T.C.
İSTANBUL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DOKTORA TEZİ

HAM VERİLERİN GENETİK ALGORİTMALARLA İLİŞKİSEL VERİTABANLARINA DÖNÜŞTÜRÜLMESİ ve BİR UYGULAMA

Emre AKADAL

Enformatik Anabilim Dalı

Enformatik Programı

DANIŞMAN
Doç. Dr. Mehmet Hakan SATMAN

Mayıs, 2017

İSTANBUL

Bu çalışma 17.05.2017 tarihinde aşağıdaki jüri tarafından Enformatik Anabilim Dalı, Enformatik Programında Doktora tezi olarak kabul edilmiştir.

Tez Jürisi

Doç. Dr. Mehmet Hakan SATMAN (Danışman)
İstanbul Üniversitesi
İktisat Fakültesi

Prof. Dr. Sevinç GÜLSEÇEN
İstanbul Üniversitesi
Enformatik Bölümü

Prof. Dr. Zuhal TANRIKULU
Boğaziçi Üniversitesi
Yönetim Bilişim Sistemleri Bölümü

Prof. Dr. Mehpare TİMOR
İstanbul Üniversitesi
İşletme Fakültesi

Prof. Dr. Vedat COŞKUN
Işık Üniversitesi
Fen-Edebiyat Fakültesi

20.04.2016 tarihli resmi gazetede yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince; Bu Lisansüstü teze, İstanbul Üniversitesi'nin abonesi olduğu intihal yazılım programı kullanılarak Fen Bilimleri Enstitüsü'nün belirlemiş olduğu ölçütlere uygun rapor alınmıştır.

ÖNSÖZ

1676'da Isaac Newton, "Eğer ileriyi görebilmişsem, devlerin omuzlarında durduğumdandır" cümlesiyle bilimin işleyişinin özetini yapmıştır. Şüphesizdir ki bilim dünyasının geldiği nokta, omuzlarında yükseldiğimiz bilim insanları sayesindedir. Bu tez çalışmasını gerçekleştirirken omuzları üzerinde durduğum tüm bilim insanlarına, bilime gönül verdikleri ve yaptıkları tüm katkılar için teşekkürü bir borç bilirim.

Devlerin omzunda durabilmek büyük bir beceri gerektirmektedir. Bu beceriye sahip olan ve tüm gayretiyle bana öğretebilme çabasını gösteren, tez çalışmam boyunca her başvurduğumda beni doğruya yönlendiren, bilim insanı olmanın ayrıntılarını sabır göstererek öğreten saygıdeğer tez danışmanım Doç. Dr. Mehmet Hakan Satman'a bu doktora tez çalışmasını bilime yaraşır şekilde tamamlayabilmemi sağladığı için büyük minnetle teşekkürlerimi sunarım.

Akademisyenlik hayatına atılmamdaki en büyük payın sahibi, her zaman öğrencilerinin arkasında duran ve her zaman bir hocadan daha fazlası olan Prof. Dr. Sevinç Gülseçen'e bana kattıkları için çok teşekkür ederim.

İyi niyet ve gayretleriyle bana her zaman yardımcı olan hocalarım Prof. Dr. Zuhal Tanrıkulu'na, Yrd. Doç. Dr. Çiğdem Erol'a, Doç. Dr. Seda Tolun'a ve Yrd. Doç. Dr. Zerrin Ayvaz Reis'e; adını sayamadığım tüm hocalarıma ve çalışma arkadaşlarıma teşekkür ederim. Çalışma arkadaşı olmanın ötesinde her zaman yanımda olan ve bana destek veren sevgili arkadaşım Dr. Serra Çelik'e ayrıca teşekkür ederim.

Hayatı öğrenmemi ve ona tutunmamı sağlayan; her zaman ellerini sırtımda hissettiğim canım annem Ayşe Akadal'a, babam Yunus Akadal'a ve kardeşim Erman Akadal'a teşekkürlerimi sunarım. Ayrıca ailemden ayrı tutamadığım dostum Oğuz Saltık'a, her zaman yanımda olduğu ve beni desteklediği için teşekkür ederim.

Elbette bu hayattaki en büyük destekçim, yol arkadaşım ve biricik eşim İnayet Akadal'a bana sunduğu tüm güzellikler ve biricik oğlumuz Hakan Umut Akadal'ı dünyaya getirdiği için gönülden teşekkür ederim.

İnsanlığa faydalı olması dileğiyle.

Mayıs 2017 Emre AKADAL

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

TABLO LİSTESİ

Sayfa No

SİMGE VE KISALTMA LİSTESİ

Simgeler **Açıklama**

⌈ ⌉	: Bir gerçek sayıyı en yakın üst tamsayıya yuvarlama operatörü
N	: Toplam nitelik sayısı
p	: Olasılık (Probability)
v	: Nitelik (Variable)
w	: Ağırlık (Weight)

Kısaltmalar **Açıklama**

1NF	: 1. Normal Form
2NF	: 2. Normal Form
3NF	: 3. Normal Form
4NF	: 4. Normal Form
5NF	: 5. Normal Form
BCNF	: Boyce-Codd Normal Formu
DKNF	: Alan-Anahtar Normal Formu (Domain-Key Normal Form)
ECBS	: European Committee for Banking Standards
ER	: Varlık-İlişki (Entity-Relationship)
FK	: İkincil Anahtar (Foreign Key)
GA	: Genetik Algoritmalar (Genetic Algorithms)
HVS	: Ham verideki sütun (nitelik) sayısı
IBAN	: Uluslararası Banka Hesap Numarası (International Bank Account Number)
MAD	: Ortalama Mutlak Sapma (Median Absolute Deviation)
maks	: En çok (maksimum)
min	: En az (minimum)
MOGA	: Multi-Objective Genetic Algorithm
NPGA	: Niched-Pareto Genetic Algorithm
NSGA	: Non-dominated Sorting Algorithm
NF	: Normal Form

PK : Birincil Anahtar (Primary Key)

SPEA : Strength Pareto Evolutionary Algorithm

SQL : Structured Query Language

VEGA : Vector Evaluated Genetic Algorithm

VYS : Veritabanı Yönetim Sistemi

ÖZET

DOKTORA TEZİ

HAM VERİLERİN GENETİK ALGORİTMALARLA İLİŞKİSEL VERİTABANLARINA DÖNÜŞTÜRÜLMESİ ve BİR UYGULAMA

Emre AKADAL

İstanbul Üniversitesi

Fen Bilimleri Enstitüsü

Enformatik Anabilim Dalı

Danışman : Doç. Dr. Mehmet Hakan SATMAN

Ham veriler, veriye özel tasarlanmış bir veri yapısı sayesinde etkin biçimde saklanabilir ve işlenebilirler. İlişkisel veritabanları, Codd (1970) tarafından ortaya atılan, verinin daha küçük tablolar (varlıklar) içerisinde ifade edilmesiyle birlikte veri tekrarını en aza indirebilen; veriyle ilgili işlemlerde performansı arttırabilecek bir veri saklama yöntemidir. Elektronik tablo biçiminde saklanan bir veri setinin, ilişkisel veritabanı formuna getirilmesi işlemine normalizasyon adı verilmektedir. Normalizasyon, verinin normal formlara getirilmesiyle uygulanmaktadır. Ancak normal formlar ile birlikte gelen ve uyulması gereken kurallar, sezgisel ve öznel kararlara bağlı olarak uygulandığı için tasarımcıya göre farklılık gösterebilmektedir. Bu sebeple her zaman en iyi ilişkisel veritabanı tasarımına ulaşılamayabilir. Bu durum, normal form kurallarının sistematik ya da otomatik olarak uygulanmasının zorluğunu göstermektedir. Literatürde normalizasyon işleminin otomatik olarak gerçekleştirilebilmesiyle ilgili çeşitli çalışmalar bulunmaktadır. Ancak bu çalışmaların ortak dezavantajı, kullanıcıdan fonksiyonel bağımlılıkların talep ediliyor olmasıdır. Fonksiyonel bağımlılıkların hatalı belirlenmesi, ilişkisel veritabanı tasarımının hatalı oluşturulmasına sebep olabileceği için sürecin otomatik hale getirilmesindeki zorluk, varlığını korumaktadır. Bu tez çalışması kapsamında normalizasyon işlemi bir optimizasyon problemi olarak ele alınmıştır. Amaç fonksiyonu, normalizasyon tanım ve kuralları gözetilerek önerilmiştir. Literatürdeki çalışmalardan farklı olarak bu çalışmada kullanıcıdan herhangi bir bilgi istenilmemekte, verilen ham veri setine karşılık ilişkisel veritabanı tasarımı önerisi sunulabilmektedir. Uygulama,

seçilen 20 veritabanının veri setine dönüştürüldükten sonra önerilen algoritmanın tekrar aynı tasarımı önermesi beklenerek gerçekleştirilmiştir. Uygulama sonuçları, algoritmanın başarılı bir şekilde ilişkisel veritabanı tasarımı önerisi sunabileceğini göstermiştir.

Mayıs 2017, 199 sayfa.

Anahtar kelimeler: İlişkisel veritabanı, normalizasyon, otomatik normalizasyon, genetik algoritmalar, çok amaçlı optimizasyon.

SUMMARY

Ph.D. THESIS

CONVERTING RAW DATASETS TO RELATIONAL DATABASES USING GENETIC ALGORITHMS and AN APPLICATION

Emre AKADAL

İstanbul University

Institute of Graduate Studies in Science and Engineering

Department of Informatics

Supervisor : Assoc. Prof. Dr. Mehmet Hakan SATMAN

Raw datasets can be stored and processed through a specific modelled data modal. Relational database, suggested by Codd (1970), is a data management method that can minimize the redundant parts of data thanks to the identification of the data in smaller tables (entities) and increase the performance in data-related processes. The process of converting a data set stored in electronic table form into relational database form is called normalization. Normalization is applied by converting the data into normal forms. However, since the rules arising from the normal forms and to be followed are applied depending on intuitive and subjective decisions they can vary according to the designer. Therefore, the best relational database design cannot always be achieved. This indicates the difficulty of applying normal form rules systematically or automatically. In literature, there are various studies regarding the automatic execution of the normalization process. However, the common disadvantage of these studies is that they demand functional dependencies from the user. Since the determination of functional dependencies incorrectly causes the incorrect design of the relational database, the difficulty in automatizing the process preserves its existence. Within the scope of this thesis study, normalization process was discussed as an optimization problem. The objective function was proposed by considering the definition and rules of normalization. Unlike the studies in the literature, in this study no information is demanded from the user and a relational database design suggestion can be offered in response to the given raw data set. The

application was performed after the selected 20 databases were turned into data set with an expectation that the proposed algorithm results the same design again. The application results showed that the algorithm could successfully put forward a relational database design proposal.

May 2017, 199 pages.

Keywords: Relational database, normalization, automatic normalization, genetic algorithms, multi-objective optimization.

