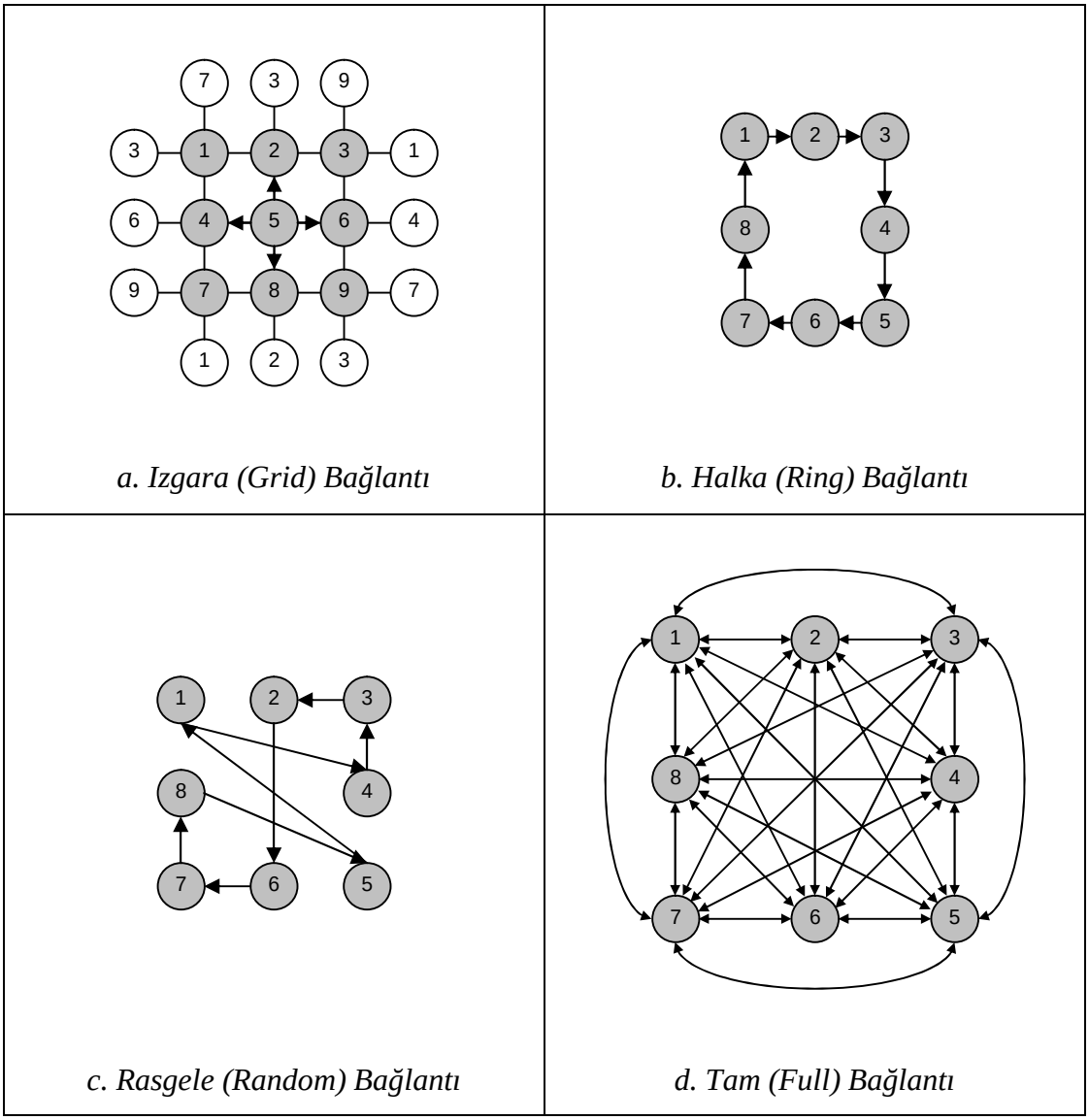
Göç Eden Birey Sayısı (*m*): Alt popülasyonlar arası paylaşılacak (göç edecek) birey sayısıdır. Göç edecek birey sayısının yükseltilmesi, sonuçta ulaşılan uygunluk ortalamasının kötüleşmesine neden olur (Rebaudengo ve Reorda 1993).

Göç Aralığı (ξ): Göç aralığı alt popülasyonlar arası yapılacak olan göçün sıklığının belirlenmesidir. Rebaudengo ve Reorda (1993) göç aralığının yüksek değerlerinde PGA'nın optimal değerlere ulaşmak için gereken jenerasyon sayısının arttığını, düşük değerlerde ise iletişim prosedürlerinin artmasından dolayı çalışma süresinin yükseldiğini belirtmektedirler.

Göç Edecek Olan Bireylerin Seçimi (ω_S): Alt popülasyonlardan hangi birey yada bireylerin diğer düğümlerdeki alt popülasyonlara göç edileceğinin seçilmesi işlemidir. Bu seçim işlemi için Rasgele, En İyiler veya Rulet Tekerleği stratejileri kullanılabilir. Tez çalışmalarında En İyiler stratejisi uygulanmıştır.

Göç Yerleşim Politikası (ω_R): Seçilen *m* bireyin göç işleminde hangi alt popülasyonlara, hangi bireyler olarak yerleşeceğinin seçilmesi işlemidir. Yerleşim işlemi genellikle rasgele (seçkin birey haricinde) veya alt popülasyondaki en kötülerin yerine yapılmaktadır. Geliştirilen yazılımda *m* bireyleri düğümdeki en kötü bireylerin yerine yerleşmektedir.

Düğümlerin Bağlantı Şekli: Düğümlerin birbirlerine bağlanması işlemi için birçok topoloji mevcuttur. Halka (*Ring*), Izgara (*Grid*), Rasgele (*Random*), En İyi/Kötü (*Best/Bad*), ve Tam Bağlı (*Full Connected*) bunlardan bazılarıdır (Belkadi ve ark. 2006). Geliştirilen yazılımda düğümler birbirlerine *"Tam Bağlı"* olacak şekilde düzenlenmiştir (Şekil 4.35).



a. Izgara (Grid) Bağlantı

b. Halka (Ring) Bağlantı

c. Rasgele (Random) Bağlantı

d. Tam (Full) Bağlantı

Şekil 4.35. PGA Bağlantı Topolojileri

4.8. Diğer Kullanılan Metotlar

Genetik Algoritmalara verilen bir GSP probleminde optimum değere yaklaştıkça popülasyonda yakınsama meydana gelmektedir yani tüm bireyler gittikçe birbirlerine benzemektedir. Bu yakınsama sonucunda da GA'nın temel görevlerinden olan güçlü bireylerin soylarını yeni nesillere aktarmak için kullanılan çaprazlama operatörleri aynı bireyleri veya çok benzer bireyleri çaprazlamaktadır. Bu yüzden de çözüm uzayının bir bölgesinin optimum değeri elde edilmektedir. Her ne kadar mutasyon operatörü popülasyonu bu yerellikten çıkarmakta ise de, aynı bireylerin

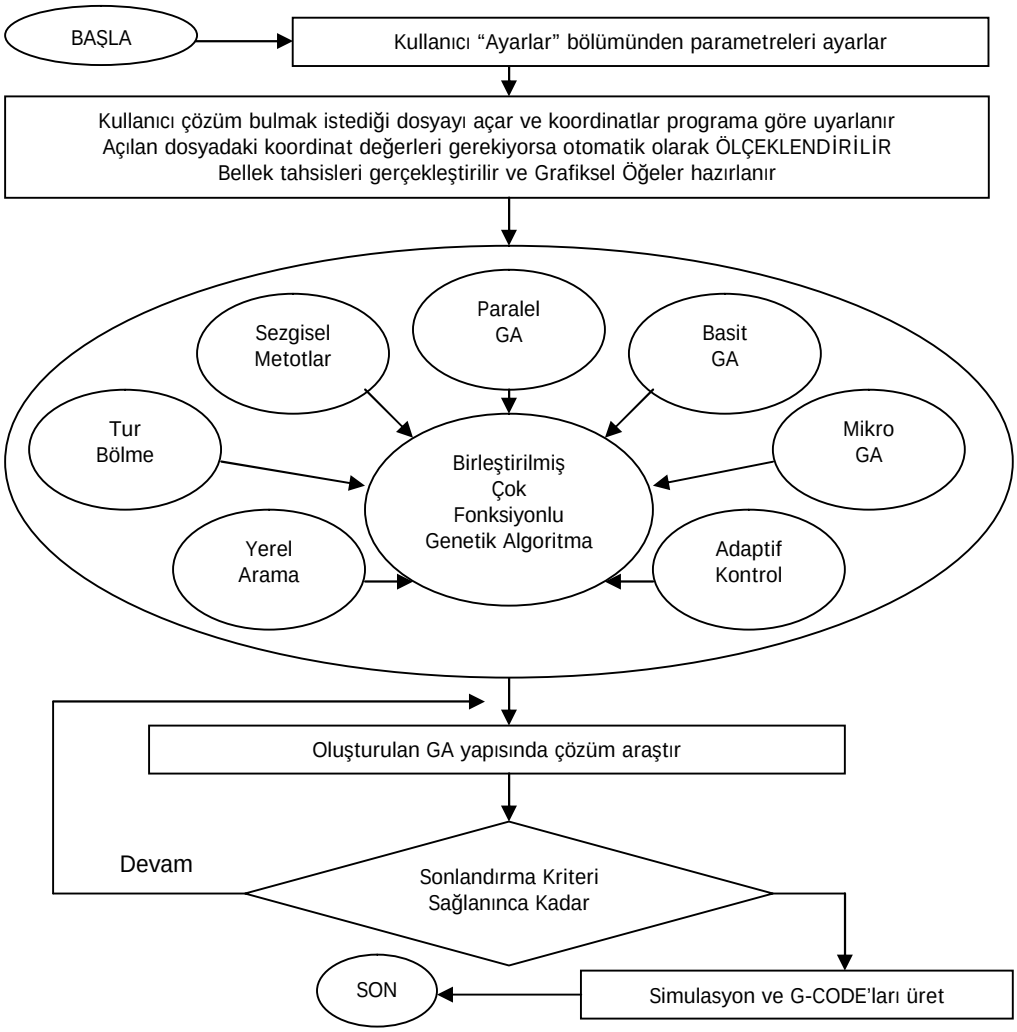
çaprazlanması atıl sürelere neden olmaktadır. Ayrıca optimuma daha yakın değerler elde etmek için popülasyonun çözüm uzayında birçok bölge içerisinden çözüme ulaşması gerekmektedir. Bu problem aşağıdaki yollar ile çözülebilir.

- Mutasyon operatörü çaprazlamadan önce uygulanmalıdır,
- Popülasyon sayısı oldukça büyük tutulmalıdır,
- Üretilen yavruların popülasyona kazandırılması sırasında popülasyondaki bireylere benzerlikleri kontrol edilerek eklenmelidir,
- Eşleştirme işleminde aynı bireylerin eşleştirilmesi engellenmelidir,
- Popülasyona belirli bir ihtimal dâhilinde bir birey eklenir,
- Popülasyonun yakınsama durumu kontrol edilerek, popülasyona belirli yüzdeliklerde çözüm uzayından görülmemiş örnekler eklenmelidir

Yukarıda sayılan yöntemlerden birincisi çaprazlama işlemine farklı bireylerin girmesini sağlasa da evrimsel algoritmaların temeli olan “güçlü bireyler soylarını sürdürür” kavramı ortadan kalkmaktadır. Çünkü mutasyona uğrayan bireyler artık uygunluk yönünden *iyi* olmamaktadır. Diğer bir alternatif olan popülasyon sayısının büyük seçilmesi çözüm süresini oldukça uzatacaktır. Üretilen yavruların popülasyona tekrar kazandırılması sırasında yapılacak olan kontrolde de aynı bireylerin popülasyonda zaten mevcut olma ihtimalinden atıl zamanlar oluşturacaktır. Bu yüzden yakınsamanın engellenerek popülasyonun sürekli uzaydaki farklı bölgeler üzerinden optimuma yaklaşmasını sağlamak için eşleştirmede aynı bireylerin eşleştirilmesi engellenmekte, belirli bir yüzdelik ihtimal dâhilinde popülasyondan rasgele seçilen bir birey (elitist bireyler hariç) yerine yeni rasgele bireyler eklemek (Jayalakshmi ve ark. 2001) ve kullanıcının belirlediği bir yakınsama süresinde popülasyona belli yüzdelerde yeni adaylar eklenmektedir (Rocha ve Neves 1999).

5. GENETİK ALGORİTMA VE BİLEŞENLERİ İÇİN UYGUN PARAMETRELERİ BELİRLEME

Simetrik GSP (S-GSP) problemlerinin çözülmesi için geliştirilen programın kodlanması için Borland Delphi 7.0 programlama dili kullanılmıştır. Geliştirilen yazılımda bütün kullanılan metotlar ve teknikler kullanıcı tabanlı olacak şekilde ayarlanmıştır. Ayrıca bellek organizasyonu dinamik olarak yapılmıştır. Genetik Algoritma tabanlı geliştirilen programının basit bir akış diyagramı Şekil 5.1.'deki gibidir.



Şekil 5.1. Geliştirilen Programın Basit Akış Diyagramı

Geliştirilen yazılımında birçok farklı teknik ve metot birleştirilerek bir optimizasyon işlemi yapıldığı için parametrelerin seçimi problem çözümünün kalitesini önemli ölçüde etkilemektedir. Bu yüzden çözüm kalitesini etkileyen bütün parametreler ayrı ayrı analiz edilerek en ideal parametreler belirlenmelidir. Parametrelerin belirlenmesi işlemi için TSPLIB'de (Gezgin Satıcı Problemi Kütüphanesi) yer alan 15 farklı problem üzerinde analizler yapılmıştır.

Analizlerde üç farklı bilgisayar konfigürasyonu kullanılmıştır. Bilgisayarların konfigürasyonları Tablo 5.1'deki gibidir.

Tablo 5.1. Kullanılan Bilgisayar Konfigürasyonları

Parça	Konfigürasyon-1 (*Konf*-1)	Konfigürasyon-2 (*Konf*-2)	Konfigürasyon-3 (*Konf*-3)
İşlemci	Intel E6600 Core 2 Duo-2.4Ghz	Intel Pentium IV 2.8Ghz HT	AMD Turion-64 ML32 1.8Ghz
RAM Bellek Frekans	1 GB DDR2-667 Mhz	512 MB DDR-333 Mhz	512 MB DDR-333 Mhz
Ekran Kartı Bellek	Bütünleşik 128 MB	GeForce FX5200 128 MB	Bütünleşik 64 MB
Sabit Disk Bağlantı	160 GB SATA	80 GB SATA	80 GB IDE
İşletim Sistemi	Windows XP Professional – SP2	Windows XP Professional – SP2	Windows XP Professional – SP2

5.1. Geliştirilen Yazılım Özellikleri

Borland Delphi 7.0 programlama dili kullanılarak geliştirilmiş olan yazılım yaklaşık 20000 satırın üstünde kaynak koddan oluşmakta ve Windows platformunda çalışmaktadır. Program tamamen prosedürler, fonksiyonlar ve dinamik bellek tahsislerine sahip olduğundan dolayı kullanıcı her an programa müdahale edebilmekte, parametreleri istediği şekilde değiştirebilmektedir.

Program algoritma olarak aşağıdaki modelleri gerçekleştirebilmektedir.

- Genetik Algoritma (GA)
- Mikro Genetik Algoritma (μGA)
- Paralel Genetik Algoritma (PGA)
- Geometrik Tur Bölme + GA Optimizasyonu
- Yerel Arama Algoritması
- Genetik Yerel Arama Algoritması
- Paralel Genetik Yerel Arama Algoritması

Ayrıca sezgisel metotlar ve adaptif strateji yukarıdaki her bir algoritmanın üstüne eklenerek de çalıştırılabilmektedir. Geliştirilen yazılım, içerik yönünden oldukça zengin olup GSP problemi için kullanılan birçok metodu bünyesinde toplamaktadır. Yazılım içinde kullanılabilecek metotlar aşağıdaki gibi listelenebilir.

- 16 farklı çaprazlama operatörü
- 12 farklı mutasyon operatörü
- 7 farklı yerel arama metodu
- 2 farklı aday listesi yapısı ve 3 farklı uygulama düzeni
- 7 farklı tur başlangıç metodu
- 2 farklı adaptif strateji
- 2 farklı yakınsamayı engelleyici modül
- 2 farklı tur bölme metodu
- 4 farklı sonlandırma kriteri
- 6 farklı seçim metodu
- 7 farklı eşleştirme metodu

Genel olarak değerlendirildiğinde geliştirilen programda yer alan bazı özellikler aşağıdaki gibi sıralanabilir.

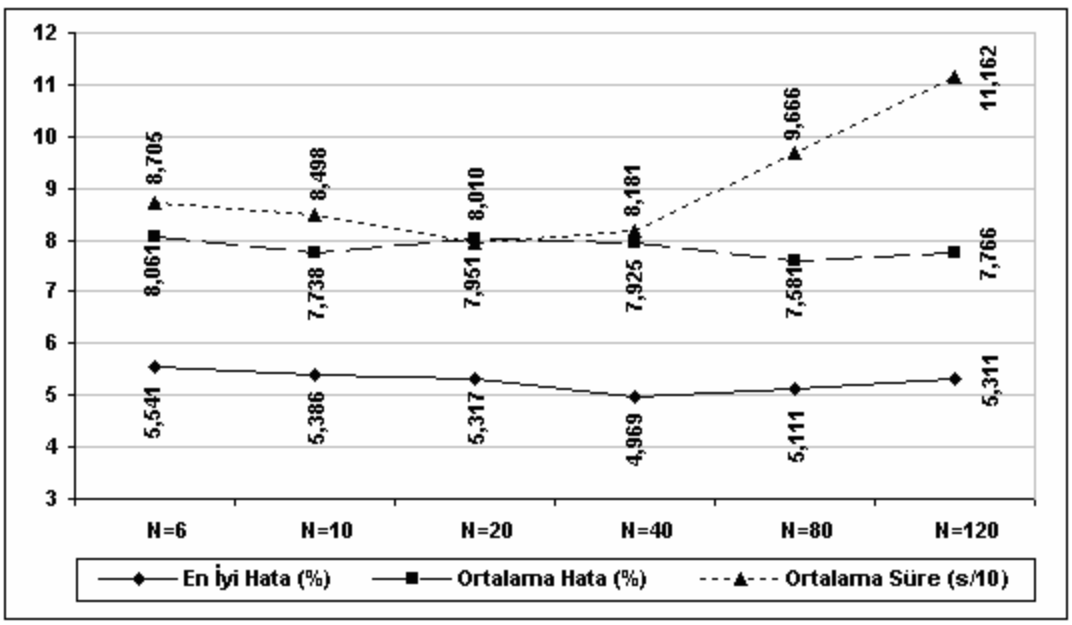
- Hiçbir ek bileşene veya kütüphane dosyasına ihtiyaç duymadan çalışabilir,
- Birden fazla algoritmayı birleştirebilme imkânı sağlar,
- Optimizasyon süresince GA ayarlarını değiştirebilme imkânı sağlar,
- Rasgele GSP dosyası oluşturabilmeyi sağlar,
- Kullanıcı arabirimi ile koordinat oluşturabilmeyi sağlar,
- GSP üzerinde otomatik ölçeklendirme işlemi yapabilmeyi sağlar,
- GSP problemindeki şehirleri ve popülasyondaki bireylerin güzergahlarını görsel olarak izleyebilme imkânı sağlar,
- Analizler sırasında değişim gösteren parametrelerin değerlerini grafik olarak işletilebilmeyi sağlar. Örneğin, çaprazlama ve mutasyon oranı, iterasyon başı çevrim süresi, aday listesi büyüklüğü, popülasyon bireyleri, uygunluk değişimi ve PGA'daki farklı düğümlerin çalışmaları gibi,
- Grafik işlemlerin kullanıcı tarafından durdurulabilmesini sağlar,
- Sabit diskte yaklaşık 1,10MB gibi oldukça az bir alana ihtiyaç duyar,
- Literatürde yer alan EUC-2D tipi S-GSP problemlerini destekler,
- Boşta iken yaklaşık 5,5MB RAM belleğe ihtiyaç duyar,
- Popülasyonların kaydedilebilmesini ve sonra açılabilmesini sağlar,
- Optimal turun kaydedilmesini ve sonradan açılabilmesini sağlar,
- Optimal tur için simülasyon yapılabilmeyi sağlar,
- G-Code üretebilmeyi sağlar (*Fanuc Cincinnati* kontrol ünitesi için),
- 16 farklı parametre için analiz yapabilmeyi sağlar,
- Optimizasyon boyunca Metot-Süre ilişkilerini takip eder ve
- Dizilerin sıralanmasında Hızlı Sıralama Metodunu (*Quick Sort*) kullanır.

Özellikle büyük GSP problemlerinin geliştirilen yazılım ile optimize edilebilmesi için yüksek frekanslı bir işlemcinin yanında büyük RAM belleğe ihtiyaç vardır. Çünkü her açılan GSP probleminde yapılacak olan bellek tahsisi problemdeki şehir sayısına bağlıdır. Özellikle oluşturulan ağırlık matrisi, komşuluk matrisi ve komşuluk ağırlıkları matrisi RAM bellekte oldukça fazla alan işgal edebilir.

5.2. Genetik Algoritmada Popülasyon Büyüklüğünün Etkisi

Genetik Algoritmaların performansını ve çözüm kalitesini önemli ölçüde etkileyen Popülasyon Sayısı (*N*), GSP problemlerinin büyüklüğüne göre değişiklik göstermektedir. Ek-Tablo 1'deki her bir GSP problemi için 10 deneme yapılmış ve ortalama hata değerleri verilmiştir. Her bir araştırma sürecinde, popülasyondaki en iyi bireyin 3000 jenerasyon değişmemesi gözetilmiştir. Araştırmalarda BGA kullanılmıştır. Popülasyon rasgele üretilmiş, çaprazlama operatörü olarak OX ($P_Ç$=%100) ve mutasyon operatörü olarak da GST (P_M=%100) kullanılmıştır.

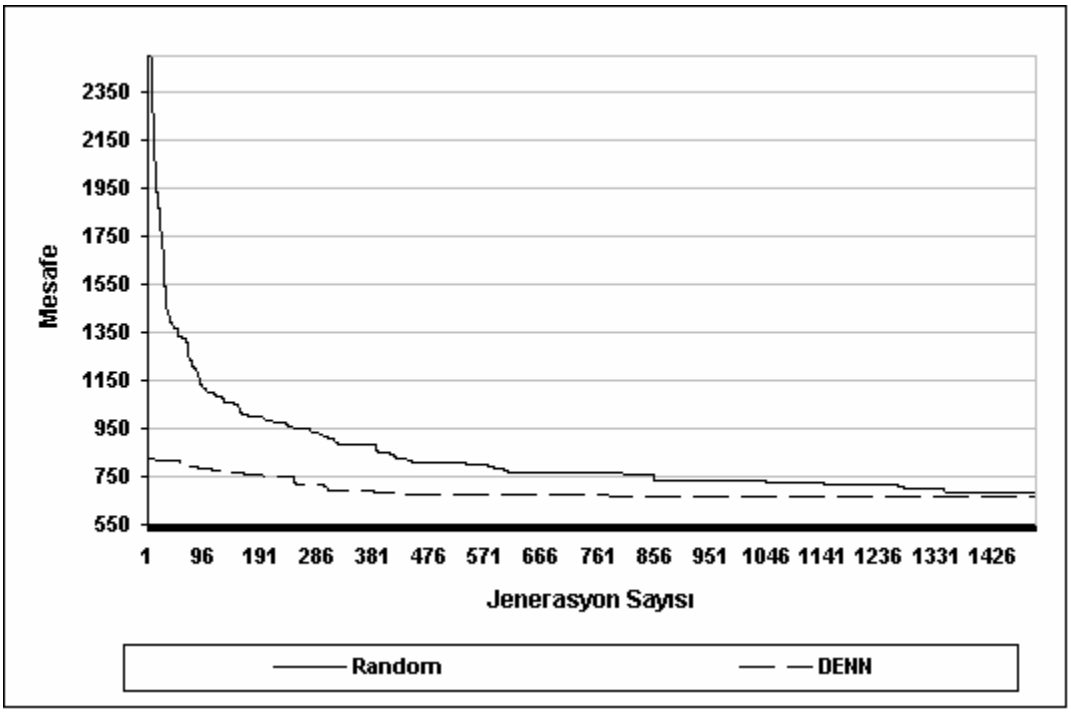
En ideal popülasyon sayısının belirlenmesi için yapılan analizlerde görülmüştür ki, popülasyon sayısının çok küçük seçilmesi hem yüksek hata oranına sahip çözümler vermekte hem de çok kısa sürede popülasyondaki bireylerin yerel optimum bölgelerine düşmesine neden olmaktadır. Popülasyon sayısının büyütülmesi durumunda ise popülasyondaki bireylerin yerel optimuma yakalanma riski ve hata oranı azalmaktadır. Fakat popülasyon sayısı arttıkça da toplam çevrim süresi ve bellek tahsisi artmaktadır. İncelemelerde popülasyonun en ideal büyüklüğün [40-80] arasında olması gerektiği görülmüştür (Şekil 5.2).



Şekil 5.2. Popülasyonunun Büyüklüğünün Etkisi

5.3. Tur Oluşturma Metotlarının Çözüm Kalitesi

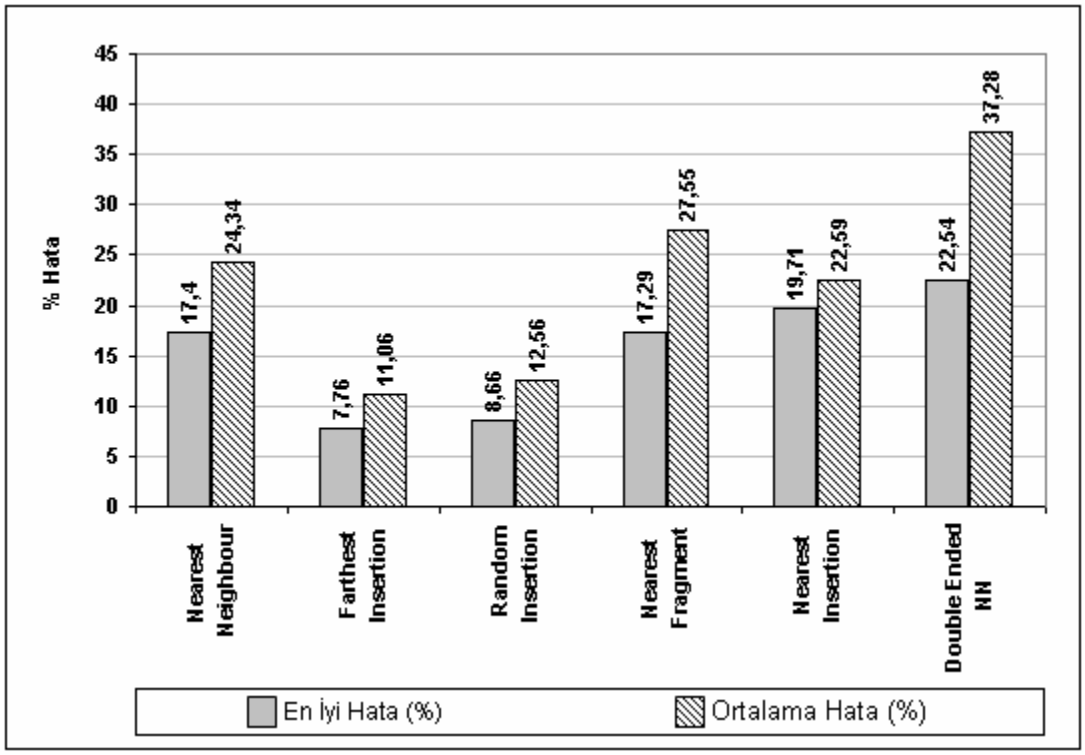
BGA'larda başlangıç popülasyonları genellikle Rasgele (*Random*) üretilmektedir. Rasgele üretilen popülasyonlar çözüm uzayından birbirine benzemeyen daha çok birey örnekleyebilmesi yönünden avantajlı gibi gözükse de optimum çözümü elde etmede harcanacak süreyi çok büyük ölçüde arttırmaktadır. Çünkü BGA'lar, rasgele üretilmiş bir popülasyonu Tur Oluşturma Metotlarıyla başlatılmış bir popülasyon seviyesine getirinceye kadar oldukça fazla nesil üretmesi gerektiğinden gereksiz yere efor harcar ve özellikle büyük GSP problemlerinde Tur Oluşturma Metotlarıyla oluşturulan popülasyon seviyesine gelebilmek için harcanan süre azımsanmayacak kadar fazladır. Şekil 5.3'de gösterildiği gibi, Eil101 probleminin rasgele üretilmiş bir başlangıç popülasyonunun DENN ile üretilmiş başlangıç popülasyonu seviyesine gelebilmesi için yaklaşık 530 *ms* süreye, aynı hata oranına gelebilmesi için de yaklaşık 1460 *ms* ek süreye ihtiyacı vardır.



Şekil 5.3. Başlangıç Popülasyonunun Zaman Üstündeki Etkisi

Bu yüzden GA'da başlangıç popülasyonunun probleme özgü olarak daha iyi ifade edilebilmesi GA'nın performansını etkileyen kriterlerdendir. Bu amaçla

başlangıç popülasyonunun oluşturulması için kullanılan çeşitli Tur Oluşturma Metotları Ek-Tablo 2'deki gibi analiz edilmiştir. Ek-Tablo 2'deki her bir analizde 100 bireylik bir popülasyon oluşturulmuş, bu popülasyondaki en iyi bireylerin (optimum değere en yakın) ve popülasyondaki bireylerin hata değerlerinin ortalaması hesaplanmıştır. Buna göre Şekil 5.4'deki gibi en ideal sonuçları üreten Tur Oluşturma Metodu, en iyide ($\varepsilon_{İYİ}$=%7.76) ve ortalamada (ε_{ORT}=%11.06) *Farthest Insertion* (FI) olarak görülmüştür.

