EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

ÇOKLU GEZGİN SATICI PROBLEMİNİN ÇÖZÜMÜ İÇİN BİR ENİYİLEME KÜTÜPHANESİNİN TASARIMI VE GÖRSEL YAZILIM GELİŞTİRME ORTAMI İLE BİRLİKTE GERÇEKLEŞTİRİMİ

Utku CEVRE

Bilgisayar Mühendisliği Anabilim Dalı
Bilim Dalı Kodu : 619.01.00
Sunuş Tarihi : 01.08.2008

Tez Danışmanı : Yrd. Doç. Dr. Aybars UĞUR

BORNOVA – İZMİR

II

Utku CEVRE tarafından YÜKSEK LİSANS TEZİ olarak sunulan "**Çoklu Gezgin Satıcı Probleminin Çözümü İçin Bir Eniyileme Kütüphanesinin Tasarımı ve Görsel Yazılım Geliştirme Ortamı ile Birlikte Gerçekleştirimi**" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve **01.08.2008** tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		İmza
Jüri Başkanı	: Yrd. Doç. Dr. Aybars UĞUR	.....
Raportör Üye	: Yrd. Doç. Dr. Murat Osman ÜNALIR	.....
Üye	: Yrd. Doç. Dr. Korhan KARABULUT	.....

IV

ÖZET

ÇOKLU GEZGİN SATICI PROBLEMİNİN ÇÖZÜMÜ İÇİN BİR ENİYİLEME KÜTÜPHANESİNİN TASARIMI VE GÖRSEL YAZILIM GELİŞTİRME ORTAMI İLE BİRLİKTE GERÇEKLEŞTİRİMİ

CEVRE, Utku

Yüksek Lisans Tezi, Fen Bilimleri Enstitüsü
Tez Yönetici: Yrd. Doç. Dr. Aybars UĞUR
Ağustos 2008, 139 sayfa

Çoklu Gezgin Satıcı Problemi (ÇGSP), verilen belirli sayıda şehrin her biri ayrı bir satıcıya atanmak üzere m adet tura bölünerek en düşük maliyet ile dolaşılmasını hedefleyen karmaşık bir kombinasyonel eniyileme problemidir.

Bu tez projesinde, ÇGSP'nin çözümü için bir eniyileme kütüphanesi tasarlanmış ve görsel yazılım geliştirme ortamı ile birlikte gerçekleştirilmiştir. Kütüphanede, melez olarak da uygulanabilen Genetik Algoritmalar ve Yerel Eniyileme (2-opt ve 3-opt) yöntemlerine yer verilmiştir. Web tabanlı ortam, otomatik GSP/ÇGSP kodları da üretebilen etkileşimli bir grafik arayüz teşkil etmektedir. Kütüphane çeşitli TSPLIB verileri ile test edilmiş ve sonuçlar sunulmuştur. Projenin kullanıcılar açısından yararı belirtilmiştir.

Anahtar Sözcükler: Çoklu Gezgin Satıcı Problemi, Genetik Algoritmalar, Yerel Eniyileme, Yazılım Geliştirme.

ABSTRACT

DESIGN AND IMPLEMENTATION OF AN OPTIMIZATION LIBRARY WITH VISUAL SOFTWARE DEVELOPMENT ENVIRONMENT FOR THE SOLUTION OF MULTIPLE TRAVELING SALESMAN PROBLEM

CEVRE, Utku

MSc, Fen Bilimleri Enstitüsü

Supervisor: Asst. Prof. Dr. Aybars UĞUR

August 2008, 139 pages

Multiple Traveling Salesman Problem (MTSP) is a complex combinatorial optimization problem, which aims a given collection of cities to be traveled with minimum cost by dividing them into m tours, all of which are to be appointed to a different salesman.

In this thesis, an optimization library was designed and implemented with visual software development environment for the solution of Multiple Traveling Salesman Problem. The library contains Genetic Algorithms and Local Optimization (2-opt and 3-opt) methods which can be applied as hybrid. Prepared Web based environment forms an interactive GUI which can also produce automatic TSP/MTSP codes. The library was tested with a variety of TSPLIB instances and results presented. The benefits of the project for users were mentioned.

Keywords: Multiple Traveling Salesman Problem, Genetic Algorithms, Local Optimization, Software Development.

TEŞEKKÜR

Tez çalışmam süresince bilgisini esirgemeyerek daima destek olan değerli hocam Yrd. Doç. Dr. Aybars UĞUR'a; gösterdikleri sabır ve fedakarlık, paylaştıkları pozitif düşünceler ile her zaman yanımda olan sevgili anne ve babama; yola birlikte çıktığımız kıymetli dostum ve ortağım Barış ÖZKAN'a; üzerimde emeği olan tüm hocalarıma ve muhabbetlerini paylaşan dostlarıma gönülden teşekkür ederim.

Ayrıca tahsis ettikleri burs dolayısıyla TÜBİTAK Bilim İnsanı Destekleme Daire Başkanlığı (BİDEB)'na teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

KISALTMALAR

ACO:	Ant Colony Optimization
API:	Application Programming Interface
CPU:	Central Processing Unit
ÇGSP:	Çoklu Gezgin Satıcı Problemi
GA:	Genetik Algoritma(lar)
GAlib:	Genetic Algorithms Library
GAUL:	The Genetic Algorithm Utility Library
GNU:	GNU's Not Unix
GP:	Genetik Programlama
GSP:	Gezgin Satıcı Problemi
GSX:	Greedy Subtour Crossover
GUI:	Graphical User Interface
IDA*:	Iterative Deepening A*
JAGA:	Java API for Genetic Algorithms
JGAP:	Java Genetic Algorithms Package
KKO:	Karınca Kolonisi Optimizasyonu (Eniyilemesi)
MTSP:	Multiple Traveling Salesman Problem
NP:	Nondeterministic Polynomial
TSP:	Traveling Salesman Problem
TSPLIB:	TSP Library
UML:	Unified Modeling Language
XML:	Extensible Markup Language

1. GİRİŞ

Gezgin Satıcı Problemi (*Traveling Salesman Problem*) veya kısaca GSP (*TSP*), aralarındaki uzaklıklar bilinen N adet noktanın (şehir, parça veya düğüm gibi) her birisinden yalnız bir kez geçen en kısa veya en az maliyetli turun bulunmasını hedefleyen bir kombinasyonel eniyileme problemidir. Problemin çözümü günlük hayatta, yol ve rota planlama (uçak, otobüs, dağıtım kamyonları, bilgisayar ağları, posta taşıyıcılar, vb.), iş planlama, baskı devre kartlarındaki delgi işlemi sırasının belirlenmesi gibi birçok alanda kullanılmaktadır. Problem boyutunun büyük olduğu durumlarda hızlı bir şekilde iyi çözümlerin bulunmasını sağlayan sezgi (*heuristics*) ve yaklaşım algoritmaları, 1950'lerden bu yana, problemin çözümünde kullanılmaktadır. GSP üzerine yapılan çalışmalar günümüzde de yoğun ilgi görmekte ve artarak sürmektedir.

Çoklu Gezgin Satıcı Problemi (*Multiple Traveling Salesman Problem*) ya da ÇGSP (*MTSP*) ise çok sayıda günlük hayat problemini modellemek için kullanılabilecek karmaşık bir problemdir ve Gezgin Satıcı Problemi'nin türevidir. ÇGSP'de *n* adet şehir, her biri ayrı bir satıcıya atanmak üzere *m* adet tura bölünmektedir. Problemin çözümü, araç rotalama, yük dengeleme, takvim oluşturma, üretim yönetimi gibi pek çok alanda kullanılmaktadır. Çoklu GSP, GSP'den daha zor bir problemdir, zira çözümü için öncelikle her bir satıcıya hangi şehirlerin atanacağının ve satıcıların turları üzerindeki şehirlerin optimal sıralamasının belirlenmesi gerekmektedir. Bu nedenle Çoklu Gezgin Satıcı Problemi'nin geleneksel programlama metotlarıyla çözülmesi zordur. Genetik Algoritmalar, Karınca Kolonisi Eniyilemesi ve Yerel

Eniyileme gibi birçok yöntem, problemin çözülmesini kolaylaştırabilir. Problem üzerine 1970'lerden beri yapılan çalışmalar yıllar içinde artmış, ancak yine de belirli bir miktarı geçememiştir. Mevcut olan çalışmalara dayanarak ÇGSP'nin gelişmeye çok açık bir alan olduğu söylenebilir.

Bu tez projesi kapsamında, Java platformu kullanılarak Çoklu Gezgin Satıcı Problemi'nin çözümü için bir eniyileme kütüphanesinin tasarımı ve görsel yazılım geliştirme ortamı ile birlikte gerçekleştirimi yer almaktadır. Eniyileme kütüphanesinde Genetik Algoritmalar ve Yerel Eniyileme (2-opt, 3-opt) yöntemleri ile bu yöntemlerin uygulanmasında görev yapan çeşitli algoritmalara yer verilmektedir. İşbu kütüphane, yazılım geliştiricilerin, spesifik olarak bu problemi çözmek üzere veya daha geniş kapsamlı bir projenin bir kısmında problemin çözülmesini gerektiren yazılımları oluştururken yararlanabilecekleri platform bağımsız bir kaynak teşkil etmektedir. Başka bir yazılım gerekmeksizin, problemin çözümü için planlanan en uygun yöntem tercih edilebilmekte ve görsel yazılım geliştirme ortamının sunduğu seçenekler aracılığıyla, tercih edilen yöntem kolayca uygulanabilmektedir. Eniyileme kütüphanesinde birden çok yönteme yer verilmesi sayesinde, yazılım geliştiricilerin tek bir algoritmayla sınırlanması önlenmektedir. Ayrıca çeşitli yöntemlerin melez olarak uygulanabilmesine de olanak verilmektedir. Kütüphane Çoklu GSP ile birlikte özgün GSP için de çözüm bulabilmektedir. Kütüphane ile birlikte gerçekleştirilen görsel yazılım geliştirme ortamı, Web üzerinden de kullanılabilmektedir.

GSP için mevcut birçok kütüphane ve yazılım bulunmasına rağmen, Çoklu GSP için bu ciddiyette Web tabanlı bir yazılım ve kütüphane bulunmamaktadır. Tez projesi kapsamında yapılan çalışmanın

temel katkısı, bu konudaki boşluğu doldurmaktır. Ayrıca kütüphanenin nesneye dayalı yaklaşım ile tasarlanıp gerçekleştirilmesi sayesinde kolayca genişletilebilir olması, konu üzerinde çalışan yazılım geliştiriciler için önemli bir noktadır.

Tez metninin akışı şu şekildedir:

Bölüm 2'de Gezgin Satıcı Problemi hakkında bilgi verilmektedir. Problemin tanımı ve önemi ile çözümünde kullanılan algoritmalara değinildikten sonra, tez çalışmasının temelinde yer alan Çoklu Gezgin Satıcı Problemi anlatılmaktadır.

Bölüm 3'te Eniyileme yöntemlerinden bahsedilmektedir. Tez projesi kapsamında uygulanan Genetik Algoritmalar ve Yerel Eniyileme yöntemleri etraflı bir biçimde işlenmekte, projede kullanılmayan ancak literatürdeki çalışmaların bir kısmında yararlanılmış olan Karınca Kolonisi Algoritması ise kısaca anlatılmaktadır.

Bölüm 4'te Gezgin Satıcı Problemi ve Çoklu Gezgin Satıcı Problemi ile ilgili literatürdeki çalışmalar incelenmektedir. Bu çalışmaların haricindeki birçok kaynağa ise tez boyunca yeri geldikçe atıfta bulunulmaktadır.

Bölüm 5'te tez projesi kapsamında tasarımı yapılan Eniyileme Kütüphanesi anlatılmaktadır. Tasarım, sınıf diyagramları ile desteklenmekte; kullanılan yöntemler ve tasarım desenleri söz konusu edilmektedir. Bundan sonra ise kütüphanenin gerçekleştirimi ayrıntılı olarak irdelenmektedir. Bu bağlamda önemli bazı metotların içeriği gösterilmekte, yazılan programın akışı anlatılmaktadır.

Bölüm 6'da Çoklu Gezgin Satıcı Kütüphanesi'nin eniyileme kısmında yer verilen algoritma ve yöntemlerin başarısını ölçmek adına bir dizi standart veri üzerinde gerçekleştirilen deneylerin sonuçları verilmektedir. GSP ve ÇGSP için ayrı ayrı *TSPLIB* örnekleri için yapılan deneylerin sonuçları tablolar ve grafikler halinde gösterilmekte, bu tablo ve grafiklerden yola çıkarak sonuçların karşılaştırmalı değerlendirimleri yapılmaktadır.

Bölüm 7'de kütüphanenin gerçekleştirimi kapsamında hazırlanan görsel yazılım geliştirme ortamının detaylarına değinilmektedir. Ortamın tasarımı ve gerçekleştirimi ile yazılımın kullanımından başka, yazılım geliştirme fonksiyonu da ayrıntılı olarak aktarılmaktadır. Ayrıca uygulamanın çalıştırılmasına ilişkin ekran görüntüleri de sunulmaktadır.

Bölüm 8'de ise Çoklu Gezgin Satıcı Problemi'nin eniyilemesinin konu edildiği bu projenin önemi irdelenerek, yazılım geliştirme aşaması sonunda çıkarılan sonuçlar ele alınmakta ve konuyla ilgili ilerde yapılabilecek çalışmalar değerlendirilmektedir.

Yüksek Lisans Eğitimi ve Tez Kapsamında Yapılan Yayınlar

Uğur, A., Özkan, B. ve Cevre, U., 2008, Gezgin satıcı problemini melez bir eniyileme yöntemi ile çözen web tabanlı etkileşimli bir simülasyon aracı geliştirilmesi, *Gazi Üniversitesi Mühendislik – Mimarlık Fakültesi Dergisi* (Hakem değerlendirme sürecinde).

Cevre, U., Özkan, B. ve Uğur, A., 2007, Gezgin satıcı probleminin genetik algoritmalarla eniyilemesi ve etkileşimli olarak Internet üzerinde görselleştirilmesi, *INET-TR 2007*, Ankara.

Özkan, B., Cevre, U. ve Uğur, A., 2008, Melez bir eniyileme yöntemi ile rota planlama, *Akademik Bilişim 2008*, Çanakkale.

2. GEZGİN SATICI PROBLEMİ

2.1 Problemin Tanımı

Gezgin Satıcı Problemi (*Traveling Salesman Problem*) veya kısaca GSP (*TSP*), aralarındaki uzaklıklar bilinen N adet noktanın (şehir, parça veya düğüm gibi) her birisinden yalnız bir kez geçen en kısa veya en az maliyetli turun bulunmasını hedefleyen bir problemdir. Ayrık ve Kombinasyonel Eniyileme (*Combinatorial Optimization*) problemlerinin kapsamına girer. Problemdeki maliyet, uzaklık, zaman veya para gibi unsurlardan biri olabilir. Çizge teorisinde ise, gezgin satıcı problemi "Verilen bir ağırlıklı çizgede (düğümler yani köşeler şehirleri; kenarlar ise şehirlerin arasındaki yolları göstermek; ağırlıklar da yolun maliyeti veya uzunluğu olmak üzere), en düşük maliyetli *Hamilton* Çevrimi'nin bulunması" şeklinde tanımlanabilir.

Problemin matematiksel ifadesi şu şekilde yapılabilir (Helsgaun, 2000):

Verilen bir "maliyet matrisi" $C = (c_{ij})$ için - ki burada c_{ij} şehir i'den şehir j'ye gitme maliyetini temsil eder (i,j = 1,..., n) - 1'den n'e kadar tamsayıların aşağıdaki niceliği minimize eden bir permütasyonunu (i_1, i_2, i_3, ..., i_n) bulunuz:

$$c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_n i_1}$$

Maliyet matrisi C'nin özellikleri problemlerin sınıflandırılmasında kullanılır:

Tüm i ve j değerleri için $c_{ij} = c_{ji}$ ise problem simetriktir ve Simetrik GSP adını alır; aksi takdirde ise asimetriktir ve Asimetrik GSP adını alır. Problemin en yaygın şekli Simetrik GSP'dir. Ancak bazı durumlarda A şehrinden B şehrine gitmenin maliyeti ile, B şehrinden A şehrine gitmenin maliyeti farklıdır. Bazı şehirler arasında tek yön yolların olması, gidiş ve geliş yönlerindeki yolların trafik sıkışıklığının getireceği süre farklılıkları gibi durumlar dikkate alındığında Asimetrik GSP devreye girer.

Eğer üçgen eşitsizliği sağlanıyorsa (tüm i,j ve k değerleri için $c_{ik} \leqq c_{ij} + c_{jk}$ ise), problem metriktir ve Metrik GSP adını alır.

Eğer c_{ij} düzlemdeki noktalar arasındaki Öklit uzaklığı ise problem Öklit tabanlıdır ve Öklit Tabanlı GSP adını alır. Problemin bu tipi için düğümler, R^2 (daha genel anlamda herhangi bir d için, R^d şeklinde) uzayındadır. Öklit Tabanlı GSP doğal olarak hem simetrik hem de metriktir.

2.2 Gezgin Satıcı Probleminin Önemi

GSP'nin önemi, yolculuk mesafelerini minimize etmek isteyen satıcıların büyük ihtiyaçlarından doğmaz. Asıl önem pek çoğu yolculuk rotalarıyla ilişkili görünmeyen çok sayıda başka uygulamadan ileri gelir (Helsgaun, 2000).

Örneğin, şu süreç planlama problemini ele alalım: Belirli bir sayıda işin tek bir makine üzerinde yapılması gerekmektedir. Makine aynı anda yalnızca bir iş yürütebilmektedir. Bir işin yapılmasına başlanmadan önce

makine hazırlanmalıdır (temizlenme, ayarlanma vb). Her bir işin yürütülme zamanının ve bir işten diğerine geçerken kaybolan zamanın verildiği durumda, amaç toplam yürütülme zamanını olabildiğince kısa tutacak bir işletim sırası bulmaktır.

Bu problemin bir GSP örneği olduğu kolayca görülebilir. Burada c_{ij} i işinden sonra j işini tamamlamak için gereken süreyi temsil eder (iki iş arasındaki geçiş zamanı ve j işini yapmak için gereken zaman). Yürütülme zamanı 0 olan sözde bir iş makine için başlama ve bitiş durumunu belirtir.

Çok sayıda gerçek dünya problemi GSP örnekleri olarak formüle edilebilir. Problemin çok yönlülüğü aşağıdaki örnek uygulama alanları ile ortaya konulmaktadır:

- Bilgisayar elektrik tertibatı kurulumu
- Araç rotalama
- Yol ve rota planlama (uçak, otobüs, dağıtım kamyonları, bilgisayar ağları, posta taşıyıcılar vb.)
- İş planlama
- Kristalografi
- Robot kontrolü
- Baskı devre kartlarının delgi işlemi
- Zamandizinsel (kronolojik) sıralama

GSP türünün, yani kombinasyonel eniyilemenin tipik bir problemidir. Bu, GSP üzerinde yapılan çalışmalardan edinilen teorik ve

pratik kavrayışın, bu alandaki diğer problemlerin çözümü için de sıklıkla yarar sağlayabileceği anlamına gelir. Aslında kombinasyonel eniyilemedeki birçok gelişmenin GSP üzerindeki araştırmalara kadar izi sürülebilir. Şu anda iyi bilinen bir hesaplama yöntemi olan dallandır ve bağla (*branch and bound*) ilk defa GSP kapsamında kullanılmıştır (Dantzig et al., 1954). Ayrıca şunu da belirtmek gerekir ki, GSP üzerindeki araştırmalar 1970lerin başında hesaplama karmaşıklığı (*computational complexity*) teorisinin geliştirilmesinde de önemli bir itici güç olmuştur (Karp, 1972).

Fakat GSP'ye duyulan ilgi yalnızca pratikte ve teorideki öneminden kaynaklanmamaktadır. Problemi çözmenin zihinsel zorluğu da büyük rol oynar. Basit tanımına rağmen GSP çözülmesi güç bir problemdir. Zorluk, olası turların sayısı düşünüldüğünde - görece az sayıda şehir için bile astronomik bir değer - aşikar hale gelir. n şehirlik simetrik bir problem için (n-1)!/2 olası tur bulunmaktadır. n = 20 ise 10^{18}'den fazla tur var demektir. Örnek olarak Helsgaun'un (2000) çalışmasındaki 7397 şehirlik problem içinse 10^{25000}'den fazla tur mevcuttur. Karşılaştırma yapmayı kolaylaştırmak adına, evrendeki temel parçacıkların sayısının yalnızca (!) 10^{87} olduğunu belirtmek gerekir.

2.3 Gezgin Satıcı Problemi için Çözüm Algoritmaları

GSP'nin NP-tam problemler kümesinin elemanı olduğu ispatlanmıştır. Bu, zaman karmaşıklıkları üstel olan çok zor problemlerden oluşan bir sınıftır. Sınıfın elemanları birbiriyle ilişkilidir, yani bir problem için bir polinomsal zaman bulunursa, tüm problemler

için polinomsal zaman algoritmaları mevcut olabilir. Ancak yaygın kanı böyle bir polinomsal algoritmanın olmadığı yönündedir. Bu yüzden GSP için optimal çözümler bulmak için genel bir algoritma oluşturma girişimlerinin tümü (muhtemelen) başarısız olacaktır.

Çünkü, böyle bir algoritma dahilinde üretilecek problem örnekleri için çalışma zamanı, girdinin büyüklüğüyle orantılı olarak en azından üstel olarak büyüyecektir. Elbette, buradaki zaman karmaşıklığının herhangi bir algoritmanın en kötü senaryolardaki davranışına tekabül ettiğine dikkat edilmelidir. Ortalama çalışma zamanları polinomsal olan algoritmaların bulunabileceği göz ardı edilemez. Bu tür algoritmaların varlığı ise hala cevaplanmamış bir sorudur.

En kısa güzergahı (turu) bulmanın en basit yolu, verilen N adet şehir için tüm şehir permütasyonlarını listeleyerek her bir olası güzergahın toplam yol uzunluğunu hesaplamaya dayanmaktadır (Özkan vd., 2008). En küçük değere sahip olan güzergah veya güzergahlar kesin çözümdür. Ancak bu yöntemin zaman karmaşıklığı O(n!)'dir. Dolaşılacak 6 şehir varsa, toplam 6!=720 farklı olası güzergahın toplam yol uzunluğunun hesaplanması gerekecektir. 40 şehir olduğu durumda bile, permütasyonların sayısı, bilgisayarın kısa sürede çözemeyeceği kadar büyük olmaktadır. Bu nedenle, kesin yöntemlerin yanında, kısa sürede iyi çözümlerin bulunmasını sağlayan yaklaşım yöntemleri de günümüzde birçok alanda kullanılmaktadır. Problem boyutunun büyük olduğu durumlarda hızlı bir şekilde iyi çözümlerin bulunmasını sağlayan sezgi (*heuristics*) ve yaklaşım algoritmaları, 1950'lerden bu yana, gezgin satıcı problemlerinde kullanılmaktadır (Gambardella and Dorigo, 1996).

Bu ön bilgiyle birlikte GSP için çözüm algoritmaları Kesin algoritmalar ve Yaklaşık (sezgisel) algoritmalar olarak iki sınıfa ayrılabilir.

2.3.1 Kesin Algoritmalar

Kesin algoritmalar, optimal çözümü sınırlı sayıda adımda bulmayı garanti eder. Günümüzde birkaç bin şehirlik simetrik problemler için kesin sonuçlar bulunabildiği gibi, daha fazla sayıda şehir içeren problemlerin çözüm haberleri de gelmektedir.

En etkili kesin algoritmalar, kesit düzlem (*cutting-plane*) veya yüzey bulma (*facet-finding*) algoritmalarıdır (Padberg and Rinaldi, 1991; Grötchel and Holland, 1991; Applegate et al., 1995). Bu algoritmalar oldukça karmaşıktır ve 10.000 satır düzeyinde kodlara sahiptirler. Ayrıca algoritmalar yüksek bilgisayar gücüne ihtiyaç duyarlar. Örneğin, 2392 şehirlik bir simetrik problemin kesin çözümü güçlü bir süper bilgisayar üzerinde 27 saatten uzun bir süreçte bulunabilmiştir (Grötchel and Holland, 1991). 7397 şehirlik bir problemin kesin çözümü ise çok geniş bir bilgisayar ağı üzerinde yaklaşık 3-4 yıllık CPU zamanı almıştır (Applegate et al., 1995). Simetrik problemlerin çözülmesi genellikle asimetrik problemlerden daha zordur (Bellmore and Malone, 1972).

2.3.2 Yaklaşık Algoritmalar

Kesin algoritmaların aksine, yaklaşık algoritmalar iyi çözümler elde etmelerine rağmen optimal çözümün bulunacağını garanti etmezler.

Bu algoritmalar genellikle oldukça basittir ve görece kısa çalışma zamanlarına sahiptir. Bazı algoritmalar ortalama olarak optimal çözümden yalnızca küçük bir yüzdeyle farklı olan çözümler verir. Bundan dolayı, optimum değerden küçük bir sapma kabul edilebilirse, yaklaşık algoritmaları kullanmak uygun olabilir.

Yaklaşık algoritmalar sınıfı üç alt sınıfa ayrılabilir:

- Tur oluşturma algoritmaları
- Tur iyileştirme algoritmaları
- Karma algoritmalar

Tur oluşturma algoritmaları her adımda yeni bir şehir ekleyerek bir güzergahı kademeli olarak meydana getirir. Tur iyileştirme algoritmaları çeşitli değiş tokuşlarla bir tur üzerinde iyileştirmeler yapar. Karma algoritmalar ise bu iki özelliği birleştirir.

Basit bir tur oluşturma algoritması örneği olarak en yakın komşu algoritması (Rosenkrantz et al., 1977) gösterilebilir. Bu algoritmada rastgele seçilmiş bir şehirden başlanır. Ziyaret edilmemiş şehir kaldığı müddetçe turda henüz belirmemiş en yakın şehir ziyaret edilir. Son olarak ilk şehre geri dönülür.

Bu yaklaşım basit ancak genellikle gereğinden fazla aç gözlüdür. Oluşturma sürecindeki ilk mesafeler makul kısalıktadır ancak sürecin sonundaki mesafeler genellikle daha ziyade uzun olacaktır. Bu soruna çare bulmak için pek çok başka tur oluşturma algoritması da geliştirilmiştir (Laporte, 1992).

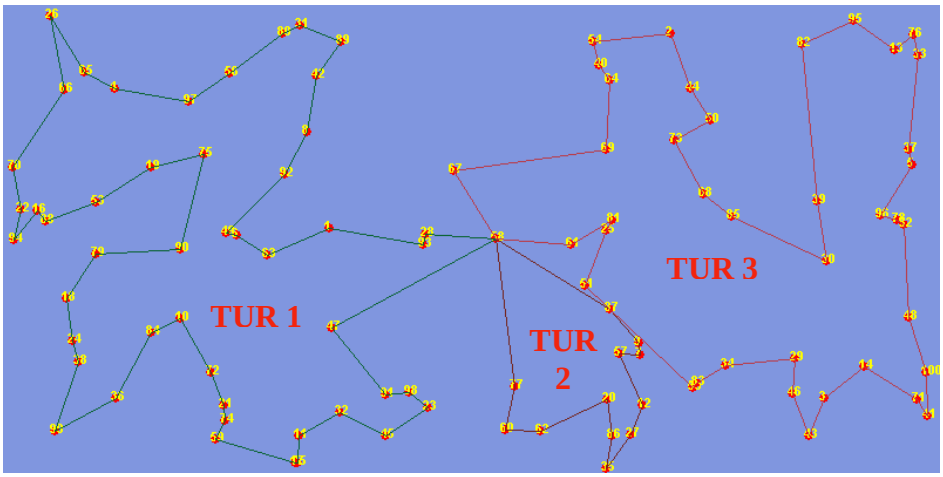
Bununla birlikte, en büyük başarıyı tur iyileştirme algoritmaları yakalamıştır. Bu tip algoritmaların temel bir örneği 2-opt algoritmasıdır. Verilen tur ile başlanır. Turdaki iki bağlantı başka iki bağlantı ile yeni tur uzunluğu daha kısa olacak şekilde yer değiştirilir. Daha fazla iyileştirme yapılması mümkün olmayıncaya kadar bu şekilde devam edilir.

Bu yöntemin genelleştirilmesi simetrik GSP'yi çözmede en etkili yaklaşık algoritmalardan biri olan Lin-Kernighan algoritmasına temel teşkil etmektedir (Lin and Kernighan, 1973). Lin ve Kernighan tarafından ilk olarak 1971 yılında gerçekleştirildiği şekliyle orijinal algoritma ortalama $O(n^{2.2})$ çalışma zaman karmaşıklığındadır ve 100 şehirden az şehirlik problemlerin çoğu için optimal çözümleri bulabilmektedir. Yine de işbu algoritmanın gerçekleştirimi kolay değildir. 1989'da yapılan bir çalışmada (Melamed et al., 1989) yazarlar algoritmanın o tarihlerdeki başka hiçbir uygulamasının, Lin ve Kernighan tarafından elde edilen etkinliği veremediğini belirtmişlerdir.

2.4 Çoklu Gezgin Satıcı Problemi

Çoklu Gezgin Satıcı Problemi (*Multiple Traveling Salesman Problem*) ya da ÇGSP (MTSP) çok sayıda günlük hayat problemini modellemek için kullanılabilecek karmaşık bir problemdir. N adet şehir için alt-tur olmaksızın, her şehri yalnızca bir defa ziyaret ederek en iyiye yakın turu bulmayı hedefleyen ve yine karmaşık bir problem olan Gezgin Satıcı Problemi'nin (GSP) türevidir (Carter, 2003). ÇGSP'de *n* adet şehir, her biri ayrı bir satıcıya atanmak üzere *m* adet tura bölünmektedir. ÇGSP, GSP'den daha zor bir problemdir, zira çözümü için öncelikle her

bir satıcıya hangi şehirlerin atanacağının ve satıcıların turları üzerindeki şehirlerin optimal sıralamasının belirlenmesi gerekmektedir. Örnek bir Çoklu GSP ve 3 gezgin satıcının turları Şekil 2.1'de gösterilmektedir.



Şekil 2.1: 100 şehirlik bir harita için 3 gezgin satıcılı örnek bir ÇGSP

ÇGSP'nin en yaygın uygulaması planlama alanındadır. Bir üretim hattı üzerindeki işlerin planlanması işlemi genellikle GSP olarak modellenir. Üretimin, işlere tahsis edilecek birden çok paralel hat üzerine genişlediği durumlarda ise problem ÇGSP olarak modellenebilir. Yaygın olarak ÇGSP olarak modellenen bir başka problem ise araç planlama problemidir. İşbu problem her birinden yalnızca bir defa geçmek üzere, birçok adresi ziyaret etmek için aynı depodan yola çıkan bir araç kümesinin planlanmasını içerir (Park, 2001). GSP üzerinde ÇGSP olarak modellenebilecek şekilde yapılabilecek bir çeşitleme de, tek satıcının *n* adet şehri *m* adet daha küçük turlar serisi şeklinde dolaşmasıdır. Bu çeşitlemenin günlük hayattaki uygulaması, *n* adet şehri

belirli bir süre içinde dolaşırken, hafta içinde seyahat eden, hafta sonlarında ise eve dönen satıcıların planlanması problemidir.

ÇGSP'nin kombinasyonel karmaşıklığı dolayısıyla, gerçekçi büyüklüklerdeki problemlerin çözülmesi için sezginin (*heuristics*) kullanılması gerekmektedir. Genetik Algoritmalar (GA) GSP için bir sezgisel arama teknikleri sınıfını temsil eder. Araç planlama problemi üzerinde yapılan araştırmalar, GSP için geliştirilen GA'nın kullanımının ÇGSP için de çözüm üretmek üzere yayılmasını sağlamıştır.

Araç planlama probleminin genetik algoritmalar ile çözümü üzerine yapılan araştırmalar üç temel kromozom gösterimi üzerine yoğunlaşmış, özellikle bunlardan sonuncusunun gereksiz çözümleri azaltıp çözüm alanını daraltarak, aramanın etkinliğini arttırdığı gözlenmiştir.

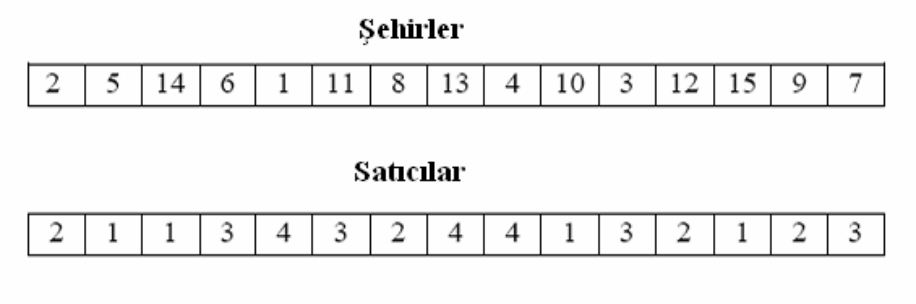
2.4.1 ÇGSP İçin Kromozom Gösterimleri

Çoklu GSP için (özgün GSP'de olduğu gibi) olası çözümlerde yer alan şehir sırasının temsil edileceği bir yapıya ihtiyaç vardır. ÇGSP'ye özel olarak şehirlerin yanında, her bir satıcının hangi şehirlerden sorumlu olacağı da bu yapı içerisinde tutulmalıdır. Şehirlerin permütasyonunu ve satıcıları gösteren bu yapı Genetik Algoritmalar yönteminde kromozom olarak geçmektedir (Bkz: 3.1.3.1 Kromozom ve Topluluk).