1. GİRİŞ

Veriyi kaydetmek, her zaman bilgisayarların temel özelliklerinden biri olmuştur. Rowley (2007) tarafından sunulan Bilgelik Hiyerarşisi (Wisdom Hierarchy), bilgiye erişmede temel bileşenin veri olduğunu göstermektedir. Hiyerarşiye göre, ele alınan veri miktarındaki artış, daha ayrıntılı ve daha nitelikli bilgiye erişmede fayda sağlayabilmektedir. Yeterli veri miktarına sahip olmak bilgiye erişme sürecinde önemli rol oynamaktadır (Fayyad, Piatetsky-Shapiro ve Smyth, 1996).

Teknolojik gelişmelerle birlikte, elektronik ortamda kayıt yapılabilme kapasitesinin de artmasıyla birlikte daha fazla miktarda veriyi kaydetme ihtiyacı meydana gelmiştir. Verinin kaydedilmesiyle ilgili yöntemler, bilgiye ulaşma sürecinde verinin daha kolay işlenebilmesi için önem taşımaktadır. Kaydedilen veri miktarının artması, veriyle yapılacak işlemler için dikkatli olunmasını gerektirmektedir (Dunkel ve Soparkar, 1999). Yadav ve Goyal (2013) tarafından bilgisayarlaştırılmış kayıt sistemi olarak tanımlanan veritabanları, veri miktarı arttıkça veri işleme süreçlerinde kullanılabilecek çok sayıda yöntem olarak ortaya çıkmıştır. Bu çok sayıdaki yöntemin birçoğu tamamlanamamış olsa da öne çıkan veri yönetimi modelleri kronolojik sırayla hiyerarşik, ağ ve ilişkisel veritabanıdır (Codd, 1981).

Hiyerarşik veritabanları, geleneksel ve herhangi bir kural ya da düzene sahip olmayan veri saklama yöntemlerinin getirdiği dezavantajlardan dolayı önerilmiştir (Henry, 1969). Bu veritabanları, tüm veri parçalarını hiyerarşik olarak birbirine bağlayarak çalışmaktadır. Hiyerarşik veritabanlarının getirdiği en büyük dezavantaj, verinin her zaman tek tip hiyerarşiye sahip olacağı varsayımıdır. Hiyerarşik veritabanında veri yalnızca tek bir yere bağlı olmak zorundadır. Bir veri parçasının birden fazla bağlantıya sahip olması gerekliliği, hiyerarşik veritabanları için çözülmesi zor bir sorun olarak ortaya çıkabilmektedir.

Bachman (1969), ağ veritabanı kavramını ortaya çıkartmıştır. Ağ veritabanı, hiyerarşik veritabanının en büyük zaafiyetini ortadan kaldırmaktadır. Her bir veri parçası, işaretçiler

vasıtasıyla birbirleri ile ilişkilendirilebilmektedir. Bu sayede hiyerarşik veritabanlarında bulunan, her veri parçasının tek bir yere bağlı olma zorunluluğu ortadan kalkmaktadır.

Codd (1970), ağ veritabanı fikrini geliştirerek ve içerdiği problemlere çözüm bularak ilişkisel veritabanı fikrini ortaya atmıştır. Günümüzde ilişkisel veritabanları, bilgisayar sistemlerinde ve yazılımlarda sıklıkla kullanılmaktadır. Başlık 2.1 altında ilişkisel veritabanları ve özelliklerine ayrıntılı olarak değinilmiştir.

Bu tez çalışmasında, bir veri setinin ilişkisel veritabanı formuna getirilebilmesi için uygulanması gereken normalizasyon sürecinin otomatik olarak gerçekleştirilebilmesi konusu üzerine çalışılmıştır. Normalizasyon süreci ve ayrıntıları Başlık 2.1.4'te; otomatik normalizasyon süreci ve daha önce gerçekleştirilmiş çalışmalar Başlık 2.1.5'te ele alınmıştır. Bilişsel olarak uygulanması gereken kurallardan oluşan, dolayısıyla öznel değerlendirme ile gerçekleştirilebilen normalizasyon işlemi, tez çalışması içerisinde bir optimizasyon problemi olarak ele alınmıştır. Bir veri modelinin normalizasyonunu sağlaması beklenen bir amaç fonksiyonu tasarlanmış ve önerilmiştir. Optimizasyon işleminin gerçekleştirilebilmesi için araç olarak bir sezgisel optimizasyon yöntemi olan genetik algoritmalar tercih edilmiştir.

İkinci bölüm, iki alt başlıktan oluşmakta ve genel literatür taraması sonuçlarını içermektedir. Birinci alt başlıkta ilişkisel veritabanı kavramı incelenmiştir. İkinci alt başlıkta ise seçilen optimizasyon yöntemi olan genetik algoritmalardan bahsedilmiş; temel operatör ve özellikleri anlatılmıştır. Üçüncü bölüm çalışmanın yönteminin ayrıntılarını teknik bir anlatımla sunmaktadır. Dördüncü bölümde uygulanan yöntem sonucunda elde edilen bulgular sunulmuştur. Beşinci bölümde, tüm bu çalışma ile elde edilen sonuçlar, fayda ve eksiklikleriyle birlikte sunulmuş; ileri çalışmalar için önerilerde bulunulmuştur.

2. GENEL KISIMLAR

2.1. İLİŞKİSEL VERİTABANLARI

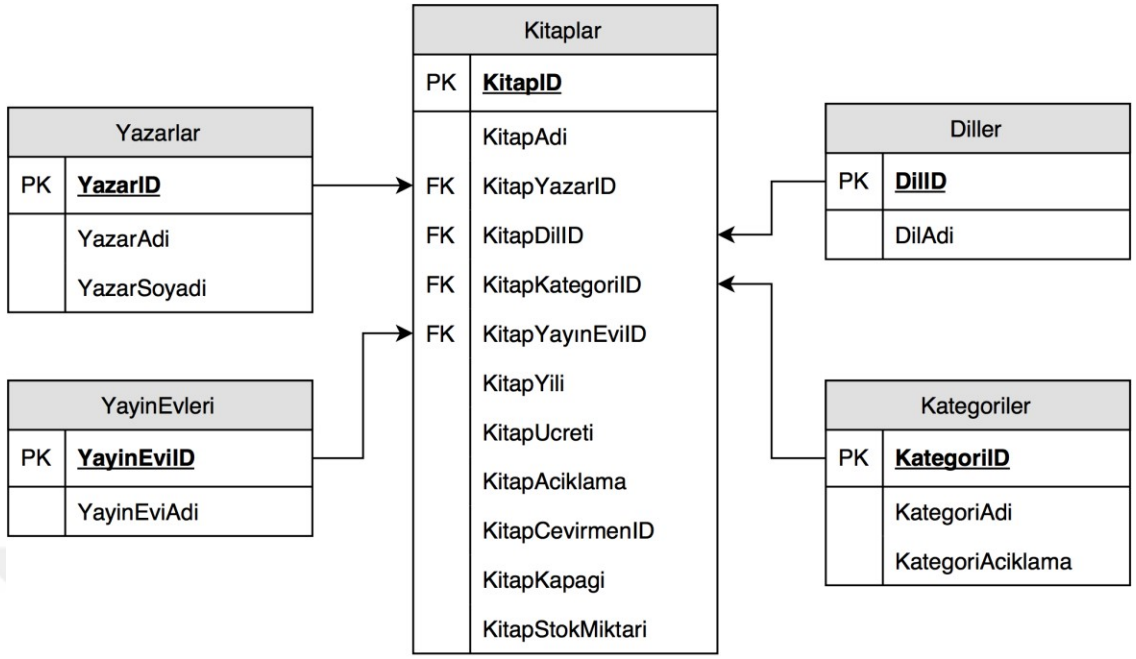
Hiyerarşik ve ağ veritabanları, geleneksel dosya temelli sistemlere göre daha kullanışlı olmasına rağmen tüm ihtiyaçları karşılayamamaktadır. Bu veritabanlarının sınırlılıkları, ilişkisel modelin (the relational model) geliştirilmesine öncülük etmiştir. İlişkisel model kavramı, bir yapı ve dil kullanarak veri yönetme yaklaşımı olarak ortaya çıkmıştır (Codd, 1969). Buna bağlı olarak Codd (1970), ilişkisel veritabanı modelini sunmuştur. Veritabanı, verilere kolaylıkla erişilmesini ve organize edilmesini sağlayan yapılardır (Çağıltay ve Tokdemir, 2010). Codd (1982), ilişkisel veritabanının ortaya çıkmasındaki motivasyonunu üç amaç ile belirtmiştir. Birincisi ve en önemlisi veritabanı yönetimindeki fiziksel ve mantıksal beklentilerin kesin ve net olarak birbirinden ayrılabilmesidir. Buna verinin bağımsızlığı (data independence) adı verilmiştir. İkinci motivasyon tüm kullanıcı ve programcıların anlayabileceği basit bir yapı ortaya koyabilmektir. Buna ifade edilebilirlik (communicability) denilmiştir. Üçüncü motivasyon ise kullanıcıların veri üzerinde işlem yürütebilmesini sağlayacak, kaydı işleme (set-processing) adı verilmiş üst seviye bir dil oluşturmaktır.

İlişkisel veri modeli; ilkel veri tipleri, kayıtlar, kayıt kümeleri (tablo/varlık) ve ilişkilerden oluşmaktadır (Read, Fussell ve Silberschatz, 1992; Nizam, 2011). Veri, ilişkisel veritabanı içerisinde, varlık adı verilen yapılarda saklanmaktadır. Veriye erişim, verinin pozisyonu yerine değeriyle sağlanmaktadır. Bu özellik, son kullanıcı ve programcıların üreticiliğini arttırmaktadır (Codd, 1982). İlişkisel veritabanları bir ya da birden fazla ilişkisel varlıktan oluşabilir. İlişkisel yapıdaki her bir veri kaydı, birincil anahtar (primary key) adı verilen, her bir kaydın benzersiz olmasını sağlayabilen, bir ya da birden fazla nitelik birleşimi ile oluşturulabilen bir niteliğe (ya da nitelik grubuna) sahip olmalıdır. Yapı içerisindeki bazı niteliklerin birbirleriyle çapraz şekilde ilişkili olması durumunda, bu çapraz ilişkiyi gösteren ve ikincil anahtar (secondary/foreign key) adı verilen bir nitelik kullanılabilmektedir (Codd, 1970). Birincil ve ikincil anahtarlar sırasıyla Başlık 2.1.1 ve Başlık 2.1.2 içerisinde ayrıntılı olarak ele alınmaktadır.

