T.C.

BEYKENT ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS PROGRAMI

TELEFON BANKACILIĞINDA İNTERAKTİF SESLİ YANIT SİSTEMLERİ VE BECERİ BAZLI YÖNLENDİRME ALGORİTMASI

(Yüksek Lisans Tezi)

Tezi Hazırlayan: **Abdulkerim POŞUL**

İstanbul, 2011

T.C.

BEYKENT ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS PROGRAMI

TELEFON BANKACILIĞINDA İNTERAKTİF SESLİ YANIT SİSTEMLERİ VE BECERİ BAZLI YÖNLENDİRME ALGORİTMASI

(Yüksek Lisans Tezi)

Tezi Hazırlayan:

Abdulkerim POŞUL

Öğrenci No:

090820002

Danışman:

Yrd.Doç.Dr.Gökhan SİLAHTAROĞLU

İstanbul, 2011

YEMİN METNİ

Yüksek lisans projesi / tezi olarak sunduğum "Telefon bankacılığında interaktif sesli yanıt sistemleri ve Beceri bazlı yönlendirme algoritması" başlıklı çalışmamın, bilimsel ahlak ve geleneklere uygun şekilde tarafımdan yazıldığını, yararlandığım eserlerin tamamının kaynaklarda gösterildiğini ve çalışmamın içinde kullanıldıkları her yerde bunlara atıf yapıldığını belirtir ve bunu onurumla doğrularım. 19/12/2011

Abdulkerim POŞUL

T.C.
BEYKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ SAVUNMA SINAVI SONUÇ TUTANAĞI

Beykent Üniversitesi
Fen Bilimleri Enstitüsü Müdürlüğü'ne,

Aşağıda tez adı belirtilen yüksek lisans öğrencisi 090820002 no'lu Abdülkerim POŞUL'un ..19../...12./..2011. tarihinde yapılan tez savunma sınavı[1] sonucunda ...60.....dakika süreyle sunduğu ve savunduğu tezi hakkında[2] oybirliğiyle/~~oyçokluğuyla,~~ Kabul/~~Red/Düzeltme(.....ay içinde~~) kararı verilmiştir.

Bilgilerinize saygılarımızla arz ederiz.

Anabilim Dalı : .BİLGİSAYAR MÜHENDİSLİĞİ .....
Programı : BİLGİSAYAR MÜHENDİSLİĞİ.....
Tez Başlığı[3] : TELEFON BANKACILIĞINDA İNTERAKTİF SESLİ YANIT SİSTEMLERİ ve BECERİ BAZLI YÖNLENDİRME ALGORİTMASI

Tez Sınav Jürisi	**Öğretim Üyesi**	**İmza**
Danışman	: Yrd. Doç. Dr. Gökhan SİLAHTAROĞLU	.....
Üye	: Yrd. Doç. Dr.Ediz ŞAYKOL	.....
Üye	: Yrd. Doç. Dr. Ebru ABDULLAHOĞLU	.....

---

[1] Jüri üyeleri söz konusu tezin kendilerine teslim edildiği tarihten itibaren en geç bir ay içinde toplanarak öğrenciyi tez savunma sınavına alır. Belirlenen günde yapılamayan jüri toplantısı, katılanların hazırladığı bir tutanakla enstitü yönetimine bildirilir. Bu durumda jüri en geç onbeş gün içinde toplanarak adayı tez savunma sınavına alır. Tez savunma sınav süresi en az 45 dakikadır. Yüksek lisans tez savunma sınavı, tez çalışmasının sunulması ve bunu izleyen soru-yanıt bölümlerinden oluşur ve dinleyiciye açıktır. (Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-3)

[2] Tez sınavının tamamlanmasından sonra jüri, tez hakkında "kabul", "düzeltme" veya "red" kararı verir. Jüri başkanı, jüri üyelerince imzalanmış sınav tutanağını, tez sınavını izleyen üç gün içinde ilgili enstitü yönetimine teslim eder. Tezi başarısız bulunan öğrencinin Enstitü ile ilişiği kesilir. Tezi hakkında düzeltme kararı verilen öğrenci en geç üç ay içinde gerekli düzeltmeleri yaparak ve yönetmelikte belirtilen usullere uygun olarak tezini aynı jüri önünde yeniden savunur. Bu savunma sınavında da tezi kabul edilmeyen öğrencinin enstitü ile ilişiği kesilir. (Beykent Lisansüstü eğitim ve Öğretim Yönetmeliği-Madde30-4)

[3] İleride doğabilecek aksaklıkların engellenmesi için tezin başlığının yazılması gerekmektedir.

TELEFON BANKACILIĞINDA İNTERAKTİF SESLİ YANIT SİSTEMLERİ VE BECERİ BAZLI YÖNLENDİRME ALGORİTMASI,

Tezi Hazırlayan: Abdulkerim POŞUL

ÖZET

Bu tez çalışmasındaki amaç banka müşterilerinin, çağrı merkezini kullanırken beceri bazlı yönlendirme algoritması ile efektif hizmet almalarını sağlamak ve müşteri memnuniyetini kazanmaktır.

Beceri bazlı yönlendirme, yönteminde müşteri temsilcileri becerilerine göre ve yapmaya yetkili oldukları işlere (muhasebe, hisse senetleri, halka arz, satış, destek vb) göre gruplara ayrılır. Arayan müşterinin yaptığı tercihe göre çağrı uygun becerilere sahip müşteri temsilcilerinin kuyruğuna yönlendirilir.

Ayrıca müşterinin veri tabanı kayıtlarında bulunan belli bilgilerine göre çağrı yönlendirmesi yapılır. Mesela arayan müşterinin hesabında belli bir tutardan fazla borç varsa bu çağrıyı önce denetim departmanına veya muhasebeye aktar veya müşterinin kredili işlem sözleşmesinin tarihi geçmişse çağrıyı kredili işlem departmanına aktar gibi çağrı aktarım kuralları tanımlanabilir.

Temel hedef müşteri temsilcine bağlanmak isteyen müşterilerin, hem aktarıldıkları kuyruk bazında hem de müşteri segmentleri bazında önceliklendirilerek aktarılmasını sağlamaktır.

.**Anahtar Kelimeler:** Telefon Bankacılığı, İnteraktif Sesli Yanıt Sistemleri, Beceri bazlı yönlendirme

TELEPHONE BANKING INTERACTIVE VOICE RESPONSE SYSTEMS AND SKILLS BASED ROUTING ALGORITHM

Presented by: Abdulkerim POSUL

ABSTRACT

The aim of this thesis, is, using the call center with skills-based routing algorithm to enable bank customers to receive effective service and customer satisfaction.

Skill-based routing, the method according to ability and customer representatives which are authorized to carry out tasks (accounting, securities, public offerings, sales, support, etc.) are divided into groups according to customer's preference, caller calls are routed to the queue of the customer representative with appropriate skills.

In addition, routing and queueing is also conducted according to the information about the customer provided by the database. Some of the information used for this process is debt exceding a certain amount,credit card limit and frequency of telephone banking usage

The algorithm propose, basically, aims to prioritize the customers in accordance with the customer segmentation and the queue itself.

Key Words: Telephone Banking, IVR (Interactive Voice Response),Skill Base Routin

İÇİNDEKİLER

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

ŞEKİLLER LİSTESİ

Sayfa No

KISALTMALAR

ÜİY: Üye İş Yeri

MVT: Müşteri Veri Tabanı

TBŞ: Telefon Bankacılığı Şifresi

KKŞ: Kredi Kartı Şifresi

IVR: Sesli Yanıt Sistemi

MT: Müşteri Temsilcisi

AGENT: Müşteri Temsilcisi

SBR: Skill Base Routing (Kuyruk Bekleme/Yönlendirme)

CIF: Müşteri Numarası

Bkz: Bakınız

TCKN: Türkiye Cumhuriyeti Kimlik Numarası

INTBANK: İnternet Bankacılığı

1.GİRİŞ

Son yıllarda bilgi teknolojilerindeki gelişmelerin yaşamımızdaki pek çok alanı etkilemesi kaçınılmazdır. Kuşkusuz bu gelişmelerden en fazla etkilenen sektör, bankacılık sektörü olmuştur. Bankalar, bilgi teknolojilerindeki gelişmelere paralel olarak yeniden yapılanmakta, maliyetleri azaltıcı önlemler almakta ve böylece ellerindeki teknolojik alt yapıyı en iyi şekilde kullanarak müşteri tabanını genişletme ve pazarlama faaliyetlerini arttırmaya yönelik projeler oluşturmaya çalışmaktadırlar.

Bankaların teknolojik gelişmelere ayak uydurması, hizmet maliyetlerinde düşüş sağlamanın yanı sıra çok çeşitli ürünlerin müşteriye en hızlı ve etkin biçimde ulaşmasını sağlayacaktır.

Son yarım yüzyıldır elektronik ticaretin gelişmesiyle bankacılık sektöründe, toplumun gereksinimleri doğrultusunda önemli değişiklikler yaşanmaya başlanmıştır. Özellikle küreselleşmeyle birlikte bankalar, yoğun rekabet ortamından etkilenmiş, müşterilerinin gereksinim ve istekleri doğrultusunda daha nitelikli hizmetler sunabilme yarışı içine girmişlerdir. Bu durum, hizmet sunumunda klasik şube bankacılığının yanında alternatif dağıtım kanallarının kullanımını da hızlandırmıştır.

Türk bankacılık sektörünün alternatif dağıtım kanalları uygulamaları açısından gelişiminde özellikle 1994 yılı sonrasında müşteriye yönelik olarak yapılanmaya giden ve müşteri bölümlendirmesini ön plana çıkaran bir profil örülmektedir.

Satış ekiplerinin ve şubelerin üzerinde bulunan işlemsel yükü kaldırmayı ve işlem maliyetlerini düşürmeyi hedefleyen bankalar öncelikli olarak Internet, telefon ve bankamatik (ATM) gibi alternatif kanalları dağınık yapıdan merkezi yapıya çevirerek aktif bir şekilde kullanmaya başlamışlardır.

Çoklu kanal stratejisi olarak ortaya çıkan bu uygulama tamamen şube ağırlıklı satış ve hizmet üzerine odaklanan bankacılık sektörünü, hizmet standartları, tek bir veri tabanı üzerinden merkezi olarak işleyen bir kanal yönetimi stratejisi oluşturmaya yöneltmektedir.

Telefon Bankacılığı, müşterinin kendisine verilen şifre ile kendi hesap bilgilerine telefon üzerinden 444 *** arayarak sesli yanıt sistemi veya Müşteri

temsilcisi aracılığıyla, telefon tuşlarını kullanarak kendisini yönlendiren sesli uyarılar ile Talep ettiği işlemi yerine getirmesi olayıdır.

Bu tez çalışmasında Bankacılık sektörü içerisinde yer alan çağrı merkezileri hedef olmak üzere diğer çağrı merkezlerinden yararlanan müşterilerin belirlenecek olan kriterlere göre çağrı önceliklendirmesi ile çağrı kuyruğunda daha az süre bekletilmesi ve çağrı merkezi hizmetlerinden daha efektif bir şekilde faydalanmaları amaçlanarak beceri bazlı yönlendirme konusu ele alınmıştır. Beceri bazlı yönlendirme ile sesli yanıt sisteminden müşteri temsilcine bağlanmak isteyen müşterilerin, hem aktarıldıkları kuyruk bazında hem de müşteri segmentleri bazında önceliklendirilerek aktarılması ele alınmıştır. Bu tezde günümüzde kullanılan Algoritmalardan yararlanılmış olmakla birlikte yeni bir yaklaşımda getirilmesi hedeflenmiştir. Algoritma, matematikte ve bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler (adımlar) kümesidir.

Algoritmalar bilgisayarlar tarafından işletilebilirler. Algoritma kelimesi, Özbekistan'ın Harezm, bugünkü Türkmenistan'ın Khiva kentinde doğmuş Ebu Abdullah Muhammed bin Musa el Harezmi isimli Türk matematikçinin adından gelir. Batılılar, el Harezmi (Al-Khwarizmi)(Latincede Algoritmi) sözcüğünü telaffuz edemedikleri için terim bu şekilde kalmıştır. [1]

Algoritma sözcüğü Ebu Abdullah Muhammed İbn Musa el Harezmi adındaki Türkistanlı âlimden kaynaklanır. Bu alim 9. yüzyılda cebir alanindaki algoritmik çalışmalarını kitaba dökerek matematiğe çok büyük bir katkı sağlamıştır. "Hisab el-cebir ve el-mukabala" kitabı dünyanın ilk cebir kitabı ve aynı zamanda ilk algoritma koleksiyonunu oluşturur. Latince çevirisi Avrupa'da çok ilgi görür - alimin ismini telaffuz edemeyen Avrupalılar "algorizm" sözcüğünü "Arap sayıları kullanarak aritmetik problemler çözme kuralları" manasında kullanırlar. Bu sözcük daha sonra "algoritma"ya dönüşür ve genel kapsamda kullanılır. [1]

Bir problemin ideal çözümüne giden yola algoritma denir. Yazılacak programın dili değil de, algoritması en önemli kısmıdır. Programı çalıştıracak algoritmayı en iyi şekilde çözümledikten sonra, kullanılacak dilin yapısına göre kodlama aşamasına geçilir. [2]

Programlama ihtiyacı duyulan her konuda, çözümleri koda aktarırken en temel algoritmalar kullanılır. Örneğin bir listenin sıralanması işleminde, sıralama algoritması kullanılmalıdır. Veya bir liste içinde en yüksek sayısal değeri bulmak için programcı en büyük elemanı bulma algoritmasını kullanmalıdır. [2]

2.KUYRUK ve KUYRUK ALGORİTMALARI

2.1 Algoritma Tarihi ve Tanımı

Algoritma, matematikte ve bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler (adımlar) kümesidir. [1]

Algoritmalar bilgisayarlar tarafından işletilebilirler. Algoritma kelimesi, Özbekistan'ın Harezm, bugünkü Türkmenistan'ın Khiva kentinde doğmuş Ebu Abdullah Muhammed bin Musa el Harezmi isimli Türk matematikçinin adından gelir. Batılılar, el Harezmi (Al-Khwarizmi)(Latincede Algoritmi) sözcüğünü telaffuz edemedikleri için terim bu şekilde kalmıştır. [1]

Algoritma sözcüğü Ebu Abdullah Muhammed İbn Musa el Harezmi adındaki Türkistanlı âlimden kaynaklanır. Bu alim 9. yüzyılda cebir alanindaki algoritmik çalışmalarını kitaba dökerek matematiğe çok büyük bir katkı sağlamıştır. "Hisab el-cebir ve el-mukabala" kitabı dünyanın ilk cebir kitabı ve aynı zamanda ilk algoritma koleksiyonunu oluşturur. Latince çevirisi Avrupa'da çok ilgi görür - alimin ismini telaffuz edemeyen Avrupalılar "algorizm" sözcüğünü "Arap sayıları kullanarak aritmetik problemler çözme kuralları" manasında kullanırlar. Bu sözcük daha sonra "algoritma"ya dönüşür ve genel kapsamda kullanılır. [2]

Bir problemin ideal çözümüne giden yola algoritma denir. Yazılacak programın dili değil de, algoritması en önemli kısmıdır. Programı çalıştıracak algoritmayı en iyi şekilde çözümledikten sonra, kullanılacak dilin yapısına göre kodlama aşamasına geçilir. [1]

Programlama ihtiyacı duyulan her konuda, çözümleri koda aktarırken en temel algoritmalar kullanılır. Örneğin bir listenin sıralanması işleminde, sıralama algoritması kullanılmalıdır. Veya bir liste içinde en yüksek sayısal değeri bulmak için programcı en büyük elemanı bulma algoritmasını kullanmalıdır. [2]

Algoritma belirli bir görevi yerine getiren sonlu sayıdaki işlemler dizisidir.

Her algoritma aşağıdaki kriterleri sağlamalıdır:

Girdi: Sıfır veya daha fazla değer dışarıdan verilmeli.

Çıktı: En azından bir değer üretilmeli.

Açıklık: Her işlem (komut) açık olmalı ve farklı anlamlar içermemeli.

Sonluluk: Her türlü olasılık için algoritma sonlu adımda bitmeli.

Etkinlik: Her komut kişinin kalem ve kâğıt ile yürütebileceği kadar basit olmalıdır.

Her program için sonluluk özelliği geçerli değildir. Örneğin işletim sistemleri sonsuza dek çalışan programlara örnektir. [2]

Bir dizi içerisindeki en büyük sayıyı bulmak için izlenecek yol şu şekilde olabilir:

Örnek: 1'den 100'e kadar olan sayıların toplamını veren algoritma.

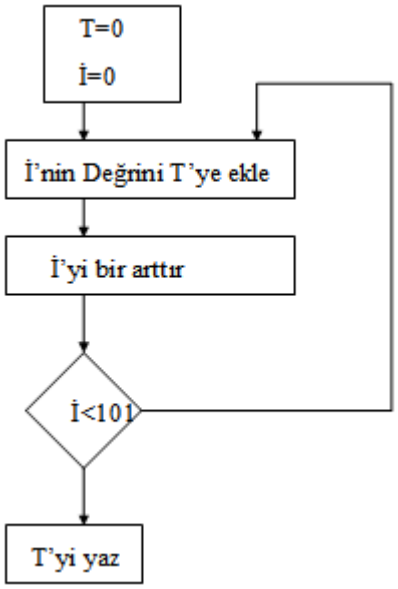
1. Toplam T, sayılar da i diye çağırılsın.
2. Başlangıçta T'nin değeri 0 ve i'nin değeri 1 olsun.
3. i'nin değerini T'ye ekle.
4. i'nin değerini 1 arttır.
5. Eğer i'nin değeri 100'den büyük değil ise 3. adıma git.
6. T'nin değerini yaz.

Algoritmaların yazım dili değişik olabilir. Günlük konuşma diline yakın bir dil olabileceği gibi simgelere dayalı da olabilir. Akış şeması eskiden beri kullanıla gelen bir yapıdır. Algoritmayı yazarken farklı anlamlar taşıyan değişik şekildeki kutulardan yararlanılır. Yine aynı amaç için kullanılan programlama diline yakın bir (sözde kod = pseudo code) dil, bu kendimize özgü de olabilir, kullanılabilir. [2]

Aynı algoritmayı aşağıdaki gibi yazabilir:

1. T=0 ve i=0
2. i'nin değerini T'ye ekle.
3. i'yi 1 arttır.
4. i<101 ise 2.adıma git.
5. T'nin değerini yaz.

Algoritmayı bir de akış şeması ile gerçekleyelim.



Şekil-1 Algoritma akış şeması

2.2 Algoritma Türleri

Önemli algoritma türleri:

- Arama algoritmaları
- Bellek yönetimi algoritmaları
- Bilgisayar grafiği algoritmaları
- Birleşimsel algoritmalar
- Çizge algoritmaları
- Evrimsel algoritmalar
- Genetik algoritmalar
- Kripto algoritmaları veya kripto grafik algoritmalar
- Kök bulma algoritmaları
- Optimizasyon algoritmaları
- Sıralama algoritmaları
- Veri sıkıştırma algoritmaları

2.2.1 Arama Algoritmaları

Bilgisayar bilimlerinde, çeşitli veri yapılarının (data structures) üzerinde bir bilginin aranması sırasına kullanılan algoritmaların genel ismidir. Örneğin bir dosyada bir kelimenin aranması, bir ağaç yapısında (tree) bir düğümün (node) aranması veya bir dizi (array) üzerinde bir verinin aranması gibi durumlar bu algoritmaların çalışma alanlarına girer. [3]

Yapısal olarak arama algoritmalarını iki grupta toplamak mümkündür.