İki Kromozom Tekniği

Şekil 2.2 ÇGSP'ye (n=15 ve m=4) getirilen çözümlerin gösterimi için İki Kromozom Tekniği olarak anılan bir yaklaşımı göstermektedir.

Adından da anlaşılacağı gibi, bu yöntem bir çözümü temsil etmek için her biri *n* uzunluğa sahip iki kromozoma ihtiyaç duyar. İlk kromozom *n* adet şehrin bir permütasyonunu içerirken, ikinci kromozom satıcıları ilk kromozomdaki sırasına göre şehirlere atamaktadır (Park, 2001).



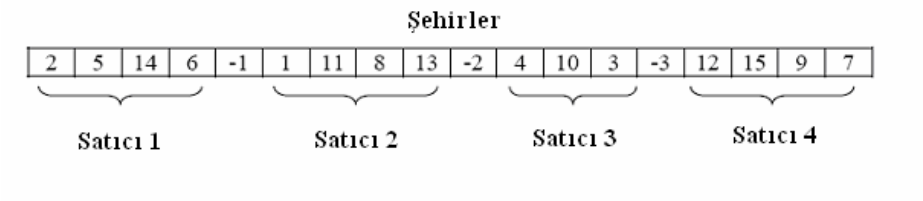
Şekil 2.2: 4 Satıcı için 15 Şehirli ÇGSP'nin İki Kromozom Gösterimi Örneği (Carter, 2003)

Şekil 2.2'deki örnekte 2,8,12 ve 9 numaralı şehirler (bu sırayla) Satıcı 2 tarafından ziyaret edilmektedir. 5, 14, 10 ve 15 numaralı şehirler (bu sırayla) Satıcı 1 tarafından ziyaret edilmekte, aynı şekilde kalan şehirler de Satıcı 3 ve Satıcı 4'e atanmaktadır. Bu kromozom tasarımı kullanıldığında, *n* şehir sayısı ve *m* satıcı sayısı olmak üzere, probleme $n!m^n$ muhtemel çözüm getirilebilir. Bununla birlikte muhtemel çözümlerin pek çoğu gereksizdir. Örneğin, yukarıdaki kromozomların her birindeki ilk iki genin yer değiştirmesiyle elde edilecek farklı kromozom, aynı (gereksiz) çözümün bulunmasına yol açacaktır.

Tek Kromozom Tekniği

Şekil 2.3 aynı ÇGSP'ye (n=15 ve m=4 olan) getirilen çözümlerin temsil edilmesi için ikinci bir yöntem göstermektedir. Bu teknik *n+m-1*

uzunluğundaki tek bir kromozomun kullanımını içerir ve Tek Kromozom Tekniği olarak anılır (Tang et al., 2000). Bu yaklaşımda *n* adet şehir 1'den *n*'e kadar olan tamsayıların bir permütasyonu ile temsil edilir. Bu permütasyon, bir satıcıdan bir sonraki satıcıya geçişi temsil eden *m-1* adet negatif tamsayının (-1'den –(*m-1*)'e kadar) araya eklenmesi ile *m* adet alt-tura bölünmektedir. Bu *n+m-1* tamsayının herhangi bir permütasyonu problem için olası bir çözümü temsil eder.

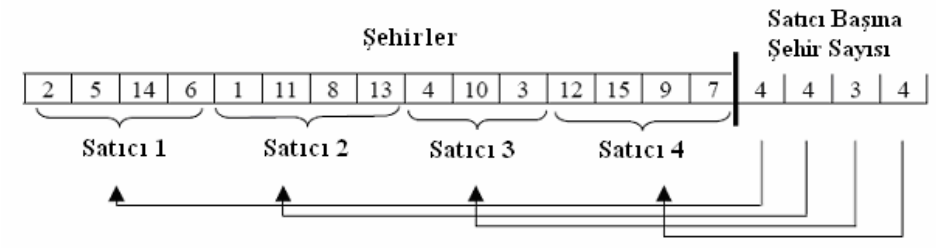


Şekil 2.3: 4 Satıcı için 15 Şehirli ÇGSP'nin Tek Kromozom Gösterimi Örneği (Carter, 2003)

Şekil 2.3'teki örnekte, ilk satıcı 2, 5, 14 ve 6 numaralı şehirleri (bu sırayla) ziyaret etmektedir. İkinci satıcı 1, 11, 8 ve 13 numaralı şehirleri (bu sırayla) ziyaret etmekte ve Satıcı 3 ve Satıcı 4 de kalan şehirleri aynı şekilde ziyaret etmektedirler. Bu kromozom planı kullanıldığında, probleme getirilebilecek *(n+m-1)*! olası çözüm oluşmaktadır. Muhtemel kromozomların birçoğu gereksizdir. Örneğin, ilk beş gen ile sonraki beş genin basitçe yer değiştirilmesi eşdeğer (gereksiz) bir çözüm üretecektir. Ayrıca, bu yaklaşımda iki veya daha fazla negatif tamsayının arka arkaya gelmesi olasılık dahilinde olduğu için yararlanılacak satıcı sayısında belirgin düşüş yaşanabilir.

İki Parçalı Kromozom Tekniği

Şekil 2.4 aynı ÇGSP için (n=15 ve m=4) birbirinden ayrı iki parçaya sahip yeni bir kromozomu göstermektedir. Yöntem bu sebeple İki Parçalı Kromozom Tekniği olarak anılmaktadır (Carter, 2003). ÇGSP için iki parçalı kromozom kullanma fikri, gruplama genetik algoritmasının (Falkenauer, 1998) iki parçalı kromozomuyla benzerdir. İlk parça *n* adet şehri temsil eden 1'den *n*'e kadar olan tamsayıların bir permütasyonudur. Kromozomun ikinci parçası ise *m* uzunluğundadır ve *m* adet satıcının her birine atanmış şehir sayılarını gösterir. Kromozomun ikinci parçasına tahsis edilen değerler, toplamları ziyaret edilecek şehirlerin sayısı (*n*) eden *m* adet pozitif tamsayı olma kısıtındadır.



Şekil 2.4: 4 Satıcı için 15 Şehirli ÇGSP'nin İki Parçalı Kromozom Gösterimi Örneği (Carter, 2003)

Şekil 2.4'te Satıcı 1 2, 5, 14 ve 6 numaralı şehirleri (bu sırayla) ziyaret etmekte, Satıcı 2 1, 11, 8 ve 13 numaralı şehirleri (bu sırayla) ziyaret etmekte, bu durum Satıcı 3 ve Satıcı 4 için de aynı şekilde sürmektedir. Yeni iki parçalı kromozom yönteminin kullanılması, gereksiz çözümlerin bir kısmının (hepsinin değil) budanması dolayısıyla, arama alanının büyüklüğünün azaltılmasını sağlar. Örneğin, Satıcı 1'e

atanan şehirler iki parçalı kromozomda her zaman başta bulunur ve ikinci satıcıya atanan şehirler tarafından takip edilir. Bu önceki iki kromozom gösteriminin hiçbirinde söz konusu değildir, zira verilen bir satıcı için şehirler kromozomdaki herhangi bir göreli pozisyonda bulunabilmektedir (zaten kromozomlardaki gereksizliğin çoğu da bu sebepten dolayıdır). ÇGSP için iki parçalı kromozomun kullanılmasıyla kromozomun ilk parçası için *n*! olası permütasyon oluşur. Kromozomun ikinci parçası toplamı *n* eden bir pozitif vektörü (x_1, x_2, ..., x_m) göstermektedir.Bu gereksinimi sağlayan $\binom{n-1}{m-1}$ tane farklı pozitif tamsayı değerli m-vektörü mevcuttur. Bu yüzden iki parçalı kromozom için çözüm alanının büyüklüğü $n!\binom{n-1}{m-1}$ dir.

3. ENİYİLEME YÖNTEMLERİ

GSP ve ÇGSP'nin çözümü için sezgisel ve yaklaşıma dayalı birçok eniyileme yöntemi kullanılmaktadır. Bunlardan en çok öne çıkanlar Genetik Algoritmalar, Yerel Eniyileme ve Karınca Kolonisi Eniyilemesi yöntemleridir. Tez projesi kapsamında Genetik Algoritmalar ve Yerel Eniyileme (2-opt, 3-opt)'den yararlanılmaktadır, ancak literatürdeki bir kısım çalışmada kullanıldığı için, tez projesinde yer almamasına rağmen Karınca Kolonisi Eniyilemesi de kısaca anlatılmaktadır.

3.1 Genetik Algoritmalar

Genetik Algoritmalar (GA), Yapay Zeka'nın hızlı gelişen alanlarındandır. Özellikle kombinasyonel eniyileme problemlerine yaklaşık iyi sonuçlar bulmayı hedefleyen arama yöntemleridir (Uğur, 2008). Problemin çözümünde kullanılacak rastgele seçilmiş bir çözüm kümesi oluşturabilmek için evrimsel mekanizmaların kullanıldığı bu yöntemlerin temel mantığı topluluğun nesilden nesle geçmesi sırasında kötü çözümlerin yok olmasına ve iyi çözümlerden daha iyi çözümlere ulaşılmasına dayanır.

3.1.1 Evrimsel Hesaplamanın Tarihçesi

1950ler ve 1960larda bilgisayar bilimleriyle uğraşan pek çok kişi, evrimin mühendislik problemleri için bir eniyileme yöntemi olarak kullanılabileceği fikrinden yola çıkarak evrimsel sistemler üzerinde bağımsız olarak çalışmalar yapmışlardır. Bu sistemlerin tümünde yer alan

fikir, bir problem için verilen çözüm adayları topluluğunun, doğal genetik varyasyon ve doğal seçilimden esinlenilerek oluşturulmuş operatörler kullanılarak evrilmesidir. İlk olarak Rechenberg (1973) kanat profili gibi araçlar için gerçek-değer parametrelerini eniyilemek üzere kullandığı bir yöntem olan "evrim stratejileri"ni (Alm. *Evolutionsstrategie*) tanıtmıştır. Bu fikir Schwefel (1975) tarafından daha da geliştirilmiştir. Evrim stratejileri halihazırda GA'dan bağımsız aktif bir araştırma alanı olarak varlığını sürdürmektedir. Fogel, Owens ve Walsh (1966) "evrimsel programlama"yı geliştirmişlerdir. Bu teknikte görevler için verilen aday çözümler, durum geçiş diyagramları rastgele mutasyona uğratılıp en uygunun seçilmesiyle evrimlenen sonlu durum makineleri olarak gösterilmektedir. Evrimsel programlama da halen bir aktif araştırma alanı olarak mevcuttur. Evrim stratejileri, evrimsel programlama ve genetik algoritmalar hep birlikte evrimsel hesaplama (*evolutionary computation*) sahasının omurgasını oluşturur.

1950ler ve 1960larda çalışmalar yapan birçok başka kişi de eniyileme ve makine öğrenmesi için evrim benzeri algoritmalar geliştirmişlerdir. Box (1957) ve Bremermann (1962) bu alanda çalışmalarda bulunmuşlardır, ancak hazırladıkları çalışmalar evrim stratejileri, evrimsel programlama ve genetik algoritmalar kadar ilgi çekmemiştir. Bunlara ek olarak, bazı evrimsel biyologlar kontrollü deneyler gerçekleştirmek amacıyla, evrimi bilgisayar kullanarak simüle etmişlerdir (Baricelli, 1957 gibi). Bilgisayarın gelişimindeki süreçte evrimsel hesaplamanın da oldukça etkili bir faktör olduğu söylenebilir.

Genetik Algoritmalar, 1960larda John Holland tarafından keşfedilmiş ve 1960lar ve 1970ler boyunca Holland, öğrencileri ve

Michigan Üniversitesi'ndeki meslektaşları tarafından geliştirilmiştir. Evrim stratejilerinin ve evrimsel programlamanın aksine, Holland'ın ilk hedefi belirli problemleri çözebilecek algoritmalar tasarlamak değil, adaptasyon olgusunu doğada gerçekleştiği biçimiyle inceleyerek doğal adaptasyon mekanizmasını bilgisayar sistemlerine uygulamayı sağlayacak yollar geliştirmektir. Holland'ın 1975 tarihli kitabı "*Adaptation in Natural and Artificial Systems"* genetik algoritmayı biyolojik evrimin bir soyutlaması olarak sunmuştur ve GA kapsamı içindeki adaptasyonun teorik bir çerçevesini vermektedir. Holland'ın GA'sı, bir kromozom (birler ve sıfırlardan ya da bitlerden oluşan diziler) topluluğunu yeni bir topluluğa, doğal seçilimi genetik benzeri çaprazlama, mutasyon ve ters çevirme (*inversion*) operatörleri ile birlikte kullanarak taşımaya yarayan bir yöntemdir. Her kromozom genlerden (bitler) oluşur, her bir gen ise belirli bir alelin (0 veya 1) örneğidir. Seçilim operatörü topluluktaki kromozomların içinden çoğalmasına müsaade edilenleri seçecektir. Ortalama olarak daha yüksek uygunluğa sahip kromozomlar daha çok çocuk üretir. Çaprazlama, biyolojideki iki haploid (tek kromozomlu) organizmanın tekrar birleşmesine kabaca benzer bir biçimde, iki kromozomun alt parçalarını değiş tokuş eder. Mutasyon, kromozomun içinde belirli yerlerdeki alel değerlerini rastgele değişikliğe uğratır. Ters çevirme ise kromozomun sürekli bir kısmındaki düzeni değiştirerek, genlerin sıralarını yeniden düzenler.

Holland'ın tanıştırdığı çaprazlama, mutasyon ve ters çevirme içeren topluluk tabanlı algoritma büyük bir yeniliktir. Ayrıca Holland evrimsel hesaplamayı sağlam bir teorik temele oturtan (Holland, 1975) ilk kişidir. Yakın zamana kadar genetik algoritmalar üzerindeki hemen tüm

çalışmalar bu plan temel alınarak yapılmaktayken, son yıllarda ise pek çok evrimsel yaklaşımı etkileşime uğratarak yeni yöntemler geliştirilmeye başlanmıştır. Holland'ın öğrencisi olan Goldberg'in hazırladığı doktora tezi ile "*National Science Foundation*" tarafından verilen Genç Araştırmacı ödülünü alması ve dört yıl içinde klasik eserini (Goldberg, 1989) yayınlaması ile GA'nın pratik kullanımına olan ilgi çoğalmıştır. John Koza'nın 1992 tarihli çalışması ise, genetik algoritmadaki bireylerin yerine bilgisayar programlarını koyarak "genetik programlama" kavramını yaratmış (Koza, 1992) ve evrimsel hesaplamaya yeni bir dal eklemiştir.

3.1.2 Biyolojik Terminoloji

Genetik Algoritmalar bağlamında adı geçen biyolojik terimler biyoloji bilimiyle paralel olarak kullanılmaktadır, ancak gerçek biyolojik karşılıklarına göre çok daha basit kavramları kapsarlar.

Tüm canlı organizmalar hücrelerden oluşur ve her hücre, organizma için bir şifre olarak hizmet veren, bir veya daha fazla kromozomdan oluşan aynı kümeyi - DNA dizisi - içerir. Bir kromozom kavramsal olarak genlere bölünebilir, bu genlerin her biri belirli bir proteini şifreler. Kabaca açıklamak gerekirse, bir genin belirli bir özelliği (ör; göz rengi) şifrelediği düşünülebilir. Bu özellik için farklı olası seçeneklere (ör; mavi, kahverengi, ela) aleller denir. Her gen kromozom üzerindeki belirli bir pozisyonda (*locus*) bulunur.

Çoğu organizma her bir hücresinde birden çok kromozom taşır. Genetik materyalin tamamına (bütün kromozomların toplamı)

organizmanın genomu denir. Genotip terimi, bir genomdaki belirli bir gen kümesine tekabül eder. Aynı genoma sahip iki birey aynı genotipe sahiptir. Genotip, bireyin gelişimi sırasında organizmanın fenotipine - fiziksel ve zihinsel karakteristiklerine (göz rengi, boy, beyin büyüklüğü ve zeka gibi) - yön verir.

Kromozomları çiftler biçiminde dizili organizmalara diploid; bu biçimde dizili olmayan organizmalara ise haploid organizmalar adı verilir. Doğada, çoğalabilen türlerin çoğu (vücuttaki tüm somatik hücreleri 23er çift kromozom içeren insanlar da dahil olmak üzere) diploid organizmalardır. Üreme sırasında çaprazlama gerçekleşir: her ebeveynde genler tüm kromozom çiftleri arasında değiş tokuş edilerek bir gamet (tek bir kromozom) oluşturulur. Daha sonra ise iki ebeveynden gelen gametler birleşerek tam bir diploid kromozom kümesi yaratır. Haploid çoğalmada genler iki ebeveynin tek ipli kromozomları arasında değiş tokuş edilir. Çocuklar mutasyona maruz kalır, bu sırada nükleotidler (DNA'nın temel parçacıkları) değiştirilir. Bir organizmanın uygunluğu genellikle organizmanın üreme ihtimali ya da sahip olduğu çocuk sayısı (doğurganlık) olarak tanımlanır.

GA'da kromozom terimi, bir problem için genellikle bir bit dizisi şeklinde kodlanmış olası bir çözüme tekabül eder (Mitchell, 1999). Genler, ya tek bitler ya da yan yana bulunan ve aday çözümün belirli bir unsurunu kodlayan kısa bit bloklarıdır. Bir bit dizisindeki alel 0 veya 1'dir; daha geniş alfabeler içinse her bir locusta daha birçok alel bulunması ihtimal dahilindedir. Çaprazlama, iki adet tek kromozomlu haploid ebeveynin arasındaki genetik materyal değişimini içerir. Mutasyon, rastgele seçilen bir locustaki biti tersine çevirmeyi (ya da daha

geniş alfabeler için rastgele seçilen bir locustaki sembolü rastgele seçilen yeni bir sembol ile değiştirmeyi) içerir.

GA'nın bir çok uygulaması haploid, özellikle de tek kromozomlu bireyleri kullanır. Bit dizileri kullanan bir GA'da bir bireyin genotipi, o bireyin kromozomundaki bitlerin yerleşim biçimidir. GA'lar için genellikle bir fenotip nosyonundan söz edilmese de, yakın tarihli çalışmalarda bu yönde denemeler yapılmaktadır. Bir yapay sinir ağının bit dizisi kodlaması ve yapay sinir ağının kendisi hem genotipik hem de fenotipik seviyenin bulunduğu bir örnektir.

3.1.3 Genetik Algoritmaların Temel Kavramları

Evrimsel hesaplama topluluğunda yerleşik, tek bir Genetik Algoritma tanımı bulunmamakla beraber, Genetik Algoritma sınıfına girdiği kabul edilen yöntemlerde en azından şu ortak unsurlara rastlanır: kromozom topluluğu, uygunluğa göre seçilim, yeni bireyler üretmek için çaprazlama, yeni bireylerin rastgele mutasyonu. Holland'ın GA'nın unsurları içinde saydığı ters çevirme operatörü günümüzde nadiren kullanılmaktadır.

3.1.3.1 Kromozom ve Topluluk

GA'da kromozomlar, problem için olası çözümleri temsil ederler. Bu çözümlerden her birine birey adı verilir. Topluluk (popülasyon) kromozomlardan (bireylerden) oluşan kümedir. GA'nın kromozom toplulukları üzerinde işlemler yapması ile, bir önceki topluluklar her seferinde yeni üretilen topluluklarla yer değiştirir.

3.1.3.2 Uygunluk

Uygunluk değeri, çözümün kalitesini belirler ve uygunluk fonksiyonu kullanılarak hesaplanır. Uygunluk fonksiyonu mevcut topluluktaki her kromozoma bir puan (uygunluk değeri) atar. Kromozomun uygunluğu, o kromozomun eldeki problemi ne kadar iyi çözdüğüyle ilişkilidir. Bu bakımdan uygunluk fonksiyonu bir çözüm kalitesi değerlendirme birimi olarak iş yapar. Kalite değerlendirme biriminde normalizasyon işlemine gerek duyulur. Genetik Algoritmalar normalizasyon işlemine karşı çok hassastır. Araştırma bir taraftan, iyi çözümlerin bulunduğu tarafa doğru odaklanmaya çalışırken, diğer taraftan, algoritmanın küresel optimizasyon işlemini gerçekleştirebilmesi için araştırmaya yeterli bir ıraksamanın sağlanması gerekir. Aksi halde erken yakınsama (*premature convergence*) problemi ortaya çıkacak ve bu da tüm araştırma uzayının tamamen araştırılmasını engelleyecektir (Mitchell, 1999). Bu olay küresel araştırmada arzu edilmeyen bir durumdur. Normalizasyon işleminde yaygın olarak kullanılan teknik, ölçekleme penceresi (*scaling window*) işlemidir.

Uygunluk fonksiyonunun çalışma mekanizması bir örnekle daha iyi açıklanabilir. Bilindiği gibi GA'nın yaygın bir uygulaması da fonksiyon optimizasyonudur. Bu işlemde örneğin çok parametreli bir fonksiyonu en yüksek değerine çıkaran parametre değerlerini bulmak amaçlanmaktadır. Basit bir örnek olarak, reel değerli tek boyutlu bir fonksiyon olan:

$f(y) = y + |sin(32y)|$, $0 \leq y < \pi$ verilsin. Burada aday çözümler y'nin değerleridir ve gerçek sayıları temsil eden bit dizileri olarak kodlanabilirler. Uygunluk hesaplaması verilen bir x bit dizisini y

gerçek sayısına çevirir ve fonksiyonu o değer ile hesaplar. Bit dizisinin (kromozomun) uygunluğu o noktada fonksiyonun değeridir.

3.1.3.3 Genetik Operatörler

GA'nın en basit formu üç tip genetik operatör içerir: seçilim, çaprazlama (tek noktalı) ve mutasyon.

Seçilim: Bu operatör topluluktaki kromozomları üreme için seçer. Kromozomun yüksek uygunluk değerine sahip olması, üremek için seçilmesi ihtimalini arttırır.

Çaprazlama: Bu operatör rastgele bir locus belirleyip, iki kromozomun bu locustan önceki ve sonraki kısımlarını değiş tokuş ederek iki yeni birey yaratılmasını sağlar. Örneğin 11110000 ve 01011010 dizileri, her birisindeki beşinci locustan itibaren çaprazlama yapılması halinde 11110010 ve 01011000 bireylerini üretirler. Çaprazlama operatörü doğadaki iki haploid organizma arasındaki biyolojik yeniden oluşturma işlemini yaklaşık olarak taklit eder.

Mutasyon: Bu operatör kromozomdaki bazı bitleri rastgele olarak ters çevirir. Örneğin, 11000101 dizisi dördüncü bitte mutasyona uğrayarak 11010101 dizisine dönüşebilir. Mutasyon, bir dizinin herhangi bir bit pozisyonunda belirli bir ihtimal dahilinde - genellikle çok küçük (ör; 0.0001) - gerçekleşebilir.

3.1.4 Genetik Algoritma Süreci

Çözülmek üzere tanımlı bir problem verilmesi ve aday çözümlerin birer bit dizisi ile temsil edilmesi halinde, GA şu şekilde çalışır:

1. *N* adet kromozom (problem için aday çözümler) içeren rastgele oluşturulmuş bir topluluk ile başla.
2. Topluluktaki her *x* kromozomu için *f(x)* uygunluk değerini hesapla.
3. Aşağıdaki adımları birey *n* (topluluk büyüklüğü) oluşturuluncaya kadar tekrarla.
    a. Güncel topluluktan, yüksek uygunluk değerinin seçilme ihtimalini arttırdığını göz önünde bulundurarak, iki ebeveyn kromozom seç. Seçilim, aynı kromozomun birden çok defa ebeveyn olarak seçilmesine olanak verecek şekilde yapılır.
    b. P_c olasılığı ("çaprazlama olasılığı" ya da "çaprazlama oranı") ile seçilen çifti iki yeni birey oluşturmak üzere rastgele belirlenen bir noktadan çaprazla. Eğer çaprazlama gerçekleşmezse ebeveynlerinin birebir kopyası olan iki çocuk oluştur.
    c. P_m olasılığı ("mutasyon olasılığı" ya da "mutasyon oranı") ile, oluşan iki çocuğu tüm locuslarında mutasyona uğrat.

d. Sonuçta elde edilen kromozomları yeni topluluğa ekle.

4. Önceki topluluğu yeni topluluk ile değiştir.

5. Sonlandırma koşulu sağlandıysa mevcut topluluktaki en iyi çözümü döndür, sağlanmadıysa 2. adıma dön (Cevre vd., 2007).

Bu sürecin her bir iterasyonuna bir nesil adı verilir. Bir genetik algoritma genellikle 50 ila 500 zaman zaman ise daha fazla nesil için döndürülür. Nesillerin toplamına bir koşu (çalıştırma) adı verilir. Bir çalıştırma seansının sonunda genellikle bir veya daha fazla yüksek uygunluk değerine sahip kromozom mevcut olacaktır. Çalışma süresince rastgelelik büyük önem arz ettiği için farklı sayılarda topluluk büyüklükleri, değişik davranışlara dolayısı ile değişik sonuçlara neden olacaktır. GA araştırmacıları deneysel sonuçlarını aynı problem üzerindeki birden çok koşunun ortalamasını alarak verebilmektedirler.

3.1.4.1 Kromozomların Kodlanması

Bir problemin çözümünde Genetik Algoritmalar kullanılacaksa çözümün ilk adımı kromozomların nasıl kodlanacağına karar vermektir. Kodlama yaklaşımı çözümün başarısına doğrudan etki eder ve problemin türüne ve özelliklerine göre farklılık gösterir.

İkili Kodlama: En yaygın kodlama yöntemidir. İkili kodlamada her kromozom 1 ve 0'lardan oluşan bir karakter dizisi biçiminde ifade edilir (Bkz: Şekil 3.1). İlk defa Holland ve öğrencileri tarafından uygulanmış

ve daha sonra gelen GA araştırmacıları tarafından da benimsemiştir. Gri kodlama (Bethke, 1980) ve Hillis'in diploid ikili kodlama planı (1990) gibi yöntemler bu metodun uzantılarıdır. İkili kodlamanın uygun olduğu bir problem örneği olarak *Knapsack* (Sırt Çantası) problemi verilebilir. Yöntemin çok sayıda avantajına rağmen yine de yapay sinir ağlarının evrimi gibi birçok problem için uygun olmadığı ve kimi zaman fazlaca rastgele dizilimlere açık olduğu söylenebilir.

Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

Şekil 3.1: İkili kodlama (Obitko, 1998)

Permütasyon Kodlama: Sabit uzunluk ve sabit sıra yaklaşımı bakımından ikili kodlamaya benzediği söylenebilecek olan yöntem, genelde sıralama problemlerinde kullanılır. Permütasyon kodlamada her kromozom, ilgili karakterin sıralamadaki pozisyonunu belirten sayılardan oluşan bir dizi ile ifade edilir (Bkz: Şekil 3.2). Metodun kullanıldığı yaygın bir güzergah sıralaması örneği olarak Gezgin Satıcı Problemi verilebilir.

Chromosome A	1 5 3 2 6 4 7 9 8
Chromosome B	8 5 6 7 2 3 1 4 9

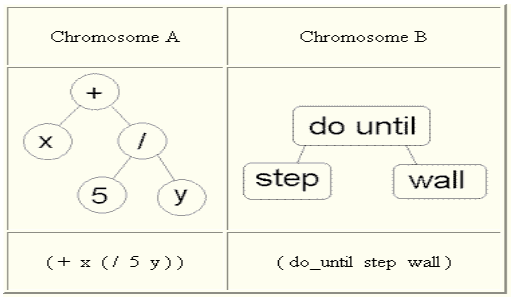
Şekil 3.2: Permütasyon kodlama (Obitko, 1998)

Çok Karakter ve Gerçek Değer Kodlaması: Pek çok uygulama için kromozom oluştururken birden çok karakter içeren bir alfabe veya gerçek sayılar kullanmak uygun olacaktır (Bkz: Şekil 3.3). Bu gibi uygulamalara örnek olarak Kitano'nun (1990) çizge oluşturma grameri ile Montana ve Davis'in (1989) sinir ağları için gerçek değerli gösterimi verilebilir.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Şekil 3.3: Çok karakter ve gerçek değer kodlamaları (Obitko, 1998)

Ağaç Kodlamaları: Kodlama için ağaç yapısını kullanmak arama uzayını ucu açık hale getirerek bir avantaj yarattığı halde, bu ucu açıklık arama uzayının aşırı büyümesine neden olarak bir dezavantaj haline de gelebilir. Ağaç kodlamalarının en önemli örneği Koza'nın (1992) bilgisayar programlarını temsil etmek için kullandığı yöntemdir (Bkz: Şekil 3.4). Ağaç kodlamalarının yaygın olmayan, hatta deneysel sayılabilecek bir kodlama türü olduğu söylenebilir.

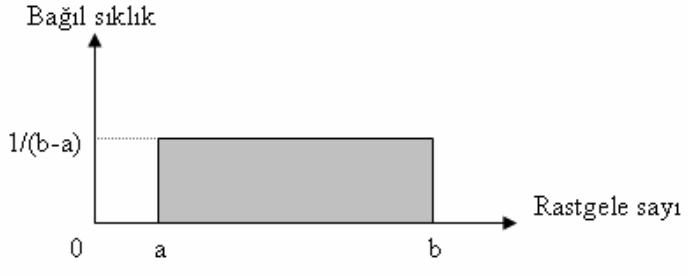


Şekil 3.4: Ağaç kodlamaları (Obitko, 1998)

3.1.4.2 Topluluğun İlklenmesi

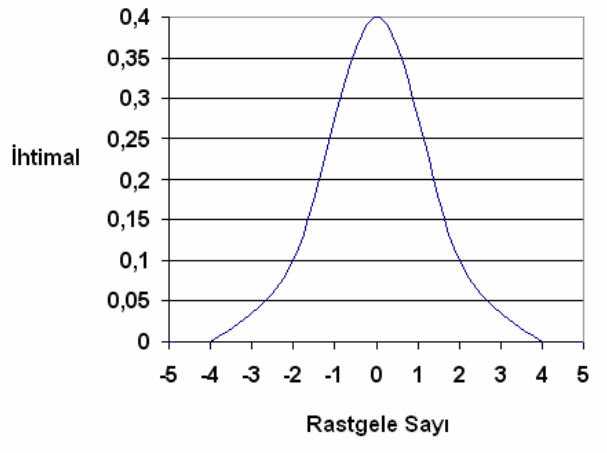
Kromozomların hangi yöntemle kodlanacağı belirlendikten sonra yapılması gereken bir diğer görev ise topluluğun ilklenmesi, yani başlangıç topluluğunun oluşturulmasıdır (Şen, 2004). Pek çok problemde (sıralama problemleri gibi) topluluğun ilklenmesi, kromozomlardaki bitlerin rastgele bir biçimde alel değerlerini almasından ibarettir. Örneğin GSP için oluşturulan toplulukta, her kromozom dolaşılacak tüm şehirlerin numaralarını karışık olarak içerir. Ancak kimi problemler için (bazı arama problemleri gibi) kromozomlara yerleştirilmek üzere rastgele sayıların üretilmesi gerekir.

Rastgele sayıların bağıl frekans dağılımlarının bazı durumlarda göz önünde tutulması gereklidir. GA çözümlemelerinde biri tekdüze (*uniform*) diğeri de normal (*Gauss*) dağılımlı olmak üzere iki tür kullanılır. Yaygın olarak kullanılan tür ise tekdüze olandır. Bu a ve b gibi iki sayı arasındaki tüm gerçek sayıların birinin diğerine üstünlüğü (seçim üstünlüğü) olmayacak biçimde eşit ihtimalle seçilmesini temsil eder. Bu tür rastgele sayıların bağıl sıklık diyagramları (histogramları) Şekil 3.5'te gösterildiği üzere rastgele sayıları verilen alt ve üst sınırlar (a ve b) arasında içeren ama yüksekliği 1/(b-a)'ya eşit olan bir dikdörtgenden ibarettir. Bu dikdörtgenin altındaki alan tanım üzerine 1'e eşittir.



Şekil 3.5: Tekdüze dağılı a ve b arasında değer alan rastgele sayılar (Şen, 2004)

GA işlemlerinde zaman zaman kullanılan bir başka rastgele sayı dağılımı teorik olarak eksi ve artı sonsuz arasında değer alan ve bir ortalama değer, μ, etrafında belirli bir standart sapmaya, σ, göre salınım yapan sayılardır. Bunlara normal (*Gauss*) dağılımlı sayılar adı verilir. Ortalama değerin sıfır, standart sapmanın da bire eşit olması halinde bu sayıların bağıl sıklık (ihtimal) dağılımı Şekil 3.6'da standart bir çan eğrisi halinde verilmiştir. Görüleceği üzere, normal sayıların ortaya çıkma ihtimalleri eşit olmayıp, ortalamaya yakın olanlar için büyük ama buradan kuyruklara doğru gidildikçe küçük ihtimaller söz konusudur. Kuyruklara yaklaşmak rastgele sayının uç (ekstrem) değerler alması anlamına gelir.