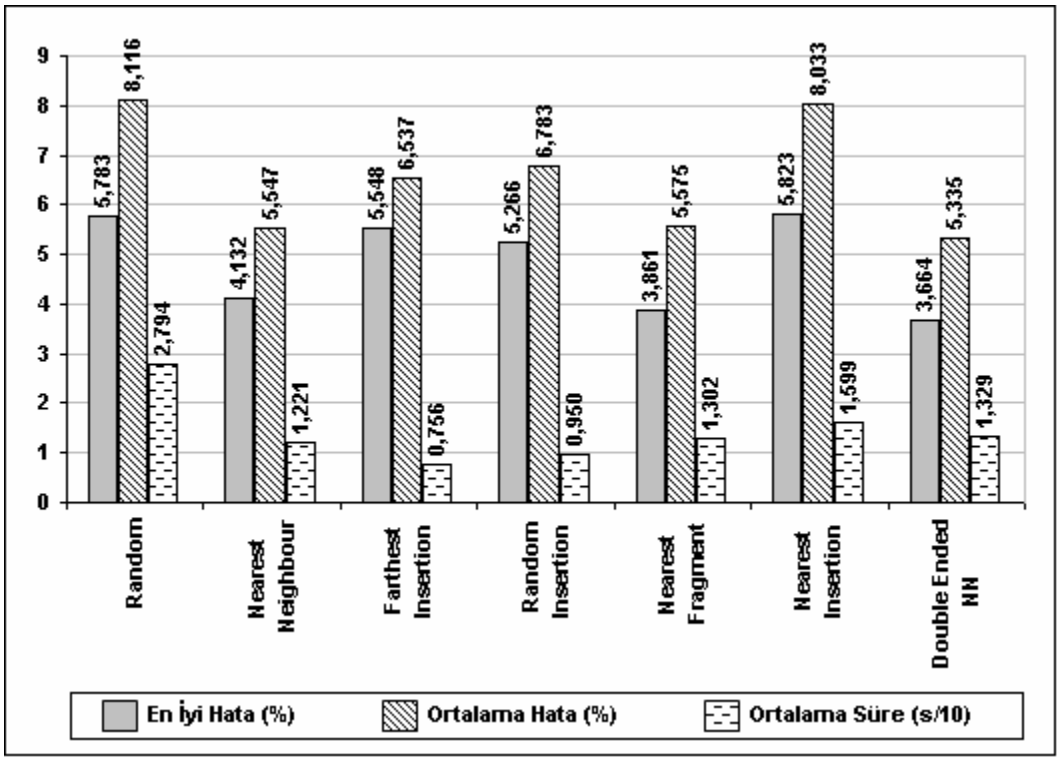


Şekil 5.4. Tur Oluşturma Metotlarının Hata Oranları

5.4. Tur Oluşturma Metotlarının Genetik Algoritma Üstündeki Etkisi

Tur Oluşturma Metotları ile üretilen başlangıç popülasyonlarının daha düşük hata değerleri ile çözüme alınmasıyla çözüm süresinin azaltılması mümkündür. Fakat optimum değere çok yakın üretilmiş popülasyonların kullanılması GA'ları yerel optimumda tıkanmasına neden olabilir. Bu nedenle Ek-Tablo 3'de 7 farklı başlangıç metodu ile başlatılan BGA'nın en iyi ve ortalama hata değerleri analiz edilmiştir. Bu

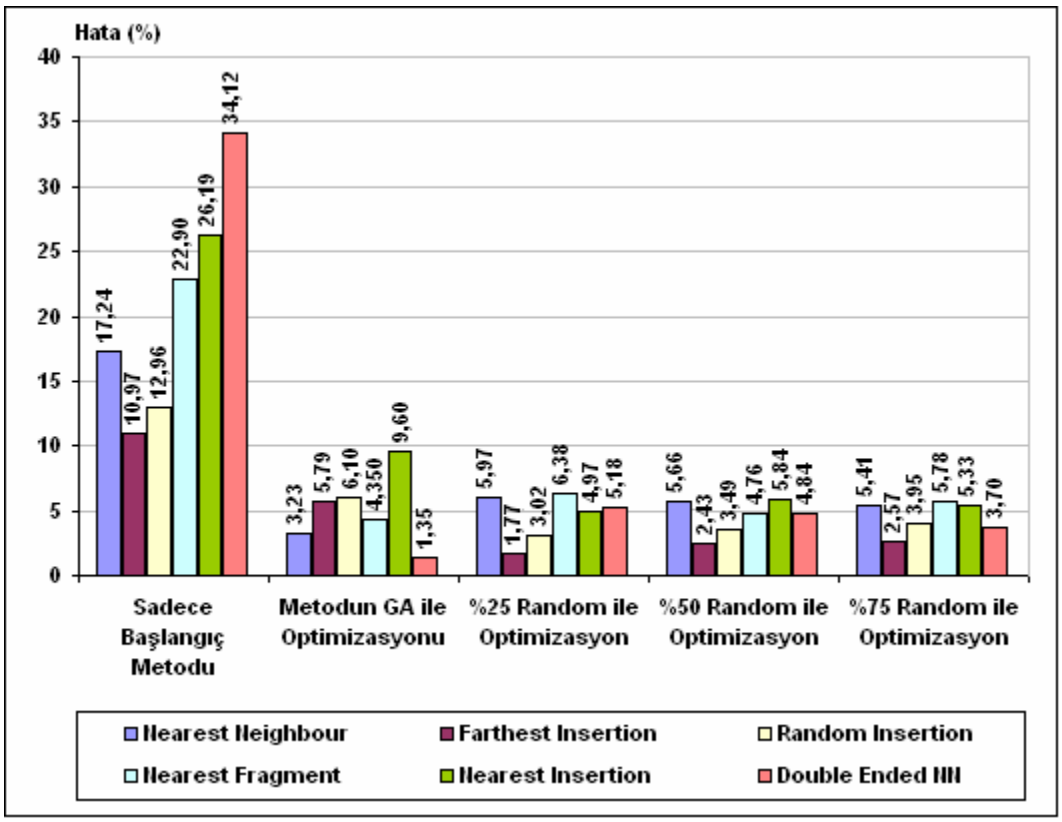
analizlerde her bir başlangıç metodu ile başlatılan problem için 10'ar deneme yapılmıştır. GA ayarları olarak, popülasyon sayısı 40, çaprazlama operatörü OX ($P_Ç$=%100), mutasyon operatörü GST (P_M=%100) ve sonlandırma kriteri olarak ta popülasyondaki en iyi bireyin 3000 jenerasyon değişmemesi seçilmiştir.



Şekil 5.5. Başlangıç Metotlarının Genetik Algoritma Üzerindeki Etkisi

Başlangıç popülasyonunun üretimindeki hata değeri olarak en iyi değerleri FI verse de, GA ile optimizasyon sürecinde daha neredeyse en kötü sonuçlardan birisini ($\varepsilon_{İYİ}$=%5.548, ε_{ORT}=%6.537) vermektedir. Öte yandan başlangıç popülasyonu üretimindeki hata değerlerinde en kötü olarak gözüken DENN ise GA ile optimize edildiğinde en iyi sonucu vermiştir. Buna göre Ek-Tablo 3'deki değerlere göre Şekil 5.5'deki grafik çizilebilir. Fakat FI ve DENN arasında çalışma süresi bakımından ciddi bir farklılık vardır. FI ile üretilen popülasyonlar daha kısa sürede optimize edilebilirken, DENN ile üretilen popülasyonların optimuma yaklaşma süreci daha uzun sürebilmektedir. Buradan, başlangıç popülasyonun optimumdan daha uzak başlaması ve GA ile araştırma sürecinde optimize edilmesi gerektiği, bunun da daha fazla çevrim süresine karşılık geldiği görülmektedir.

Bu problemi aşabilmek için başlangıç popülasyonunun belirli bir yüzdelik değerinde Tur Oluşturma Metotları ve Rasgele (*Random*) üretilmiş bireyler ile oluşturularak GA ile optimize edilmesinden Ek-Tablo 4'deki veriler bulunmuş ve bu veriler ile önceki verilerin KroA150 S-GSP için birleştirilmesi ile Şekil 5.6'daki grafik elde edilmiştir.



Şekil 5.6. Başlangıç Metotlarının Karıştırmalı Analiz Ortalamaları

Sonuç olarak yapılan analizler ışığında başlangıç popülasyonları için aşağıdaki gibi bazı genelleştirilmiş sonuçlar çıkartılabilir.

1. Kullanılan metotlar içerisinde sadece başlangıç popülasyonu üretme safhasında en düşük ortalama hata (ε_{ORT}) değerlerini veren Tur Oluşturma Metodu %11.06 ile FI metodudur (Ek-Tablo 2).
2. Başlangıç popülasyonu olarak Tur Oluşturma Metotları seçilerek GA ile optimizasyon yapıldığı zaman en düşük hata değerlerini veren Tur Oluşturma Metodu %5.335 ile DENN metodudur (Ek-Tablo 3). Fakat

DENN (130234 *ms*) ile analize başlanan bir popülasyonda optimum değere yaklaşmak için FI (95008 *ms*) metodundan yaklaşık %37 daha fazla ek süreye ihtiyaç duyulmaktadır (Şekil 5.5).

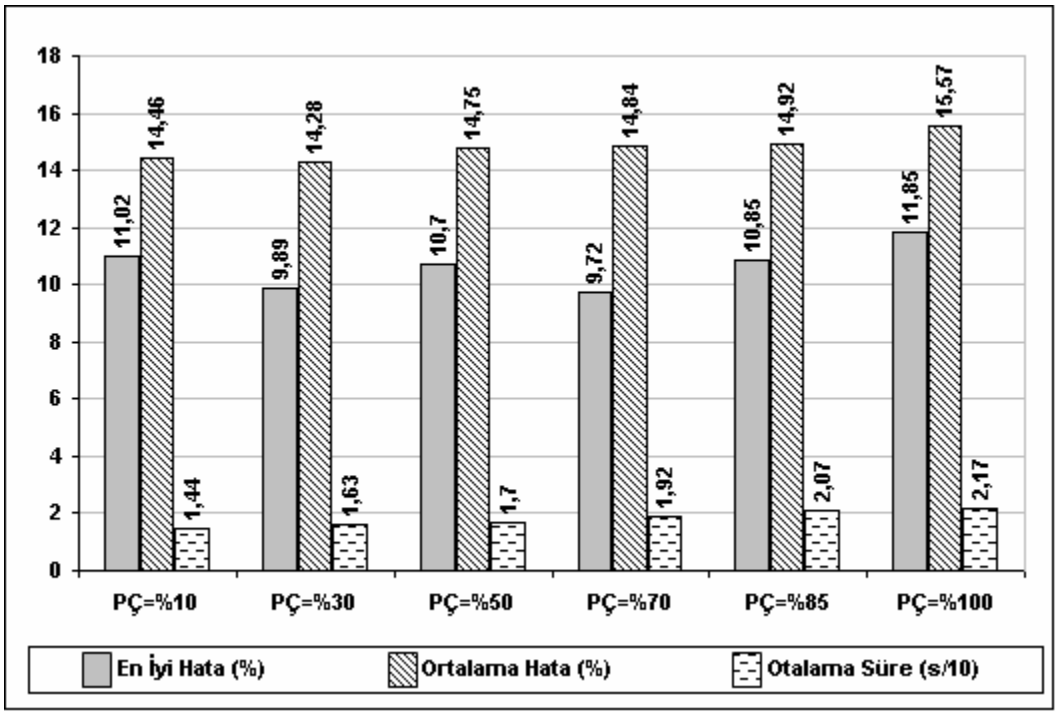
3. Popülasyondaki bireylerin hepsinin optimum değerlere yakın olması popülasyonun yerel optimuma takılma ihtimalini arttırmaktadır. Bu yüzden başlangıç popülasyonunun sadece Tur Oluşturma Metotları ile değil belli yüzdelik değerlerinde Rasgele (*Random*) üretilmiş bireyler de içermesi yerel optimuma takılma ihtimalini azaltmakta ve ulaşılan ortalama hata değerini (ε_{ORT}) düşürebilmektedir. Bu amaçla yapılan analizlerde (Ek-Tablo 4) kullanıcı tabanlı belirli yüzdelerde (%25, %50, %75) oluşturulan karışık başlangıç popülasyonlarından KroA150 problemi için (%25NN + %75Random) popülasyonunun (ε_{ORT}=%5.41) %100NN ile üretilen popülasyonundan (ε_{ORT}=%6.00), (%75FI + %25Random) popülasyonun (ε_{ORT}=%1.77) %100FI popülasyonundan (ε_{ORT}=%2.06), KroB200 problemi için (%50NN + %50Random) popülasyonunun (ε_{ORT}=%5.04) %100NN ile üretilen popülasyonundan (ε_{ORT}=%6.89) daha iyi olduğu gibi örnekler görülmüştür. Ek-Tablo 4'deki tüm verilerin ortalamaları alındığında bu yüzdelik karıştırma değerinin [25-50) arasında uygun sonuçlar ürettiği görülmüştür.
4. GA ile bir GSP optimizasyonunda, başlangıç popülasyonunun Rasgele üretilmesi (t_{ORT} = 279425 *ms*) Tur Oluşturma Metotlarına (t_{FI_ORT}=75621 *ms*, t_{NI_ORT} = 159890 *ms*) göre yaklaşık %74-270 daha fazla sürede optimum değere ulaşılmasını sağlamaktadır (Şekil 5.5).

5.5. Çaprazlama Oranının Etkisi

GA'larda çaprazlama operatörleri popülasyonda var olan iyi ebeveynler kullanılarak onların kalıtsal özelliklerini taşıyan yeni nesiller üreterek problemi daha iyi örnekleyen çözümler elde etmek için kullanılır. Ek-Tablo 5'deki her GSP probleme her bir çaprazlama oranı ($P_Ç$) için 10 ayrı deneme yapılmıştır. Tablodaki her çaprazlama oranı için en iyi ve ortalama hata değerleri ayrıca ortalama çevrim

süreleri verilmiştir. Her bir araştırma süreci popülasyondaki en iyi bireyin 2000 jenerasyon değişmemesi durumunda son bulmaktadır. Araştırma işleminde BGA kullanılmıştır. Popülasyon 10 büyüklüğünde ve rasgele üretilmiş, çaprazlama operatörü olarak OX, mutasyon operatörü olarak da SIM (P_M=%100) kullanılmıştır.

Ek-Tablo 5'deki veriler ışığında çaprazlama oranı sabit olacak şekilde seçildiğinde $P_Ç$=(30-70] arasında seçilmesi gerektiği görülmüştür. Bu analizde P_M sabit alınmış, hiçbir $P_Ç$ oranında değişmemiştir. Ek-Tablo 5'den çıkartılan Şekil 5.7 ile çaprazlama oranının daha net bir şekilde belirlenmesi mümkündür.

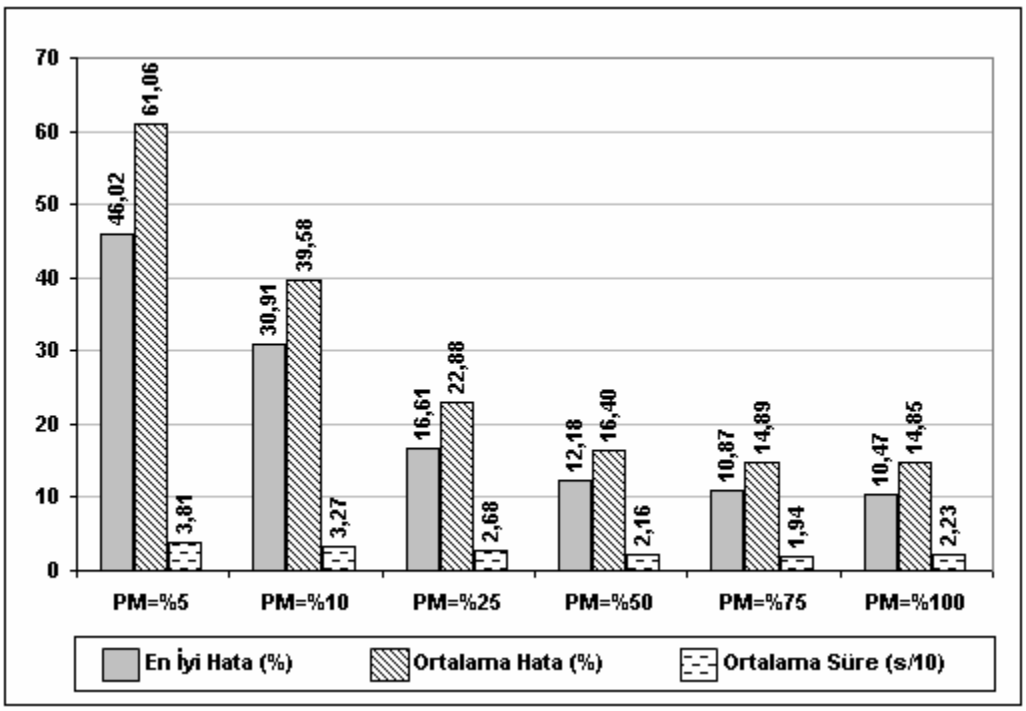


Şekil 5.7. Çaprazlama Oranlarının GSP Optimizasyonundaki Etkisi

5.6. Mutasyon Oranının Etkisi

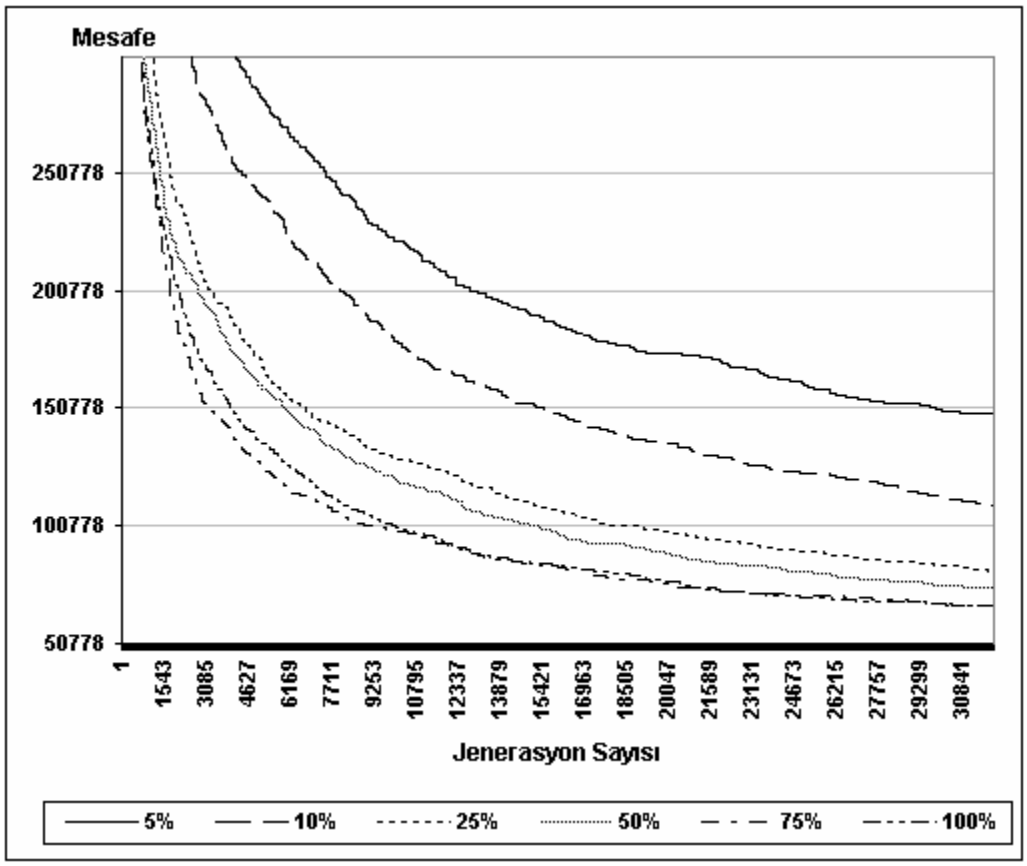
GA'larda mutasyon operatörleri popülasyonu yerel optimumdan çıkarmak, çözüm uzayında görülmemiş örnekleri sağlamak için kullanılmaktadır. Ek-Tablo 6'daki her GSP probleme her bir mutasyon oranı (P_M) için 10 ayrı deneme yapılmış en iyi ve ortalama hata değerlerinin yanında ortalama çevrim süresi verilmiştir. Her bir araştırma süreci, popülasyondaki en iyi bireyin 2000 jenerasyon değişmemesine bağlıdır. Araştırma işleminde BGA kullanılmıştır. Popülasyon 10 büyüklüğünde ve rasgele üretilmiş, çaprazlama operatörü olarak OX, mutasyon operatörü olarak da SIM kullanılmıştır.

Ek-Tablo 6'daki veriler ışığında mutasyon oranının diğer optimizasyon problemlerinin aksine, büyük seçilmesinin daha iyi sonuçlar verdiği görülmüştür. Ek-Tablo 6'dan çıkartılan Şekil 5.8 ile mutasyon oranının daha net bir şekilde belirlenmesi mümkündür. Yapılan analizlerde P_M=%(75-100] arasında olması durumunda daha düşük hatalar elde edildiği ölçülmüştür.



Şekil 5.8. Mutasyon Oranlarının GSP Optimizasyonundaki Etkisi

Şekil 5.9’daki grafikte 6 farklı mutasyon oranı BGA ile Pcb442 problemine uygulanmış ve mutasyon oranının tekrarlamalı çözümdeki etkisi gösterilmiştir (çaprazlama operatörü olarak OX, sonlandırma kriteri 50000 jenerasyon tercih edilmiştir). Görüldüğü gibi düşük seçilen mutasyon operatörü çözümün oldukça uzun bir zamanda bulunmasına, hatta bulunamamasına neden olmaktadır. Çünkü çaprazlama operatörü ile bireyler gitgide birbirlerine benzeyerek popülasyonun yerel optimuma düşmesine ve erken yakınsamaya neden olmaktadır.



Şekil 5.9. Mutasyon Oranlarının GSP Optimizasyonundaki Hızı

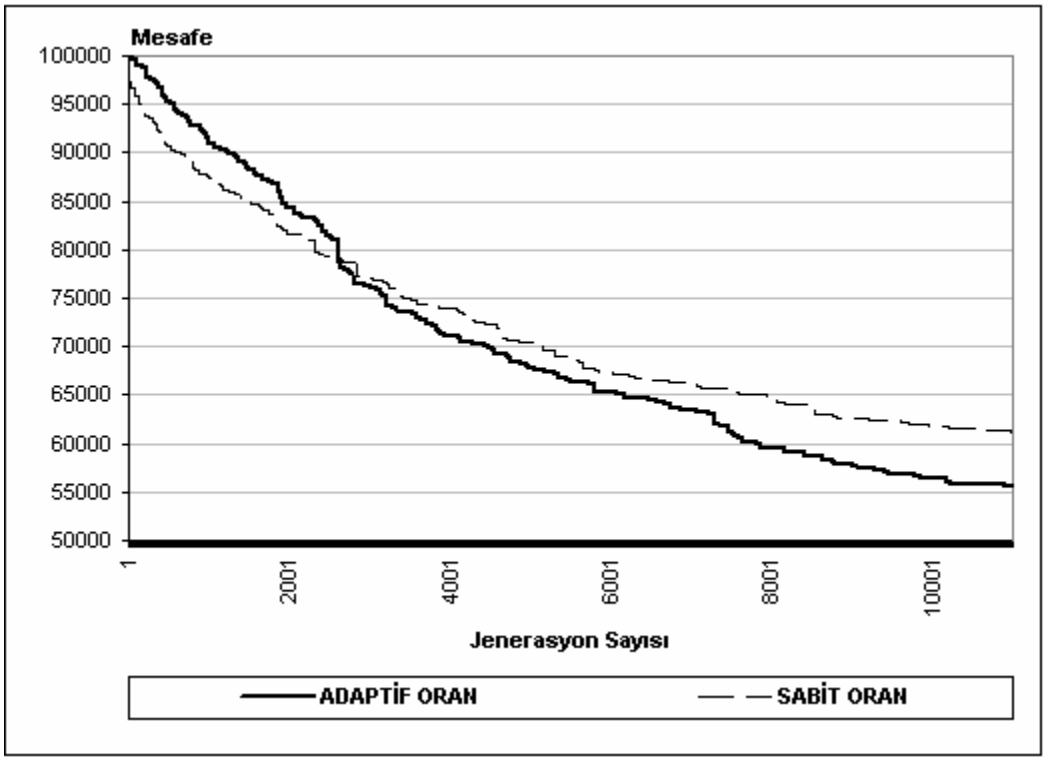
5.7. Adaptif Kontrol Parametresi Belirlemenin Etkisi

BGA kullanarak Lin318 problemine hem sabit hem de adaptif mutasyon oranı 10’ar kere uygulanmıştır. Sonuçlar Tablo 5.2’deki gibidir. Tablo 5.2’de sonuçları görünen analizlerde OX çaprazlama metodu, SIM mutasyon metodu kullanılmış ve sonlandırma kriteri en iyi bireyin 3000 jenerasyon değişmemesine bağlıdır.

Tablo 5.2. Adaptif Mutasyon Oranı Etkisi

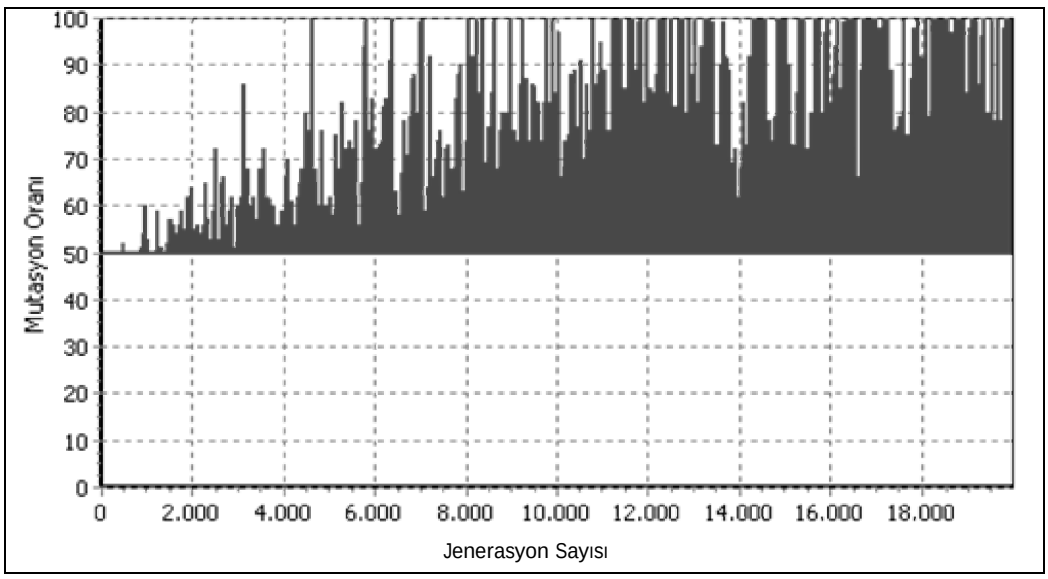
Yapılan Deneme Sayısı	Adaptif P_M (ST-1) Max=100, Min=75, $RP1$=10, $RP2$=75		Sabit P_M (P_M=%100)	
	Uygunluk	Jenerasyon	Uygunluk	Jenerasyon
1	47083	48265	46977	69811
2	47632	50037	49131	44919
3	47207	47510	48974	45294
4	46688	47259	47039	45739
5	47976	57722	46975	69657
6	47142	58808	47359	61021
7	46940	48614	47586	60033
8	48644	51786	47591	54139
9	47495	38728	47459	56246
10	46107	55997	48573	56597
Ortalama	**47291.4**	**50472.6**	47766.4	56345.6
Hata (%ε_{ORT})	**12.52**		13.65	

Analizlerden görüldüğü gibi operatör oranının adaptif olarak belirlenmesi, hem daha kaliteli çözümler üretilmesini yaklaşık %1 oranında arttırmakta, hem de uygun çözümün bulunma süresini yaklaşık %11 oranında azaltmaktadır. Sonuç olarak operatörlerin uygulanma ihtimallerinin çözüm süresince hesaplanması, işlem boyunca sabit kalmasından daha etkilidir. Şekil 5.10'da GA'ya verilen Lin318 probleminin çözüm grafiği verilmektedir.



Şekil 5.10. GSP Optimizasyonunda Adaptif ve Sabit Mutasyon Oranlarının Etkisi

Mutasyon oranının zamana bağlı olarak kendi kendini düzenler (adaptif) bir şekilde hesaplanması sonucunda uygulanan oranlarının örnek bir gösterimi Şekil 5.11'de verilmiştir.