- Uninformed Search (Bilmeden arama)
- Informed Search (Bilerek arama)

Arama işleminin bilmeyerek yapılması demek, arama algoritmasının, probleme özgü kolaylıkları barındırmaması demektir. Yani her durumda aynı şekilde çalışan algoritmalara uninformed search (bilmeden arama) ismi verilir. Bu aramaların bazıları şunlardır: [3]

Listeler (diziler (array)) üzerinde çalışan arama algoritmaları:

- Doğrusal Arama (Linear Search)
- İkili arama (binary search)
- Interpolasyon Araması (Ara değer araması, Interpolation Search)
- Şekiller (graflar (Graphs)) üzerinde çalışan algoritmalar
- Sabit Maliyetli Arama (Uniform Cost Search)
- Floyd Warshall algoritması
- Prim’s Algoritması
- Kruskal Algoritması
- Dijkstra Algoritması
- Bellman Ford Algoritması
- İkili arama ağacı (Binary Search Tree)
- Prüfer dizilimi
- Ağaçlarda Sığ öncelikli arama (breadth first search)
- Şekillerde (Graph) sığ öncelikli arama (Breadth First Search, BFS)
- Derin öncelikli arama (depth first search)
- Derin Limitli Arama (Depth Limited Search) Algoritması

- Yinelemeli Derinleşen Derin Öncelikli Arama Algoritması (Iterative Deepining Depth First Search, IDDFS)
- Patricia Ağaçları
- Trie Ağaçları (metin ağaçları, trie trees)
- B-ağaçları (B-Tree)
- Metin arama algoritmaları (bir yazı içerisinde belirli bir dizgiyi (string) arayan algoritmalar)
- Knuth-Morris Prat arama algoritması
- Boyer-Moore Arama algoritması
- Kaba Kuvvet Metin Arama Algoritması (Brute Force Text Search, Linear Text Search)
- DFA Metin Arama Algoritması

Arama işleminin bilerek yapılması ise, algoritmanın probleme ait bazı özellikleri bünyesinde barındırması ve dolayısıyla arama algoritmasının problem bazlı değişiklik göstermesi demektir. Bu algoritmaların bazıları a aşağıda listelenmiştir:

- Minimax Ağaçları
- Simulated Annealing (Benzetimli Tavlama) algoritması
- Tepe Tırmanma Algoritması (Hill Climbing Algorithm)
- Arı sürüsü arama algoritması (bees search algorithm)
- A* Araması (astar search)
- Geri izleme (backtracking)
- Işın arama (beam search)

2.2.2 Evrimsel Algoritmalar

Evrimsel Algoritmalar (Evolutionary Algorithms), bilgisayar bilimlerinde, yapay zekâ çalışmalarının altında geçen bir konudur. Kabaca, doğadaki değişimin bilgisayar algoritmalarına uygulanması olarak düşünülebilir. [4]

Evrim kavramı, insanın gözlem yeteneğine dayanarak doğayı ve doğadaki değişimi algılama sürecidir. Buna göre evrimin dayandığı pek çok temel, insan bakış

açısında göre şekillenir. Örneğin doğada rast gelelik bulunup bulunmaması bakan kişiye göre değişmektedir. İnsan için bilinmeyen değerler rast gele olabilir ancak tam bilgi durumunda rast gelelik kalkar. [4]

Evrimsel algoritmaların dayandığı önemli bir dayanak bu rast geleliktir. Örneğin çeşitli canlı toplulukları zaman içinde değişime uğramaktadır. Evrimsel yaklaşımla bu topluluklardan birey seçimi rast gele yapılmaktadır. Yeni nesiller, rast gele seçilen bireylerden türemektedir. [4]

Bütün bu kabuller, aslında insanın bu seçimi matematiksel olarak modelleyememesinden kaynaklanmaktadır. Kısacası rast geleliğin olması için bilgi eksikliğinin bulunması gerekir. [4]

Evrimsel algoritmalar da tam bu noktada devreye girer ve bilinmeyen, bilginin eksik olduğu problemlerde çözüm yolu bulmak için kullanılırlar. Evrimsel algoritmalarının tamamı bir rast gele süreç üzerine kurulur ve kısıtlı bilgi bulunan problemlere çözüm ararlar. [4]

Aslında çözüm aramanın en doğru ve kesin yolu bütün ihtimalleri denemektir. Ancak ne yazık ki bazı problemler için bütün yolların denenmesi kabul edilemeyecek kadar uzun sürer. [4]

Evrimsel algoritmalar, çözüm adayları içinde zeki bir rastgele arama yapar. Zor problemlerde verimli olmasına rağmen, analiz, basit sezgisel arama veya ayrıntılı numaralama yöntemleriyle çözülebilen kolay problemlerde seçilmemelidir. [4]

Evrimsel metotlar çeşitli akıllı robot mimarilerinde uygulanma fırsatı bulmuşlardır. Mesela, evrimsel algoritmalar kural tabanlı otonom ajanların kural kümelerinin öğrenilmesinde, robot kontrolü için kullanılan sinir ağlarının ağırlıklarının ve topolojisinin öğrenilmesinde, bulanık mantık kontrol sistemlerinde ve davranış tabanlı robotların kurallarında kullanılmaktadır. [4]

- **Çözüm Uzayının Seçimi ve Temsili**

Bir evrimsel algoritma kullanıcısının ilk belirlemesi gereken şey çözüm kümesi ve onun nasıl temsil edileceğidir. Bu seçim, işlerin ne kadarının insanlar ne kadarının makineler tarafından yapılacağını belirler. Mesela, tüm alıcı verilerinden tüm harekete geçirici komutlarına olan tüm eşlemeler arasında bir arama yapılacağında

yükün tamamı öğrenme sisteminin, dolayısıyla makinenin üzerindedir. Daha mantıklı olanı, alıcı verilerini işleyerek robot davranışlarını buna göre belirlemek ve eşlemeleri öğrenme işinden kurtulmaktır. Tasarımcı istenen davranışı sayısal parametrelerle belirleyebilirse, evrimsel algoritmalar en uygun parametre değerlerinin çözüm uzayında aranmasında kullanılabilir. [4]

- **Başlangıç Popülasyonu**

Evrimsel algoritmalar genellikle çözüm uzayından rastgele seçilmiş aday çözümlerle başlar. Başlangıç popülâsyonu seçilirken sezgi kullanılırsa yöntemin hızı artabilir. Bu seçim dikkatlice yapılmalıdır, çünkü birbirinden yeterince ayrık olmayan adayların olduğu bir başlangıç popülasyonu en optimal çözüme değil yerel optimal çözümlere yaklaşacaktır. [4]

- **Uygunluk Fonksiyonu**

Uygunluk fonksiyonunun tasarımında birkaç önemli zorunluluk vardır. Bu fonksiyon istenen performansı tam olarak yansıtmalıdır. Evrimsel algoritmalar çok duruma bağlı yöntemlerdir ve uygunluk fonksiyonu tasarımcının istemediği durumlara iyi sonuç veriyorsa sistemin çıktısı şaşırtıcı olabilir. Fonksiyon, değişken cezalarla alternatif çözümlerin de popülasyon da kalmasını sağlamalıdır. Tek çözümün pozitif uygunluk değerinin olduğu bir sonuç popülasyonu iyi değildir[4]

- **Genetik işlemciler**

Birçok genetik algoritma çalışması, yeni popülasyonlar oluşturmak için çaprazlama ve mutasyon işlemcilerini kullanmıştır. Ama bazı yeni çalışmalarda, robotun tecrübelerine dayanan daha sezgisel işlemciler kullanılmıştır. Mesela, bazı genetik sınırlandırıcı sistemlerde, alışılmadık bir durumla karşılaşıldığında yeni bir kural yaratan 'tetiklenen işlemciler' kullanılmıştır. [4]

- **Hibrit Yöntemler**

Çoğu zaman evrimsel algoritmaları, zayıf oldukları yerlerde kendilerinden daha güçlü metotlarla birlikte çalıştırmak iyi sonuçlar verir. Evrimsel algoritmaların gücü çözüm uzayının iyi sonuç vermeye en yatkın kısımlarını çabuk belirlemelerindedir ama aday çözümlerin ince ayarında daha zayıf kalmaktadırlar. Bu

yüzden, evrimsel algoritmanın bulduğu sonuçlar üzerinde bir yerel optimizasyon metodu kullanmak başarıyı artıracaktır. [4]

2.2.3 Genetik Algoritmalar

Genetik Algoritmalar evrimsel hesaplamanın bir parçasıdır. Bu alan Yapay Zekâ'nın hızla gelişen bir dalıdır. Genetik algoritmalar Darwin' in evrim teorisinden etkilenerek geliştirilmiştir. Basitçe açıklayacak olursak problemler evrimsel bir süreç kullanılarak bu süreç sonunda en iyi sonucu veren çözüme erişmeye çalışmaktadır. Başka bir ifadeyle çözüm evrimleşmektedir. Evrimsel hesaplama 1960'larda I.Rechenberg'in Evrim Stratejileri ("Evolutionsstrategie") adlı çalışmasında tanıtılmıştır. Daha sonra fikri diğer araştırmacılar tarafından geliştirilmiştir. Genetik algoritmalar John Holland tarafından icat edilmiş ve öğrencileri ve iş arkadaşları tarafından geliştirilmiştir. 1992 yılında John Koza, genetik algoritmaları kullanarak programları evrimleştirerek belli işleri yapmakta kullandı. Bu yönteme "genetik programlama" adını verdi. LISP dilinde programlar Ayrıştırma Ağaçları" ("Parse Tree") şeklinde ifade edildiği için LISP diliyle geliştirilmiştir. Ayrıştırma Ağaçları genetik algoritmaların çalıştığı temel nesnedir. [5]

Biyolojik Altyapı

Kromozom: Tüm yaşayan organizmalar hücrelerden oluşur. Her hücrede aynı kromozom kümeleri bulunur. Kromozomlar DNA dizileri olup, tüm organizmanın örneği olarak hizmet ederler. Bir kromozom gen adı verilen DNA bloklarından oluşur. Her gen belirli bir proteini kodlar. Basitçe, her genin, örneğin göz rengi gibi bir özelliği kodladığı söylenebilir. Bir özellik için olası ayarlar, (Örn. Mavi, Yeşil) alel olarak adlandırılır. Her gen kromozom üzerinde kendine ait bir konuma sahiptir. Bu konuma yörünge ("locus") adı verilir. Tüm genetik malzeme kümesine (tüm kromozomlar) genom adı verilir. Genom üzerindeki belli gen kümelerine genotip adı verilir. Genotipler, doğumdan sonra gelişmeyle fenotiplere - canlının göz rengi, zekâ v.b. fiziksel ve zihinsel özellikleri- dönüşür. [5]

Tekrar Üretim: Tekrar üretim sırasında, yeniden birleşme (veya çaprazlama) ilk önce ortaya çıkar. Atalardan gelen genler yepyeni bir kromozom üretmek için bir araya gelirler. Bu yeni yaratılmış nesil daha sonra mutasyona uğrayabilir. Mutasyon

DNA elemanlarının değişmesidir. Bu değişimler genellikle atalardan gen kopyalanması sırasındaki hatalardan kaynaklanır. Bir organizmanın uygunluğu ("fitness") organizmanın yaşamındaki başarısıyla (hayatta kalma) ölçülür. [5]

Arama Uzayı Eğer bir problemi çözüyorsak, genellikle çözümler arasındaki en iyi olanını arıyoruz demektir. Mümkün tüm çözümlerin uzayına (istenen çözümün aralarından bulunduğu çözümler kümesi) arama uzayı (durum uzayı) adı verilir. Arama uzayındaki her nokta bir olası çözümü temsil eder. Her olası çözüm değeri (uygunluğu) ile problem için "işaretlenebilir". Genetik algoritmalar yardımıyla arama uzayındaki olası çözümler arasından en iyi çözümü araştırırız. Çözümü aramak, arama uzayında aşırı noktaları (azami veya asgari) aramak ile aynı anlamdadır. Zaman zaman arama uzayı iyi tanımlanmış olabilir, ama bu arama uzayında sadece bir kaç noktayı biliyor olabiliriz. GA kullanma sürecinde, çözüm bulma süreci diğer noktaları (olası çözümleri) evrim sürdükçe üretir. Sorun, arama çok karmaşık olabilir. Nereden başlanacağı veya nereye bakılacağı bilinemeyebilir. Uygun çözümün bulunması için birçok yöntem vardır, fakat bu yöntemler en iyi çözümü üretmeyebilir. Bu yöntemlerin bazıları, tepe tırmanma ("hill climbing"), yasak arama ("tabu search"), benzetimli tavlama ("simulated annealing") ve genetik algoritmalardır. Bu yöntemler sonucu bulunan çözümler genellikle iyi çözümler olarak kabul edilir, çünkü sık sık en iyiyi bulmak ve ispatlamak mümkün değildir. [5]

NP-Zor ("NP-Hard") Problemler: "Geleneksel" yolla çözülemeyecek problem sınıfına bir örnek NP ("Non-deterministic Polynomial Time") problemlerdir. Hızlı (Çokterimli) algoritmaların uygulanabildiği birçok görev vardır. Ancak algoritmik olarak çözülemeyen bazı problemler de vardır. Çözüm bulmanın çok zor olduğu önemli problemler vardır, fakat çözüm bulununca bu çözümü kontrol etmek kolaydır. Bu gerçek "NP-Complete" Problemleri ortaya çıkarır. NP Nondeterministic Polynomial anlamına gelir ve bunun anlamı çözüm (Nondeterministic algoritma yardımıyla) "tahmin" edilebilir ve kontrol edilebilir. NP Problemlere örnek olarak tatmin problemini, gezgin satıcı problemini veya sırt çantası problemini verebiliriz. Daha fazla örnek için "A Compendium of NP Optimization Problems" sitesi incelenebilir. [5]

Genetik algoritmalar Darwin'in Evrim teorisinden esinlenilerek üretilmiştir. Bir problemin çözümü evrimsel süreç kullanılarak çözülmektedir. [5]

Algoritma toplum adı verilen ve kromozomlarla temsil edilen bir çözüm kümesi ile başlamaktadır. Bir toplumdaki çözümler yeni toplumların üretilmesinde kullanılmaktadır. Bu işlem, yeni toplumun eskisinden daha iyi olacağı umuduyla yapılmaktadır. Yeni çözümler (yavru) üretmek için alınan çözümler uygunluklarına (fitness) göre seçilmektedir. Daha uygun olan tekrar üretim için daha fazla şansa sahiptir. Bu süreç belli bir durum (örneğin belli sayıda toplum veya en iyi çözümün gelişmesi) karşılanana kadar tekrar edilmektedir. [5]

Basit Genetik Programlama Taslağı

Başlangıç: n kromozom oluşan rasgele toplum oluşturulur (problemin olası çözümleri)

Uygunluk: Toplumdaki her x kromozomu için f(x) uygunluk değeri değerlendirilir.

Yeni Toplum: Aşağıdaki adımlar izlenerek yeni toplum üretilir;

Seçim: Toplumdan uygunluklarına göre iki ata seçilir (daha uygun olanın seçilme şansı daha fazladır)

Çaprazlama: Çaprazlama olasılığı ile ataları yeni yavru oluşturmak için birbirleriyle eşleştirilir. Eğer çaprazlama yapılmazsa, yavru ataların tıpatıp aynısı olacaktır.

Mutasyon: Mutasyon olasılığı ile yeni yavru üzerinde her yörünge için mutasyon işlemi yapılacaktır.

Kabul: Yeni yavru, yeni topluma eklenir.

Değiştir: Yeni toplum algoritmanın tekrar işlenmesinde kullanılır.

Deney: Eğer bitiş durumu sağlandıysa, durup toplumdaki en iyi çözüm döndürülür.

Döngü: Adım 2'ye gidilir. (Kaynak)

Yukarıda da görüldüğü gibi, genetik algoritmanın akışı oldukça kolaydır. Birçok parametre ve ayar farklı problemler için farklı şekillerde gerçekleştirme için vardır. Sorulması sorulan ilk soru kromozomun nasıl kodlanacağıdır. Daha sonra

çaprazlama ve mutasyon, GA'nın iki basit işleci adreslenecektir. Bir sonraki soru çaprazlama için ataların nasıl seçileceğidir. Bu farklı birçok yolla yapılabilir, ancak ana fikir daha iyi ataların daha iyi yavrular üreteceği düşüncesiyle seçilmesidir. Bu şekilde en iyi çözümün kaybedilmemesi için seçkinlik, en iyi çözümün değiştirilmeden yeni nesle aktarılması, böylece en iyi çözümün yaşatılması uygulanabilir. [5]

Genetik algoritma taslağında görebileceğimiz gibi, çaprazlama ve mutasyon genetik algoritmanın en önemli kısımlarıdır. Başarım en çok bu iki işleçten etkilenir. Bu işleçlerden bahsetmeden önce kromozomlardan daha fazla bahsetmek gereklidir.

Kromozomun Kodlanması: Bir kromozom temsil ettiği çözüm hakkında bir şekilde bilgi içermelidir. En çok kullanılan kodlama ikili karakter dizisidir. Bu yöntemle kromozom şu şekilde görülmektedir: [5]

- Kromozom 1 : 1101100100110110
- Kromozom 2 : 1101111000011110

Her kromozom ikili karakter dizisi şeklinde temsil edilmektedir. Karakter dizisindeki her bit çözümün bir özelliğini temsil eder. Bir başka olasılık tüm karakter dizisinin bir sayıyı temsil etmesidir. Elbette, birçok başka kodlama yöntemi vardır. Kodlama daha çok çözülen probleme bağlıdır. Örneğin bazı problemler için tamsayı veya gerçek sayı şeklinde kodlamak gerekirken, bazı problemlerde permütasyon şeklinde kodlamaya ihtiyaç vardır.

Çaprazlama: Kodlamaya karar verdikten sonra, çaprazlama işlemiyle devam edebiliriz. Çaprazlama, atalardaki seçili genler üzerinde işlem yapar ve yeni yavrular oluşturur. Bunun en basit şekli, rasgele bir kesme noktası (çaprazlama noktası) seçip, bu noktadan önceki her şeyi ilk atadan, sonraki her şeyi ikinci atadan alıp birleştirerek yavruyu oluşturmaktır.

Çaprazlama aşağıdaki şekilde gösterilebilir: (| kesme noktasıdır):

- Kromozom 1: 11011 | 00100110110
- Kromozom 2: 11011 | 11000011110
- Yavru 1 : 11011 | 11000011110
- Yavru 2 : 11011 | 00100110110

Çaprazlamanın birçok yolu mevcuttur, örneğin birden fazla kesme noktası seçilebilir. Çaprazlama daha da karmaşık olabilir ve tamamen kromozomların kodlanmasına bağlıdır. Özel problemler için yapılmış özel çaprazlamalar genetik algoritmanın başarımını arttırabilir. [5]

Mutasyon: Çaprazlama işlemi gerçekleştirildikten sonra, mutasyon işlemi yapılır. Mutasyonun amacı, toplumdaki tüm çözümlerin çözülen problemlerin bir yerel uygun değerine düşmesinin önüne geçmektir. Mutasyon işlemi çaprazlama sonucu oluşan yavruyu rastgele değiştirmektedir. İkili kodlamada rastgele seçilmiş bir kaç biti 1'i 0'a, 0'ı 1'e şeklinde değiştirmek bir mutasyondur.

- Asıl Yavru 1: 1101111000011110
- Mutasyon Geçirmiş Yavru 1:1100111000011110
- Asıl Yavru 2: 1101100100110110
- Mutasyon Geçirmiş Yavru 2:1101101100110110

Mutasyon tekniği (çaprazlama tekniği de) kromozomların kodlamasına çoğunlukla bağlıdır. Örneğin permütasyon şeklinde kodlamada mutasyon rasgele seçilen iki genin yer değiştirmesi olarak gerçekleştirilir. [5]

Genetik Algoritmalarının Parametreleri

Çaprazlama Olasılığı: Bu parametre çaprazlamanın ne kadar sıklıkla yapılacağını belirtir. Eğer herhangi bir çaprazlama yoksa yavrular ataların aynısı olacaktır. Eğer bir çaprazlama yapılırsa yavrular ataların parçalarından oluşur. Eğer çaprazlama olasılığı %100 ise yavrular tamamen çaprazlama ile yapılır. Eğer %0 ise yavrular ataların kromozomlarının aynısına sahip olurlar. (Bu yeni toplumun aynı olduğu anlamına gelmez) Çaprazlama, yeni kromozomların eski kromozomların iyi parçalarını alıp daha iyi olacakları düşüncesiyle yapılır, ancak eski toplumun bazı parçalarının bir sonraki nesle aktarılması da iyidir. [5]

Mutasyon Olasılığı: Kromozom parçalarının ne kadar sıklıkla mutasyon geçireceğini belirtir. Eğer mutasyon yoksa yavrular çaprazlamadan hemen sonra değiştirilmeden üretilir (veya doğrudan kopyalanır). Eğer mutasyon varsa, yavruların kromozomlarının bir veya daha fazla parçası değişir. Eğer mutasyon olasılığı %100 ise tüm kromozom değişecektir. %0 ise hiçbir şey değişmez. Mutasyon genellikle

GA'nın yerel aşırılıklara düşmesini engeller. Mutasyonlar çok sık oluşmamalıdır, çünkü GA rasgele aramaya dönüşebilir.