Şekil 3.6: Gauss dağılımlı rastgele sayılar (Şen, 2004)

Buna standart rastgele sayı bağıl sıklığı adı verilir. Ancak, standart olmayan yani ortalaması, μ, sıfırdan ve standart sapması, σ, da birden farklı olan rastgele sayılar, Sr, aşağıdaki basit denkleme göre üretilebilir:

$$Sr = \mu + \sigma sr$$

Kromozomların içeriği belli olup, topluluk ilklendikten sonra her kromozom için uygunluk fonksiyonu çalıştırılarak kromozomların uygunluk değerleri hesaplanır (Bkz. 3.1.3.2 Uygunluk).

3.1.4.3 Seçilim

GA'da bir sonraki nesle geçiş sırasında, yeni topluluğu oluşturmak için mevcut topluluktan çaprazlama ve mutasyon işlemlerine tabi tutulacak bireylerin seçilmesi gerekir. Teoriye göre iyi olan, bir başka deyişle uygunluk değeri yüksek olan bireyler yaşamını sürdürmeli ve bu bireylerden yeni bireyler oluşturulmalıdır. Bu nedenle tüm seçilim yöntemlerinde uygunluk değeri fazla olan bireylerin seçilme olasılığı

daha yüksek tutulur. Seçilim sırasında dengenin gözetilmesi gerekir. Çok güçlü bir seçilim, o seviyede yüksek uygunluğa sahip bireylerin topluluğu kaplamasını ve çözüme ulaşmak için gereken çeşitliliğin azalmasını beraberinde getirir. Çok zayıf bir seçilim ise aşırı yavaş bir evrime neden olur.

Rulet Seçilimi (Uygunluk Orantılı Seçilim): Holland'ın orjinal GA'sı bir bireyin seçilimi için beklenen değerin, o bireyin uygunluk değerinin topluluğun ortalama uygunluğuna olan oranına eşit olduğu uygunluk orantılı seçilim yöntemini kullanmaktadır. Bunu gerçekleştirmek için uygulanan en yaygın yöntem "rulet tekerleği" örneklemesi adını almaktadır. İşbu yöntemde her bir bireye dairesel rulet tekerleğinde belirli bir parça ayrılmıştır; parçaların büyüklüğü ise bireyin uygunluğuyla orantılıdır. Tekerlek N (topluluktaki birey sayısı) defa döndürülerek, tekerleğin gösterdiği parçaya sahip olan bireyin çoğalarak bir sonraki nesile aktarılmak üzere seçilmesi sağlanır. Rulet seçiliminin en büyük problemi, aramanın başında topluluktaki bireylerin uygunluk değerleri arasındaki farkın fazla olması ve uygunluk değeri yüksek bireylerin toplulukta kısa sürede çoğalmasıdır. Bu durumda birbirine çok yakın ama çözümden uzak uygunluk değerlerine sahip bireylerden oluşan bir topluluk oluşur; evrim yavaşlar; GA ise yerel minimumlara takılarak daha fazla arama yapmayı bırakır ve "erken yakınsama" olarak anılan problem ortaya çıkar. Buradan çıkan sonuç GA'daki evrim oranının topluluktaki uygunluk değerlerinin farklılığına bağlı olmasıdır.

Sıralı Seçilim: Baker tarafından (1985) çok erken yakınsama sorununu çözmek üzere önerilen sıralı seçilim yönteminde, topluluktaki bireyler uygunluk değerlerine göre sıralanır. Bireylerin beklenen değerleri mutlak uygunluklarından çok sıralamalarına göre belirlenir. Bu durumda uygunluklara bağlı bir orantılama yapılmaz. Mutlak uygunluk bilgisinin gözönüne alınmamasının, yakınsama problemlerini bertaraf etmek gibi bir avantajı olmasının yanısıra, uygunlukları arasında çok büyük fark olan iki bireyi tespit etmenin faydalı olabileceği kimi durumlarda fark bilgisinden yoksun olmak gibi bir dezavantajı da vardır. Sıralama, uygunluk değeri yüksek küçük bir grup bireyin topluluğu ele geçirmesini engelleyerek bireylerin uygunlukları arasındaki farkların yüksek olduğu topluluklardaki seçilim baskısını azaltır. Doğrusal sıralama yönteminde, en kötü uygunlukta olan bireyden en iyi uygunluğa sahip bireye doğru tüm bireylere, 1'den *N*'e kadar birer birer artan değerler verilir. Bir bireyin seçilme olasılığı, o bireye verilen değerin tüm bireylere verilen değerler toplamına oranı kadardır. Doğrusal sıralamadan başka üstel sıralama gibi çeşitli sıralama yöntemleri de önerilmiştir. Sıralı seçilim yöntemi kimi durumlarda GA'nın yavaş çalışmasına sebebiyet verse de, çoğu durumda topluluktaki çeşitliliği koruması sayesinde daha başarılı bir aramayı garantiler.

Turnuva Seçilimi: Seçilim baskısı (hangi uygunluğa kadar olan bireylerin çoğalmak üzere seçilebilceğine karar verilmesi) anlamında sıralı seçilime benzeyen turnuva seçilimi yöntemi, özellikle paralel çalışmaya olanak vermesiyle verimlilik anlamında üstünlük taşır. Bu yöntemde topluluktan rastgele iki birey alınır. Daha sonra ise 0 ile 1

arasında rastgele bir *r* değeri belirlenir. Eğer $r < k$ ise (burada k belirli bir parametredir, ör; 0.7) iki bireyden uygunluk değeri daha yüksek olanı ebeveyn olmak üzere seçilir; aksi takdirde ise uygunluk değeri daha düşük olan birey seçilir. Bu iki birey orjinal topluluktan çıkarılmaz, böylece daha sonra tekrar seçilebilirler. Yöntemin bir analizi Goldberg ve Deb tarafından (1991) yapılmıştır.

Boltzmann Seçilimi: Seçilim baskısının bir koşu sırasında, çeşitli ölçekleme yöntemleri ile (ör; sigma ölçeklemesi) sabit tutulmaya çalışılması yaygındır. Ancak bazen, belirli zaman aralıklarında farklı miktarlarda seçilim baskısına ihtiyaç duyulabilir. Örneğin başlangıçta düşük uygunluk değerine sahip bireylerin de yüksek uygunluktaki bireyler oranında çoğalmasına izin verilerek, yavaş bir seçilim ile topluluktaki çeşitliliği gözönünde bulundurmak akıllıca olabilir. Daha sonra ise, başlangıçtaki çeşitliliğin topluluğu doğru arama uzayına yönlendirdiği varsayılarak, güçlü bir seçilim ile yüksek uygunluk değerine sahip bireylere vurgu yapılmak suretiyle evrimin devamlılığı sağlanır. Benzetimli tavlama yaklaşımına benzeyen bir yöntem olan Boltzmann seçilimi bu farklı ihtiyaçlara cevap veren bir ayarlama mekanizması sunar. Sürekli değişim gösteren bir "sıcaklık" değeri seçilim oranını önceden belirlenmiş bir plana göre kontrol eder. Sıcaklık yüksek seviyede başlar, buna bağlı olarak seçilim baskısı düşük tutulmuş olur. Böylece hemen her birey makul bir çoğalma ihtimaline sahip olur. Sıcaklık yavaş yavaş düşürülerek, seçilim baskısının kademeli olarak artması sağlanır. Bu şekilde GA hem en iyi çözüm uzayına ulaşabilecek,

hem de topluluktaki çeşitliliği elverişli bir seviyede tutabilecektir. Goldberg (1990) bu yöntem üzerinde çalışmalarda bulunmuştur.

3.1.4.4 Çaprazlama

GA'da çaprazlama işlemi, iyi çözümlerin farklı bölümlerini birleştirip daha iyi çözümler oluşturabilmek amacıyla kullanılır. Çaprazlamanın en kolay yolu rastgele bir çaprazlama noktası belirleyip, bu noktadan önceki bölümü ilk ebeveynden, sonraki bölümü ise diğer ebeveynden alarak yeni bir birey oluşturmaktır. Permütasyon kodlama yaklaşımının kullanıldığı algoritmalarda sıra mantığını koruyan çaprazlama yöntemlerinin kullanılması gerekir. Herhangi bir Genetik Algoritma için hangi çaprazlama yönteminin kullanılmasının faydalı olabileceği, uygunluk fonksiyonu, kodlama ve GA'nın diğer unsurları ile de doğrudan ilintilidir. Bu etkileşim, halihazırda üzerinde çalışılan açık bir problemdir.

Tek Noktalı Çaprazlama: En temel çaprazlama yöntemidir. Rastgele bir çaprazlama noktası seçilir ve iki ebeveynin o noktadan sonra gelen kısımları değiş tokuş edilir. Buradaki amaç, farklı dizilerdeki blokları yeniden birleştirmektir. Ancak tek noktalı çaprazlama bütün olası dizileri oluşturamaz. Örneğin 11*****1 şeklindeki dizilerin örnekleri ile ****11** şeklindeki dizilerin örneklerini birleştirerek 11**11*1 formunda bir dizinin örneğini meydana getiremez. Tek noktalı çaprazlama altında çok uzun dizilerin de yok olması ihtimali yüksektir. Eshelman vd. (1989) buna "pozisyonel eğilim (*İng. positional bias*)" adını vermektedirler. Bu tanıma göre, çaprazlama ile oluşturulabilecek

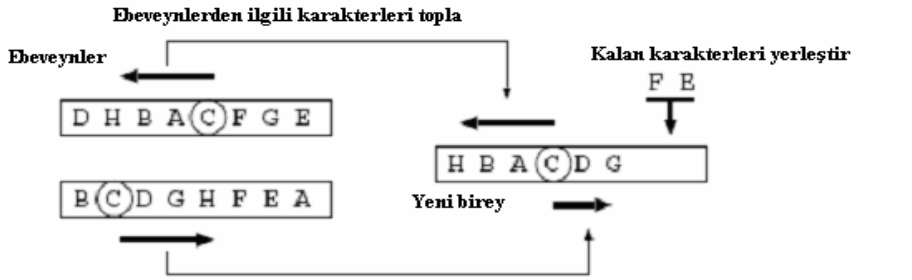
veya yok edilebilecek diziler ağırlıklı olarak kromozomdaki bitlerin yerleşimine bağlıdır. Tek noktalı çaprazlama kısa şemaların diziler için işlevsel yapıtaşı olduğunu varsayar, ancak genellikle işlevsel olarak ilişkili bitlerin hangi sıralama ile yan yana geleceği önceden bilinemez. Eshelman, Caruana ve Schaffer işlevsel olarak ilişkili tüm bitleri yan yana getirmenin mümkün olmadığını, zira bazı bitlerin birden çok şema için önemli olabileceğini belirtmektedirler. Araştırmacılar yöntemin kısa şemaları bir arada tutma eğiliminin istenilen şemaya dahil olmayan ancak dizide birbirine yakın duran, "otostopçu" bitler olarak tanımlanan, bazı bitlerin de çoğalma sırasında diziye taşınmasına sebebiyet verdiğini de eklemektedirler. Ayrıca iki ebeveyn arasında değiş tokuş edilen bölümlerin her zaman dizilerin bitiş noktalarını içerdiği belirtilmektedir.

Çift Noktalı Çaprazlama: Pozisyonel eğilimi azaltmak amacıyla geliştirilen çift noktalı çaprazlama yönteminde, rastgele iki nokta belirlenir ve bu noktaların arasında kalan bölümler değiş tokuş edilir. Çift noktalı çaprazlamanın uzun tanımlanmış şemaları bozma ihtimali düşüktür ve bu yöntem ile tek noktalı çaprazlamadan daha çok dizi oluşturulabilir. Ayrıca değiş tokuş edilen parçalar dizilerin bitiş noktalarını içermez.

Tekdüze Çaprazlama: Spears ve De Jong (1991) gibi bazı araştırmacılar parametreli tekdüze (*uniform*) çaprazlama yönteminin üstünlüğünü savunmaktadırlar. Bu yöntemde, her bit pozisyonunda p olasılığıyla (genellikle $0.5 < p < 0.8$) değiş tokuş gerçekleştirilir. Tekdüze çaprazlamada pozisyonel eğilim yoktur; ebeveynlerdeki farklı

pozisyonlarda bulunan tüm şemalar, çocuk kromozomlarda yeniden oluşturulabilir. Ancak pozisyonel eğilimin eksikliği toplulukta herhangi bütünleşik alellerin oluşumunu engellemektedir, zira tekdüze çaprazlama tüm şemaları bozabilir.

Açgözlü Alt-Tur Çaprazlaması: Sengoku ve Yoshihara'nın (1998) geliştirdikleri bir yöntemdir (*İng. Greedy Subtour Crossover - GSX*). Çaprazlama aşamasında yerel minimumlardan kurtulma yeteneğinden dolayı tercih edilmektedir. Sıklıkla kullanıldığı Gezgin Satıcı Problemi'nde turun belli kısımları için en iyi alt turu içeren çözümler yerel minimumlara neden olabilmektedir. Ancak GSX yöntemi, algoritmanın bu yerel minimumlara takılmasını engelleyebilmektedir.



Şekil 3.7: GSX Yönteminin Gerçekleştirimi

Yöntemi uygularken öncelikle kromozomun güzergah listesinden rastgele bir şehir seçilir. Seçilen şehrin Şekil 3.7'deki gibi C şehri olduğunu varsayılsın. Bu durumda C şehrinin konumu her iki ebeveynde de bulunur ve C şehri yeni bireye eklenir. İlk ebeveyn, seçilen şehrin bulunduğu konumdan sola doğru ikincisi ise sağa doğru dolaşılır. Bu

dolaşma sırasında C şehrinden önceki (kromozomda solunda bulunan) şehirler yeni bireyde C şehrinin soluna, sonraki şehirler ise sağına eklenir. Dolaşma herhangi bir yönde yeni bireye eklenmiş şehre ulaştığımızda sonlandırılır. Dolaşma her iki yönde de durduysa ve hala yeni birey tamamen dolmadıysa, kalan şehirler yeni bireye karışık sırayla eklenir. Orijinal GSX'ten farklı olarak bu kalan şehirler ilgili kromozomdaki sıralarına göre de eklenebilir (Cevre vd., 2007). Yöntemin bu geliştirmesiyle çaprazlama aşamasında rastgelelik oluşmasının önüne geçilebilir. Ayrıca bu versiyon, her çaprazlamada iki birey üretecek şekilde geliştirilmiştir. İkinci birey aynı ebeveynlerin ters yönlerde (ilk ebeveyni seçilen şehirden sağa, ikincisini sola doğru) dolaşılmasıyla elde edilmiştir.

Diğer Çaprazlama Yöntemleri: GSX örneğinde de görüldüğü gibi araştırmacılar çaprazlama için pek çok farklı yöntem geliştirmektedirler. Kromozomun uzunluğuna bağlı olarak ikiden fazla noktadan kesim yapılarak da çaprazlama yapılabilir. Bu yöntemlerin genel adı Çok Noktalı Çaprazlama yöntemleridir. Aritmetik Çaprazlama, özellikle ikili kodlama uygulanan algoritmalarda mantıksal VE (AND) ya da VEYA (OR) işlemlerinin çaprazlama aracı olarak kullanıldığı bir yöntemdir. Çevrim (*Cycling*) (Oliver et al., 1987) adı verilen çaprazlama işlemi özellikle ayrık optimizasyon problemlerinde yaygın olarak kullanılmaktadır. Bunların haricinde Karıştırmalı Çaprazlama, Ara Birleşmeli Çaprazlama ve Doğrusal Birleşmeli Çaprazlama yöntemleri ile sıra mantığını koruyan Sıra Çaprazlaması (*Order Crossover*) (Syswerda, 1991), Değiştirilmiş Çaprazlama (*Modified Crossover*)

(Davis, 1985), Parçalı Planlanmış Çaprazlama (*Partially Mapped Crossover*) (Goldberg and Lingle, 1985) ve Çok Hızlı Çift Onarımlı Çaprazlama (*2-quick / 2-repair*) (Gorges-Schleuter, 1989) yöntemleri belli başlı çaprazlama metotları olarak sayılabilir.

3.1.4.5 Mutasyon

Bireyin bir sonraki nesle geçirilmesi sırasında kromozomu oluşturan karakter dizisinde yapılan rastgele değişikliğe mutasyon denir. Mutasyon, oluşan yeni çözümlerin önceki çözümü kopyalamasını önleyerek çeşitliliği sağlamak ve sonuca daha hızlı ulaşmak amacıyla gerçekleştirilir. Mutasyon olasılığı çok düşük (%0.01 gibi) tutulmalıdır. Yüksek mutasyon olasılığı uygun çözümleri de bozacak ve GA'nın çalışma sırasında problemlerle karşılaşılmasına yol açacaktır. Yer Değiştirme Mutasyonu (*Displacement Mutation*) (Michalewicz, 1992), Değiş Tokuş Mutasyonu (*Exchange Mutation*) (Banzhaf, 1990), Araya Ekleme Mutasyonu (*Insertion Mutation*) (Michalewicz, 1992), Basit Ters Çevirme Mutasyonu (*Simple Inversion Mutation*) (Holland, 1975) ve Ters Çevirme Mutasyonu (*Inversion Mutation*) (Fogel, 1993) sıra tabanlı mutasyon operatörlerinden bazılarıdır.

Holland'ın eserinden başlamak üzere GA topluluğundaki genel kanı, GA'lardaki çeşitliliğin ve yeniliğin en önemli aracının çaprazlama olduğu; mutasyonun ise topluluğun herhangi bir locusta kesin bir duraksamaya uğramasını engelleyerek bir çeşit yardımcı rolü oynadığıdır. Bu görüş, varyasyonun tek kaynağının mutasyon olduğu diğer evrimsel hesaplama metotlarının (evrimsel programlama, evrim

stratejilerinin erken dönem versiyonları gibi) geleneksel duruşuyla ters düşmektedir. Ancak GA'nın karmaşık problemleri nasıl çözdüğünün anlaşılmaya çalışılmasıyla beraber, GA topluluğunda da mutasyona gösterilen itibar artmaktadır. Mutasyon ve çaprazlamanın güçlerinin karşılaştırılmasıyla ilgili pek çok çalışma yapılmaktadır. Örneğin Spears (1993) mutasyon ve çaprazlamanın varolan şemaları bozma konusunda aynı yeteneğe sahip olduğunu, ancak çaprazlamanın yapıcılık konusunda daha sağlam bir kaynak olduğunu kanıtlamıştır. Mühlenbein (1992) ise geleneksel GA'da mutasyonun gücünün azımsandığını, pek çok senaryoda bir tepe-tırmanma stratejisinin çaprazlama içeren bir GA'dan daha iyi çalışacağını öne sürmüştür. Mutasyon ve çaprazlama arasında süregelen bu rekabete karşın, asıl söz konusu olan bu operatörlerin arasında yapılacak bir tercih değil, çaprazlama, mutasyon ve seçilimin arasında kurulacak dengedir.

3.1.4.6 Seçkincilik

Seçilim, çaprazlama ve mutasyon işlemleri sonrasında mevcut topluluğun en iyi uygunluk değerine sahip bireyi bir sonraki nesle aktarılamayabilir. Bunu önlemek için bu işlemlerden sonra, bir önceki topluluğun en iyi (elit) bir veya daha çok bireyi, yeni oluşturulan topluluğa doğrudan aktarılır. Seçkincilik (elitizm) adı verilen bu yaklaşım ilk olarak Kenneth De Jong (1975) tarafından ortaya atılmıştır ve halihazırda genetik algoritmalarda yaygın olarak kullanılmaktadır. Seçkincilik yaklaşımı temelde seçilim evresine yapılan bir katkıdır ve GA'nın performansını dikkate değer bir oranda arttırdığı araştırmacılar tarafından gözlenmektedir.

3.1.5 Genetik Algoritma Parametreleri

Genetik Algoritmalar için kullanıcı tarafından belirlenen genetik operatörler gibi, algoritmanın kontrol parametreleri değerlerinin seçimi de algoritmanın performansı üzerinde oldukça etkilidir. Bir GA için en temel kontrol parametreleri arasında topluluk büyüklüğü, çaprazlama oranı, mutasyon oranı ve maksimum nesil sayısı sayılabilir.

3.1.5.1 Topluluk Büyüklüğü

Topluluk büyüklüğü için seçilen değer, algoritmanın performansını iki şekilde etkilemektedir. Birincisi, topluluk büyüklüğünün aşırı küçülmesi araştırma uzayının yetersiz örneklenmesine sebep olacağından kontrollü ıraksamayı sağlamak zorlaşacak ve araştırma belirli bir alt optimal noktaya doğru sürüklenecektir. İkincisi, topluluk için aşırı yüksek değer seçildiğinde bir nesillik gelişim oldukça uzun süreye ihtiyaç duymaktadır. Bu, özellikle gerçek zamanlı veya çevrimiçi problem uygulamalarında hiç istenmeyen bir durumdur. Bu yüzden topluluk büyüklüğü için uygun bir değer belirlenmelidir (Goldberg, 1989).

3.1.5.2 Çaprazlama Oranı

Bireylerin çoğalması sırasında kromozomlara uygulanacak çaprazlama operatörünün frekansını belirlemek amacıyla kullanılan parametredir. Düşük çaprazlama oranı yeni nesle çok az sayıda yeni yapının (ebeveyninden farklı yeni bireyler) girmesine sebep olmaktadır. Dolayısıyla tekrar üreme operatörü algoritmada aşırı etkili bir operatör

haline gelmekte ve araştırmanın yakınsama hızı düşmektedir. Yüksek çaprazlama oranı araştırma uzayının çok hızlı bir şekilde araştırılmasına sebep olmaktadır. Ama oran aşırı yüksek ise, çaprazlama operatörü benzer veya daha iyi yapıları üretmeden kuvvetli olan yapılar çok hızlı olarak bozulduğundan, algoritmanın performansı düşmektedir.

3.1.5.3 Mutasyon Oranı

Mutasyon operasyonun frekansı, etkili bir genetik algoritma tasarlamak için çok iyi kontrol edilmelidir. Mutasyon operasyonu, araştırma sahasına yeni bölgelerin girmesini sağlar. Yüksek mutasyon oranı, araştırmaya aşırı bir rastgelelik kazandıracak ve araştırmayı çok hızlı olarak ıraksatacaktır. Başka bir deyişle, topluluğun gelişmesine değil tahribatına sebep olacaktır. Bu durumun tersine, çok düşük mutasyon oranının kullanılması ıraksamayı aşırı düşürecek ve araştırma uzayının tamamen araştırılmasını engelleyecektir. Dolayısıyla, algoritmanın alt optimal çözüm bulmasına sebep olacaktır.

3.1.5.4 Maksimum Nesil Sayısı

Bir genetik algoritmanın çalışması iki şekilde bitirilebilir. Birincisi, sezgi ile tahminlenen bir hedef sonuç değerine ulaşılması halinde algoritmanın çalışması durdurulabilir. Ancak Genetik Algoritmalar her zaman için optimal sonucu garantilemediği için, algoritma alt optimal bir sonucun etrafında da dolaşabilir. Bu durumda belirlenen hedef sonuç değerine ulaşılamayacağı için, algoritmanın çalışmasını bitirebilecek ikinci unsur olan maksimum nesil sayısı devreye girer. Algoritma

maksimum nesil sayısı parametresi ile belirlenen sayı kadar nesil boyunca çalışır ve hedef sonuca ulaşamaması halinde durur. Maksimum nesil sayısının çok az olarak belirlenmesi sonuca yeterince yakınsanamamasına yol açar, koşuyu erken bitirerek algoritmanın başarısını azaltır. Maksimum nesil sayısı parametresinin çok yüksek tutulması ise varolan parametre kümesi ile algoritmanın daha iyi bir sonuç bulamayacağı durumlarda, yani algoritmanın daha fazla yakınsayamayacağı hallerde GA'nın gereksiz bir biçimde fazladan çalışmasına yol açar. Bu da özellikle çözüme ulaşma süresinin kritik olduğu gerçek zamanlı veya çevrimiçi uygulamalar için istenmeyen bir durumdur.

3.1.5.5 Parametre Değerleri ile İlgili Çalışmalar

De Jong (1975), GA'nın performansı üzerinde kontrol parametrelerinin etkisini incelemek amacıyla çeşitli test problemleri kullanarak çalışmalar yapmış ve bu çalışmalar sonucunda iyi bir çevrimiçi ve çevrimdışı performansı elde etmek için kontrol parametrelerine uygun değerler önermiştir (Bkz: Çizelge 3.1). Çevrimdışı performans, tüm nesillerde toplulukta bulunan sadece en iyi çözümün kalite değerini kullanmak suretiyle hesaplanır. Çevrimiçi performans ise tüm nesiller boyunca toplulukta bulunan çözümlerin ortalama kalite değerleri aracılığıyla hesaplanmaktadır.

Grefenstette (1986), GA'nın kontrol parametrelerinin optimize edilmesi için ikinci bir GA'nın kullanılmasının mantıklı olacağını savunmuştur. Bu ikinci Genetik Algoritma (üst seviye GA) De Jong'un

önerdiği parametre değerlerini kullanmaktadır ve seçkincilikten yararlanılmıştır. Bu yöntem ile bulunan parametre değerleri çevrimiçi performansta De Jong'a ufak da olsa üstünlük sağlamış, ancak çevrimdışı performansta geride kalmıştır. Grefenstette tarafından önerilen parametre değerleri Çizelge 3.1'de verilmektedir.

Schaffer vd. (1989) GA'nın performansı üzerinde kontrol parametrelerinin etkisi incelemek amacıyla bir yılın üzerinde CPU zamanı harcayan ayrıntılı bir çalışma yapmış ve çevrimiçi performans için en iyi parametreleri bulduklarını iddia etmişlerdir (Bkz: Çizelge 3.1). Önceki çalışmalarının aksine topluluk büyüklüğünü düşük tutmalarının sebebi çevrimiçi çalışmayı imkanlı hale getirmektir.

İkili kodlu kromozomların kullanıldığı durumlarda optimal topluluk büyüklüğünün ne olması gerektiğinin teorik araştırması Goldberg (1985) tarafından yapılmıştır ve topluluk büyüklüğü ile kromozom uzunluğu arasındaki bağlantı aşağıda verilen denklem ile tanımlanmıştır (Karaboğa, 2005):

$$\text{Topluluk büyüklüğü} = 1{,}65 \times 2^{0{,}2 \times \text{Uzunluk}}$$

Çizelge 3.1: Genetik Algoritmalar için önerilen parametre değerleri (Karaboğa, 2005)

Kontrol parametreleri	**De Jong**	**Grefenstette**	**Schaffer**
Topluluk büyüklüğü	50 – 100	30	20 – 30
Çaprazlama oranı	0.60	0.95	0.75 – 0.95
Mutasyon oranı	0.001	0.01	0.005 – 0.01

Bu çalışmalarda bulunan parametre değerleri tüm uygulama boyunca sabit kalmaktadır. Bununla birlikte optimal parametre değerlerinin problemden probleme değişebileceği aşikardır (Karaboğa, 2005). Yani, herhangi bir uygulamada algoritmadan iyi performans elde etmek isteniyorsa, kontrol parametre değerlerinin yeni optimal setinin çözülecek problem için elde edilmesi gerekmektedir. Bu da oldukça zaman alıcı bir ön çalışmadır. Kullanılacak parametrelerin otomatik olarak değişmesi hem zaman kazandıracak hem de performansın gelişmesini sağlayacaktır. Bu yüzden optimizasyon süresince kontrol parametrelerinin adaptasyonu üzerine çeşitli çalışmalar yapılmıştır. Örneğin, Davis (1989) uygulanan genetik operatörlerin olasılıklarının araştırma esnasında belirlenmesi için bir teknik önermiştir. Teknik, optimizasyon yapılırken farklı operatörlerin izlenen performanslarına göre olasılıklarının ayarlanmasını içermektedir. Ancak bu teknik oldukça karmaşık bir işlem gerektirmektedir.

Whitley ve Hanson (1989), yaptıkları çalışmalar sonucunda uyarlanabilir mutasyon işlemini önermişlerdir. Bu metot, tekrar üretme esnasında ebeveyn kromozomlar arasında *Hamming* mesafesini ölçmek suretiyle çözüm topluluğunun homojenliği dolaylı olarak izlemeyi içerir. Bu işlem, topluluktaki ıraksamayı arttırmak için birbirine daha çok benzeyen ebeveyn çözümlerden elde edilen yeni çözümlere, operatörün uygulanma olasılığının arttırılmasını amaçlamaktadır. Bununla birlikte, araştırmanın sonuna doğru iyi kromozomların kopyalarının tabii olarak toplulukta çok fazla olacağından bu strateji ileriki nesillerde bozucu etkisi gösterebilmekte ve bundan dolayı yakınsamayı geciktirebilmektedir.

Fogarty (1989), iki farklı değişken mutasyon stratejisi önermiştir. İlki, bir kromozomun farklı parçaları için farklı mutasyon olasılıklarının kullanılmasıdır. Bununla birlikte, olasılıklar tüm kromozomlarda ve tüm nesiller boyunca sabittir. İkinci stratejide ise bir kromozomun tüm parçaları için aynı mutasyon olasılığı uygulanmakta ve daha önceden belirlenen belirli nesil sayısından itibaren olasılık değeri sabit bir seviyeye indirgemektedir. Her iki stratejide de mutasyon oranlarının değişimi daha önceden belirlenmektedir.

3.1.6 Genetik Algoritmaların Avantajları ve Dezavantajları

Genetik Algoritmalar diğer arama / eniyileme yöntemlerinden belirli noktalarda farklılaşır. GA stokastik bir arama metodudur. Probleme tek bir çözüm aramak yerine bir çözüm kümesi üzerinde çalışır. Olası çözümlerin tümü hemen yaratılmaz. Optimum çözüme olası çözümlerin bir bölümü üzerinden gidilir. Genetik Algoritmalar deterministik değildir, olasılık üzerine kurulmuştur. Bir problem uzayında ilerleyebilmek için sadece ilgili noktaların uygunluk değeri bilgisine ihtiyaç duyar. Aynı anda birden çok noktada çözüm aradığı için paralel çalışır. Ayrıca örneğin $f(x) = x^4 + y^3 + 6$ gibi bir fonksiyonun minimum değerini bulmak için, GA doğrudan x ve y değerleriyle değil, bu değerleri şifreleyen dizilerle ilgilenir.

Bu bilgiler ışığında, eniyileme problemlerinin çözümünde GA'nın kullanılmasının getirdiği avantajlar şu şekilde sıralanabilir:

1. Genetik Algoritmalar karmaşık ve büyük bir çözüm uzayına sahip problemlerde çok etkilidir.

2. Geliştirilmesi ve gerçekleştirimi düşük maliyetlidir.
3. Çözüm uzayında aynı anda geniş bir alandan çok sayıda noktadan araştırmaya başlanır ve pek çok alternatif çözüm sunar.
4. Problem uzayı hakkında varsayımlarla çalışmaz.
5. Diğer yöntemlerle birlikte, melez olarak kullanılabilir.
6. Paralel işletim olanağı tanır.
7. Kesikli ve sürekli değişkenlerle optimizasyon yapılabilir.
8. Yerel minimumları sıçrayarak aşabilir.
9. Türev alma işlemine gerek yoktur.

Bu avantajların yanı sıra, GA'nın beraberinde getirdiği dezavantajlar ise şunlardır:

1. Rastgele arama yapıldığı için probleme bağlı olarak çözüme ulaşmak uzun sürebilir.
2. Sınırlı bir süre içinde en iyi çözümü garantilemez.
3. Teorik altyapısı basittir.
4. Çözüm uzayının özelliklerinden fazla yararlanmaz.
5. Yerel eniyilemelerde etkili değildir.

3.1.7 Genetik Algoritmaların Uygulama Alanları

Genetik Algoritmalar, arama uzayının büyük ve karmaşık olduğu, mevcut bilgiyle sınırlı arama uzayında çözümün zor olduğu, problemin belirli bir matematiksel modelle ifade edilemediği veya geleneksel eniyileme yöntemlerinden istenen sonucun alınamadığı problemlerde etkili ve kullanışlıdır.

GA'nın günümüzdeki uygulamaları ise üç ana gruba ayrılabilir (Reeves, 1995).

Deneysel Uygulamalar: Mevcut diğer optimizasyon algoritmalarına karşı GA'nın üstünlüğünü ispat etmek amacı ile belirli problemlerin çözümlerini bulmak için GA'nın kullanıldığı çalışmalardır. Bu gruba örnek olarak Gezgin Satıcı Problemi (*Traveling Salesman Problem*), Kör Sırt Çantası Problemi (*Blind Knapsack Problem*), Yol Kesme Problemleri (*Bandit Problems*), Grafik Bölme Problemi gibi çalışmalar gösterilebilir.

Pratik Uygulamalar: GA'nın endüstri ve diğer gerçek problemlerin çözümlerinde kullanıldığı uygulamalardır. İkinci grup için Nümerik Optimizasyon Problemleri, Çizge Planlama Problemleri, Yerleşim Problemleri ve Görüntü İşleme uygulamaları örnek olarak verilebilir.

Sınıflandırıcı Sistemler Uygulamaları: Bu grup sınıflandırıcı sistemler uygulamalarını ihtiva etmektedir. İşbu sınıf GA'nın tümevarım amacı ile kullanıldığı çalışmaları içerir ve genellikle uzman sistemlerin

bilgi tabanının oluşturulmasında kullanılan uygulamalar bu gruba örnek olarak gösterilebilir.