Varlıklar, niteliklere ve bu niteliklerin değerlerine sahip olan iki boyutlu yapılardır. Tasarım sırasında kaç nitelik olacağı, niteliklerin alabileceği değerlerin türü ve karakter sayısının üst sınırı belirlenir. Varlığa zaman içerisinde yeni kayıtlar eklenebilir, mevcut kayıtlar silinebilir ya da kayıtlar üzerinde değişiklik yapılabilir. Bu durum varlığın, dolayısıyla veritabanının yönetimi ile ilgilidir. Tasarım hatasından kaynaklanan herhangi bir problemle karşılaşılmadığı veya yapıda geliştirme yapılması planlanmadığı sürece niteliklerde değişiklik yapılmasına ihtiyaç duyulmamakta, yalnızca kayıtlar ile ilgili işlemler gerçekleştirilmektedir.

Veritabanında kaç adet varlık olacağı, tasarım aşamasında belirlenir ve yönetilmek istenilen veriye bağlı olarak belirlenir. *Kullanıcı*, *Ders*, *Şube*, *Klasör* varlık ismine birer örnektir. Varlık içerisindeki niteliklerin birbirleriyle ilişkili olması beklenmektedir. Varlıklar da kendi aralarında, niteliklerin arasında bulunandan farklı bir ilişki türü bulundurmaktadır. Bu ilişki türü 1:1, 1:n ve n:m olmak üzere üç çeşide sahiptir (Teorey, Yang ve Fry, 1986). Bu ilişki türleriyle ilgili ayrıntılı bilgi Başlık 2.1.6 içerisinde sunulmuştur.

Şekil 2.1'de verilen örnekte *Yazarlar*, *YayinEvleri*, *Diller*, *Kategoriler* ve *Kitaplar* adlı 5 varlık bulunmaktadır. Her bir varlığın nitelikleri, varlığa ait kutu içerisinde gösterilmiştir. Bazı niteliklerin başında bulunan PK ifadesi, o niteliğin birincil anahtar özelliği taşıdığını; FK ifadesi ise ikincil anahtar özelliği taşıdığını göstermektedir.



Şekil 2.1: Örnek İlişkisel Veritabanı Tasarımı.

İlişkisel veritabanının önde gelen avantajları;

- yapılandırılmış veri olması,
- gereksiz tekrarın azlığı,
- genişletmenin kolaylığı,
- yöntem ve veri bağımsızlığı,
- veri yönetimini kolaylaştırması ve merkezileştirilmesi,
- veri entegrasyonu sağlanabilmesi,
- veri tutarsızlığının olmaması,
- çok kullanıcının işlem yapmasına izin vermesi

olarak belirtilmiştir (Sumathi ve Esakkirajan, 2007; Zhu, Zeng ve Cao, 2010).

Raghu ve Johannes (2000), veri yönetiminde ilişkisel veritabanı kullanılmasının getirebileceği bazı problemleri bir örnek ile açıklamıştır. Örnek ele alındığında yaşanabilecek problemler;

1. Çok büyük miktarda verinin tek bir disk ya da bilgisayar üzerinde toparlanabilmesi çok zor olabilir

2. Çok büyük boyutta bir veri dosyasının açılması bilgisayar belleği için yetersiz gelebilir
3. Verinin işlenmesi için özel programlar yazılması gerekebilir. Bu programın yazılmasında dikkate alınmayan bir durum veri bütünlüğünü bozabilir
4. Veriye farklı kullanıcıların erişim sağlaması sonucunda tutarsızlık oluşabilir
5. Herhangi bir işlem esnasında sistem hatası oluşması, veri bütünlüğünün bozulmasına sebep olabilir
6. Sisteme erişimde etkin bir güvenlik yönteminin bulunmaması, veriye erişmesi istenmeyen kişilerin engellenmesinin sağlanamamasına yol açabilir

olarak sıralanabilmektedir.

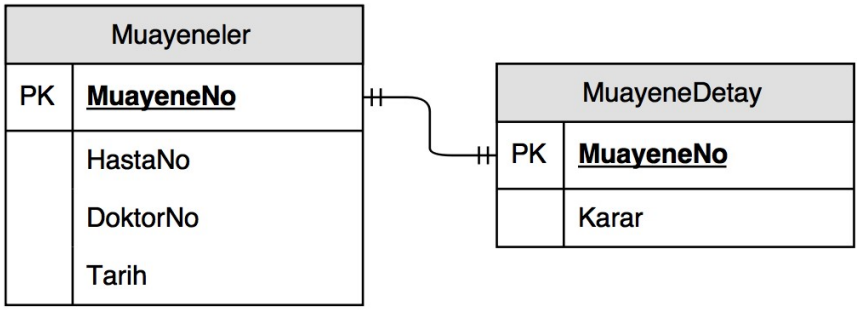
2.1.1. Birincil Anahtar

Birincil anahtar, ele alınan bir varlık içerisindeki kayıtları tanımlayabilmeyi sağlayan ve eşsiz değer alan nitelik ya da nitelik grubudur (Chen, 1976). Tek başına benzersizlik sağlamayan nitelikler, birlikte dikkate alındığında benzersizlik sağlıyorsa birincil anahtar olarak kabul edilebilirler. Birincil anahtarlar genellikle bir varlık içerisinde ilk sıradaki nitelik olarak yer almaktadırlar. Varlık içerisinde, tüm kayıtlar için benzersizliği sağlayabilecek bir nitelik bulunması durumunda birincil anahtar özelliği bu nitelik tarafından taşınabilmektedir. Ancak böyle bir nitelik olmaması ya da olsa bile tüm durumlar için benzersizlik sağlayamayacağı düşünülen durumlarda veritabanı tasarımcısı tarafından otomatik artan, sayısal değerli bir nitelik oluşturulur. Bu nitelik genellikle "ID" adını almaktadır. Günlük hayatta karşılaşılan "ID" kavramı da benzer şekilde meydana gelmektedir. T.C. Kimlik Numarası, birincil anahtar için verilebilecek en basit örneklerden biridir. Benzer şekilde banka hesap numaraları ECBS (European Committee for Banking Standards) tarafından ISO 13616:1997 standartları ile IBAN (International Bank Account Number) formatı sayesinde benzersiz ve tanımlayıcı hale getirilmiştir[1].

Varlık içerisinde bulunan ve birincil anahtar özelliği taşıyabileceği düşünülen nitelikler, varlığa eklenecek yeni kayıtlarla bu özelliğini kaybetme olasılığına sahip olabilir. Bu sebeple birincil anahtarın varlık içerisindeki niteliklerden herhangi biri olarak seçilmesi yerine yeni bir nitelik olarak varlığa dahil edilmesi, olası problemlerden kaçınmayı

[1] EBS204 V3 (2001) URL: http://arlindo-correia.com/IBAN.pdf [Erişim tarihi: 01.11.2016]

sağlayabilecektir. Bu çözüm sayesinde, varlık içerisindeki eklenmesi muhtemel tüm kayıtlar için benzersizlik sağlayabilecek bir nitelik elde edilmiş olmaktadır.



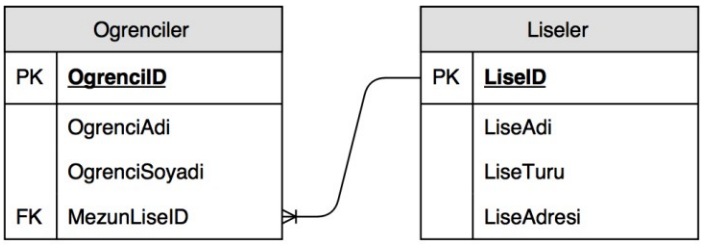
Şekil 2.2: Birincil Anahtar Örneği.

Şekil 2.2'de *Muayeneler* ve *MuayeneDetay* varlıklarının *MuayeneNo* nitelikleri birincil anahtar olarak belirlenmiştir. Şekil 2.2'de, nitelik önüne gelen PK ifadesi, birincil anahtar olduğunu göstermektedir. Örnekte *MuayeneNo* niteliği, tablodaki her bir kaydın benzersiz olmasını sağlayan sayısal bir değerdir. *MuayeneNo* niteliğine ait bir değer, karşılık gelen kaydın tamamına doğru bir şekilde erişmeyi sağlayabilmektedir. Bu sayede her bir birincil anahtar değeri, tek bir kayda karşılık gelmektedir.

2.1.2. İkincil Anahtar

İkincil anahtar (Secondary Key/Foreign Key), iki varlık arasındaki bağlantının korunabilmesi için bir varlığın birincil anahtar niteliğine ait değeri başka bir varlık içerisinde bulunduran niteliktir. İkincil anahtarlar, iki varlık arasındaki ilişkiyi oluşturabilmeye yardımcı olmaktadır. Bir varlık içerisindeki kaydın, diğer varlık içerisindeki hangi kayda karşılık geldiğinin sistematik biçimde tanımlanabilmesi için kullanılmaktadır.

Şekil 2.3, ikincil anahtara bir örnek içermektedir. Şekil, *Ogrenciler* ve *Liseler* isimli iki varlık ve bunlara bağlı nitelikleri göstermektedir. Şekil 2.3'te, yanında PK ifadesi bulunan nitelikler birincil anahtar, FK ifadesi bulunan nitelik ise ikincil anahtar özelliği taşımaktadır.



Şekil 2.3: İkincil Anahtar Örneği.

Şekil 2.3 ile verilen örnekte; öğrenci kayıtları *OgrenciID* niteliğinin, lise kayıtları ise *LiseID* niteliğinin birincil anahtar olarak seçilmesiyle ifade edilmektedir. Ayrıca her bir öğrencinin mezun olduğu lise bilgisi *MezunLiseID* niteliği ile ilişkilendirilmektedir. *MezunLiseID* niteliği, *Liseler* varlığı içerisindeki *LiseID* niteliğinin değerini taşımaktadır. Bu sayede aynı liseden mezun tüm öğrenciler için lise bilgileri tekrarlı olarak yazılmamaktadır. Her lise, *Liseler* varlığı içerisinde bir kez yer almakta ve bir birincil anahtara sahip olmaktadır. *Ogrenciler* varlığı, ilgili liseye birincil anahtarı vasıtasıyla ulaşmaktadır.

2.1.3. Fonksiyonel Bağımlılık

Fonksiyonel bağımlılık kavramı, veri yapısı hakkında düşünmek için oldukça kullanışlı bir araçtır. Ele alınan bir varlıkta $\{X_1..X_n\}$ ve $\{Y_1 ... Y_n\}$ olarak ifade edilen iki nitelik kümesi bulunduğu durumda Y, X'e fonksiyonel bağımlı ise X'in herhangi bir değeri Y'nin yalnızca tek bir değerine denk gelmektedir (Riordan, 2005). Fonksiyonel bağımlılık, varlık içerisindeki niteliklerin tek bir nitelik ile ifade edilebilmesi açısından fayda sağlamaktadır. Bir Y niteliği, X niteliği tarafından temsil edilebiliyor ve bir X niteliği yalnızca tek bir Y niteliğini işaret ediyorsa Y, X'e fonksiyonel bağımlıdır denir ve

$$X \rightarrow Y$$

şeklinde gösterilir (Vardi, 1987).