Toplum Büyüklüğü: Toplumdaki kromozom (birey) sayısını belirtir. Eğer çok az birey varsa, GA'nın çaprazlama yapacağı olasılıklar azalacaktır ve arama uzayının çok küçük bir kısmı araştırılacaktır. Eğer çok fazla birey varsa, GA bayağı yavaşlayacaktır. Araştırmalar bazı sınırlardan sonra çok büyük toplumların kullanılmasının yararlı olmadığını göstermiştir, bu problemin daha hızlı bir şekilde çözülmesine yardımcı olmamaktadır. [5]

Kodlama: Kromozomların kodlanması bir problem çözümüne başlarken sorulması gereken ilk sorudur. Kodlama problemin kendisine yoğun şekilde bağlıdır.

İkili kodlama: İkili kodlama en çok kullanılan yöntemdir, çünkü ilk GA araştırmaları bu kodlama yöntemini kullanıldı ve görece basit bir yöntemdir. İkili kodlamada, her kromozom bit (0 veya 1) karakter dizilerinden oluşmaktadır. [5]

- Kromozom A: 10110010110010101110010 1
- Kromozom B: 11111110000011000001111 1

İkili kodlama, fazla olasılıkta kromozomlar verir, bunlara düşük sayıda alel içerenler de dahildir. Ancak, bu yöntem çoğu problem için doğal bir kodlama değildir ve çaprazlama ve/ya mutasyondan sonra düzeltmeler yapılması gerekir.

Örnek Problem: Sırt çantası problemi

Problem: Elimizde değeri ve boyutu verilmiş olan nesneler var. Sırt çantasının kapasitesi verilmiştir. Elimizdeki nesneler sırt çantasına azami sayıda çantanın kapasitesini aşmayacak şekilde yerleştirilmelidir. [5]

Kodlama: Her bit, şeyin sırt çantasında olup olmadığını belirtiyor.

Çaprazlama Yöntemleri:

Tek Noktalı Çaprazlama: Tek bir kesme noktası seçilir, ilk atanın kromozomundan kesme noktasına kadar baştan itibaren alınır ve geri kalan kısım ikinci atanın kesme noktasından sonraki kısmıyla birleştirilip yavrunun kromozomu oluşturulur (11001011+11011111 = 11001111).

İki Noktalı Çaprazlama: İki kesme noktası seçilir, kromozomun başından ilk kesme noktasına kadar olan ikili karakter dizisi ilk atadan, iki kesme noktası arasındaki kısım ikinci atadan ve ikinci kesme noktasından sonraki kısım tekrar ilk atadan alınarak yeni yavru oluşturulur (11001011 + 11011111 = 11011111).

Tek biçimli çaprazlama: Bitler atalardan rasgele olarak seçilip kopyalanır (11001011 + 11011101 = 11011111).

Aritmetik çaprazlama: Bazı aritmetik bit işlemleri atalar üzerinde uygulanarak yeni yavru oluşturulur (VE işlemi 11001011 + 11011111 = 11001001).

Mutasyon Yöntemleri:

Bit ters çevirme: Seçilen bitler terslerine çevrilir. (Bkz. Şekil 4.8 Yeni Bit= (NOT)Eski Bit işlemi 11001001 => 100010001)

Permütasyon Kodlama: Permütasyon kodlama, gezgin satıcı problemi veya görev sıralama gibi sıralama problemlerinde kullanılabilir. Permütasyon kodlamada, her kromozom sıra'da konum belirten numara karakter dizisinden oluşur. [5]

- Kromozom A: 1 5 3 2 6 4 7 9 8
- Kromozom B: 8 5 6 7 2 3 1 4 9

Permütasyon kodlama, sıralama problemleri için yararlıdır. Bazı problemlerde bazı çaprazlama ve mutasyon türleri için kromozomların tutarlılığı için (örneğin içerisinde gerçek sırayı tutan) düzeltmeler yapılması gerekmektedir. [5]

Örnek Problem: Gezgin satıcı problemi

Problem: Şehirler ve bu şehirler arasındaki uzaklıklar verilmektedir. Gezgin satıcı tüm bu şehirleri dolaşmak zorundadır. Fakat gereğinden fazla dolaşmamalıdır. En küçük dolaşma uzunluğunu verecek olan şehir dolaşma sırası bulunmalıdır. Kodlama: Kromozom şehirlerin gezgin satıcının dolaşacağı sırasını tutar. [5]

Çaprazlama yöntemleri (Tek Noktalı Çaprazlama): Bir kesme noktası seçilir, kesme noktasına kadar ilk atadan, kesme noktasından sonraki kısımlar da ikinci atadan olmak üzere permütasyonlar kopyalanır. Aynı sayılar olmayan sayılarla değiştirilerek tutarlı yeni yavru elde edilir. Bundan çok farklı, daha fazla sayıda yöntem de uygulanabilir. (1 2 3 4 5 6 7 8 9) + (4 5 3 6 8 9 7 2 1) = (1 2 3 4 5 6 8 9 7)

Mutasyon Yöntemi (Sıra Değiştirme): İki sayı seçilir ve yerleri değiştirilir. (1 2 3 4 5 6 8 9 7) => (1 8 3 4 5 6 2 9 7)

Değer kodlama: Gerçek sayılar gibi karmaşık değerlerin kullanıldığı problemlerde doğrudan değer kodlama kullanılabilir. İkili kodlamanın bu tip problemler için kullanılması problemlerin zorlaşmasına neden olacaktır. Değer kodlamada, her kromozom bazı değerlere eşittir. Değerler problemle ilgili herhangi bir şey olabilir. Gerçek sayılar, karakterler veya herhangi nesneler olabilir.

- Değer kodlama ile kodlanmış kromozom örnekleri:
- Kromozom A: 1.2324 5.3243 0.4556 2.3293 2.4545
- Kromozom B: ABDJEIFJDHDIERJFDLDFLFEGT
- Kromozom C: (geri), (geri), (sağ), (ileri), (sol)

Değer kodlama bazı özel problemler için iyi bir seçimdir. Ancak, bu tip kodlamada probleme özgü yeni çaprazlama ve mutasyon yöntemleri geliştirmek gereklidir. [5]

Örnek Problem: Bir sinir ağı için ağırlıkları bulma

Problem: Bir sinir ağı belirlenmiş mimariyle birlikte verilmektedir. Ağdan beklenen değeri almak için sinir ağındaki sinirler arasındaki ağırlıklar istenmektedir.

Kodlama: Kromozomlardaki gerçek değerler sinir ağındaki ağırlıkları temsil eder.

Çaprazlama yöntemi: İkili kodlamadaki tüm çaprazlamalar kullanılabilir.

Mutasyon Yöntemi (Küçük bir sayı ekleme -Gerçek sayı kodlama için-): Seçilen değerlere küçük bir sayı eklenir (veya çıkarılır). (1.29 5.68 2.86 4.11 5.55) => (1.29 5.68 2.73 4.22 5.55)

Ağaç Kodlama: Ağaç kodlama genellikle evrimleşen program veya ifadeler için kullanılmaktadır. Örneğin genetik programlama için Ağaç kodlamada her kromozom bazı nesnelerin ağacıdır, örneğin işlevler veya programlama dilindeki komutlar gibi. Ağaç kodlama evrimleşen programlar veya ağaç şeklinde kodlanabilecek herhangi diğer yapılar için uygundur. LISP programlama dilinde programların ağaç şeklinde temsil edilmesi nedeniyle LISP bu iş için en çok

kullanılan dildir. LISP'te bu ağaçlar kolayca ayrıştırılıp, çaprazlama ve mutasyon kolayca yapılmaktadır.

Örnek Problem: Verilen değer çiftlerini yaklaştıran fonksiyonu bulma

Problem: Girdi ve çıktılar verilmektedir. Görev tüm girdiler için en iyi çıktıları veren fonksiyonun üretilmesidir.

Kodlama: Kromozomlar ağaçta fonksiyonlar şeklinde temsil edilir.

Çaprazlama: Bir kesme noktası her iki atada da seçilir, atalar bu noktada bölünür ve kesme noktasının altında kalan parçalar değiştirilerek yeni yavrular oluşturulur.

Mutasyon (İşlev veya Numara değiştirme): Seçilen düğümler yer değiştirilir.

Seçim: Genetik Algoritması Akış şemasından da bildiğimiz gibi, kromozomlar toplum içerisinden çaprazlama için ata olmak için seçilmektedir. Problem bu kromozomları nasıl seçileceğidir. Darwin'in evrim teorisine göre en iyi olan yeni yavruyu yaratmak için yaşamına devam eder. Seçim düzeneklerine rulet tekeri seçimi, Boltzman seçimi, turnuva seçimi, sıralama seçimi, sabit durum seçimi ve diğerleri verilebilir. [5]

Rulet Tekeri Seçimi: Atalar uygunluklarına göre seçilirler. Daha iyi kromozomlar, daha fazla seçilme şansına sahip olanlardır. Toplumdaki tüm kromozomların yerleştirildiği bir rulet tekerini hayal edelim. Rulet tekeri üzerindeki kromozomun yerinin boyutu kromozomun uygunluğuyla orantılıdır. Daha uygun olan kromozom daha geniş bir kısma sahip olur. Bir bilye rulet tekerine atılmakta ve bilyenin durduğu yerdeki kromozom seçilir. Daha uygun olan kromozomlar böylece daha fazla sayıda seçilecektir. Süreç aşağıdaki algoritma ile anlatılabilir. Toplam: Toplumdaki tüm kromozomların uygunluk toplamını hesaplar - S. Seçim: (0,S) aralığından rasgele bir sayı üretilir - r. Döngü: Toplum üzerinden gidip 0'dan itibaren uygunlukların toplamını al - s, s r'den büyük olduğu zaman dur ve bulunduğumuz yerdeki kromozomu döndür. Elbette, aşama bir her toplum için bir kez yapılmaktadır. [5]

Sıralama Seçimi: Bir önceki seçim düzeneğinde uygunluk değerleri arasında büyük farklar oluşunca problemler ortaya çıkacaktır. Örneğin, eğer en iyi

kromozomun uygunluğu diğer tüm kromozomların toplamının %90'ı ise diğer kromozomların seçilme şansı çok azalacaktır. Sıralama seçimi ilk önce toplumu sıralar ve her kromozom uygunluk değeri olarak sırasını kullanır. En kötü 1 uygunluğunu, ikinci kötü 2,…., en iyi N (toplumdaki kromozom sayısı) uygunluğunu alır. Uygunluk sıraya göre belirlendiği zaman durum değişmektedir. Bu şekilde tüm kromozomların seçilme şansı olacaktır. Ancak bu yöntem daha yavaş yakınsama neden olabilir, çünkü en iyi kromozomlar birbirlerinden çok farklı değillerdir. Sabit Durum Seçimi: Bu özel bir ata seçme yöntemi değildir. Bu tip seçimin ana fikri, toplumun var olan kromozomlarının büyük bir kısmının yeni nesle aktarılmasıdır. Sabit durum seçimi şu şekilde çalışmaktadır. Her yeni nesilde yüksek uygunluk değerine sahip kromozomlar yeni yavruları oluşturmak için seçilir ve düşük uygunluk değerine sahip yavrular kaldırılarak yerlerine bu yeni oluşturulan yavrular koyulur. Toplumun geri kalan kısmı aynen yeni nesle aktarılır. [5]

Seçkinlik (Elitizm): Seçkinliğin ana fikri daha önce açıklandı. Çaprazlama ve Mutasyon yöntemleriyle yeni bir nesil oluştururken, en iyi kromozomları kaybetme olasılığımız vardır. Seçkinlik, en iyi kromozomların (ya da bir kısmının) ilk önce kopyalanıp yeni nesle aktarıldığı yöntemin adıdır. Geri kalan kromozomlar yukarıda anlatılan yöntemlerle üretilir. Seçkinlik GA'nın başarımını hızlı bir şekilde arttırabilir, çünkü bulunan en iyi çözümün kaybolmasını önler. [5]

2.2.4 Kripto Algoritmaları

Şifreleme işlevinin güvenli bir şekilde gerçekleştirilmesi kriptolama sırasında kullanılan tüm yöntem ve bilgilerin gizliliğine dayanır. Ancak herhangi bir nedenle kripto işlevlerinin açığa çıkabileceği düşünülerek iletişim güvenliği, kripto anahtarı denen ek bilgi ile arttırılmıştır. Bu durumda kriptolama işlemi sırasında açık mesaj, kripto anahtarı aracılığı ile şifrelenir. Bir başka deyişle açık mesaj ile kriptolanmış mesaj arasındaki geçişler kripto algoritmasına bağlı olduğu kadar kullanılan anahtar bilgisine de bağlıdır, [6]

Günümüzde kullanılan kripto grafik algoritmalar ikiye ayrılır. Bunlar, kullandıkları anahtar biçimine göre simetrik veya asimetrik olarak adlandırılırlar. Simetrik algoritmalarda verinin kriptolanmasında kullanılan anahtar bilgisi ile kriptolanmış verinin kriptosunun çözülmesinde kullanılan anahtar aynıdır. Bu sebeple açık kullanılan anahtarın üçüncü kişilerden gizlenmesi gerekmektedir. Bu

algoritmalara örnek olarak DES'i verebiliriz. Asimetrik algoritmalarda ise her iki kullanıcı ver iyi farklı anahtar bilgisi ile kriptolar ve çözerler. Bu amaçla her bir kullanıcı biri gizli diğeri açık iki anahtar kullanır. Açık anahtar bilgisi alıcı tarafa herhangi bir koruma yapılmadan iletilir. Açık anahtarı alan taraf bu bilgiden kriptoyu çözmek amacı ile kullanacağı gizli anahtarı üretir. Açık anahtarın üçüncü bir kişinin eline geçmesi tek başına hiç bir şey ifade etmez. Her ne kadar bu yöntemin, anahtarların alıcı tarafa iletilmesi işlemini basitleştirdiği düşünülse de asimetrik algoritmaların işlem süresinin yüksek oluşu, veri kriptolama da yaygın olarak kullanılmalarını engellemektedir. Hem simetrik hem de asimetrik algoritmaların temel özelliği yapılarının açık olarak bilinmesidir. Böylece kripto algoritmalarının tasarımında, kriptolanmış verinin anahtar bilgisi olmadan şifrenin çözülememesi ilkesi göz önüne alındığından, iletişim güvenliği, kripto anahtarlarının güvenliği problemine indirgenmiştir.

Sezar Şifresi: En eski kripto algoritmalarından biri olan Sezar şifresi adını Julius Caesar'dan almaktadır. Orijinal olarak kripto algoritması şu şekilde ifade edilebilir:

$$S=M+3 \pmod{26}$$

Burada S açık metin harfini M gizli metin harfini göstermektedir. Daha sonra bu algoritma genelleştirilerek şu biçimi almıştır,

$$S = M + i \pmod{26}, \quad 0 <= i < = 25$$

Burada i öteleme katsayısı genel anlamda bir kriptografik anahtara karşılık düşmektedir. Burada kullanılan i anahtarının tanım aralığının 25 sayısı ile kısıtlı olması algoritmanın güvenilirliğinin düşük olmasına yol açmaktadır.

Tekli Alfabetik Yer Değiştirme: Daha yüksek güvenlikli kripto sistemleri düşünüldüğünde alfabede bulunan her harfin yerine başka bir harfin, belirli bir tablo uyarınca yerleştirildiği yapılar ortaya çıkmıştır. Bu tekniğe tekli alfabetik yer değiştirme adı verilir. Bu teknik özel bir alt uygulaması olarak Sezar şifresini de içermektedir. Olası uygulama sayısının yüksekliği göz önüne alındığında tekli alfabetik yer değiştirme güçlü bir algoritma olarak düşünülebilir. Ancak şifrelemeden sonra ortaya çıkan gizli metinde bulunan harflerin dağılım sıklığı, kullanılan diller

göz önüne alındığında bazı istatistiksel ipuçları verdiğinden dolayı, bu tür şifrelemenin de çözülebilmesi oldukça basittir[6]

Çoklu Alfabetik Yer Değiştirme: Çoklu alfabetik yer değiştirme tekniğinde, alfabede bulunan her bir harf üzerine n periyotlu bir dizi yer değiştirme uygulanır. Bu yöntem, dilin istatistiksel olarak tahmin edilebilen yapısından kaynaklanan birebirliği bozmayı amaçlamaktadır. Çoklu alfabetik yer değiştirme yönteminin bir uygulaması olan Vigenere şifresi periyodik olarak belirli bir anahtar değerinde yer değiştirme uygular. Bu uygulamada K anahtarı bir dizi harf olar ak belirlenir. Bu durumda

K = k1, k2,.... kd olarak gösterilen ifadede kullanılan k; değeri i=l,2,...,d olmak üzere, alfabede yapılması gereken i'nci ötelemeyi gösterdiğinde, şifreleme sonucu elde edilen harf,

$f(a) = a + ki \pmod n \quad (n = 26)$

Biçiminde gösterilir. Yer değiştirme şifrelerinde sistemin güvenilirliği anahtar uzunluğuna bağlı olarak değişir. Eğer anahtar uzunluğu mesaj uzunluğunda olur ise bu duruma, çalışan anahtar şifresi adı verilir. [6]

Tek Kullanımlı Şerit Yöntemi Analitik olarak koşulsuz güvenli tek şifreleme yöntemidir. Bunu gerçekleştirmek amacı ile tek bir kez kullanılmak üzere, mesaj uzunluğuna eşit veya daha uzun, tümüyle rassal bir anahtar seçilir. Anahtar, ikili sayı düzeninde düşünülen mesaj ile dışveya'lanır. Ancak bu sistem birçok uygulama için oldukça kullanışsızdır. Çünkü her bir kullanım için en az mesaj uzunluğunda olan anahtarın, haberleşme öncesi her iki tarafa da ulaştırılması gerekmektedir. [6]

Dönüşüm Şifreleri: Yer değiştirme şifreleri açık metinde bulunan tüm sembollerin yerini değiştirmezken sadece bu sembollerin yerine şifreleme amacı ile kullanılacak olan yeni sembolleri belirler. Ancak dönüşüm şifrelerinde açık metnin harfleri de yer değiştirir. Bu yöntem ilk olarak Eski Yunanda kullanılmıştır. Uzun dar bir şerit üzerine yazılan açık metnin, çapı anahtar bilgisi olan bir silindire spiral olarak dolandırılması ile elde edilen gizli metnin çözülmesi ancak aynı çaplı bir silindire gizli mesajın sarılması ile mümkün oluyordu. Bu sistemde, kullanılan harflerin dağılım sıklığı korunuyordu. Ancak dilin yapısında olan ikili, üçtü ve daha yukarı seviyede harflerin bir araya gelmesi olasılığı bozulduğundan, bu yöntem, dil

üzerinde basit yer değiştirmeye dayalı kripto yöntemlerine göre daha güvenli olarak kabul edilir. [6]

Günümüzde kullanılan kriptografik algoritmalar ikiye ayrılır. Bunlar, kullandıkları anahtar biçimine göre simetrik veya asimetrik olarak adlandırılırlar. Simetrik algoritmalarda verinin kriptolanmasında kullanılan anahtar bilgisi ile kriptolanmış verinin kriptosunun çözülmesinde kullanılan anahtar aynıdır. Bu sebeple açık kullanılan anahtarın üçüncü kişilerden gizlenmesi gerekmektedir. Bu algoritmalara örnek olarak DES'i verebiliriz. Asimetrik algoritmalarda ise her iki kullanıcı ver iyi farklı anahtar bilgisi ile kriptolar ve çözerler. Bu amaçla her bir kullanıcı biri gizli diğeri açık iki anahtar kullanır. Açık anahtar bilgisi alıcı tarafa herhangi bir koruma yapılmadan iletilir. Açık anahtarı alan taraf bu bilgiden kriptoyu çözmek amacı ile kullanacağı gizli anahtarı üretir. Açık anahtarın üçüncü bir kişinin eline geçmesi tek başına hiç bir şey ifade etmez. Her ne kadar bu yöntemin, anahtarların alıcı tarafa iletilmesi işlemini basitleştirdiği düşünülse de asimetrik algoritmaların işlem süresinin yüksek oluşu, veri kriptolamada yaygın olarak kullanılmalarını engellemektedir. Bu algoritmalara örnek olarak RSA'ı verebiliriz Hem simetrik hem de asimetrik algoritmaların temel özelliği standart haline gelebilmesi için algoritma yapılarının açık olarak bilinmesidir. [6]