Şekil 5.11. Adaptif Mutasyon Oranının Zamana Bağlı Değişimi

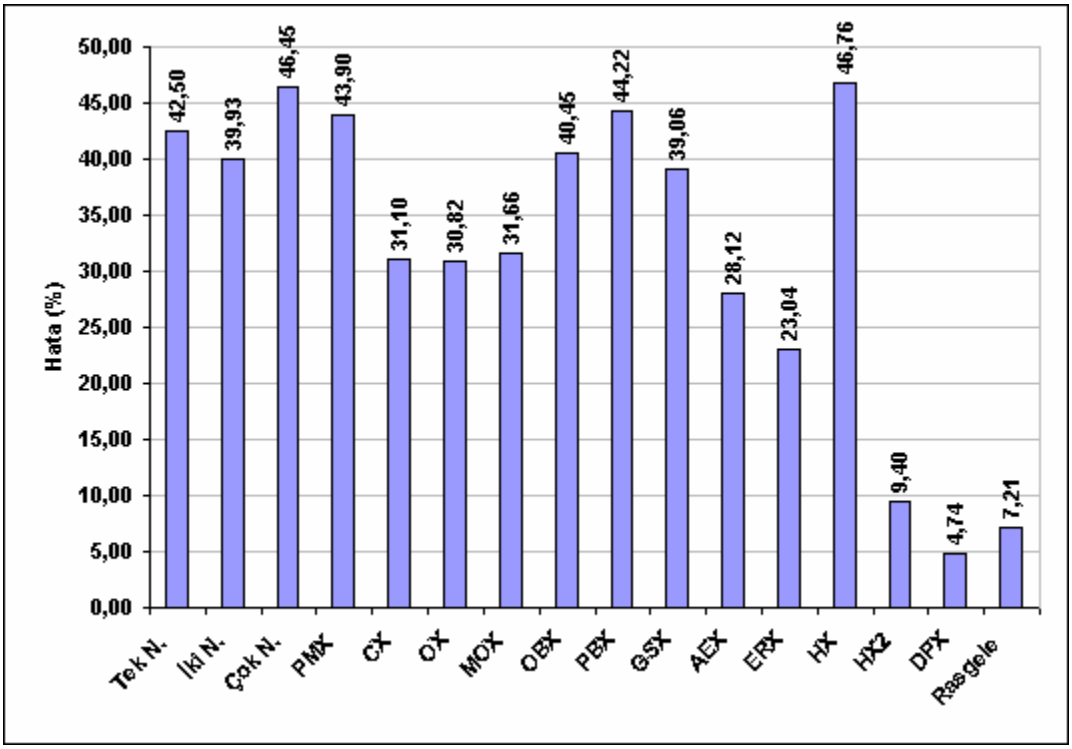
5.8. Çaprazlama ve Mutasyon Operatörlerinin Etkisi

GA'da kullanılan birçok çaprazlama ve mutasyon operatörü bulunmaktadır. Geliştirilen uygulama yazılımında 15 çaprazlama, 11 mutasyon operatörü ve her iki operatör için de birer karışık uygulama düzeni kullanılmaktadır. Bu operatörlerden her birisinin GSP problemi çözümündeki etkisi diğerinden farklıdır. Aynı zamanda operatörlerin kombinasyonlarının da etkilerinin çözüm üzerindeki performansları farklılık göstermektedir. Bu yüzden Ek-Tablo 7, Ek-Tablo 8 ve Ek-Tablo 9'da operatörlerin kombinasyonlarının 3 farklı GSP problemlerinde verdikleri ortalama hata değerleri analiz edilmiş ve en efektif çaprazlama operatörü, mutasyon operatörü ve çaprazlama-mutasyon kombinasyonu belirlenmeye çalışılmıştır.

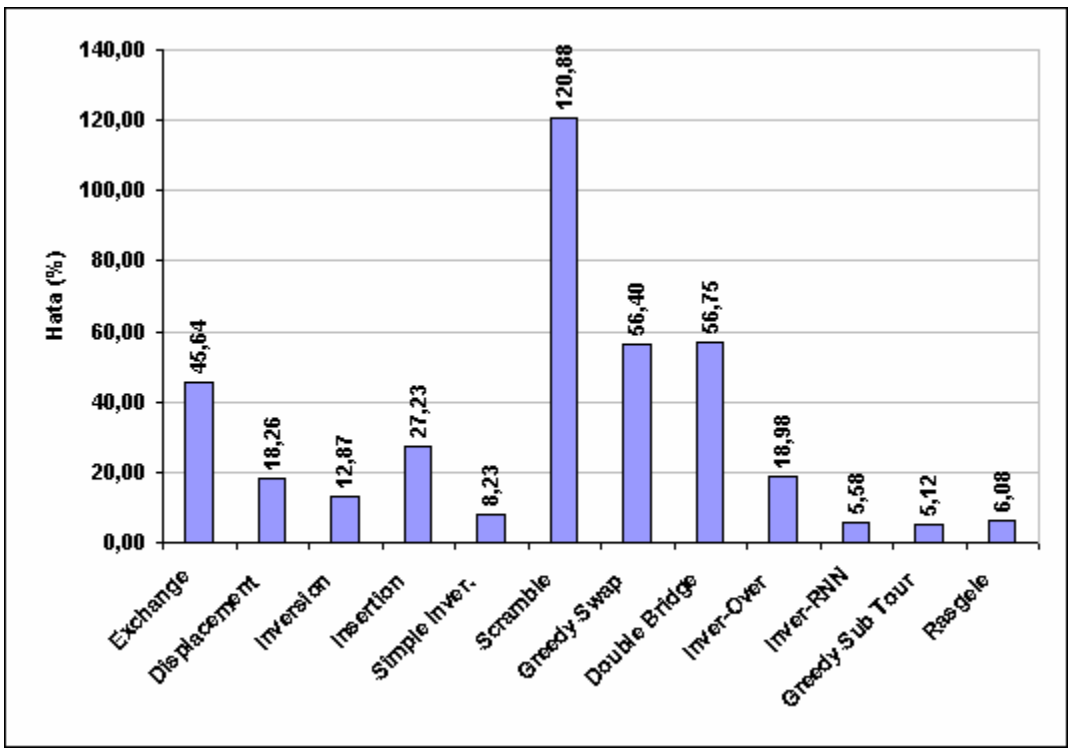
Ek-Tablo 7, Ek-Tablo 8 ve Ek-Tablo 9'daki yapılan analizlerde popülasyon büyüklüğü 40, $P_Ç$=%50, P_M=%75 ve sonlandırma kriteri olarak ta en iyi bireyin 3000 jenerasyon değişmemesi seçilmiştir. Ayrıca her bir operatör çifti için 10'ar deneme yapılarak ortalama hata değerleri hesaplanmıştır. Buna göre reel olarak en etkili çaprazlama operatörü (Şekil 5.12) ortalamada ε_{ORT}=%4.74 ile DPX (*Distance Preserving Crossover*), en etkili mutasyon operatörü de (Şekil 5.13) ortalamada ε_{ORT}=%5.12 ile bu çalışmada geliştirilen GST (*Greedy Sub Tour Mutation*) operatörüdür. Analizlerde HX çaprazlama operatörünün ve SCM (*Scramble*) mutasyon operatörünün ortalamada en kötü sonuçları verdiği görülmüştür.

Operatörlerin etkileşimleri sonucunda Berlin52 probleminde ε_{ORT}=%1.14 hata ile DPX+DB, Eil76 probleminde ε_{ORT}=%0.76 hata ile DPX+EXC, KroB200 probleminde ise ε_{ORT}=%3.54 ile DPX+SCM olarak görülmüştür.

Ayrıca, GSP'nin GA ile optimizasyonu süresince hem çaprazlama, hem de mutasyon operatörlerinin rasgele bir düzende uygulanmasının oldukça iyi sonuçlar verdiği görülmüştür. Çaprazlama operatörlerinin rasgele uygulanması ortalamada ε_{ORT}=%7.21 hata oranı ile ikinci sırada, mutasyon operatörlerinin rasgele uygulanması da ε_{ORT}=%6.08 hata oranı ile üçüncü sırada yer almaktadır.



Şekil 5.12. Çaprazlama Operatörlerinin Ortalama Hata Değerleri



Şekil 5.13. Mutasyon Operatörlerinin Ortalama Hata Değerleri

5.9. Seçim ve Eşleştirme Metotlarının Etkisi

Genetik Algoritmalarda, popülasyondan çeşitli metotlar ile seçilen bireyler bir eşleşme havuzuna alınır ve bu havuz içerisinde yer alan bireyler de yine farklı metotlarla eşleştirilerek yeniden üretme işlemine maruz kalıp yeni nesillerin üretilmesi gerçekleştirilir. Optimal çözüm için seçim ve eşleştirme prosedürlerinin her ikisinin de önemli etkisi olduğu aşikârdır. Fakat buradaki asıl önemli olan unsur bu iki prosedürün birbirine bağımlılığından dolayı aralarındaki etkileşimdir. Bu yüzden seçim ve eşleştirme metotlarının tek başlarına alınarak yapılacak olan analizlerin kıyaslanması somut bir değerlendirme olmayacaktır. Kullanılan tüm seçim ve eşleştirme metotlarının kombinasyonları Ek-Tablo 10'da verilen Berlin52, KroA150 ve Ek-Tablo 11'de verilen Ts225, Pcb443 GSP problemleri üzerinde analiz edilmiştir. Her iki tabloda yer alan analizler için popülasyon rasgele üretilmiş ve büyüklüğü 40'dır. Ayrıca çaprazlama OX ($P_Ç$=%50) ve mutasyon GST (P_M=%85) operatörleriyle yapılmaktadır. Sonlandırma ise popülasyonda yer alan en iyi bireyin 3000 jenerasyon değişmemesine bağlıdır.

Ek-Tablo 10'da Berlin52 ve KroA150 problemlerinin her ikisinde de Rasgele eşleştirme metodu diğer eşleştirme metotlarından daha iyi sonuç vermekte olup ortalamada sırasıyla ε_{ORT}=%3.714 ve ε_{ORT}=%5.293 hata oranına sahiptir. Ek-Tablo 11'de Ts225 probleminde En İyi İle En Kötü (ε_{ORT}=%7.053), Pcb443 probleminde ise İmparator Seçilimi (ε_{ORT}=%8.344) en iyi eşleştirme metodu olarak gözükmektedir. Aynı şekilde Berlin52 probleminde (ε_{ORT}=%3.982) Rasgele, KroA150 ve Pcb443 probleminde (ε_{ORT}=%5.182 ve ε_{ORT}=%8.307) Rulet Tekerleği ve Ts225 probleminde (ε_{ORT}=%7.244) Turnuva seçim metotları ortalamada en optimal sonuçları üretmişlerdir. Ek-Tablo 10 ve Ek-Tablo 11'deki 4 farklı GSP problemi için yapılan seçim-eşleştirme kombinasyonların analizlerin ortalamaları alındığında Tablo 5.3'deki veriler elde edilmektedir.

Tablo 5.3'deki ortalamalar baz alınarak genelleştirme yapılırsa, reel olarak seçim metodu dikkate alındığında Rulet Tekerleği (ε_{ORT}=%6.216), eşleştirme metodu dikkate alındığında ise Rasgele (ε_{ORT}=%6.472) ortalamada en optimal çözümler

üretmektedir. Fakat seçim-eşleştirme kombinasyonlarında en ideal metot çifti Rasgele-Rasgele (ε_{ORT}=%5.729) olarak gözükmektedir.

Tablo 5.3. Seçim-Eşleştirme Kombinasyonu Hata ve Süre Ortalamaları

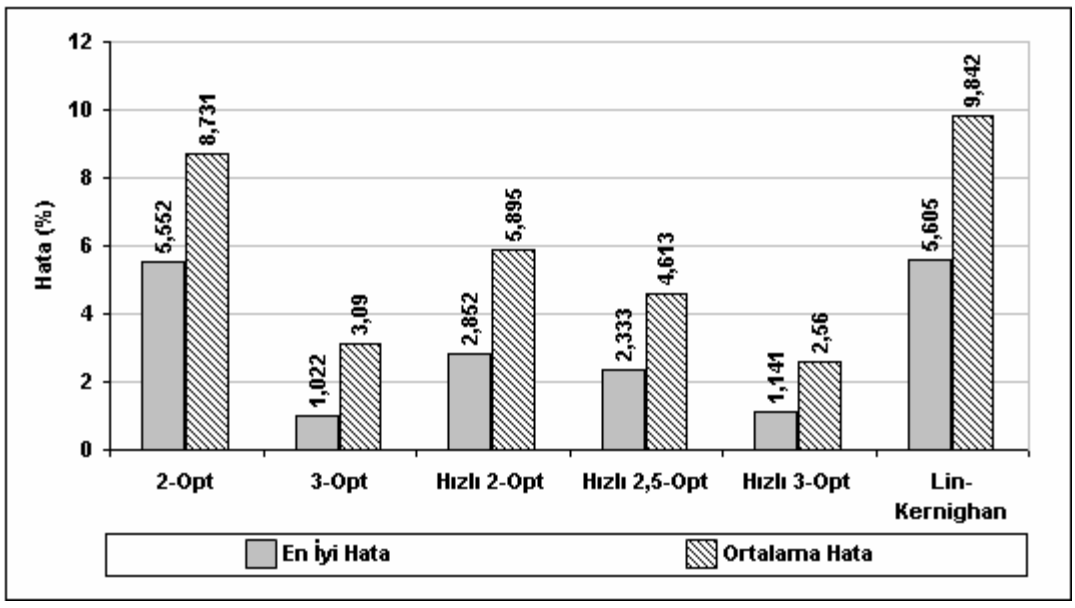
Problemler		**Berlin52 + KroA150 + Ts225 + Pcb443**						
Eşleştirme Metodu \ Seçim Metodu		Rasgele	Rulet Tekerleği	Turnuva	Sıralama	Ağırlıklı	Karışık	Ortalamalar
Rasgele	ε_{ORT}(%)	**5.729**	5.843	6.639	6.453	7.705	6.465	**6.472**
	$t_{ORT}(ms)$	16308	12869	11376	10551	8353	9868	11554
Yukarıdan Aşağıya	ε_{ORT}(%)	6.717	6.502	6.711	7.244	7.516	6.457	6.858
	$t_{ORT}(ms)$	21541	15131	10563	10025	9184	9584	12672
Sıra Ağırlıklı	ε_{ORT}(%)	6.411	6.521	6.383	6.974	7.398	6.454	6.690
	$t_{ORT}(ms)$	11607	10662	10615	11561	9580	8843	10478
Değer Ağırlıklı	ε_{ORT}(%)	6.327	6.143	6.570	6.291	7.628	6.302	6.543
	$t_{ORT}(ms)$	17433	13311	10740	10166	8591	10271	11752
En İyi İle En Kötü	ε_{ORT}(%)	5.875	5.917	6.985	6.677	7.632	6.750	6.639
	$t_{ORT}(ms)$	14352	13051	10769	10001	8671	10261	11184
İmparator Seçilimi	ε_{ORT}(%)	6.488	6.382	6.655	7.540	6.572	6.451	6.681
	$t_{ORT}(ms)$	15545	12705	11069	9987	8098	9796	11200
Karışık	ε_{ORT}(%)	6.980	6.201	6.796	6.976	6.831	6.805	6.765
	$t_{ORT}(ms)$	14952	13340	10185	10323	8156	8490	10907
Ortalama	ε_{ORT}(%)	6.361	**6.216**	6.677	6.879	7.326	6.526	
	$t_{ORT}(ms)$	15962	13010	10759	10374	8662	9587	

Diğer taraftan Tablo 5.3'deki ortalama sürelere dikkat edildiğinde ise Rasgele seçim metodunun çözüm süresini arttırdığı görülmektedir. Bunun sebebi havuz için seçilen bireylerin her zaman popülasyondaki en iyi bireyler olmamasından, kötü bireyler ile dolu havuzlar oluşturabilmesinden kaynaklanmaktadır.

5.10. Yerel Arama Metotlarının Etkisi

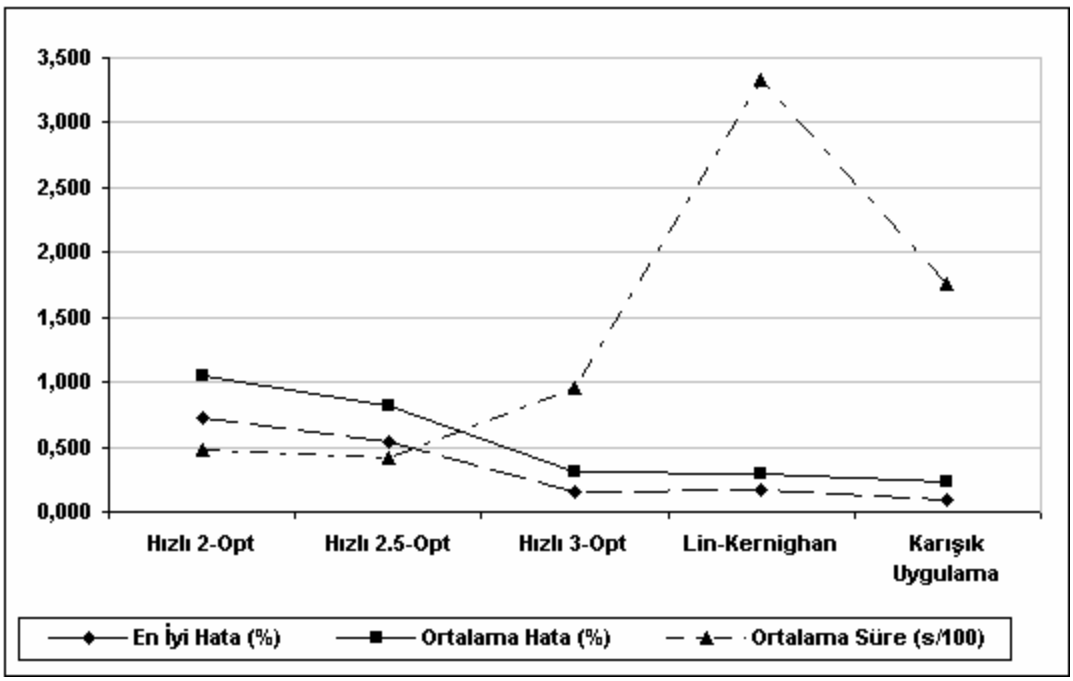
Yerel Arama metotları en ideal GSP çözümü geliştiren yöntemlerinden birisidir. Hazırlanan yazılımında Genetik Algoritmalar ile Tur Geliştirme Metotları (Yerel Arama Metotları) kombine edilebildiği için analizlerin Genetik Yerel Arama ve Yerel Arama adı altında ayrı ayrı yapılması daha doğru olacaktır.

Ek-Tablo 12'de yer alan analizde Yerel Arama metotları için 50 deneme yapılmış ve standart 2-Opt ve 3-Opt metotlarının oldukça fazla çevrim süresi talep ettiği görülmüştür (uygulanan metotlar tekrarlamalı olarak uygulanmamıştır). Aday Listeleri, Değişken Yarıçaplı Komşu Arama ve Bitlere Bakmama hızlandırma teknikleri ile uygulanan 2-Opt, 2.5-Opt ve 3-Opt metotlarının çevrim sürelerinin ve hata değerlerinin önemli ölçüde düştüğü görülmüştür. Ek-Tablo 12'de yer alan analizlerde aday listesi CL_{QN} (*Quadrant Neighbors*) ve 50 büyüklüğünde oluşturulmuştur. Buna göre optimuma en yakın sonuçların hızlandırılmış 3-Opt (ε_{ORT}=%2.560) metodu ile elde edildiği görülmüştür. Optimal değerden en uzak sonuçları Lin-Kernighan (ε_{ORT}=%9.842) vermektedir (Şekil 5.14).



Şekil 5.14. Yerel Arama Metotlarının Hata Değerleri

Ek-Tablo 13'de ise Yerel Arama metotları ile Genetik Algoritmalar kombine edilerek Genetik Yerel Arama analizleri yapılmıştır. Yerel arama metotlarının GA ile tekrarlamalı (iteratif) bir şekilde uygulanması ile optimal çözümlere oldukça yaklaşılmaktadır ve tek bir iterasyonda ideal gibi gözükmeyen metotlar (Ek-Tablo 12) daha uygun sonuçlar vermektedir. Ek-Tablo 13'deki analizlerde Standart 2-Opt ve 3-Opt metotları fazla çevrim süresi gerektirdiği için gruba dâhil edilmemiştir. Analizlerde komşu listesi $CL_{QN\text{-}MAX}$=20, popülasyon 40 büyüklüğünde (Rasgele), çaprazlama OX ($P_Ç$=%50), mutasyon SIM (P_M=%75) ve sonlandırma kriteri olarak da en iyi bireyin 1000 jenerasyon değişmemesi seçilmiş ve 10'ar deneme yapılmıştır.



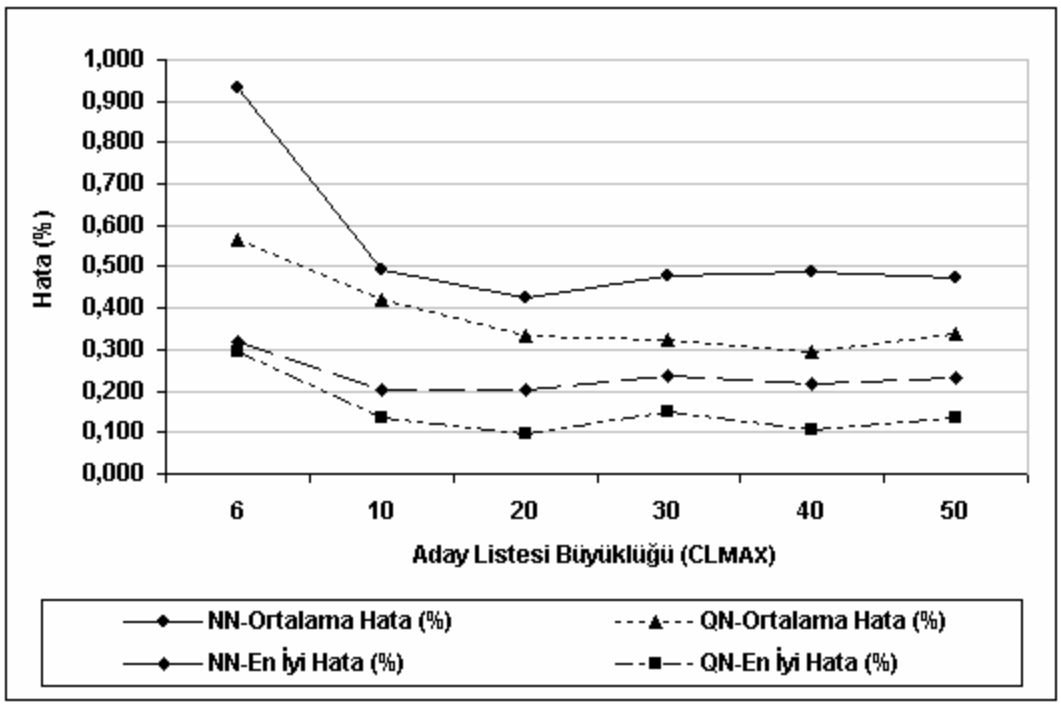
Şekil 5.15. Genetik Yerel Arama Hata ve Süre Ortalamaları

Ek-Tablo 13'deki analiz verileri ışığında ortalama hatada en güçlü metodun Lin-Kernighan (LK) ($\varepsilon_{İYİ}$=%0.168, ε_{ORT}=%0.296) olduğu görülmüştür. Fakat LK çevrim süresi bakımından en yavaş metottur. Ayrıca tüm metotların her bir çevrimde rasgele düzende uygulanması ($\varepsilon_{İYİ}$=%0.095, ε_{ORT}=%0.225) özerk bir metot olmamasına rağmen LK'dan hem en iyi hata ($\varepsilon_{İYİ}$), hem ortalama hata (ε_{ORT}) hem de ortalama sürede (t_{ORT}) daha iyi sonuçlar vermiştir (Şekil 5.15).

Yerel arama metotlarının karışık bir şekilde uygulanması ile özellikle LK'nın ek süre dezavantajı (100 * (t_{LK} - $t_{KARIŞIK}$) / t_{LK}) yaklaşık %47.378 oranında azaltılarak araştırma süreci biraz daha hızlandırılabilir. Ayrıca karışık uygulama ile LK'nın ortalama hata oranında (ε_{ORT}) yaklaşık %24.035, en iyi hata değerinde ($\varepsilon_{İYİ\text{-}ORT}$) ise yaklaşık %43.827'lik bir iyileştirme etkisi oluşmaktadır.

5.11. Aday Listesi Metotları ve Büyüklüklerinin Etkisi

Yerel Arama metotlarının oldukça fazla çevrim süresi gerektirmesinden ötürü (2Opt = $\Theta(n^2)$, 3Opt = $\Theta(n^3)$) çeşitli hızlandırma teknikleri kullanılmaktadır ki bunlardan birisi de Aday Listeleri'dir. Aday listelerinin oluşturulması için kullanılan *Nearest Neighbors* (CL_{NN}) ve *Quadrant Neighbors* (CL_{QN}) metotlarının ve oluşturulan liste büyüklüklerinin optimal çözümlerdeki etkisi Ek-Tablo 14 ve Ek-Tablo 15'de görüldüğü gibi analiz edilmiştir. Analizlerde popülasyon büyüklüğü 40, çaprazlama operatörü OX ($P_Ç$=%50), mutasyon operatörü SIM (P_M=%75), yerel arama metodu LK ve sonlandırma operatörü olarak popülasyondaki en iyi bireyin 500 jenerasyon değişmemesi seçilmiştir.



Şekil 5.16. Aday Listesi Metotlarının Hata Değerleri

Bu analizler ışığında CL_{QN} aday listelerinin en iyi çözüm ve ortalama çözümde daha düşük hata oranına sahip olduğu görülmüştür. Her iki aday listesi için de yapılan analizlerdeki ortalama hata değerlerinin ortalamasına bakıldığında (ε_{QN_ORT}=%0.379 ve ε_{NN_ORT}=%0.549) CL_{QN}'nin yaklaşık olarak ortalama hatada CL_{NN}'den %30.965 daha iyi sonuçlar ürettiği görülmektedir (Şekil 5.16).

Ek-Tablo 14 ve Ek-Tablo 15'deki ortalama hata değerleri aday listelerinin büyüklükleri arttırılarak düşürülebilir. Öte yandan aday listesinin büyüklüğünü arttırmak arama uzayını genişletmek anlamına geldiği için çevrim süresinin artmasına neden olacaktır. Artan süre özellikle büyük GSP problemlerinde önemli ölçüde belli olmaktadır. Bu yüzden kabul edilebilir bir aday listesi büyüklüğü tercih edilmelidir. Fakat tercih edilecek aday listesi büyüklüğü problemden probleme farklılık göstermektedir. Çok fazla sayıda şehir içeren bir liste çevrim süresini arttırmakta, düşük büyüklüktü şehir içeren bir liste de hata değerini yükseltmektedir. Aynı zamanda büyük seçilen listeler optimal değerden uzaklaşmalara da neden olmaktadır. Bu yüzden aday listelerinin büyüklüğünün çalışma esnasında *rasgele* ve *adaptif* olarak uygulanabilirliği 3-OPT metodu (popülasyon sayısı 40, çaprazlama operatörü OX ($P_Ç$=%50), mutasyon operatörü SIM (P_M=%75), $CL_{QN\text{-}MAX}$=30 ve sonlandırma koşulu olarak en iyi bireyin 2000 jenerasyon değişmemesi) ile Ek-Tablo 16'daki gibi denenmiştir. Çalışma süresince rasgele büyüklükte uygulanan aday listesinin hem en iyi hata oranında, hem ortalama hata oranında hem de düşük çevrim süresi ile sabit büyüklükteki listeden daha iyi olduğu görülmüştür. Öte yandan adaptif olarak aday listesinin büyüklüğünün ayarlanması hem ortalama hata değerini hem de çevrim süresini önemli ölçüde azaltmaktadır.

Sonuç olarak Ek-Tablo 14, Ek-Tablo 15 ve Ek-Tablo 16'daki yer alan analiz sonuçlarına göre CL_{QN} için en ideal sonuçlar 20 ve 40'da, CL_{NN}'de ise 20-30 arasında elde edilmiştir. Ayrıca aday listesi büyüklüğünün araştırma süresince rasgele bir düzende uygulanması ile sabit bir liste büyüklüğünden zaman olarak yaklaşık %54.147, en iyi bulunan hata değerinden yaklaşık %73.833 ve ortalama hatada ise yaklaşık %0.237 kazanç sağlanmaktadır.

5.12. Genetik Algoritma ve Tur Bölme Metodu İle Optimizasyon

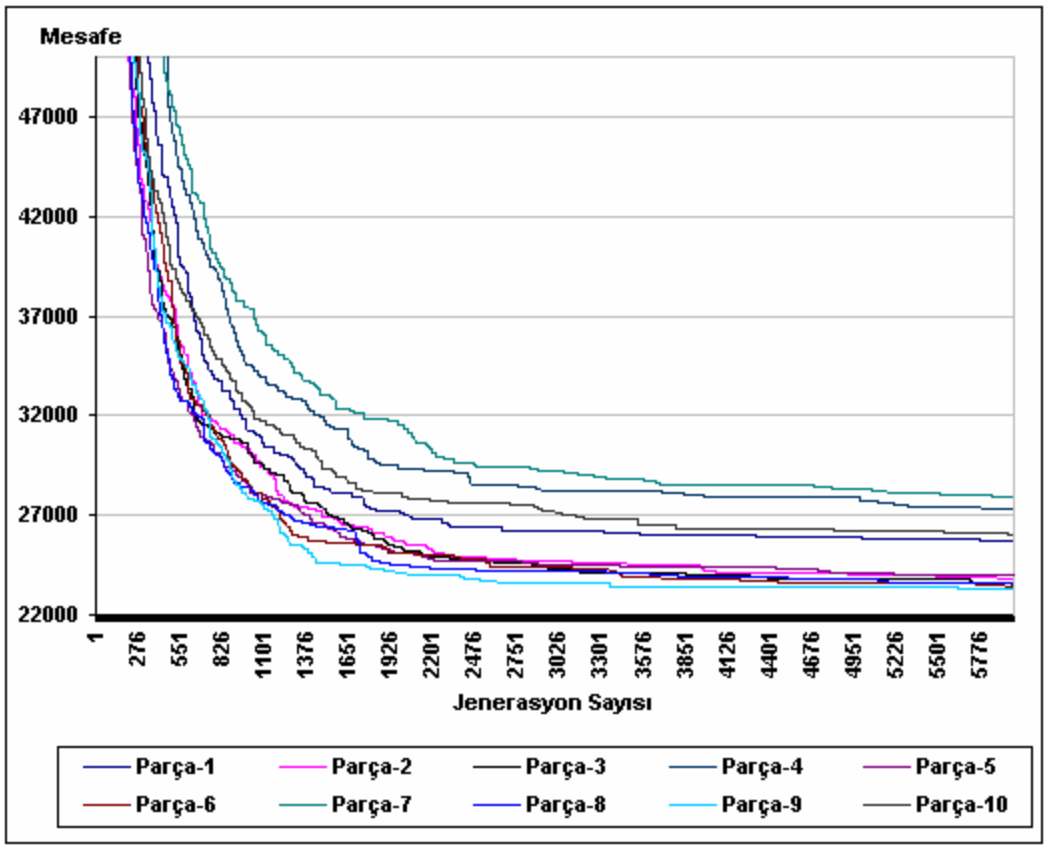
GSP problemlerinde delik sayısı arttıkça BGA'ların optimum çözümü elde etmesi oldukça zorlaşmaktadır. BGA'lar büyük GSP problemlerinde optimum çözüme yaklaşsa dahi bu süreç oldukça uzun zaman almaktadır. Bu yüzden GSP probleminin kullanıcının belirlediği oranda parçalara ayrılarak, her parçanın ayrı ayrı BGA ile optimize edilmesi hem harcanan süreyi azaltmakta hem de optimuma daha yakın değerler elde edilebilmesini sağlamaktadır. Geliştirilen yazılımda iki farklı bölme tekniği kullanılmıştır. Her iki teknikte Bölüm 4.6'da anlatılmıştır.