Basit bir örnek olarak; *Kimlik Numarası* ile *Ad Soyad* nitelikleri incelendiğinde, *Ad Soyad* niteliğine ait herhangi bir değer, *Kimlik Numarası* niteliğine ait bir değerle temsil edilebilir. Böylece *Kimlik Numarası* niteliğine ait her bir değer, *Ad Soyad* niteliğine ait tek bir değere denk gelir. Bu durumda *Ad Soyad*, *Kimlik Numarası* niteliğine fonksiyonel bağımlıdır denilebilir ve aşağıdaki gibi ifade edilir.

$$Kimlik\ Numarası \rightarrow Ad\ Soyad$$

Şekil 2.3'te bulunan *Ogrenciler* varlığı incelendiğinde, *OgrenciAdi*, *OgrenciSoyadi* ve *MezunLiseID* niteliklerinin *OgrenciID* niteliğine fonksiyonel bağımlı olduğu görülmektedir. Bu durum şu şekilde ifade edilebilir;

$$OgrenciID \rightarrow \{OgrenciAdi,\ OgrenciSoyadi,\ MezunLiseID\}$$

Benzer şekilde *Liseler* varlığı için fonksiyonel bağımlılık, aşağıdaki şekilde ifade edilebilir.

$$LiseID \rightarrow \{LiseAdi,\ LiseTuru,\ LiseAdresi\}$$

Bir varlık içerisindeki niteliklerin, bir nitelik kümesine fonksiyonel bağımlı olabileceği gösterilmişti. Bu durumda ilgili nitelik kümesi birincil anahtar olarak belirlenebilir. Ancak bu durumda seçilen birincil anahtarın bir alt kümesine fonksiyonel bağımlılık oluşursa birincil anahtar seçimi yanlış yapılmış olabilir. Bu sebeple bir nitelik kümesinin birincil anahtar olarak atanması riskli sayılabilir.

2.1.4. Normalizasyon

Normalizasyon, çok sayıda niteliğe sahip veri seti için her bir niteliğin karşılıklı bağımlılıkları analiz edilerek veri setinin, aykırılık içermeyen ve iyi yapılandırılmış ilişkilerle daha küçük gruplara ayrılması sürecidir (Lee, 1994; Hoffer, Ramesh ve Topi, 2016). Normalizasyonun amacı, tutarlı ve geçerli bir şekilde güncellenebilen, gereksiz tekrar eden veri miktarını en aza indiren ilişkili varlıklar kümesi oluşturmaktır (Bahmani, Naghibzadeh ve Bahmani, 2008). Bu amaç sayesinde, verinin tutarlılığını ve geçerliliğini bozmadan, değiştirilebilir ve gereksiz veriden arınmış ilişkiler yaratılabilmektedir (Soler ve diğ., 2006). Normalizasyon sayesinde bir veritabanı, genel veritabanı prensiplerine uygun hale getirilmiş olmaktadır (Köseoğlu, 2007).

Normalizasyonun hedefleri;

1. Hatalardan kaçınarak ve boş disk alanını koruyarak gereksiz tekrar eden veriyi en aza indirme,
2. Kısıt uygulamayı kolaylaştırma,
3. Veri bakımını daha kolay hale getirme (veri ekleme, güncelleme ve silme),

4. Gerçek dünyayı daha iyi yansıtan ve oluşabilecek durumlar için daha güçlü bir temel oluşturabilen bir tasarım sağlama

olarak sıralanmaktadır (Hoffer, Ramesh ve Topi, 2016).

Veri, normalizasyon süreci içerisinde çok sayıda tabloya bölünebilmektedir ancak bu işlemin gelişigüzel yapılması mümkün değildir. Normalizasyon sürecinin ilk adımında fonksiyonel bağımlılıkların doğru olarak tanımlanması beklenmektedir. Bu sayede her nitelik, en uygun varlık içerisinde yer alabilecektir. Varlıklar, yalnızca birbirleriyle doğrudan ilişkili niteliklere sahip olmalıdır. Bu gereklilik, gereksiz tekrarlı veri içermeyen, bilgi bütünlüğü ve tutarlılığı korunan ve üzerinde işlem yapmaya izin veren bir ilişkisel model elde edilmesini sağlayabilmektedir (Nizam, 2011). Normalizasyon, iyi bir veritabanı tasarımının hayati öneme sahip parçası olarak kabul edilmektedir (Akehurst, Bordbar, Rodgers ve Dalgliesh, 2002).

Normalizasyon süreci, normal formlar ile belirlenmiş bir dizi veri bağımlılığı kuralının adım adım uygulanmasıyla gerçekleştirilir (Lee, 1994; Teorey ve diğ., 2011; Hoffer, Ramesh ve Topi, 2016). Tanımlanan kurallar yardımıyla veritabanı tasarımının gerçekleştirilmesinin faydası, tasarım aşamasında tasarımcının bir yol haritasına sahip olabilmesi ve olası tasarım hatalarına uğramadan sonuca ulaşabilmesinin sağlanmasıdır. Normalizasyon kuralları bir kılavuz niteliği taşısa da karar veren kişi veritabanı tasarımcısı olduğu için bu süreç öznel olarak gerçekleştirilebilmektedir. Bu sebeple elde edilen bir ilişkisel veritabanı tasarımı, en iyi şekilde gerçekleştirilememiş olabilir.

Veritabanı tasarımcısının, iyi bir veritabanının taşıması gereken tüm özellikleri ve yapılandırılacak verinin özelliklerini gözeterek en uygun tasarımı oluşturması beklenmektedir. İyi tasarlanmış bir veritabanı yapısına ulaşabilmek için sırasıyla uygulanması gereken adımları belirleyen normal formlar, normalizasyon teorisinin temelinde bulunmaktadır (Soler ve diğ., 2006). Literatürde 7 normal form ile karşılaşılmıştır. Bunlar;

- 1. Normal Form
- 2. Normal Form
- 3. Normal Form

- Boyce-Codd Normal Formu
- 4. Normal Form
- 5. Normal Form
- Alan/Anahtar (6.) Normal Form

olarak sıralanabilir (Fagin, 1981; Hoffer, Ramesh ve Topi, 2016).

Normalizasyon süreci içerisinde; veri kaydında, düzenlenmesinde ve işlenmesinde karşılaşılabilecek durumların göz önünde bulundurulması önemlidir. Bu sebepledir ki normalizasyonu anlamak ve öğrenmek için en iyi yol pratik yapmaktır (Soler ve diğ., 2006). Normal formlar, ilişkisel veritabanı kitaplarında ayrıntılı ve örnekli olarak açıklanmaktadır (Ramakrishnan ve Gehrke, 2007; Teorey, Yang ve Fry, 2011; Hoffer, Ramesh ve Topi, 2016). Ayrıca çok sayıda elektronik ve herkesin erişimine açık ücretsiz eğitim materyali mevcuttur.[2] Bu sebeple tez çalışması içerisinde normal formların yalnızca tanım ve genel özelliklerine değinilmiştir.

2.1.4.1. 1. Normal Form

Verinin 1. normal formda (1NF) olabilmesi için, varlığa ait niteliklerin her bir satırda atomik değer alması gerekmektedir (Teorey, Yang ve Fry, 2011). Verinin 1NF'de olmaması, genelde verinin hatalı kaydedilmesinden kaynaklanmaktadır. Tablo 2.1'de, 1NF'ye uygun olmayan bir örnek verilmiştir. Varlık içerisinde yer alan *Renk* niteliği, bazı kayıtlarda birden fazla değer almaktadır. Birden fazla değer aldığı durumlarda değerlerin virgül ile ayrıldığı görülmektedir.

Tablo 2.1: 1NF'ye Uygun Olmayan Veri Yapısı.

UrunID	Renk	Fiyat
1	kırmızı, yeşil	15,99
2	sarı	23,99
3	yeşil	17,5
4	sarı, mavi	9,99
5	kırmızı	29,99

[2] URL: https://www.udacity.com/course/intro-to-relational-databases--ud197 [Erişim tarihi: 04.11.2016]
URL: https://lagunita.stanford.edu/courses/DB/2014/SelfPaced/about [Erişim tarihi: 04.11.2016]
URL: https://www.coursera.org/courses?query=relational+database [Erişim tarihi: 04.11.2016]

Tablo 2.1'de verilen varlıkta, *UrunID* niteliğinin 4 değerini aldığı kayıtta *Fiyat* niteliğinin 9,99 değerini aldığı görülmektedir. Ancak aynı kaydın *Renk* niteliğinin iki farklı değer içerdiği görülmektedir. Bu sebeple varlığın birincil anahtarı gerçekte iki kaydı birden işaret etmektedir. Bu durum, ilişkisel veritabanı yapısına aykırıdır ve bu varlık tasarımı 1NF'de değildir.

Karşılaşılabilecek tüm benzer durumlar için 1NF'ye ulaşmanın iki yolu bulunmaktadır. Hangi yolun seçileceği, verinin niteliğine ve ifade ettiği anlama göre farklılık göstermektedir. Birinci durumda, normal forma uygunluk göstermeyen nitelik birden fazla nitelikten oluşan birleşik bir değeri içeriyorsa, her bir nitelik varlık içerisinde ayrı nitelikler olarak tanımlanmalıdır. Bu sayede yapı 1NF'ye ulaşabilir. İkinci durum, aynı niteliğe ait birden fazla değer bulunması durumudur. Tablo 2.1 bu duruma örnek teşkil etmektedir. Aynı ürünün birden fazla renk seçeneğinin bulunması ve bunun tek kayıt ile ifade edilmeye çalışılması varlığın 1NF'ye uygun olmamasına sebep olmuştur. Bu durumda 1NF'ye ulaşabilmek için, birden fazla değerin bulunduğu satırların, değer sayısı kadar tekrar edilmesi gerekmektedir. Tablo 2.1'deki 1. satır, *UrunID* niteliğinin değeri 1, *Fiyat* niteliğinin değeri 15,99, *Renk* niteliğinin değeri kırmızı ve yeşil olmak üzere iki kayıt haline getirilmelidir. Benzer şekilde *UrunID* niteliğinin değeri 4 olan kayıt, *UrunID* ve *Fiyat* niteliklerinin değerleri tekrarlanarak, iki farklı *Renk* niteliği değeri için iki kayıt haline getirilmelidir.

1NF'ye getirilmeyen yapılar, diğer normal formlara uygun gibi görülebilir ancak bu durum yanıltıcıdır. 1NF uygulanmadan diğer normal formlar için işlem yapmak hatalı bir yapı oluşmasına sebep olacaktır. Nitekim Tablo 2.1'de 1NF'nin uygulanmadığı durumda veri tekrarı problemi görülmüyorken, uygulandıktan sonra veri tekrarı ortaya çıkmaktadır. Bu yeni bir problem değil, tasarımın 1NF'ye uygun hale getirilmesi sonucunda fark edilebilen bir problemdir.