Uygulama sınıflandırmasından bağımsız olarak Genetik Algoritmaların uygulama alanlarına bakılacak olursa, yöntemin çok geniş bir kullanım alanı olduğunu görülebilir. Genetik Algoritmalar, optimizasyon alanında fonksiyon optimizasyonu ve kombinasyonel optimizasyon için, otomatik programlama ve bilgi sistemleri sahasında bilgisayar çiplerinin tasarımı, ders programı hazırlanması, dağıtık bilgisayar ağlarının tasarımı ve dağıtık sistemlerde dosya tahsisatı için, makine öğrenmesi alanında hava tahmini ve protein yapısı tahmini, yapay sinir ağlarındaki ağırlık hesaplamaları, robotların alıcıları için kural geliştiren sistemlerin tasarımı ve robotlarda yörünge planlaması için, ekonomi alanında fiyat verme stratejilerinin gelişim süreçlerini, kazanç getiren pazarların ortaya çıkış süreçlerini ve yenileşim (inovasyon) sürecini modellemek için, sosyal sistemler alanında karınca kolonilerindeki iz takibi davranışının, çok etmenli sitemlerdeki işbirliğinin ve iletişimin evrimi için, finans sektöründe hisse senedi fiyatlarındaki değişimleri tahminlemek, uluslararası sermaye tahsisi stratejilerini belirlemek, müşterilerin kredi değerliliğini ölçmek ve yatırım araçlarının performanslarını belirlemek için, pazarlama alanında müşteri profili çıkarmak ve veri madenciliği yapmak için; bu alanların dışında tıp, ekoloji, popülasyon genetiği ve sanat alanlarındaki birçok işlem için kullanılabilir.

Genetik Algoritmaların kullanıldığı bazı spesifik problemler şu şekilde sıralanabilir:

- Montaj Hattı Dengeleme Problemi
- Çizelgeleme Problemi
- Tesis Yerleşim Problemi
- Baskı Devre Kartlarında İşlem Sırası Belirleme Problemi
- Atama Problemi
- Hücresel Üretim Problemi
- Sistem Güvenilirliği Problemi
- Taşıma Problemi
- Gezgin Satıcı Problemi
- Araç Rotalama Problemi
- Minimum Kapsayan Ağaç Problemi

3.1.8 Hazır Genetik Algoritma Kütüphaneleri

Geliştiriciler, uygulamalarında Genetik Algoritma altyapısı kurmaları gereken durumlarda, hazır araçlardan da yararlanabilirler. Internet üzerinde pek çok hazır Genetik Algoritma kütüphanesi bulunmaktadır. Her seviyeden geliştiriciler açık kaynak kodlu bu kütüphanelere / paketlere erişerek, kendi ihtiyaçlarına göre bir Genetik Algoritma tasarlayabilirler.

JGAP (Java Genetic Algorithms Package): Java'da yazılmış hazır bir Genetik Algoritma ve Genetik Programlama paketidir. Evrimsel

prensipleri problemlerin çözümlerine uygulayabilmek için kolaylıkla kullanılabilen temel genetik mekanizmalar sağlar. Pakete http://jgap.sourceforge.net/ adresinden ulaşılabilir.

JAGA (Java API for Genetic Algorithms): Her türlü Genetik Algoritma ve Genetik Programlama uygulamasını hızlı ve kolay bir biçimde gerçekleştirmeyi sağlayan bir Java API'sidir. API, çok sayıda hazır GA/GP kodları, metotları ve operatörleri içermektedir. Arayüze http://sourceforge.net/projects/j-a-g-a/ adresinden ulaşılabilir.

GAUL (The Genetic Algorithm Utility Library): GNU lisansı altında yayınlanan esnek bir açık kaynak kodlu programlama kütüphanesidir. Evrimsel veya Genetik Algoritmalar kullanan uygulamalar geliştirmeye yardımcı olmak üzere tasarlanmıştır. Seri veya paralel evrimsel algoritmalar için gerekli olan verilerin yönetimi ve değiştirilmesi için metot ve veri yapıları sağlar. Genetik Algoritmalarla karşılaştırma yapılabilmesi için ilave stokastik algoritmalar da birlikte gelmektedir. Kütüphaneye http://gaul.sourceforge.net/ adresinden ulaşılabilir.

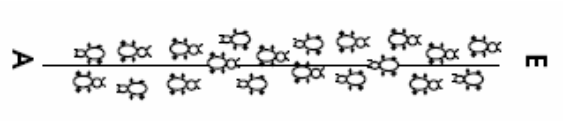
GAlib: Genetik Algoritma bileşenleri için açık bir C++ kütüphanesidir. GAlib içeriğinde bir C++ Genetik Algoritma nesne kümesi barındırır. Kütüphane, hangi gösterim şeklini veya genetik operatörleri kullanıyor olursa olsun herhangi bir C++ programı üzerinde Genetik Algoritmaları kullanarak eniyileme yapmak için gereken araçları içerir. Kütüphaneyle birlikte gelen dokümantasyon bir GA'nın nasıl gerçekleştirilebileceği üzerine kapsamlı bir inceleme sunduğu gibi,

GAlib sınıfları için yapılabilecek özelleştirmelerin resimli örneklemelerini de içerir. Kütüphaneye http://lancet.mit.edu/ga/ adresinden ulaşılabilir.

3.2 Karınca Kolonisi Eniyilemesi

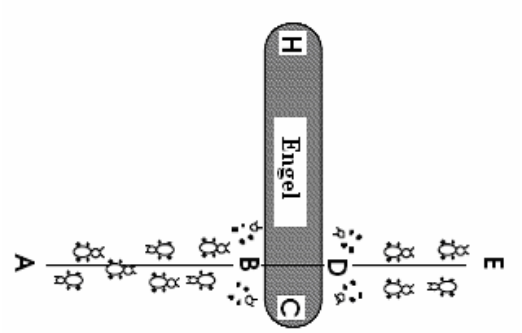
3.2.1 Gerçek Karınca Kolonisi Davranışı

Doğada koloni halinde yaşayan, aralarında belirli bir iş paylaşımının olduğu ve karşılaştıkları problemleri yardımlaşarak birlikte çözen hayvanlara, sosyal hayvanlar adı verilmektedir (Karaboğa, 2005). Bu gruba giren hayvanlara çok sayıda örnek verilebilir. Bu tür hayvanların problem çözme özellikleri örnek alınarak gerçek dünya problemlerinin çözümü için başarılı yeni yaklaşımlar türetilebilir. Karıncalar koloni halinde yaşayan ve problem çözmede örnek alınabilecek önemli davranış özellikleri olan hayvanlardır. Örneğin, karıncalar tam olarak görebilme yeteneğine sahip olmayan, yani çevrelerinde ne olup bittiğini izleyemeyen böceklerdir, ancak buna rağmen yuvalarından yiyecek kaynağına ve tersine yiyecek kaynağından yuvaya uzanan en kısa yolu bulabilmektedirler. Başka bir ifadeyle, çevredeki değişimlere uyum sağlama kabiliyetine sahiptirler. Yani, karıncaların bulup takip ettiği yuva-yiyecek arasındaki en kısa yol herhangi bir şekilde çevre şartlarından dolayı en kısa yol olmaktan çıkarsa (örneğin bir cismin yola konulması suretiyle), mevcut şartlardaki yeni en kısa yolu bulma özellikleri vardır. Şekil 3.8'de gösterilen yuva ile yiyecek kaynağını birbirine bağlayan en kısa hat üzerinde hareket halinde bulunan karıncaların durumunu inceleyelim.



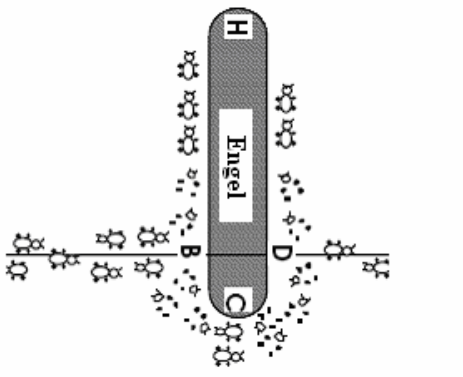
Şekil 3.8: Yuva ile yiyecek arasında karıncaların bulduğu en kısa yol (Dorigo et al., 1996)

Yiyecek ve yuva arasındaki bu en kısa yolu keşfetmek için karıncalar tarafından kullanılan temel maddenin kimyasal feromon maddesi olduğu bilinen bir gerçektir. Karıncalar, hareket halinde bulundukları yola belirli miktarda feromon maddesi bırakırlar. Gitmek için seçecekleri yönün belirlenmesinde bu maddenin miktarı önemlidir. Karıncaların, feromon maddesinin yoğun olduğu yönleri tercih etme olasılığı, daha az feromon maddesine sahip yönleri tercih etme olasılığından daha fazladır. Ancak, feromon maddesinin yoğun olduğu yönün seçilememe olasılığı söz konusudur. Bu basit prensiple hareket eden karıncalar en uygun yolu kısa süre içerisinde bulabilmektedir. Şekil 3.8'de gösterilen yol, önceden keşfedilen en kısa yol olsun ve bu yola bir cisim konularak bozulduğunu varsayalım. Yani, kullanılmakta olan en kısa yol artık en kısa değildir. Bu durum Şekil 3.9'da gösterilmektedir.



Şekil 3.9: Yola bir cisim konulmasıyla en kısa yolun bozulması (Dorigo et al., 1996)

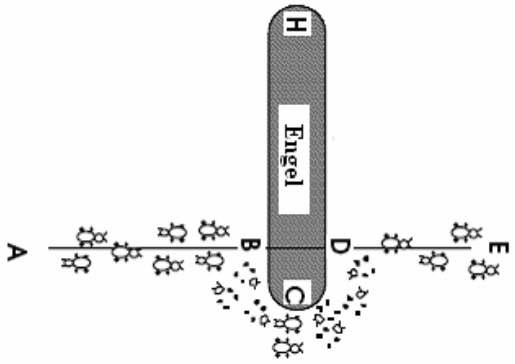
Cisim yola konduğunda cismin hemen önündeki ve arkasındaki karıncalar, tercih edilmesi gereken feromonsuz yönlerle karşılaşırlar. Bu yüzden sağa veya sola dönmeler arasından rastgele birini tercih etmek zorunda kalırlar. Koloni (grup) halindeki karıncaların yaklaşık olarak yarısının sol tarafa dönmeleri beklenir. Bilindiği gibi, kolonideki karınca sayısı arttıkça feromonsuz olarak sağa ve sola dönen karıncaların sayıları arasındaki fark azalacaktır. Başka bir ifadeyle iki grubun sayıları yaklaşık olarak birbirine eşittir. Bu durum Şekil 3.10'da gösterilmektedir.



Şekil 3.10: Cismin konulmasından hemen sonraki durum (Dorigo et al., 1996)

Cisim etrafındaki daha kısa yolu rastgele seçen karıncalar, uzun yolu tercih eden karıncalarla kıyaslandığında, kesilmiş olan feromon maddesi bağlantı yolunu çok daha hızlı şekilde oluştururlar. Bunun sebebi şu şekilde izah edilebilir:

Karıncaların hızlarının hemen hemen aynı olduğunu varsayarsak, birim zamanda kısa yol üzerinden geçen karıncaların sayısı uzun yoldan geçenlerin sayısından daha fazla olacaktır. Kolonideki karıncaların yaklaşık olarak geçtikleri yola eşit seviyede feromon maddesi bıraktıklarını kabul edersek, birim zamanda kısa yola bırakılan koku miktarı daha fazla olacaktır. Yukarıda belirtildiği gibi yön tercihinin koku miktarı ile bağlantılı olması, yeni gelen karıncaların kısa yolu daha yüksek ihtimalle seçmelerine ve bu yola daha fazla kokunun depolanmasına sebep olmaktadır. Böylece, kısa sürede kolonideki çoğu karınca yeni kısa yolu seçerek değişmiş olan çevreye adapte olmaktadırlar. Bu olaydan anlaşılacağı üzere bir pozitif geri besleme söz konusudur: bir yoldan geçen karınca sayısı arttıkça yola yapıştırılan koku miktarı artmakta ve koku miktarı arttıkça da yolu tercih eden karınca sayısı artmaktadır. Bu olay oto-katalitik işlem olarak da adlandırılır. Pozitif geri besleme olayından dolayı, karıncaların çoğunluğunun kısa süre içerisinde daha kısa yolu seçmesi gerçekleşmektedir (Şekil 3.11).



Şekil 3.11: Cismin konulmasından belirli bir süre sonraki durum (Dorigo et al., 1996)

Yukarıda tanımlanan pozitif geri besleme işleminin ilginç yönü şudur: cisim etrafındaki en kısa yolun keşfedilmesi, karıncaların dağıtılmış davranışı ile cisim şekli arasındaki etkileşimin ortaya çıkardığı bir özellik olarak görülebilir. Tüm karıncaların yaklaşık olarak aynı hızda hareket etmelerinden ve yola aynı oranda feromon maddesi bırakmalarından dolayı, cisimlerin daha uzun taraflarından dolaşmak kısa taraflarından dolaşmaktan daha uzun süre alacağı ve bu sebeple feromon maddesinin kısa tarafta daha hızlı toplanacağı da bir gerçektir. Bundan dolayı feromon maddesinin kısa yol üzerinde daha hızlı toplanmasını sağlayan, karıncaların yön seçerken daha yüksek feromon maddesine sahip yönleri tercih etmeleridir.

3.2.2 Karınca Kolonisi Algoritması

Karınca Kolonisi Eniyilemesi (Optimizasyonu) – KKO (*Ant Colony Optimization – ACO),* gerçek karınca kolonisi davranışlarının matematiksel modelleri üzerine dayalı bir algoritmadır. İlk çalışma Dorigo vd. (1991) tarafından yapılmış ve kendi sistemlerini “karınca

sistemi", ortaya çıkan algoritmayı ise "karınca algoritması" olarak tanımlamışlardır. Karınca kolonilerinin davranışlarının tam olarak modellenmesi yerine yapay karınca kolonilerinin bir optimizasyon aracı olarak değerlendirilmesinden dolayı, önerilen algoritmalar gerçek karınca davranışlarından biraz farklı yapıdadır. Örneğin, yapay karıncalar belirli bir hafızaya sahiptir ve gerçek karıncaların tersine tamamen kör değildir. Aynı zamanda, yapay karıncalar ayrık zamanlı bir çevrede yaşamaktadırlar. Herhangi bir dağıtık sistemin tanımlanmasında en temel ve en önemli nokta bireyler arasında haberleşme işleminin nasıl yapıldığının ifade edilmesidir.

Dorigo vd. (1991), çok sayıda bölgesel olarak etkileşen basit bireylerin davranışlarının benzetişimini temel alan karınca algoritmasını zor problemlerin dağıtılmış çözümüne bir yaklaşım olarak tanıtmışlardır. Yani bu yaklaşımda karıncalar basit etkileşen bireylerdir. Dorigo et al, yaptıkları ilk çalışmada üç karınca algoritması tanımlamış ve bu algoritmaları test etmek amacıyla, herkesçe bilinen ve bir ayrık (kombinasyonel) test problemi olan Gezgin Satıcı Problemi'nin çözümünde kullanmışlardır.

Gezgin Satıcı Problemi (GSP), verilen *n* adet şehir için, sanal bir gezgin satıcının her bir şehri bir kez ziyaret etmek şartıyla minimum uzunluklu kapalı bir turu oluşturma problemi olarak tanımlanabilir. *i.* ve *j.* şehirler arasındaki yolun uzunluğu d_{ij} olarak tanımlansın. Öklit tabanlı gezgin satıcı problemi durumunda d_{ij}, *i* ve *j* şehirleri arasındaki Öklit uzaklığıdır. Yani $d_{ij} = [(x_i - x_j)^2 + (y_i - y_j)^2]^{1/2}$ dir. GSP herhangi bir durumda ağırlaştırılmış bir *çizge(N,E)* ile tanımlanabilir. Burada *N*, şehirler kümesi ve *E* ise mesafelerle ağırlıklaştırılmış şehirler arasındaki

bağlantılar kümesidir. *m*, kolonideki toplam karınca sayısını gösterirse ve *t* anındaki *i.* şehirde bulunan karıncaların sayısı $b_i(t)$ (i=1,2,.....,n) ile verilirse bu durumda bu iki büyüklük arasındaki ilişki için

$$m = \sum_{i=1}^{n} b_i(t)$$

bağıntısı yazılabilir.

İlave olarak her karıncanın aşağıdaki özelliklere sahip bir bireyi temsil ettiği kabul edilsin. Karınca,

- *i* şehrinden *j* şehrine giderken *(i,j)* hattına bir miktar feromon maddesi bıraksın,
- Gideceği şehri, o hatta mevcut olan feromon maddesi miktarı ile iki şehir arasındaki mesafenin fonksiyonu olan bir olasılıkla seçsin,
- Karıncalar, şartları sağlayan turu gerçekleştirebilmeleri için zorlanmalıdır. Bunun için ziyaret edilmiş şehirlerin tekrar ziyaret edilmesi yasaklanarak tur tamamlansın.

$\tau_{ij}(t)$, *t* anında *i* ve *j* şehirleri arasındaki *(i,j)* hattında depolanan feromon maddesi miktarını gösterirse *t+1* anındaki feromon maddesi Denklem 1 ile tanımlanabilir.

$$\tau_{ij}(t+1)=\rho\cdot\tau_{ij}(t)+\Delta\tau_{ij}(t,t+1) \qquad \text{(Denklem 3.1)}$$

Burada ρ, buharlaşma katsayısı olarak tanımlanan bir parametredir ve feromon maddesinin sınırsız şekilde büyümesini önlemek için 1'den

küçük pozitif bir değer almalıdır. $(1 - \rho)$ faktörü ise feromon maddesinin buharlaşan oranını belirlemektedir. Birim zamanda, *(i,j)* hattına bırakılan feromon maddesi miktarı ise

$$\Delta\tau_{ij}(t,t+1) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t,t+1) \qquad \text{(Denklem 3.2)}$$

olarak tanımlanabilir ve burada $\Delta\tau_{ij}^{k}(t,t+1)$, *t* ve *t+1* zaman aralığında *k.* karınca tarafından *(i,j)* şehir hattına bırakılan feromon maddesinin birim uzunluk başına miktarıdır.

Bir karıncanın *n* farklı şehri (*n* şehirli tur) ziyaret etmesi şartını sağlaması amacıyla her karıncaya, bir veri yapısı atanır (tabu listesi). Böylece, karınca *t* anına kadar ziyaret etmiş olduğu şehirleri hafızaya alır ve tur tamamlanıncaya kadar bu şehirlerin tekrar ziyaret edilmesi önlenir. Tur tamamlandığında tabu listesi boşaltılır ve bu karınca yolunu seçmek için tekrar serbest kalır. *k.* karıncanın tabu listesini ihtiva eden bir vektör, $tabu_k$ tanımlanır ve $tabu_k(s)$, *k.* karıncanın tabu listesindeki *s.* elemanını yani karınca tarafından ziyaret edilmiş *s.* şehri göstermektedir.

Seçilebilirlik parametresi olarak adlandırılan η_{ij}, $\eta_{ij} = 1/d_{ij}$ şeklinde tanımlandıktan sonra, *k.* karıncanın *i.* şehirden *j.* şehre geçiş olasılığı aşağıdaki bağıntı ile tanımlanır:

$$p_{ij}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{j \in \text{izin verilen}} [\tau_{ij}(t)]^{\alpha} [\eta_{ij}]^{\beta}} & \text{if } j \in \text{izin verilen} \\ 0 & \text{yoksa} \end{cases}$$

(Denklem 3.3)

Burada; izin verilen şehir = {j: j»tabu$_k$} anlamındadır ve α ile β parametreleri kullanıcıya, olasılık değerini hesaplarken yapay feromon maddesi ile seçilebilirlik arasında nispi önemi belirleme imkanı vermektedir. Bundan dolayı geçiş olasılığı, seçilebilirlik değeri ve koku maddesi miktarı arasında bir uzlaşmayı tayin eder. Yani, seçilebilirlik, yakın şehirlerin daha yüksek ihtimalle seçilmesini teşvik ederek bir açgözlü sezgisel kural tanımlarken, koku maddesi miktarı ise bir hatta daha fazla trafik varsa bu hattın daha çok tercih edilmesi gerektiğini belirtir. Bu şekilde koku maddesi, oto-katalitik bir işleme sebep olmaktadır.

3.3 Yerel Eniyileme

3.3.1 Yerel Arama Sezgileri

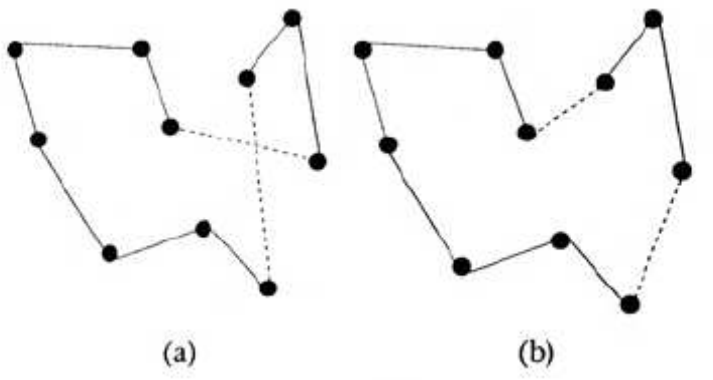
Geleneksel arama yöntemlerinde yeni düğümlerin açılmasında kullanılan yöntem, başlangıç düğümüne olan uzaklığa dayanır. Oysa hedef düğüme olan uzaklığın bilinmesi halinde yalnızca gerekli düğümler açılarak iyileştirme sağlanabilir. Hedefe olan uzaklık tahminlenebilir. Bu tahmine Sezgi (*Heuristic*) ya da *h(n)* adı verilir. İyi bir sezgi, arama süresini üstelden doğrusala indirir. Örnek olarak bir karayolu ağı üzerinde rota planlama problemlerinde, bir noktadan diğer noktaya olan düz hat uzaklığı uygun bir sezgi ölçüsü sayılabilir. En iyi öncelikli arama (*Best-First Search*), A*, yinelemeli derinleşen A* (*Iterative-Deepening A* - IDA**), 2-opt ve 3-opt, belli başlı yerel arama sezgileri arasında sayılabilir. Yerel arama sezgileri evrimsel yöntemler ile beraber kullanıldıklarında gerçekten kaliteli sonuçlar üretmektedirler. Bu

birlikteliklerin en bilinenleri Genetik Algoritmalar ile 2-opt ve 3-opt yöntemleri arasında olup, işbu yöntemler algoritmanın mutasyon aşamasında kullanılabilirler.

3.3.2 Yerel Eniyileme Algoritmaları

3.3.2.1 2-opt

Yerel eniyileme algoritmalarının, özellikle Gezgin Satıcı Problemi için kullanılabilecek en basiti 2-opt yöntemidir. İlk defa Croes (1958) tarafından önerilmiştir. Bu yöntem şehirlerin rastgele bir permütasyonu (*T* turu) ile başlar. *T*'nin komşuluğu, *T*'deki yan yana olmayan iki kenarın yerlerini değiştirerek ulaşılabilecek tüm turların kümesi olarak tanımlanabilir. Bu hareket ikili yer değiştirme (*2-interchange*) adını almakta ve Şekil 3.12'de gösterilmektedir.



Şekil 3.12 : Bir turun (a) ikili yer değiştirme uygulanmadan önce ve (b) uygulandıktan sonraki durumu. Kesikli çizgiler etkilenen kenarları göstermektedir.

Yeni bir *T'* turu, daha iyi olması halinde *T*'nin yerini alır. Turdaki bu değişim her gelişme durumunda gerçekleştirilir, yani *T*'nin komşuluğunda yapılan arama ilk iyileştirme tespit edilince durdurulur. *T*'nin komşuluğundaki hiçbir tur *T*'den iyi değilse, *T* turu *2-optimal*'dir denir ve algoritma sonlandırılır.

3.3.2.2 k-opt

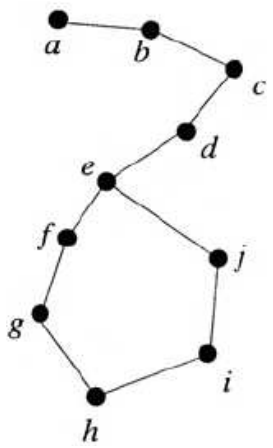
2-opt algoritması rahatlıkla *k*-opt yordamına genellenebilir. Burada *k* veya *k*'ya kadar sayıda kenar çıkarılmak üzere seçilir ve yerlerine daha az maliyetli bağlantılar getirilir. Komşuluğun büyüklüğü ile aramanın verimliliği arasındaki ödünleşme doğrudan konuyla ilgilidir. *k* küçükse (ör; komşuların toplam büyüklüğü azsa), komşuluğun tamamı hızla aranabilir, ancak menzilin kısa olması alt-optimal bir çözüm bulunması ihtimalini arttırır. Diğer yandan, *k*'nın yüksek değerleri için komşuluktaki çözüm sayısı devasa değerlere çekilir, zira altkümelerin sayısı k ile üstel olarak büyür. Bu sebepten *k*-opt algoritması $k > 3$ olan durumlarda nadiren kullanılır.

3.3.2.3 Lin-Kernighan

Özellikle GSP çözümlerinde kullanılan en iyi yöntemlerden biri de Lin-Kernighan algoritmasıdır. Kernighan ve Lin'in 1970 tarihli kendi bildirilerinden yola çıkarak 3-opt'un genelleştirilmesi üzerine kurdukları çalışma ile önerilmiştir (1973). Bu yöntem bir anlamda *k*-opt stratejisini, *k*'nin değerinin iterasyonlar arasında farklılık göstermesine izin vererek, saflaştırır. Ayrıca, *k*-opt'taki gibi ilk iyileştirmeyi değil, en büyük

iyileştirmeyi hedefler. Halihazırdaki en iyi tur daha iyi bir tur bulunduğu anda hemen değiştirilmez.

Temel Lin-Kernighan algoritmasının pek çok farklı versiyonu bulunmaktadır. Burada anlatılan yöntem gösterimde *Hamilton* rotasından ziyade varsayımsal bir δ-rotasını temel almaktadır (Michalewicz, 2004).



Şekil 3.13 : Bir δ-rotası

Bir δ-rotası, düğüm sayısının kenar sayısından bir fazla olduğu ve önceden ortaya çıkmış olan son düğüm dışında hiçbir düğümün birden çok defa gözükmediği bir rota olarak tanımlanabilir. Şekil 3.13, aşağıdaki δ-rotasını resimlemektedir:

a - b - c - d - e - f - g - h- i - j - e

Bu rota bir δ-rotasıdır, zira 10 kenar, 11 düğüme sahiptir ve son düğüm “e” dışında tüm düğümler farklıdır.

Herhangi bir δ-rotası tek bir kenarın, yani son kenarın yerinin değiştirilmesiyle meşru bir tura dönüştürülebilir. Örneğin (j e) kenarını (j

a) ile yer değiştirmek meşru bir tur oluşturur. Ayrıca, son kenar (j e) ile j'den başlayan herhangi bir kenarın yer değiştirilmesi suretiyle yeni bir δ-rotası elde edilebilir (ör; (j e) ile (j f)'yi yer değiştirmek). Bir δ-rotasının bu şekilde bir başkasıyla değiştirilmesine *değiştir(e,f)* diyelim; burada e ve f değiştirilen düğümleri gösterir. Düğümlerin sırası önemlidir: ilk düğüm (e) çıkarılır ve yerine ikinci düğüm (f) konur. Etiketler önemli olmasaydı, bu harekete yalnızca *değiştir* diyebilirdik.

Yeni δ-rotasının maliyeti:

maliyet(j,f) - maliyet(j,e)

kadar artar ve bu "artış" değeri negatif de olabilir. Burada bir δ-rotasının iki adet son kenarının olduğunu belirtmek gerekir. Şekil 3.13'te meşru bir tur oluştururken veya yeni bir δ-rotası üretirken, (j e) ya da (f e) kenarlarından birinin yeri değiştirilebilir.

Lin-Kernighan'ın temel adımlarını anlatmak gerekirse, algoritma rastgele bir *T* turu ile başlar ve bir dizi δ-rotası üretir. İlk rota *T*'den oluşturulur ve bir sonraki rota her seferinde bir önceki rotadan, *değiştir* hareketinin uygulanması ile oluşturulur. Algoritmanın temel mantığına göre bir δ-rotası *p* için maliyet iyileştiren bir *değiştir* hareketi mevcutsa, yeni bir δ-rotası *p* oluşturmak üzere *değiştir* hareketi yapılır.

Lin-Kernighan prosedürü çok hızlı çalışabilir ve binlerce şehirden oluşan GSP'ler için optimale yakın sonuçlar bulabilir. Modern bir iş istasyonunda bir saatin altında bir sürede çözüme ulaşılabilir.

4. ÖNCEKİ ÇALIŞMALAR

4.1 Gezgin Satıcı Problemi

GSP, en iyiye yakın çözümleri bulmaya çalışan kombinasyonlara dayalı eniyileme yöntemleri için standart bir test ortamı haline gelmiştir (Garey and Johnson, 1979). Literatürde, yerel arama algoritmaları, genetik algoritmalar (Goldberg, 1989), benzetimli tavlama (*simulated annealing)* (Laarhoven, 1987), yasak arama (*tabu search*) (Fiechter, 1994), karınca kolonisi optimizasyonu (Angus and Hendtlass, 2005) ve yapay sinir ağları (Jin et al., 2003) gibi eniyileme tabanlı birçok yaklaşım yöntemi önerilmiştir. (Johnson and McGeoch, 1997), GSP için kullanılan sezgiler ve yerel arama algoritmaları hakkında mükemmel literatür taraması niteliğindedir. Literatürde, Dallandır ve Kes (*Branch and Cut*) yöntemi tabanlı büyük GSP örneklerinde bile kesin sonuç verebilen algoritmalar da tanımlanmıştır (Baraglia et al., 2001). *Concorde* bilgisayar kodu, simetrik GSP çözücülere en meşhur örnektir ve 2006'da bir devre kartı içerisindeki 85900 noktanın tümünün en etkin hangi sırada dolaşılması gerektiğini belirlemiştir.

Sengoku ve Yoshihara'nın 1998'te yaptıkları çalışma GSP'nin GA ile çözümü alanındaki en önemli işlerden biridir, çünkü geliştirdikleri algoritma, mutasyon evresinde çok etkili olan 2-opt yöntemini kullanmaktadır. Bu çalışmada, çaprazlama işlemi için, olası en uzun ebeveyn dizisinin alt-turlarını elde eden ve *Greedy Subtour Crossover* (GSX - Açgözlü Alt-tur Çaprazlama) adını verdikleri bir yöntem önermişlerdir. Bu metot ile çözüm, yerel minimumlardan, benzetimli

tavlama yöntemlerinin yapabileceğinden daha verimli bir şekilde kurtulabilmektedir. Ayrıca, doğal seçilim evresinde, en düşük uygunluk değerine sahip olan çözümleri elemişlerdir. Bu yaklaşım seçkinciliğe benzemektedir, ancak en yüksek uygunluk değerine sahip bireyin korunması yerine, en az uygun olan birey topluluktan çıkarılmaktadır.

4.2 Çoklu Gezgin Satıcı Problemi

Çoklu Gezgin Satıcı Probleminin çözümüyle ilgili 1970lerden beri birçok çalışma yapılmıştır. İşbu çalışmalar kombinasyonel eniyileme alanındaki gelişmeler, özellikle de evrimsel hesaplamanın ivme kazanması ile yıllar içinde artmış, ancak yine de belirli bir miktarı geçememiştir. Mevcut olan az ama kaliteli çalışmalara dayanarak, ÇGSP'nin ilgili problemlere nazaran marjinal ancak gelişmeye çok açık bir alan olduğu söylenebilir.

ÇGSP'yi GA kullanarak çözme çalışmaları özellikle araç planlama problemlerine odaklanmıştır (Park, 2001). Araç planlama problemi *m* adet araçtan oluşan bir filonun *n* adet şehri ziyaret etmesini, her şehrin bir (ve yalnızca bir) araç tarafından ziyaret edilmesi koşulu ile, planlamayı içerir. 2003 yılında Carter tarafından yapılan çalışma, ÇGSP için araç planlama problemi dışındaki spesifik problemler üzerinde de örnekler vermiştir (üretim planlama problemi, kalite kontrol planlaması problemi gibi). Yazar ÇGSP'yi GA kullanarak çözmek üzerine yoğunlaşmış ve ÇGSP'nin GA ile çözümünde kullanılmak üzere yeni bir kromozom gösterimi geliştirmiştir. İki parçalı kromozom tekniği adını verdiği yöntemde, daha önceden kullanılan çift kromozom ve tek kromozom

tekniklerinin iyi özelliklerini birleştirmiş, ayrıca geliştirdiği yöntemin her iki yöntemden de iyi performans gösterdiğini deneysel sonuçlarla ispatlamıştır. İki parçalı kromozom tekniğine ilişkin ayrıntılar Bölüm 2'de verilmektedir (Bkz: 2.4.1 Çoklu GSP İçin Kromozom Gösterimleri, İki Parçalı Kromozom Tekniği).