Veri Şifreleme Standardı – Data Encryption Standard (DES) En çok kullanılan şifreleme tekniği 1977'de şimdiki adı Ulusal Standart ve Teknolojiler Enstitüsü olan Ulusal Standartlar Bürosunda ortaya atılan Veri Şifreleme Standardıdır (DES). DES' de veri, 56-bitlik bir anahtar kullanılarak 64-bitlik bloklar halinde şifrelenir. Algoritma, 64-bitlik bir girişi bazı aşamalar sonucu 64-bitlik bir çıktı oluşturacak şekilde dönüştürür. Şifrelemeyi geri almak için aynı adımlar, aynı anahtar kullanılarak işlenir. DES' in çok geniş bir kullanım sahası vardır. Güvenilirlik derecesi ise tartışıla gelen bir konu olmuştur. DES şifrelemesinin tümüyle akışı Şekil 2'de gösterilmiştir. Diğer tüm şifreleme yöntemleri gibi bunda da iki girdi vardır: şifrelenecek düzyazı ve anahtar. Burada düzyazı 64-bit, anahtar ise 56-bit uzunluğunda olmalıdır[6]

Rivest-Shamir-Adleman -RSA- Kriptosistemi RSA şifreleme algoritması 1977 yılında R.Rivest, A.Shamir ve L.Adleman tarafından bulunmuş ve daha sonra asimetrik şifreleme algoritmalarına (genel anahtar şifrelemesi) uygun biçimde

geliştirilmiştir. Bu algoritma, asimetrik şifreleme algoritmalarında ve dijital imza işlemlerinde güvenli bir şekilde kullanılır. Görünüşte son derece basit matematiksel ilişkilerle çalışan bu yöntem de iki ayrı anahtar bulunmaktadır. Anahtarlardan birisi kamuya açık, birisi de gizlidir. Herkes açık anahtarını yayınlar ve kendisine şifreli bir mesaj göndermek isteyen birisi bu anahtarı kullanarak mesajı şifreler ve gönderir. Ancak mesajı sadece gizli anahtar kimde ise o çözebilir. Gizli anahtar da sadece sahibinde bulunur. Böylece, herkes çözüm için gerekli anahtarı bilmeden, güçlü bir şifreyle mesajları gizleyebilir. Daha önce hiç karşılaşmamış, birbirini tanımayan kişiler bile birbirlerine gizli mesajlar gönderebilir. Örneğin Internet'ten alışveriş yapan birisi, kendisini hiçbir şekilde tanımayan bir web sitesine giderek, sitenin kamuya açık anahtarını alır, kart numarasını bu anahtarla şifreleyerek gönderir. Şifreli bilgiyi gönderen dahil hiç kimse çözemez, sadece web sitesinde bulunan gizli anahtarla gelen kart numarasını web sitesi çözebilir. Böylece kart hamili kart numarasının başkası tarafından okunmayacağından emin olacaktır. Ama acaba Web sitesi gerçekten dürüst bir satıcı mı, yoksa sahte bir site mi? Bundan emin olamayacaktır, ancak bunun da çözümü sertifika yöntemiyle sağlanmaktadır[6]

2.2.5 Kök Bulma Algoritmaları

Kök-bulma Algoritması verilen bir fonksiyonda fonksiyonun değerini sıfır yapacak bir x değerini bulmaya yarayan bir nümerik metot ya da algoritmadır (öyle bir x bul ki f(x) = 0 olsun). Böyle bir x değerine fonksiyonun kökü denir. f − g kökünü bulma işlemi, f(x) = g(x) denklemini çözmekle aynı işlemdir. Buradaki x değerine ise denklemin bilinmeyeni denir. Bunun yanında her denklem, denklem çözmenin fonksiyonun bilinmeyenini bulmaya eşit olduğu f(x) = 0 şeklinde bir kanonik form alabilir. Bütün nümerik kök-bulma metotları tekrarlama, sonunda kök olacak bir limite yakınsayacak sayı serisi üretme, yöntemini kullanır. Kök-bulma algoritmalarının davranışları nümerik analizde incelenir. [7]

En basit kök bulma algoritması ikiye bölme metodudur. Yalnızca f sürekli fonksiyonsa uygulanabilir. Ayrıca iki ilk tahmine ihtiyacı vardır. Bu ilk tahminler a ve b öyle değerler olmalıdırlar ki; f(a) ve f(b)'nin birbirine zıt işaretli olmalıdır.Bunun yanında Newton metodu, sekant metodu, yanlış pozisyon metodu, Müller metodu ve Brent metodu gibi algoritmalar kök bulmada kullanılmaktadırlar.

Polinomların köklerini bulmak için özel algoritmalar geliştirilmiştir. Bunlar genel olarak, polinomların kompanyon matrisinin bulunması, Laguerre metodu, Bairstow metodu, Durand-Kerner metodu ve daire bölme metodu gibi algoritmalardır. [7]

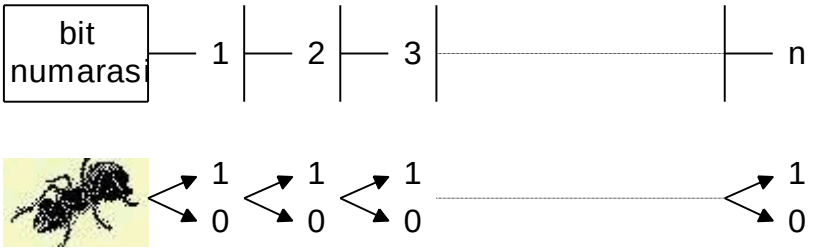
2.2.6 Karınca Kolonisi Optimizasyon Algoritmaları

Bu algoritmalar, karıncaların doğal davranışlarını taklit ederek sürekli ve süreksiz problemleri çözmek için kullanılır. Bu algoritmaların en çok kullanıldığı problem TSP yani seyahat eden tüccar (travelling salesman problem) problemidir. Biz daha sonra bir Japon bilim adamı ve arkadaşları tarafından ortaya atılan TACO (touring ant colony optimisation algorithm) yöntemini sürekli fonksiyonların optimizasyonu için kullanacağız. [8]

Karınca kolonisi yöntemi pozitif geri beslemeli bir yöntemdir. Bu yöntemde temel alınan karıncaların davranışını biraz açıklarsak , bir karınca evi(nest) ile besin(food) kaynağı arasında gidip gelmek için çevre şartlarına göre gidebileceği yolları belirler. Bu yollardan birinden geçen ilk karınca yolun kısalığına göre (yol kısa ise daha çok olmak üzere) yola koku bırakır bu kokuya phremone denir. Daha sonra gelen karıncalar da aynı şeyi yaparak yolda devam ederler. Bir karınca iki yolun birleştiği noktada bir yol tercihi yapacaktır. Bu yolun seçimini öncelikle yoldaki koku miktarına, ikinci derecede ise rasgelelikle yapar yani bir yol hem koku miktarına hem de randomize bir ölçütle tercih edilir. Eğer yol sadece koku ile seçilecek olsaydı bütün karıncalar sadece ilk geçen karıncanın geçtiği yolları tercih edecek ve bu nedenle ilk seçilen yola kıyasla daha kısa yolları keşfetme imkanını bulamayacaktı. [8]

Görüldüğü gibi bu şekilde kısa yollarda daha fazla koku bulunacak ve bir karıncanın bu yolu seçme ihtimali artacaktır. Daha geniş manada ise bu algoritmalarda yola bırakılan kokunun artması durumunda bu kokuyu azaltmak için yoldan karınca geçtikten sonra veya belirli bir zaman periyodu sonrasında belirli miktarda koku buharlaşması(phremone decay) olmalıdır. Bu buharlaşma ile daha değişik yol kombinasyonu ve dolayısıyla daha kısa bir yol bulma ihtimali doğacaktır. Bunun dışında her karıncanın bir hafızası olmalı bu şekilde her karınca attığı en iyi turu hafızasında tutar. [8]

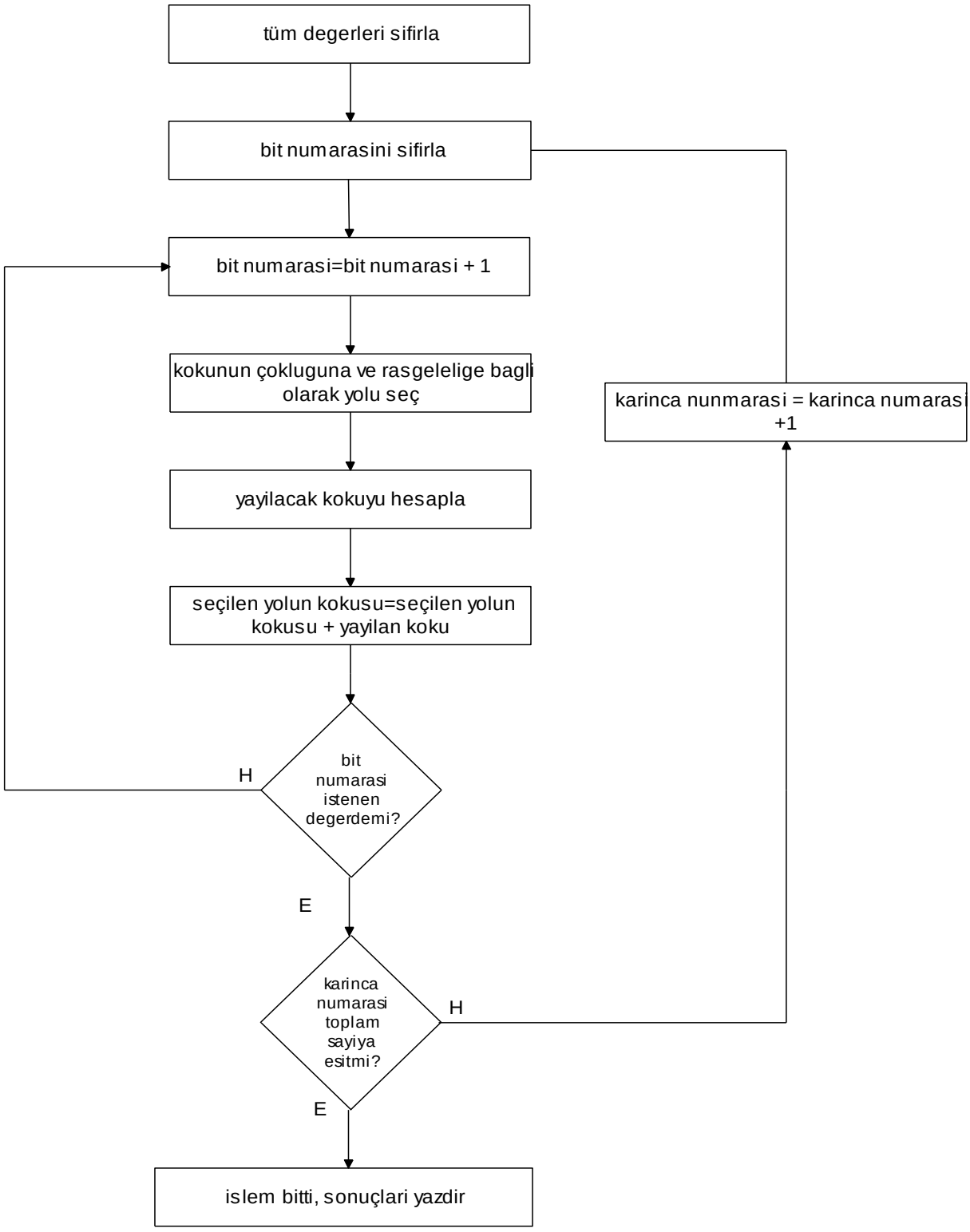
Daha genel baktığımız karınca kolonisi algoritmalarını daha özelleştirirsek önce bir TACO yöntemine bakalım.



TACO yönteminde bir karınca şekilde görüldüğü üzere n adet seçim yaparak 2n adet yoldan tercih eder ,her seçimden sonra daha önce seçtiği yolların 2'lik sistemdeki değerliklerini 10'luk sisteme çevirerek koku miktarını hesaplar ve seçtiği yola koku bırakır. Böylece bir karınca en yüksek değerlikli bitten en düşük değerlikliye yani 10'luk sistemde ondalıktan küsurata doğru seçim yapar. Bu yöntem için koku miktarının hesaplanması şöyledir: [8]

$$\text{deltaphremone} = \frac{Q}{f(x_i)}$$

Burada Q , koku yayma katsayısı olup optimizasyon işlemi sırasında değiştirilerek en iyi değer bulunur. f(x) fonksiyonu ise optimize edilecek fonksiyondur.Bir değişken için yapılacak işlemin diyagramı ise aşağıdaki gibidir: [8]



Şekil 2 Karınca Algoritması Akış şeması

optimize edeceğimiz fonksiyon birden çok değişkenlidir. İşlem bütün karıncalar yolu takip ederek sonuca ulaştığında biter. TACO' da buharlaşma yani koku azalması yoktur. Ancak Gaziantep üniversitesinde yapılan çalışmalarda koku buharlaşması kullanılmıştır. Bu çalışmalar daha iyi sonuç vermiştir. [8]

2.2.7 Sıralama Algoritmaları

Sıralama algoritmalarını kullanmamızdaki amaç, -algoritmanın isminden de anlaşılacağı üzere- sahip olduğumuz veriyi en hızlı şekilde büyükten küçüğe (ya da küçükten büyüğe) bir sıraya sokmak. Bunun için kullanılan bir çok sıralama algoritması mevcut. Bazısı çok hızlı ama yazımı zor, bazısı az sayıda veri için çok hızlı, bazısının da yazması kolay. Bu yazımızda, en populer sıralama algoritmaları ile ilgili bilgiler vermeye çalışacağız. [9]

Elemeli Sıralama(Bubble Sort): Sıralama algoritmaları arasında en yavaş çalışanıdır. Aslında bilgisayar bilimi eğitiminde, sıralama algoritması hakkında bir temel oluşturulması için öğretilmesi dışında pek kullanılmaz. Elemeli sıralamada yapılan kısaca şöyledir: Veri setinin ilk iki elemanı okunup karşılaştırılır. İlk veri ikinciden küçükse, ikinci ve üçüncü veri karşılaştırılır; büyükse ikisi yer değiştirir. Bu işlemler son veriye ulaşana kadar devam eder ve doğru sıralamaya ulaşana kadar tekrar baştan başlar. Örnek verelim: [9]

(5 1 4 2 8) --> (1 5 4 2 8) : burda, ilk iki veri karşılaştırılıyor ve ikinci küçük olduğu için yer değiştiriyorlar.

(1 5 4 2 8) --> (1 4 5 2 8)

(1 4 5 2 8) --> (1 4 2 5 8)

(1 4 2 5 8) --> (1 4 2 5 8) : son elemana ulaşılıyor ama setimiz hala sıralı değil, o yüzden tekrar baştan başlıyoruz.

(1 4 2 5 8) --> (1 4 2 5 8)

(1 4 2 5 8) --> (1 2 4 5 8)

(1 2 4 5 8) --> (1 2 4 5 8)

(1 2 4 5 8) --> (1 2 4 5 8)

Seçmeli sıralama (selection sort): Basit ve yazması kolaydır. Ama elemeli sıralamaya göre daha iyi olsa da, diğer algoritmalara göre yine yavaş çalışır. Seçmeli sıralamada yapılan kısaca şöyledir: Veri seti sırayla okunur ve en küçük olan eleman, setimizin ilk elemanıyla yer değiştirir. İkinci elemandan itibaren set bir daha okunur

ve en küçük veriyle ikinci veri yer değiştirir. Bu işlem son veriye gelene kadar devam eder. Örnek verelim:

31 25 12 22 11: veri setimiz

11 25 12 22 31: en küçükleri olan 11, ilk sıradaki veri ile yer değiştirdi.

11 12 25 22 31

11 12 22 25 31

Eklemeli sıralama(Insertion Sort): Eklemeli sıralamaya göre daha hızlı olan bu sıralama, küçük setler için daha uygundur. Seçmeli sıralamaya benzer ama ona oranla daha hızlıdır. Bu sıralama iki şekilde yapılabilir. Birincisi, bundan önce anlattığımız sıralamalarda olduğu gibi, aynı set içerisinde sıralama, diğer yöntem ise, fazladan bir boş set daha kullanılarak yapılan sıralama. İkinci yöntemde fazladan set ve dolayısıyla fazladan hafıza kullanıldığı için, ilk yöntem daha çok tercih edilir. Eklemeli sıralamada yapılan işlem kısaca şöyledir:
[9]

Veriler sırayla okunur ve kendinden önceki verilerle kıyaslanıp en uygun yere konur. Bu algoritmayı yerden sırayla çektiğimiz iskambil kartlarını sıralamamıza benzetebiliriz. Örnek verelim:

34 8 64 51 32 21: ilk listemiz

8 34 64 51 32 21: 8, 34 ün önüne eklendi.

8 34 64 51 32 21: 64, kendisinden önceki rakamlardan büyük olduğu için yerinde kaldı.

8 34 51 64 32 21

8 32 34 51 64 21

8 21 32 34 51 64

Birleştirmeli Sıralama (Merge sort): Böl ve işgal et (divide and conquer) stratejisi kullanılır. [9] Bu sıralama algoritmasında, fazladan hafıza kullanılır.

Yukarıda bahsettiğimiz algoritmalara göre çok daha hızlıdır. Yapılan işlem kısaca şöyledir:

Setimiz, iki veri kalana kadar parçalara ayrılır (sürekli yarıya bölünerek) ve her ikili parça kendi içinde sıralanır. Her iki parça sıralandıkça, diğer ikili parçayla kıyaslanıp sıralanır ve böylece elimizde sıralı veri dizisi oluşur. [9]

Çabuk Sıralama (Quick Sort): Birleştirmeli sıralama algoritmasında olduğu gibi, çabuk sıralama algoritması da böl ve işgal et (divide and conquer) stratejisini kullanır. Çabuk sıralama performans olarak diğer sıralamalara göre çok iyidir. Özellikle büyük boyutlu setlerde bu algoritmayı kullanmak çok önemlidir. Fakat çabuk sıralama algoritmasını koda dökebilmek zordur. Bu yüzden, sıralamaya ihtiyaç duyan kişi, bu algoritmayı yazmak için harcayacağı zamanın, sıralama yaparken kazanacağı zamana oranını iyi hesaplaması gerekir. Çabuk sıralamada yapılan işlem kısaca şöyledir: [9]

Veri setimiz iki parçaya ayrılır, bu işlem için bir veri pivot olarak seçilir, kalan iki parça ise, pivottan küçükler ve pivottan büyüklerin oluşturduğu parçalar olur. Bu iki parçaya da yine çabuk sıralama algoritması uygulanır. Yani çabuk sıralama algoritması, kendi içerisinde yine kendini çağırır (özyineleme - recursion). Örnek verelim:

27 63 1 72 64 58 14 9: Pivot 9 olsun.

1 9 63 72 64 58 14 27 : 9'dan küçükler 9 un soluna, büyükler sağına yerleştirildi. Sağ taraftaki set için pivot 27 olsun.

1 9 14 27 64 58 72 63: 27 den küçükler 27'nin soluna, büyükler sağına yerleştirildi.