İlk etapta kullanılan her iki bölme metodu için ideal bölme sabitleri (β) belirlenmesi işlemi yapılmıştır. Tur bölme tekniklerinin analizleri için GSP problemlerinden U2319 seçilmiştir. Popülasyon sayısı olarak 10, çaprazlama operatörü olarak OX ($P_Ç$=%60), mutasyon operatörü olarak da GST (P_M=%85) kullanılmıştır. Araştırılan alt turların sonlandırma kriteri en iyi bireyin 3000 jenerasyon sabit kalması olarak seçilmiştir. Analiz sonuçları Tablo 5.4'deki gibidir.

Tablo 5.4. U2319 Probleminde Tur Bölme Katsayısı Analizi

Bölme Sabiti	β Oranında Bölme			β^2 Oranında Bölme		
	BGA Hata (%ε)	BGA+3Opt Hata (%ε)	Toplam Süre (*ms*)	BGA Hata (%ε)	BGA+3Opt Hata (%ε)	Toplam Süre (*ms*)
β=2	-	-	-	**5.8393**	3.3156	195594
β=3	4.7465	3.0735	102781	10.1769	2.9655	161328
β=4	4.5100	2.9429	92078	14.0722	2.3662	97406
β=5	4.9821	2.8135	75047	16.8123	2.8477	104859
β=6	4.8131	3.2161	69609	15.8271	2.7683	85734
β=7	4.7115	2.8490	**54859**	20.7653	2.4635	85202
β=8	4.4570	**2.5002**	67563	19.2844	2.6283	80765
β=9	4.9262	2.5134	58781	25.3419	2.4289	78140
β=10	**4.3811**	2.7162	57844	25.8665	2.4413	94343
β=15	4.7853	3.0996	63172	33.5440	**2.0430**	**69203**
Ortalama	**4.7014**	2.8583	**71304**	18.7530	**2.6268**	105257

Yapılan analizlerde BGA için β Oranında Bölme metodunun hem çevrim süresi hem de hata değeri olarak daha iyi sonuçlar verdiği görülmektedir. β değeri de en ideal 10 olarak gözükmektedir. Fakat bölme sabitinin (β) problem büyüklüğüne göre değişmesi gerekliliği aşikârdır. U2319 GSP problemi β=10 olarak bölünmesi sonucunda elde edilen 10 alt problemin BGA ile araştırılması Şekil 5.17'de gösterilmektedir.



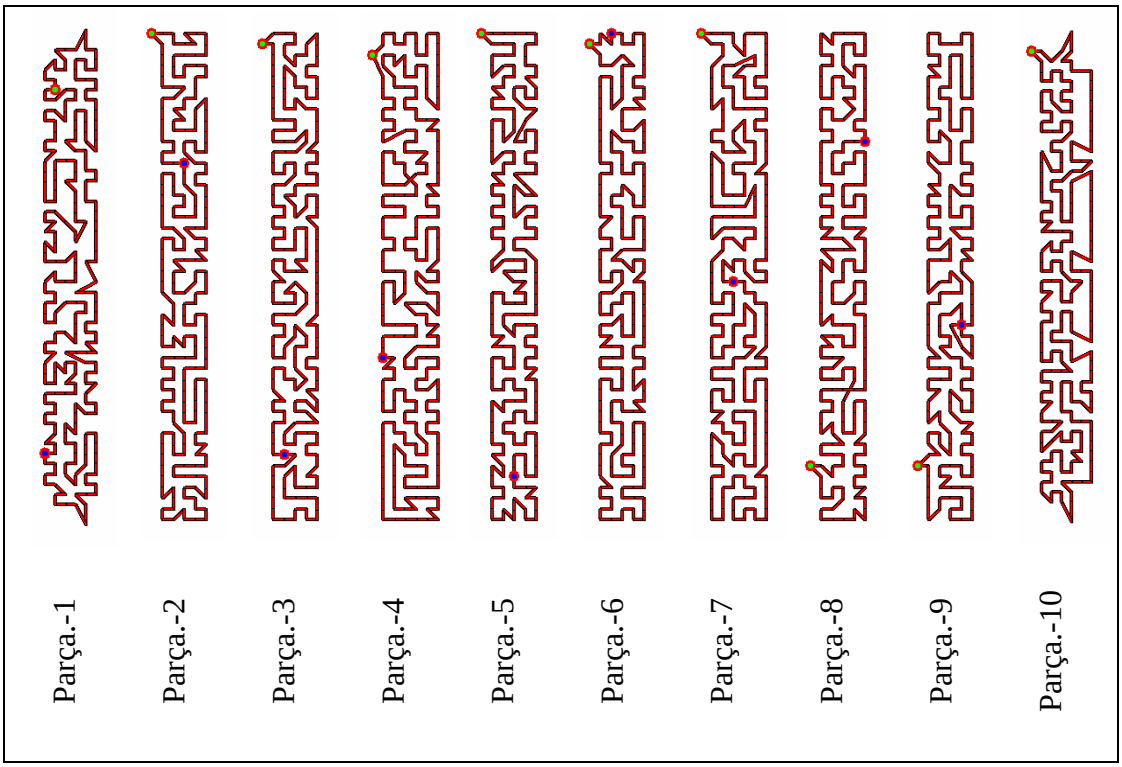
Şekil 5.17. Alt Problemlerin BGA ile Araştırılması

U2319 için β=10 olarak yapılan bölme işleminde, alt problemler BGA ile araştırılırken sonlandırma kriteri olarak popülasyonun 3000 jenerasyon değişmemesi, OX, GST operatörleri ve popülasyon sayısı 10 (*Random*) olarak alınmıştır. Alt turların birleştirilmesi sonucunda ε=%5.2882 hata oranı 39468 *ms* sürede elde edilmiştir.

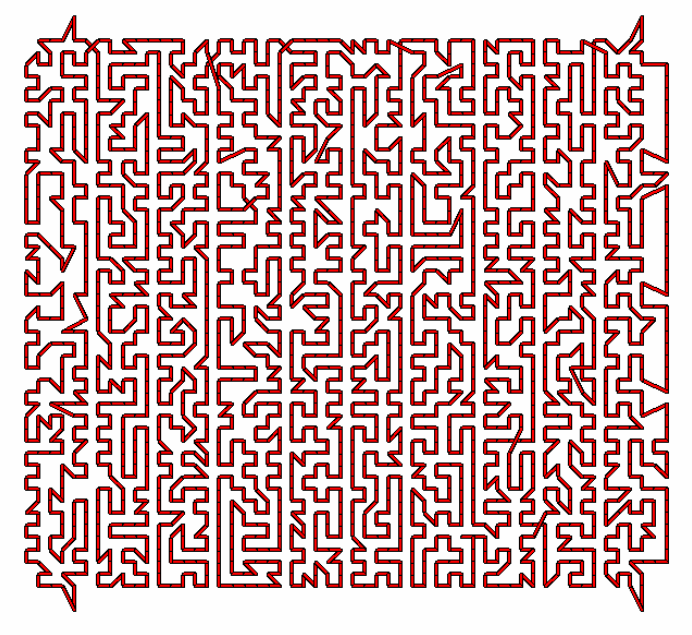
Tur bölme metodunun BGA ile kıyaslanabilmesi için 2 farklı deneme yapılması gereklidir. Birincisi tur bölmedeki hata oranının BGA ile kaç *ms*'de yakalanacağı, ikincisi de tur bölme için harcanan sürede BGA'nın vereceği yüzdelik hata oranının tespitidir. Bu analizler yapıldığında BGA %5.2882'lik hata oranını 238625 *ms*'de, süreler eşit tutulduğunda ise BGA %66.06 hata vermektedir.

Sonuç olarak tur bölme metodu BGA'ların performansını oldukça önemli bir ölçüde arttırmaktadır. Bu değer sadece U2319 problemindeki analizler baz alınarak bir katsayı haline getirilirse, BGA'nın tur bölme metotlarıyla birleştirilmesi BGA'ya yaklaşık 6-12 kat arasında performans artışı sağlamaktadır.

Şekil 5.18'de U2319 probleminin β=10 ile bölünmesi ve BGA ile optimizasyonu sonucu oluşan alt turlar ve Şekil 5.19'de sonuç turları verilmektedir.

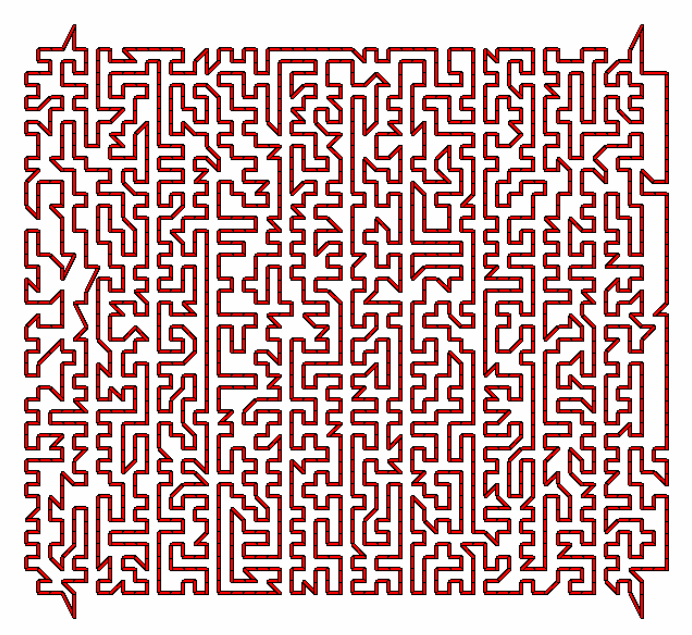


Şekil 5.18. Tur Bölme ile Oluşan Alt Turlar (β=10)



Sonuç Turu – 1

Alt Turların Açgözlü Birleştirilmesi



Sonuç Turu – 2

Birleştirilen Tura 3-Opt Uygulanması

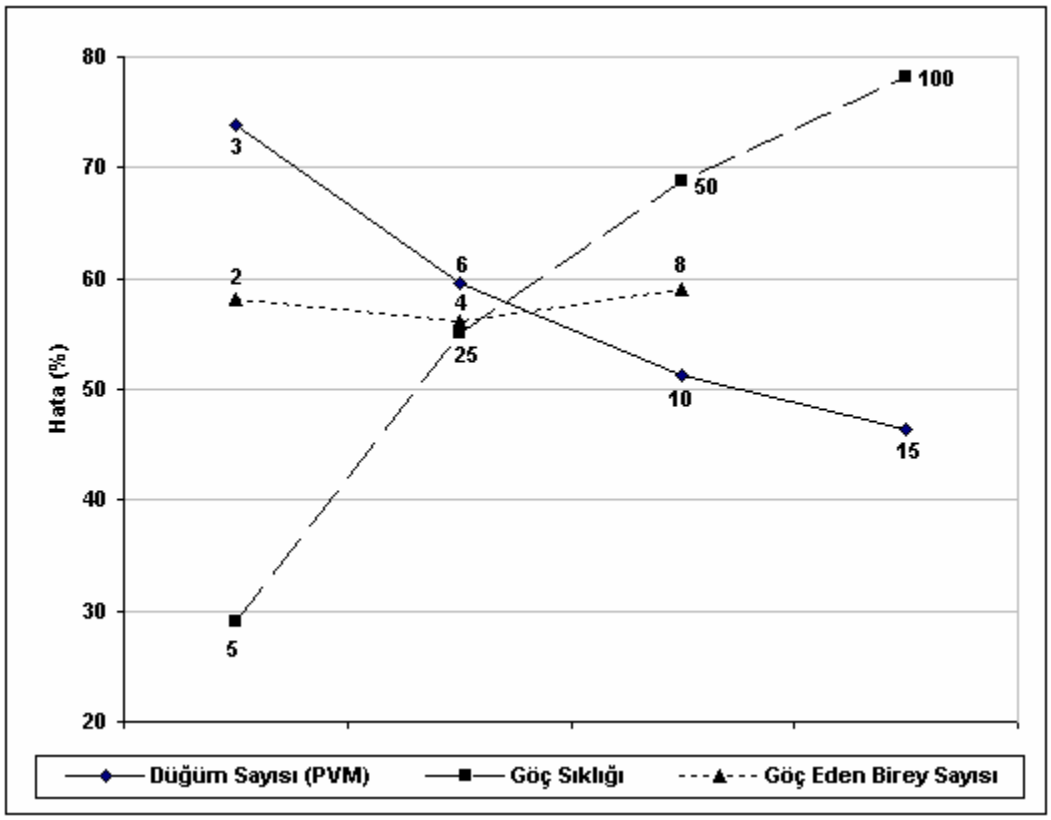
Şekil 5.19. U2319 için Tur Bölme ile Oluşan Sonuç Turları

5.13. Paralel Genetik Algoritmanın Etkisi ve Parametrelerin Belirlemesi

Geliştirilen yazılımda test amaçlı sanal PGA benzetişimi uygulanmıştır. PGA, Paralel Sanal Makinelerde (*Parallel Virtual Machines*), İnce Taneli, Senkron ve Homojen yapıdadır. PGA'da kullanıcı alt popülasyon sayısını, göç edecek en iyi birey sayısını (m) ve göç sıklığını (ξ) parametrik olarak belirlemektedir. Bu yüzden Ek-Tablo 17'de Pcb1173 problemi için sonuçları verilen analizde hem PGA performansı hem de PGA için ideal parametre değerleri belirlenmektedir. Ek-Tablo 17'deki analiz sürecinde her GA'daki alt popülasyon büyüklüğü 40 olarak seçilmiş, bireyler rasgele üretilmiş, çaprazlama operatörü olarak OX ($P_Ç$=%50), mutasyon operatörü olarak GST (P_M=%75) ve sonlandırma kriteri olarak 5000 jenerasyon kullanılmıştır. Buna göre PGA için en ideal parametre değerleri için sırasıyla aşağıdakiler söylenebilir.

- PGA'da düğüm sayısının artması ile optimum değere yaklaşma oranı artmaktadır. Geliştirilen yazılım ile maksimum 20 Paralel Sanal Makine (*Parallel Virtual Machine*: PVM) oturumu açılabilmektedir. Yapılan analizlerde düğüm sayısı olarak 3, 6, 10 ve 15 denemeleri yapılmış en ideal sonuçlar düğüm sayısının 15 olduğunda alınmıştır.
- PGA'da göç eden birey sayısının (m) arttırılması optimumdan uzaklaşmalara neden olmaktadır. İdealde bu değer m=2 olarak ölçülmüştür.
- PGA'da göç sıklığı (ξ) değerinin arttırılması optimumdan oldukça uzaklaşmalara neden olurken, bu değerin düşürülmesi ile optimum değerlere yaklaşılmaktadır. Fakat düşük göç sıklığı değerleri ile alt popülasyonlar arası haberleşme prosedürlerinin artmasından dolayı çevrim süresi de artmaktadır. Yapılan analizde göç sıklığının 5-25-50-100 değerleri denenmiş ve en iyi sonuçlar ξ =[1-5)'de elde edilmiştir.

Şekil 5.20'de yer alan grafikte düğüm sayısı, göç eden birey sayısı ve göç sıklığı değerlerinin değişimlerine göre elde edilen ortalama hata değerleri verilmektedir.



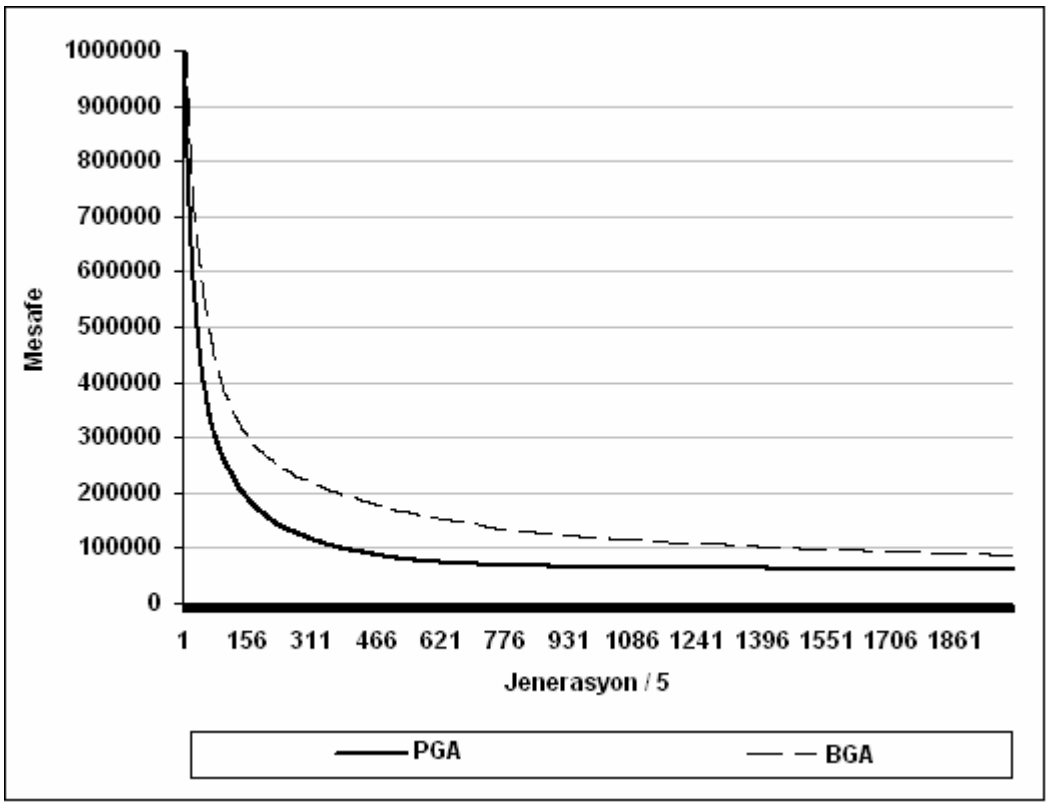
Şekil 5.20. PGA Parametre Değişimlerinin Ortalama Hata Oranları

Pcb1173 problemi BGA üzerinde aynı parametreler ile incelendiğinde (10 deneme) ortalamada ε_{ORT}=%111.386 hata oranını 12975 *ms* sürede vermektedir. Öte yandan hazırlanan PGA'da her parametre için BGA'dan iyi sonuçlar elde edilmiştir.

Aynı sonlandırma şartları (5000 jenerasyon) baz alındığında PGA ε=%15.413 hata değerine kadar inebilirken, BGA ε=%111.386 gibi çok büyük bir hata değeri vermektedir. Aynı sonlandırma şartı durumunda PGA, BGA'ya yaklaşık %86.159 hata performansı sağlamıştır.

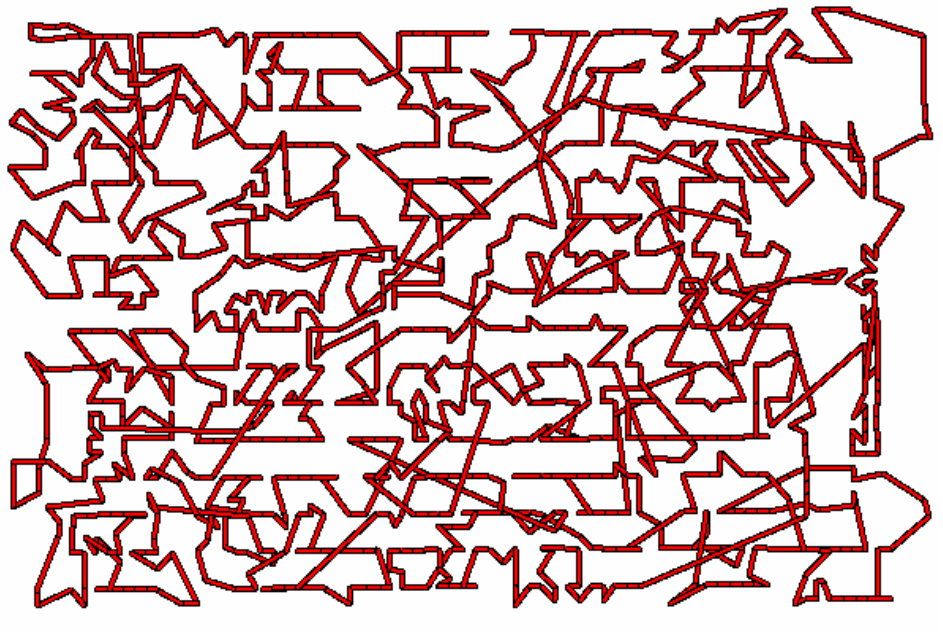
PGA'lar göç politikası süreçlerinin de devreye girmesinden dolayı BGA'ya göre birim zamanda daha yavaş çalışır. Yani PGA'nın *i* iterasyon yapması BGA'ya göre ortalamada daha uzun zaman alır ($t_{ORT_PGA} > t_{ORT_BGA}$). Bu yüzden PGA'nın en iyi değerlerinin elde edildiği (ε=%15.413 değeri 48923 *ms*) kriterler doğrultusunda BGA'nın sonlandırma koşulu 48923 ms yapıldığında BGA'dan ortalamada (10 deneme) ε_{ORT}=%26.107 ($Jenerasyon_{ORT}$=20465) hata alınmaktadır. Çalışma süreleri

veya sonlandırma jenerasyonları aynı seçilerek çalıştırılan PGA, BGA'ya göre çok daha iyi sonuçlar vermektedir. Şeçilen parametrelere göre, PGA %86-%26 arasında bir başarım sağlamaktadır. Bu performans farkı Şekil 5.21'deki gibi görülebilir.

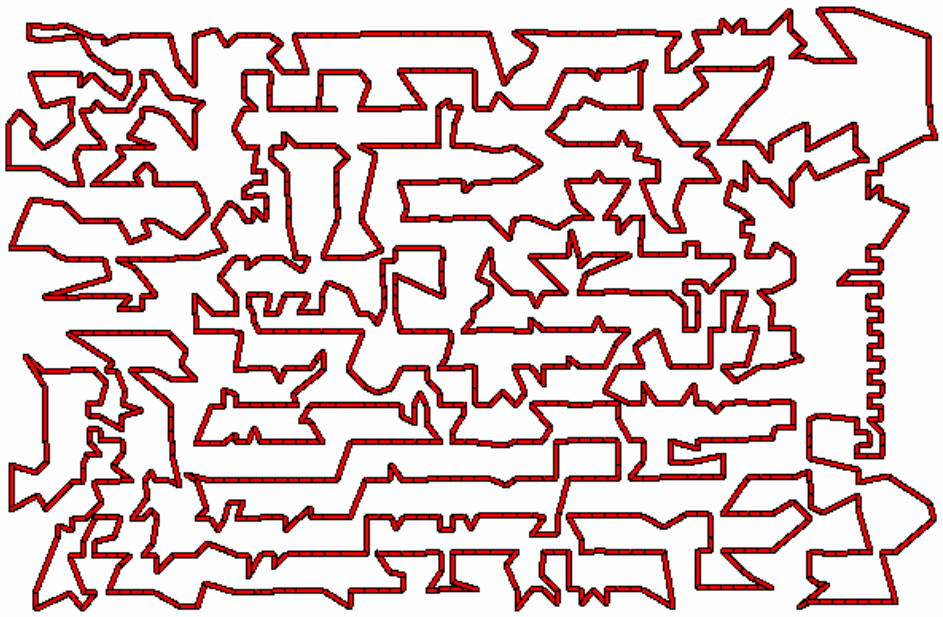


Şekil 5.21. PGA-BGA için Mesafe-Jenerasyon Grafiği

Şekil 5.22 ve 5.23'de yukarıda belirtilen parametreler doğrultusunda yapılan bir analiz sonucu (sonlandırma 10000 jenerasyon) PGA ve GA'dan elde edilen güzergahlar verilmiştir.



Şekil 5.22. Pcb1173 Problemi için GA ile Bulunan Güzergâh (ε=%53.708)



Şekil 5.23. Pcb1173 Problemi için PGA ile Bulunan Güzergâh (ε=%9.779)

6. SONUÇLAR VE ÖNERİLER

Optimizasyon algoritmalarından kısa sürede optimum çözüme ulaşabilmesi veya çok yakın sonuçlar üretilmesi istenmektedir. GSP için optimum sonuçlar üreten Kesin Çözüm Algoritmaları mevcutken, çalışma süreleri onlarca paralel bağlı süper bilgisayarlarda dahi kabul edilemez ölçüde büyüktür. Çünkü bu algoritmalar tüm problem uzayını tarayarak optimumu bulmaya çalışırlar. Bu yüzden optimum değere çok yakın değerleri kısa sürede elde etmek çoğu zaman yeterlidir.

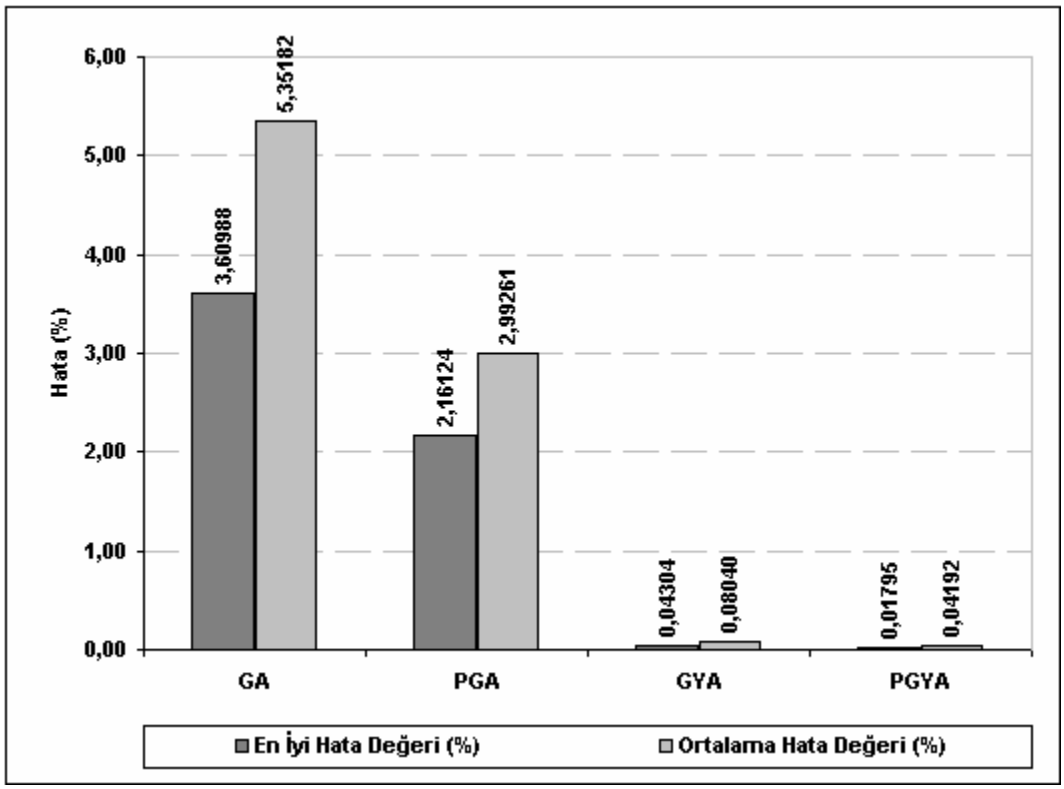
Tablo 6.1. Analizlerde Kullanılan Parametre Değerleri

<table>
<tr><th>Parametreler</th><th>GA</th><th>PGA</th><th>GYA</th><th>PGYA</th></tr>
<tr><td>Popülasyon Sayısı</td><td>40</td><td>40 *</td><td>40</td><td>40 *</td></tr>
<tr><td>Başlangıç Popülasyonu</td><td>DENN</td><td>DENN</td><td>DENN</td><td>DENN</td></tr>
<tr><td>Rasgelelik Oranı</td><td>%25</td><td>%25</td><td>%25</td><td>%25</td></tr>
<tr><td>Seçim Metodu</td><td>Rasgele</td><td>Rasgele</td><td>Rasgele</td><td>Rasgele</td></tr>
<tr><td>Eşleştirme Metodu</td><td>Rasgele</td><td>Rasgele</td><td>Rasgele</td><td>Rasgele</td></tr>
<tr><td>Çaprazlama Metodu</td><td colspan="4">DPX</td></tr>
<tr><td>Çaprazlama Oranı ($P_Ç$)</td><td colspan="4">Adaptif Strateji-1 (Max=70, Min=30, Rm1=20, Rm2=25)</td></tr>
<tr><td>Mutasyon Metodu</td><td>GST</td><td>GST</td><td>SIM + 5xDB</td><td>SIM + 5xDB</td></tr>
<tr><td>Mutasyon Oranı (P_M)</td><td colspan="2">Sabit – 100</td><td colspan="2">Sabit – 100</td></tr>
<tr><td>Sezgisel Süreçler</td><td colspan="4">En İyi Bireyin 100 Jenerasyon Değişmemesinde %85 DENN Ekle Popülasyonu %3 İhtimalinde Karıştır</td></tr>
<tr><td>Yerel Arama Metodu</td><td colspan="2">-</td><td colspan="2">Karışık</td></tr>
<tr><td>Yerel Arama Yeri</td><td colspan="2">-</td><td colspan="2">Çaprazlama & Mutasyon Sonu</td></tr>
<tr><td>Aday Listesi (CL)</td><td colspan="2">-</td><td colspan="2">NN</td></tr>
<tr><td>CL_{MAX} ve Uygulanışı</td><td colspan="2">- -</td><td colspan="2">60 ve Rasgele</td></tr>
<tr><td>Düğüm Sayısı</td><td>-</td><td>15</td><td>-</td><td>15</td></tr>
<tr><td>Göçen Birey Sayısı (m)</td><td>-</td><td>2</td><td>-</td><td>2</td></tr>
<tr><td>Göç Sıklığı (ξ)</td><td>-</td><td>25</td><td>-</td><td>25</td></tr>
<tr><td>Sonlandırma Kriteri</td><td>En İyi Bireyin 10000 Jenerasyon Sabit Kalması</td><td>En İyi Bireyin 1000 Jenerasyon Sabit Kalması</td><td>En İyi Bireyin 10000 Jenerasyon Sabit Kalması</td><td>En İyi Bireyin 1000 Jenerasyon Sabit Kalması</td></tr>
</table>

(*: Alt Popülasyon Sayısı)

Geliştirilen yazılımın, kullanıcının bütün parametrelere ve algoritmalara müdahale edebileceği bir şekilde (dinamik) hazırlanmasından dolayı sonuçlar 4 farklı algoritma üzerinde literatürde yer alan 63 farklı problem kullanılarak 10'ar deneme yapılmıştır. Bu analizlerde Tablo 5.1'de verilen Intel E6600 (Core 2 Duo-2.4Ghz) sistemi ve Tablo 6.1'deki parametreler kullanılmıştır.