Junjie ve Dingwer'in 2006 senesinde yayınladıkları çalışma, ÇGSP'nin karınca kolonisi optimizasyonu ile çözülmesine ilişkin modern bir örnektir. Yazarlar KKO'yu problemin çözümüne uygularken yetenek kısıtlamasından faydalanmışlardır. Bu, ÇGSP'de satıcılara atanacak şehir sayısını rastgelelikten uzaklaştırmayı hedefleyen bir yöntemdir. ÇGSP'de bir satıcıya tek şehir düşerken, diğer bir satıcıya toplam şehir sayısının yarısı düşebilir. Oysa gerçek dünyada satıcılar benzer yeteneğe sahiptir. Yetenek kısıtlaması problemi pratikteki haline yaklaştırır. Deneysel sonuçlar göz önüne alındığında, yöntemin özellikle çok geniş olmayan şehir kümeleri için *TSPLIB* test problemleri çerçevesinde karşılaştırıldığı yöntemlerden daha etkili olduğu söylenebilir.

Dang, Wang ve Zhao'nun 2007 yılında yayınladıkları çalışmada ise ÇGSP'yi çözmek için Darwin evrimini simüle eden bir GA önerilmektedir. Bu çalışmada ÇGSP'yi standart GSP'ye dönüştürerek çözmek için yeni bir yöntem sunulmuştur. Problemin GSP mantığı ile çözülmeye çalışılması yaygın bir yaklaşımdır ve katman sayısını azaltmaya yöneliktir. Sonuçta bulunan algoritmanın zaman karmaşıklığının mevcut en hızlı sıralama yöntemlerininkine eşdeğer olduğu belirtilmiştir. Ayrıca bu metodolojinin diğer kombinasyonel eniyileme problemlerine de uygulanabileceği vurgulanmaktadır. Uygulama örneği olarak Çin'in şehirlerini dolaşmaya dayanan Çin

ÇGSP'si seçilmiş ve algoritma bu örnek üzerinde Fogel'in GA'sı ve açgözlü algoritma ile karşılaştırılmıştır.

Çoklu GSP bilgisayar mühendisliği dışındaki disiplinlerin de çalışma alanına girdiği için bu alanlarda yapılan bazı önemli çalışmaları da belirtmekte fayda vardır. Örneğin endüstri mühendisliği alanındaki (Bektaş, 2006), Çoklu GSP'nin tanımlanması, kullanım alanları ve çözüm yöntemleri hakkında derleme niteliğinde bir çalışmadır. Çalışmada, ÇGSP tek ve çok istasyonlu türleri açısından ele alınmakta, çözüm için kullanılan algoritmalar hakkında bilgi verilerek, problemin çözümünde kullanılabilecek yeni formüller de önerilmektedir. Yine (Bektas ve Kara, 2006)'da ise probleme, her satıcının belirlenmiş asgari bir sayıda şehri dolaşması kısıdını eklenerek, tek ve çok istasyonlu (*depot*) senaryolar için tamsayı doğrusal programlama formülleri önerilmektedir. Çalışma kapsamında yapılan analizler sonucu, önerilen formülasyonun uygulandığı çok istasyonlu senaryolarda, önceki yaklaşımlara nazaran önemli gelişmeler kaydedildiği gösterilmektedir. Bilgisayar bilimleri alanındaki (Mitrovic-Minic ve Krishnamurti, 2006)'da Çoklu GSP, zaman pencereleri kavramı ile birlikte incelenmekte ve araçlar arasında üstünlük çizgelerini temel alan bağlantılardan söz edilmektedir. Yöneylem araştırma çerçevesinden bakan (Malik et al., 2007)'de ise genelleştirilmiş çok istasyonlu Çoklu GSP için bir yaklaşık algoritma önerilmektedir.

Çoklu GSP'nin çözümü için karşılaştırma yapılabilecek hazır veri setlerinin, kütüphanelerin ve ortamların eksikliği, problem üzerine yapılan tüm çalışmaların önündeki en büyük engel olarak görülmektedir. Bu tez çalışması ile bu engeli giderebilecek bir kütüphane, karşılaştırma

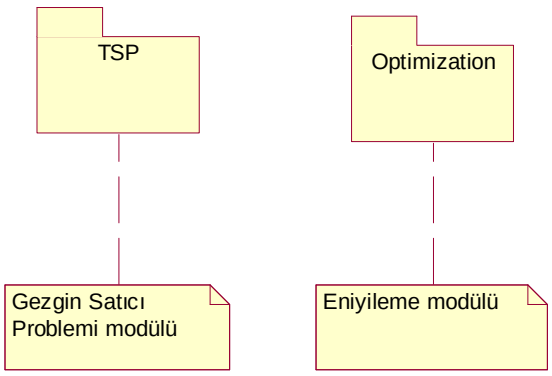
yapmayı kolaylaştıran Web tabanlı görsel ortam ile birlikte sunulmaktadır. Bu alandaki boşluğu doldurarak katkı sağlayan kütüphanenin, kolayca genişletilebilmek üzere nesneye dayalı yaklaşım ile gerçekleştirilmesi de yazılım geliştiriciler için önem arz etmektedir.

5. ÇOKLU GSP KÜTÜPHANESİ

5.1 ÇGSP Kütüphanesinin Tasarımı

Tez projesinin ilk hedefi Çoklu Gezgin Satıcı Problemi için bir eniyileme kütüphanesi tasarlamak olduğu için, işbu hedefte geçen iki kavram olan "Gezgin Satıcı Problemi" ve "Eniyileme" tasarım aşaması için temel teşkil etmektedir. Çoklu GSP özgün GSP'den türetildiğinden, kütüphanenin tek satıcı içeren senaryolar için de kullanılabilir olması gerekmektedir. Kütüphane tasarımında esneklik, genişletilebilirlik ve yeniden kullanılabilirlik gibi unsurlara özen gösterilmiş ve nesneye dayalı paradigma uygulanmıştır. Bu bağlamda *Strategy*, *Template* ve *Singleton* tasarım desenlerinden de yararlanılmıştır.

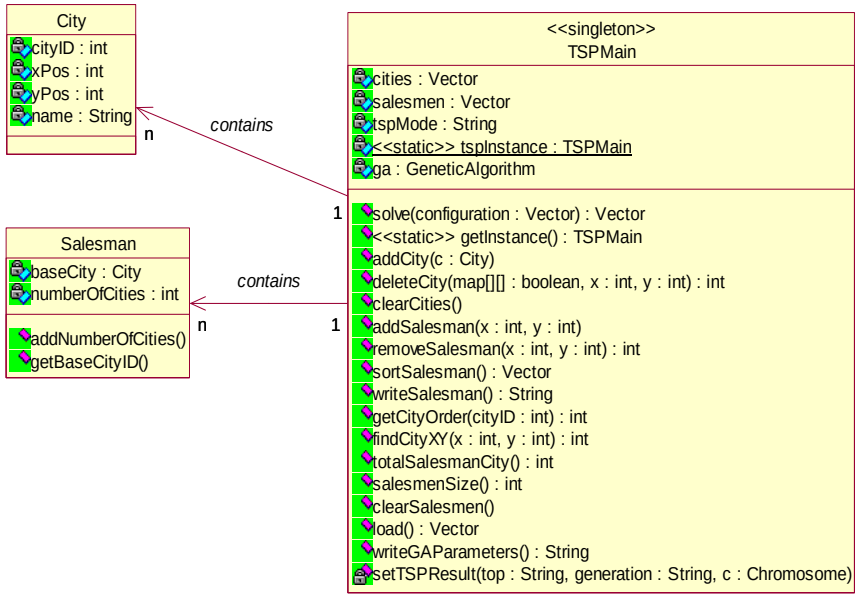
Çoklu GSP Kütüphanesi, Gezgin Satıcı Problemi için *TSP* ve eniyileme için *Optimization* olmak üzere iki modülden oluşmaktadır (Şekil 5.1).



Şekil 5.1: Çoklu GSP Kütüphanesi modülleri

5.1.1 Gezgin Satıcı Problemi Modülü

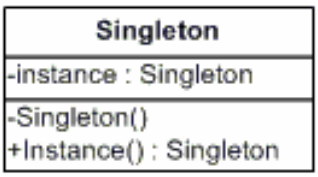
Gezgin Satıcı Problemi modülü ya da *TSP* eldeki problem senaryosunu programa aktararak çözüme ulaşmak için gereken, sınıf, saha ve metotları içermektedir (Şekil 5.2).



Şekil 5.2: Gezgin Satıcı Problemi modülü

GSP, *n* adet şehirden oluşan bir şehir kümesi için oluşturulmuş bir problemdir. Şehir kümesinin problem süresince elde tutulması ve tek bir şehir kümesi üzerinde çalışılması gerekir. Bu sebeple, programın çalışması boyunca şehirleri saklayacak ve şehirler üzerindeki işlemler ile problemin çözümüne dair metotları barındıracak bir sınıf olan *TSPMain* oluşturulmuştur. *TSPMain* sınıfı *Singleton* tasarım deseni kullanılarak yazılmıştır, dolayısıyla tek bir statik örneğe sahiptir ve bu örneğe

yalnızca statik *getInstance* metoduyla ulaşılabilmektedir. *Singleton* tasarım deseninin sınıf diyagramı Şekil 5.3'te görülebilmektedir.



Şekil 5.3: Singleton tasarım deseni (Dofactory, 2007)

Şehirler *City* sınıfıyla temsil edilirler. *X* ve *Y* eksenlerindeki pozisyon bilgileri ile şehir numaralarını da içen bu sınıf, *TSPLIB* verilerinin kullanılmasına elverişli olabilecek şekilde tasarlanmıştır. *City* nesneleriyle ilgili, şehir ekleme (a*ddCity*), şehir silme (*deleteCity*), şehir kümesini sıfırlama (*clearCities*), numarası bilinen bir şehrin kümedeki sırasını bulma (*getCityOrder*) ve *x,y* koordinatları bilinen şehri bulma (*findCityXY*) işlemleri için gereken metotlar *TSPMain* sınıfında bulunmaktadır.

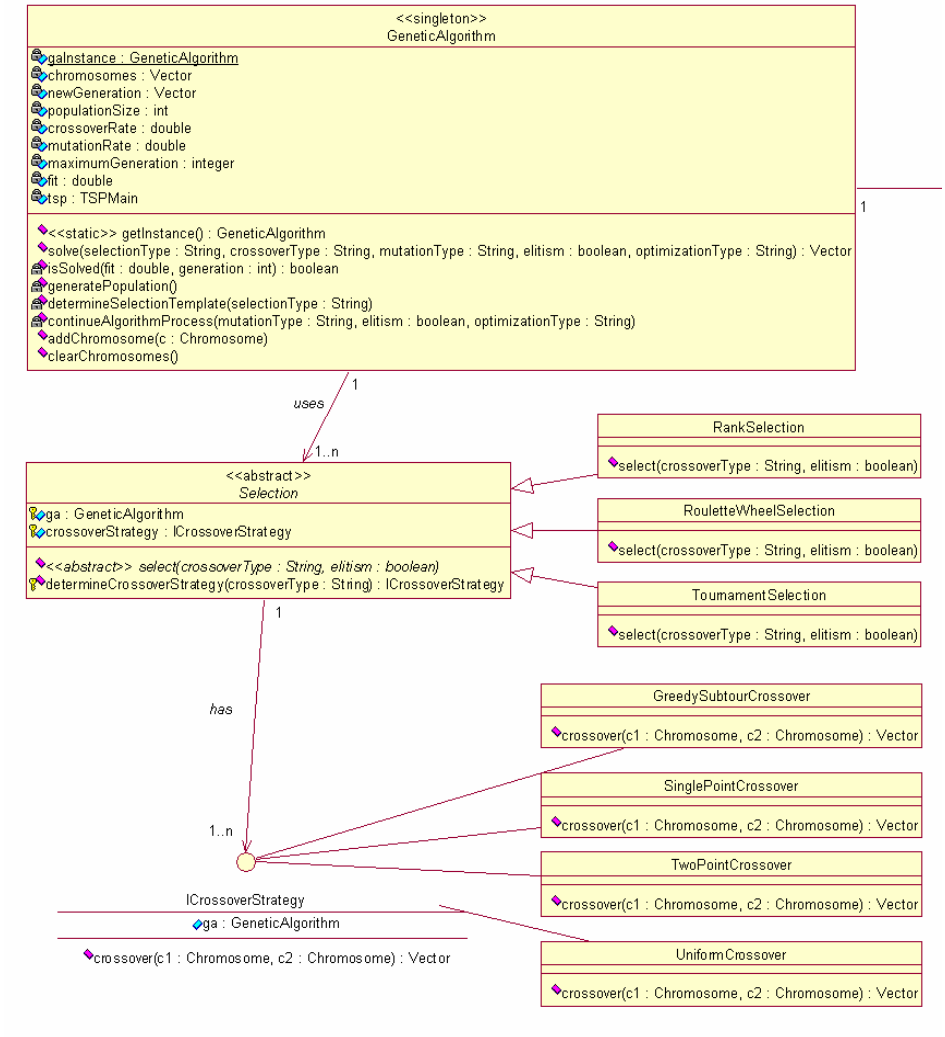
Çoklu GSP şehirler kadar satıcıların da önem kazandığı bir problemdir. Satıcılar *Salesman* sınıfıyla temsil edilirler. İşbu sınıf her bir satıcının yola çıktığı başlangıç şehrini *City* nesnesi olarak tutar (birden fazla satıcı aynı şehirden yola çıkabilir); ayrıca şehir kümesindeki şehirlerden satıcı başına düşen şehir sayısı da *Salesman* sınıfının bir sahasıdır. Satıcı filosu *TSPMain* sınıfında saklanır. *Salesman* nesneleriyle ilgili, satıcı ekleme (*addSalesman*), satıcı silme (*removeSalesman*), satıcıları başlangıç şehrine göre sıralama (*sortSalesman*), problemdeki tüm satıcıları silme (*clearSalesmen*), toplam satıcı sayısını getirme

(*salesmenSize*), satıcıların başlangıç şehirleri haricindeki toplam şehir sayısını getirme (*totalSalesmanCity*) ve satıcı numarası ile başlangıç şehri numarasını formatlı bir şekilde yazdırma (*writeSalesman*) işlemleri için gereken metotlar da *TSPMain* sınıfında yer almaktadır.

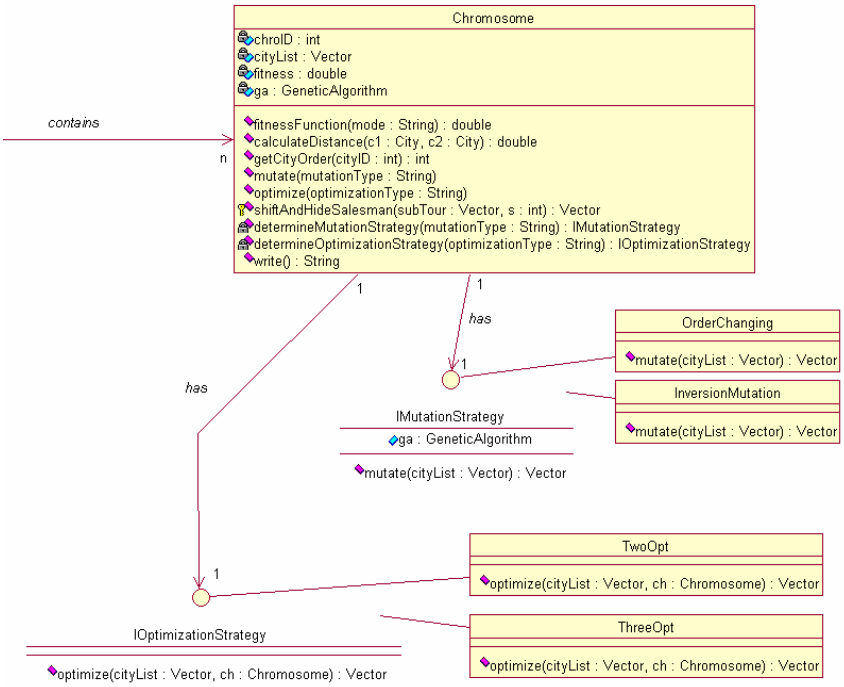
Verilen bir problemin Çoklu GSP olması için iki veya daha fazla satıcı bulunması gerekmektedir. Problemin türü (GSP veya ÇGSP) *TSPMain* sınıfında dinamik olarak belirlenir. *TSPMain* sınıfı problemi çözmek için asıl metot olan *solve* ile ekrana yazdırma ve dışarıdan GSP bilgilerini yüklemeyle ilgili prosedürleri de içermektedir.

5.1.2 Eniyileme Modülü

Eniyileme modülü ya da *Optimization* GSP'nin çözümünün eniyilemesinde kullanılan algoritma ve yöntemler için gereken sınıf, arayüz, saha ve metotları içerir (Şekil 5.4a; 5.4b). Modülün *UML* diyagramı tam sayfa halinde Ek 1'de verilmektedir.



Şekil 5.4a: Eniyileme modülü



Şekil 5.4b: Eniyileme modülü (devam)

Kütüphane kapsamında problemin çözümünde ağırlıklı olarak kullanılan eniyileme yöntemi Genetik Algoritmalar yöntemidir. Bu sebeple modülde GA ile ilgili pek çok sınıf ve arayüz bulunmaktadır. Bunun haricinde 2-opt ve 3-opt yerel eniyilemeleri de kütüphanede yer almaktadır. Varsayılan olarak Genetik Algoritmalar ile diğer eniyileme yöntemlerinden biri melez olarak kullanılmaktadır, ancak tüm eniyileme yöntemleri tek başlarına da kullanılabilecek esneklikte tasarlanmıştır.

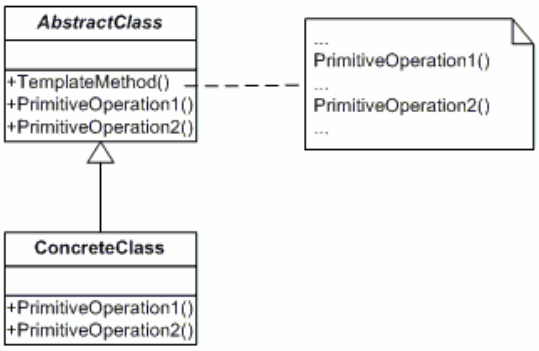
TSPMain sınıfında bulunan şehir kümesinin permütasyonlarını içeren kromozomların ve bu kromozomların genetik operatörler ile yerel eniyilemelerden geçirilmesiyle elde edilen bir sonraki neslin (birden çok

olabilir) program boyunca saklanması gerekmektedir. Ayrıca genetik algoritma parametreleri de - devamlılık açısından - fiziksel olarak bir sınıfta yer almalıdır. Bu sebeplerle, işbu sahaları taşıyan ve algoritmanın çalışması için gereken temel metotları içeren *GeneticAlgorithm* sınıfı bir önceki bölümde anlatılan *Singleton* tasarım deseni kullanılarak oluşturulmuştur. *GeneticAlgorithm* sınıfı ayrıca algoritmanın çözümü bulmak için çalışması (*solve*), hedeflenen çözüme ulaşılıp ulaşılamadığının kontrol edilmesi (*isSolved*), topluluğun ilklenmesi (*generatePopulation*) ile mutasyon, seçkincilik ve yerel eniyileme uygulanması (*continueAlgorithmProcess*) gibi işlemler için gereken metotları içerir.

Şehir kümesinin permütasyonları olan kromozomlar *Chromosome* sınıfı ile temsil edilir. Kromozom numarası, şehir listesi ve şehirlerin uygun sıralanmasından ileri gelen uygunluk değerini de içeren *Chromosome* sınıfı, diğer eniyileme yöntemlerinin GA'dan bağımsız çalıştırılabilmesine de olanak verecek şekilde tasarlanmıştır. Sınıfın içerisinde ayrıca uygunluk fonksiyonu (*fitnessFunction*) ile iki şehir arasındaki Öklid uzaklığının bulunması (*calculateDistance*), numarası verilen bir şehrin listedeki sırasının bulunması (*getCityOrder*), kromozomla ilgili bilgilerin getirilmesi (*write*), problemin Çoklu GSP olması durumunda, iki parçalı kromozom yapısı gereği (Bkz: Bölüm 2.4.1), şehir listesinin sonuna eklenecek *Salesman* nesnelerinin, eniyileme sırasında kaydırılması ve gizlenmesi (*shiftAndHideSalesman*), mutasyon (*mutate*) ve eniyileme (*optimize*) işlemleri için gereken metotlar da bulunmaktadır. *Chromosome* nesnesiyle ilgili kromozom listesine yeni eleman ekleme (*addChromosome*) ve listeyi boşaltma

(*clearChromosomes*) işlemleri *GeneticAlgorithm* sınıfında gerçekleştirilmektedir.

GeneticAlgorithm sınıfı seçilim işlemi için soyut *Selection* sınıfını kullanır. Kütüphane birden fazla seçilim yöntemini desteklediği için, hangi yöntemin kullanılacağı algoritmanın yapısı bozulmadan belirlenebilmelidir. Bu sebeple *Selection* sınıfı altsınıfları olan Sıralı Seçilim (*RankSelection*), Rulet Seçilimi (*RouletteWheelSelection*) ve Turnuva Seçilimi (*TournamentSelection*) ile birlikte, sınıf diyagramı Şekil 5.5'te görülebilecek olan *Template* tasarım deseninden esinlenilerek oluşturulmuştur.

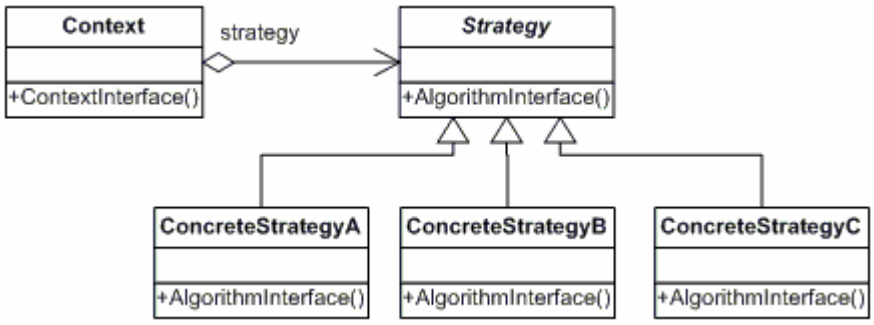


Şekil 5.5: Template tasarım deseni (Dofactory, 2007)

Buradaki yapıda seçilim işleminin yapıldığı metodun (*select*) gerçekleştirimi altsınıflarda yapılmaktadır. Hangi seçilim tipinin uygulanacağına ise *GeneticAlgorithm* sınıfında karar verilir (*determineSelectionTemplate*).

Benzer bir yapı çaprazlama işlemi için de kurulmuştur. Seçilim işleminin içinde yer bulan çaprazlama işlemi için Açgözlü Alt-Tur

Çaprazlaması (*GreedySubtourCrossover*), Tek Noktalı Çaprazlama (*SinglePointCrossover*), Çift Noktalı Çaprazlama (*TwoPointCrossover*) ve Tekdüze Çaprazlama (*UniformCrossover*) altsınıflarında, *ICrossoverStrategy* arayüzünde yer alan çaprazlama metodunun (*crossover*) gerçekleştirimi yapılır. Böylece algoritma, istemcisinden bağımsız olarak değişebilir. Burada uygulanan yöntem, Şekil 5.6'da sınıf diyagramı gösterilen *Strategy* tasarım desenidir.



Şekil 5.6: Strategy tasarım deseni (Dofactory, 2007)

Hangi çaprazlama stratejisinin uygulanacağına *Selection* sınıfında karar verilir (*determineCrossoverStrategy*). İki yapı arasındaki temel farklılık da bu metottan kaynaklanmaktadır. *Selection* sınıfı *Template* desenine birebir uymamaktadır. Çünkü görünüşte bir "*template method*"u yoktur. Ancak çaprazlamaya karar verme sorumluluğunu da taşıdığı için *Strategy* desenindeki gibi basit bir arayüz olmak için fazla karmaşıktır. Ayrıca, karar verme işlemini yapan *determineCrossoverStrategy* metodu, tüm seçilim yöntemlerinin ortak kullandığı bir prosedürdür. Bu yüzden, seçilim yapısında gizli bir *Template* deseni olduğundan bahsedilebilir.

Mutasyon ve eniyileme işlemleri de çaprazlama gibi *Strategy* deseni kullanılarak modellenmiştir. Mutasyonda *IMutationStrategy* arayüzünde bulunan mutasyon metodu (*mutate*) farklı mutasyon stratejilerini içeren Sıra Değiştirme (*OrderChanging*) ve Ters Çevirme Mutasyonu (*InversionMutation*) altsınıfları tarafından gerçekleştirilir. Hangi mutasyon stratejisinin kullanılacağına ise *Chromosome* sınıfının içerisinde dinamik olarak karar verilir (*determineMutationStrategy*). Eniyilemede ise *IOptimizationStrategy* arayüzünde bulunan eniyileme metodu (*optimize*) farklı eniyileme stratejilerini içeren 2-opt (*TwoOpt*) ve 3-opt (*ThreeOpt*) altsınıfları tarafından gerçekleştirilir. Hangi eniyileme stratejisinin kullanılacağına yine *Chromosome* sınıfı içerisinde karar verilmektedir (*determineOptimizationStrategy*).

5.2 ÇGSP Kütüphanesinin Gerçekleştirimi

Tez projesinin ikinci hedefi tasarlanan Çoklu GSP kütüphanesinin görsel yazılım geliştirme ortamı ile birlikte gerçekleştirilmesidir. Görsel yazılım geliştirme ortamının ayrıntıları Bölüm 7'de anlatılmaktadır. Burada ise, çözüm işleminden yola çıkılarak, sınıf diyagramları üzerinden anlatılan tasarımın, Java ile yazılan koda yansıması aktarılmaktadır.

Gezgin Satıcı Problemi için gereken şehir kümesi *TSPMain* sınıfındaki *load* metodu kapsamında dosyadan okunabilir ya da tek tek girilebilir. Her iki durumda da *addCity* metodu kullanılarak şehirlerin tutulduğu *cities* vektörüne yeni oluşturulan bir *City* nesnesi eklenir. Çoklu GSP için satıcılar da yine *load* metoduyla dosyadan okunabilir

veya elle girilebilir. Yeni oluşturulan *Salesman* nesneleri *addSalesman* metodu ile satıcıların saklandığı *salesmen* vektörüne eklenir.

TSPMain.solve

Problemin çözüm işleminin yer aldığı *solve* metodu, varsayılan olarak Genetik Algoritmalar ve Eniyileme yöntemlerinden birinin melez olarak kullanıldığı senaryoyu takip ettiği için, işbu metoda seçilim tipi, çaprazlama tipi, mutasyon tipi, seçkinciliğin kullanılıp kullanılmayacağı ve eniyileme tipi parametrelerinin aktarılması gerekmektedir. Bunlar metoda bir vektör içerisinde geçirilir. Ayrıca varsayılan genetik algoritma parametrelerinden (Çizelge 5.1) farklı değerler ile çalışılmak isteniyorsa, bunlar da *GeneticAlgorithm* sınıfındaki *setter* metotlar vasıtasıyla ya da *TSPMain* sınıfındaki *load* metodu kapsamında dosyadan okunarak programa aktarılabilir.

Çizelge 5.1: Varsayılan GA parametre değerleri

Genetik Algoritma Parametresi	**Varsayılan Değer**
Topluluk büyüklüğü	60
Çaprazlama oranı	0.75
Mutasyon Oranı	0.01
Maksimum Nesil	10
Hedeflenen Uzaklık	0

Eğer *salesmen* vektörünün eleman sayısı 2 veya daha fazlaysa problem Çoklu GSP (MTSP) olarak işaretlenir. Bu noktada eldeki *GeneticAlgorithm* nesnesi üzerinden, *GeneticAlgorithm* sınıfındaki *solve*

metodu çalıştırılır. Bu metot geriye sonuç uzaklığını ve sonuç neslini döndürmektedir (Bkz: Çizelge 5.2).

Çizelge 5.2: GeneticAlgorithm.solve metodu

```
Vector results = ga.solve(selectionType, crossoverType,
mutationType, elitism, optimizationType);
```

Metot dönen değerlerin ekrana yazdırıldığı *setTSPResult* prosedürü ile son bulur.

GeneticAlgorithm.solve

GeneticAlgorithm.solve metodu içerisinde ilk olarak *generatePopulation* metodu ile topluluğun ilklenmesi işlemi gerçekleştirilir. Topluluk, *TSPMain.cities* vektöründen şehirlerin rastgele çekilip yeni kromozom oluşturularak *GeneticAlgorithm.chromosomes* vektörüne atılması ve bu işlemin topluluk büyüklüğü değeri kadar tekrar edilmesi ile oluşur. Çoklu GSP durumunda ise satıcıların başlangıç şehirleri haricindeki toplam şehir sayısı, satıcıların arasında her bir kromozom için ayrı ayrı rastgele paylaştırılır ve satıcıya düşen şehir sayısı sahası olan *Salesman.numberOfCities* değişkenine atanır. *Salesman* nesneleri daha sonra tüm kromozomların sonuna eklenir. Buradaki rastgelelik işleminde de denge gözetilmekte, rastgele sayılar yavaş yavaş arttırılmaktadır. *generatePopulation* metodu kromozomların uygunluğa göre en kötüden en iyiye sıralanması ile son bulur. Bir kromozomun uygunluğu ardışık şehir çiftlerinin aralarındaki Öklid uzaklıklarının toplamıdır. İki boyutlu uzaydaki $P(p_x, p_y)$ ve $Q(q_x, q_y)$ noktaları arasındaki Öklid uzaklığının formülü aşağıda verilmektedir

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Bu noktada yeni bir *Selection* nesnesi yaratılır ve yeni yaratılan nesne, *selectionType* parametresine göre *determineSelectionTemplate* metodu ile ilklenir (Bkz: Çizelge 5.3). *determineSelectionTemplate* metodu bir "*Selection Factory*" gibi çalışmaktadır:

Çizelge 5.3: GeneticAlgorithm.determineSelectionTemplate metodu

```
private Selection determineSelectionTemplate(String
selectionType)
{
      if (selectionType.equals("Rank Selection")) return
      new RankSelection();
      if (selectionType.equals("Roulette Wheel
      Selection")) return new RouletteWheelSelection();
      if (selectionType.equals("Tournament Selection"))
      return new TournamentSelection();
      return null;
}
```

Buradan sonra hedeflenen uzaklığa ulaşılmaması (*isSolved*) ve maksimum nesil sayısına gelinmemesi şartlarıyla girilen bir *while* döngüsü çalıştırılır. Döngüde sırasıyla *Selection.select* metoduyla seçilim ve bu metodun içerisinde bulunan çaprazlama işlemleri yapılır, *GeneticAlgorithm.continueAlgorithmProcess* metodu ile de yeni neslin kromozomları üzerinde mutasyon, seçkincilik ve eniyileme işlemleri gerçekleştirilir. Yeni nesil (*newGeneration* vektörü) *GeneticAlgorithm.chromosomes* vektörüne atanarak sıfırlanır. Nesil sayısının bir arttırılmasıyla döngü işlemleri sonlandırılır. Döngüden çıkıldığında *GeneticAlgorithm.chromosomes* vektörünün son elemanı bulunan en iyi sonucu temsil etmektedir. Gerekli değerlerin döndürülmesiyle metot tamamlanır.

Selection.select

Selection nesnesine *GeneticAlgorithm.determineSelectionTemplate* metodundan dönen *RankSelection*, *RouletteWheelSelection* ve *TournamentSelection* ile olmak üzere üç tür ilkleme yapılabilir. Bunların her birinde yer alan *select* metodu seçilim işlemini yerine getirmesine karşın, bunu yaparken izledikleri yöntemler farklıdır. Tüm seçilim yöntemlerinde toplam kromozom sayısının yarısı uzunluğunda bir döngü içerisinde iki adet kromozom seçilerek çaprazlanır ve yeni nesil vektörüne eklenir. Seçkincilik uygulanması halinde kromozom sayısı çift ise döngü bir eksik döner, böylece daha sonra en yüksek uygunluğa sahip iki kromozomun doğrudan bir sonraki nesle geçirilmesi için yer açılır. seçkincilik uygulanmaması halinde ise kromozom sayısının tek olduğu durumlarda döngü bir fazla döner, ancak seçilim ve çaprazlama sonucunda en düşük uygunluğa sahip olan kromozom listeden çıkarılarak kromozom sayısı sabit tutulmuş olunur.

RankSelection.select metodunda daha önce anlatılan Sıralı Seçilim yöntemi (Bkz: 3.1.4.3 Seçilim) uygulanır. Bunun için düşük uygunluktan yüksek uygunluğa doğru sıralanmış kromozomlar (*GeneticAlgorithm.chromosomes* vektörü) için, en düşük uygunluğa sahip kromozomdan bir adet, bir sonraki kromozomdan iki adet, ondan sonrakinden üç adet şeklinde tüm kromozomlardan sıraları kadar kopyalama yapılarak, bu kopya kromozomlar bir vektöre atılır. Örnekle açıklamak gerekirse, yeni oluşturulan bu seçilim değerleri vektöründe ilk eleman *chromosomes.get*(0), ikinci ve üçüncü elemanlar *chromosomes.get*(1), dördüncü, beşinci ve altıncı elemanlar ise *chromosomes.get*(2) kromozomlarıdır. Bundan sonra girilen seçilim

döngüsünün içinde, seçilim değerleri vektörünün elemanı olan kromozomlardan ikisi rastgele seçilerek, çaprazlama oranı sağlanıyorsa *ICrossoverStrategy.crossover* metodu ile çaprazlanarak yeni nesle atılır; oran sağlanmıyorsa doğrudan yeni nesle aktarılır.