2.1.4.2. 2. Normal Form

2. normal forma (2NF) uygunluk sağlanabilmesi için öncelikle 1NF'ye uygunluk sağlanmalıdır. 2NF'ye aykırılık, varlıkta yer alan ve anahtar olmayan bir niteliğin, varlığın birincil anahtarının bir alt seti olması durumunda ortaya çıkmaktadır (Kent, 1981). Bir varlık içerisindeki tüm niteliklerin aday birincil anahtara fonksiyonel bağımlı

olması beklenmektedir. 2NF, yalnızca ilişkili niteliklerin varlık içerisinde yer almasını sağladığı için, ilişkisel varlıkların oluşturulduğu süreç olarak kabul edilebilir.

Tablo 2.2: 2NF'ye Uygun Olmayan Veri Yapısı.

PK				PK			
SiparisID	**SiparisTarih**	**MusteriID**	**MusteriAdi**	**UrunID**	**UrunAdi**	**Sayi**	**Fiyat**
1111	2 Subat 2015	10	Mehmet	111	Ürün 1	8	20
1112	2 Mart 2015	20	Kaan	112	Ürün 2	6	8
1113	2 Nisan 2015	30	Zeynep	113	Ürün 3	2	14

Tablo 2.2'de, 1NF'ye uygun, ancak 2NF'ye uygun olmayan veri yapısı gösterilmektedir. Şekildeki varlıkta *SiparisTarih*, *MusteriID* ve *MusteriAdi* nitelikleri, *SiparisID* niteliğine fonksiyonel bağımlıdır. *UrunAdi*, *Sayi* ve *Fiyat* nitelikleri ise *UrunID* niteliğine fonksiyonel bağımlıdır. Her ne kadar sipariş tarihi ile ürün adı ilişkili olsa da doğrudan bir ilişki (fonksiyonel bağımlılık) bulunmadığı için Şekil 2.2'deki veri yapısı, 2 varlık ile ifade edilmelidir. Böylece her bir varlık içerisinde, tüm nitelikler tek bir niteliğe fonksiyonel bağımlı olacak; bu nitelik de birincil anahtar özelliği kazanabilecektir.

2.1.4.3. 3. Normal Form

3. normal forma (3NF) ulaşmak için, anahtar özelliği taşımayan niteliklerden hiçbiri anahtar özelliği taşımayan başka bir niteliğe bağımlı olmamalıdır (Bernsteing, 1976). 3NF'nin öncül kuralı, 2NF'nin sağlanmış olmasıdır. Veri yapısı 2NF'ye getirilmeden bu adıma geçiş tasarım açısından sağlıklı olmayacaktır.

Veri yapısının 3NF'ye getirilebilmesi için, varlık içerisindeki niteliklerin birbirleri arasında fonksiyonel bağımlılık bulundurmaması gerekmektedir.

Tablo 2.3: 3NF'ye Uygun Olmayan Veri Yapısı.

MusteriNo	SehirKodu	SehirAdi
1	34	İstanbul
2	6	Ankara
3	6	Ankara
4	34	İstanbul

Tablo 2.3'te, *MusteriNo* niteliği birincil anahtar özelliğindedir. *SehirAdi* niteliği, birincil anahtarın yanı sıra *SehirKodu* niteliğine de fonksiyonel bağımlılık göstermektedir. Bu

sebeple Tablo 2.3, 3NF'ye uymamaktadır. 3NF'ye ulaşabilmek için varlık, 2 varlığa bölünmelidir. Tablo 2.4'te, 3NF'ye getirilmiş veri yapısı gösterilmektedir.

Tablo 2.4: 3NF'ye Uygun Veri Yapısı.

Tablo A	
MusteriNo	SehirKodu
1	34
2	6
3	6
4	34

Tablo B	
SehirKodu	SehirAdi
6	Ankara
34	İstanbul

2.1.4.4. Boyce-Codd Normal Form

3NF'ye getirilen veri yapısında anahtar olmayan tüm nitelikler bir birincil anahtara bağımlı haldedir (Hoffer, Ramesh ve Topi, 2016). Boyce-Codd normal forma (BCNF), varlık içerisinde tek bir niteliğin birincil anahtar olması ve nitelikler arasında herhangi bir kısmi bağımlılık olmaması durumunda erişilmektedir (Teorey, Yang ve Fry, 2011). Fonksiyonel bağımlılık başlığı (2.1.3) altında da bahsedildiği gibi, birincil anahtarın bir nitelik grubundan oluşması durumunda, diğer niteliklerin bu nitelik setine fonksiyonel bağımlı olması gerekmektedir. Bir niteliğin, birincil anahtarın bir alt kümesine fonksiyonel bağımlı olması durumu, kısmi fonksiyonel bağımlılık olarak adlandırılır. BCNF, veritabanı tasarımında kısmı fonksiyonel bağımlılık olmamasıyla erişilen bir formdur.

2.1.4.5. 4. Normal Form

4. normal form (4NF), Fagin (1977) tarafından öne sürülmüştür. 4NF'nin amacı veri yapısını 3NF ve BCNF'den daha güçlü hale getirmektir. Bir varlığa yeni kayıt eklenmesi sonucunda varlığın niteliklerinden birinin aynı değeri tekrar tekrar alması, veritabanı ilkelerine aykırıdır. 4NF'ye, bu problemin ortadan kaldırılmasıyla ulaşılabilmektedir. Örnek olarak, *Ad*, *Meslek*, *Şehir* niteliklerinden oluşan bir varlık ele alındığında, Adı, mesleği ve şehir bilgisi kayıtlı bir kişi için yeni bir meslek tanımlaması yapıldığında *Ad* ve *Şehir* nitelikleri tekrar aynı değerleri, *Meslek* niteliği ise yeni değeri almaktadır. Bu durumda varlık içerisinde yer alan iki kayıt için *Ad* ve *Şehir* nitelikleri aynı değere sahipken, *Meslek* niteliği iki farklı değere sahip olur. Bu durum gereksiz veri tekrarı oluşturduğu için aşılması gereken bir problem olarak ele alınmalıdır.

2.1.4.6. 5. Normal Form

5. normal form (5NF ya da projection-join normal form (PJNF)), Ronald Fagin tarafından öne sürülmüş olan, 4NF'ye göre daha güçlü bir normal formdur (Fagin, 1979). İlk 4 normal form, veri yapısındaki fonksiyonel bağımlılık ve veri tekrarını ortadan kaldırmak üzerine kuruludur. 5NF ise, veritabanı tasarımının en iyi hale gelip gelmediğini test eden bir kontrol aşamasıdır. Veritabanı tasarımı, veri kaybı yaşamadan bölünebileceği en küçük varlıklara bölündüğünde 5NF'ye ulaşmış kabul edilebilir. Yapının 5NF'de olduğundan emin olabilmek için verinin daha küçük varlıklara bölünemeyeceğinden emin olmak gerekmektedir.

2.1.4.7. Alan/Anahtar (6.) Normal Form

6. normal form (6NF) ya da diğer adıyla alan/anahtar normal formu (DKNF: Domain/Key Normal Form), Ronald Fagin (1981) tarafından önerilmiştir. Önceki NF'lere ulaşıldıktan sonra bu NF'ye ulaşılması gerekmektedir. Bir niteliğin değerinin, varlığın bir başka niteliğinin değerine bağlı olarak belirlenmesi durumunda veri içsel olarak koşullu kısıt içermektedir. 6NF'ye, bu durumun düzeltilmesiyle erişilebilmektedir.

Akademik Unvan ve *Maaş* niteliklerinin bulunduğu bir varlık bu normal formun ihlaline bir örnek teşkil etmektedir. *Akademik Unvan* niteliğinin aldığı değer, *Maaş* niteliğinin alacağı değer aralığını belirlemektedir. Benzer şekilde öğrencilerin bir sınava ait notlarının bulunduğu varlık içerisinde *Not* niteliği ile "Geçti" ya da "Kaldı" değerlerinden birini alan *Durum* niteliği birbiriyle doğrudan ilişkili olabilir. *Not* niteliğinin aldığı herhangi bir değer ile *Durum* niteliğinin alabileceği değere ulaşılabiliyorsa bu nitelik varlık içerisinde yer almamalıdır.

2.1.5. Otomatik Normalizasyon

Normalizasyonun amacı ve önemi, Başlık 2.1.4 altında ayrıntılı olarak sunulmuştur. İlgili başlık altında, normalizasyon sürecinin sezgisel ve tasarımcı kararlarına bağlı olarak gerçekleştirildiğinden bahsedilmişti. Bu başlık altında literatürde yer alan, normalizasyon işleminin bir tasarımcı yerine bilgisayar yazılımı yardımıyla yapılmasıyla ilgili çalışmalar incelenecektir. Literatürde, normalizasyon sürecini otomatik hale getirmeyi konu alan kısıtlı sayıda çalışma ile karşılaşılmıştır. Karşılaşılan tüm çalışmalar; eğitim, araç ve yöntem çalışmaları olarak üç grupta incelenmiştir. Eğitim çalışmaları grubunda, normalizasyonun öğretilmesiyle ilgili çalışmalar yer almaktadır. Araç çalışmaları

grubunda, normalizasyon sürecinin gerçekleştirilmesinde yardımcı olabilecek yazılımları konu alan çalışmalar incelenmiştir. Yöntem çalışmaları grubunda ise ilişkisel veritabanı tasarımını otomatik olarak gerçekleştirebilmek üzerine yapılmış çalışmalar sunulmaktadır. Bu tez çalışması da niteliği açısından yöntem grubu içerisinde incelenmesi gereken bir çalışmadır.

2.1.5.1. Normalizasyon Eğitimiyle İlgili Çalışmalar

Mitrovic (2002) tarafından yapılan çalışmada, NORMIT adında, üniversite öğrencilerine normalizasyonu öğretmeyi amaçlayan bir akıllı eğitim sistemi geliştirilmiştir. Çevrimiçi olmanın faydalarından yararlanabilmek için sistem web tabanlı olarak tasarlanmıştır (Mitrovic, 2002).

Benzer şekilde Fanguay ve Kleen (2005), normalizasyon eğitimini konu alan bir çalışma yapmıştır. Çalışmada oyunla öğretmenin başarısı göz önünde bulundurulmuştur. Normalizasyon Kapışması (Normalization Shootout) adı verilen puan ödüllü oyun, birden fazla turdan oluşmaktadır. Her tur, normalizasyonun farklı süreçleri hakkında öğrencilere sorular sorarak onları yarıştırır (Fanguy ve Kleen, 2005).

Soler ve diğ. (2006) tarafından gerçekleştirilen çalışmada, Girona Üniversitesi'nde web tabanlı eğitim sürecine katkı sağlayabilmek için, otomatik olarak veritabanı normalizasyonu problemleri üretebilen bir modül sunulmuştur. Sistem, nitelikleri ve nitelikler arasındaki fonksiyonel bağımlılıkları sunarak bireyin BCNF'de bir veritabanı tasarlamasını istemektedir (Soler ve diğ., 2006).