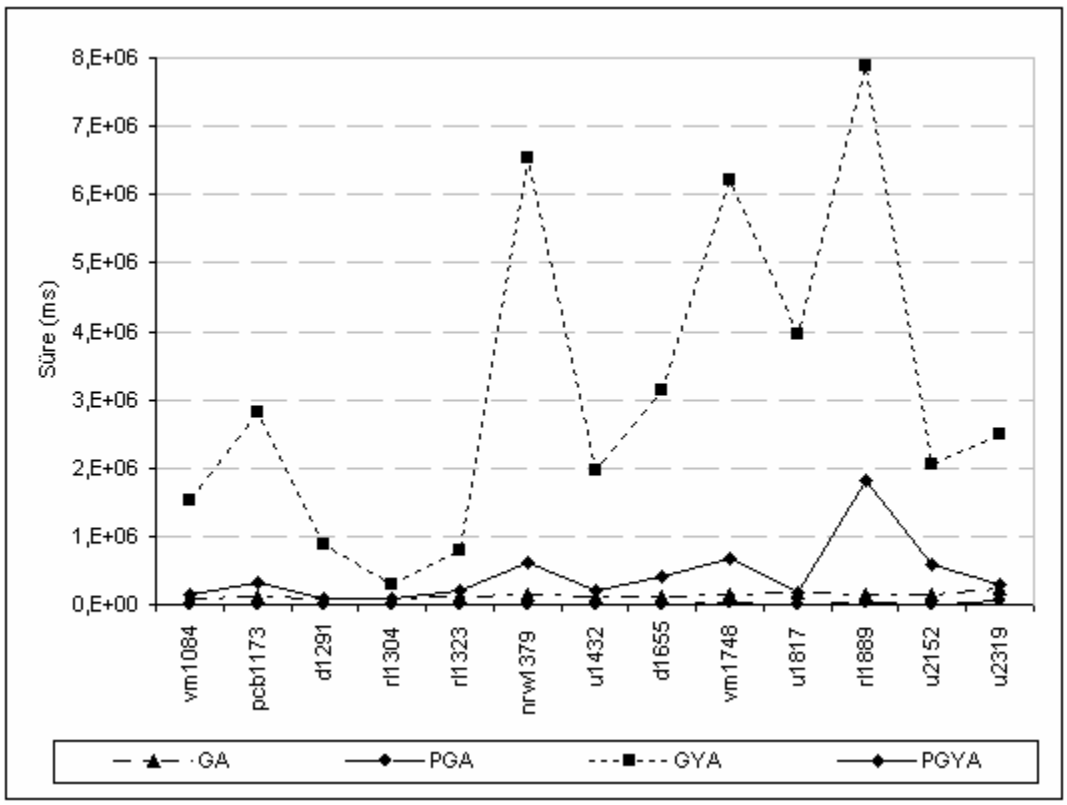
Ek-Tablo 18 ve Ek-Tablo 19'daki araştırmalarda Basit Genetik Algoritmalar (BGA), Ek-Tablo 20 ve Ek-Tablo 21'de Paralel Genetik Algoritmalar (PGA), Ek-Tablo 22 ve Ek-Tablo 23'de Genetik Yerel Arama (GYA), Ek-Tablo 24 ve Ek-Tablo 25'da ise Paralel Genetik Yerel Arama (PGYA) sonuçları verilmektedir. Bu tablolardaki en iyi bulunan ve ortalama hata değerleri baz alınarak algoritmaların başarımları Şekil 6.1'deki gibi özetlenebilir.



Şekil 6.1. BGA, PGA, GYA ve PGYA Hata Değerleri

Yapılan analizler sonucunda GYA ve PGYA ile çoğu problemde optimum değer, bazı büyük problemlerde optimuma oldukça yakın değerler elde edilmiştir.

Özellikle, geliştirilen PGYA modeli oldukça etkili değerler vermektedir. Analizlerde BGA, GSP için ideal bir algoritma olarak gözükmese de çeşitli metotlarla kombinasyonu sonucunda oluşan algoritmalar (PGA, GYA, PGYA) oldukça iyi sayılabilecek sonuçlar doğurmaktadır.



Şekil 6.2. Algoritmaların Çalışma Süreleri

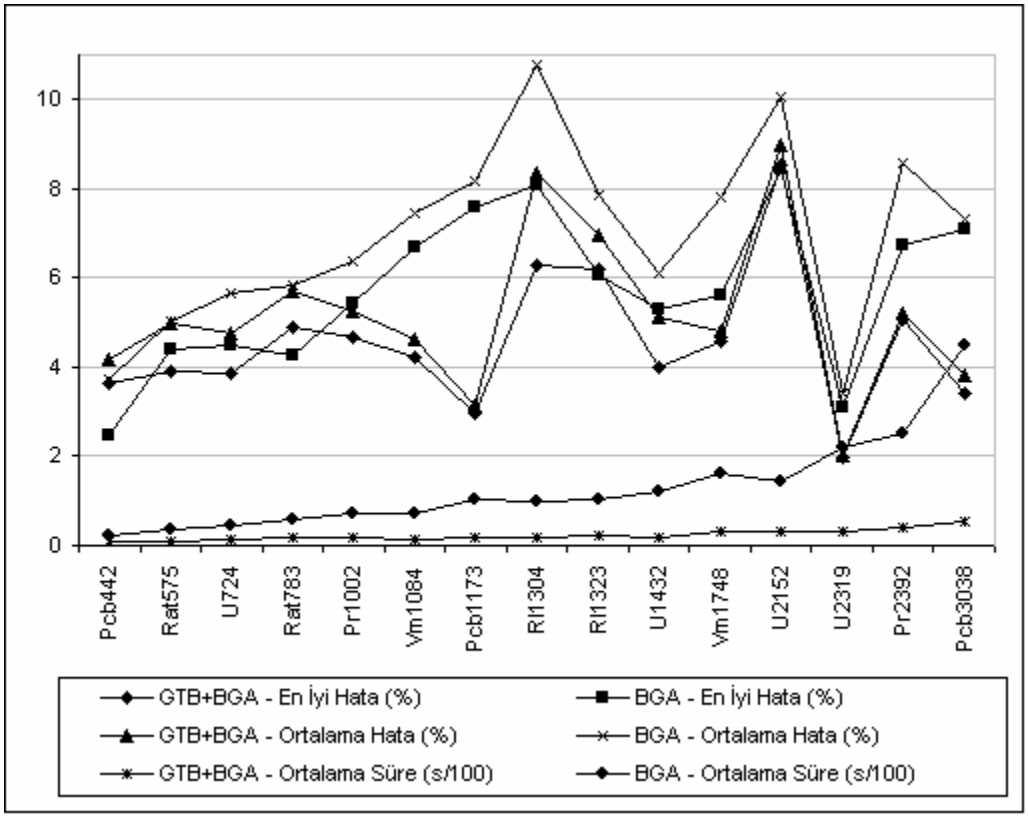
Hazırlanan yazılım ile yapılan analizlerde paralelizm, yazılımla tek bir bilgisayar üzerinde kurulan sanal bilgisayarlarda (düğüm) alt popülasyonların ardışık olarak çalıştırılmasıyla gerçekleştirilmektedir. Ek-Tablo 20, Ek-Tablo 21, Ek-Tablo 24 ve Ek-Tablo 25'deki "Gerçek Ortalama Süre (*ms*)", paralel araştırmaların tek bir bilgisayar üzerinde sanal bilgisayarlar oluşturulmasıyla elde edilen toplam çalışma sürelerdir. PGA ve PGYA tek bilgisayar üzerinde denendiğinde işlemcinin yükü oluşturulan düğüm sayısı kadar artmaktadır. Gerçek paralel bilgisayar ağında yapılacak olan optimizasyonda harcanacak süre, yaklaşık olarak tek bilgisayarda harcanan toplam sürenin düğüm sayısına (*paralel sanal makine*) oranına eşittir (bilgisayarlar arası göç işlemleri için harcanacak süreler göz ardı edildiğinde). Ek-

Tablo 20, Ek-Tablo 21, Ek-Tablo 24 ve Ek-Tablo 25'de "Yaklaşık Ortalama Süre (*ms*)" sütununda verilen süre değerleri paralel bilgisayar ağı kullanıldığında elde edilecek yaklaşık süreye karşılık gelmektedir (Yaklaşık Ortalama Süre (YOS) = Gerçek Ortalama Süre / Düğüm Sayısı). Algoritmaların çalışma süreleri incelendiğinde, PGYA'nın çalışma süresi (YOS) olarak da GYA'dan daha etkili olduğu görülmüştür (Şekil 6.2).

Sadece BGA üzerinde tekrarlamalı yerel arama metotları kullanmadan çalıştırılan Geometrik Tur Bölme (GTB) metodu da BGA'nın performansını önemli ölçüde etkilemektedir. GTB+BGA için Tablo 6.1 GA sütunundaki araştırma şartları kullanılarak yapılan 10 farklı analizde Tablo 6.2'deki sonuçlar elde edilmiştir. GTB+BGA analizinde Tablo 6.1'den sadece sonlandırma kriteri farklıdır. Tablo 6.2 analizleri için sonlandırma kriteri en iyi bireyin 1000 iterasyon değişmemesidir. Geometrik bölme sabiti (β) yaklaşık [*n*/100-*n*/200] oranlarında seçilmiştir.

Tablo 6.2. Genetik Algoritma ve Geometrik Tur Bölme Sonuçları

Problem Adı	Optimum Mesafe	β Bölme Oranı	Bulunan En İyi Mesafe	En İyi Hata Değeri (% ε_{iyi})	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Süre (*ms*)
Pcb442	50778	4	52630	3.64725	52887	4.15337	7061
Rat575	6773	4	7037	3.89783	7109.2	4.96383	10658
U724	41910	5	43533	3.87258	43898.2	4.74398	11320
Rat783	8806	5	9238	4.90575	9307.4	5.69385	17712
Pr1002	259045	6	271101	4.65402	272701.2	5.27175	17538
Vm1084	239297	6	249371	4.20983	250390	4.63566	13053
Pcb1173	56892	7	58577	2.96175	58675.8	3.13541	15893
Rl1304	252948	8	268855	6.28864	274076.6	8.35294	18334
Rl1323	270199	8	286982	6.21135	288986.2	6.95310	22645
U1432	152970	8	159053	3.97660	160825	5.13499	19125
Vm1748	336556	9	351917	4.56417	352716.8	4.80182	29542
U2152	64253	11	69730	8.52412	70008.6	8.95771	31959
U2319	234256	11	238832	1.95342	238960.4	2.00823	30605
Pr2392	378032	11	397221	5.07603	397659.8	5.19210	41692
Pcb3038	137694	15	142419	3.43152	142975.8	3.83590	55335
Ortalama Hatalar				**4.54499**		**5.18898**	



Şekil 6.3. BGA ve GTB+BGA Metotları Hata ve Süre Değerleri

Tablo 6.2 ve Ek-Tablo 19 analizlerine dikkat edildiğinde, geometrik tur bölme metotları ile ortalamada ε_{ORT}=%5.18898 hata elde edilirken, BGA'da aynı problemler baz alındığında ortalamada ε_{ORT}=%6.94557 hata görülmektedir. Sonuç olarak GTB ile BGA'ların çalışma süreleri yaklaşık %82, ortalama hata değerleri yaklaşık %25.29 ve en iyi tur hatası %20.46 oranında azaltılmıştır. Şekil 6.3'de BGA ve GTB+BGA metotlarının 15 farklı GSP problemi üzerindeki ortalama hata değerleri ve çalışma süreleri gösterilmektedir.

Yapılan literatür araştırmalarındaki deneysel sonuçlarla karşılaştırıldığında hazırlanan model (PGYA) BGA ve GYA'dan daha iyi sonuçlar vermektedir. PGYA analizlerinde (Ek-Tablo 25) çevrim süresinin çok uzun zaman almasından dolayı U2152, U2319, Pr2392 ve Pcb3308 problemleri küçük aday listeleriyle denenmiş, bu yüzden hata değerleri bulunabilecek değerin üstünde kalmıştır. Marinakis ve arkadaşları (2005) hazırladıkları GRASP algoritmasını, *DIMACS Implementation Challenge* sitesinden (*http://www.research.att.com/~dsj/chtsp/*) alınan diğer GSP

metotların sonuçları ile kıyaslamıştır (Tablo 6.3). Marinakis ve arkadaşları tarafından sunulan Tablo 6.3'deki 10 GSP problemindeki Fnl1400 ve Fnl1577 problemleri tez analizlerinde değerlendirilmediği için, D2103 ve Pr2392 problemleri de tam olarak incelenemediğinden değerlendirilmeye alınmaz ise Tablo 6.3 Tablo 6.4'deki hale gelecektir. PGYA modeli ile Pr1002, Pcb1173, D1291, Rl1304, Rl1323 ve Rl1889 Simetrik-GSP problemleri için elde edilen ortalama hata değerlerinin (sırasıyla %0.04909, %0.02918, %0.13287, %0.00000, %0.00999, %0.04897) ortalaması alındığında PGYA ε_{ORT} = %0.27010 / 6 ≈ **%0.045017** hataya sahiptir. Geliştirilen Paralel Genetik Yerel Arama Algoritması modeli bu hali ile Tablo 6.4'de verilen 54 metot arasından 3. en iyi metot seviyesine gelmektedir.

1. TourMerge= %0.00333
2. ILK-NYYY= %0.00333
3. Geliştirilen PGYA= **%0.045017**
4. ILK-J= %0.05667
5. K-MLK= %0.05833

Sonuç olarak GSP için BGA etkili bir algoritma değilken, güçlendirilmesi ile oluşturulan GYA ve PGYA algoritmaları GSP optimizasyonunda oldukça etkilidir.

Hazırlanan PGYA sistemi bu haliyle literatürde incelenen GA temelli BGA, GYA, PGA ve PGYA çalışmalarının hepsinden daha iyi sonuç vermektedir (EAX, NX ve VQX çaprazlama metotları kullananlar hariç). PGYA, gelişmiş tur birleştirme teknikleri (*Tour Merge*), grafik temelli çaprazlama metotları (EAX, NX, VQX), LKH ve CLK gibi gelişmiş yerel arama metotları ile birleştirilerek bu hata değeri biraz daha düşürülebilir.

Tablo 6.3. Çeşitli GSP Metotlarının Ortalama Hata Değerleri

Metot	Pr1002	Pcb1173	D1291	Rl1304	Rl1323	Fl1400	Fl1577	Rl1889	D2103	Pr2392	Ortalama Hata
GRASP	1.16	1.37	1.60	0.88	1.07	0.90	0.80	0.85	1.07	2.11	1.181
Strip	52.21	29.68	*	118.76	102.38	*	*	103.18	*	47.18	75.565
BW-Strip	52.21	29.68	*	56.45	47.11	*	*	63.80	*	47.18	49.405
Spacefil	40.73	40.95	*	63.50	62.21	*	*	63.52	*	44.36	47.545
FRP	64.08	61.13	*	90.09	83.69	*	*	86.14	*	61.03	74.36
B-Greedy	20.10	18.15	22.00	19.31	14.66	23.45	18.33	17.42	13.47	20.19	18.708
C-Greedy	19.43	19.76	20.78	21.16	15.95	20.11	14.71	19.68	10.52	20.21	18.23
Boruvka	16.84	20.14	21.09	16.33	16.89	22.31	17.92	13.97	13.41	18.78	17.76
Q-Boruvka	20.13	14.82	13.57	18.93	21.37	31.80	24.04	21.06	9.62	20.66	19.6
NN	20.71	28.26	25.50	28.62	28.04	33.01	23.35	21.93	14.85	24.27	24.85
CHCI	14.24	17.87	*	19.73	19.91	*	*	18.66	*	17.36	17.96
RI	13.46	16.55	*	19.51	20.60	*	*	18.29	*	16.42	17.47
FI	10.67	15.13	*	22.65	20.89	*	*	17.43	*	13.11	16.64
CW	11.32	11.45	*	11.00	10.25	*	*	11.44	*	12.15	11.26
CCA	9.87	11.91	*	8.24	9.16	*	*	12.05	*	11.37	10.43
HKChrist	6.64	5.29	5.86	7.21	6.49	10.41	6.10	7.23	3.55	6.18	6.496
2opt-J	5.63	6.23	9.45	5.24	3.96	3.86	10.71	5.42	3.87	6.76	6.113
2opt-B	5.88	6.20	*	6.94	5.61	*	*	8.60	*	7.77	6.83
2opt-C	15.70	16.60	12.43	15.74	15.36	15.89	13.34	13.46	9.57	17.51	14.56
2.5opt-B	5.69	4.57	*	5.91	5.87	*	*	6.39	*	7.16	5.93
3opt-J	3.24	3.09	4.24	3.33	1.93	3.32	5.65	4.02	2.84	3.34	3.5
3opt-B	4.11	4.01	*	3.63	2.24	*	*	4.47	*	3.35	3.635
H2	5.50	4.50	6.91	6.44	4.32	8.93	9.04	7.87	2.67	6.00	6.218
H3	3.86	3.40	5.53	5.00	3.27	3.49	8.12	5.78	1.62	5.52	4.559
H4	3.30	3.72	4.97	5.52	2.47	2.13	7.79	4.29	2.04	4.39	4.062
GENI	9.83	8.40	*	9.75	6.08	*	*	5.44	*	8.81	8.05
GENIUS	4.97	2.66	*	7.98	4.39	*	*	3.77	*	4.34	4.685
CLK	2.81	2.14	5.15	4.00	5.47	4.21	9.01	2.00	1.07	3.30	3.916
ACRLK	2.51	5.08	3.91	4.42	3.31	3.12	11.91	3.99	1.77	3.06	4.308
LKHKCh	1.27	1.34	1.79	1.33	0.93	1.87	0.68	1.05	0.48	1.28	1.202
LK-J	1.76	1.52	2.46	1.25	1.13	1.17	0.93	1.21	1.70	1.61	1.41
LK-N	1.32	1.97	2.70	1.45	1.32	1.19	2.83	1.21	2.19	1.86	1.804
LK-NYYY	1.68	1.08	4.51	1.20	0.95	0.89	3.25	1.46	0.99	1.05	1.706
SCE	1.25	1.42	1.58	2.01	1.62	1.27	2.23	2.38	1.78	1.48	1.702
ALK	0.00	0.00	0.12	1.29	0.42	0.67	*	0.33	*	0.53	0.42
H-LK	0.15	0.18	0.92	0.58	0.12	1.09	5.56	0.64	0.21	0.24	0.969
CCLK	0.35	0.54	2.02	0.92	0.32	0.66	2.80	0.49	0.20	0.50	0.88
ACRCLK	0.24	0.13	0.62	0.16	0.21	0.01	2.72	0.56	0.27	0.75	0.567
ILK-J	0.00	0.01	0.13	0.00	0.01	0.00	0.02	0.19	0.00	0.05	0.041
I3opt	0.87	0.25	0.22	0.00	0.11	0.21	0.11	0.69	0.61	1.06	0.413
ILK-N	0.76	1.31	0.41	0.64	0.21	0.76	0.08	0.37	1.11	0.77	0.642
ILK-NYYY	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.06	0.01	0.009
M-LK	0.27	0.41	0.97	0.97	0.47	1.24	0.35	1.39	1.00	0.36	0.743
K-MLK	0.00	0.18	0.17	0.00	0.00	0.18	0.06	0.00	0.05	0.00	0.064
VNS-2H	1.05	1.88	1.24	0.48	0.81	1.38	4.26	3.40	1.48	3.95	1.993
VNS-3H	1.11	1.13	1.99	1.21	0.57	0.24	1.10	2.09	1.20	2.28	1.292
BSDP	0.48	0.28	0.53	0.96	0.53	0.66	7.05	0.47	1.60	0.52	1.308
TourMerge	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.02	0.00	0.004
TS22	4.14	5.36	4.71	7.24	4.33	7.32	3.81	5.06	1.50	4.35	4.83
TS2DB	3.74	4.03	3.90	4.46	3.25	1.25	2.69	3.17	1.13	3.48	3.11
TSLKLK	0.88	0.70	1.90	0.63	0.81	6.40	8.79	0.55	0.69	0.65	2.2
TSLKDB	0.94	1.29	0.69	1.16	1.04	0.84	4.39	0.36	0.73	0.97	1.241
TSSCSC	0.63	1.15	0.31	0.69	0.74	2.95	11.75	1.08	0.50	0.54	2.034
TSSCDB	0.86	1.12	0.36	1.21	0.25	0.29	4.66	0.72	0.27	0.60	1.034

(*: Denenmemiş)

Tablo 6.4. Çeşitli GSP Metotlarının Ortalama Hata Değerleri – 2

Metot	Pr1002	Pcb1173	D1291	Rl1304	Rl1323	Rl1889	Ortalama Hata
GRASP	1.16	1.37	1.60	0.88	1.07	0.85	1.15500
Strip	52.21	29.68	*	118.76	102.38	103.18	81.24200
BW-Strip	52.21	29.68	*	56.45	47.11	63.80	49.85000
Spacefil	40.73	40.95	*	63.50	62.21	63.52	54.18200
FRP	64.08	61.13	*	90.09	83.69	86.14	77.02600
B-Greedy	20.10	18.15	22.00	19.31	14.66	17.42	18.60667
C-Greedy	19.43	19.76	20.78	21.16	15.95	19.68	19.46000
Boruvka	16.84	20.14	21.09	16.33	16.89	13.97	17.54333
Q-Boruvka	20.13	14.82	13.57	18.93	21.37	21.06	18.31333
NN	20.71	28.26	25.50	28.62	28.04	21.93	25.51000
CHCI	14.24	17.87	*	19.73	19.91	18.66	18.08200
RI	13.46	16.55	*	19.51	20.60	18.29	17.68200
FI	10.67	15.13	*	22.65	20.89	17.43	17.35400
CW	11.32	11.45	*	11.00	10.25	11.44	11.09200
CCA	9.87	11.91	*	8.24	9.16	12.05	10.24600
HKChrist	6.64	5.29	5.86	7.21	6.49	7.23	6.45333
2opt-J	5.63	6.23	9.45	5.24	3.96	5.42	5.98833
2opt-B	5.88	6.20	*	6.94	5.61	8.60	6.64600
2opt-C	15.70	16.60	12.43	15.74	15.36	13.46	14.88167
2.5opt-B	5.69	4.57	*	5.91	5.87	6.39	5.68600
3opt-J	3.24	3.09	4.24	3.33	1.93	4.02	3.30833
3opt-B	4.11	4.01	*	3.63	2.24	4.47	3.69200
H2	5.50	4.50	6.91	6.44	4.32	7.87	5.92333
H3	3.86	3.40	5.53	5.00	3.27	5.78	4.47333
H4	3.30	3.72	4.97	5.52	2.47	4.29	4.04500
GENI	9.83	8.40	*	9.75	6.08	5.44	7.90000
GENIUS	4.97	2.66	*	7.98	4.39	3.77	4.75400
CLK	2.81	2.14	5.15	4.00	5.47	2.00	3.59500
ACRLK	2.51	5.08	3.91	4.42	3.31	3.99	3.87000
LKHKCh	1.27	1.34	1.79	1.33	0.93	1.05	1.28500
LK-J	1.76	1.52	2.46	1.25	1.13	1.21	1.55500
LK-N	1.32	1.97	2.70	1.45	1.32	1.21	1.66167
LK-NYYY	1.68	1.08	4.51	1.20	0.95	1.46	1.81333
SCE	1.25	1.42	1.58	2.01	1.62	2.38	1.71000
ALK	0.00	0.00	0.12	1.29	0.42	0.33	0.36000
H-LK	0.15	0.18	0.92	0.58	0.12	0.64	0.43167
CCLK	0.35	0.54	2.02	0.92	0.32	0.49	0.77333
ACRCLK	0.24	0.13	0.62	0.16	0.21	0.56	0.32000
ILK-J	0.00	0.01	0.13	0.00	0.01	0.19	0.05667
I3opt	0.87	0.25	0.22	0.00	0.11	0.69	0.35667
ILK-N	0.76	1.31	0.41	0.64	0.21	0.37	0.61667
ILK-NYYY	0.00	0.00	0.00	0.00	0.01	0.01	0.00333
M-LK	0.27	0.41	0.97	0.97	0.47	1.39	0.74667
K-MLK	0.00	0.18	0.17	0.00	0.00	0.00	0.05833
VNS-2H	1.05	1.88	1.24	0.48	0.81	3.40	1.47667
VNS-3H	1.11	1.13	1.99	1.21	0.57	2.09	1.35000
BSDP	0.48	0.28	0.53	0.96	0.53	0.47	0.54167
TourMerge	0.00	0.00	0.00	0.00	0.01	0.01	0.00333
TS22	4.14	5.36	4.71	7.24	4.33	5.06	5.14000
TS2DB	3.74	4.03	3.90	4.46	3.25	3.17	3.75833
TSLKLK	0.88	0.70	1.90	0.63	0.81	0.55	0.91167
TSLKDB	0.94	1.29	0.69	1.16	1.04	0.36	0.91333
TSSCSC	0.63	1.15	0.31	0.69	0.74	1.08	0.76667
TSSCDB	0.86	1.12	0.36	1.21	0.25	0.72	0.75333

(Tablo 6.3'ün Düzenlenmiş Hali, *: Denenmemiş)

KAYNAKLAR

Alba, E., Troya J.M. 2002. “Improving Flexibility and Efficiency by Adding Parallelism to Genetic Algorithms”, Statistics and Computing 12, pp. 91-114

Amraii, S.A., Ajallooeian M., Lucas C. 2007. “A Dynamic Fuzzy-Based Crossover Method for Genetic Algorithms”, 19th IEEE International Conference on Tools with Artificial Intelligence, Vol. 1, 29-31 October, Patras, Greece, pp. 465-471

Applegate D., Bixby R., Chvatal V., Cook W. 1995. “Finding cuts in the TSP”, DIMACS, Techical Report 95-05

Applegate, D., Bixby R., Chvatal V., Cook W. 2001. “TSP Cuts Which Do Not Conform to the Template Paradigm”, In Computational Combinatorial Optimization, Springer, Berlin, pp. 261-304

Applegate, D., Cook W., Rohe A. 2003. “Chained Lin-Kernighan for Large Traveling Salesman Problems”, Informs Journal on Computing, Vol. 15, pp. 82-92

Babin, G., Deneault S., Laporte G. 2007. “Improvements to the Or-Opt Heuristic for the Symmetric Traveling Salesman Problem”, Journal of the Operational Research Society 58 (3), pp. 402-407

Banzhaf, W. 1990. “The ‘molecular’ traveling salesman”, Biological Cybernetics, Vol. 64, pp. 7-14

Belkadi, K., Gourgand M., Benyettou M. 2006. “Parallel genetic algorithms with migration for the hybrid flow shop scheduling problem”, Journal of Applied Mathematics and Decision Science, Vol. 2006, Art.ID 65746

Bentley, J. L. 1990. “Experiments on Geometric Traveling Salesman Heuristics”, Computing Science Technical Report 151, AT&T Bell Laboratories, Holmdel, NJ

Bentley, J.L. 1992. “Fast algorithms for geometric traveling salesman problems”, ORSA Journal on Computing, 4: pp. 387-411

Brady, R.M. 1985. “Optimization Strategies gleaned from Biological Evolution”, Nature 317, pp. 804-806

Cerny, V. 1985. “A Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm”, Journal of Optimization Theory and Applications, Vol. 45, No.1, pp. 41-51

Cevre, U., Özkan B., Aybars U. 2007. “Gezgin Satıcı Probleminin Genetik Algoritmalarla Eniyilemesi Ve Etkileşimli Olarak İnternet Üzerinde Görselleştirilmesi”, 12. Türkiye'de İnternet Konferansı, Kasım, Bilkent Üniversitesi