RouletteWheelSelection.select metodunda daha önce anlatılan Rulet Seçilimi (Uygunluk Orantılı Seçilim) yöntemi (Bkz: 3.1.4.3 Seçilim) uygulanır. Gezgin Satıcı Problemi için uygunluk değerleri, tüm şehirlerden geçen turun toplam uzunluğuna eşit olduğu ve bu uzunluk azaldıkça kromozomun uygunluğunun arttığı bilindiğinden, rulet seçilimi için uygunluk fonksiyonundan çıkan uygunluk değerlerinin çarpmaya göre tersi (1/uygunluk) kullanılmaktadır. En yüksek uygunluğa sahip kromozomdan en düşük uygunluğa sahip kromozoma doğru sırayla kromozomların uygunluk değerlerinin çarpmaya göre tersi alınıp, bu sayılar her seferinde kendinden önceki değere eklenerek bir seçilim değerleri vektöründe saklanır. Bu şekilde seçilim yönteminde bahsi geçen "Rulet Tekerleği" oluşturulur. Örnek vermek gerekirse, toplulukta 100, 200, 250 ve 500 uygunluk değerlerine sahip dört kromozom olduğunu varsayalım. Bunların çarpmaya göre tersleri 0.01, 0.005, 0.004 ve 0.002 olacaktır. Bu durumda seçilim değerleri vektörünün elemanları sırasıyla **0.01**, **0.015** (yani 0.01 + 0.005), **0.019** (yani 0.015 + 0.004) ve **0.021** (yani 0.019 + 0.002) olur. Seçilim için girilen döngüde 0 ile tüm uygunluk değerlerinin toplamı (ya da vektörün son elemanı - buradaki örnekte 0.021 -) aralığındaki rastgele bir sayı, hangi sayı aralığındaysa (rulet tekerleğinde nereye tekabül ediyorsa), *chromosomes* vektöründe o konumda bulunan kromozom seçilir. Tekerleğin ikinci kez döndürülmesiyle bir kromozom daha seçilir ve - çaprazlama oranının

sağlanması koşuluyla - iki kromozom çaprazlamadan geçirilerek yeni nesle aktarılır.

TournamentSelection.select metodunda daha önce anlatılan Turnuva Seçilimi yöntemi (Bkz: 3.1.4.3 Seçilim) uygulanır. Yöntemin anlatımında belirtilen *k* değerine 0.7 atanmıştır. Burada seçilim için girilen döngü içerisinde rastgele iki kromozom belirlenir. Daha sonra ise 0 ile 1 arasında rastgele bir *r* değeri oluşturulur. Eğer $r < k$ ise yüksek uygunluk değerine sahip kromozom, aksi halde ise düşük uygunluğa sahip olan kromozom seçilir. Bu işlem diğer kromozomun seçilimi için de tekrarlanır ve - çaprazlama oranının sağlanması koşuluyla - iki kromozom çaprazlamadan geçirilerek yeni nesle aktarılır.

Seçilim için kullanılacak olan çaprazlama stratejisi *determineCrossoverStrategy* metodu ile belirlenir (Bkz: Çizelge 5.4).

Çizelge 5.4: Selection.determineCrossoverStrategy metodu

```
protected ICrossoverStrategy
determineCrossoverStrategy(String crossoverType)
{
      if (crossoverType.equals("Greedy Subtour
      Crossover")) return new GreedySubtourCrossover();
      if (crossoverType.equals("Single Point
      Crossover")) return new SinglePointCrossover();
      if (crossoverType.equals("Two Point Crossover"))
      return new TwoPointCrossover();
      if (crossoverType.equals("Uniform Crossover"))
      return new UniformCrossover();
      return null;
}
```

Programın çalışması sırasında, *ICrossoverStrategy* tipindeki *Selection.crossoverStrategy* nesnesi (ilklenmemişse) *determineCrossoverStrategy* metodunun çalıştırılması ile ilklenir.

ICrossoverStrategy.crossover

ICrossoverStrategy nesnesine *Selection.determineCrossover-Strategy* metodundan dönen *GreedySubtourCrossover*, *SinglePointCrossover*, *TwoPointCrossover* ve *UniformCrossover* ile olmak üzere dört çeşit ilkleme yapılabilir. Bunların hepsinde yer alan *crossover* metodu çaprazlama işlemini yerine getirir ancak uyguladıkları yöntemler farklıdır. Bütün yöntemlerde çaprazlamaya giren iki kromozomun belirli bölümlerinin yer değiştirmesi uygulanır, ancak Çoklu Gezgin Satıcı problemi söz konusu ise kromozomların sonunda bulunan satıcı listesi de tek noktadan bölünerek kendi içinde (eşeysiz) çaprazlamaya, ardından ise satıcı başına düşen şehir sayısı değerinin satıcılar arasında değiştirilerek rastgele azaltılıp arttırılması ile mutasyona uğrar. Değer değiştirme mutasyonu sonucu gelişme sağlanmıyorsa, yapılan işlem geri alınır.

GreedySubtourCrossover.crossover metodunda daha önce anlatılan Açgözlü Alt-Tur Çaprazlaması yöntemi (Bkz: 3.1.4.4 Çaprazlama) uygulanır. Bunun için çaprazlamaya giren iki kromozomun ilkinden rastgele bir şehir seçilerek yeni oluşturulacak ilk kromozomun şehir listesine eklenir. Daha sonra iki kromozomun şehir listeleri, ilk kromozomda seçilen şehirden ileri (sona doğru), ikinci kromozomda ise seçilen şehirden geri (başa doğru) istikamette eşzamanlı olarak taranır ve taranan şehirler seçilen şehrin de eklenmiş olduğu şehir listesine, ilk kromozomdan geliyorsa seçilen şehirden sonraya, ikinci kromozomdan geliyorsa seçilen şehrin öncesine eklenir. Buradaki şart herhangi bir şehrin yalnızca bir defa eklenebilmesi ve taranan kısmın sonuna gelindiğinde taramanın durdurulmasıdır. İleri ve geri yapılan taramalar

sonucu hala şehir listesine eklenmeyen şehirler, ilk kromozomdan sırayla alınarak yeni şehir listesine eklenirler. Böylece yeni kromozomların ilki oluşturulmuş olur. Aynı işlemler ikinci kromozom üzerinde tekrarlanır, ancak bu defa yeni bir şehir seçilmez. Önceden seçilen şehrin ikinci kromozomdaki pozisyonuna göre, tarama bu kez ikinci kromozomda ileriye, ilk kromozomda ise geriye doğru yapılır. Sonuçta eklenmemiş şehirler ise, ikinci kromozomda bulundukları sırayla yeni şehir listesine eklenirler. İkinci kromozomun da bu şekilde oluşturulmasıyla çaprazlama tamamlanmış olur.

SinglePointCrossover.crossover metodunda daha önce anlatılan Tek Noktalı Çaprazlama yöntemi (Bkz: 3.1.4.4 Çaprazlama) uygulanır. Bu yöntemde rastgele belirlenen bir çaprazlama noktası için, bir kromozomun şehir listesinden bu noktaya kadar olan şehirler aynen alınarak yeni listeye atılır, kalan şehirler ise diğer kromozomdaki yerleşim sırasıyla listeye eklenir. İkinci kromozomun oluşturulması sırasında yeni bir çaprazlama noktası belirlenmez.

TwoPointCrossover.crossover metodunda daha önce anlatılan Çift Noktalı Çaprazlama yöntemi (Bkz: 3.1.4.4 Çaprazlama) uygulanır. Bu yöntemde rastgele belirlenen iki çaprazlama noktası için, bir kromozomun şehir listesinden ilk noktaya kadar olan şehirler ile ikinci noktadan şehir listesinin sonuna kadar olan şehirler aynen alınarak yeni listeye atılır, kalan şehirler ise diğer kromozomdaki yerleşim sırasıyla, ilk çaprazlama noktasından itibaren listeye eklenir. İkinci kromozomun oluşturulması sırasında yeni çaprazlama noktaları belirlenmez.

UniformCrossover.crossover metodunda daha önce anlatılan Tekdüze Çaprazlama yöntemi (Bkz: 3.1.4.4 Çaprazlama) uygulanır. Yöntemin anlatımında belirtilen *p* olasılık değeri 0.5 olarak belirlenmiştir. Burada baştan sona doğru ilk kromozomun şehir listesindeki tüm şehirler dolaşılır. Her konum için: eğer 0 ile 1 arasında her seferinde yeniden ve rastgele belirlenen olasılık değeri *p* olasılığından küçükse ve oluşturulacak yeni liste ikinci kromozomun şehir listesinde aynı konumda bulunan şehri içermiyorsa, ikinci kromozomdaki işbu şehir; eğer rastgele belirlenen olasılık değeri *p*'den büyükse ve oluşturulacak yeni liste ilk kromozomun şehir listesinin bulunulan konumundaki şehri içermiyorsa, ilk kromozomdaki işbu şehir yeni listeye eklenir. Kalan eklenmemiş şehirler ise ikinci kromozomdaki yerleşim sıralarıyla listeye eklenir. Aynı işlemler ikinci kromozomun şehir listesinin dolaşılması ile ikinci kromozom için de tekrarlanır, ancak bu defa eklenmemiş şehirler ilk kromozomdaki yerleşim sıralarıyla listeye eklenir.

GeneticAlgorithm.continueAlgorithmProcess

GeneticAlgorithm.solve metodunda seçilim ve çaprazlama gerçekleştikten sonra kromozomlar ve topluluk üzerindeki işlemlerin gerçekleşmesi için *continuAlgorithmProcess* metodu çalıştırılır. Burada yeni nesildeki her bir kromozom için mutasyon oranının sağlanması halinde *Chromosome.mutate* metodu ile mutasyon işlemi, *Chromosome.optimize* metodu ile de eniyileme işlemi gerçekleştirilir. Ancak eniyileme yönteminin 3-opt seçilmesi halinde işlemci yükünü hafifletmek adına, yeni nesildeki kromozomların yalnızca en yüksek

uygunluğa sahip olan %20'lik kesimine eniyileme uygulanır. Bu aşamadan sonra seçkinciliğin geçerli olduğu durumda topluluktaki kromozom sayısı tek ise, topluluktaki en yüksek uygunluğa sahip kromozom; kromozom sayısı çift ise, topluluktaki en yüksek uygunluğa sahip iki kromozom, eniyilemeden geçtikten sonra yeni nesle doğrudan aktarılırlar. Yeni nesil kromozomlar düşük uygunluktan yüksek uygunluğa doğru sıralandıktan sonra, seçkincilik uygulanmamış ve önceki nesildeki kromozom sayısı tek ise, yeni nesildeki en düşük uygunluğa sahip kromozom listeden çıkarılır. Böylece seçilim evresinde artan kromozom sayısı dengelenerek sabit tutulmuş olur.

Chromosome.mutate

Chromosome.mutate metodu, *IMutationStrategy.mutate* metodu için delegasyon içeren bir metottur. *IMutationStrategy* nesnesi *Chromosome.determineMutationStrategy* metodundan dönen *OrderChanging* veya *InversionMutation* ile olmak üzere iki şekilde ilklenebilir (Bkz: Çizelge 5.5).

Çizelge 5.5: Chromosome.determineMutationStrategy metodu

```
private IMutationStrategy
determineMutationStrategy(String mutationType)
{
      if (mutationType.equals("Order Changing"))
      return new OrderChanging();
      if (mutationType.equals("Inversion Mutation"))
      return new InversionMutation();
      return null;
}
```

Her iki yöntemde de *mutate* metoduna parametre olarak aktarılan şehir listesi mutasyon işleminden geçirilerek geri döndürülür. *Chromosome.mutate* metodu, uygunluk değerinin güncellenmesiyle son bulur.

OrderChanging.mutate

Burada listedeki rastgele seçilen iki bitin yerleri değiştirilerek mutasyon sağlanır. Rastgele seçilen ikinci bitin konumu, ilk bitle aynı ise farklı konumdaki bir bit seçilinceye kadar rastgele bit seçimi denenir.

InversionMutation.mutate

Burada listeden rastgele seçilen iki nokta arasında kalan bitlerin dizilişi ters çevrilerek mutasyon sağlanır. Örnek vermek gerekirse, 12**3**45**6**78 şeklindeki bir dizi, 3 ve 6 noktaları arasındaki bitlerin ters çevrilerek yerleştirilmesiyle 12**6**54**3**78 haline gelir. Diğer yöntemde olduğu gibi, burada da rastgele seçilen ikinci konum, ilk pozisyonla aynı ise farklı bir konum seçilene kadar rastgele konum seçimi denenir.

Chromosome.optimize

Chromosome.optimize metodu, *IOptimizationStrategy.optimize* metodu için delegasyon içeren bir metottur. *IOptimizationStrategy* nesnesi *Chromosome.determineOptimizationStrategy* metodundan dönen *TwoOpt* veya *ThreeOpt* ile olmak üzere iki şekilde ilklenebilir (Bkz: Çizelge 5.6).

Çizelge 5.6: Chromosome.determineOptimizationStrategy metodu

```
private IOptimizationStrategy
determineOptimizationStrategy (String optimizationType)
{
      if (optimizationType.equals("2-opt")) return new
TwoOpt();
      if (optimizationType.equals("3-opt")) return new
ThreeOpt();
      return null;
}
```

Her iki yöntemde de *optimize* metoduna parametre olarak aktarılan şehir listesi eniyileme işleminden geçirilerek geri döndürülür. *Chromosome.optimize* metodu, uygunluk değerinin güncellenmesiyle son bulur.

TwoOpt.optimize

TwoOpt.optimize metodunda daha önce anlatılan 2-opt yerel eniyilemesi (Bkz: 3.3.2.1 2-opt) uygulanır. Ancak algoritmanın gücünü arttırmak için şehir çiftlerinin kontrol edilmesi sırasında ilk bulunan iyileştirme değil, her şehir çifti için en başarılı iyileştirme tercih edilir. Bu yaklaşım, en başarılı iyileştirmenin arandığı bir başka yöntem olan Lin-Kernighan algoritmasına (Bkz: 3.3.2.1 Lin-Kernighan) benzemektedir. İşbu durum algoritmanın zaman karmaşıklığını arttırmakta ancak optimal sonuca ulaşmayı da kolaylaştırmaktadır.

Çoklu Gezgin Satıcı Problemi söz konusu olduğunda ise, kromozomun şehir listesinde yer alan *Salesman.City* nesneleri ile satıcının sorumlu olduğu şehirler birleştirilerek birer alt-tur oluşturulur ve

2-opt algoritması bu alt-turlar üzerinde ayrı ayrı uygulanır. Daha sonra ise şehirler satıcının başlangıç şehrinin alt-turdaki konumuna kadar sola doğru kaydırılır ve satıcının başlangıç şehri alt-turdan çıkarılır. Alt-turdaki şehirlerin yeni oluşturulacak listeye eklenmesi ile algoritma sonlandırılır.

ThreeOpt.optimize

ThreeOpt.optimize metodunda daha önce anlatılan k-opt yerel eniyilemesinin (Bkz: 3.3.2.1 k-opt) bir türü olan 3-opt uygulanır. Burada sırayla üçerli olarak tüm şehir çiftleri kontrol edilir. Ancak *TwoOpt.optimize*'da olduğu gibi algoritmanın gücünü arttırmak için şehir çiftlerinin üçerli kontrol edilmesi sırasında ilk bulunan iyileştirme değil, her şehir çifti için en başarılı iyileştirme tercih edilir. Bu şekilde optimal sonuca kolayca ulaşılabilmesine rağmen, zaman karmaşıklığı çok arttığı için algoritmanın kullanımına bir sınırlama getirmekte fayda vardır.

ÇGSP söz konusu olduğunda ise, kromozomun şehir listesinde yer alan *Salesman.City* nesneleri ile satıcının sorumlu olduğu şehirler birleştirilerek birer alttur oluşturulur ve 3-opt algoritması bu alt-turlar üzerinde ayrı ayrı uygulanır. Daha sonra ise şehirler satıcının başlangıç şehrinin altturdaki konumuna kadar sola doğru kaydırılır ve satıcının başlangıç şehri altturdan çıkarılır. Altturdaki şehirlerin yeni oluşturulacak listeye eklenmesi ile algoritma sonlandırılır. Altturlara ayrılarak çalışmanın algoritmanın zaman karmaşıklığı üzerinde iyi yönde etki gösterdiği ve 3-opt'un Çoklu GSP senaryolarında GSP senaryolarından daha kullanışlı olduğu gözlenmiştir.

6. DENEYSEL SONUÇLAR

Çoklu Gezgin Satıcı Kütüphanesi'nin eniyileme kısmında yer verilen algoritma ve yöntemlerin başarısını ölçmek adına, bir dizi standart veri üzerinde deneyler gerçekleştirilmiştir. Bunun için *TSPLIB* kütüphanesinden alınmış, yaygın olarak tercih edilen simetrik GSP örnek kümelerinden olan Berlin52, KroA100, KroB100, KroC100, KroA150 ve KroA200 kullanılmıştır. Kütüphane hem GSP hem de Çoklu GSP problemlerine çözüm üretebildiği için, deneyler sırasında her iki problem de ayrı ayrı ele alınmıştır.

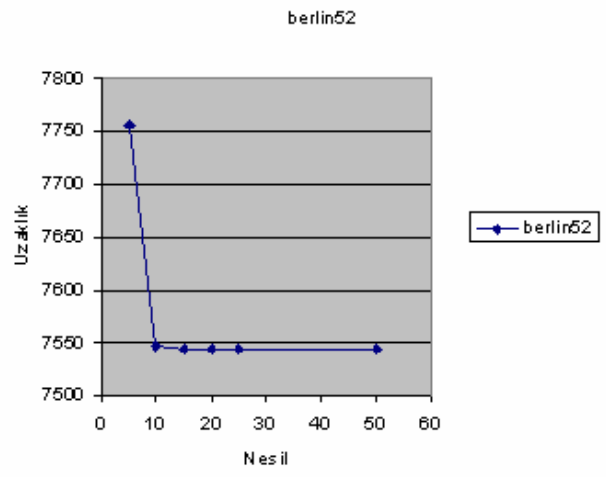
6.1 GSP İçin Elde Edilen Deneysel Sonuçlar

Gezgin Satıcı Problemi'nin çözümü için gerçekleştirilen deneylerde, algoritmanın bulduğu sonuçlar, *TSPLIB* örnekleri için bilinen en iyi sonuçlarla karşılaştırılmıştır. Algoritmanın tüm örnekler için 20'şer defa çalıştırılmasıyla deneyler tekrarlanmış ve her deney için 50'şer nesil üretilmiştir. Bu deneylerde en iyi değerler ile deneylerin ortalamaları kaydedilmiştir. Problemin çözümünde kullanılan yöntem, en iyi sonuçlara ulaşmayı kolaylaştırdığı gözlenen, Sıralı Seçilim, Açgözlü Alt-Tur Çaprazlaması, Sıra Değiştirme, 2-opt ve Seçkincilik metotlarının birlikte kullanıldığı senaryodur. Genetik parametreler, üzerinde problemin çözüldüğü haritanın büyüklüğüne uygun olarak verilmiştir. Çizelge 6.1'de, daha önce sayılan *TSPLIB* örnekleri için elde edilen sonuçlar karşılaştırılmaktadır. Örneklerin isimlerinin sonunda yer alan sayılar, şehir kümesindeki toplam şehir sayısını vermektedir.

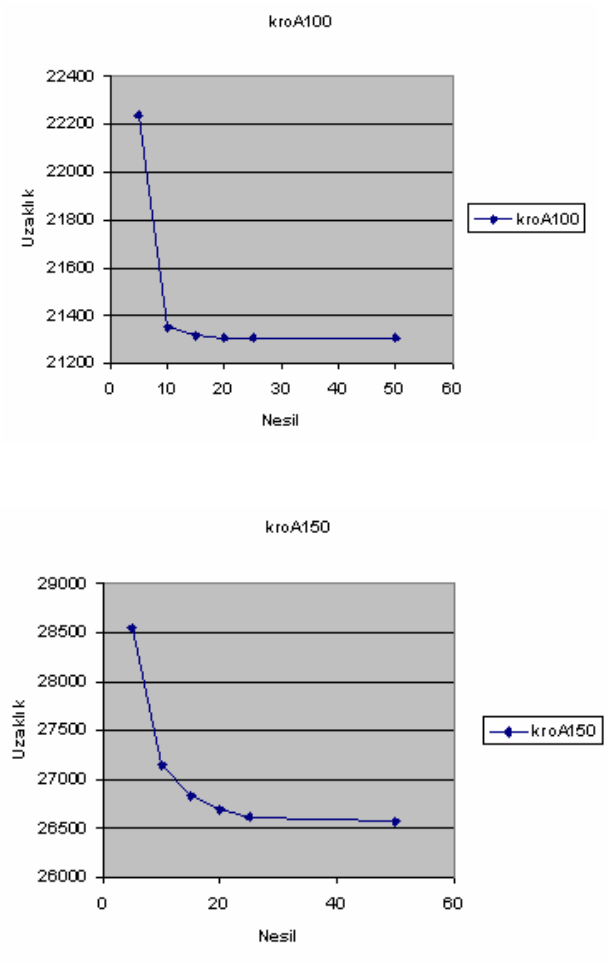
Çizelge 6.1: TSPLIB örnekleri üzerinde GSP için elde edilen sonuçlar

GSP Örneği	Bilinen En İyi Sonuç	En İyi	Ortalama	Hata (%)
Berlin52	7544.37	7544.37	7544.37	0
KroA100	21285.44	21285.44	21303.73	0.08
KroB100	22139.07	22139.07	22167.03	0.1
KroC100	20750.76	20750.76	20778.11	0.1
KroA150	26524.86	26524.86	26578.26	0.2
KroA200	29369.41	29369.41	29582.49	0.7

Uygulanan melez yöntemin az sayılabilecek iterasyondan sonra iyi sonuçlar ürettiği, tüm örnekler için ortalamanın bilinen en iyi sonuçtan farkının %1'den düşük olduğu görülmektedir. Değişik şehir sayılarındaki örnekler için 5, 10, 15, 20, 25 ve 50. nesillerde ulaşılan ortalama değerlerin grafiği Şekil 6.1a ve 6.1b'de verilmektedir.

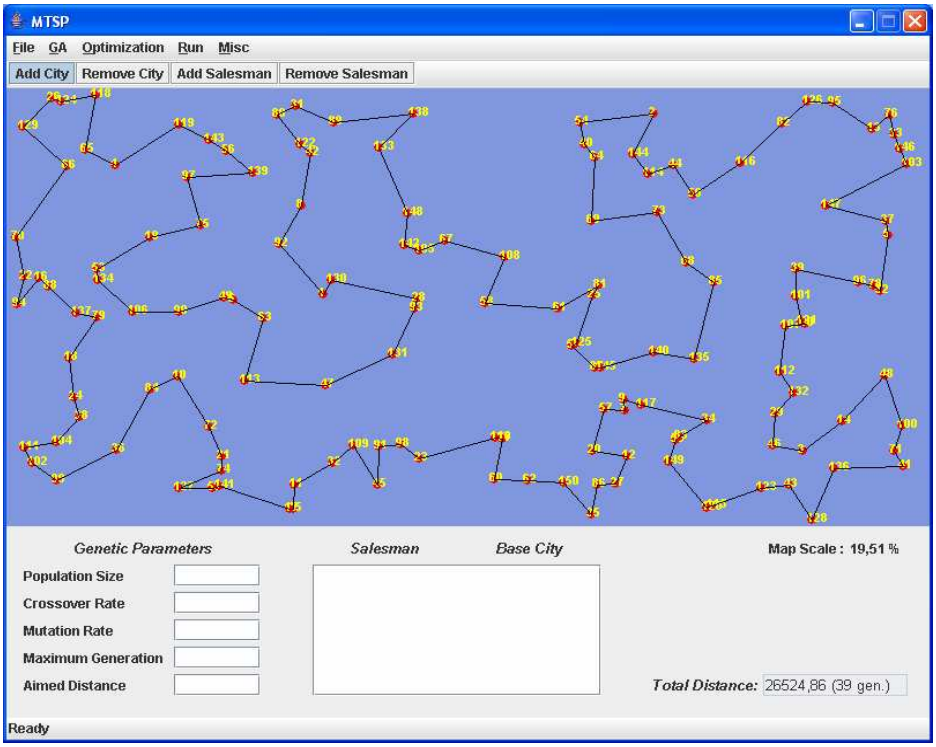


Şekil 6.1a: berlin52 üzerinde GSP için en iyi tur uzunluğunun nesiller boyunca gelişimi



Şekil 6.1b: kroA100 ve kroA150 üzerinde GSP için en iyi tur uzunluğunun nesiller boyunca gelişimi

Şekil 6.1'deki grafikler incelendiğinde, sabit topluluk büyüklüğü için, maksimum nesil parametresi belirli bir değerin üstüne çıktıktan sonra, bu parametrenin öneminin azaldığı görülmektedir. Şekil 6.2'de ise yukarıda ortalama gelişim grafiği verilen kroA150 için bulunan en iyi tur gösterilmektedir.

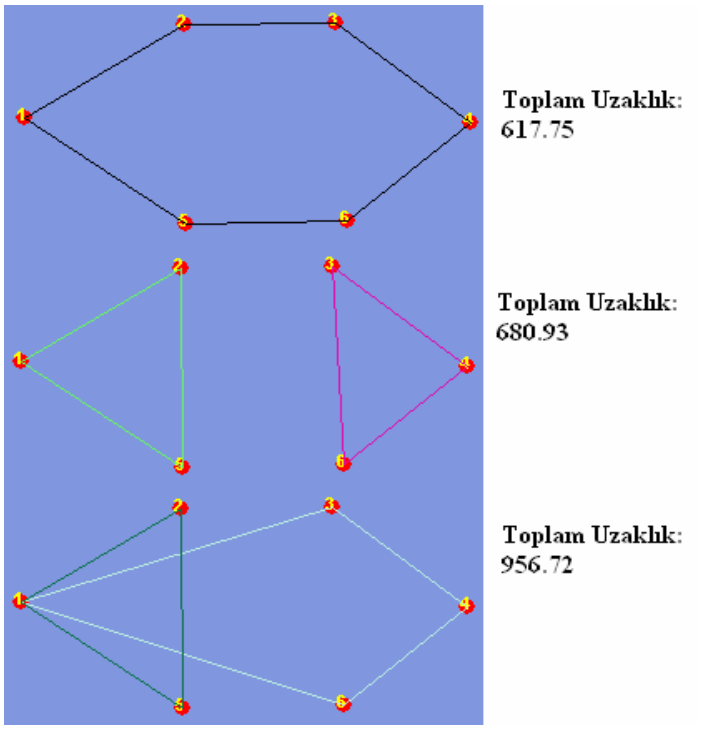


Şekil 6.2: kroA150 için bulunan en iyi tur

6.2 Çoklu GSP İçin Elde Edilen Deneysel Sonuçlar

Çoklu Gezgin Satıcı Problemi'nin çözümü için gerçekleştirilen deneylerde, algoritma çeşitli sayılarda gezgin satıcının bulunduğu senaryolar üzerinde denenmiştir. Literatürde Çoklu GSP için karşılaştırma yapılabilecek standart bir sonuç verisi mevcut olmadığı için, bulunan sonuçlar, *TSPLIB* örneklerinin - özgün GSP için - bilinen en iyi çözümleri ile karşılaştırılmıştır. Bu elbette Çoklu GSP adına adil bir karşılaştırma değildir, zira aynı harita üzerinde birden fazla satıcının bulunduğu senaryolar çoğu zaman tek gezgin satıcının bulunduğu senaryolardan daha büyük toplam uzaklığa sahip çözümler

üreteceklerdir. Sonuçta elde edilen uzaklıkta, gezgin satıcıların başlangıç şehirleri de başlı başına önem kazanmaktadır. Bu konuyla ilgili 6 şehirlik basit bir örnek Şekil 6.3'te gösterilmektedir.



Şekil 6.3: ÇGSP'nin toplam uzaklık açısından GSP ile karşılaştırılması

Şekil 6.3'teki örnek incelendiğinde, aynı şehir kümesine ait üç farklı çözüm içerdiği görülebilir. İlk şekil klasik GSP uygulamasıdır ve optimum tur 617.75 olarak bulunmuştur. İkincide ise 2 ve 3 numaralı şehirlere birer gezgin satıcı eklenmiştir. Eklendikleri yerler itibariyle optimuma yakın sonuç vermeleri beklenmektedir, yine de en iyi sonuçtan açıkça farklı 680.93 sonucu bulunmuştur. Son şekil ise optimum sonuca ulaşması beklenmeyecek, kenardaki 1 numaralı şehre eklenen 2 gezgin

satıcılı senaryoyu göstermektedir. Burada en iyi sonuçtan iyice uzaklaşılmış, turlar üst üste binmiştir. Anlatılan bu durum daha büyük sayılardaki şehir kümeleri için de yaşanmaktadır.

Çoklu Gezgin Satıcı Problemi deneylerinin, başarı için kıstas olarak alınabilecek iki tür sonucu vardır. Birincisi az önce bahsedilen toplam uzaklığı, ikincisi ise sonuç neslinde herhangi bir satıcı için bulunan en uzun turu baz almaktadır. İkinci tür sonuç alınan deneyler, genellikle satıcıların başlangıç şehirlerinin baştan belirlenmediği, kümeleme yöntemleriyle çözüme ulaşılan deneyler olmakla beraber, bir satıcı filosu en yavaş elemanı kadar hızlı olduğundan dolayı, deney sonuçlarında bu verinin de kaydedilmesinde fayda görülmüştür. Algoritma tüm örnekler üzerinde 2, 3, 4 ve 5 gezgin satıcı için ayrı ayrı 20şer tekrar ile çalıştırılmış ve her çalıştırma için 100er nesil üretilmiştir. Bu deneylerde toplamda bulunan en iyi tur uzunlukları, ortalama tur uzunlukları ve herhangi bir satıcı için bulunan en uzun tur değerleri kaydedilmiştir. Problemin çözümünde GSP deneylerinde olduğu gibi, en iyi sonuçlara ulaşmayı kolaylaştırdığı gözlenen, Sıralı Seçilim, Açgözlü Alt-Tur Çaprazlaması, Sıra Değiştirme, 2-opt ve Seçkincilik yöntemleri birlikte kullanılmıştır. Genetik parametreler, üzerinde problemin çözüldüğü haritanın büyüklüğüne uygun olarak verilmiştir. Satıcıların başlangıç şehirleri ise optimuma yakın sonuç vermesi beklenecek biçimde seçilmiştir. Çizelge 6.2'de, daha önce sayılan *TSPLIB* örnekleri için elde edilen sonuçlar karşılaştırılmakta ve elde edilen en iyi çözümde yer alan en uzun alttur uzunluğu verilmektedir. Çizelge 6.3'te ise m=2, m=3, m=4 ve m=5 olarak parantez içinde gösterilen farklı satıcı sayıları

için ortalama değerler verilmektedir. Örneklerin isimlerinin sonunda yer alan sayılar, şehir kümesindeki toplam şehir sayısını vermektedir.

Çizelge 6.2: TSPLIB örnekleri üzerinde ÇGSP için elde edilen en iyi sonuçlar (m=2 için) ve en uzun altturlar

GSP Örneği	Bilinen En İyi Sonuç	En İyi	Kayıp (%)	En Uzun Alttur
Berlin52	7544.37	7586.81	0.6	7310.39
KroA100	21285.44	21381.28	0.4	12035.98
KroB100	22139.07	22532.70	1.8	11524.42
KroC100	20750.76	21316.29	2.7	13827.62
KroA150	26524.86	27151.82	2.4	13915.64
KroA200	29369.41	30565.76	4.0	19022.37

Çizelge 6.3: TSPLIB örnekleri üzerinde ÇGSP için farklı sayıda satıcılar ile bulunan ortalama sonuçlar

GSP Örneği	GSP Ortalama	ÇGSP Ort. (m=2)	ÇGSP Ort. (m=3)	ÇGSP Ort. (m=4)	ÇGSP Ort. (m=5)
Berlin52	7544.37	7640.73	8389.98	9193.87	10135.84
KroA100	21303.73	21784.42	23389.59	26247.82	27857.89
KroB100	22167.03	22962.01	24587.27	27221.00	29986.14
KroC100	20778.11	21731.28	23251.30	25743.47	28719.74
KroA150	26578.26	27601.73	29545.85	29700.99	29479.60
KroA200	29582.49	31184.18	33129.84	35382.72	38717.00

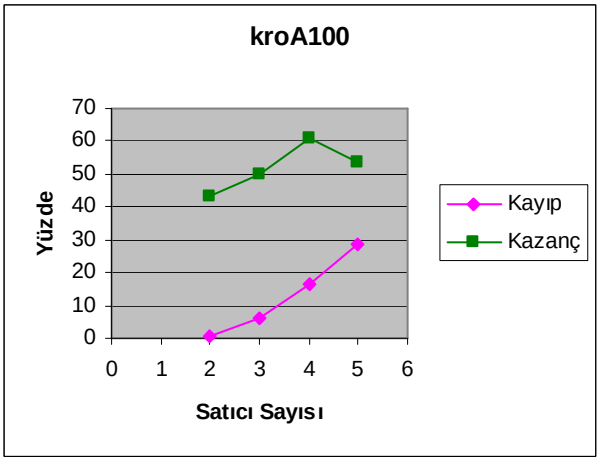
Çizelge 6.2 incelendiğinde, elde edilen en iyi sonuçların *TSPLIB* örneklerinin bilinen en iyi sonuçlarından farkının oldukça az olduğu görülmektedir. İlk iki örnek için %1'den az kayıpla çözüme ulaşılırken, en büyük haritada bile (200 şehirlik kroA200) kayıp %4'ü geçmemektedir. 100er nesil üretilen deneyler, ÇGSP için kısa sürede başarılı sonuçlar alınabileceğini kanıtlamaktadır. En uzun alttur değerlerinin ise görece az şehirli bir harita olan berlin52 dışındaki

örnekler için toplam uzaklığa oranla dengeli bir seyir izlediği görülmüştür. Bu durum, şehir paylaşımının ya da iş yükü dengelemesinin adil ve gerçekçi bir şekilde yapıldığını göstermektedir. Kümeleme yaklaşımı uygulanmamasına rağmen böyle bir sonucun ortaya çıkması, algoritmanın gücüne işaret etmektedir. En iyi sonuçların ve en uzun altturun ise tüm örnekler için 2 gezgin satıcının (m=2) yer aldığı deneyler sonucu elde edildiği izlenmiştir. Çizelge 6.3 incelendiğinde ise, *TSPLIB* örnekleri üzerinde ÇGSP için elde edilen sonuçların kullanılan satıcı sayısına göre, GSP için bulunan ortalama çözümlerden farklılık gösterdiği izlenmektedir. 2 gezgin satıcının bulunduğu deneylerde (m=2) farklar düşük çıkarken, gezgin satıcı sayısı arttıkça ortalama toplam uzaklıkların giderek arttığı gözlenmiştir. Bu durum olağandır, zira her şehri bir ve yalnızca bir satıcı ziyaret ettiği için, tur sayısı arttıkça normalde birbiri ardına ziyaret edilmeyen şehirlerin mecburen arka arkaya dolaşılması söz konusu olmaktadır. Ancak çok satıcılı çözümlerin zaman açısından maliyetinin en uzun alttur kadar olduğu düşünülürse, toplam uzaklıktaki kayıp önemsizleşmektedir. Bu durumun kroA100 haritası için yapılan bir karşılaştırması Çizelge 6.4'te verilmektedir.