Ahmedi, Jakupi ve Jajaga (2012) tarafından gerçekleştirilen bir çalışmada, veritabanı normalizasyonu ve denormalizasyonu üzerine NormalDB adında interaktif bir e-öğrenme aracı geliştirilmiştir. Yöntem, fonksiyonel bağımlılık ve normalizasyon işlemini kavramadaki güçlüğü aşabilmek amacıyla geliştirilmiştir. Beklenti, öğrencilerin veritabanı tasarımıyla ilgili karmaşık işler için deneyimli hale gelmeleri ve süreci daha iyi anlayabilmeleridir (Ahmedi, Jakupi ve Jajaga, 2012).

Duggal, Srivastav ve Kaur (2014), veritabanı normalizasyonuna oyunlaştırma bakış açısı ile yaklaşmaktadır. Geleneksel yöntemlere göre oyunlaştırmanın avantajlarından bahsedilen çalışmada, her bir normal form birer oyun seviyesi olarak görülerek, doğru

hareketin de puan ile ödüllendirilmesi temeliyle normalizasyonun daha etkin bir şekilde öğretilebileceği savunulmaktadır (Duggal, Srivastav ve Kaur, 2014).

2.1.5.2. Normalizasyon Araçlarıyla İlgili Çalışmalar

Du ve Wery (1999), gerçekleştirdikleri çalışmada Micro adı verilen bir yardımcı normalizasyon aracı geliştirdiklerini belirtmişlerdir. Micro, nitelik listesi ve nitelikler arasındaki fonksiyonel bağımlılıkların verilmesi durumunda 2NF, 3NF ve BCNF'yi otomatik olarak gerçekleştirebilmektedir. Micro, normalize ettiği veritabanını Microsoft Access dosyası olarak çıktı vermektedir. Sonuca ulaşabilmek için veritabanı tasarımcılarının niteliklere ait fonksiyonel bağımlılıkları tanımlaması yeterli olmaktadır (Du ve Wery, 1999).

Yazıcı ve Karakaya (2007), Mathematica[3] yazılımı kullanarak JMathNorm adında interaktif bir normalizasyon aracı geliştirmiştir. Çalışma, gerçek zamanlı veritabanı tasarımı yapılabilmesi ve sınırlı matematik temeline sahip öğrencilere normalizasyon kavramının öğretilmesi amacını içermektedir. Program, nitelikleri ve bu niteliklere ait fonksiyonel bağımlılıkları alarak 3NF'ye ulaşmış bir veritabanı tasarımını sunmaktadır. Arayüz Java ve JLink[4] ile hazırlanmıştır (Yazıcı ve Karakaya, 2007).

2.1.5.3. Normalizasyon Yöntemleriyle İlgili Çalışmalar

Bahmani, Naghibzadeh ve Bahmani (2008), otomatik veritabanı normalizasyonu ve birincil anahtar üretimi konusunda bir çalışma gerçekleştirmiştir. Çalışmada bağımlılık grafik diyagramı adı verilen bir diyagram kullanılarak, nitelikler arasındaki bağımlılıklar görsel olarak ifade edilmektedir. Normalizasyon sürecinin devamı, bu bağımlılıkların tanımlanmasına bağlıdır. Diyagram elde edildikten sonra yönlü grafik matrisi tanımlanmaktadır. Ardından çalışmada belirtilen diğer adımlar da tamamlanarak tasarım BCNF'ye ulaştırılmaktadır (Bahmani, Naghibzadeh ve Bahmani, 2008). Bu çalışmanın devamı niteliğinde Bahmani ve diğ. (2010), otomatik veritabanı normalizasyonu için yeni algoritmalar kullanarak bir çalışma gerçekleştirmiştir. Bu çalışma, Bahmani, Naghibzadeh ve Bahmani (2008) tarafından sunulan algoritmanın, paralel algoritmalar ile güçlendirilerek uygulanmasını içermektedir (Bahmani ve diğ., 2010).

---

[3] URL: https://www.wolfram.com/mathematica [Erişim tarihi: 12.12.2016]
[4] URL: http://reference.wolfram.com/language/JLink/guide/JavaInterface.html [Erişim tarihi: 12.12.2016]

Dongare, Dhabe ve Deshbukh (2011), RDBNorma adında, 3NF'ye kadar ulaşabilen yarı otomatik bir normalizasyon aracı önermiştir. Yöntem, işlem başlamadan önce fonksiyonel bağımlılıkları ve ilişkilerin önsel olarak tanımlanmasını gerektirmektedir. RDBNorma, Micro'dan 2,89 kat daha hızlı ve 2,17 kat daha az sistem gereksinimi gerektirmektedir. Bu sebeple RDBNorma'nın Micro'dan daha iyi olduğu savunulmaktadır (Dongare, Dhabe ve Deshbukh, 2011).

Verma (2012) tarafından gerçekleştirilen çalışmada, ilişkisel veritabanları için geleneksel ve otomatik normalizasyon teknikleri incelenmiştir. Çalışma sonucunda otomatik normalizasyon çalışmalarının özellikle birincil anahtar belirleme anlamındaki başarısı vurgulanmıştır. Ayrıca geleneksel yöntemlerde, normal formlarla ortaya konulan kuralların hatırlanmasının gerekliliğinden ve bunun öğrenciler için zorlayıcı olduğundan bahsedilmiştir (Verma, 2012).

Ahmad, Saknakosnak ve Hooi (2014) tarafından gerçekleştirilen bir çalışmada Excel dosyasının veritabanına normalizasyon teknikleri kullanılarak dönüştürülmesi konu alınmaktadır. Önerilen sistemde bir Excel dosyası girdi olarak verilmekte ve kullanıcıdan nitelikler arasındaki fonksiyonel bağımlılıkları tanımlaması istenmektedir. Kullanıcının fonksiyonel bağımlılıkları hatalı olarak tanımlayabilecek olması, sistemin dezavantajı olarak görülmektedir. Hatalı tanımlanan fonksiyonel bağımlılıklar, veritabanının yanlış tasarlanmasına sebep olabilecektir (Ahmad, Saknakosnak ve Hooi, 2014). Niteliklerin ve bu nitelikler arasındaki fonksiyonel bağımlılıkların girdi olarak verilmesiyle, normalizasyonun gerçekleştirilebileceğine dair farklı çalışmalarla da karşılaşılmıştır (Sunitha ve Jaya, 2013; Demba, 2013).

Bahsedilen çalışmalarda önerilen yöntemler, normalizasyon sürecini gerçekleştirme konusunda tasarımcıya yardımcı olan ve süreci otomatik hale getirme odaklı yöntemlerdir. Ancak önerilen bu yöntemler, girdi olarak ilişkisel veritabanı tasarımında yer alacak niteliklerin yanında, bu nitelikler arasındaki fonksiyonel bağımlılıkları da gerektirmektedir. Bu durum bir risk etmenini oluşturmaktadır. Hatalı belirlenen bir fonksiyonel bağımlılık, Ahmad, Saknakosnak ve Hooi'nin (2014) de değindiği gibi yöntemin ilişkisel veritabanı tasarımını hatalı oluşturmasına sebep olabilir.

2.1.6. Varlıklar Arası İlişkiler

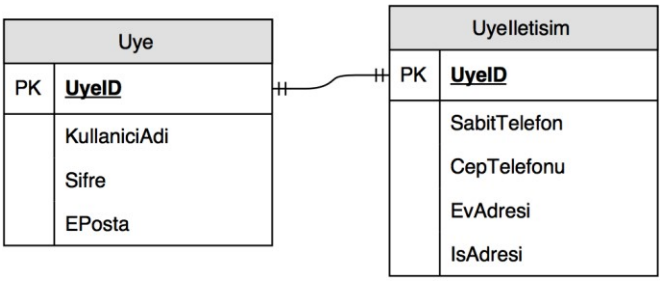
İlişkisel veritabanları, ham verilerin küçük tablolara bölünmesiyle elde edilmiş olsa da, daima göz önünde bulundurulması gereken en önemli koşul, veri bütünlüğü ve geçerliliğinin korunmasıdır. İyi yapılandırılmış bir veritabanında, istenilen veriye ulaşılabilmesi için varlıklar arasında ilişkiler kurulur (Hoffer, Ramesh ve Topi, 2016). Veritabanı içerisinde tutulan veriler, istenildiği anda verinin ilk halindeki forma tam doğrulukla geri döndürülebilmelidir. İlişkisel veritabanı tasarımı bu esas gözetilerek oluşturulmalıdır. Tasarım aşamasında, bu bütünlüğü sağlayan ilişkilerin kullanılması gerekmektedir. Tüm ikili varlıklar arasında bir ilişki tanımlanması zorunlu değildir ancak aynı veri setinden oluşturulan tüm varlıkların doğrudan ya da dolaylı olarak birbirleri ile ilişkili olması beklenebilir.

Veritabanı tasarımında varlık ve niteliklerin özellikleri gözetilerek ilişki türü tayin edilmelidir. Tasarım esnasında ilişki türünün doğru tayin edilmesi, veritabanının etkin işleyebilmesi için gereklidir.

İlişki türleri 1:1, 1:n ve m:n olarak üç gruba ayrılmaktadır. 1:1, her kaydın tek bir kayda karşılık geldiğini; 1:n, her bir kayda karşılık n tane kayıt olduğunu; m:n, her m tane kayda karşılık n tane kayıt olduğunu göstermektedir (Chen, 1976). İlişki türleriyle ilgili ayrıntılı bilgi alt başlıklarda sunulmaktadır.

2.1.6.1. 1:1 İlişi

Bire bir ilişki olarak adlandırılan ilişki türü, bir varlık içerisinde her bir kaydın, diğer varlıkta tek bir kayda denk gelmesi durumunda kullanılmaktadır. Şekil 2.4'te iki varlık görülmektedir. Her bir üye için tek bir iletişim kaydı tutulması, her bir iletişim kaydının da tek bir üyeye ait olması durumunda bu iki varlık arasındaki ilişki türünün bire bir olduğu söylenebilir.

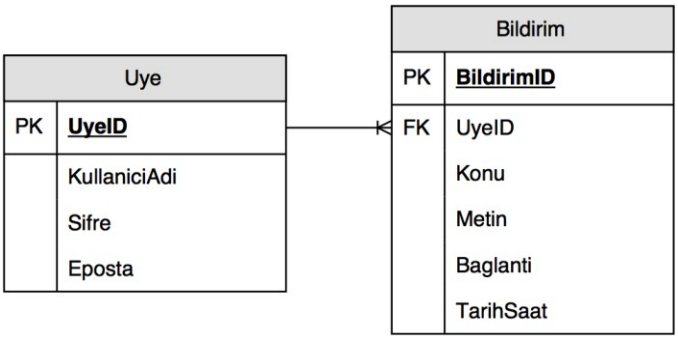


Şekil 2.4: 1-1 İlişki Türü Örneği.