Chainate, W., Thapatsuwan P., Pongcharoen P. 2007, “A New Heuristic for Improving the Performance of Genetic Algorithm”, Proceedings of World Academy of Science, Engineering and Technology, Vol. 21, pp. 217-220

Chatterjee, S., Cerrera C., Lynch L.A. 1996. “Genetic Algorithms and Traveling Salesman Problem”, In Europan Journal of Operations Reseaech, Vol. 93, No. 3, pp. 490-510

Croes, G.A. 1958. “A method for solving traveling-salesman problems”, Operations Research, Vol. 6, No. 6, pp. 791-812

Dantzing, G., Fulkerson D., Johnson S. 1954. “Solution of large-scale traveling-salesman problem”, Journal of the Operations Research Society of America, Vol. 2, No. 4, pp. 393-410

Davis, L. 1985. “Applying Adaptive Algorithms to Epistactic Domains”, in Proceedings of the Int. Joint Conf. on Artificial Intelligence (IJCAI'85), Los Angeles, CA, pp. 162-164

Dorigo, M., Gambardella L.M. 1997. “Ant Colonies for the Traveling Salesman Problem”, BioSystems, Vol. 43, pp. 73-81

Dorigo, M., Gambardella L.M. 1997. “Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem”, IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 53-66

Dorigo, M., Maniezzo V., Colorni A. 1991. “Positive feed-back as a search strategy”, Technical Report 91016, Dipartimento di Elettronica e Informatica, Politecnico di Milano, Italy

Dzubera, J., Whitley D. 1994. “Advanced Correlation Analysis of Operators fort he Traveling Salesman Problem”, In Parallel Problem Solving form Nature – PPSN III, Y. Davidor, H.P. Schwefel and R. Manner, eds., Springer-Verlag, pp. 68-77

Emel, G.G., Taşkın Ç. 2002. “Genetik Algoritmalar ve Uygulama Alanları”, Uludağ Üniversitesi, İktisadi ve İdari Bilimler Fakültesi Dergisi, Cilt 21, Sayı 1, s. 129-152

Fogel, D.B. 1988. “An evolutionary approach to the traveling salesman problem”, Biological Cybernetics, Vol. 60, No. 2, pp. 139–144

Fogel, D.B. 1990. “A parallel processing approach to a multiple traveling salesman problem using evolutionary programming”, Proceedings of the Fourth annual Symposium on Parallel Processing, Fullerton, California, pp. 318–326

Freisleben, B., Merz P. 1996. “A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems”, In IEEE International Conference on Evolutionary Computation, IEEE Press, pp. 616-621

Gen, M., Cheng R. 2000. “Genetic Algorithms and Engineering Optimization”, Wiley, New York

Ghoseiri, K., Sarhadi H. 2007. “A 2opt-DPX Genetic Local Search for Solving Symmetric Traveling Salesman Problem”, IEEE International Conference on December, pp. 903-906

Glover, F. 1989. “Tabu Search - Part I”, ORSA Journal of Computing Vol. 1, No. 3, pp. 190-206

Glover, F. 1990. “Tabu Search – Part II”, ORSA Journal of Computing Vol. 2, No. 1, pp. 4-32

Goldberg, D.E. 1989. “Genetic Algorithms in Search Optimization and Machine Learning”, Addison-Wesley Publishing Company Inc

Goldberg, D.E., Lingle R. 1985. “Alleles, Loci and the Traveling Salesman Problem”, in Proceedings of the First Int. Conf. on Genetic Algorithms (ICGA'85), Carnegie-Mellon University, Pittsburgh, PA, pp. 154-159

Grefenstette J. 1987. “Incorporating Problem Specific Knowledge into Genetic Algorithms”, in Genetic Algorithms and Simulated Annealing, L. Davis Eds, Morgan Kaufmann, pp. 42-60

Grefenstette, J., Gopal R., Rosmaita B.J., Gucht D.V. 1985. “Genetic Algorithms for the Traveling Salesman Problem”, Proceedings of the First Int. Conf. on Genetic Algorithms (ICGA'85), Carnegie-Mellon University, Pittsburgh, PA, pp. 160-168

Haupt, R.L, Haupt S.E. 2004. “Practical Genetic Algorithms”, 2nd Ed., Wiley, New Jersey

Helsgaun K. 2000. “An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic”, European Journal of Operational Research, Vol. 126, No. 1, pp. 106-130

Helsgaun, K. 2006. “An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic”, Datalogiske Skrifter - Writings on Computer Science, No. 109, Roskilde University

Holland, J. H. 1975. “Adaptation in Natural and Artificial Systems”, Ann Arbor, MI: University of Michigan Pres.

Hu, Y.W.Y., Gu K. 2007. “Parallel Search Strategies for TSPs Using a Greedy Genetic Algorithm”, Natural Computation, Third International Conference on Volume 3, 24-27 August, pp. 786 – 790

Hulin, M. 1997. “An Optimal Stop Criterion for Genetic Algorithms: A Bayesian Approach”, Proceedings of the 7th International Conference on Genetic Algorithms, East Lansing, MI, USA, July 19-23, pp. 135-143

Ibraki, T. 1997. “Combinations with other optimization methods”, Beack T., Fogel D., Michalewicz Z. (Eds.), Handbook of Evolutionary Computation (Section D3), Oxford University Press, London

Jayalakshmi, G.A., Sathiamoorthy S., Rajaram R. 2001. “A Hybrid Genetic Algorithm - A New Approach to Solve Traveling Salesman Problem”, International Journal of Computational Engineering Science, Vol. 2, No. 2, pp. 339-355

Jog, P., Suh J.Y., Gucht D.V. 1989. “The Effects of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem”, in Proceedings of the Third Int. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Fairfax, VA, pp. 110-115

Johnson, D.S. 1990. “Local optimization and the traveling salesman problem”, In ICALP'90, Proceedings of the 17th Colloquium on Automata, Languages, and Programming, Springer-Verlag, pp. 446-461

Johnson, D.S., McGeoch L.A. 1997. “The traveling salesman problem: A case study in local optimization”. In Aarts E.H.L., Lenstra J.K. (Editors), Local Search in Combinatorial Optimization, pp. 215-310

Johnson, D.S., McGeoch L.A. 2002. “Experimental Analysis of Heuristics for the STSP” chapter of “The Traveling Salesman Problem and its Variations”, Gutin G., Punnen A. (Editors), Kluwer Academic Publishers, pp. 369-443

Jung, S., Moon B.R. 2002. “Toward minimal restriction of genetic encoding and crossovers for the two-dimensional Euclidean TSP”, IEEE Transactions on Evolutionary Computation, Vol. 6, No. 6, pp. 557-565

Karaboğa, D. 2004. “Yapay Zeka Optimizasyon Algoritmaları”, 1. Baskı, Atlas, İstanbul

Karaoğlan, İ., Altıparmak F. 2005. “Konkav Maliyetli Ulaştırma Problemi İçin Genetik Algoritma Tabanlı Sezgisel Bir Yaklaşım”, Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi, Cilt 20, No 4, s. 443-454

Karp, R.M. 1977. “Probabilistic analysis of partitioning algorithm for the Traveling- Salesman Problem in the plane”, Mathematics of Operations Research, Vol. 2, No. 3, pp. 209-224

Katayama, K., Narihisa H. 1999. “A New Iterated Local Search Algorithm using Genetic Crossover for the Traveling Salesman Problem”, Proc. of the 14th Annual ACM Symposium on Applied Computing (SAC-99), Texas, Feb. 28, pp. 302-306

Kirkpatrick, S., Gelatt C.D., Vecchi P.M. 1983. “Optimization by Simulated Annealing”, Science, Vol. 220, No. 4598, pp.671-680

Krishnakumar, K. 1989. “Micro-genetic algorithms for stationary and non-stationary function optimization”, Intelligent Control and Adaptive Systems, Proc. of the SPIE, Vol. 1196, pp. 289-296

Kuvat, G., Adar N., Canbek S., Seke E. 2007. "Hızlı Yakınsayan Göç Yönteminin Farklı Test Fonksiyonları İçin İncelenmesi", 12. Elek. Elektro. Bilg. ve Biyomedikal Müh. Ulusal Kongresi, Kasım, Eskişehir

Lawler, E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B. 1985. “The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization”, New York, Wiley

Liepins, G.E., Hilliard M.R., Palmer M., Morrow M. 1987. “Greedy Genetics”, in Proceedings of the Second Int. Conf. on Genetic Algorithms (ICGA'87), Massachusetts Institute of Technology, Cambridge, MA, pp. 90-99

Lin, S. 1965. “Computer solutions of the traveling-salesman problem”. Bell System Technology Journal, 44, 2245–2269

Lin, S., Kernighan B. 1973. “An effective heuristic algorithm for the traveling-salesman problem”. Operations Research, Vol. 21, No. 2, pp. 498-516

Liu, Z., Kang L. 2003. “A Hybrid Algorithm of n-OPT and GA to Solve Dynamic TSP”, Grid and Cooperative Computing (GCC 2003), Second International Workshop, Shanghai, December 7-10, pp. 1030-1033

Louis, S.J., Tang R. 1999. “Interactive genetic algorithms for the traveling salesman problem”, In Proceedings of the 1999 Genetic and Evolutionary Computing Conference (GECCO 1999), pp. 385-392

Marinakis, Y., Migdalas A., Pardalos P.M. 2005. “Expanding Neighborhood GRASP for the Traveling Salesman Problem”, Computational Optimization and Applications, Volume 32, Number 3, pp. 231-257

Martin, O., Otto S.W., Felten E.W. 1991. “Large-step Markov chains for the traveling salesman problem”, Complex Systems 5, pp. 299-326

Matayoshi, M., Nakamura M., Miyagi H. 2004. “A genetic algorithm with the improved 2-opt method”, IEEE International Conference on Systems, Man and Cybernetics, Volume 4, 10-13 October, pp. 3652-3658

Mathias, K., Whitley D. 1992. “Genetic Operators the Fitness Landscape and the Traveling Salesman Problem”, In Parallel Problem Solving from Nature, R. Manner and B. Manderick, eds., North Holland-Elsevier, pp. 219-228

Merz, P., Freisleben B. 1997. “Genetic local search for the TSP: new results”, Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, IEEE Press, Indianapolis, USA, pp. 159-164

Metropolis, N., Rosenbluth A., Rosenbluth M., Teller A. 1953. “Equation of state calculations by fast computing machines”, Journal of Chemical Physics, Vol. 21, No.6, pp. 1087-1092

Michalewicz, Z. 1992. “Genetic Algorithms + Data Structures = Evolution Programs”, Springer, Berlin

Michalewicz, Z., Fogel D.B. 2000. “How to Solve It: Modern Heuristics”, Springer-Verlag, Berlin, Germany

Milos, O., Josef S. 2004. “Hybrid Parallel Simulated Annealing Using Genetic Operations”, Mendel 2004 10th International Conference on Soft Computing, Brno, CZ, FSI VUT, pp. 89-94

Misevicius, A., Blazauskas T., Blonskis J., Smolinskas J. 2004. “An Overview of Some Heuristic Algorithms for Combinatorial Optimization Problems”, Information Technology and Control, Kaunas, Lithuania, No. 1(30), pp. 21-31

Mitchell, G.G. 2005. “Validity Constraints and the TSP-GeneRepair of Genetic Algorithms”, Artificial Intelligence and Applications, pp. 306-311

Mulhenbein, H. 1991. “Evolution in Time and Space – The Parallel Genetic Algorithm”, Foundations of Genetic Algorithms, G.J.E. Rawlins Eds, Morgan Kaufmann, pp. 316-337

Nabiyev, V.V. 2003. “Yapay Zeka”, Seçkin, Ankara

Nagata, Y. 2006. “New EAX crossover for large TSP instances”, Proc. of Parallel Problem Solving from Nature, PPSN IX, pp. 372-381

Nagata, Y., Kobayashi S., 1997. “Edge Assembly Crossover: A High-power Genetic Algorithm for the Traveling Salesman Problem”, Proc. of the 7th Int. Conference on Genetic Algorithms, pp. 450-457

Nguyen, H.D., Yoshihara I., Yasunaga M. 2000. “Modified Edge Recombination Operators of Genetic Algorithms for the Traveling Salesman Problem”, Proc. IEEE Int. Conf. on Industrial Electronics, Control, and Instrumentation (SEAL session), pp.2815-2820

Oliveira, A.C.M., Lorena L.A.N., Preto, A. J., Stephany S. 2004. “An Adaptive Hierarchical Fair Competition Genetic Algorithm for Large-Scale Numerical Optimization”, BEC2004 - I Brazilian Workshop On Evolutionary Computation 2004, São Luís, Proceedings of SBRN 2004 - 8th Brazilian Symposium on Neural Networks

Oliver, I. M., Smith D. J., Holland J. R. C. 1987. “A Study of Permutation Crossover Operators on the Traveling Salesman Problem”, in Proceedings of the Second Int. Conf. on Genetic Algorithms (ICGA'87), Massachusetts Institute of Technology, Cambridge, MA, pp. 224-230

Or, I. 1976. “Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking”. Ph.D. Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL

Osman, I.H., Laporte G. 1996. “Metaheuristics: A Bibliography”, Annals of Operational Research, Vol. 63, No. 5, pp. 513-618

Potvin, J.Y. 1996. “Genetic algorithms for the traveling salesman problem”, Annals of Operations Research, Vol. 63, No. 3, pp. 337-370

Ray, S.S., Bandyopadhyay S., Pal S. K. 2004. “New Operators of Genetic Algorithms for Traveling Salesman Problem”, 17th International Conference on Pattern Recognition (ICPR'04) - Volume 2, pp. 497-500

Ray, S.S., Bandyopadhyay S., Pal S. K. 2005. “New Genetic Operators for Solving TSP: Application to Microarray Gene Ordering”, Pattern Recognition and Machine Intelligence, First International Conference, PReMI 2005, Kolkata, India, December 20-22, pp. 617-622

Rebaudengo, M., Reorda M.S. 1993. “An Experimental Analysis of Effects of Migration in Parallel Genetic Algorithms”, EWPDP93: IEEE/Euromicro Workshop on Parallel and Distributed Processing, Gran Canaria (E), pp. 232-238

Rocha, M., Neves J. 1999. “Preventing Premature Convergence to Local Optima in Genetic Algorithms via Random Offspring Generation”, Multiple approaches to intelligent systems, Springer, Vol. 1611, pp. 127-136

Rosenkrantz, D.J., Stearns R.E., Lewis P.M. 1977. “An analysis of several heuristics for the traveling salesman problem”, Society for Industrial and Applied Mathematics (SIAM), Vol. 6, No.3, pp. 563-581

Sastry, K., Goldberg D., Kendall G. 2005. “Genetic Algorithms”, Kluwer Academic Publishers, Dordrecht, Boston, pp. 97-125

Schleuter, M.G. 1989. “Asparagos: An Asynchronous Parallel Genetic Optimization Strategy”, Proceedings of the third international conference on Genetic Algorithms (ICGA'89), December, George Mason University, United States, pp. 422-427

Schleuter, M.G. 1997. “Asparagos96 and the Traveling Salesman Problem”, in Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, IEEE Press, pp. 171-174

Sengoku, H., Yoshihara I. 1998 “A Fast TSP Solver Using GA on JAVA”, in Third International Symposium on Artificial Life, and Robotics (AROB III’98), pp. 283-288

Seo, D., Moon B. 2002. “Voronoi Quantized Crossover for Traveling Salesman Problem”, Genetic and Evolutionary Computation Conference, pp. 544-552

Snyder, L.W., Daskin M.S. 2006. “A random-key genetic algorithm for the generalized traveling salesman problem”, European Journal of Operational Research, Vol. 174 (1), pp. 38-53

Stützle, T., “The Traveling Salesman Problem: State of the Art”, İnternet Erişimi Adresi: “www.sls-book.net/slides/tsp.pdf” - Son Erişim: 13.01.2008

Stützle, T., Hoos H. 1997. “The MAX-MIN Ant System and Local Search for the Traveling Salesman Problem”, Proceedings of the IEEE International Conference on Evolutionary Computation, April 13-16, Indianapolis, Indiana, USA, pp. 308-313

Stützle, T., Hoos H. 1999. “Analyzing the Run-time Behaviour of Iterated Local Search for the TSP”, III. Metaheuristics International Conference

Suh, J.Y., Gucht D.V. 1987. “Incorporating Heuristic Information into Genetic Search”, in Proceedings of the Second Int. Conf. on Genetic Algorithms (ICGA'87), Massachusetts Institute of Technology, Cambridge, MA, pp. 100-107

Syswerda G. 1990. “Schedule Optimization using Genetic Algorithms”, in Handbook of Genetic Algorithms , L. Davis Eds, Van Nostrand Reinhold, pp. 332-349

Tao, G., Michalewicz Z. 1998. “Inver-over Operator for the TSP”, Proceedings of the 5th Parallel Problem Solving from Nature, T. Baeck, A.E. Eiben, M. Schoenauer, and H.-P. Schwefel (Editors), Amsterdam, September 27-30, pp. 803-812

TSPLIB Kütüphanesi (Son Erişim: 03.04.2008),
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

Türkoğlu, Ö. 2001. “Genetik Algoritma Metodu İle Düzlemsel Yakın Alan Elde Etmek İçin Uygunluk Fonksiyonu Analizi”, İstanbul Üniversitesi Mühendislik Fakültesi, Elektrik ve Elektronik, Cilt 1, Sayı 2, s. 257-265

Ulder, N.L.J., Aarts E.H.L., Bandelt H.J., Laarhoven P.J.M.V., Pesch E. 1991. “Genetic Local Search Algorithms for the Traveling Salesman Problem”, Parallel Problem Solving from Nature, Springer-Verlag, Vol. 496, pp. 109-116

Valenzuela, C.L., Jones A.J. 1994. “Evolutionary Divide and Conquer (I): a novel genetic approach to the TSP”, Evolutionary Computation, Vol. 1, No. 4. pp. 313-333

White, C.M., Yen G.G. 2004. “A hybrid evolutionary algorithm for traveling salesman problem”, Evolutionary Computation (CEC2004), Congress on Volume 2, 19-23 June, pp. 1473-1478

Whitley, D. 1989. “The Genitor Algorithm and Selection pressure: Why Rank-Based Allocation of Reproductive Trials is Best”, in Proceedings of the Third Int. Conf. on Genetic Algorithms (ICGA'89), George Mason University, Fairfax, VA, pp. 116-121

Wiese, T. 2007. “Global Optimization Algorithms - Theory and Application”, GNU Free Documentation License

Yagiura, M., Ibaraki T. 1996. “Metaheuristics as robust and simple optimization tools” Proceedings 1996 IEEE Conference on Evolutionary Computation, 20-22 May, pp. 541-546

Zhang, B.T., Kim J.J. 2000. “Comparison of selection methods for evolutionary optimization”, Evolutionary Optimization, Vol. 2, No. 1, pp. 55-70

EKLER

Ekler bölümünde, 5. ve 6. bölüm içerisinde yapılan analizlere ait 25 farklı tablo bulunmaktadır. Ek-Tablo 1'den Ek-Tablo 17'ye kadar Genetik Algoritma ve bileşenlerine ait uygun parametreleri bulmak amacıyla Bölüm 5'de yapılan analizlerin sonuçlarını içeren tablolar; Ek-Tablo 18'den Ek-Tablo 25'e kadar ise GA, GYA, PGA, PGYA modellerinin değerlendirilmesi için Bölüm 6'da yapılan analizlere ait sonuçları içeren tablolar sunulmuştur.

Ek-Tablo 1. Popülasyon Büyüklüğünün GA Çözümündeki Etkisi [*Konf.*-1]

Problem	Optimum Değer	**Pop. Sayısı = 6**			**Pop. Sayısı = 10**			**Pop. Sayısı = 20**			**Pop. Sayısı = 40**			**Pop. Sayısı = 80**			**Pop. Sayısı= 120**		
		ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (s/10)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (s/10)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (s/10)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (s/10)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (s/10)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (s/10)
Berlin52	7542	0.00	3.23	0.23	0.00	2.98	0.22	0.00	3.87	0.24	0.00	3.94	0.21	0.00	3.19	0.27	0.00	4.66	0.33
Eil76	538	2.97	5.03	0.32	2.41	4.73	0.29	2.97	4.36	0.35	1.48	3.84	0.33	2.23	3.81	0.38	3.15	4.33	0.49
Lin105	14379	3.15	5.88	0.34	1.05	3.98	0.43	0.76	3.75	0.43	0.94	5.62	0.42	0.77	4.85	0.45	0.53	4.63	0.71
KroA150	26524	1.17	5.45	0.59	3.78	5.41	0.59	3.99	6.24	0.53	1.61	5.96	0.67	3.00	5.36	0.83	3.66	5.15	0.93
KroB200	26130	4.23	6.68	0.89	4.09	6.98	0.80	4.60	7.44	0.86	3.43	6.09	0.84	5.49	7.19	0.89	4.88	7.03	1.04
Ts225	126643	4.41	7.19	0.73	4.31	7.51	0.87	1.68	7.37	0.79	3.36	6.73	0.86	2.32	7.31	1.04	5.29	8.03	1.32
Pr264	49135	4.71	9.85	1.14	6.49	10.11	1.09	5.08	11.85	1.03	4.33	11.24	1.12	5.08	9.36	1.64	4.95	10.33	1.54
Lin318	42029	6.17	8.50	1.60	6.22	9.13	1.31	5.92	8.84	1.53	5.22	9.18	1.30	4.91	7.60	1.45	5.95	7.00	2.16
Pcb442	50778	7.19	9.29	1.99	6.65	8.63	2.21	7.24	9.43	2.01	7.29	9.59	2.07	6.70	9.39	2.26	6.06	10.41	2.50
Rat783	8806	9.30	10.56	5.16	7.22	9.61	5.01	8.13	9.66	5.19	8.05	9.76	5.04	8.28	10.27	6.02	7.65	9.20	8.65
Vm1084	239297	9.19	11.23	9.78	9.09	10.67	8.75	9.33	11.20	7.66	9.43	10.70	8.77	7.84	10.14	10.5	7.80	9.87	11.7
Pcb1173	56892	9.80	12.38	10.1	9.36	11.30	11.7	9.49	11.22	10.2	9.83	11.95	10.3	9.79	11.19	12.7	9.22	10.88	17.0
U2319	234256	4.24	5.37	24.7	4.68	5.44	22.6	4.72	5.04	22.8	4.58	5.38	22.4	4.48	4.97	29.9	4.53	5.10	35.0
Pcb3038	137694	11.04	12.22	64.3	10.06	11.85	63.1	10.53	11.87	57.7	10.02	10.97	60.2	10.67	11.51	67.0	10.68	12.10	72.9
Ortalamalar		5.54	8.06	8.71	5.39	7.74	8.50	5.32	8.01	**7.95**	**4.97**	7.93	8.18	5.11	**7.58**	9.67	5.31	7.74	11.16

Ek-Tablo 2. Tur Oluşturma Metotlarının Hata Değerleri

Problem	Optimum Değer	**Random**		**NN**		**FI**		**RI**		**NF**		**NI**		**DENN**	
		ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)
Berlin52	7542	240	300	8.80	25.04	0.00	7.43	3.51	11.73	9.46	28.25	16.73	20.12	27.51	44.84
Eil76	538	316	369	13.01	24.08	4.46	8.42	5.01	9.16	11.52	28.68	10.03	16.28	15.42	35.80
Lin105	14379	659	760	17.77	30.67	1.66	7.06	3.04	10.42	17.89	35.99	21.65	28.66	27.65	52.40
KroA150	26524	746	867	18.68	26.73	3.32	7.51	4.35	9.63	20.64	31.65	14.48	17.34	28.35	44.24
KroB200	26130	956	1032	21.00	27.18	3.92	8.12	5.50	10.15	21.00	31.19	22.37	24.03	20.71	40.46
Ts225	126643	1052	1158	11.17	17.24	5.43	10.97	7.80	12.96	10.93	22.90	19.83	26.19	14.02	34.12
Pr264	49135	2001	2183	10.90	17.47	9.22	11.86	7.35	11.21	9.82	23.02	25.88	29.48	14.92	43.19
Lin318	42029	1202	1302	18.51	25.18	6.50	9.18	6.71	10.59	17.06	28.62	21.80	24.55	21.39	34.90
Pcb442	50778	1350	1420	16.28	22.97	9.39	13.05	10.80	14.41	16.64	26.71	16.91	18.90	20.55	33.66
Rat783	8806	1872	1937	19.83	27.57	9.27	11.40	9.53	13.15	20.46	28.92	21.93	23.25	25.82	36.84
Vm1084	239297	3351	3472	22.50	26.18	9.57	12.00	9.97	12.55	21.97	28.16	16.86	17.98	28.49	38.85
Pcb1173	56892	2303	2381	24.09	28.99	13.74	15.40	14.40	16.76	23.52	30.21	24.74	26.95	27.33	37.43
U2319	234256	2412	2460	16.00	18.13	5.78	6.80	6.67	7.40	14.77	18.90	13.08	13.84	17.64	22.32
Pcb3038	137694	3745	3825	23.30	25.48	13.33	14.66	14.38	15.80	23.93	26.71	23.07	23.63	25.08	30.38
Rl5915	565530	7330	7423	19.11	22.24	20.78	22.10	20.89	22.54	19.72	23.38	26.27	27.63	23.29	29.74
Ortalama Hata		1969	2059	17.4	24.34	**7.76**	**11.06**	8.66	12.56	17.29	27.55	19.71	22.59	22.54	37.28

Ek-Tablo 3. Tur Oluşturma Metotları ile Başlayan BGA'nın Hata Değerleri

Problem	Optimum Değer	**Random**		**NN**		**FI**		**RI**		**NF**		**NI**		**DENN**	
		ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)	ε_{iyi} (%)	ε_{ORT} (%)
Berlin52	7542	0.00	2.23	0.00	1.59	0.00	0.00	0.00	1.47	0.00	1.87	0.00	5.36	0.00	1.73
Eil76	538	0.92	4.07	2.97	3.60	2.41	3.14	3.71	4.62	1.30	2.80	1.48	3.71	1.11	2.76
Lin105	14379	3.03	4.88	1.49	2.69	1.07	1.50	0.69	3.03	0.41	1.99	4.05	6.69	0.61	1.93
KroA150	26524	1.95	5.06	3.72	6.00	0.66	2.06	2.00	2.89	4.55	5.93	4.06	5.81	1.63	4.99
KroB200	26130	4.25	6.68	3.99	6.89	1.35	3.07	1.29	3.78	1.92	4.91	4.22	6.05	2.38	5.00
Ts225	126643	5.00	7.89	0.95	3.23	5.05	5.79	2.93	6.10	1.50	4.35	2.42	9.60	0.52	1.35
Pr264	49135	4.85	9.30	4.39	6.22	5.06	6.50	2.85	4.73	3.83	6.53	10.58	11.50	6.95	9.22
Lin318	42029	5.07	7.90	4.32	5.21	4.12	4.59	4.19	5.31	4.09	5.33	5.76	7.47	3.40	5.69
Pcb442	50778	5.56	8.56	2.89	4.14	8.09	8.61	6.32	8.51	3.55	4.81	6.38	8.24	2.70	4.67
Rat783	8806	7.73	9.59	4.69	5.96	6.71	8.06	7.81	8.84	5.56	6.94	9.05	10.22	5.40	6.46
Vm1084	239297	7.99	9.87	6.53	8.21	6.84	7.85	6.52	7.61	6.19	8.14	6.91	9.20	5.78	7.41
Pcb1173	56892	9.96	12.03	6.70	8.07	10.51	11.86	10.23	11.47	6.68	8.05	8.44	9.77	6.62	8.11
U2319	234256	4.57	5.04	2.73	3.25	4.58	4.92	4.67	5.17	3.00	3.76	4.60	5.28	2.82	3.36
Pcb3038	137694	10.95	11.94	6.99	7.98	10.88	11.91	10.29	11.47	6.88	8.07	9.44	10.11	6.73	7.37
Rl5915	565530	14.91	16.70	9.62	10.17	15.89	18.19	15.49	16.74	8.45	10.14	9.96	11.48	8.31	9.97
Ortalama Hata		5.783	8.116	4.132	5.547	5.548	6.537	5.266	6.783	3.861	5.575	5.823	8.033	**3.664**	**5.335**

Ek-Tablo 4. Tur Oluşturma Metotlarının Karıştırılmasının Etkisi [*Konf.*-1]