Çizelge 6.4: kroA100 için farklı sayıda satıcılar için en iyi sonuç ve en uzun alttur karşılaştırması

Satıcı Sayısı (m)	**Bilinen En iyi Sonuç**	**En İyi Toplam Uzaklık**	**Kayıp (%)**	**En Uzun Alttur**	**Kazanç (%)**
2	21285.44	21381.28	0.4	12035.98	43.4
3	21285.44	22623.82	6.2	10668.32	49.8
4	21285.44	24826.55	16.6	8362.45	60.7
5	21285.44	27455.26	28.9	9923.42	53.3

Çizelge 6.4 incelendiğinde, bulunan en iyi toplam uzaklıkların içerdiği en büyük hata oranının 5 gezgin satıcılı deneyde gözlendiği ve %29'un altında olduğu görülebilir. Bu sonuçlar için kaydedilen en uzun altturlar ise, eşit hızdaki satıcılar için zaman biriminden maliyette, kroA100 örneğinin bilinen en iyi sonucuna göre %43'ten %61'e kadar kazanç sağlandığını göstermektedir. Zaman maliyetinden sağlanan kazanç, toplam yol maliyetinde yaşanan kayıptan daha büyüktür. Ancak bu iki kalemdeki maliyetlere göre problem tipini ve satıcı sayısını belirlemek yine de kullanıcının inisiyatifindedir. Yüzde cinsinden kayıp ve kazançlara ilişkin grafik Şekil 6.4'te verilmektedir.

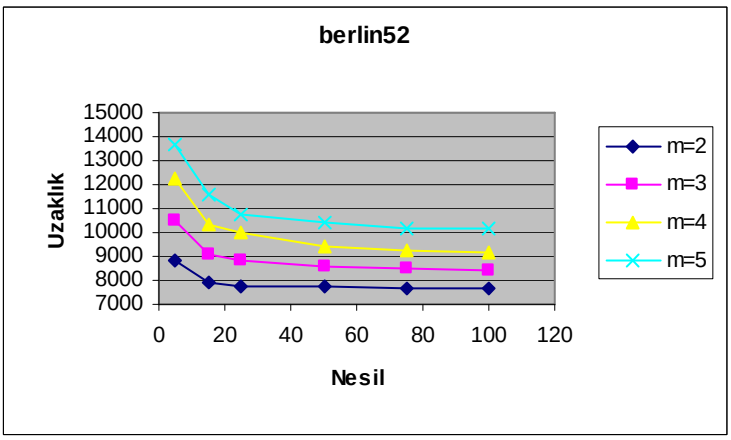


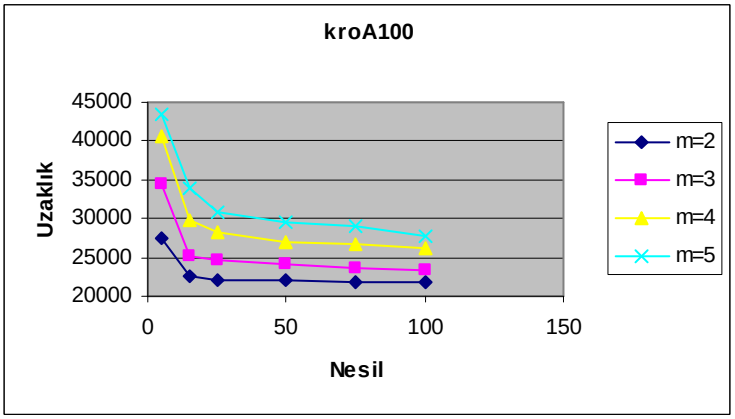
Şekil 6.4: kroA100 haritası için ÇGSP ile zamandan sağlanan kazancın yoldan yaşanan kayıp ile karşılaştırılması

Grafikte satıcı sayısının dördü geçtiği noktada gözlenen kırılmanın, en uzun altturların minimize edilmesine yönelik özel bir çalışma yapılmadığı için, rastlantısal olduğu söylenebilir. Bu konu üzerinde daha

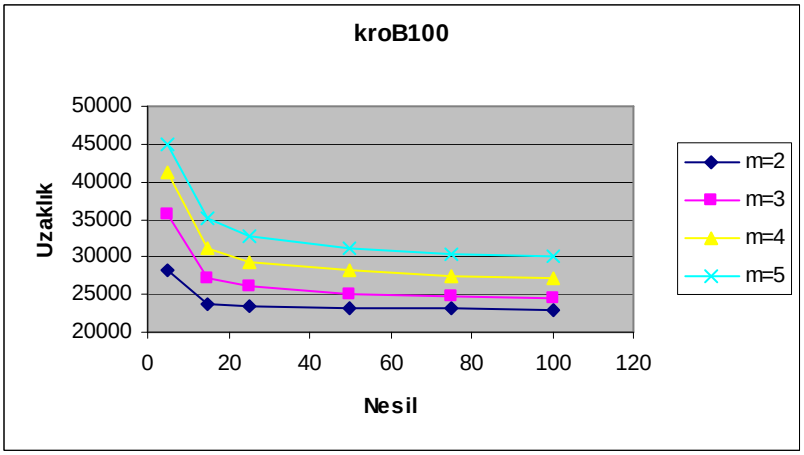
ayrıntılı deneyler yaparak, tüm TSPLIB örnek kümeleri için en uygun satıcı sayılarını belirlemek ise başka bir çalışmanın konusudur.

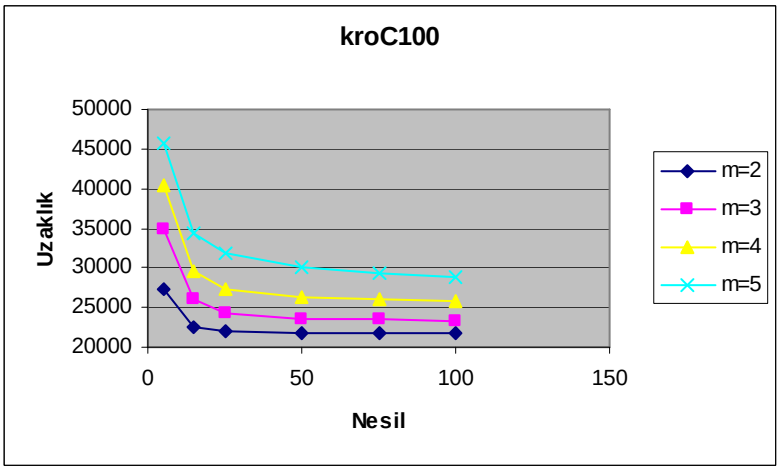
Önceden belirtildiği gibi, Çizelge 6.3'te verilen ortalama değerler 100'er nesillik deneyler sonunda elde edilmiştir. Tüm örnekler için farklı sayılarda satıcı içeren deneylerin 5, 15, 25, 50, 75 ve 100. nesillerdeki ortalamalarına ait grafikler Şekil 6.5a, 6.5b ve 6.5c'de verilmektedir.



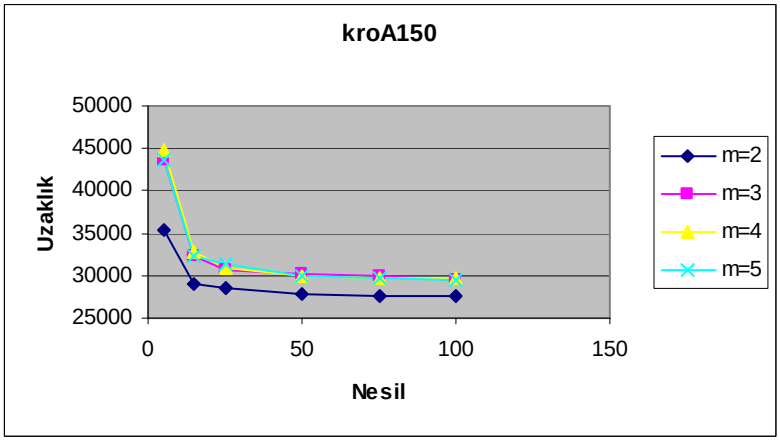


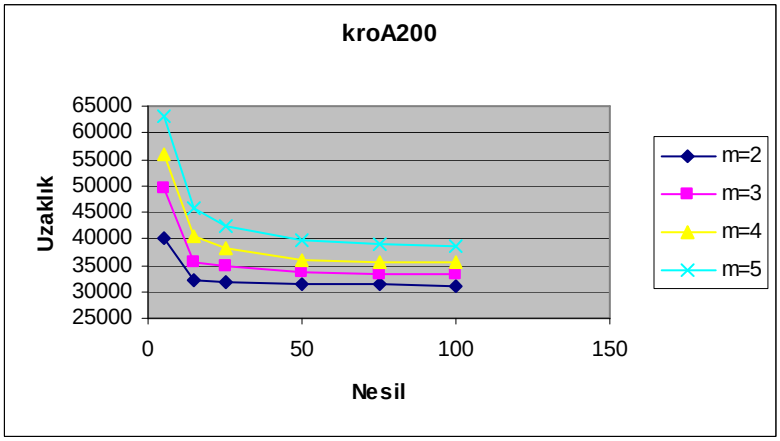
Şekil 6.5a: berlin52 ve kroA100 için en iyi tur uzunluğunun nesiller boyunca gelişimi





Şekil 6.5b: kroB100 ve kroC100 için en iyi tur uzunluğunun nesiller boyunca gelişimi

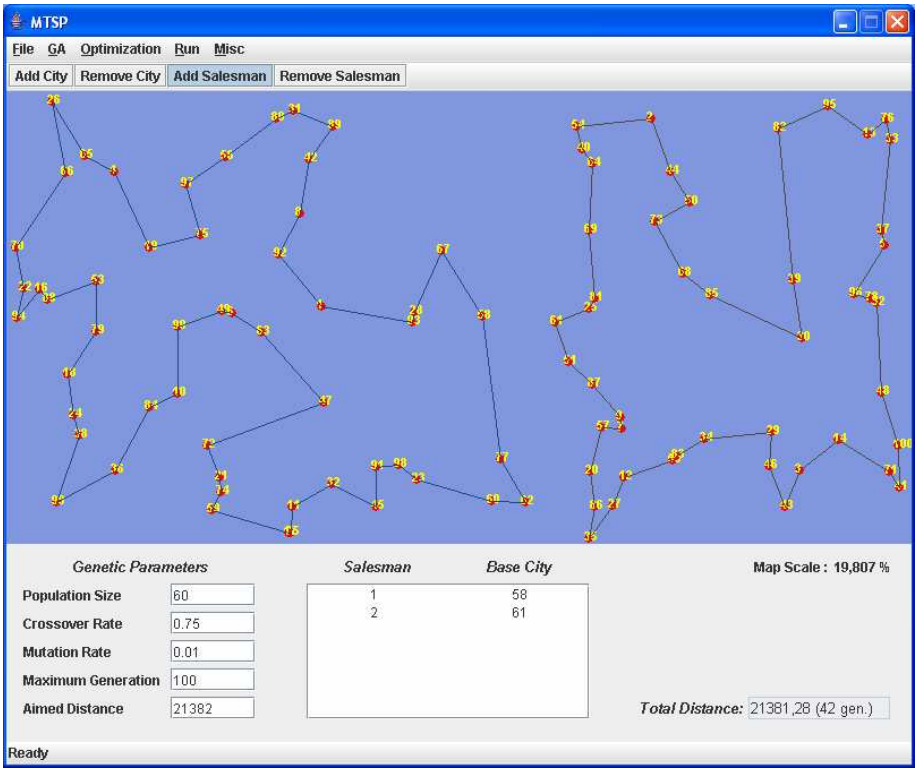




Şekil 6.5c: kroA150 ve kroA200 için en iyi tur uzunluğunun nesiller boyunca gelişimi

Grafikler incelendiğinde, sabit topluluk büyüklüğü için belirli bir nesil sayısına gelindikten sonra eğimin, dolayısıyla gelişmenin azaldığı; maksimum nesil parametresinin önemini kaybetmeye başladığı gözlenmektedir. Ancak GSP'den farklı olarak Çoklu GSP'de maksimum nesil parametresinin etkisi daha fazladır. Deneyler bu yüzden GSP'deki gibi 50şer değil, 100er nesil üretmektedir.

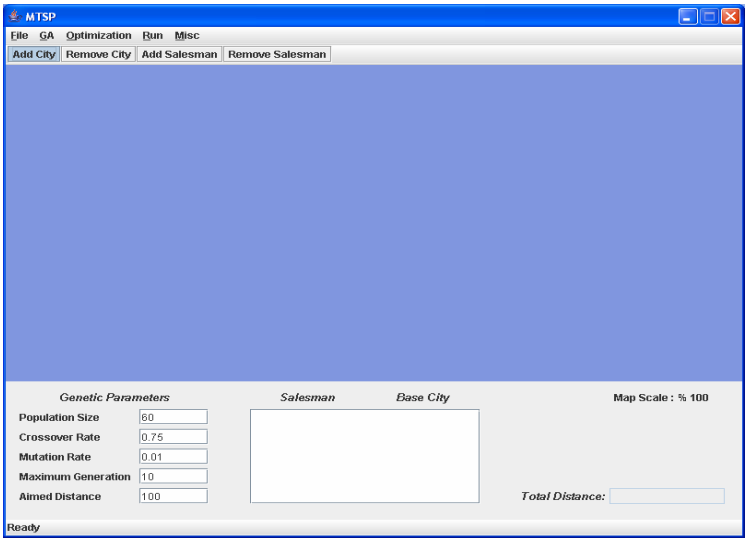
Şekil 6.6'da kroA100 üzerinde 2 satıcı için bulunan en iyi tur gösterilmektedir.



Şekil 6.6: kroA100 üzerinde 2 satıcı için bulunan en iyi tur

7. GÖRSEL YAZILIM GELİŞTİRME ORTAMI

Tez projesi kapsamında, tasarlanan ve gerçekleştirimi yapılan kütüphaneyi kullanan bir görsel yazılım geliştirme ortamı da hazırlanmıştır. Bu uygulama sayesinde, hem Gezgin Satıcı Problemi'nin çözümü için etkileşimli bir grafik arayüz oluşturulmuş hem de kullanıcıların kendi projelerinde kullanabilecekleri, GSP ve Çoklu GSP'nin çözümüne ilişkin otomatik kod oluşturma imkanı sağlanmıştır. Kullanıcılar ayrıca tercih edilen yöntemleri (seçilim, çaprazlama vb.) ve GA parametrelerini, diğer bir deyişle konfigürasyonu *XML* belgesi olarak; üzerinde çalıştıkları haritaları ise *TSPLIB* formatında (*.tsp* uzantılı) saklayabileceklerdir. İşbu ortam *Java Applet* biçiminde oluşturulduğu için, Web üzerinden de kullanılabilmektedir. Görsel yazılım geliştirme ortamının arayüzü Şekil 7.1'de görülmektedir.



Şekil 7.1: Görsel yazılım geliştirme ortamının arayüzü

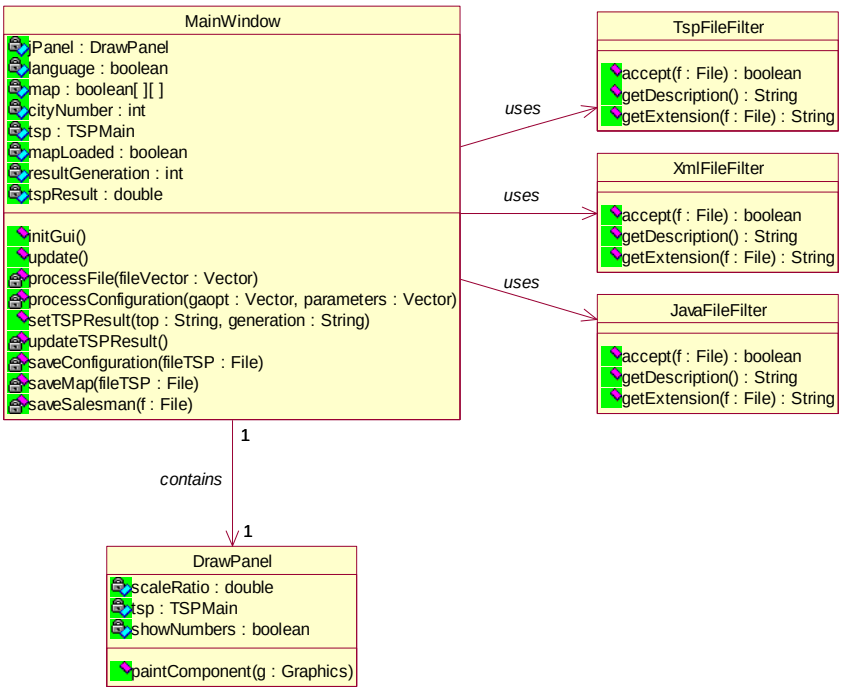
Görsel yazılım geliştirme ortamının sunduğu özelliklerin tamamı aşağıdadır:

- Şehir ekleme / silme
- Koordinat vererek şehir ekleme
- Gezgin satıcı ekleme / silme
- Harita açma / kaydetme
- Rastgele harita oluşturma
- Konfigürasyon yükleme / kaydetme
- Kod oluşturma
- Çeşitli seçilim, çaprazlama, mutasyon ve yerel eniyileme yöntemleri
- GA parametrelerini değiştirme
- Harita ölçeği
- Dil seçeneği
- Şehir numaralarını gösterme / saklama
- Ekran temizleme

7.1 Ortamın Tasarımı ve Gerçekleştirimi

Görsel yazılım geliştirme ortamı tüm işlemlerin gerçekleştirildiği tek modül olan *Graphics*'ten oluşmaktadır. Programın grafik arayüzünü teşkil eden *JFrame*'in bulunduğu ve *JApplet* sınıfından türetilen

MainWindow sınıfı ile GSP/ÇGSP için çizim işlemlerinin gerçekleştirildiği ve *JPanel* sınıfından türetilen *DrawPanel* sınıfları modülün temelini oluşturmaktadır. Bunların dışında, *MainWindow* sınıfının dosya yükleme ve kaydetme işlemleri için kullandığı *JFileChooser* diyalog pencerelerinde filtreleme görevi yapan ve *FileFilter* sınıfından türetilen *TspFileFilter*, *XmlFileFilter* ve *JavaFileFilter* sınıfları ile modül tamamlanmaktadır. Grafik modülünün sınıf diyagramı Şekil 7.2'de görülebilmektedir.



Şekil 7.2: Grafik modülü

7.2 Yazılımın Kullanımı

Görsel yazılım geliştirme ortamının kullanımı için öncelikle şehirlerden oluşan bir haritaya ihtiyaç vardır. Panele fare ile tıklanarak ya da Dosya menüsündeki yöntemlerden biri seçilerek şehirler eklenir ve harita oluşturulur. Bu aşamadan sonra ortamın sunduğu olanaklardan, yapılmak istenilen işleme göre faydalanılır. Bu şehirler üzerinde GSP çözülebilir ya da yine fare ile tıklanarak eklenecek satıcılar ile ÇGSP'ye de çözüm aranabilir. Problemlerin çözümü için izlenecek yöntemler menüden seçilebilir; genetik parametreler değiştirilebilir. Harita ve konfigürasyon kaydedilebilir ya da otomatik kod oluşturularak, yazılım geliştirme işlemi yapılabilir. Arayüz, menüler sayesinde anlaşılır ve kolayca kullanılabilir bir biçimde oluşturulmuştur. Ayrıca klavye kısayolları ile menü seçeneklerine hızlı erişim sağlanmaktadır (Bkz: Çizelge 7.1).

Çizelge 7.1: Klavye Kısayolları

Menü Seçeneği	**Kısayol Tuş Kombinasyonu**
Harita Aç	*Ctrl + O*
Haritayı Kaydet	*Ctrl + S*
Rastgele Harita Oluştur	*Ctrl + R*
Şehir Ekle	*Ctrl + A*
Konfigürasyon Yükle	*Ctrl + L*
Konfigürasyonu Kaydet	*Ctrl + K*
Kod Oluştur	*Ctrl + G*
Çöz	*Ctrl + F5*
Temizle	*Ctrl + C*
Türkçe	*Ctrl + T*
English	*Ctrl + E*
Şehir Bilgisi Göster	*Ctrl + I*

7.2.1 Fare Modu

Panele fare ile tıklanıldığında yapılacak işlemin belirlenmesi için fare modu değiştirilebilir.

Şehir Ekle (Add City): Eğer tıklanılan noktada şehir yoksa, o noktaya yeni şehir eklenir. Dosyadan okunarak açılmış haritalara şehir eklemesi yapılmaz.

Şehir Sil (Remove City): Tıklanılan noktada şehir varsa, o noktadaki şehir silinir. Başlangıç şehri tıklanılan şehir olan satıcılar mevcutsa, şehirle birlikte bu satıcılar da silinir. Dosyadan okunarak açılmış haritalardan şehir silinmez.

Satıcı Ekle (Add Salesman): Tıklanılan noktada şehir varsa, o noktadaki şehir başlangıç şehri olmak üzere yeni gezgin satıcı eklenir. Birden çok satıcının başlangıç şehri aynı olabilir. Eklenen satıcının numarası ile başlangıç şehri pencerede bulunan tabloya yazdırılır.

Satıcı Sil (Remove Salesman): Tıklanılan noktada şehir varsa ve başlangıç şehri bu şehir olan satıcı mevcutsa, o gezgin satıcı silinir. Başlangıç şehri tıklanılan nokta olan birden çok gezgin satıcı varsa, son eklenen satıcı silinir. Satıcının tablodaki referansı da silinir.

7.2.2 Dosya Menüsü

Dosya (*File*) menüsü aşağıdaki seçenekleri içermektedir:

Harita Aç (Open Map): Gezgin Satıcı Problemi'nde kullanılmak üzere çok çeşitli şehir haritaları içeren *TSPLIB* kütüphanesinde bulunan "*.tsp*" uzantılı haritalar ve *TSPLIB* formatında oluşturulmuş diğer haritalar okunarak panele çizdirilebilir. Yeni harita açıldığında panelde çözülmüş bir problem varsa silinir, satıcılar sıfırlanır. Açılan harita panelin boyutlarına (800*400) olan oranına göre ölçeklenir. Harita Ölçeği (*Map Scale*) yüzde olarak panelin sağ altına yazdırılır.

Haritayı Kaydet (Save Map): Panelde bulunan şehir haritası *TSPLIB* formatında "*.tsp*" uzantılı dosya olarak kaydedilebilir.

Rastgele Harita Oluştur (Random Map): Girilen şehir sayısı kadar şehirlik rastgele bir harita oluşturulur.

Şehir Ekle (Add City): Verilen x,y koordinatlarında mevcut şehir yoksa yeni şehir eklenir.

Konfigürasyon Yükle (Load Configuration): TSPSolver.xsd *XML* Şeması biçiminde hazırlanmış *XML* belgeleri okunarak, seçilim, çaprazlama, mutasyon ve eniyileme yöntemleri; seçkincilik uygulanıp uygulanmayacağı; GA parametreleri bilgileri yüklenebilir. TSPSolver.xsd ve kullandığı veri sözlüğü olan GAOptDataDictionary.xsd şemaları Ek 2 ve Ek 3'te verilmektedir.

Konfigürasyonu Kaydet (Save Configuration): Seçilim, çaprazlama, mutasyon ve eniyileme yöntemleri; seçkincilik uygulanıp

uygulanmayacağı; GA parametreleri bilgileri bir önceki menü seçeneğinde anlatılan formatta *XML* belgesi olarak kaydedilebilir.

Kod Oluştur (Generate Code): Panelde yer alan harita ve seçilmiş olan konfigürasyon ile varsa mevcut gezgin satıcılardan oluşan senaryo üzerinde, Çoklu GSP kütüphanesini kullanarak, GSP/ÇGSP problemini çözebilen Java kodu otomatik olarak oluşturulabilir. Kod oluşturmanın detaylarına Bölüm 7.3'te değinilmektedir.

7.2.3 GA Menüsü

GA menüsü Genetik Algoritmalar ile ilgili aşağıdaki seçenekleri içermektedir.

Seçilim Tipi (Selection Type): Seçilim yöntemi olarak Rulet Seçilimi (*Roulette Wheel Selection*), Sıralı Seçilim (*Rank Selection*) veya Turnuva Seçilimi (*Tournament Selection*) yöntemlerinden biri tercih edilebilir. Varsayılan olarak Sıralı Seçilim işaretlidir.

Çaprazlama Tipi (Crossover Type): Çaprazlama yöntemi olarak Tek Noktalı Çaprazlama (*Single Point Crossover*), Çift Noktalı Çaprazlama (*Two Point Crossover*), Tekdüze Çaprazlama (*Uniform Crossover*) veya Açgözlü Alt-Tur Çaprazlaması (*Greedy Subtour Crossover)* yöntemlerinden biri tercih edilebilir. Varsayılan olarak Açgözlü Alt-Tur Çaprazlaması işaretlidir.

Mutasyon Tipi (Mutation Type): Mutasyon yöntemi olarak Sıra Değiştirme (*Order Changing*) veya Ters Çevirme Mutasyonu (*Inversion Mutation*) yöntemlerinden biri tercih edilebilir. Varsayılan olarak Sıra Değiştirme işaretlidir.

Seçkincilik (Elitism): Seçkincilik uygulanıp uygulanmayacağına karar verilebilir. Varsayılan olarak seçkincilik uygulanır.

7.2.4 Eniyileme Menüsü

Eniyileme (*Optimization*) menüsünde, kullanılacak eniyileme yöntemi olarak 2-opt veya 3-opt yöntemlerinden biri tercih edilebilir. Eniyileme Yok (*No Optimization*) seçeneği seçilerek eniyileme uygulanmayabilir.

7.2.5 Çalıştır Menüsü

Çalıştır (*Run*) menüsü aşağıdaki seçenekleri içermektedir.

Çöz (Solve): Halihazırdaki harita ve satıcılar ile seçilen yöntemler ve girilen genetik parametreler doğrultusunda problemin çözümü gerçekleştirilir. Sonuçta bulunan değer ile nesil sayısı (parantez içinde) Toplam Uzaklık (*Total Distance*) kutusuna yazdırılır.

Temizle (Clear): Paneldeki şehirler ve varsa problem çözümü ile satıcılar silinerek ekran temizlenir.

7.2.6 Diğer Menüsü

Diğer (*Misc*) menüsü aşağıdaki seçenekleri içermektedir.

Türkçe/English: Dil tercihi yapılabilir. Varsayılan olarak *English* işaretlidir.

Şehir Bilgisi Göster (Show City Info): Paneldeki şehirlerin numaraları veya varsa isimleri gösterilebilir ya da saklanabilir. Varsayılan olarak şehir bilgisi gösterilir.

7.2.7 Genetik Parametreler

Aşağıdaki genetik parametrelerin değerleri üzerinde değişiklik yapılabilir.

Topluluk Büyüklüğü (Population Size): Başlangıç topluluğunun büyüklüğü pozitif tamsayı olarak belirlenir.

Çaprazlama Oranı (Crossover Rate): Seçilen iki kromozom için çaprazlama gerçekleşmesi ihtimali 0 ile 1 arasında ondalık sayı olarak belirlenir.

Mutasyon Oranı (Mutation Rate): Tüm kromozomlar için mutasyon gerçekleşmesi ihtimali 0 ile 1 arasında ondalık sayı olarak belirlenir.

Maksimum Nesil (Maximum Generation): Optimal sonuç bulunup bulunmadığına bakılmaksızın, algoritmanın çalışmasının durdurulacağı nesil numarası pozitif tamsayı olarak belirlenir.

Hedeflenen Uzaklık (Aimed Distance): Ulaşıldığında algoritmanın çalışmasının durdurulması için optimal sonuç kabul edilen uzaklık sıfır ya da pozitif tam/ondalık sayı olarak belirlenir.

7.3 Kod Oluşturma

Görsel yazılım geliştirme ortamı kullanıcıların kendi projelerinde kullanmak üzere Çoklu GSP kütüphanesini kullanan Java kodu, başka bir ifadeyle GSP/ÇGSP'yi çözen yazılım üretmelerine imkan verir. Kullanıcılar grafik arayüz yardımıyla oluşturdukları senaryoyu, tek satır kod yazmadan kendi projelerinde çalışabilecek bir program haline getirebilirler. Kod oluşturma sırasında panelde yer alan harita, mevcut konfigürasyon ve varsa satıcılar da otomatik olarak kaydedilir.

Kullanıcı menüden "Kod Oluştur" seçeneğini seçtiğinde, öncelikle panelde yer alan şehirler "*cities.tsp*" isimli bir dosyaya *TSPLIB* formatında kaydedilir. Bu dosyada şehir numaraları ile şehirlerin x,y koordinat bilgileri yer almaktadır. Örneğin 10 şehirlik rastgele bir harita için oluşturulan örnek dosyanın içeriği Çizelge 7.2'de gösterilmektedir.

Çizelge 7.2: cities.tsp dosyası içeriği

```
NAME: cities
TYPE: TSP
COMMENT: 10-city problem
DIMENSION: 10
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 664 355
2 266 123
3 276 180
4 262 302
5 726 112
6 288 26
7 32 84
8 200 183
9 278 46
10 196 390
       EOF
```

Daha sonra mevcut konfigürasyon “*ga_configuration.xml*” isimli bir dosyaya kaydedilir. Bu dosya “Konfigürasyon Yükle” seçeneği ile okunabilecek formattadır. Örnek bir konfigürasyon dosyasının içeriği Çizelge 7.3’te gösterilmektedir.

Çizelge 7.3: ga_configuration.xml dosyasının içeriği

```
<?xml version='1.0'?>
<TSPSolver xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="TSPSolver.xsd">
        <TSPSolverForm>
                <GA>
                        <Selection>Rank Selection</Selection>
                        <Crossover>Greedy Subtour Crossover</Crossover>
                        <Mutation>Order Changing</Mutation>
                        <Elitism>true</Elitism>
                </GA>
                <GAParameters>
                        <PopulationSize>60</PopulationSize>
                        <CrossoverRate>0.75</CrossoverRate>
                        <MutationRate>0.01</MutationRate>
                        <MaximumGeneration>10</MaximumGeneration>
                        <AimedDistance>0</AimedDistance>
                </GAParameters>
                <Optimization>2-opt</Optimization>
       </TSPSolverForm>
      </TSPSolver>
```

Son olarak eğer problem Çoklu GSP ise yani eklenmiş satıcı varsa, bu satıcıların başlangıç şehirlerine ait x,y koordinat bilgileri de "*salesman.sal*" isimli bir dosyada saklanır. Bu dosya kendisinden biraz önce oluşturulan "*cities*.tsp" dosyasındaki şehirlere özeldir ve başka bir şehir dizisi ile hatalı sonuç vermesi kaçınılmazdır. Daha önce verilen "*cities.tsp*" dosyasındaki şehirlere özel, örnek bir satıcı dosyası Çizelge 7.4'te gösterilmektedir.

Çizelge 7.4: salesman.sal dosyası içeriği

```
Salesman Locations X,Y for 10-city problem
IMPORTANT: Specific for cities.tsp
664 355
262 302
EOF
```

Bu dosyaların oluşturulmasının ardından diyalog penceresinden girilen dosya adıyla "*.java*" uzantılı bir Java kaynak kod dosyası oluşturulur ve seçilen konuma kaydedilir. Oluşturulan örnek bir dosyanın içeriği Çizelge 7.5'te gösterilmektedir.