1 9 14 27 58 63 72 64

1 9 14 27 58 63 64 72

En yaygın olan beş algoritmayı sizlerle paylaştık. Gördüğünüz gibi sıralama işlemi için kullanılacak algoritmayı seçmek için; hız mı, bellek kullanımı mı yoksa sıralama programını yazmanın kolaylığının mı önemli olduğuna karar vermek çok önemli. [9]

2.2.7 Veri Sıkıştırma Algoritmaları

Veri sıkıştırma işlemi, belirli uzunluktaki verilerin çeşitli yöntemlerle daha az bellek kullanılması amacıyla geliştirilmiştir. Bu sayede bellek üzerinde yer tasarrufu, veri aktarımında da zaman tasarrufu yapılabilmektedir. [10]

Veri sıkıştırma yöntemleri iki grupta incelenir. Kayıplı ve Kayıpsız sıkıştırma. Kayıplı sıkıştırma daha çok multimedia verilerde kullanılır. Mpeg ve MP3 bunun en yaygın örneğidir. MP3'lerde insan kulağının duyamayacağı ses dalgaları kayıda alınmaz. Ayrıca JPEG resim sıkıştırma formatında da resim üzerinde belirli ara renk kayıpları olmaktadır. [10]

İkinci tip sıkıştırma ise kayıpsız sıkıştırmadır. Özellikle sayısal sonuçların önemli olduğu durumlarda kullanılır. Mesela bir exe dosyayı sıkıştırdığımızda hiçbir verinin kaybolmasını istemeyiz. Çünkü birkaç byte'ın bile farklı olması exe'nin çalışmamasına yol açabilir. Yukarıda bahsedildiği gibi tüm Multimedia formatları kayıplı değildir. GIF ve PCX formatı kayıpsız olarak sıkıştırma yapabilmektedir. [10]

RLE Sıkıştırma Algoritması: En basit veri sıkıştırma yöntemidir. RLE(Run Lenght) yönteminde veriler adetleri tutularak sıkıştırılır.

A A A A B B B C C A A A A A A A B B B B B A B B B B C C C C B B B (33 karakter)
4A 3B 2C 7A 5B 1A 4B 4C 3B (18 karakter)

Kırmızı sayılar verilerin tekrar miktarını göstermektedir. Bu şekilde 33 karakterlik bir veri dizisi 18 karaktere sıkıştırılmıştır. Bu metod teoride doğrudur fakat pratikte o kadar da verimli değildir. Hatta orjinal veriden daha fazla yer kaplayabilme ihtimali de vardır. Eğer "A B A B A B" verisini sıkıştırmak istersek "1A 1B 1A 1B 1A 1B" şekline dönüştürmemiz lazım bu durumda 6 karakterlik veriyi 12 karaktere çıkarmış oluruz. Yani genişletmiş oluruz. Ayrıca çok uzun verilerde veri uzunluğu bilgisini Byte tipinden Word tipine çevirmek gerekir. Fakat 65535 karakterden uzun bir veriyle karşılaşılırsa yeni bir veri paketi oluşturmak gerekir. Yani 100 MB'ı sıkıştırsam diye bir bakmışsın 200MB'lık sıkıştırma dosyası oluşturmuşsun. Etkin bir metod değil. RLE metodu tekrarı fazla olan verilerde kullanıldığında başarılıdır. Tekrarsız verilerde ise kesinlikle uygun değildir. Fakat bir çok sıkıştırma tekniği bu metod üzerine kurulmuştur. [10]

HUFFMAN Algoritması: Sıkıştırma oranı en yüksek algoritmalardan biridir. Karşılaşılan veri katarları bir tabloda tutularak aynı veri katarı ile karşılaşıldığında sadece tablodaki numarası yazılarak veri kısaltılmış olur. [10]

ABCDEFABCABCABCABCDEABCDEFABCABCDE (34 karakter)

ABCDEF + ABC + ABC + ABC + ABCDE + ABCDEF + ABC + ABCDE

ABCDEF	1	6 byte
ABC	2	3 byte
ABCDE	3	5 byte
ABCDEF	4	6 byte

20 byte

1 2 2 2 3 4 2 3 (8 karakter)

Yukarıdaki örnekte 34 byte'lık bir veri dizisi (20 + 8 = 28) byte uzunlukta sıkıştırılmıştır. Eğer aynı veriyi RLE metodu ile sıkıştırmış olsaydık 68 byte'lık bir veri elde edecektik. Yani tam 2 katına çıkaracaktık. LWZ / HUFFMAN algoritmasının başarı oranı, tekrarını bulabildiği daha uzun katarlar ile ölçülür. Yukarıdaki veri dizisi aşağıda daha iyi sıkıştırılmıştır. Bu sefer tekrarı olan daha uzun metin katarları seçilmiştir. Bu şekilde (17 + 6 = 23) byte uzunluk elde edilmiştir. [10]

ABCDEFABCABCABCABCDEABCDEFABCABCDE (34 karakter)

ABCDEFABC + ABC + ABC + ABCDE + ABCDEFABC + ABCDE

ABCDEFABC	1	9 byte
ABC	2	3 byte
ABCDE	3	5 byte

17 byte

1 2 2 3 1 3 (6 karakter)

2.3 Kuyruk Algoritmaları

- Sokuşturma Algoritması (Insertion Algorithm)
- Silme Algoritması (Deletion Algorithm)
- Yığın Kuyruk Algoritması (Heap queue algorithm)
- Çoklu Kuyruk Algoritması (Multi Queue Algorithm)
- Azalan Kuyruk Algoritması (Tail Drop Algorithm)
- Adil Kuyruk Algoritması (Fair Queue Algorithm)
- Stokastik Adil Kuyruk Algoritması (Stochastic Fair Queuing Algorithm)

- Robin Açık Daire Algoritması (Deficit Round Robin Algorithm)
- Rastgele Erken Bulma Algoritması (Random Early Detection Algorithm)
- Sınıf Tabanlı Kuyruk Algoritması (Class Based Queuing Algorithm)

2.3.1 Sokuşturma (Insertion) Algoritması

Günlük yaşamda hoşumuza gitmeyen bir olayla sık sık karşılaşırız. Banka gişesi, alışveriş kasası, otobüs durağı gibi yerlerde, insanlar geliş önceliklerine göre bir kuyruk oluştururlar. Diyelim ki banka gişesi önünde 9 kişi sıraya girmiş bekliyor. O anda yeni bir müşteri geliyor. [11]

a. Yeni gelen müşteri kuyruğun sonuna geçerse, öndekiler için bir sorun doğmaz, ama gişenin yapacağı işlem sayısı 1 artar.

b. Yeni gelen müşteri kuyruğun en önüne geçerse, kuyruktaki 9 kişinin sırası birer geriye kayacaktır.

c. Yeni gelen müşteri kuyrukta ortalarda bir yere girerse, o konumdan sonrakilerin hepsinin sırası birer geriye kayacaktır.

Bu olay, bilgisayar jargonunda Insertion Algorithm (sokuşturma algoritması) adıyla adlandırılır ve programlanabilir. Bu kesimde onu inceleyeceğiz. Söz konusu kuyruk, bizim için bir array (dizi) dir.

Gösterimlerde basitliği sağlamak için, bir a[] dizisini (array) a*alt…üst+ simgesiyle gösterelim. Burada alt dizinin en küçük indisi, üst ise en büyük indisidir. Örneğin, a[0,10] gösterimi

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10]

Dizisi yerine kullanılacaktır. a[4,7] gösterimi ise a[4] a[5] a[6] a[7]
alt dizisi yerine kullanılacaktır.

Bir dizi yaratıldığında, her terimi için ana bellekte bir adres ayrılır. Verilen diziye bir terim eklemek için, ana bellekte o terime bir yer açmak gerekir. Yeni terimin dizinin başına, sonuna ya da ortalarda
bir yere eklenmesi istenebilir. Her durumda, eklenecek terimin gireceği yer (konum) belli ise, indisi biliniyor demektir. Dolayısıyla, dizinin o indis ve sonraki indisli

terimlerinin hepsinin indisleri birer artırılmalıdır ki eklenen terim istenen yere yerleşebilsin. Şimdi bunu bir örnekle açıklayalım. [11]

tavşan karabaşın üstünden atladı

tümcesindeki sözcükleri sırayla bir dizinin terimleri olarak yazalım.

a[0]	a[1]	a[2]	a[3]
tavşan	karabaşın	üstünden	atladı

a. Array(dizi)'in sonuna ekleme (sokuşturma): tümcesini tavşan karabaşın üstünden atladı kaçtı diye değiştirelim. O durumda dizimiz

a[0]	a[1]	a[2]	a[3]	a[4]
tavşan	karabaşın	üstünden	atladı	kaçtı

biçimini alır. Bunu yapabilmek için, a[4] öğesi için bellekte yeni bir yer açılmalıdır. Ancak, array(dizi) yaratılırken terim sayısı belirlenir ve o sayı değiştirilemez. O nedenle, bu işlem için ana bellekte şu işlerin yapılması gerekir:

1. Beş terimli bir b*0…4+ array(dizi)'ini yarat
2. a*0…3+ array(dizi)'inin terimlerini sırayla b*0…3+ alt array(dizi)'i üstüne kopyala
3. b*4+ = "kaçtı" atamasını yap
4. a*0…3+ array(dizi)'ini sil
5. b array(dizi)'inin adını değiştir; a yap

b. Array(dizi)'in önüne ekleme (sokuşturma): tümcesini çevik tavşan karabaşın üstünden atladı diye değiştirelim. O durumda dizimiz
biçimini alır.

a[0]	a[1]	a[2]	a[3]	a[4]
çevik	tavşan	karabaşın	üstünden	atladı

Bunu yapabilmek için, "çevik" sözcüğünü ana bellekte a[0] adresine koymalıyız. Bunu yapınca, öteki sözcükleri birer geri iteceğiz; o zaman, en sondaki "atladı" sözcüğüne array(dizi) de yer kalmaz. O halde array(dizi)'in terim sayısını 1 artırmalıyız. Ancak, array(dizi) yaratılırken terim sayısı belirlenir ve o sayı değiştirilemez. O nedenle, bu işlem için ana bellekte şu işlerin yapılması gerekir:

Beş terimli bir b*0…4+ array(dizi)'ini yarat

a*0…3+ array(dizi)'inin terimlerini sırayla b[1…4] alt array(dizi)'i üstüne kopyala b[0+ = "çevik" atamasını yapa*0…3+ array(dizi)'ini sil b array(dizi)inin adını değiştir; a yap

c. Array(dizi)'in ortasına ekleme (sokuşturma):tümcesini tavşan tembel karabaşın üstünden atladı diye değiştirelim. O durumda dizimiz

a[0]	a[1]	a[2]	a[3]	a[4]
tavşan	tembel	karabaşın	üstünden	atladı

biçimini alır. Bunu yapabilmek için, "tembel" sözcüğünü ana bellekte a[1] adresine koymalıyız. Bunu yapınca, öteki sözcükleri birer geri iteceğiz; o zaman en sondaki "atladı" sözcüğüne arrayde yer kalmaz. O halde arrayin terim sayısını 1 artırmalıyız. Ancak, array yaratılırken terim sayısı belirlenir ve o sayı değiştirilemez. O nedenle, bu işlem için ana bellekte şu işlerin yapılması gerekir:

1. Beş terimli bir b*0…4+ array(dizi)'ini yarat
2. b[0] = a[0] atamasını yap
3. a[1…3+ array(dizi)'inin terimlerini sırayla b[2…4] alt array(dizi)'i üstüne kopyala
4. b[1+ = "tembel" atamasını yap
5. a*0…3+ array(dizi)'ini sil
6. b array(dizi)'inin adını değiştir; a yap

Yukarıda söylediklerimizi bir araya getirirsek, (sol <= ind <= sağ) ise yeniTerim değerini a*sol…sağ+ arrayi içinde indisi ind olan konuma yerleştirmek istiyoruz. Tabii, bu işi yaparken, arrayin hiçbir terimini kaybetmeden ve onların konumlarının sırasını koruyarak arrayin uygun başka konumlarına taşımalıyız. Şimdi yukarıda söylediklerimizi yapan bir yalancı kod (pseudo code) yazabiliriz.

1. a[ind… sağ-1] alt arrayi(dizi)'ni a[ind+1…sağ+ al tarray(dizi)'i üstüne kopyala
2. a[ind] = yeniTerim atamasını yap
3. Dur

Bunu java kodları biçiminde yazalım. Object, array(dizi)'in veri tipi olsun. Her veri tipinden olabileceğini biliyoruz. [11]

```
// Bu metot yeniTerimi arrayde istenen konuma yerleştirir
void insert(Object[] a, Object yeniTerim, int ind) {
      if (0 <= ind && ind < a.length)
            for (int i = a.length - 1; i > ind; i--)
                  a[i] = a[i - 1];
      a[ind] = yeniTerim;
}
```

Aşağıdaki java programı "tembel" sözcüğünü arrayin str[1] konumuna sokuşturmaktadır.

```
package insert;

import java.util.Arrays;

public class Insertion {
      String[] str = { "tavşan", "karabaşın", "üstünden", "atladı" };
      String[] yedek = new String[str.length + 1];

      public static void main(String[] args) {
            Insertion ins = new Insertion();
            System.out.println("Sokuşturmadan önce :");
            System.out.print(Arrays.toString(ins.str));

            //str arrayini yedek üzerine kopyalar
            ins.arrayCopy(ins.str, ins.yedek);
```

```
            ins.str = null;          //str arrayini siler
            ins.str = ins.yedek;     //str arrayine yedek arrayi atar.
            ins.insert(ins.str, "tembel", 1); //Sokuşturmayı yapar

            System.out.println("\nSokuşturmadan sonra:");
            System.out.print(Arrays.toString(ins.str));
      }

      // Bu metot yeniTerimi arrayde istenen konuma yerleştirir
      void insert(Object[] a, Object yeniTerim, int ind) {
            if (0 <= ind && ind < a.length)
                  for (int i = a.length - 1; i > ind; i--)
                        a[i] = a[i - 1];
            a[ind] = yeniTerim;
      }

      // Bu metot bir arrayi aynı tipten başka bir arraye kopyalar
      void arrayCopy(String[] str1, String[] str2) {
            if (str1.length <= str2.length)
                  for (int i = 0; i < str1.length; i++)
                        str2[i] = str1[i];
      }
}

/*
Çıktı:
sokuşturmadan önce :
[tavşan, karabaşın, üstünden, atladı]
sokuşturmadan sonra:
[tavşan, tembel, karabaşın, üstünden, atladı]
*/
```

2.3.2 Silme (Deletion) Algoritması

Insertion (sokuşturma) algoritmasının ters işlemidir. Diyelim ki banka gişesi önünde 9 kişi sıraya girmiş bekliyor. O anda bir müşteri kuyruktan ayrılıyor. [12]

a. Ayrılan müşteri kuyruğun sonunda idiyseyse, öndekiler için bir sorun doğmaz, ama gişenin yapacağı işlem sayısı 1 azalır.

b. Ayrılan müşteri kuyruğun en önünde idiyse, kuyruktaki 8 kişinin sırası birer ileri kayacaktır.

c. Ayrılan müşteri kuyrukta ortalarda bir yerde idiyse, o konumdan sonrakilerin hepsinin sırası birer ileriye kayacaktır.

Bu olay, bilgisayar jargonunda Deletion Algorithm (silme algoritması) adıyla adlandırılır ve programlanabilir. Bu kesimde onu inceleyeceğiz. Söz konusu kuyruk, bizim için bir array (dizi) dir. Gösterimlerde basitliği sağlamak için, bir a[] dizisini (array) a[alt…üst] simgesiyle gösterelim. Burada alt dizinin en küçük indisi, üst ise en büyük indisidir. Örneğin, a[0,10] gösterimi

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10]

dizisi yerine kullanılacaktır. a[4,7] gösterimi ise

a[4] a[5] a[6] a[7]

alt dizisi yerine kullanılacaktır. Bir dizi yaratıldığında, her terimi için ana bellekte bir adres ayrılır. Verilen diziden bir terim çıkarmak için, ana bellekte o terime ayrılan yeri yok etmek ya da o yere hemen arkasındaki öğeyi yerleştirmek gerekir. Çıkarılan terim dizinin başında, sonunda ya da ortalarda bir yerde olabilir. Her durumda, çıkarılacak terimin durduğu yer (konum) belli ise, indisi biliniyor demektir. Dolayısıyla, dizinin o indisten sonraki indisli terimlerinin hepsinin indisleri birer azaltılmalıdır ki çıkan terim yerinde ve sonrasında boşluklar oluşmasın. Şimdi bunu bir örnekle açıklayalım. [12]

çevik tavşan tembel karabaşın üstünden atladı kaçtı

Tümcesindeki sözcükleri sırayla bir dizinin terimleri olarak yazalım.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
çevik	tavşan	tembel	karabaşın	üstünden	atladı	kaçtı

a. Array(dizi)'in son öğesini silme: tümcesini çevik tavşan tembel karabaşın üstünden atladı diye değiştirelim. Bunun kısa bir yolu a[6] = null atamasıdır. Bu atamadan sonra, array(dizi)

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
çevik	tavşan	tembel	karabaşın	üstünden	atladı	*null*

biçimini alır. Bu durumda ana bellekte a*6+ adresi yerini korumayı sürdürür. Çünkü array(dizi) yaratılırken terim sayısı belirlenir ve o sayı değiştirilemez. Eğer a[6] bellek adresini bellekten silip, array(dizi)'i 6 öğeli bir array(dizi) haline getirmek istiyorsak, ana bellekte şu işlerin yapılması gerekir:

i) 6 öğeli bir yedek b array(dizi)'i yarat

ii) a[0…6] alt array(dizi)'inin terimlerini sırayla b[0…6] array(dizi)'i üstüne kopyala

iii) a = null atamasını yap

iv) a = b atamasını yap

b. Array(dizi)'in ilk öğesini silme: tümcesini tavşan tembel karabaşın üstünden atladı kaçtı diye değiştirelim. Kısa bir yol, a[0] = null ataması ile array(dizi)'i

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
null	tavşan	tembel	karabaşın	üstünden	atladı	kaçtı

biçimine sokmaktır. Bu durumda ana bellekte a[0] adresi yerini korumayı sürdürür. Çünkü array(dizi) yaratılırken terim sayısı belirlenir ve o sayı değiştirilemez. Eğer a[0] bellek adresinin boş kalmasını istemiyorsak, ondan sonra gelen bütün terimlerin içlerini birer göz sola kaydırıp, en sondaki gözü silmeliyiz. Bunun için, ana bellekte şu işlerin yapılması gerekir:

i) 6 öğeli bir yedek b array(dizi)'i yarat

ii) a[1…6] alt array(dizi)'inin terimlerini sırayla b[0…5] array(dizi)'i üstüne kopyala

iii) a = null atamasını yap

iv) a = b atamasını yap

c. Array(dizi)'in ortasından bir öğe silme: tümcesini çevik tavşan karabaşın üstünden atladı kaçtı diye değiştirelim. Bunu yapmanın kısa bir yolu a[2] = null atamasını yapmaktır. Bu atama, array(dizi)'i

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
çevik	tavşan	*null*	karabaşın	üstünden	atladı	kaçtı

biçimine sokar. Bu durumda ana bellekte a[2] adresi yerini korumayı sürdürür. Çünkü, array(dizi) yaratılırken terim sayısı belirlenir ve o sayı değiştirilemez. Eğer a[2] bellek adresinin boş kalmasını istemiyorsak, ondan sonra gelen bütün terimlerin içlerini birer göz sola kaydırıp, en sondaki gözü silmeliyiz. Bunun için, ana bellekte şu işlerin yapılması gerekir:

i) 6 öğeli bir yedek b array(dizi)'i yarat

ii) a[0…1] alt array(dizi)'inin terimlerini sırayla b[0…1] array(dizi)'i üstüne kopyala

iii) a[3…6] alt array(dizi)'inin terimlerini sırayla b[2…5] array(dizi)'i üstüne kopyala

iv) a = null atamasını yap

v) a = b atamasını yap

Yukarıda söylediklerimizi bir araya getirirsek, (sol <= ind <= sağ) ise, indisi ind olan terimi silmek için yapacağımız işleri şöyle özetleyebiliriz:

i) a.length-1 uzunluğunda bir yedek b array(dizi)'i yarat

ii) a[0…ind-1] alt array(dizi)'inin terimlerini sırayla b[0…ind-1] array(dizi)'i üstüne kopyala

iii) a[ind+1…sağ] alt array(dizi)'inin terimlerini sırayla b[ind…sağ] array(diz)'i üstüne kopyala

iv) a = null atamasını yap

v) a = b atamasını yap

Bunu java kodları biçiminde yazalım. Object, arrayin veri tipi olsun. Her veri tipinden olabileceğini biliyoruz. Bu metot, arrayin ind+1 indisli teriminden sonraki bütün terimleri birer öne taşır ve arrayin son öğesini null yapar. [12]

```
static void sil(Object[] a, int ind) {
            for (int i = ind; i < a.length-1 ; i++)
                  a[i] = a[i + 1];
      a[a.length-1] = null;
}
```

Aşağıdaki java programı "tembel" sözcüğünü arrayin str[1] konumuna sokuşturmaktadır.

```
//package silme;

import java.util.Arrays;

public class Deletion {
      String[] str = { "çevik","tavşan","tembel", "karabaşın", "üstünden",
"atladı","kaçtı" };
      String[] yedek = new String[str.length - 1];

      public static void main(String[] args) {
            Deletion ins = new Deletion();
            System.out.println("Silmeden önce :");
            System.out.print(Arrays.toString(ins.str));
            ins.sil(ins.str, 2);  //indisi 2 olan terimi siler
            //str arrayini yedek üzerine kopyalar
            ins.arrayCopy(ins.str, ins.yedek);
            ins.str = null;       //str arrayini siler
            ins.str = ins.yedek;  //str arrayine yedek arrayi atar.


            System.out.println("\nSildikten sonra:");
            System.out.print(Arrays.toString(ins.str));
      }
```

```
	// ind indisli terimi arrayden siler
	void sil(Object[] a, int ind) {
		if (0 <= ind && ind <= a.length-1)
		for (int i = ind; i < a.length-1 ; i++)
			a[i] = a[i+1];
		a[a.length-1] = null;
}


	// bir arrayi başka bir arraye kopyalar
	void arrayCopy(String[] str1, String[] str2) {
		//if (str1.length <= str2.length)
			for (int i = 0; i < str2.length; i++)
				str2[i] = str1[i];
	}
}

/*
Çıktı:
Silmeden önce :
[çevik, tavşan, tembel, karabaşın, üstünden, atladı, kaçtı]
Sildikten sonra:
[çevik, tavşan, karabaşın, üstünden, atladı, kaçtı]
*/
```

3. UYGULAMA

Beceri bazlı yönlendirme, yönteminde müşteri temsilcileri becerilerine göre ve yapmaya yetkili oldukları işlere (muhasebe, hisse senetleri, halka arz, satış, destek vb) göre gruplara ayrılır. Arayan müşterinin yaptığı tercihe göre çağrı uygun becerilere sahip müşteri temsilcilerinin kuyruğuna yönlendirilir.