Bire bir ilişki türü, diğer ilişki türlerine göre bir farklılık göstermektedir. Bu farklılık, bir varlığın kullanımdaki ihtiyaca bağlı olarak iki varlığa bölünmesi ve bunun sonucunda bire bir ilişki türüne sahip olmasıdır. Bu iki varlık birleştirilip tek varlık yapılabilir ve ilişkisel veritabanı özelliklerine göre herhangi bir aykırı duruma sebep olmazlar. Ancak kullanım aşamasında kolaylık sağlaması, yalnızca erişilmek istenen niteliklere erişilerek işlem yapılması v.b. amaçlarla bire bir ilişki türü kullanılmaktadır. Sistematik ya da bulunması zorunlu bir ilişki türü olarak görülmemelidir.

2.1.6.2. 1:n İlişki

1:n ilişki türü bire çok ilişki olarak adlandırılmaktadır. Bire çok ilişki türünde, birinci varlıktaki herhangi bir kayıt, ikinci varlıkta birden fazla kayda karşılık gelmekte, ancak ikinci varlıktaki herhangi bir kayıt birinci varlıkta yalnızca tek bir kayda karşılık gelmektedir.

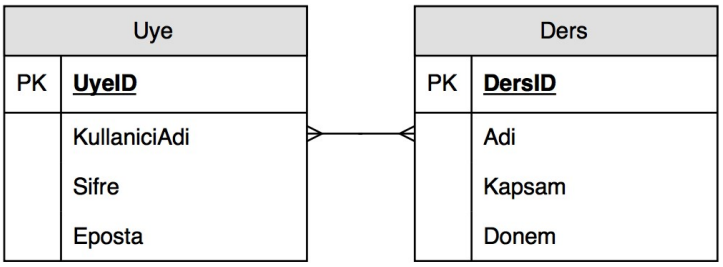


Şekil 2.5: 1:n İlişki Türü Örneği.

Şekil 2.5'teki örnekte *Uye* ve *Bildirim* varlıkları ve bunlara ait nitelikler görülmektedir. Bu iki nitelik arasında 1:n ilişki bulunmaktadır. Bir üye, birden fazla bildirime sahip olabilir ancak bir bildirim ancak bir kişiye ait olabilir. Varlıkların yerleri değiştirilseydi bu ilişkiye n:1 (çoğa bir) de denilebilirdi.

2.1.6.3. m:n İlişki

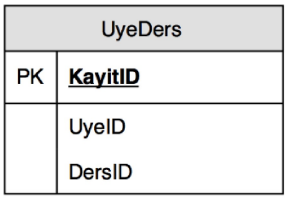
m:n ilişki türü, çoğa çok olarak adlandırılmaktadır. Bir varlık içerisindeki herhangi bir kayıt, diğer varlıkta birden fazla kayda denk gelebilir. Aynı durum diğer varlık için de geçerlidir.



Şekil 2.6: n:m İlişki Türü Örneği.

Şekil 2.6'da verilen örnekte *Uye* ve *Ders* varlıkları ve bunlara ait nitelikler görülmektedir. Bu örnekte, bir üye birden fazla derse kayıtlı olabilir. Ayrıca bir derse birden fazla üye kayıt olabilir. Bu durumda iki varlık arasında m:n ilişki olduğu söylenebilir.

Bu ilişki türünde bağlantı, kayıtların tutarlılığı sağlayabilmesi için Şekil 2.7'deki gibi üçüncü bir varlık ile sağlanır. Bu varlık hangi üyenin hangi derse kayıtlı olduğu bilgisini saklamaktadır. Üç tablo birleştirilerek sorgulandığında veri setine tam doğrulukla ulaşılabilir.

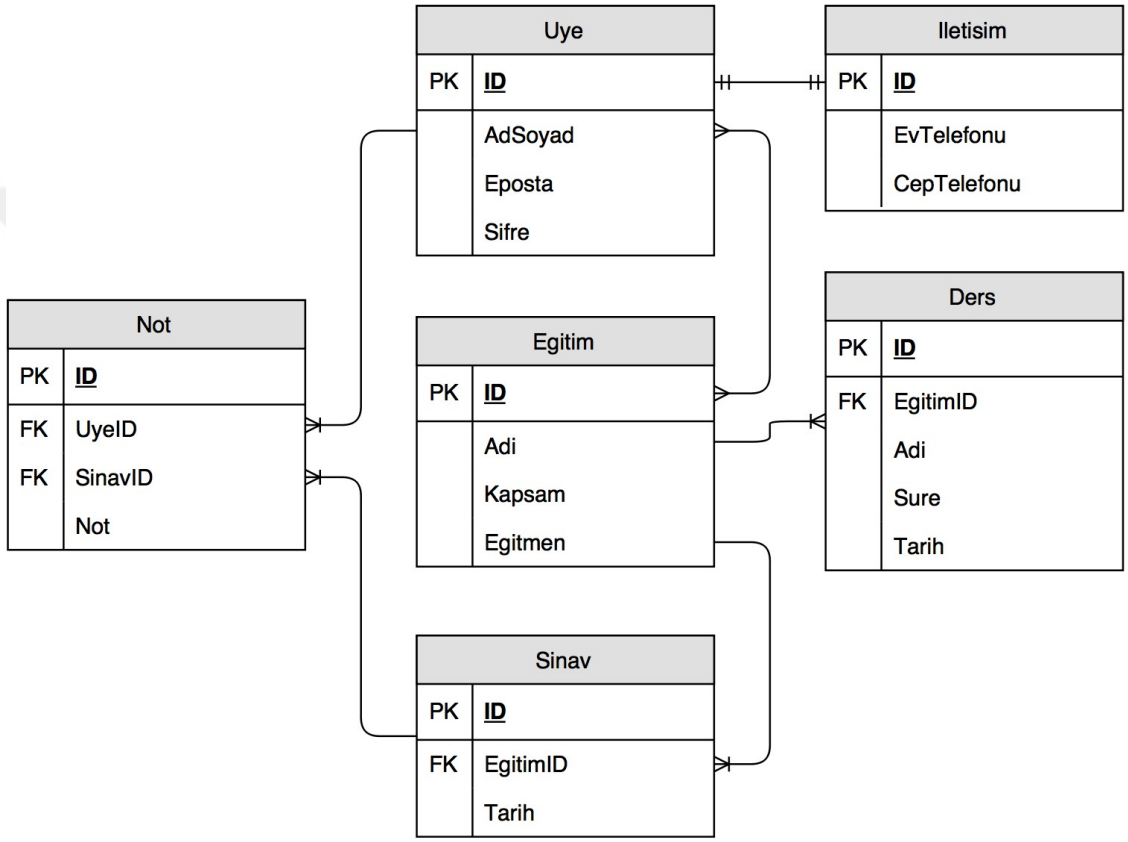


Şekil 2.7: n:m İlişki Türü İçin Bağlantı Varlığı Örneği.

2.1.7. Varlık-İlişki Diyagramları

Varlık-İlişki (ER: Entity-Relationship) modeli, Chen (1976) tarafından önerilmiş, önde gelen veri modellerinin avantajlarını taşıyan, varlık ve ilişkilerden meydana gelen ve gerçek dünyaya daha doğal bir yaklaşım oluşturan, küme ve ilişki teorisi temelli yüksek derecede veri bağımsızlığı sağlayan bir modeldir. ER diyagramları, bir ilişkisel veritabanı gösteriminin en basit, anlaşılır ve kolay yolu olmakla birlikte evrenseldir. İlişkisel veritabanı geliştirilirken farklı altyapılar kullanılabilir ancak ER diyagramı ile ifade edilen bir veritabanı, tüm veritabanı tasarımcıları tarafından rahatça anlaşılabilir

olacaktır. ER diyagramları için Chen, Bachman, Kaz Ayağı (crow's foot) veya Martin, IDEF1X gibi farklı standartlar mevcuttur (Bachman, 1969; Gogolla, 1991; Menzel ve Mayer, 1998; Purchase ve diğ., 2004; Nizam, 2011). Tez çalışması içerisinde Kaz Ayağı gösterimi kullanılmıştır. Tez metni içerisinde bu noktaya kadar kullanılan varlık gösterimleri birer ER diyagram parçası olmakla birlikte, tüm veritabanı yapısının ifade ediliş biçimi ile ilgili bir örnek Şekil 2.8'de verilmektedir.



Şekil 2.8: ER Diyagram Örneği (Kaz Ayağı).

Şekil 2.8'de, 6 varlık ve bunlara ait nitelikler görülmektedir. Ayrıca nitelikler arasındaki ilişkiler, ilişki türünü belirten çizgilerle ifade edilmektedir. Diyagramda, hepsi doğrudan olmasa da tüm varlıkların ilişkili olduğu görülebilir. Örneğin *Uye* ile *Ders* varlığı ilişkili değildir. Ancak bir dersin eğitime bağlı olması, üyenin de eğitime kayıt olması sebebiyle bu varlıklar dolaylı olarak ilişkili sayılabilir.

İlişkisel veritabanı tasarımının önce ER diyagramı olarak gerçekleştirilip sonrasında uygulanması, ortaya çıkabilecek hataların çok daha erken fark edilmesini sağlayabilir.

Ayrıca etkin bir veritabanı elde edebilmek için tasarımın tamamının somut bir şekilde görülebilmesi de tasarım sürecinin kalitesini arttıracak unsurlardan biridir.

2.1.8. Veritabanı Yönetim Sistemleri

Veritabanı Yönetim Sistemleri (Database Management Systems - DBMS), büyük miktarda verinin varlığını korumasına ve ondan faydalanılabilmesine yardımcı olmak için tasarlanmış yazılımlardır (Raghu ve Johannes, 2000). Veri ve veritabanı yönetiminin işletmelerdeki rolü, yüksek performanslı sistemlerin hızlıca hayata geçirilebilmesinde verinin kalitesinin korunabilmesi için güç sarf edilmesiyle birlikte değişime uğramaktadır (Hoffer, Ramesh ve Topi, 2016). İlişkisel veritabanları temelde aynı özelliklere sahiptir. Ancak VYS'ler, ilişkisel veritabanını güçlendirecek bazı özellikler ekleyerek kullanıma sunarlar. Bu sebeple VYS seçiminde yönetilecek verinin özellikleri önem taşımaktadır (Elmasri ve Navathe, 2003). MySQL[5], Microsoft SQL Server[6], Oracle Database[7] ve Microsoft Access[8] sık karşılaşılan VYS'ler içerisinde sayılabilir. Veritabanından etkin bir şekilde faydalanılabilmesi için ilişkisel veritabanı tasarımının yanında VYS seçiminin de büyük önemi bulunmaktadır. Bu sebeple kurulum aşamasında veritabanının kullanım amacının net bir şekilde belirlenmesi ve buna uygun VYS seçimi yapılması gerekmektedir.