Çizelge 7.5: Oluşturulan Java kodu içeriği

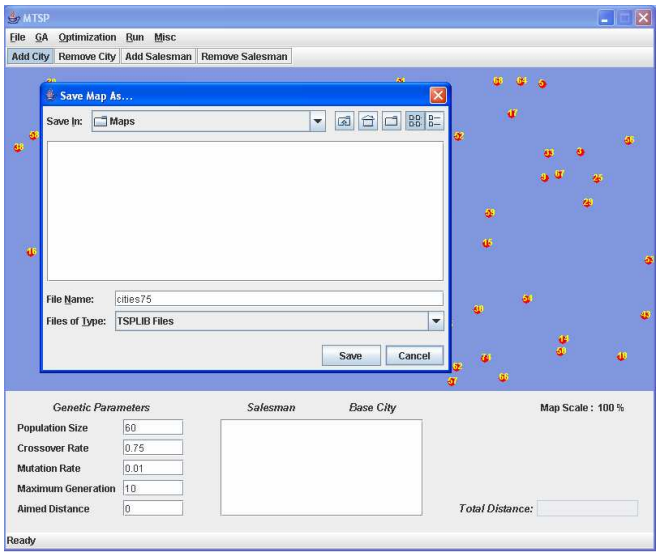
```
import java.util.Vector;
import TSP.TSPMain;

public class exampleCode {

	public static void main(String[] args) {
		TSPMain tsp = TSPMain.getInstance();
		Vector configuration = tsp.load();
		tsp.solve(configuration);
	}
}
```

Bu program, kaydedilen "*cities.tsp*", "*ga_optimization.xml*" ve "*salesman.sal*" dosyalarını kullandığı için kodun yer aldığı klasörde bu

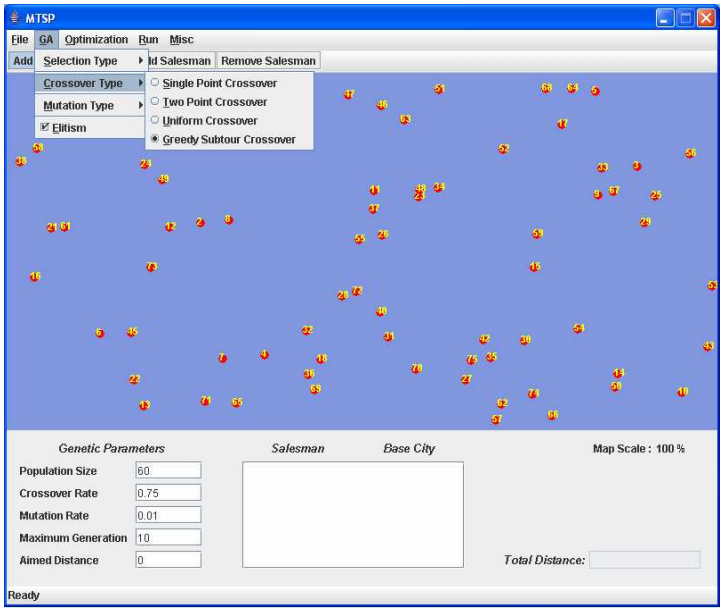
belgelerin de bulunması gerekmektedir. Program konsola çıktı olarak, problemin çözümü sonucu elde edilen uzaklık, sonuç neslinin numarası ve şehir sırası bilgilerini yazdırır. *tsp.solve* metodu ise geriye sonuç uzaklığı ve sonuç nesli ile birlikte, en iyi çözümü içeren kromozomun kendisini döndürür. Kullanıcılar kendi projelerinin ihtiyaçlarına göre bu bilgileri değerlendirebilirler ve kendi grafik arayüzlerine bağlayabilirler. Programcılar ise yazacakları bir uygulamaya, oluşturulan hazır koddan yola çıkarak başlangıç yapabilirler.

7.4 Ekran Görüntüleri

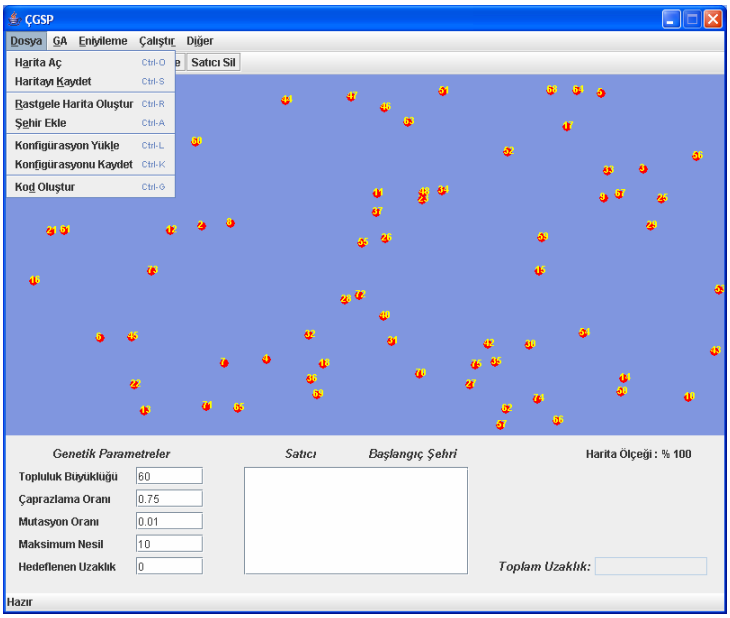
Burada, Çoklu GSP Kütüphanesi için geliştirilen Görsel Yazılım Geliştirme Ortamı'na ait bazı ekran görüntüleri sunulmaktadır (Bkz: Şekil 7.3; 7.4; 7.5; 7.6; 7.7).



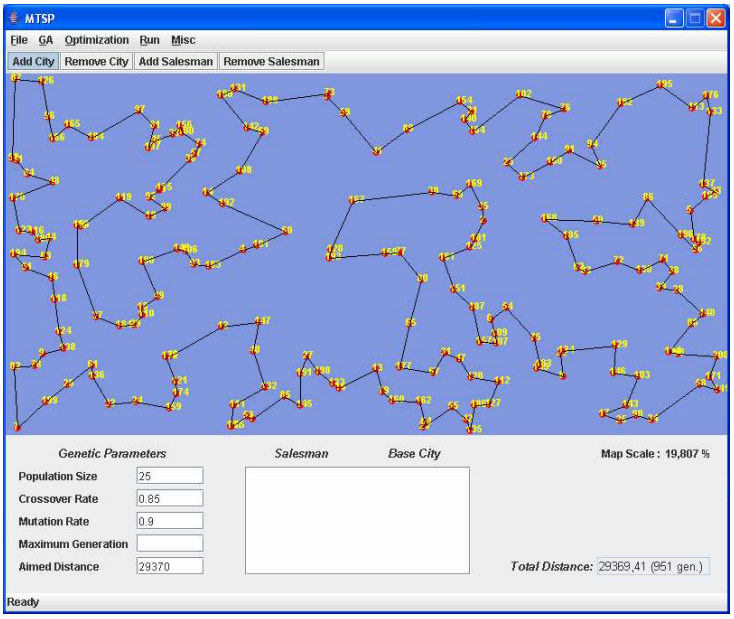
Şekil 7.3: Haritayı Kaydet diyalog penceresi



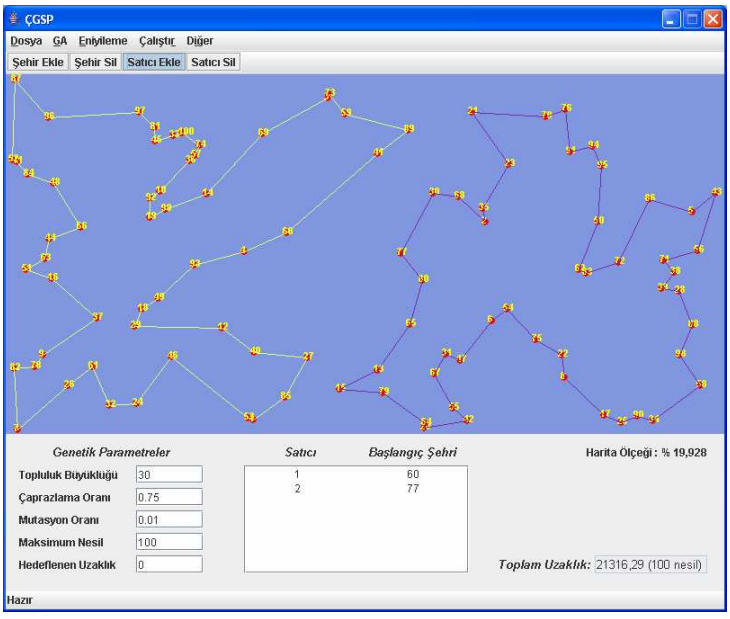
Şekil 7.4: GA Menüsünden çaprazlama tipinin değiştirilmesi



Şekil 7.5: Türkçe arayüz



Şekil 7.6: kro200 haritası üzerinde GSP için bulunan optimal tur



Şekil 7.7: kroC100 haritası üzerinde ÇGSP için iki satıcı ile bulunan çözüm

8. SONUÇ

Bu tez projesi kapsamında, Java platformu kullanılarak Çoklu Gezgin Satıcı Problemi'nin çözümü için bir eniyileme kütüphanesi tasarlanmış ve görsel yazılım geliştirme ortamı ile birlikte gerçekleştirimi yapılmıştır. Eniyileme kütüphanesinin temelini Genetik Algoritmalar ve Yerel Eniyileme (2-opt, 3-opt) yöntemleri oluşturmuştur. Bu sebeple öncelikle bu alanlarda yürütülen çalışmalar üzerine bir kaynak çalışması yapılarak, Çoklu GSP'ye nasıl uygulanabilecekleri araştırılmıştır. Birden fazla yöntemin araştırılmasının nedeni, melez yöntemlerin GSP/ÇGSP için çok daha başarılı sonuçlar vermesidir. Araştırma aşamasının ardından tasarıma geçilmiştir. Bu evrede nesneye yönelik tasarımdan faydalanılmış; çeşitli tasarım desenleri kullanılarak iyileştirmelere yer verilmiştir. Nesneye dayalı paradigmanın tercih edilmesinin nedeni, kütüphanenin okunabilirliğini arttırmak ve kütüphaneye esneklik ve genişletilebilirlik kazandırarak yazılım geliştiricilerin işlerini kolaylaştırmaktır. Gerçekleştirim sırasında ise yazılım geliştirmenin kod oluşturma ve konfigürasyon kaydetme fonksiyonları ekseninde uygulanması, arayüzün düzenlenmesi ve elbette eniyileme algoritmalarının başarılı bir şekilde uygulanması konuları üzerinde durulmuştur

Proje kapsamında yürütülen çalışmanın öneminden bahsetmek gerekirse; Java ortamında geliştirilen kütüphane, yazılım geliştiricilerin, tamamında veya bir kısmında GSP/ÇGSP'yi çözmesi gereken yazılımları oluştururken yararlanabilecekleri platform bağımsız bir kaynak teşkil edecektir. Java ortamının tercih edilmesi temelde platform bağımsızlığı

vurgusundan ileri gelmektedir. Kütüphane, istenirse arayüzüyle birlikte geldiği için başka bir yazılım gerekmeksizin problemin çözümü için planlanan senaryo (yöntemler, parametreler, şehir kümesi vb.) tercih edilebilecek ve kolayca uygulanabilecektir. Bu sırada kullanıcılar kendi veri setlerini de dışarıdan verebileceklerdir. Birden çok algoritmaya yer verildiği için, melez uygulamalara da imkan tanınmıştır. Görsel yazılım geliştirme ortamının Web tabanlı tasarlanması ise kütüphanenin yaygınlaşmasını sağlayacaktır. Projenin gelişimi sırasında alınan eleştiriler doğrultusunda bu özellik üzerine de yoğunlaşılarak çalışmalar yapılmıştır.

Çalışmanın bir diğer faydası ise eniyileme konusunda çalışmaya başlayanlar veya yeni eniyileme yöntemi geliştirmeye çalışan araştırmacılar için olacaktır. Araştırmacılar, kendi sonuçlarını bu kütüphaneyi kullanarak elde edecekleri sonuçlar ile karşılaştırabileceklerdir. Örneğin ilerde Çoklu GSP kapsamında *TSPLIB* kütüphanesindeki örnekler üzerinde yapılabilecek çalışmalar için bir test ortamı sağlanmıştır. Değişik meslek gruplarından kişiler, günlük hayatta veya işlerinde karşılarına çıkan parça toplama, parça yerleştirme ve dolaşmaya dayalı birçok problemi farklı nokta sayıları ve konumları için deneyerek en uygun sonucu alabileceklerdir. Bu şekilde kütüphanede yer alan yöntemler, baskı devrelerdeki bileşenlerin yerleştirilmesinden, turizm acentalarının seyahat planlamasına, bir fabrikada paralel bantlar üzerinde üretim yapılmasından, şehirlerdeki posta dağıtım mekanizmalarına, büyük bir şirketin depolarından yola çıkarak bayiliklerine varan tedarik ağından, büyük bir arazinin güvenlik ağının yerleşimine kadar birçok uygulamada kullanılabilecektir. Bunun için

sağlanan arayüz kullanılabileceği gibi, kütüphane kullanıcıların kendi arayüzlerine de sorunsuz bağlanabilir.

Proje kapsamında yer almayan eniyileme yöntemlerinin de programlanarak kütüphaneye dahil edilmesi ilerde yapılabilecek çalışmalardan biridir. Nesneye dayalı yaklaşımın ve kullanılan tasarım desenlerinin getirdiği bir avantajla kütüphaneye yapılması düşünülen eklemeler de çok kolay gerçekleştirilebilecektir. Örneğin Eniyileme modülüne yeni seçilim, çaprazlama, mutasyon ve eniyileme algoritmaları eklemek için ilgili sınıfların ve arayüzlerin (Bkz: 5.1.2 Eniyileme Modülü, *Selection* sınıfı ile *ICrossoverStrategy*, *IMutationStrategy* ve *IOptimizationStrategy* arayüzleri), eklenecek algoritmayı içeren sınıf ile gerçekleştirilmesi yeterli olacaktır. Bir diğer geliştirme ise Web tabanlı görsel yazılım geliştirme ortamına benzer bir görsel yazılımın *Eclipse* platformu için plug-in halinde gerçekleştirilmesi olabilir. Endüstri Mühendisliği alanında çalışan kod geliştiricilerin, Çoklu GSP'ye satıcı başına düşen yük kısıtlaması gibi kısıtlar ekleyerek problemi Araç Rotalama Problemi haline getirebilmeleri de söz konusudur. Kütüphaneye dinamik özellikler ekleyerek (çalışma anında satıcının başlangıç şehrini değiştirmek vb.) çözülebilen problemlere Dinamik Gezgin Satıcı Problemi'ni de katmak, görsel yazılım geliştirme ortamını da bu çerçevede özellikleri destekleyecek hale getirmek olasıdır.

Projenin pratik yararlarının haricinde, sağladığı saf simülasyon ortamından da bahsetmek gerekmektedir. Projede sağlanan türde web-tabanlı eniyileme araçlarının ve simülasyon ortamlarının geliştirilmesi, araştırmacılara test olanağı sunması ve değişik meslek gruplarındaki kişilerin ellerindeki problemlere çözüm getirmesinin yanında, Genetik

Algoritmalar ve Yerel Eniyileme (2-opt, 3-opt) gibi Yapay Zeka alanları ile daha çok kişinin tanışmasını da sağlamaktadır. Ayrıca sağladığı görsel destek ile öğrenme sürecini hızlandırmakta ve konunun öneminin anlaşılmasına katkıda bulunmaktadır. Bu çalışma böyle yazılımların geliştirilmesi için bir örnek, hatta altyapı teşkil etmesi bakımından da önem kazanmaktadır.

KAYNAKLAR DİZİNİ

Angus, D. and Hendtlass, T., 2005, Dynamic ant colony optimization, *Applied Intelligence*, 23(1):33-38.

Applegate, D., Bixby, R., Chvàtal V. and Cook, W., 1995, Finding Cuts in the TSP (A Preliminary Report), DIMACS, Tech. Report 95-05.

Baker, J. E., 1985, Adaptive selection methods for genetic algorithms, *Proceedings of the 1st Int. Conf. on Genetic Algorithms and Their Applications*, 101-111.

Banzhaf, W., 1990, The molecular travelling salesman, *Biol. Cybern.*, 64: 7-14.

Baraglia, R., Hidalgo, J.I. and Perego, R., 2001, A hybrid heuristic for the traveling salesman problem, *IEEE Trans. on Evolutionary Computation*, 5:613-622.

Baricelli, N. A., 1957, Symbiogenetic evolution processes realized by artificial methods, *Methodos*, 9(35-36):143-182.

Bektaş, T., 2006, The multiple traveling salesman problem: an overview of formulations and solution procedures, *Omega – Int. Journal of Management Science*, 34(3):209-219.

Bellmore, M. and Malone, J.C., 1972, Pathology of traveling-salesman subtour-elimination algorithms, *Oper. Res.*, 19:278-307.

Bethke, A. D., 1980, Genetic Algorithms as Function Optimizers, PhD Thesis, University of Michigan, Ann Arbor.

Box, G. E. P., 1957, Evolutionary operation: A method for increasing industrial productivity, *Journal of the Royal Statistical Society*, 6(2): 81–101.

Bremermann, H. J., 1962, Optimization through evolution and recombination, In: Self-Organizing Systems, Spartan Books, Washington, D.C., 93-106.

Carter, A., 2003, Design and Application of Genetic Algorithms for the Multiple Traveling Salesperson Assignment Problem, Doctoral Dissertation, Virginia Polytechnic Institute and State Uni., Virginia

KAYNAKLAR DİZİNİ (Devam)

Cevre, U., Özkan, B. ve Uğur, A., 2007, Gezgin satıcı probleminin genetik algoritmalarla eniyilemesi ve etkileşimli olarak Internet üzerinde görselleştirilmesi, *INET-TR 2007*, Ankara.

Croes, G., 1958, A method for solving traveling salesman problems, *Oper. Res.*, 6.

Dang, J., Wang, Y. and Zhao, S., 2007, Study on a novel genetic algorithm for the combinatorial optimization problem, *Int. Conf. on Control, Autom. and Syst. 2007*.

Dantzig, G.B., Fulkerson D. R. and Johnson, S. M., 1954, Solution of a large-scale traveling-salesman problem, *Oper. Res.*, 2:393-410.

Davis, L., 1985, Applying adaptive algorithms to epistatic domains, *Proceedings of the International Joint Conference on Artificial Intelligence*, 162-164.

Davis, L., 1989, Adapting operator probabilities in genetic algorithms, *Proceedings of the Third International Conference on Genetic Algorithms*, 61-69.

De Jong, K. A., 1975, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, PhD Thesis, University of Michigan, Ann Arbor.

Dofactory, 2007, http://www.dofactory.com/Patterns/Patterns.aspx

Dorigo, M., Maniezzo, V. and Colorni, A., 1991, Positive Feedback as a Search Strategy, Technical Report No. 91-016, Politecnico di Milano, Italy.

Dorigo, M., Maniezzo, V. and Colorni, A., 1996, The ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems., Man, and Cybernetics - Part B*, 26(1):29-41.

Eshelman, L. J., Caruana, R. A. and Schaffer, J. D., 1989, Biases in the landscape, *Proceedings of the 3rd International Conf. on Genetic Algorithms*, 10-19.

Falkenauer, E., 1998, Genetic Algorithms and Grouping Problems, John Wiley & Sons, New York, 238p.

KAYNAKLAR DİZİNİ (Devam)

Fiechter, L., 1994, A parallel tabu search algorithm for large traveling salesman problems, *Discrete Applied Math. and Comb. Oper. Res. and Comp. Sc.*, 51:243-267.

Fogarty, T. C., 1989, Varying the probability of mutation in genetic algorithms, *Proceedings of Third International Genetic Algorithms*, 104-109.

Fogel, D. B., 1993, Empirical estimation of the computation required to discover approximate solutions to the travelling salesman problem using evolutionary programming, *Proc. of 2nd Annual Conf. on Evolutionary Programming*, 56-61.

Fogel, L. J., Owens, A. J., and Walsh, M. J., 1966, Artificial Intelligence through Simulated Evolution, John Wiley & Sons, New York, 170p.

Gambardella, L. M. and Dorigo, M., 1996, Solving symmetric and asymmetric TSPs by ant colonies, *Proc. of IEEE Int. Conf. on Evol. Computation*, 622-627.

Garey, M.R. and Johnson, D.S, 1979, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 340p.

Goldberg, D.E. and Lingle, J.R., 1985, Alleles, loci and the TSP, *Proceedings of the 1st International Conf. on Genetic Algorithms and Their Applications*, 154-159.

Goldberg, D.E., 1989, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, Reading, MA, 432p.

Goldberg, D. E., 1990, A note on Boltzmann tournament selection for genetic algorithms and population−oriented simulated annealing, *Complex Syst*., 4: 445-460.

Goldberg, D. E., and Deb, K, 1991, A comparitive analysis of selection schemes used in genetic algorithms, *Foundations of Genetic Algorithms*, 69-91.

Gorges-Schleuter, M., 1989, ASPARAGOS An asynchronous parallel genetic optimization strategy, *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, 422-427.

KAYNAKLAR DİZİNİ (Devam)

Grefenstette, J. J., 1986, Optimization of control parameters for genetic algorithms, *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128.

Grötchel, M. and Holland, O., 1991, Solution of large scale symmetric travelling salesman, *Math. Programming*, 51:141-202.

Helsgaun, K., 2006, An effective implementation of k-opt moves for the Lin-Kernighan TSP heuristic, *Writings on Computer Science*, Roskilde University.

Hillis, W. D., 1990, Co-evolving parasites improve simulated evolution as an optimization procedure, *Physica D*, 42:228–234.

Holland, J. H., 1975, Adaptation in Natural and Artificial Systems, University of Michigan Press, 228p.

Jin, H. D., Leung, K. S., Wong, M. L. and Xu Z. B., 2003, An efficient self-organizing map designed by genetic algorithms for the traveling salesman problem, *IEEE Trans. Syst., Man. Cyber. - Part B*, 33(6):877–887.

Johnson, D. S. and McGeoch, L. A., 1997, The traveling salesman problem: A case study in local optimization, Local Search in Combinatorial Optimization, John Wiley & Sons, New Jersey, 215–310.

Junjie, P. and Dingwei, W., 2006, An ant colony optimization algorithm for multiple travelling salesman problem, *ICICIC'06*, 210-213.

Kara, I. and Bektaş, T., 2006, Integer linear programming formulations of multiple salesman problems and its variations, *European Journal of Operational Research*, 174(3):1449-1458.

Karaboğa, D., 2004, Yapay Zeka Optimizasyon Algoritmaları, Nobel Yayın Dağıtım, İstanbul, 199s.

Karp, R. M., 1972, Reducibility among combinatorial problems, Complexity of Computer Computations, Plenum Press, New York, 85-103.

Kitano, H., 1990, Designing neural networks using genetic algorithms with graph generation system, *Complex Systems*, 4:461-476.

KAYNAKLAR DİZİNİ (Devam)

Koza, J. R., 1992, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 840p.

Laarhoven, P.V. and Aarts, E.H.L., 1987, Simulated Annealing: Theory and Applications, Springer, 186p.

Laporte, G., 1992, The traveling salesman problem: An overview of exact and approximate algorithms, *Eur. J. Oper. Res.*, 59:231-247.

Lin, S. and Kernighan, B. W., 1973, An effective heuristic algorithm for the traveling-salesman problem, *Oper. Res.*, 21:498-516.

Malik, W., Rathinam, S. and Darbha, S., 2007, An approximation algorithm for a symmetric generalized multiple depot, multiple traveling salesman problem, *Oper. Res. Letters*, 35(6):747-753.

Melamed, I. I., Sergeev, S. I. and Sigal, I., 1989, The traveling salesman problem: Approximate algorithms, *Avtomat. Telemekh.*, 11:3-26.

Michalewicz, Z., 1992, Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, Berlin, Germany, 340p.

Michalewicz, **Z. and Fogel**, **D. B.**, 2004, How to Solve It: Modern Heuristics, Springer, 554 p.

Mitchell, M., 1999, An Introduction to Genetic Algorithms, MIT Press, 205p.

Mitrovic-Minic, S. and Krishnamurti, R., 2006, The multiple TSP with time windows: vehicle bounds based on precedence graphs, *Operations Research Letters*, 34(1):111-120.

Montana, D. J. and Davis, L. D., 1989, Training feedforward networks using genetic algorithms, *Proc. of the Int. Joint Conf. on Artificial Intelligence*, 762-767.

Mühlenbein, H., 1992, How genetic algorithms really work: 1. Mutation and hillclimbing, Parallel Problem Solving from Nature 2, North−Holland, 618p.

Obitko, 1998, http://cs.felk.cvut.cz/~xobitko/ga/index.html

KAYNAKLAR DİZİNİ (Devam)

Oliver, I. M., Smith, D. J. and Holland, J. R. C., 1987, A study of permutation crossover operators on the TSP, *Proceeding of ICGA2*, 224-230.

Özkan, B., Cevre, U. ve Uğur, A., 2008, Melez bir eniyileme yöntemi ile rota planlama, *Akademik Bilişim 2008*, Çanakkale.

Padberg, M. W. and Rinaldi, G., 1991, A branch-and-cut algorithm for the resolution of symmetric traveling salesman problems, *SIAM Review*, 33:60-100.

Park, Y.B., 2001, A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines, *Int. Journal of Productions Econ.*, 73(2):175-188.

Rechenberg, I., 1973, Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, Frommann-Holzboog, Stuttgart, 170p.

Reeves, C. R., 1995, Genetic algorithms and combinatorial optimization, Applications of Modern Heuristic Techniques, Alfred Waller, UK, 111-126.

Rosenkrantz, D. E. Stearns, R. E. and Lewis II, P. M., 1977, An analysis of several heuristics for the traveling salesman problem, *SIAM J. Comput.*, 6:563-581.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J. and Das, R., 1989, A study of control parameters affecting online performance of genetic algorithms for function optimization, *Proc. of the 3rd International Conf. on Genetic Algorithms*, 51-60.

Schwefel, H. -P., 1975, Evolutionsstrategie und Numerische Optimierung, PhD Thesis, Technische Universität Berlin, Berlin.

Sengoku, H. and Yoshihara, I., 1998, A Fast TSP Solution using Genetic Algorithm, *Proc. of the 3rd AROB*, 283-288.

Spears, W. M. and De Jong, K. A., 1991, On the virtues of parameterized uniform crossover, *Proceedings of the 4th Int. Conf. on Genetic Algorithms*, 230-236.

KAYNAKLAR DİZİNİ (Devam)

Spears, W. M., 1993, Crossover or mutation?, *Foundations of Genetic Alg. 2*, 221-237.

Syswerda, G., 1991, Schedule optimization using genetic algorithms, Handbook of Genetic Algorithms, Van Nostrand Reinhold, New York, 350-372.

Şen, Z., 2004, Genetik Algoritmalar ve En İyileme Yöntemleri, Su, İstanbul,142s.

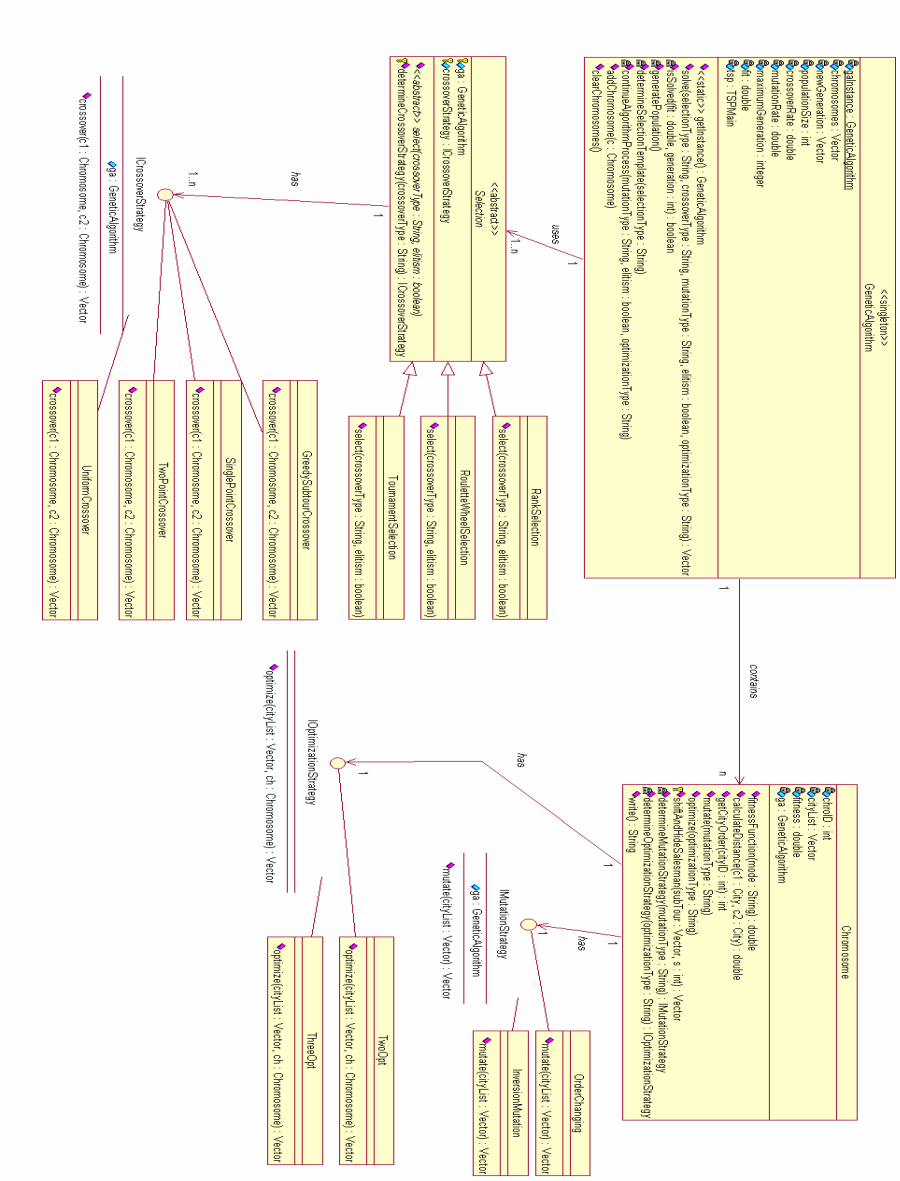
Tang, L., Liu, J., Rong, A. and Yang, Z., 2000. A multiple traveling salesman problem model for hot rolling schedule in Shanghai Baoshan Iron & Steel Complex, *European Journal of Operational Research*, 124(2):267-282.

Uğur, A., 2008, Path planning on a cuboid using genetic algorithms, *Information Sciences*, 178(16):3275-3287.

Whitley D., Hanson T., 1989, Optimizing neural networks using faster, more accurate genetic search, *Proc. of the 3rd Int. Conf. on Genetic Alg.*, 391-396.

EKLER

Ek 1 Eniyileme Modülünün Sınıf Diyagramı



Ek 2 TSPSolver XML Şeması

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:include schemaLocation="GAOptDataDictionary.xsd"/>
   <xsd:element name="TSPSolverForm">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="GA" />
            <xsd:element ref="GAParameters" />
            <xsd:element ref="Optimization" />
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Ek 3 GAOptDataDictionary XML Şeması

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Selection" type="xs:string" />
  <xs:element name="Crossover" type="xs:string" />
  <xs:element name="Mutation" type="xs:string" />
  <xs:element name="Elitism" type="xs:boolean" />
  <xs:element name="GA">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Selection" />
        <xs:element ref="Crossover" />
        <xs:element ref="Mutation" />
        <xs:element ref="Elitism" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="PopulationSize" type="xs:int" />
  <xs:element name="CrossoverRate" type="xs:double" />
  <xs:element name="MutationRate" type="xs:double" />
  <xs:element name="MaximumGeneration" type="xs:int" />
  <xs:element name="AimedDistance" type="xs:double" />
  <xs:element name="GAParameters">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="PopulationSize" />
        <xs:element ref="CrossoverRate" />
        <xs:element ref="MutationRate" />
        <xs:element ref="MaximumGeneration" />
        <xs:element ref="AimedDistance" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Optimization" type="xs:string" />
</xs:schema>
```

ÖZGEÇMİŞ

Utku Cevre 1984'te İzmir'de doğmuştur. TC vatandaşıdır. Ortaöğrenimini İzmir Güzelbahçe 60. Yıl Anadolu Lisesi'nde tamamladıktan sonra lisans eğitimi için 2002 yılında Ege Üniversitesi Bilgisayar Mühendisliği Bölümü'ne girmiştir. 2006'dan beri aynı bölümde lisansüstü eğitimine devam etmektedir. Profesyonel ilgi alanları arasında, Yapay Zeka, Genetik Algoritmalar, Bilgisayar Grafikleri ve Yazılım Geliştirme sayılabilir.

2005'te IBM Türk'ün açtığı sınavı kazanarak kurumda yaz stajı yapmaya hak kazanmıştır. 2006'da TÜBİTAK BİDEB Yurtiçi Lisansüstü Bursu'nu (2228) almaya hak kazanmıştır. Çok iyi derecede İngilizce (2006-KPDS 98), orta derecenin üzerinde Fransızca ve orta derecede Almanca bilmektedir.