Ayrıca müşterinin veri tabanı kayıtlarında bulunan belli bilgilerine göre çağrı yönlendirmesi yapılır. Mesela arayan müşterinin hesabında belli bir tutardan fazla borç varsa bu çağrıyı önce denetim departmanına veya muhasebeye aktar veya müşterinin kredili işlem sözleşmesinin tarihi geçmişse çağrıyı kredili işlem departmanına aktar gibi çağrı aktarım kuralları tanımlanabilir.

Temel hedef müşteri temsilcine bağlanmak isteyen müşterilerin, hem aktarıldıkları kuyruk bazında hem de müşteri segmentleri bazında önceliklendirilerek aktarılmasını sağlamaktır.

3.1 Telefon Bankacılığı

Son yıllarda bilgi teknolojilerindeki gelişmelerin yaşamımızdaki pek çok alanı etkilemesi kaçınılmazdır. Kuşkusuz bu gelişmelerden en fazla etkilenen sektör, bankacılık sektörü olmuştur. Bankalar, bilgi teknolojilerindeki gelişmelere paralel olarak yeniden yapılanmakta, maliyetleri azaltıcı önlemler almakta ve böylece ellerindeki teknolojik alt yapıyı en iyi şekilde kullanarak müşteri tabanını genişletme ve pazarlama faaliyetlerini arttırmaya yönelik projeler oluşturmaya çalışmaktadırlar. [13]

Bankaların teknolojik gelişmelere ayak uydurması, hizmet maliyetlerinde düşüş sağlamanın yanı sıra çok çeşitli ürünlerin müşteriye en hızlı ve etkin biçimde ulaşmasını sağlayacaktır.[13]

Son yarım yüzyıldır elektronik ticaretin gelişmesiyle bankacılık sektöründe, toplumun gereksinimleri doğrultusunda önemli değişiklikler yaşanmaya başlanmıştır. Özellikle küreselleşmeyle birlikte bankalar, yoğun rekabet ortamından etkilenmiş, müşterilerinin gereksinim ve istekleri doğrultusunda daha nitelikli hizmetler

sunabilme yarışı içine girmişlerdir. Bu durum, hizmet sunumunda klasik şube bankacılığının yanında alternatif dağıtım kanallarının kullanımını da hızlandırmıştır.

Türk bankacılık sektörünün alternatif dağıtım kanalları uygulamaları açısından gelişiminde özellikle 1994 yılı sonrasında müşteriye yönelik olarak yapılanmaya giden ve müşteri bölümlendirmesini ön plana çıkaran bir profil görülmektedir. [13]

Satış ekiplerinin ve şubelerin üzerinde bulunan işlemsel yükü kaldırmayı ve işlem maliyetlerini düşürmeyi hedefleyen bankalar öncelikli olarak Internet, telefon ve bankamatik (ATM) gibi alternatif kanalları dağınık yapıdan merkezi yapıya çevirerek aktif bir şekilde kullanmaya başlamışlardır. [13]

Çoklu kanal stratejisi olarak ortaya çıkan bu uygulama tamamen şube ağırlıklı satış ve hizmet üzerine odaklanan bankacılık sektörünü, hizmet standartları, tek bir veri tabanı üzerinden merkezi olarak işleyen bir kanal yönetimi stratejisi oluşturmaya yöneltmektedir. [13]

Telefon Bankacılığı, müşterinin kendisine verilen şifre ile kendi hesap bilgilerine telefon üzerinden 444 *** arayarak sesli yanıt sistemi veya Müşteri temsilcisi aracılığıyla, telefon tuşlarını kullanarak kendisini yönlendiren sesli uyarılar ile Talep ettiği işlemi yerine getirmesi olayıdır. [13]

3.1.1 Telefon Bankacılığı Üzerinden Yapılabilecek İşlemler

Doğrudan müşteri temsilcisi ile görüşerek; Kredi kartı işlemlerinizden hesap hareketlerinize, EFT ve havaleden yatırım işlemlerine, döviz alım ve satımına kadar tüm bankacılık işlemlerinizi yapabilirsiniz. [14]

Sesli yanıt sistemi aracılığıyla; Hesaplarınızla ilgili bilgi alabilir, kredi kartı işlemlerinizi ve TL hesaplarınız arasında para aktarımlarını gerçekleştirebilirsiniz.[14]

Ayrıca döviz, borsa, repo ve diğer yatırım araçlarına ilişkin bilgilere günün 24 saati ulaşabilirsiniz. Ayrıca, Ses Tanıma sistemi ile Gerçek zamanlı hisse senedi fiyatları - Gerçek zamanlı döviz kuru - Yatırım fonu fiyatları - Repo faiz oranları - Yerli ve yabancı para cinsinden vadeli mevduat faiz oranları bilgilerine ulaşabilirsiniz.[15]

3.1.2 Telefon Bankacılığının Faydaları

Bankalara[15];

- Düşük İşlem Maliyeti
- Müşterilere 7 /24 Hizmet Sunabilme İmkânı
- Personel ve Şube Yükünü Azaltma
- Dünyanın Her yerinden Telefon aracılığıyla Müşterilere Bankacılık Hizmetini Sunabilme İmkânı
- Zaman ve Mekan Sınırlamasının Olmaması.

Müşterilere[15];

- İstenilen Yer ve Zamanda İşlem Yapabilme İmkânı
- Zaman Tasarrufu
- Yapılan İşlem için düşük komisyon ödenmesi veya hiç ödenmemesi

3.1.3 Telefon Bankacılığında Bankaların Hedefleri

Telefon Bankacılığından yapılacak işlemlerde alınacak komisyon ve ücretler ile banka gelirlerinin artırılması. [15]

Şubelere gelen "Bilgi Sorgulama" telefonlarının ADK kanallarına kaydırılması ve şube iş yükünün azaltılması

Bankacılık işlemlerinin süratli ve yüksek güvenlik içerisinde yapabilmeleri ve aynı zamanda internet bankacılığına alternatif olarak (kullanıcının yaşayabileceği teknik durumlar, İnternet bağlantılarında yaşanması muhtemel sorunlar ve kullanıcıların yurtdışında ya da seyahatte olması) bankacılık işlemlerinin aksamadan yapılması. [15]

İnternet bankacılığını kullanmak istemeyen ya da sınırlı yetkilerle internet bankacılığı kullanıcısı olmak isteyip işlemlerini telefon bankacılığı aracılığı ile yapmak isteyen müşterilerin bankaya kazandırılması.

Müşterilerin 7 gün 24 saat hizmet alacağı Alternatif Dağıtım Kanalı sayısının arttırılması ve buna paralel istenildiği zaman müşteriye hizmet verilerek müşteri memnuniyeti ve bağlılığının arttırılması

3.1.4 Telefon Bankacılığının Dezavantajları

SAP ve EFMA'nın yaptırdığı araştırma sonucu, bankaların tüm müşterileri ayrım yapmadan; internet, telefon, ATM bankacılığı gibi daha az masraflı self servis bankacılık hizmetlerine yönlendirmesinin, müşterilerin banka şubelerinden uzaklaşmalarına neden olduğunu ortaya koyuyor. Özellikle de üst sınıfta yer alan ve yüz yüze iletişim yapılması gereken müşterilerin banka şubelerine geri çekilmesi ihtiyacı bankacılar tarafından açıkça ifade ediliyor. Araştırmada, müşteriler hakkında çok özel bilgilere sahip olmanın şube çalışmaları için çok önemli olduğu vurgulanıyor. [15]

3.2 Beceri Bazlı Yönlendirme

Beceri bazlı yönlendirme, yönteminde müşteri temsilcileri becerilerine göre ve yapmaya yetkili oldukları işlere (muhasebe, hisse senetleri, halka arz, satış, destek vb) göre gruplara ayrılır. Arayan müşterinin yaptığı tercihe göre çağrı uygun becerilere sahip müşteri temsilcilerinin kuyruğuna yönlendirilir.

Ayrıca müşterinin veri tabanı kayıtlarında bulunan belli bilgilerine göre çağrı yönlendirmesi yapılır. Mesela arayan müşterinin hesabında belli bir tutardan fazla borç varsa bu çağrıyı önce denetim departmanına veya muhasebeye aktar veya müşterinin kredili işlem sözleşmesinin tarihi geçmişse çağrıyı kredili işlem departmanına aktar gibi çağrı aktarım kuralları tanımlanabilir.

Temel hedef müşteri temsilcine bağlanmak isteyen müşterilerin, hem aktarıldıkları kuyruk bazında hem de müşteri segmentleri bazında önceliklendirilerek aktarılmasını sağlamaktır.

3.2.1 Beceri Bazlı Yönlendirme Önemi

Müşterilerin daha hızlı, kolay ve kullanışlı bir sesli yanıt sistemi deneyimi yaşaması hedeflenmektedir.

Bununla birlikte Özel Müşteri kitlesine sunulacak sabit müşteri temsilcisi hizmeti ile bu kitlenin sesli yanıt sistem'ini her aradıklarında mümkün oldukça hep aynı müşteri temsilcisine aktarılması hedeflenmektedir.

Müşterilerin en sık kullandıkları sesli yanıt sistemi menülerini bir araya getirerek kendi kişisel sesli yanıt sistemi menülerini oluşturmaları da sağlanacaktır. Müşteriler segment önceliklerine göre, doğru müşteri temsilcine daha hızlı aktarılmış olacaktır. Müşteri Temsilcilerinin daha verimli kullanılması ve Müşteri Memnuniyetinin artırılması açısından son derece önemlidir.

Çağrı Önceliklendirmesi hem Kuyruk bazında hem de Müşteri Segment'i bazında yapılacaktır. Kuyruk; Sesli yanıt sistemi akışındaki çağrıların müşteri temsilcilerine transfer edildiği her bir grubu ifade etmektedir. Müşteri Segment'i ise bankacılıkta müşterilerinin SBUXXX şeklinde kodlanmış olduğu müşteri gruplarını temsil eder.

Sanal Bekleme, müşteri kaynaklı farklı beceri yetkinliğine sahip müşteri temsilcisine gelmiş çağrının müşteri temsilcisi tarafından bekletilip doğru beceri yetkinliğine aktarılması süresince beklemesine denilmektedir.

3.2.2 Beceri Bazlı Yönlendirme Algoritması

Çağrı Yönlendirme Stratejisi'nin yazılabilmesi için aşağıdaki aksiyonların tamamlanması gerekmektedir.

1) Sesli yanıt sistemi akışının (ağaç + kurallar) tamamlanması,

2) Öncelik Matrisinin çıkarılması [Tablo-1]

3) Beceri İsimleri ve Seviye'lerinin tespit edilmesi

4) Yönlendirmede uygulanacak

a) Beceri İsmi+ Seviye + Zaman aşımı sırasının çıkarılması

b) Stratejinin rutin çalışmasının değiştiği Tarih + Saat istisna'ların tespit edilmesi.

5) Sanal bekleme için Yönlendirme Kurallarının tespit edilmesi.

Çağrı Önceliklendirmesi hem Kuyruk bazında hem de Müşteri sekmendi bazında yapılacaktır. Kuyruk; Sesli yanıt sistemi akışındaki çağrıların müşteri temsilcilerine transfer edildiği her bir grubu ifade etmektedir. Müşteri sekmendi ise bankacılıkta müşterilerinin SBUXXX şeklinde kodlanmış olduğu müşteri gruplarını

temsil eder. Her bir Segment-Kuyruk çifti için, ilk öncelik, Öncelik Artış Periyodu ve Öncelik Artış Değeri belirlenecektir.

Müşteri Segmenti bazında önceliklendirme ile aynı kuyruğa giren, farklı segmentteki müşterilerin, farklı önceliklerle hizmet alınması hedeflenmektedir. Hisse Senedi, İnternet Destek, Üye İşyeri gibi uzmanlaşmış küçük bir grup tarafından verilen hizmetlerde tüm müşteriler, segment ayrımı yapılmaksızın, girdikleri kuyruklarda aynı ilk öncelik, öncelik artış değeri ve öncelik artış periyodu ile hizmet alacaklardır.

Önceliklendirmede, her bir Kuyruk için, çağrının sırasıyla hangi "Beceri+ Beceri Seviyesinde "Ne Kadar Süre (Saniye)" uygun bir müşteri temsilcisi bekleyeceği bilgisinin hazırlanması gerekmektedir. Aynı beceri seviyesindeki müşteri temsilcileri arasında adil çağrı dağıtımını sağlayabilmek adına, çağrı "Hazır" Konumda en uzun bekleyen müşteri temsilcisine transfer edilir.

Sanal Bekleme çağrıları, transfer edildiği kuyruktaki en öncelikli çağrı olmak zorundadır. Bu amaçla, ilk öncelik ve artış değeri diğer Segment'lerden çok yüksek, artış periyodu ise çok düşük olmalıdır.

Müşteri Segmentleri:

A Tipi Müşteri: Bankadaki mevduatı 500 ile 100.999 TL arasında olan müşteri tipleri

B Tipi Müşteri: Bankadaki mevduatı 101.000 ile 999.999 TL arasında olan müşteri tipleri

C Tipi Müşteri: Bankadaki mevduatı 3.000.000 TL den başlayan müşteri tipleri

D Tipi Müşteri: Bankadaki mevduatı 2.000.000 TL ile 2.999.999 TL arasında olan müşteri tipleri.

E Tipi Müşteri: Bankadaki mevduatı 1.000.000 TL ile 1.999.999 TL arasında olan müşteri tipleri.

P Tipi Müşteri: P Tipi müşteriler iç müşterileri kapsamaktadır. Firmanın personellerinden oluşmaktadır.

S Tipi Müşteri: S tipi müşteriler Çağrı merkezini aramış ve belli bir kuyruğa dahil olup müşteri temsilcisine bağlandıktan sonra hatalı kuyruğa geldiği müşteri temsilcisi tarafından fark edilip ilgili kuyruğa aktarılan müşteri tipleridir.

C,D,E Tipi Müşteriler Özel müşteri statüsündeki müşterilerdir. Ancak telefon bankacılığını kullanma ve gelir düzeylerindeki kriterlere göre müşteriler kendi içlerinde dinamik olarak segmentleri değişmektedir. Örn. X müşterisi E tipi özel müşteri iken son 3 ay içerisinde gelir düzeyinde artış kaydetmiş ve ayrıca telefon bankacılığını oldukça sık kullanmış ise bu müşteri D segmentine dinamik olarak dahil edilecektir.

Öncelik matrisi bir defa hazırlanacak olup bu matristeki müşteri tipleri dinamik olacaktır.

Müşteri son 3 ay içersinde telefon bankacılığını 10+ kullanmış ise bu müşteri hangi segmentte yer alıyor ise o segmente ait giriş önceliği değerine 100 puan eklenecek ve toplam öncelik değeri hesaplanırken bu durum göz önünde bulundurulacaktır.

Örn. D tipi müşteri giriş önceliği 130 ancak bu müşteri son üç ay içerisinde 11 kez telefon bankacılığını kullanmış bu durumda bu müşterinin giriş önceliği 130+100=230 olacaktır. Bu durumdaki müşteriler kuyrukta daha az zaman kaybedip daha hızlı bir şekilde müşteri temsilcisine bağlanarak işlemlerini gerçekleştirebileceklerdir.

(S Tipi müşteriler)Müşteri herhangi bir kuyrukta bekledikten sonra müşteri temsilcisine bağlandığında müşteri temsilcisi tarafından yanlış kuyruğa dahil olduğu anlaşılmış ve ilgili kuyruğa aktarılmış müşterilerin daha önce bir kuyrukta bekledikleri ve aktarılacakları kuyrukta ta bekleme durumlarının müşteri memnuniyetsizliğine yol açabileceği durumu göz önüne alınıp bu tip müşterilerin toplam değerlerini +5000 puan ekleyip aktarıldıkları kuyrukta ilk önceliği almaları sağlanacaktır.

Örn. X müşteri Y işlemini yapmak isterken Z işleminin kuyruğuna dahil olmuş olsun ve kuyrukta t zamanı kadar bekledikten sonra müşteri temsilcisine bağlanmış olsun (müşteri temsilcisine bağlanana kadar öncelik matrisi ve toplam değer hesaplamaları aynen uygulanacaktır.) bağlanılan müşteri temsilcisi bu

müşterinin aslında Z işlemi değil de Y işlemini yapmak istediğini anlayıp bu müşteriyi Y kuyruğuna aktardığında bu müşterinin İlk öncelik değeri 5000 olarak atanacaktır. Bu şekilde müşteri aktarıldığı kuyrukta müşteri temsilcilerinden herhangi biri boşa çıktığı anda bekletilmeden müşteri temsilcisine aktarılacak ve işlemini gerçekleştirecektir.

Sanal bekleme çağrıları için ilk öncelik, artış değeri ve periyot bilgileri, her kuyruk için öncelik matrisine eklenmelidir.

c: Artış Değeri

k: Artış Sayısı

p: Periyot(Süre) Değeri

z: İlk öncelik

t:bekleme süresi

t=n*p;

T: toplam puan

L: Son üç ay içerisindeki telefon bankacılığı kullanımı (sadece işlem gerçekleştirme durumları)

Müşteri Sekmendi = A,B,C,D,E,P ise;

L<10; ise

$$T= \sum_{p}^{n} (c*p) + z$$

L≥10; ise

x=z+100

$$T= \sum_{p}^{n} (c*p) + x$$

Müşteri sekmendi = S ise;

Z=5000;

$$T= \sum_{p}^{n} (c*p) + z$$

Örneğin, Bankacılık Kuyruğunda en yüksek ilk önceliğe sahip segmentten (Zengin/Varlıklı) bir müşterinin kuyrukta 1 saat gibi ekstrem bir süre beklediğini varsayalım. 1 saat sonunda son önceliği 680 ((Artış Değeri (10) * Artış Sayısı (60 x 10=600)) + ilk öncelik(80)) olacak ise, Sanal bekleme için 5000 ilk öncelik gibi yüksek bir değer, bu çağrının ilk sırada yanıtlanmasını garantileyecektir.