Metot	Sonuçlar	KroA150			KroB200			Ts225			Ortalama		
		%25 *Rnd*	%50 *Rnd*	%75 *Rnd*	%25 *Rnd*	%50 *Rnd*	%75 *Rnd*	%25 *Rnd*	%50 *Rnd*	%75 *Rnd*	%25 *Rnd*	%50 *Rnd*	%75 *Rnd*
NN	$\varepsilon_{İYİ}$ (%)	4.05	3.91	2.04	4.19	2.54	4.12	1.02	1.49	2.22	3.09	2.65	2.79
	ε_{ORT} (%)	5.97	5.66	5.41	5.67	5.04	6.44	3.41	3.73	4.94	5.02	4.81	5.60
	t_{ORT} (*ms*)	5712	4209	5525	7587	6804	7642	6342	5851	5198	6547	5621	6122
FI	$\varepsilon_{İYİ}$ (%)	0.99	1.34	1.22	2.05	2.06	2.13	5.09	5.23	5.22	2.71	2.88	2.86
	ε_{ORT} (%)	1.77	2.43	2.57	3.55	3.74	3.96	5.78	6.98	7.17	3.70	4.38	4.57
	t_{ORT} (*ms*)	3826	3275	3171	5290	5236	3809	3393	3581	3925	4170	4031	3635
RI	$\varepsilon_{İYİ}$ (%)	1.39	1.48	2.55	2.69	2.29	0.98	3.33	7.12	1.63	2.47	3.63	1.72
	ε_{ORT} (%)	3.02	3.49	3.95	3.92	4.02	3.81	6.50	7.74	8.12	4.48	5.08	5.29
	t_{ORT} (*ms*)	3415	3915	4137	4162	4840	5614	5271	3871	5059	4283	4209	4937
NF	$\varepsilon_{İYİ}$ (%)	4.81	3.41	3.20	2.35	2.59	2.97	2.87	0.31	0.29	3.34	2.10	2.15
	ε_{ORT} (%)	6.38	4.76	5.78	5.07	5.40	6.07	5.35	3.69	4.18	5.60	4.62	5.34
	t_{ORT} (*ms*)	4682	5878	5381	6223	8512	7329	5417	6065	5804	5441	6818	6171
NI	$\varepsilon_{İYİ}$ (%)	3.46	2.20	3.51	4.83	4.77	3.64	8.06	7.75	3.45	5.45	4.91	3.53
	ε_{ORT} (%)	4.97	5.84	5.33	7.02	7.88	6.66	11.92	9.98	9.69	7.97	7.90	7.23
	t_{ORT} (*ms*)	6384	5984	6836	9831	9510	9803	7387	9275	8576	7867	8256	8405
DENN	$\varepsilon_{İYİ}$ (%)	2.78	1.83	2.28	2.83	3.33	3.96	0.30	0.30	0.33	1.97	1.82	2.19
	ε_{ORT} (%)	5.18	4.84	3.70	4.96	5.71	5.71	1.93	2.64	3.06	4.02	4.40	4.16
	t_{ORT} (*ms*)	4696	4918	6085	7228	6662	7848	7081	5743	6793	6335	5774	6909
Ortalama Hata (ε_{ORT}) Değeri Ortalaması		4.55	4.50	**4.46**	5.03	5.30	5.44	5.82	5.79	6.19	5.13	5.20	5.36

Ek-Tablo 5. Sabit Çaprazlama Oranının Çözümdeki Etkisi [*Konf.*-1]

Problem	Optimum Değer	$P_Ç$ = %10			$P_Ç$ = %30			$P_Ç$ = %50			$P_Ç$ = %70			$P_Ç$ = %85			$P_Ç$ = %100		
		$\varepsilon_{İYİ}$ (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	$\varepsilon_{İYİ}$ (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	$\varepsilon_{İYİ}$ (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	$\varepsilon_{İYİ}$ (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	$\varepsilon_{İYİ}$ (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	$\varepsilon_{İYİ}$ (%)	ε_{ORT} (%)	t_{ORT} (*ms*)
Berlin52	7542	5.21	8.32	532	0.00	7.21	617	0.00	7.21	754	2.93	7.20	645	2.61	6.71	526	5.70	8.29	690
Eil76	538	5.01	8.21	584	4.83	6.65	707	5.76	8.06	734	2.04	7.54	690	5.57	8.23	775	3.53	7.24	809
Lin105	14379	2.33	7.35	1087	7.24	10.94	1121	3.66	8.43	1342	3.17	8.91	1318	4.48	9.13	1239	3.56	7.47	1512
KroA150	26524	8.15	12.69	1746	4.18	9.83	1987	8.32	11.18	2082	5.80	10.58	2470	6.92	10.59	2440	9.82	11.97	2521
KroB200	26130	8.25	11.46	3175	8.94	11.58	3217	9.58	11.85	3512	6.67	11.84	3687	9.93	12.98	3918	9.52	12.89	4254
Ts225	126643	5.89	10.11	3984	7.26	10.42	4028	5.96	9.79	4493	5.60	9.89	4903	6.75	9.64	4975	4.11	9.80	5690
Pr264	49135	12.02	14.22	4745	10.18	15.46	5492	10.87	14.86	5453	9.73	14.40	6734	9.69	15.23	6289	9.42	13.78	7348
Lin318	42029	11.52	14.15	6612	9.59	13.36	7025	9.38	13.01	8400	9.64	14.29	9004	9.34	13.57	9884	11.53	16.10	10598
Pcb442	50778	15.67	18.07	11248	11.56	17.11	12935	14.65	18.16	13843	14.29	18.30	14850	13.68	17.72	16979	17.94	19.67	16812
Rat783	8806	19.99	23.00	41690	21.72	24.03	45615	20.19	23.43	51632	21.03	23.11	58553	20.89	24.41	61012	22.54	25.95	62720
Vm1084	239297	27.16	31.46	83078	23.28	30.44	96529	29.32	36.23	95231	26.07	37.16	108078	29.51	35.86	119793	32.63	38.09	126143
Ortalamalar		11.02	14.46	14407	9.89	**14.28**	16298	10.70	14.75	17043	**9.72**	14.84	19176	10.85	14.92	20712	11.85	15.57	21736

Ek-Tablo 6. Sabit Mutasyon Oranının Çözümdeki Etkisi [*Konf.*-1]

Problem	Optimum Değer	P_M = %5			P_M = %10			P_M = %25			P_M = %50			P_M = %75			P_M = %100		
		ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (ms)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (ms)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (ms)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (ms)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (ms)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (ms)
Berlin52	7542	7.90	12.64	1617	2.04	9.40	1189	6.62	11.30	732	4.85	9.81	507	1.78	6.98	507	3.08	8.08	543
Eil76	538	9.29	13.23	2384	5.94	11.18	1872	7.62	9.12	1010	5.01	8.34	829	6.50	9.25	720	5.01	8.17	679
Lin105	14379	11.23	23.57	4607	8.30	14.38	4071	3.29	9.39	2406	6.56	9.79	1568	7.44	12.46	1234	3.70	8.77	1443
KroA150	26524	18.30	25.16	8651	10.20	20.21	5675	9.06	13.84	3843	7.75	10.72	2714	7.69	10.40	2242	8.12	11.60	2592
KroB200	26130	24.38	34.91	11940	19.12	23.18	9596	12.70	15.86	6528	9.42	12.89	4700	10.01	13.62	3640	6.83	11.38	4576
Ts225	126643	35.74	51.00	15767	26.40	37.39	11192	7.37	16.41	7981	4.42	9.97	5985	3.65	7.97	5679	4.30	8.46	5660
Pr264	49135	37.91	49.48	19426	21.69	32.64	14757	13.45	21.78	10295	12.69	17.70	7089	8.19	14.68	6145	8.69	14.09	6998
Lin318	42029	38.92	54.05	27987	30.97	37.26	21340	13.60	21.59	15600	12.53	14.88	11153	10.93	14.46	9448	7.96	13.62	10615
Pcb442	50778	60.32	81.55	36893	38.78	47.12	33914	21.36	25.61	26903	15.73	19.22	19995	13.76	18.19	15532	14.09	18.42	19023
Rat783	8806	100.4	118.6	1.1e+5	60.95	75.23	96132	33.34	39.83	79053	21.06	27.39	62996	19.29	23.31	57885	21.66	25.86	62732
Vm1084	239297	161.8	207.5	1.8e+5	115.6	127.4	1.6e+5	54.28	67.00	1.4e+5	33.99	39.67	1.2e+5	30.29	32.46	1.1e+5	31.69	34.89	1.3e+5
Ortalamalar		46.02	61.06	38116	30.91	39.58	32703	16.61	22.88	26759	12.18	16.40	21594	10.87	14.89	19367	**10.47**	**14.85**	22260

Ek-Tablo 7. Berlin52 için Çaprazlama-Mutasyon Çifti Ortalama Hata Değerleri

Çaprazlama / Mutasyon	Tek N.	İki N.	Çok N.	PMX	CX	OX	MOX	OBX	PBX	GSX	AEX	ERX	HX	HX2	DPX	Rasgele	Ortalama (Mutasyon)
Exchange	32.61	27.21	34.23	30.56	14.75	23.46	20.83	28.98	25.53	30.93	20.12	19.86	37.04	4.39	1.25	2.48	22.14
Displacement	10.95	8.93	10.78	10.78	7.77	10.75	11.72	11.43	11.24	12.91	9.19	6.79	17.10	3.97	2.70	5.31	9.52
Inversion	7.36	4.60	6.50	5.74	4.88	5.78	6.97	7.53	6.05	8.53	9.41	8.07	8.14	4.36	2.26	5.57	6.36
Insertion	24.08	30.63	25.24	26.93	11.06	18.29	13.74	26.05	25.71	18.62	14.92	10.29	29.36	2.91	1.70	2.89	17.65
Simple Inversion	5.98	7.66	7.58	8.04	4.35	6.22	7.47	5.48	6.43	8.21	9.31	7.23	7.86	4.60	1.57	7.60	6.60
Scramble	87.77	80.76	78.56	82.26	68.00	31.83	57.41	56.56	76.91	62.61	29.23	21.72	54.59	5.91	2.06	4.02	50.01
Greedy Swap	38.13	35.83	40.31	37.92	39.87	33.17	34.04	36.06	35.99	26.39	17.11	15.50	35.60	16.52	9.41	11.28	28.95
Double Bridge	41.00	20.03	44.29	24.21	12.12	10.86	7.39	41.38	53.95	26.04	11.09	11.70	62.01	4.03	1.14	3.70	23.43
Inver-Over	13.93	12.94	17.22	13.39	11.34	14.19	14.75	15.82	16.67	11.13	6.43	8.04	8.04	7.87	10.82	6.61	11.82
Inver-RNN	4.81	5.01	5.38	5.72	2.86	4.30	4.36	5.18	5.37	4.18	4.32	4.84	4.02	4.18	2.92	6.28	4.61
Greedy Sub Tour	5.72	4.07	3.93	6.03	4.94	2.38	4.64	2.15	3.53	4.66	3.53	3.26	4.05	4.80	3.09	5.14	**4.12**
Rasgele	4.12	5.83	5.42	3.89	5.91	4.84	4.24	4.71	5.42	4.23	4.67	4.25	6.00	5.44	2.40	4.71	4.76
Ortalama (Çaprazlama)	23.04	20.29	23.29	21.29	15.65	13.84	15.63	20.11	22.73	18.20	11.61	10.13	22.82	5.75	**3.44**	5.47	

Ek-Tablo 8. Eil76 için Çaprazlama-Mutasyon Çifti Ortalama Hata Değerleri

Çaprazlama / Mutasyon	Tek N.	İki N.	Çok N.	PMX	CX	OX	MOX	OBX	PBX	GSX	AEX	ERX	HX	HX2	DPX	Rasgele	Ortalama (Mutasyon)
Exchange	38.34	35.89	36.65	39.81	17.11	22.37	23.38	41.24	31.44	34.85	25.09	25.42	37.19	3.38	0.76	4.10	26.06
Displacement	13.47	12.34	11.37	10.22	8.79	13.32	13.42	13.62	11.85	17.34	12.15	10.76	17.99	6.93	3.27	4.62	11.34
Inversion	7.52	8.64	9.40	7.73	6.91	8.15	8.77	9.88	6.56	11.11	10.63	9.25	11.46	5.31	1.82	3.95	7.94
Insertion	25.31	25.24	28.99	28.88	14.29	18.02	11.82	26.37	28.53	24.14	13.81	14.03	32.71	4.01	1.56	4.10	18.86
Simple Inversion	7.97	7.37	7.75	7.65	5.55	7.32	7.37	9.64	6.78	6.44	8.60	7.63	7.39	5.18	1.44	5.05	6.82
Scramble	107.0	105.3	98.66	119.3	98.82	68.75	81.52	84.16	114.7	93.90	53.97	49.70	90.94	8.21	1.69	4.16	73.80
Greedy Swap	45.05	47.17	48.08	49.62	45.91	46.02	40.18	45.22	45.39	42.36	19.57	21.20	47.37	20.44	7.04	14.10	36.55
Double Bridge	49.29	36.33	85.89	53.38	11.89	11.28	12.91	62.69	70.18	29.10	16.82	14.49	93.99	4.34	1.39	4.05	34.88
Inver-Over	21.78	19.12	20.52	20.83	13.38	19.42	20.89	14.64	19.38	17.89	8.21	8.38	10.70	8.84	9.90	7.08	15.06
Inver-RNN	4.16	5.59	4.85	4.38	4.08	4.46	4.57	5.22	3.75	4.96	5.53	4.20	4.44	3.73	2.45	4.77	**4.45**
Greedy Sub Tour	4.40	4.29	5.18	4.57	4.20	4.94	4.90	4.29	4.62	4.94	4.90	4.14	5.01	4.72	2.78	3.71	4.47
Rasgele	5.70	4.81	4.44	4.83	4.72	4.62	4.70	4.98	5.09	6.11	5.20	4.62	5.81	3.47	2.26	4.47	4.74
Ortalama (Çaprazlama)	27.50	26.01	30.15	29.27	19.64	19.06	19.54	26.83	29.02	24.43	15.37	14.49	30.42	6.55	**3.03**	5.35	

Ek-Tablo 9. KroB200 için Çaprazlama-Mutasyon Çifti Ortalama Hata Değerleri

Çaprazlama / Mutasyon	Tek N.	İki N.	Çok N.	PMX	CX	OX	MOX	OBX	PBX	GSX	AEX	ERX	HX	HX2	DPX	Rasgele	Ortalama (Mutasyon)
Exchange	114.8	116.0	121.6	125.1	68.55	101.7	96.40	126.5	117.3	119.8	82.09	84.14	126.3	8.13	4.30	6.81	88.72
Displacement	35.89	31.47	37.49	36.19	23.35	37.87	37.79	39.96	28.19	55.98	42.92	31.60	62.94	22.58	7.92	10.33	33.90
Inversion	24.48	23.61	30.06	26.99	14.85	23.42	24.16	27.91	20.06	38.03	36.79	29.77	40.52	13.40	5.52	9.54	24.32
Insertion	54.03	59.88	66.10	65.32	43.35	42.47	39.10	50.22	58.76	60.29	51.88	45.11	67.28	6.93	3.81	8.14	45.17
Simple Inversion	11.05	12.70	12.73	10.79	10.03	12.39	10.77	11.63	10.46	13.41	12.95	10.68	14.01	9.93	6.80	10.17	11.28
Scramble	329.2	338.3	307.9	334.9	325.7	274.2	283.5	262.8	331.7	306.1	228.9	164.1	282.9	36.79	3.54	10.80	238.8
Greedy Swap	123.0	115.9	120.1	127.8	123.5	129.4	128.6	138.8	127.6	118.8	100.4	86.94	124.8	45.54	17.24	30.76	103.7
Double Bridge	166.0	121.2	274.9	184.9	38.15	31.09	36.29	184.2	215.7	112.7	83.80	44.74	274.1	9.60	4.92	8.66	111.9
Inver-Over	39.70	39.71	35.80	36.90	26.38	38.60	38.39	27.08	37.66	42.10	24.82	15.63	24.63	17.84	21.77	13.83	30.05
Inver-RNN	8.68	8.19	7.59	8.71	7.26	7.69	6.18	8.12	8.71	8.49	8.32	6.79	8.18	7.29	5.58	7.09	7.68
Greedy Sub Tour	6.43	5.91	7.36	7.17	7.26	7.18	7.06	6.31	6.61	8.41	6.71	6.01	7.66	6.01	6.56	5.37	**6.75**
Rasgele	10.20	8.84	9.24	8.86	7.84	8.76	9.67	9.28	8.16	10.30	8.86	8.71	11.12	6.74	4.92	8.20	8.73
Ortalama (Çaprazlama)	76.96	73.48	85.91	81.14	58.02	59.56	59.83	74.40	80.91	74.53	57.37	44.52	87.04	15.90	**7.74**	10.81	

Ek-Tablo 10. Berlin52&KroA150 için Seçim-Eşleştirme Çifti Ortalama Hataları [*Konf-2*]

Problem		**Berlin52**							**KroA150**						
Seçim Metodu / Eşleştirme Metodu		Rasgele	Rulet Tekerleği	Turnuva	Sıralama	Ağırlıklı	Karışık	Ortalamalar	Rasgele	Rulet Tekerleği	Turnuva	Sıralama	Ağırlıklı	Karışık	Ortalamalar
Rasgele	ε_{ORT} (%)	2.873	3.805	3.835	2.879	5.487	3.406	**3.714**	4.316	4.697	5.921	5.176	5.956	5.694	**5.293**
	t_{ORT} (*ms*)	3018	2883	2196	2685	1951	2178	2485	8293	7786	6740	7254	5562	6220	6976
Yukarıdan Aşağıya	ε_{ORT} (%)	4.063	5.037	4.824	4.180	4.215	4.854	4.529	6.773	5.603	5.926	7.021	7.018	5.217	6.260
	t_{ORT} (*ms*)	4009	2953	2297	2431	1995	2060	2624	11929	8300	6170	6195	5093	5959	7274
Sıra Ağırlıklı	ε_{ORT} (%)	2.776	3.280	4.280	5.584	4.329	4.530	4.130	5.916	5.677	5.971	5.584	6.796	4.944	5.815
	t_{ORT} (*ms*)	2561	2467	2825	2629	1996	2217	2449	6548	5886	7553	6534	5342	6206	6345
Değer Ağırlıklı	ε_{ORT} (%)	4.469	3.869	4.135	3.968	4.209	2.951	3.934	5.904	4.540	5.104	5.548	8.107	5.082	5.714
	t_{ORT} (*ms*)	3313	2420	2281	2606	2311	2559	2582	10028	8750	6884	6546	4696	6249	7192
En İyi İle En Kötü	ε_{ORT} (%)	3.408	3.038	5.465	5.144	5.644	5.143	4.640	4.610	5.275	7.164	5.060	6.350	7.026	5.914
	t_{ORT} (*ms*)	3062	2926	2317	2323	2346	2265	2540	7986	7036	5981	6757	5392	6120	6545
İmparator Seçilimi	ε_{ORT} (%)	4.497	3.800	5.230	5.247	2.802	3.858	4.239	5.586	5.805	5.690	6.840	7.543	5.524	6.165
	t_{ORT} (*ms*)	3671	3167	2040	2298	1935	2356	2578	8581	8542	6168	6126	4054	6123	6599
Karışık	ε_{ORT} (%)	5.790	5.957	4.087	4.082	5.132	5.031	5.013	5.225	4.678	6.559	6.136	4.203	5.601	5.400
	t_{ORT} (*ms*)	3873	3329	2664	2601	2132	2237	2806	8778	7459	6659	5117	4948	4662	6271
Ortalama	ε_{ORT} (%)	**3.982**	4.112	4.551	4.441	4.545	4.253		5.476	**5.182**	6.048	5.909	6.568	5.584	
	t_{ORT} (*ms*)	3358	2878	2374	2510	2095	2267		8878	7680	6594	6361	5012	5934	

Ek-Tablo 11. Ts225&Pcb443 için Seçim-Eşleştirme Çifti Ortalama Hataları [*Konf*-2]

Problem		**Ts225**							**Pcb443**						
Seçim Metodu / Eşleştirme Metodu		Rasgele	Rulet Tekerleği	Turnuva	Sıralama	Ağırlıklı	Karışık	Ortalamalar	Rasgele	Rulet Tekerleği	Turnuva	Sıralama	Ağırlıklı	Karışık	Ortalamalar
Rasgele	ε_{ORT} (%)	7.469	6.031	7.984	7.827	8.367	8.321	7.667	8.257	8.839	8.815	9.928	11.01	8.440	9.215
	t_{ORT} (*ms*)	12165	9986	9412	8892	7595	8679	9455	41756	30821	27154	23373	18304	22396	27301
Yukarıdan Aşağıya	ε_{ORT} (%)	7.598	7.603	7.145	8.392	7.832	6.328	7.483	8.435	7.765	8.950	9.381	11.00	9.430	9.160
	t_{ORT} (*ms*)	15248	10976	10431	8350	8331	8420	10293	54979	38296	23354	23125	21317	21898	30495
Sıra Ağırlıklı	ε_{ORT} (%)	8.632	7.821	6.118	7.142	8.561	6.799	7.512	8.318	9.307	9.163	9.586	9.905	9.542	9.304
	t_{ORT} (*ms*)	9843	8743	8662	10614	8332	9126	9220	27475	25550	23418	26468	22649	17823	23897
Değer Ağırlıklı	ε_{ORT} (%)	5.824	7.956	7.584	6.159	8.395	8.509	7.405	9.109	8.208	9.457	9.490	9.799	8.665	9.121
	t_{ORT} (*ms*)	13390	9258	9420	8607	7415	8200	9382	42999	32815	24375	22906	19942	24074	27852
En İyi İle En Kötü	ε_{ORT} (%)	7.189	7.547	7.130	6.721	7.644	6.087	**7.053**	8.294	7.806	8.181	9.781	10.89	8.742	8.949
	t_{ORT} (*ms*)	11070	9985	9095	9329	8026	10090	9599	35289	32257	25684	21596	18918	22567	26052
İmparator Seçilimi	ε_{ORT} (%)	7.682	7.666	7.177	9.056	8.266	8.021	7.978	8.187	8.258	8.522	9.017	7.678	8.402	**8.344**
	t_{ORT} (*ms*)	12640	11151	8781	7879	9362	7721	9589	37289	27960	27286	23643	17042	22982	26034
Karışık	ε_{ORT} (%)	8.645	6.199	7.567	8.390	8.136	7.446	7.731	8.261	7.968	8.970	9.297	9.852	9.142	8.915
	t_{ORT} (*ms*)	11440	10249	8281	8695	8107	10023	9466	35715	32322	23136	24879	17435	17037	25087
Ortalama	ε_{ORT} (%)	7.577	7.260	**7.244**	7.670	8.172	7.359		8.409	**8.307**	8.865	9.497	10.02	8.909	
	t_{ORT} (*ms*)	12257	10050	9155	8909	8167	8894		39357	31432	24915	23713	19372	21254	

Ek-Tablo 12. Yerel Arama Metotlarının Çözümdeki Etkisi ($CL_{NN\text{-}MAX}$ = 50. 1 İterasyon) [*Konf*-2]

Problem	2-Opt (Standart)			3-Opt (Standart)			2-Opt (CL+FRNNS+DLB)			2.5-Opt (CL+FRNNS+DLB)			3-Opt (CL+FRNNS+DLB)			Lin-Kernighan (CL)		
	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*ms*)
Berlin52	0.000	7.146	12	0.000	3.431	119	0.000	5.105	5	0.000	3.548	5	0.000	1.768	6	0.000	6.078	7
Eil76	2.973	7.725	15	0.185	2.788	525	1.115	5.141	5	1.115	3.773	5	0.185	2.100	8	0.371	3.267	9
Lin105	1.091	5.289	26	0.000	1.320	2229	0.557	5.215	8	0.827	3.702	9	0.000	1.052	10	4.263	8.996	11
KroA150	3.155	7.115	51	0.674	2.378	9942	1.263	4.585	11	1.308	3.840	11	0.098	1.753	15	3.755	8.261	19
KroB200	5.693	8.401	96	2.231	3.194	32744	1.664	6.722	15	2.469	5.284	15	0.428	2.516	21	5.183	9.246	28
Ts225	1.384	7.268	128	1.265	4.456	50911	0.751	5.096	15	1.295	4.867	16	0.000	1.965	19	5.996	13.96	24
Pr264	4.898	9.427	227	0.138	3.659	123495	1.896	6.765	18	0.860	5.431	18	0.229	2.730	26	13.03	19.43	51
Lin318	4.918	7.386	469	2.225	2.700	630519	4.256	6.458	23	3.204	5.078	23	0.918	2.672	35	7.183	11.87	51
Pcb442	7.044	9.894	2786	2.483	3.880	4043451	3.568	6.064	32	2.965	4.846	32	1.599	2.735	44	6.217	9.120	74
Rat783	9.732	11.37	17872	-	-	∞	4.973	6.897	63	3.667	5.053	61	2.838	3.488	109	5.098	7.522	203
Vm1084	7.640	9.335	64697	-	-	∞	5.089	7.077	99	3.603	5.438	100	2.156	3.265	268	11.87	17.54	412
Pcb1173	10.52	11.83	69469	-	-	∞	6.521	7.716	93	4.831	6.154	94	3.181	4.258	181	6.781	10.08	429
U2319	7.076	7.548	951744	-	-	∞	1.833	2.093	206	1.377	1.732	209	0.975	1.221	293	1.453	2.051	1079
Pcb3038	11.60	12.50	3463685	-	-	∞	6.438	7.597	320	5.141	5.833	332	3.368	4.313	749	7.266	10.36	2570
Ortalamalar	5.552	8.731		**1.022**	3.090		2.852	5.895		2.333	4.613		1.141	**2.560**		5.605	9.842	

Ek-Tablo 13. Genetik Algoritmada Yerel Arama Metotlarının Çözümdeki Etkisi ($CL_{QN\text{-}MAX}$ = 20) [*Konf*-2]

Problem	2-Opt (CL+FRNNS+DLB)			2.5-Opt (CL+FRNNS+DLB)			3-Opt (CL+FRNNS+DLB)			Lin-Kernighan (CL)			Karışık Olarak Uygula		
	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)
Berlin52	0.000	0.000	0.345	0.000	0.000	0.323	0.000	0.000	0.463	0.000	0.000	0.907	0.000	0.000	0.504
Eil76	0.000	0.000	0.582	0.000	0.000	0.465	0.000	0.000	0.570	0.000	0.000	1.426	0.000	0.000	0.657
Lin105	0.000	0.097	0.579	0.000	0.007	0.258	0.000	0.000	0.698	0.000	0.000	2.388	0.000	0.000	0.951
KroA150	0.003	0.461	1.048	0.003	0.299	1.056	0.003	0.004	1.371	0.003	0.176	5.997	0.000	0.003	2.145
KroB200	0.084	0.500	2.195	0.030	0.417	1.387	0.000	0.007	2.231	0.000	0.044	10.684	0.000	0.001	3.314
Ts225	0.055	0.211	1.106	0.000	0.053	1.159	0.000	0.000	1.343	0.000	0.312	15.850	0.000	0.000	2.604
Pr264	0.000	0.053	1.140	0.000	0.063	0.795	0.000	0.000	1.690	0.000	0.006	14.459	0.000	0.000	5.686
Lin318	0.399	0.932	3.743	0.147	0.624	2.634	0.000	0.344	4.403	0.145	0.414	37.965	0.000	0.304	9.013
Pcb442	0.710	1.168	4.104	0.685	1.042	3.649	0.132	0.465	6.685	0.120	0.369	57.951	0.011	0.313	21.685
Rat783	2.168	2.636	18.564	1.249	2.019	18.159	0.261	0.702	51.193	0.533	0.645	166.13	0.113	0.442	57.689
Vm1084	0.781	1.299	34.640	0.822	1.035	34.383	0.070	0.217	121.30	0.134	0.213	362.04	0.117	0.231	160.69
Pcb1173	1.810	2.320	40.032	1.255	1.964	36.940	0.446	0.786	71.223	0.330	0.540	334.96	0.181	0.541	122.81
U2319	1.282	1.660	47.112	0.904	1.025	49.114	0.427	0.509	53.232	0.347	0.402	116.51	0.287	0.365	347.23
Pcb3038	2.768	3.390	504.15	2.466	2.944	438.07	0.809	1.176	1012.4	0.745	1.023	3530.3	0.615	0.948	1715.9
Ortalamalar	0.719	1.052		0.540	0.821		0.153	0.301		0.168	0.296		**0.095**	**0.225**	

Ek-Tablo 14. Komşu Listesi Büyüklüğünün Çözümdeki Etkisi (LK + CL_{NN}) [*Konf*-2]

Problem	$CL_{NN\text{-}MAX}$ = 6			$CL_{NN\text{-}MAX}$ = 10			$CL_{NN\text{-}MAX}$ = 20			$CL_{NN\text{-}MAX}$ = 30			$CL_{NN\text{-}MAX}$ = 40			$CL_{NN\text{-}MAX}$ = 50		
	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)
Berlin52	0.000	0.000	0.701	0.000	0.000	1.046	0.000	0.000	1.178	0.000	0.000	1.299	0.000	0.000	1.478	0.000	0.000	1.654
Eil76	0.000	0.000	1.025	0.000	0.000	1.109	0.000	0.000	1.282	0.000	0.000	1.412	0.000	0.000	1.556	0.000	0.000	1.634
Lin105	0.000	0.046	2.301	0.000	0.041	2.001	0.000	0.034	3.656	0.000	0.000	3.378	0.000	0.030	3.845	0.000	0.032	4.062
KroA150	0.398	0.826	5.181	0.033	0.418	6.267	0.033	0.375	8.489	0.033	0.424	11.65	0.207	0.475	10.48	0.388	0.572	9.454
KroB200	0.027	0.208	10.83	0.003	0.170	8.462	0.003	0.037	11.02	0.003	0.038	12.38	0.003	0.033	12.51	0.033	0.039	14.58
Ts225	0.286	0.717	9.106	0.277	0.698	10.95	0.724	1.016	9.303	0.719	1.013	13.92	0.472	1.072	15.52	0.453	0.891	24.31
Pr264	0.000	1.227	15.99	0.000	0.152	11.02	0.000	0.055	16.73	0.000	0.119	22.88	0.000	0.169	35.40	0.000	0.074	45.25
Lin318	0.756	2.039	33.96	0.161	0.788	28.86	0.176	0.511	34.43	0.280	0.683	38.11	0.176	0.502	54.84	0.166	0.599	56.01
Pcb442	0.397	0.837	38.92	0.151	0.785	42.45	0.196	0.637	50.80	0.076	0.563	59.37	0.088	0.657	51.42	0.161	0.583	75.91
Rat783	0.124	0.785	139.2	0.340	0.550	155.8	0.261	0.571	251.6	0.647	0.884	171.6	0.511	0.871	207.3	0.340	0.705	374.8
Vm1084	1.162	3.811	655.0	1.192	1.570	532.6	0.537	0.884	722.6	0.473	0.992	573.4	0.628	1.166	631.0	0.823	1.227	752.7
Pcb1173	0.747	1.334	357.4	0.214	0.897	314.0	0.485	1.090	375.6	0.500	1.133	609.0	0.509	1.003	585.9	0.421	1.120	561.9
U2319	0.217	0.316	574.3	0.242	0.355	796.1	0.252	0.333	864.0	0.322	0.369	962.1	0.252	0.357	977.5	0.252	0.343	1301
Ortalamalar	0.316	0.934	**141.8**	**0.201**	0.494	147.0	0.205	**0.426**	180.8	0.235	0.478	190.8	0.219	0.487	199.1	0.234	0.476	247.9