Örnek: Bankacılık Kuyruğuna aktarılan bir C tipi müşteri ve son üç ayda 12 defa telefon bankacılığını kullanmış. Müşterinin 2dk sonundaki Toplam Öncelik puanı ?

Kuyruk da bekleyen müşterilerin Puanları

	690	700	750	900	1050	1150	1180	1225	1250
10	9	8	7	6	5	4	3	2	1

Bu Müşteri kuyruğa kaçıncı sıradan dahil olacaktır.?

Çözüm:

- Müşteri Tipi C
- L=12
- c= 30
- p= 0,5
- t=n*p ise t=2 ise 2=n*0,5 ise n=4;
- n= 4 (2dk da 4 defa artmıştır)
- z=140
- L≥10 olduğu için z= 140+100 ise z=240

$$T= \sum_{0.5}^{4} (c*p) + z$$ ise;

T={ (c*p) + z }+ (c*p) + z }+{ (c*p) + z }+{ (c*p) + z } ise

T={ (30*0,5) + 240 }+ (30*0,5) + 240 }+{ (30*0,5) + 240 }+ {(30*0,5) + 240 } ise

T= 255+255+255+255 ise

T=1020

690	700	750	900	**1020**	1050	1150	1180	1225	1250
10	9	8	7	6	5	4	3	2	1

Bu müşteri kuyruğa 6.sıradan dahil olacaktır.

Not: Aynı puana sahip olan müşteriler de kuyruk sırası;

- Müşteri tiplerine göre sıralanacaktır. S>C>D>E>P>B>A
- Müşteri tipleri aynı ise kuyruğa ilk gelen öne geçecektir.
- Müşteri tipleri aynı ve kuyruğa da aynı anda dahil olma durumu söz konusu ise rastgele atanacaklardır. Örneğin ikisinin de puanı 360 müşteri tipleri C ve aynı anda kuyruğa gelmiş olsalardı, rastgele biri 6.sıraya diğeri 7.sıraya atanacaktı.

Not: Yeni Sanal bekleme kuyruğu tanımlanması durumunda, bu kuyruğa ait sanal bekleme ilk öncelik, artış değeri ve periyodu bilgileri, yeni kuyruk için belirlenmelidir.

Not: Öncelik Matrisi hazırlanırken sanal beklemeden daha öncelikli kuyruklar olması durumunda sanal bekleme öncelik değerlerinin bu kuyruklardan daha yüksek olmadığından emin olunmalıdır.

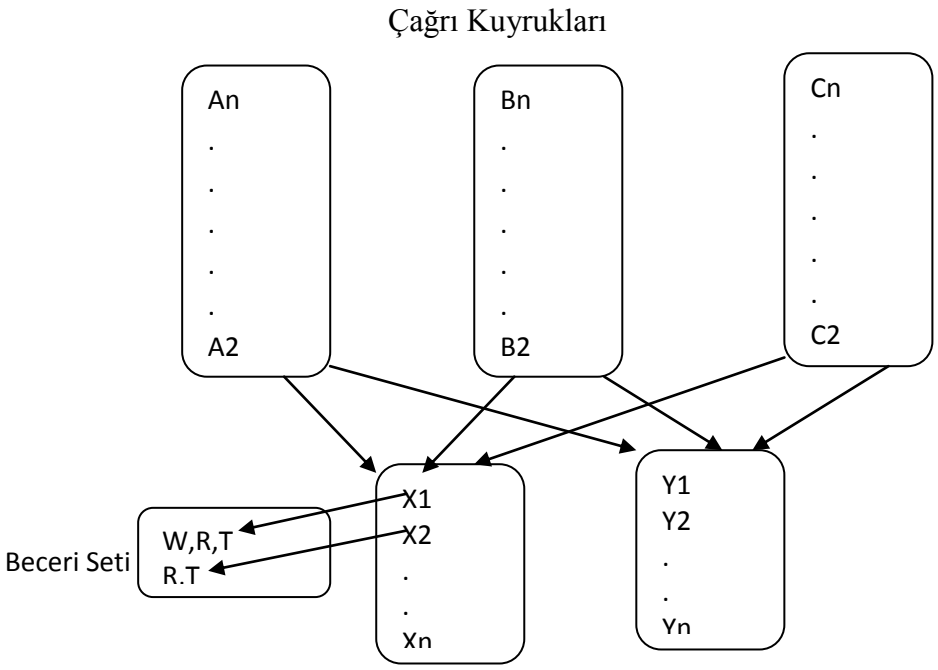
Not: Kayıp Çalıntı kart bildirimleri için gelen çağrılar kuyrukta en az beklemesi gereken acil çağrılar olduğundan bu kuyruğun öncelik matrisindeki değerleri yüksek verilmelidir. Acil çağrı statüsündeki çağrılar için kayıp çalıntıda uygulanan yapı kullanılmalıdır.

			Aktarılan Kuyruk											
			Özel Bankacılık			Bankacılık			Kredi Kartı			Kayıp Çalıntı		
			Giriş Önceliği	Artış Değeri	Periyot (sn)	Giriş Önceliği	Artış Değeri	Periyot (dk)	Giriş Önceliği	Artış Değeri	Periyot (dk)	Giriş Önceliği	Artış Değeri	Periyot (dk)
Segment	A	SBU-410	60	10	1	60	10	2	60	10	1	3000	3000	0,001
	B	SBU-430	90	20	0,5	80	10	1	80	20	0,5	3500	3000	0,001
	C	SBU-300	140	30	0,5	Özel Bankacılığa Transfer edilir (446)			100	30	0,5	3700	3000	0,001
	D	SBU-310	130	25	0,5				110	25	0,5	3500	3000	0,001
	E	SBU-311	125	20	0,5				120	20	0,5	3200	3000	0,001
	P	SBU-799	120	20	0,5	70	10	2	120	20	0,5	3200	3000	0,001
	S	SBU-000	5000	5000	0,01	5000	5000	0,01	5000	5000	0,01	-	-	-

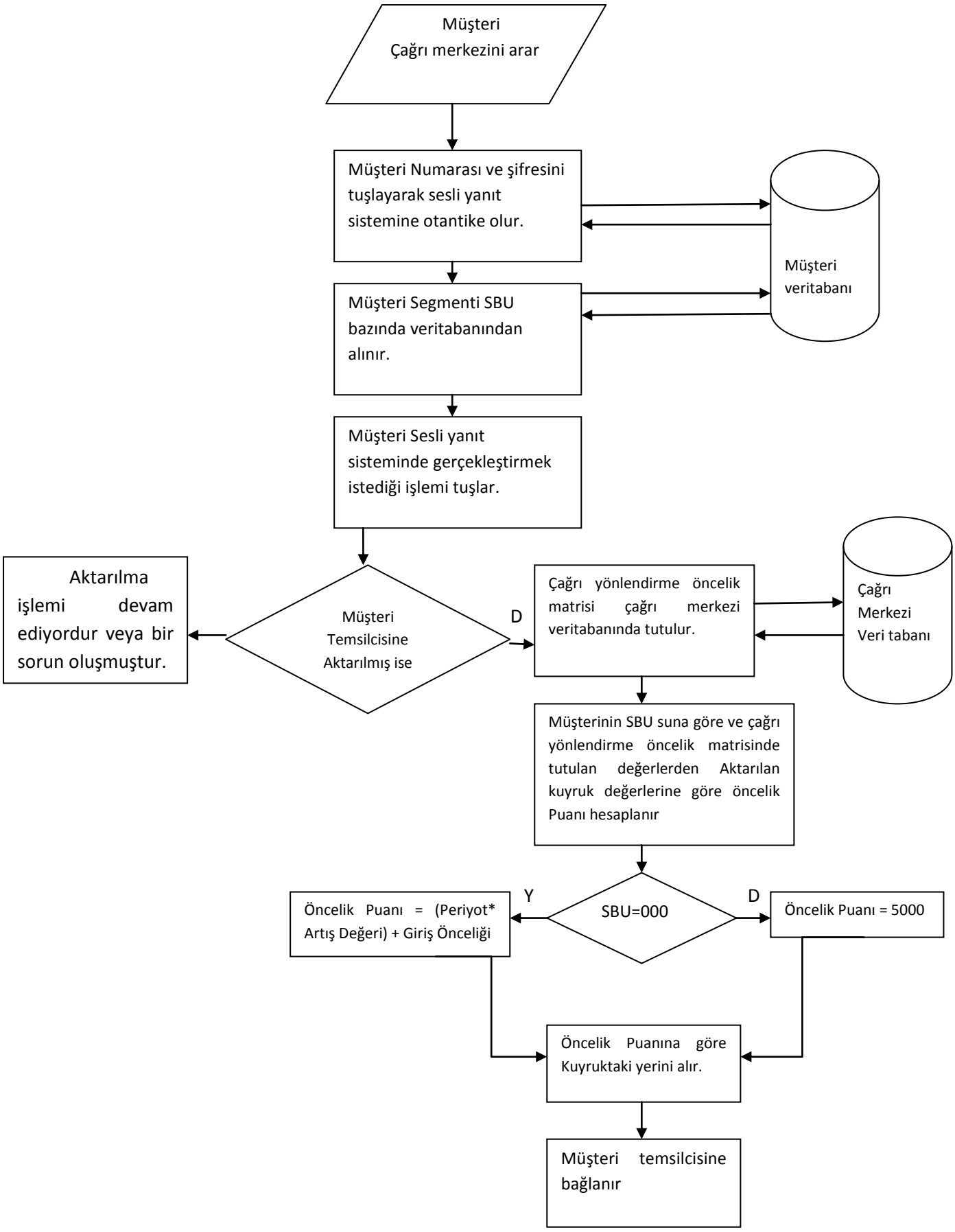
Tablo 1 – Çağrı Yönlendirme Öncelik Matrisi

Segment ve Kuyruk isimleri ile Giriş Önceliği, Artış Değeri ve Periyot bilgileri örnek amaçlı girilmiştir.

3.2.3 Beceri Bazlı Yönlendirme Diyagramı



Şekil-3 Müşteri Temsilcisi Hizmet Ekipleri



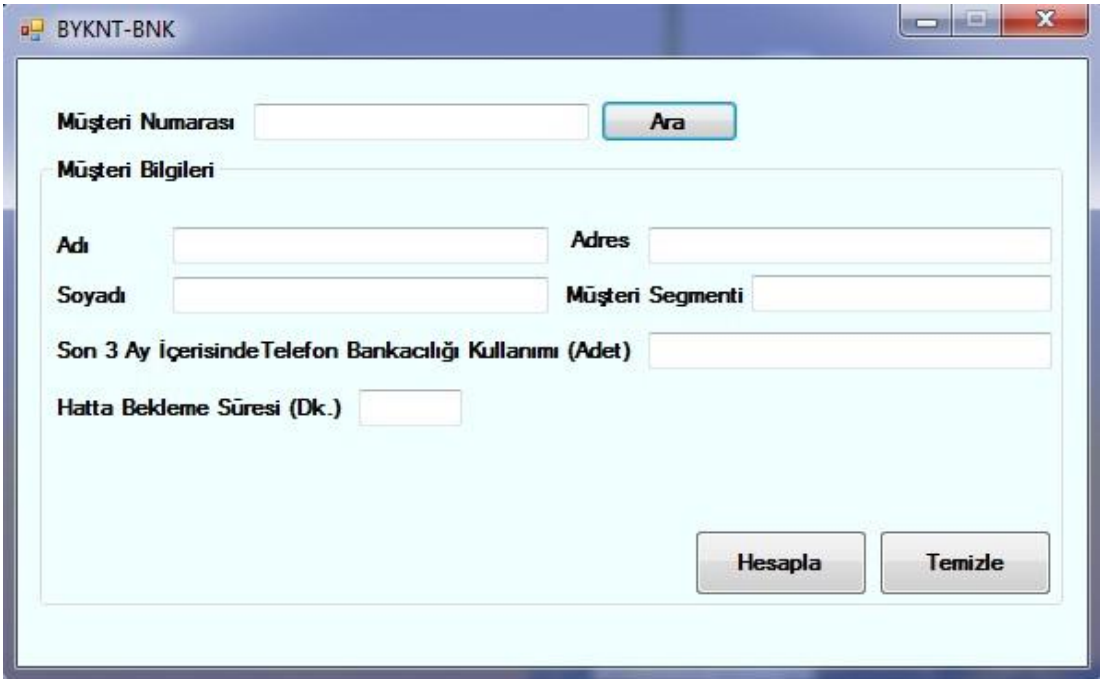
Şekil-4 Beceri Bazlı Yönlendirme Akış Şeması

3.2.4 Beceri Bazlı Yönlendirme Programı

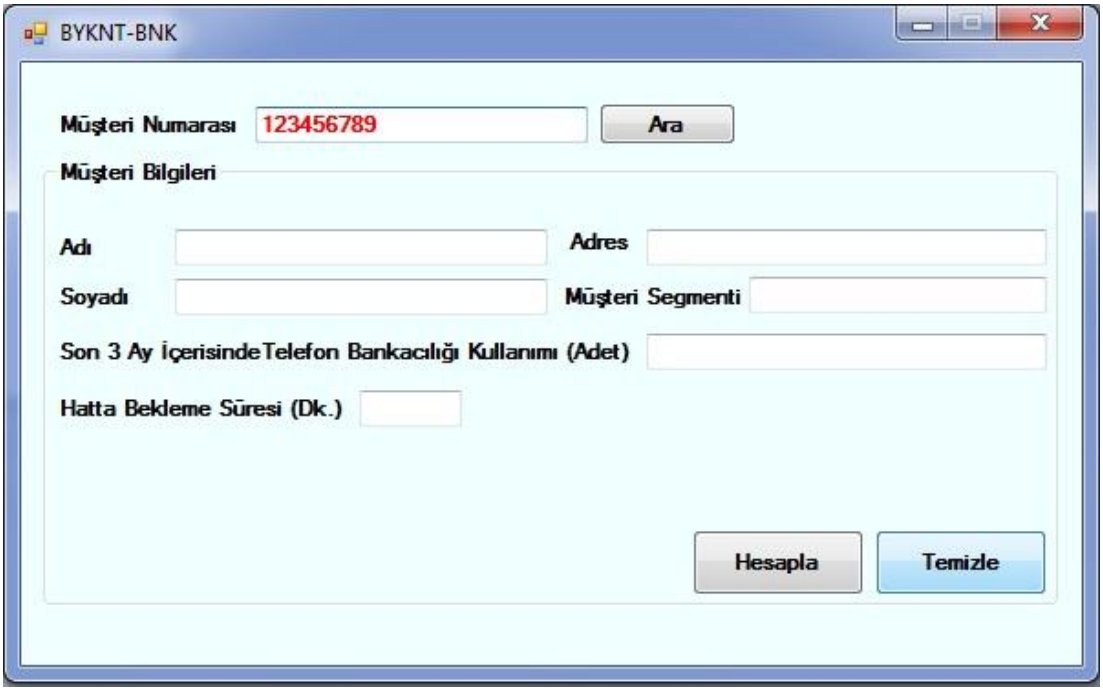
Beceri bazlı yönlendirme uygulamasının kullanımı;

Müşteri numarası alanına müşteri numarası manuel olarak eklenip ara butonuna basıldığında müşteri veri tabanından müşterinin Adı, soyadı, adresi, müşteri sekmendi ve son 3 ay içerisinde telefon bankacılığı kullanımı (adet) bilgileri alınır ve uygulamada ilgili alanlar otomatik olarak doldurulur.

Hatta bekleme süresi manuel olarak girilip hesapla butonuna basıldığında hattaki müşterinin kuyruk puanı ve kuyruk sırası hesaplamaları beceri bazlı uygulama algoritmasına göre otomatik olarak hesaplanır ve uygulama ön yüzünde gösterilir.

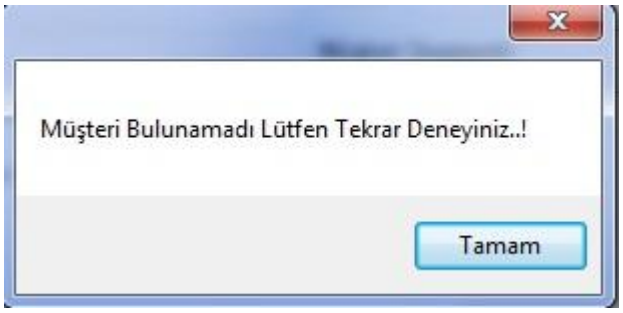


Örneğin müşteri numarası 123456789 olsun bu numarayı uygulamada müşteri numarası alanına girelim ve Ara butonuna basalım,

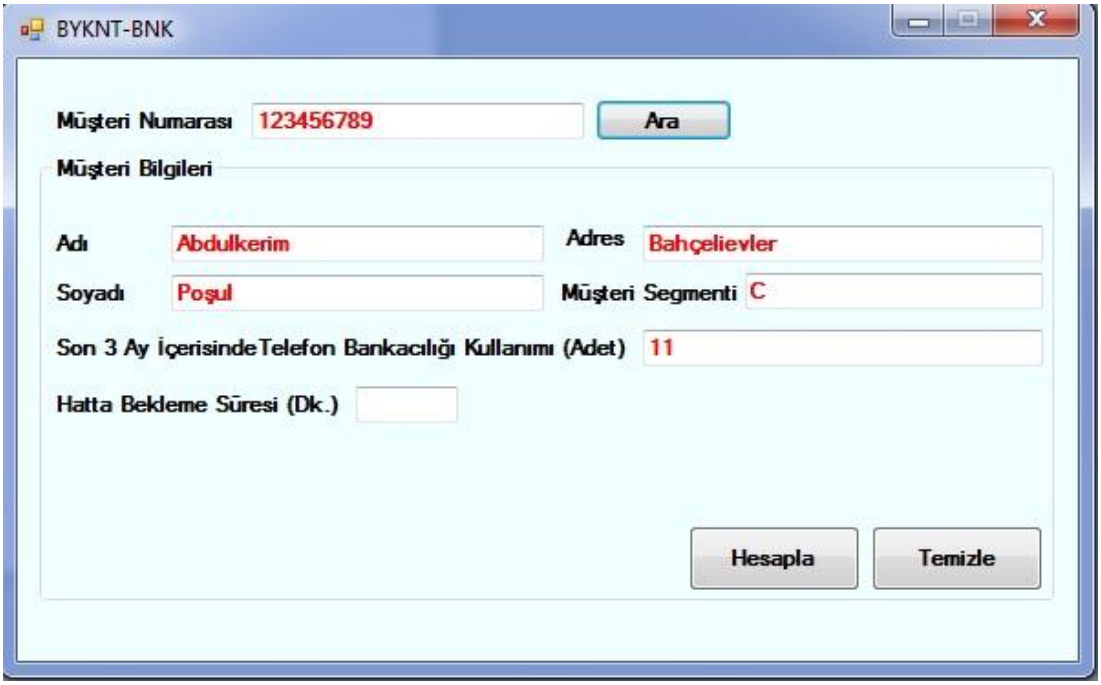


Ara butonuna bastığımızda uygulama müşteri veri tabanına bağlanacak ve girilen numaranın müşteri veri tabanında olup olmadığını kontrol edecektir.