Ek-Tablo 15. Komşu Listesi Büyüklüğünün Çözümdeki Etkisi (LK + CL_{QN}) [*Konf*-2]

Problem	$CL_{QN\text{-}MAX}$ = 6			$CL_{QN\text{-}MAX}$ = 10			$CL_{QN\text{-}MAX}$ = 20			$CL_{QN\text{-}MAX}$ = 30			$CL_{QN\text{-}MAX}$ = 40			$CL_{QN\text{-}MAX}$ = 50		
	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)	ε_{iyi} (%)	ε_{ORT} (%)	t_{ORT} (*s*)
Berlin52	0.000	0.000	0.407	0.000	0.000	0.479	0.000	0.000	0.472	0.000	0.000	0.681	0.000	0.000	0.815	0.000	0.000	0.873
Eil76	0.000	0.000	0.701	0.000	0.000	0.681	0.000	0.000	0.815	0.000	0.000	0.842	0.000	0.000	0.826	0.000	0.000	0.884
Lin105	0.000	0.132	1.098	0.000	0.000	1.120	0.000	0.000	1.015	0.000	0.000	1.200	0.000	0.000	1.193	0.000	0.000	1.413
KroA150	0.000	0.144	2.231	0.000	0.054	5.656	0.000	0.084	5.408	0.003	0.102	5.984	0.003	0.055	5.186	0.015	0.158	6.512
KroB200	0.000	0.109	10.28	0.000	0.152	9.429	0.000	0.031	9.689	0.000	0.062	10.93	0.000	0.069	11.75	0.000	0.053	12.21
Ts225	0.387	0.748	6.332	0.221	0.589	7.225	0.000	0.178	9.312	0.000	0.097	9.260	0.000	0.067	12.49	0.000	0.059	10.76
Pr264	0.000	0.172	9.323	0.000	0.237	12.43	0.000	0.074	13.65	0.000	0.138	16.66	0.000	0.161	25.77	0.000	0.326	21.28
Lin318	0.680	1.384	27.79	0.173	0.817	28.24	0.054	0.638	24.53	0.349	0.575	33.17	0.116	0.572	36.86	0.000	0.664	39.54
Pcb442	0.348	0.653	43.33	0.011	0.503	50.24	0.037	0.459	66.78	0.202	0.532	40.48	0.029	0.466	55.49	0.055	0.466	55.46
Rat783	0.647	0.957	129.8	0.238	0.647	143.1	0.363	0.629	179.5	0.158	0.587	161.1	0.317	0.656	188.8	0.352	0.646	184.4
Vm1084	0.999	1.591	405.1	0.341	1.148	356.0	0.387	0.953	261.8	0.414	0.764	363.0	0.266	0.739	475.4	0.376	0.717	368.9
Pcb1173	0.557	1.147	297.4	0.546	1.007	410.1	0.133	0.955	347.5	0.585	1.024	384.5	0.428	0.700	544.2	0.720	0.983	349.7
U2319	0.217	0.305	1035	0.259	0.313	915.1	0.287	0.337	881.9	0.252	0.337	1029	0.217	0.330	936.8	0.217	0.320	886.2
Ortalamalar	0.295	0.565	151.4	0.138	0.421	149.2	**0.097**	0.334	138.6	0.151	0.324	158.2	0.106	**0.293**	176.6	0.133	0.338	149.1

Ek-Tablo 16. Aday Listesi Uygulanış Biçiminin Çözümdeki Etkisi [*Konf*-1]

Aday Listesi Uygulaması	Problem	Ortalama Çözümün Bulunma Süresi (*ms*)	Ortalama Çözümün Bulunma Jenerasyonu	En İyi Hata $\varepsilon_{İYİ}$ (%)	Ortalama Hata ε_{ORT} (%)
SABİT	Lin318	1315	489	0.27124	0.48443
	Pcb442	7050	2529	0.26586	0.30643
	Rat768	33772	3943	0.21576	0.41563
	Vm1084	106537	3079	0.03761	0.15855
Sabit – Ortalama		37169	2510	0.19762	0.34126
RASGELE	Lin318	2512	1340	0.14752	0.35594
	Pcb442	4969	2786	0.01182	0.34936
	Rat768	13021	2438	0.04542	0.44061
	Vm1084	47669	2290	0.00209	0.21588
Rasgele – Ortalama		**17043**	**2214**	**0.05171**	0.34045
ADAPTİF	Lin318	1897	870	0.12848	0.31026
	Pcb442	8834	3422	0.26586	0.29068
	Rat768	18587	2769	0.14763	0.36566
	Vm1084	76887	2847	0.04513	0.08216
Adaptif – Ortalama		26551	2477	0.14678	**0.26219**

Ek-Tablo 17. Pcb1173 TSP Problemi için PGA Parametre Analizi [*Konf*-3]

Göç Eden Birey Sayısı	Göç Sıklığı	Düğüm Sayısı=3		Düğüm Sayısı =6		Düğüm Sayısı =10		Düğüm Sayısı =15		Ortalama Hata
		ε_{iyi} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	t_{ORT} (*ms*)	ε_{iyi} (%)	t_{ORT} (*ms*)	ε_{ORT} (%)
2	5	46.928	48840	30.008	49233	21.478	48133	16.879	48545	28.8238
2	25	69.133	21608	58.107	20364	50.005	20150	42.698	20364	54.9858
2	50	90.318	16530	72.443	16396	63.142	16412	60.696	16263	71.6498
2	100	89.992	14619	79.551	14640	70.096	14522	69.556	14520	77.2988
4	5	42.314	48251	26.585	48477	19.562	48660	**15.413**	48923	**25.9685**
4	25	67.586	20880	58.154	20155	49.093	20583	41.719	20005	54.1380
4	50	80.625	16702	67.152	16399	60.949	16356	56.736	16150	66.3655
4	100	93.344	14577	76.977	14467	75.710	14598	65.883	14464	77.9785
8	5	57.600	48048	29.092	48180	23.923	48570	17.604	48063	32.0548
8	25	73.803	20677	58.249	20495	49.190	20962	43.614	20850	56.2140
8	50	82.688	16733	74.460	16471	59.432	16487	57.266	16350	68.4615
8	100	91.484	14419	84.650	14615	71.527	14788	68.697	14487	79.0895
Ortalama Hata		73.81792	-	59.619	-	51.17558	-	**46.39675**	-	

Ek-Tablo 18. Standart Genetik Algoritma Sonuç Değerleri – 1

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% $\varepsilon_{İyi}$)	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Ortalama Süre (*ms*)
1	Eil51	426	427	0.23474	431.50	1.29108	6929	3740
2	Berlin52	7542	7542	0.00000	7737.40	2.59082	6351	3367
3	St70	675	683	1.18519	699.80	3.67407	6346	6625
4	Pr76	108159	109938	1.64480	113767.10	5.18505	10864	4714
5	Eil76	538	546	1.48699	553.70	2.91822	5918	3570
6	Rat99	1211	1229	1.48637	1250.20	3.23699	14036	5740
7	KroB100	22141	22296	0.70006	22452.20	1.40554	8508	4256
8	KroE100	22068	22130	0.28095	22450.40	1.73283	11886	5044
9	KroD100	21294	21822	2.47957	22240.30	4.44397	15265	5833
10	KroA100	21282	22056	3.63688	22410.50	5.30260	7520	4031
11	KroC100	20749	20753	0.01928	21086.40	1.62610	12574	5160
12	Eil101	629	632	0.47695	647.80	2.98887	20146	7257
13	Lin105	14379	14379	0.00000	14610.50	1.60999	11003	4798
14	Pr107	44303	44526	0.50335	44679.20	0.84915	10613	5009
15	Pr124	59030	59490	0.77926	60138.40	1.87769	16829	6270
16	Ch130	6110	6313	3.32242	6432.90	5.28478	14053	5659
17	Pr136	96772	102438	5.85500	104748.90	8.24298	22318	7617
18	Pr144	58537	58780	0.41512	60396.30	3.17628	24437	8224
19	Ch150	6528	6559	0.47488	6621.20	1.42770	14976	6037
20	KroA150	26524	27209	2.58257	27658.60	4.27764	20078	7245
21	KroB150	26130	27081	3.63949	27289.20	4.43628	27865	9211
22	Pr152	73682	74017	0.45466	75285.90	2.17679	20243	7342
23	U159	42080	44102	4.80513	45953.80	9.20580	20573	7896
24	Rat195	2323	2348	1.07619	2384.80	2.66035	24756	9321
25	D198	15780	16115	2.12294	16301.10	3.30228	30468	10173
26	KroB200	29437	30109	2.28284	30850.00	4.80008	31560	10426
27	KroA200	29368	29884	1.75701	30439.70	3.64921	29715	10042
28	Ts225	126643	127032	0.30716	129182.40	2.00516	20496	8496
29	Tsp225	3919	3996	1.96479	4079.50	4.09543	26812	10231
30	Pr226	80369	81382	1.26044	82999.50	3.27303	28372	9927
31	Gil262	2378	2469	3.82675	2518.70	5.91674	32144	11279
32	Pr264	49135	50139	2.04335	52560.50	6.97161	37864	12732

Ek-Tablo 19. Standart Genetik Algoritma Sonuç Değerleri – 2

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% $\varepsilon_{İyi}$)	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Ortalama Süre (*ms*)
33	Pr299	48191	49667	3.06281	50472.40	4.73408	43427	16056
34	Lin318	42029	43503	3.50710	44400.30	5.64206	38857	13684
35	Pr439	107217	111749	4.22694	114337.10	6.64083	55812	22850
36	Pcb442	50778	52021	2.44791	52665.10	3.71637	57678	21107
37	D493	35002	36108	3.15982	36696.80	4.84201	72845	30606
38	U574	36905	38383	4.00488	39400.20	6.76114	71525	28412
39	Rat575	6773	7070	4.38506	7112.80	5.01698	84407	37721
40	P654	34643	35465	2.37277	36299.40	4.78134	113965	45687
41	D657	48912	51238	4.75548	51877.90	6.06375	108742	50487
42	U724	41910	43784	4.47149	44288.60	5.67550	110157	46514
43	Rat783	8806	9181	4.25846	9321.30	5.85169	115151	59292
44	Pr1002	259045	273169	5.45233	275581.20	6.38352	146262	72659
45	U1060	224094	237301	5.89351	242112.00	8.04038	179663	89656
46	Vm1084	239297	255339	6.70380	257124.80	7.45007	143828	73784
47	Pcb1173	56892	61201	7.57400	61540.40	8.17057	160032	104193
48	D1291	50801	54545	7.36993	55285.00	8.82660	113690	80359
49	Rl1304	252948	273371	8.07399	280167.80	10.76103	173839	100006
50	Rl1323	270199	286592	6.06701	291467.40	7.87138	149411	105484
51	Nrw1379	56638	59772	5.53339	60028.00	5.98538	199844	145290
52	U1432	152970	161107	5.31934	162294.40	6.09557	199623	121006
53	D1655	62128	65280	5.07340	67126.80	8.04597	159929	131199
54	Vm1748	336556	355412	5.60263	362802.60	7.79858	231158	159593
55	U1817	57201	61441	7.41246	62170.80	8.68831	190598	164456
56	Rl1889	316536	343188	8.41990	349958.40	10.55880	160399	146825
57	U2152	64253	69664	8.42140	70718.00	10.06179	173267	141756
58	U2319	234256	241544	3.11113	242263.40	3.41823	204048	219965
59	Pr2392	378032	403423	6.71663	410504.20	8.58980	290134	251575
60	Pcb3038	137694	147482	7.10852	147776.60	7.32247	400762	448203
61	Fnl4461	182566	192928	5.67576	193918.60	6.21835	587598	1252393
62	Rl5915	565530	614558	8.66939	624725.40	10.46724	443246	1582569
63	Rl5934	556045	608681	9.46614	617487.20	11.04986	515852	1647234
Ortalama Hatalar				**3.60988**		**5.35182**		

Ek-Tablo 20. Paralel Genetik Algoritma Sonuç Değerleri – 1

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% ε_{iyi})	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Gerçek Ortalama Süre (*ms*)	Yaklaşık Ortalama Süre (*ms*)
1	Eil51	426	427	0.23474	427.00	0.23474	6210	7389	493
2	Berlin52	7542	7542	0.00000	7590.00	0.63644	4218	3630	242
3	St70	675	677	0.29630	682.20	1.06667	12794	11116	741
4	Pr76	108159	109768	1.48762	110563.00	2.22265	16998	15602	1040
5	Eil76	538	542	0.74349	544.50	1.20818	14994	13206	880
6	Rat99	1211	1212	0.08258	1225.00	1.15607	14906	16274	1085
7	KroB100	22141	22246	0.47423	22265.00	0.56005	16984	19869	1325
8	KroE100	22068	22174	0.48033	22240.50	0.78167	15821	17120	1141
9	KroD100	21294	21539	1.15056	21572.00	1.30553	24510	24130	1609
10	KroA100	21282	21379	0.45578	21414.50	0.62259	11906	16444	1096
11	KroC100	20749	20769	0.09639	20943.50	0.93739	24539	23571	1571
12	Eil101	629	630	0.15898	634.00	0.79491	11607	11271	751
13	Lin105	14379	14379	0.00000	14406.50	0.19125	13592	12373	825
14	Pr107	44303	44438	0.30472	444545.50	0.54737	13085	16579	1105
15	Pr124	59030	59030	0.00000	59464.50	0.73607	22530	21372	1425
16	Ch130	6110	6251	2.30769	6348.50	3.90344	28212	25183	1679
17	Pr136	96772	97609	0.86492	101836.00	5.23292	27125	26909	1794
18	Pr144	58537	58570	0.05637	58666.50	0.22123	33081	31697	2113
19	Ch150	6528	6553	0.38297	6592.20	0.98346	23858	23578	1572
20	KroA150	26524	26946	1.59101	27227.50	2.65231	23845	23981	1599
21	KroB150	26130	26812	2.61003	27040.20	3.48335	21000	22091	1473
22	Pr152	73682	73818	0.18458	74555.00	1.18482	21833	22704	1514
23	U159	42080	42080	0.00000	44338.50	5.36716	17964	19943	1330
24	Rat195	2323	2346	0.99010	2356.60	1.44641	21123	23946	1596
25	D198	15780	15865	0.53866	15899.80	0.75919	29782	31789	2119
26	KroB200	29437	30011	1.94993	30159.50	2.45439	33147	34840	2323
27	KroA200	29368	29776	1.38927	29952.60	1.99060	33741	35197	2346
28	Ts225	126643	126810	0.13187	127508.00	0.68302	22916	30480	2032
29	Tsp225	3919	3958	0.99515	3992.50	1.87548	38695	41208	2747
30	Pr226	80369	81299	1.15716	81626.20	1.56428	26015	30722	2048
31	Gil262	2378	2420	1.76619	2437.50	2.50210	40804	44729	2982
32	Pr264	49135	50424	2.62338	51127.80	4.05576	29126	37306	2487

Ek-Tablo 21. Paralel Genetik Algoritma Sonuç Değerleri – 2

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% $\varepsilon_{İyi}$)	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Gerçek Ortalama Süre (*ms*)	Yaklaşık Ortalama Süre (*ms*)
33	Pr299	48191	48610	0.86946	49243.50	2.18402	48552	58563	3904
34	Lin318	42029	42736	1.68217	43090.00	2.52445	56631	70170	4678
35	Pr439	107217	108370	1.07539	110951.60	3.48322	71978	110781	7385
36	Pcb442	50778	51801	2.01465	52099.20	2.60191	53748	82689	5513
37	D493	35002	36017	2.89983	36232.70	3.51608	65598	109721	7315
38	U574	36905	38431	4.13494	38458.20	4.20864	115115	191611	12774
39	Rat575	6773	6977	3.01196	7012.00	3.52872	98438	171041	11403
40	P654	34643	34941	0.86020	35041.80	1.15117	112121	206635	13776
41	D657	48912	50666	3.58603	51140.00	4.55512	114904	222072	14805
42	U724	41910	43110	2.86328	43241.20	3.17633	109446	207611	13841
43	Rat783	8806	9145	3.84965	9153.50	3.94617	113298	266777	17785
44	Pr1002	259045	267677	3.33224	270022.20	4.23756	171858	446845	29790
45	U1060	224094	230593	2.90012	235638.20	5.15150	213568	566171	37745
46	Vm1084	239297	247866	3.58091	250798.20	4.80624	156409	379799	25320
47	Pcb1173	56892	59617	4.78978	59923.00	5.32764	208550	629239	41949
48	D1291	50801	52336	3.02159	53182.50	4.68790	135913	469534	31302
49	Rl1304	252948	268201	6.03009	271335.00	7.26908	174810	510591	34039
50	Rl1323	270199	279598	3.47855	281756.50	4.27740	146499	557879	37192
51	Nrw1379	56638	59273	4.65235	59492.80	5.04043	232144	869034	57936
52	U1432	152970	159279	4.12434	160512.50	4.93071	152414	570966	38064
53	D1655	62128	64322	3.53142	64558.00	3.91128	167502	744806	49654
54	Vm1748	336556	349329	3.79521	353409.80	5.00773	245378	1113734	74249
55	U1817	57201	59537	4.08384	59722.80	4.40866	202545	975982	65065
56	Rl1889	316536	331303	4.66519	333627.00	5.39939	229632	1103299	73553
57	U2152	64253	66719	3.83795	67162.20	4.52773	231385	889343	59290
58	U2319	234256	240837	2.80932	241182.00	2.95659	160005	868288	57886
59	Pr2392	378032	394396	4.32873	396465.00	4.87604	420693	1884036	125602
60	Pcb3038	137694	144793	5.15563	145018.00	5.31904	593136	2740838	182723
61	Fnl4461	182566	190791	4.50522	191757.50	5.03462	725137	5473697	364913
62	Rl5915	565530	597828	5.71110	600847.00	6.24494	593452	8491833	566122
63	Rl5934	556045	586472	5.47204	594314.00	6.88236	575132	8153927	543595
Ortalama Hatalar				**2.16124**		**2.99261**			

Ek-Tablo 22. Genetik Yerel Arama Algoritması Sonuç Değerleri – 1

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% $\varepsilon_{İyi}$)	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Ortalama Süre (*ms*)
1	Eil51	426	426	0.00000	426.00	0.00000	14	10
2	Berlin52	7542	7542	0.00000	7542.00	0.00000	2	1
3	St70	675	675	0.00000	675.00	0.00000	17	36
4	Pr76	108159	108159	0.00000	108159.00	0.00000	9	34
5	Eil76	538	538	0.00000	538.00	0.00000	3	5
6	Rat99	1211	1211	0.00000	1211.00	0.00000	53	114
7	KroB100	22141	22141	0.00000	22141.00	0.00000	32	106
8	KroE100	22068	22068	0.00000	22068.00	0.00000	23	69
9	KroD100	21294	21294	0.00000	21294.00	0.00000	62	213
10	KroA100	21282	21282	0.00000	21282.00	0.00000	6	24
11	KroC100	20749	20749	0.00000	20749.00	0.00000	9	33
12	Eil101	629	629	0.00000	629.00	0.00000	7	23
13	Lin105	14379	14379	0.00000	14379.00	0.00000	2	15
14	Pr107	44303	44303	0.00000	44303.00	0.00000	8	67
15	Pr124	59030	59030	0.00000	59030.00	0.00000	41	168
16	Ch130	6110	6110	0.00000	6110.00	0.00000	27	146
17	Pr136	96772	96772	0.00000	96772.00	0.00000	232	1246
18	Pr144	58537	58537	0.00000	58537.00	0.00000	3	37
19	Ch150	6528	6528	0.00000	6528.00	0.00000	37	184
20	KroA150	26524	26524	0.00000	26524.00	0.00000	232	1337
21	KroB150	26130	26130	0.00000	26130.00	0.00000	91	472
22	Pr152	73682	73682	0.00000	73682.00	0.00000	22	205
23	U159	42080	42080	0.00000	42080.00	0.00000	3	21
24	Rat195	2323	2323	0.00000	2323.00	0.00000	1009	5670
25	D198	15780	15780	0.00000	15780.00	0.00000	865	17199
26	KroB200	29437	29437	0.00000	29437.00	0.00000	1818	14654
27	KroA200	29368	29368	0.00000	29368.00	0.00000	156	1223
28	Ts225	126643	126643	0.00000	126643.00	0.00000	12	94
29	Tsp225	3919	3919	0.00000	3919.00	0.00000	133	1122
30	Pr226	80369	80369	0.00000	80369.00	0.00000	69	798
31	Gil262	2378	2378	0.00000	2378.00	0.00000	991	12162
32	Pr264	49135	49135	0.00000	49135.00	0.00000	13	178

Ek-Tablo 23. Genetik Yerel Arama Algoritması Sonuç Değerleri – 2 (* : Sonlandırma= En İyi Bireyin 5000 Jenerasyon Değişmemesi . CL_{MAX}=30)

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% $\varepsilon_{İyi}$)	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Ortalama Süre (*ms*)
33	Pr299	48191	48191	0.00000	48191.00	0.00000	1610	20624
34	Lin318	42029	42029	0.00000	42029.00	0.00000	2223	42323
35	Pr439	107217	107217	0.00000	107222.50	0.00513	5593	151845
36	Pcb442	50778	50778	0.00000	50785.50	0.01477	9831	156378
37	D493	35002	35002	0.00000	35015.00	0.03714	13666	548093
38	U574	36905	36905	0.00000	36918.70	0.03712	7809	359442
39	Rat575	6773	6774	0.01476	6776.30	0.04872	12438	383331
40	P654	34643	34643	0.00000	34643.80	0.00231	3848	264339
41	D657	48912	48930	0.03680	48979.10	0.13719	14661	713903
42	U724	41910	41939	0.06920	41957.20	0.11262	15378	941887
43	Rat783	8806	8806	0.00000	8812.40	0.07268	16888	863957
44	Pr1002	259045	259191	0.05636	259827.10	0.30192	30458	3260187
45	U1060	224094	224163	0.03079	224306.60	0.09487	21651	2929514
46	Vm1084	239297	239306	0.00376	239463.00	0.06937	12350	1512026
47	Pcb1173	56892	56915	0.04043	56941.00	0.08613	23355	2801187
48	D1291	50801	50801	0.00000	50844.50	0.08563	9926	884362
49	Rl1304	252948	252948	0.00000	252953.10	0.00202	2468	294373
50	Rl1323	270199	270226	0.00999	270279.50	0.02979	7235	788560
51	Nrw1379	56638	56801	0.28779	56831.00	0.34076	35741	6530051
52	U1432	152970	153065	0.06210	153279.30	0.20220	23119	1967315
53	D1655	62128	62139	0.01771	62191.70	0.10253	23336	3127914
54	Vm1748	336556	336701	0.04308	336937.90	0.11347	23067	6205723
55	U1817	57201	57227	0.04545	57245.60	0.07797	30723	3951302
56	Rl1889	316536	316876	0.10741	317413.00	0.27706	14326	7895068
57	U2152 *	64253	64450	0.30660	64529.80	0.43080	16481	2061200
58	U2319 *	234256	234765	0.21728	234994.60	0.31530	14613	2478625
59	Pr2392 *	378032	378972	0.24866	379264.50	0.32603	23622	7249115
60	Pcb3038 *	137694	138036	0.24838	138417.40	0.52537	50875	30018551
61	Fnl4461 *	182566	183654	0.59595	183938.80	0.75195	57024	88672037
62	Rl5915 *	565530	566814	0.22704	567566.00	0.36002	27433	34192851
63	Rl5934	556045	556277	0.04172	556626.50	0.10458	23081	110307984
Ortalama Hatalar				**0.04304**		**0.08040**		

Ek-Tablo 24. Paralel Genetik Yerel Arama Algoritması Sonuç Değerleri – 1

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% ε_{iyi})	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Gerçek Ortalama Süre (*ms*)	Yaklaşık Ortalama Süre (*ms*)
1	Eil51	426	426	0.00000	426.00	0.00000	14	40	3
2	Berlin52	7542	7542	0.00000	7542.00	0.00000	1	44	3
3	St70	675	675	0.00000	675.00	0.00000	13	81	5
4	Pr76	108159	108159	0.00000	108159.00	0.00000	7	101	7
5	Eil76	538	538	0.00000	538.00	0.00000	11	74	5
6	Rat99	1211	1211	0.00000	1211.00	0.00000	19	121	8
7	KroB100	22141	22141	0.00000	22141.00	0.00000	37	234	16
8	KroE100	22068	22068	0.00000	22068.00	0.00000	42	238	16
9	KroD100	21294	21294	0.00000	21294.00	0.00000	40	238	16
10	KroA100	21282	21282	0.00000	21282.00	0.00000	8	117	8
11	KroC100	20749	20749	0.00000	20749.00	0.00000	13	109	7
12	Eil101	629	629	0.00000	629.00	0.00000	16	121	8
13	Lin105	14379	14379	0.00000	14379.00	0.00000	2	93	6
14	Pr107	44303	44303	0.00000	44303.00	0.00000	9	140	9
15	Pr124	59030	59030	0.00000	59030.00	0.00000	16	183	12
16	Ch130	6110	6110	0.00000	6110.00	0.00000	68	480	32
17	Pr136	96772	96772	0.00000	96772.00	0.00000	352	2320	155
18	Pr144	58537	58537	0.00000	58537.00	0.00000	3	121	8
19	Ch150	6528	6528	0.00000	6528.00	0.00000	61	417	28
20	KroA150	26524	26524	0.00000	26524.00	0.00000	553	3562	237
21	KroB150	26130	26130	0.00000	26130.00	0.00000	58	507	34
22	Pr152	73682	73682	0.00000	73682.00	0.00000	21	402	27
23	U159	42080	42080	0.00000	42080.00	0.00000	4	136	9
24	Rat195	2323	2323	0.00000	2323.00	0.00000	854	4945	330
25	D198	15780	15780	0.00000	15780.00	0.00000	451	9449	630
26	KroB200	29437	29437	0.00000	29437.00	0.00000	981	8351	557
27	KroA200	29368	29368	0.00000	29368.00	0.00000	363	3437	229
28	Ts225	126643	126643	0.00000	126643.00	0.00000	8	242	16
29	Tsp225	3919	3919	0.00000	3919.00	0.00000	324	3492	233
30	Pr226	80369	80369	0.00000	80369.00	0.00000	210	3168	211
31	Gil262	2378	2378	0.00000	2378.00	0.00000	2543	27394	1826
32	Pr264	49135	49135	0.00000	49135.00	0.00000	6	378	25

Ek-Tablo 25. Paralel Genetik Yerel Arama Algoritması Sonuç Değerleri – 2 (* : CL_{MAX}=30)

Sıra No	Problem Adı	Optimum Mesafe	Bulunan En İyi Mesafe	Bulunan En İyi Hata Değeri (% $\varepsilon_{İyi}$)	Ortalama Mesafe	Ortalama Hata Değeri (% ε_{ORT})	Ortalama Jenerasyon	Gerçek Ortalama Süre (*ms*)	Yaklaşık Ortalama Süre (*ms*)
33	Pr299	48191	48191	0.00000	48191.00	0.00000	1606	21230	1415
34	Lin318	42029	42029	0.00000	42029.00	0.00000	2324	43656	2910
35	Pr439	107217	107217	0.00000	107217.00	0.00000	6930	235504	15700
36	Pcb442	50778	50778	0.00000	50783.20	0.01024	9439	303105	20207
37	D493	35002	35002	0.00000	35009.00	0.02000	14142	882714	58848
38	U574	36905	36905	0.00000	36914.00	0.02439	9463	570683	38046
39	Rat575	6773	6774	0.01476	6777.00	0.05906	11402	701472	46765
40	P654	34643	34643	0.00000	34643.00	0.00000	3631	283027	18868
41	D657	48912	48913	0.00204	48952.60	0.08301	20409	1409511	93967
42	U724	41910	41924	0.03340	41943.50	0.07993	11063	1169402	77960
43	Rat783	8806	8806	0.00000	8810.00	0.04542	8291	706543	47103
44	Pr1002	259045	259045	0.00000	259329.20	0.10971	15978	2464621	164308
45	U1060	224094	224113	0.00848	224204.00	0.04909	12680	3059968	203998
46	Vm1084	239297	239349	0.02173	239430.70	0.05587	10786	2158461	143897
47	Pcb1173	56892	56892	0.00000	56908.60	0.02918	30981	4784953	318997
48	D1291	50801	50842	0.08071	50868.50	0.13287	5298	1336742	89116
49	Rl1304	252948	252948	0.00000	252948.00	0.00000	11365	1426753	95117
50	Rl1323	270199	270226	0.00999	270226.00	0.00999	10233	3188372	212558
51	Nrw1379	56638	56679	0.07239	56762.00	0.21893	46075	9272571	618171
52	U1432	152970	153065	0.06210	153194.00	0.14643	29570	3204031	213602
53	D1655	62128	62140	0.01931	62176.20	0.07758	35890	5994144	399610
54	Vm1748	336556	336701	0.04308	336964.50	0.12138	25065	10278406	685227
55	U1817	57201	57246	0.07867	57281.00	0.13986	10866	2632000	175467
56	Rl1889	316536	316576	0.01264	316691.00	0.04897	26029	27153859	1810257
57	U2152 *	64253	64359	0.16497	64406.50	0.23890	28822	8728031	581869
58	U2319 *	234256	234683	0.18228	234781.40	0.22428	10036	4233190	282213
59	Pr2392 *	378032	378219	0.04947	378850.60	0.21654	49920	20471800	1364787
60	Pcb3038 *	137694	137998	0.22078	138208.20	0.37344	78645	53440249	3562683
Ortalama Hatalar				**0.01795**		**0.04192**			