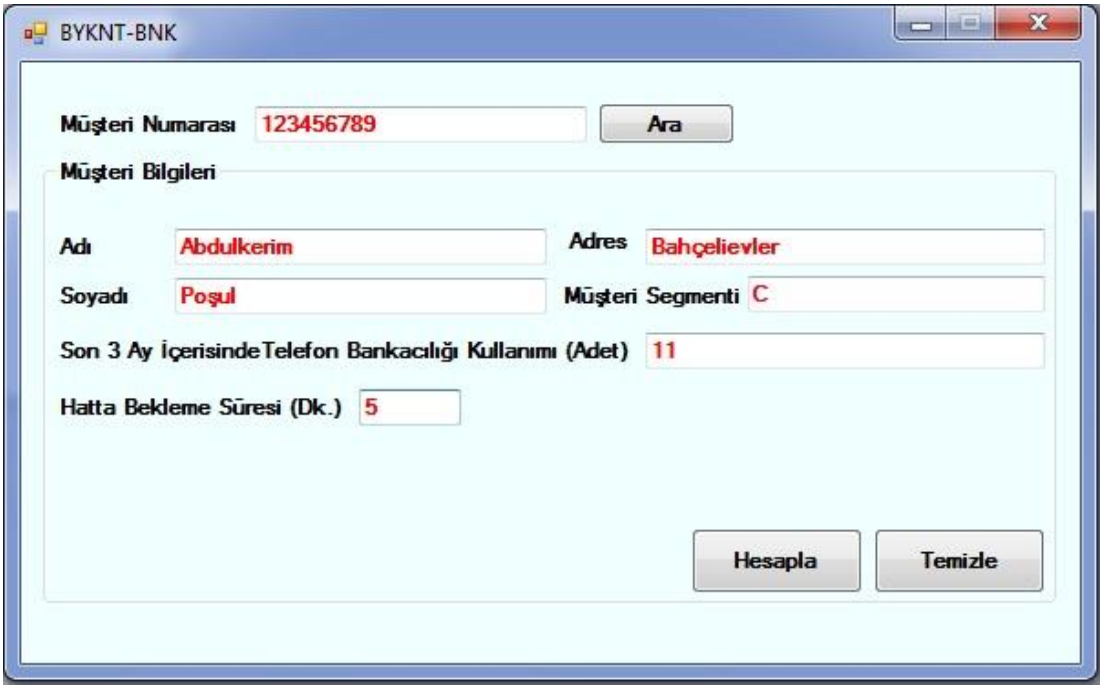
Eğer girilen numara müşteri veri tabanında bulunamazsa "Müşteri Bulunamadı Lütfen Tekrar Deneyiniz" uyarı ekranı çıkacaktır.



Eğer girilen numara müşteri veri tabanında bulunursa uygulamadaki Adı, Soyadı, Adres, Müşteri Sekmendi, Son 3 Ay içerisinde telefon bankacılığı kullanımı (Adet) alanları otomatik olarak doldurulacaktır.



Müşterinin hatta beklediği süreyi, Hatta bekleme süresi (dk) alanına manuel olarak girip Hesapla butonuna bastığımızda uygulama Beceri bazlı yönlendirme Algoritmasını kullanarak müşterinin kuyruk puanını ve kuyruk sırasını hesaplayacak ve ön yüzde yazacaktır




BYKNT-BNK
Müşteri Numarası
123456789
Ara
Müşteri Bilgileri
Adı
Abdulkerim
Adres
Bahçelievler
Soyadı
Poşul
Müşteri Segmenti
C
Son 3 Ay İçerisindeTelefon Bankacılığı Kullanımı (Adet)
11
Hatta Bekleme Süresi (Dk.)
5
Kuyruk Puanı = 2550
Kuyruk Sırası = 4
Hesapla
Temizle

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.Data.Linq;

namespace Tez
{
publicpartialclassForm1 : Form
    {
DataClasses1DataContext db = newDataClasses1DataContext();
int txt_mn;
int c;
int k;
double p;
int z;
double t;
double sum;
int L;
double n;
double T;



public Form1()
        {

            InitializeComponent();
            lbl_1.Visible = false;
            lbl_2.Visible = false;
            lbl_3.Visible = false;
            lbl_4.Visible = false;
        }

privatevoid button2_Click(object sender, EventArgs e)
        {
/*
             c: Artış Değeri
             k: Artış Sayısı
             p: Periyot(Süre) Değeri
             z: İlk öncelik
             t:bekleme süresi
             t=n*p;
             T: toplam puan
             L: Son üç ay içerisindeki telefon bankacılığı
kullanımı ( sadece işlem gerçekleştirme durumları )
             Müşteri Segmenti = A,B,C,D,E,P ise;
             L<10; ise
             T= n∑p (c*p) + z

             L≥10; ise
             x=z+100
             T= n∑p (c*p) + x
             Müşteri Segmenti = S ise;
             Z=5000;
```

```
                    T=n∑p (c*p) + z

                    Örnek: Bankacılık Kuyruğuna aktarılan bir C tipi
müşteri ve son üç ayda 12 defa
                    * telefon bankacılığını kullanmış.
                    * Müşterinin 2dk sonundaki Toplam Öncelik puanı
?
                    * Çözüm:
                       •      Müşteri Tipi C
                       •      L=12
                       •      c=  30
                       •      p= 0,5
                       •      t=n*p ise t=2 ise 2=n*0,5 ise n=4;
                       •      n= 4 (2dk da 4 defa artmıştır)
                       •      z=140
                       •      L≥10 olduğu için z= 140+100 ise z=240

                       T= 4∑0,5 (c*p) + z ise ;
                       T={ (c*p) + z }+ (c*p) + z }+{ (c*p) + z }+{
(c*p) + z } ise
                       T={  (30*0,5)  +  240  }+  (30*0,5)  +  240  }+{
(30*0,5) + 240 }+            { (30*0,5) + 240 } ise
                       T= 255+255+255+255 ise
                       T=1020

                    */

       //Özel Bankacılık Kuyruğu İçin Hesaplanmıştır..
       if (txt_segment.Text == "A")
                   {
                       c = 10;
                       p = 1.0;
                       z = 60;
                       t = Convert.ToInt32(txt_bekleme.Text);
                       n = t / p;
                       sum = 0;
                       L = Convert.ToInt32(txt_sn3.Text);

       if (L < 10)
                       {
       for (int i = 0; i < n; i++)
                           {
                               T = (c * p) + z;
                               sum = sum + T;

                           }
                           lbl_1.Visible = true;
                           lbl_2.Visible = true;
                           lbl_3.Visible = true;
                           lbl_4.Visible = true;
                           lbl_3.Text                                  =
System.Convert.ToString(sum);
       Random numbers = newRandom();
       int index = numbers.Next(1, 15);
                           lbl_4.Text                                  =
System.Convert.ToString(index);
                       }

       elseif (L >= 10)
                       {
       int x = z + 100;
```

```
        for (int i = 0; i < n; i++)
                                {
                                    T = (c * p) + x;
                                    sum = sum + T;

                                }
                                lbl_1.Visible = true;
                                lbl_2.Visible = true;
                                lbl_3.Visible = true;
                                lbl_4.Visible = true;
                                lbl_3.Text                                      =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                                lbl_4.Text                                      =
System.Convert.ToString(index);
                            }
                        }
        elseif (txt_segment.Text == "B")
                        {
                            c = 20;
                            p = 0.5;
                            z = 90;
                            t = Convert.ToInt32(txt_bekleme.Text);
                            n = t / p;
                            sum = 0;
                            L = Convert.ToInt32(txt_sn3.Text);

        if (L < 10)
                            {
        for (int i = 0; i < n; i++)
                                {
                                    T = (c * p) + z;
                                    sum = sum + T;

                                }
                                lbl_1.Visible = true;
                                lbl_2.Visible = true;
                                lbl_3.Visible = true;
                                lbl_4.Visible = true;
                                lbl_3.Text                                      =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                                lbl_4.Text                                      =
System.Convert.ToString(index);
                            }

        elseif (L >= 10)
                            {
        int x = z + 100;
        for (int i = 0; i < n; i++)
                                {
                                    T = (c * p) + x;
                                    sum = sum + T;

                                }
                                lbl_1.Visible = true;
                                lbl_2.Visible = true;
                                lbl_3.Visible = true;
                                lbl_4.Visible = true;
```

```
                            lbl_3.Text                                  =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                  =
System.Convert.ToString(index);
                        }
                    }
        elseif (txt_segment.Text == "C")
                    {
                        c = 30;
                        p = 0.5;
                        z = 140;
                        t = Convert.ToInt32(txt_bekleme.Text);
                        n = t / p;
                        sum = 0;
                        L = Convert.ToInt32(txt_sn3.Text);

        if (L < 10)
                        {
        for (int i = 0; i < n; i++)
                            {
                                T = (c * p) + z;
                                sum = sum + T;

                            }
                            lbl_1.Visible = true;
                            lbl_2.Visible = true;
                            lbl_3.Visible = true;
                            lbl_4.Visible = true;
                            lbl_3.Text                                  =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                  =
System.Convert.ToString(index);
                        }

        elseif (L >= 10)
                        {
        int x = z + 100;
        for (int i = 0; i < n; i++)
                            {
                                T = (c * p) + x;
                                sum = sum + T;

                            }
                            lbl_1.Visible = true;
                            lbl_2.Visible = true;
                            lbl_3.Visible = true;
                            lbl_4.Visible = true;
                            lbl_3.Text                                  =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                  =
System.Convert.ToString(index);
                        }
                    }
        elseif (txt_segment.Text == "D")
                    {
```

```
                    c = 25;
                    p = 0.5;
                    z = 130;
                    t = Convert.ToInt32(txt_bekleme.Text);
                    n = t / p;
                    sum = 0;
                    L = Convert.ToInt32(txt_sn3.Text);

        if (L < 10)
                    {
        for (int i = 0; i < n; i++)
                        {
                            T = (c * p) + z;
                            sum = sum + T;

                        }
                        lbl_1.Visible = true;
                        lbl_2.Visible = true;
                        lbl_3.Visible = true;
                        lbl_4.Visible = true;
                        lbl_3.Text                                      =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                        lbl_4.Text                                      =
System.Convert.ToString(index);
                    }

        elseif (L >= 10)
                    {
        int x = z + 100;
        for (int i = 0; i < n; i++)
                        {
                            T = (c * p) + x;
                            sum = sum + T;

                        }
                        lbl_1.Visible = true;
                        lbl_2.Visible = true;
                        lbl_3.Visible = true;
                        lbl_4.Visible = true;
                        lbl_3.Text                                      =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                        lbl_4.Text                                      =
System.Convert.ToString(index);
                    }
                }
        elseif (txt_segment.Text=="E")
                {
                    c = 20;
                    p = 0.5;
                    z = 125;
                    t = Convert.ToInt32(txt_bekleme.Text);
                    n = t / p;
                    sum = 0;
                    L = Convert.ToInt32(txt_sn3.Text);

        if (L < 10)
                    {
```

```
        for (int i = 0; i < n; i++)
                            {
                                T = (c * p) + z;
                                sum = sum + T;

                            }
                            lbl_1.Visible = true;
                            lbl_2.Visible = true;
                            lbl_3.Visible = true;
                            lbl_4.Visible = true;
                            lbl_3.Text                                          =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                          =
System.Convert.ToString(index);
                        }

        elseif (L >= 10)
                        {
        int x = z + 100;
        for (int i = 0; i < n; i++)
                            {
                                T = (c * p) + x;
                                sum = sum + T;

                            }
                            lbl_1.Visible = true;
                            lbl_2.Visible = true;
                            lbl_3.Visible = true;
                            lbl_4.Visible = true;
                            lbl_3.Text                                          =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                          =
System.Convert.ToString(index);
                        }
                    }
        elseif (txt_segment.Text=="P")
                    {
                        c = 20;
                        p = 0.5;
                        z = 120;
                        t = Convert.ToInt32(txt_bekleme.Text);
                        n = t / p;
                        sum = 0;
                        L = Convert.ToInt32(txt_sn3.Text);

        if (L < 10)
                        {
        for (int i = 0; i < n; i++)
                            {
                                T = (c * p) + z;
                                sum = sum + T;

                            }
                            lbl_1.Visible = true;
                            lbl_2.Visible = true;
                            lbl_3.Visible = true;
                            lbl_4.Visible = true;
```

```
                            lbl_3.Text                                  =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                  =
System.Convert.ToString(index);
                        }

        elseif (L >= 10)
                        {
        int x = z + 100;
        for (int i = 0; i < n; i++)
                            {
                                T = (c * p) + x;
                                sum = sum + T;

                            }
                            lbl_1.Visible = true;
                            lbl_2.Visible = true;
                            lbl_3.Visible = true;
                            lbl_4.Visible = true;
                            lbl_3.Text                                  =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                  =
System.Convert.ToString(index);
                        }
                    }
        elseif (txt_segment.Text=="S")
                    {
                        c = 5000;
                        p = 0.01;
                        z = 5000;
                        t = Convert.ToInt32(txt_bekleme.Text);
                        n = t / p;
                        sum = 0;
                        L = Convert.ToInt32(txt_sn3.Text);

        if (L < 10)
                        {
        for (int i = 0; i < n; i++)
                            {
                                T = (c * p) + z;
                                sum = sum + T;

                            }
                            lbl_1.Visible = true;
                            lbl_2.Visible = true;
                            lbl_3.Visible = true;
                            lbl_4.Visible = true;
                            lbl_3.Text                                  =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                            lbl_4.Text                                  =
System.Convert.ToString(index);
                        }

        elseif (L >= 10)
                        {
```

```
        int x = z + 100;
        for (int i = 0; i < n; i++)
                                {
                                    T = (c * p) + x;
                                    sum = sum + T;

                                }
                                lbl_1.Visible = true;
                                lbl_2.Visible = true;
                                lbl_3.Visible = true;
                                lbl_4.Visible = true;
                                lbl_3.Text                                      =
System.Convert.ToString(sum);
        Random numbers = newRandom();
        int index = numbers.Next(1, 15);
                                lbl_4.Text                                      =
System.Convert.ToString(index);
                            }
                        }
        else
        MessageBox.Show("Hesaplanamıyor....");

                }

        privatevoid button1_Click(object sender, EventArgs e)
                {

        if (txt_ara.Text!="")
                        {
                            txt_mn = Convert.ToInt32(txt_ara.Text);
                        }




        must_bilg   blg   =   db.must_bilgs.Where(s   =>   s.Must_no   ==
txt_mn).Take(1).SingleOrDefault();
        if (blg != null)
                        {

                            txt_ad.Text = blg.Adı;
                            txt_soyad.Text = blg.Soyadı;
                            txt_adres.Text = blg.Adres;
                            txt_segment.Text = blg.Must_seg;
                            txt_sn3.Text = Convert.ToString(blg.Son_3);



                        }
        else
                        {
        MessageBox.Show("Müşteri     Bulunamadı     Lütfen     Tekrar
Deneyiniz..!");
                            txt_ara.Text = "";
                        }
                }

        privatevoid btn_temizle_Click(object sender, EventArgs e)
                {
                    txt_ara.Text = "";
                    txt_ad.Text = "";
```

```
                txt_sn3.Text = "";
                txt_soyad.Text = "";
                txt_adres.Text = "";
                txt_bekleme.Text = "";
                txt_segment.Text = "";
                lbl_4.Text = "";
                lbl_3.Text = "";
                lbl_1.Visible = false;
                lbl_2.Visible = false;
                lbl_3.Visible = false;
                lbl_4.Visible = false;
            }



        }
    }
```

4. SONUÇ

Bu tezde telefon bankacılık sektörü başta olmak üzere çağrı merkezlerinin daha efektif kullanılması ve müşterilerin işlerini daha az zamanda gerçekleştirmesini sağlamak amacı ile beceri bazlı yönlendirme algoritması üzerine çalışmalar yapılmıştır.

Bankacılık sektöründe telefon bankacılığının önemi oldukça büyüktür. Telefon bankacılığı ile daha ucuz maliyetle daha yüksek kar elde etme şansı diğer kanallara göre daha fazla. Ayrıca bankacılık sektöründe müşteri sadakatinin oldukça az olduğu göz önünde bulundurulursa müşteriye istediği her an her yerden işlemini gerçekleştirme fırsatı vererek bununla birlikte aradığında bankasının onu tanıması ve onun sık kullandığı menülerle onu karşılaması ile birlikte müşterinin çok fazla vakit harcamadan menüler arasında boğulmadan işlemini hızlı ve doğru bir şekilde gerçekleştirebilmesini sağlayarak bu sadakat sağlanmış olur.

Beceri bazlı yönlendirmenin yanı sıra sesli yanıt sistemi müşterilere özel menü tasarımı örn. Her müşteri en sık kullandığı menüleri sıralayıp çağrı merkezini aradığında o menüler ile karşılanması veya kuyruğa dahil olduktan sonra telefonu kapatsa veya bir şekilde hat kesilse dahi kuyruktaki sırası kaybolmadan sırası geldiğin müşteri temsilcisi tarafından aranıp işlemini gerçekleştirebilmesi gibi konularda bu çalışmanın üzerinden yürütülebilir.

KAYNAKLAR

[1] http://www.yorumkat.com/programlama/98865-algoritma-nedir.html

[2] http://www.dahiweb.com/algoritma-nedir

[3] http://www.elektrik.gen.tr/icerik/evrimsel-algoritmalar

[4] http://www.bilgisayarkavramlari.com/2011/05/17/evrimsel-algoritmalar-evolutionary-algorithms/

[5] http://www.yapay-zeka.org/modules/wiwimod/index.php?page=GA

[6] http://ab.org.tr/ab06/bildiri/132.pdf - Kripto

[7] http://tr.wikipedia.org/wiki/K%C3%B6k_bulma_algoritmas%C4%B1 kök

[8] http://www.google.com.tr/url?sa=t&rct=j&q=optimizasyon%20algoritmalar%C4%B1&source=web&cd=3&sqi=2&ved=0CDAQFjAC&url=http%3A%2F%2Fakademik.maltepe.edu.tr%2F~kadirerdem%2FAlgoritmalar%2FKARINCA%2520KOLON%25DDS%25DD%2520OPT%25DDM%25DDZASYON%2520ALGOR%25DDTMALARIi.doc&ei=BB29TqnnNsrc4QSvyJzEBA&usg=AFQjCNHao4M8HASXJGavOg6YEu87USirzA optimizasyon

[9] http://e-bergi.com/2008/Mart/Siralama-Algoritmalari sıralama

[10] http://www.gokhanca.com/index.php?option=com_content&task=view&id=43&Itemid=31 –Sıkıştırma

[11] http://translate.google.com.tr/translate?hl=tr&langpair=en%7Ctr&u=http://www2.ensc.sfu.ca/people/faculty/ljilja/cnl/presentations/felix/smc00/tsld015.htm luyruk

http://www.baskent.edu.tr/~tkaracay/etudio/ders/prg/dataStructures/insertion/insertion.pdf Sokuturma

[12] http://www.baskent.edu.tr/~tkaracay/etudio/ders/prg/dataStructures/deletion/deletion.pdf deletion

[13] Rizal Ahmad, Francis Buttle, (2002) "Retaining telephone banking customers at Frontier Bank", International Journal of Bank Marketing, Vol. 20 Iss: 1, pp.5 – 16

[14] The adoption of virtual banking: an empirical study Shaoyi Liao!,*, Yuan Pu Shao!, Huaiqing Wang!, Ada Chen" !Department of Information Systems, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong"Hang Seng Bank, Hong Kong. International Journal of Information Management 19 (1999) 63Ð74

[15] Aref A. Al-Ashban, Mohammed A. Burney, (2001) "Customer adoption of tele-banking technology:the case of Saudi Arabia", International Journal of Bank Marketing, Vol. 19 Iss: 5, pp.191 – 201

[16] Workshop on Interactive Voice Technology For Telecommunications Applications, 26-27 Sep. 1994. Damhuis et al., ("A Multimodal Consumer Information Server With IVR Menu"Proceedings. Second IEEE Workshop onInteractive Voice Technology for Telecommunications Applications, 26-27 Sep. 1994, Kyoto, Japan pp. 73-76.

[17] IBM Technical Disclosure Bulletin, vol. 33, No. 11, Apr. 1991, pp. 368-371.

ÖZGEÇMİŞ

16.11.1986 tarihinde İSTANBUL' un Bakırköy ilçesinde doğdum. İlk öğrenimimi Kocasinan ilköğretim okulunda tamamladım. 2000 yılında başladığım Yenibosna lisesinden sayısal bölümünden 2003 yılında mezun oldum. 2004 yılında Uluslararası Kıbrıs Üniversitesi Bilişim Sistemleri Mühendisliği İngilizce Bölümünü kazandım. 2009 yılında bu bölümden mezun oldum. Bir buçuk yıl Grimed Sağlık Hizm. Firmasında Yazılım mühendisi olarak çalıştıktan sonra 2011 yılında Yapıkredi Bankası Bilgi Teknolojileri, Alternatif Dağıtım Kanalları Departmanında Uzman İş Analisti olarak göreve başladım ve halen aynı kurumda çalışmaktayım ve İstanbul da ikametgah etmekteyim

Abdulkerim POŞUL