2.2. GENETİK ALGORİTMALAR

Genetik algoritmalar (GA), doğal seçilim ve doğal genetik mekaniğine dayalı arama algoritmalarıdır (Goldberg, 1989). Adını, bu bilimlerdeki doğal seçilim, çaprazlama ve mutasyon olgularını taklit etmesinden dolayı almaktadır. John Holland (1975) tarafından ortaya çıkarılan GA fikri, 1960'lı ve 1970'li yıllarda kendisi ve Michigan Üniversitesi'ndeki öğrencileri tarafından geliştirilmiştir (Mitchell, 1995). Canlıların doğal evrimsel yaşamını örnek alarak çalışan bir sezgisel optimizasyon yöntemi olan GA, çözüm getireceği problemin kendisinden öte, elde ettiği aday çözümün uygunluk değeri ile ilgilenir ve çözüm aranılan problemden bağımsızdır (Satman, 2013). Klasik GA, ele alınan problem için yalnızca amaç fonksiyonuna ihtiyaç duymaktadır. GA tarafından üretilen çözüm önerisi amaç fonksiyona gönderilir, fonksiyondan dönen uygunluk değeri

[5] URL: http://www.mysql.com/about/ [Erişim tarihi: 09.11.2016]
[6] URL: https://www.microsoft.com/tr-tr/server-cloud/products/sql-server/overview.aspx [Erişim tarihi: 09.11.2016]
[7] URL: https://www.oracle.com/database/index.html [Erişim tarihi: 09.11.2016]
[8] URL: https://products.office.com/tr-tr/access [Erişim tarihi: 09.11.2016]

(ya da maliyet), çözüm önerisinin başarısını ifade eder. Bu özelliği dolayısıyla GA, Goldberg tarafından "kör" olarak nitelendirilmiştir (Goldberg, 1989).

GA, $x_i \in R^n$ ve $a, b \in R$ olmak üzere,

$$g_i(x_1, x_2, \dots x_n) \geq a$$
$$h_j(x_1, x_2, \dots x_n) = b \qquad 2.1$$

tüm g_i ve h_j kısıt fonksiyonlarını sağlayan,

$$f(x_1, x_2, \dots x_n) \qquad 2.2$$

amaç fonksiyonunu (goal function) optimize (minimize ya da maksimize) edebilmektedir. GA sayesinde elde edilen aday çözümler (candidate solution) kromozom (chromosome) olarak adlandırılmaktadır. Klasik GA'da, $x_i \in \{0, 1\}$ olmak üzere bir c kromozomu;

$$c = x_1 x_2 x_3 \dots x_n \qquad 2.3$$

olarak tanımlanmaktadır. Buradaki n değeri kromozom uzunluğunu göstermektedir. Bu şekilde tanımlanan kromozomlara ikili (binary) kromozom adı verilmektedir. İkili kromozoma bir örnek Şekil 2.9'da sunulmuştur.

1011010110001001

Şekil 2.9: Bir Kromozom Örneği.

Kromozomlar; ikili kodlamanın yanı sıra gray, gerçek sayılı, permütasyon, makine ve ağaç kodlamasıyla da oluşturulabilirler (Satman, 2016). Ayrıca ele alınan c kromozomunun $x \in Z$ veya $x \in R$ olarak tanımlanmasıyla da gerçekleştirilen GA çalışmaları mevcuttur (Wright, 1991; Jones ve diğ., 1997; Herrera ve diğ., 1998). Tez kapsamında $x_i \in \{0, 1\}$ için tanımlanan kromozom yapısı kullanılmıştır.

İkili kromozomlar, 2 sayı tabanında yazılmış sayılar olarak kabul edilebileceği için kromozomun değerinin 10 tabanında denk geldiği değer hesaplanabilmektedir. Bu sayede kromozom basitçe 10 tabanında bir tamsayıya dönüştürülebilmektedir. Kromozomun,

GA operatörlerini uygulamaya uygun olan 2 sayı tabanındaki durumuna genotip (genotype), karar değişkeni değerine dönüştürülmüş haline ise fenotip (fenotype) denilmektedir (Bekiroğlu, Dede ve Ayvaz, 2009). Örnek olarak Şekil 2.9'daki kromozomun 10 sayı tabanındaki değeri Şekil 2.10'da sunulduğu gibidir.

$$(1011010110001001)_2 = (46473)_{10}$$

Şekil 2.10: İkili Tabandaki Kromozomun Onluk Tabanda Değeri.

Ele alınan bir $f(x)$ problemi için seçilen kromozom uzunluğu, o problemin çözüm uzayını kapsamak durumundadır. Kromozom uzunluğu 5 seçilen bir GA aramasında karar değişkeninin alabileceği değerler $[0,\ldots,31]$ aralığında olabilmektedir (Şekil 2.11). Eğer ele alınan $f(x)$ probleminin çözüm uzayı $[-500,500]$ ise seçilen kromozom uzunluğu GA'nın istenilen çözüm uzayında yüksek performanslı arama yapmasına engel oluşturabilir.

$$(00000)_2 = (0)_{10}$$
$$(11111)_2 = (31)_{10}$$

Şekil 2.11: 5 Bitlik Bir Kromozomun Alabileceği Değerler.

Eğer arama yapılan uzay, gerçek sayılar kümesinin bir alt kümesi ise kromozomlar sayesinde elde edilebilen tamsayı değerleri istenilen çözüme ulaşmada yetersiz kalabilir. Bu durumda kromozom uzunluğunu arttırarak, aday çözüm ile elde edilen değerin gerçek sayılara dönüştürülebilmesini sağlayan bir uyarlama (mapping);

$$a + \frac{b-a}{2^L - 1} \times q \qquad 2.4$$

eşitliği kullanılarak gerçekleştirilebilir (Satman, 2008). Eşitlik 2.4'te, a ve b sırasıyla çözüm kümesinin alt ve üst sınırlarını, L kromozom uzunluğunu, q ise değeri hesaplanmak istenilen kromozomun 10 tabanında değerini ifade etmektedir.

Kromozom uzunluğu 10 olarak seçilen bir GA aramasında karar değişkeni için elde edilebilecek en küçük değer $(0)_2 = (0)_{10}$, en büyük değer ise $(1111111111)_2 = (1023)_{10}$'tür. Ele alınan problemin çözüm uzayının $[0,20]$ aralığında seçilmesi

durumunda rastgele seçilen 20 kromozomun 10 tabanındaki ve uyarlanmış değerleri Tablo 2.5'te sunulmuştur.

Tablo 2.5: Kromozomlar ve Değerleri.

Kromozom	10 Tabanında Değeri	Uyarlanmış Değeri
0001011011	91	1,7790811339
0010111010	186	3,6363636364
0101101011	363	7,0967741935
0110101101	429	8,3870967742
0111010110	470	9,1886608016
1000010111	535	10,4594330401
1001011010	602	11,7693059629
1001101011	619	12,1016617791
1001111011	635	12,4144672532
1011101011	747	14,6041055718
1011110111	759	14,8387096774
1011111001	761	14,8778103617
1011111110	766	14,9755620723
1100011011	795	15,5425219941
1100110111	823	16,0899315738
1100111011	827	16,1681329423
1110110000	944	18,4555229717
1110110010	946	18,4946236559
1111011100	988	19,3157380254
1111110000	1008	19,706744868

Çözüm kümesinin gerçek sayılar kümesinin bir alt kümesi olması durumunda kromozom uzunluğu, hassasiyeti doğru orantılı olarak etkilemektedir. Çözüm uzayının $[0, 20]$ aralığında seçilmesi ve kromozom uzunluğunun 10 seçilmesi durumunda, her bir aday çözüm değeri arasındaki fark en az;

$$\left(0 + \frac{20}{2^{10} - 1} x_n\right) - \left(0 + \frac{20}{2^{10} - 1} x_{n-1}\right)$$

$$= \frac{20}{1023} x_n - \frac{20}{1023} x_{n-1} \qquad 2.5$$

$$= 0{,}01955034213(x_n - x_{n-1})$$

$$= 0{,}01955034213$$

olarak hesaplanmaktadır. Aynı çözüm uzayında kromozom uzunluğunun 20 olarak belirlenmesi, aday çözümler arasındaki en az fark değerini;

$$\left(0+\frac{20}{2^{20}-1}x_n\right)-\left(0+\frac{20}{2^{20}-1}x_{n-1}\right)$$

$$=\frac{20}{1048575}x_n-\frac{20}{1048575}x_{n-1} \qquad 2.6$$

$$=0{,}0000190734(x_n-x_{n-1})$$

$$=\ 0{,}0000190734$$

olarak değiştirmektedir. En az fark arandığından değerlerin ardışık olabilmesi için;

$$x_n-x_{n-1}=1 \qquad 2.7$$

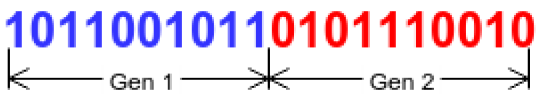
olarak kabul edilmiştir. Kromozom uzunluğunun 10'dan 20'ye çıkartılmasının, aday çözümlerin alabileceği değerlerin hassasiyetini büyük ölçüde arttırdığı eşitlik 2.5 ve 2.6'da elde edilen sonuçlar sayesinde görülebilmektedir. Kazandırdığı hassasiyete rağmen kromozom uzunluğunun arttırılması, daha fazla hesaplama gücü gerektireceğinden, optimum değere erişme süresini de arttırabilir.

Optimize edilmek istenilen f fonksiyonu birden fazla değişkene sahip bir fonksiyon olabilir. Ancak klasik GA için aday çözüm her zaman tek bir kromozom ile ifade edilmektedir. Bu sebeple kromozomun, f fonksiyonunun gerektirdiği sayıda karar değişkeninin değerini içeren bir yapıda olması gerekmektedir. Kromozomun, amaç fonksiyonunun her bir karar değişkeninin değerini içeren parçasına gen (gene) adı verilmektedir. Optimize edilmek istenilen amaç fonksiyonu, $f(x_1,x_2)$ gibi 2 değişken alan bir fonksiyon ise bu probleme uygun kromozom;

$$c=c_1c_2c_3c_4c_5c_6c_7c_8c_9c_{10}c_{11}c_{12}c_{13}c_{14}c_{15}c_{16}c_{17}c_{18}c_{19}c_{20} \qquad 2.8$$

olarak seçilebilir. Böylece 20 bit ile oluşturulmuş olan c kromozomu 10 bitten oluşan iki gruba ayrıldığında sırasıyla x_1 ve x_2 karar değişkenlerinin değerini veren genler elde edilebilmektedir. Karar değişkenlerinin $x_i \in [0,20]$ aralığında değer alması durumunda

bir c kromozomu örneği Şekil 2.12'de, örnek olarak sunulan kromozomlarla elde edilen değerler ise Tablo 2.6'da sunulmaktadır.



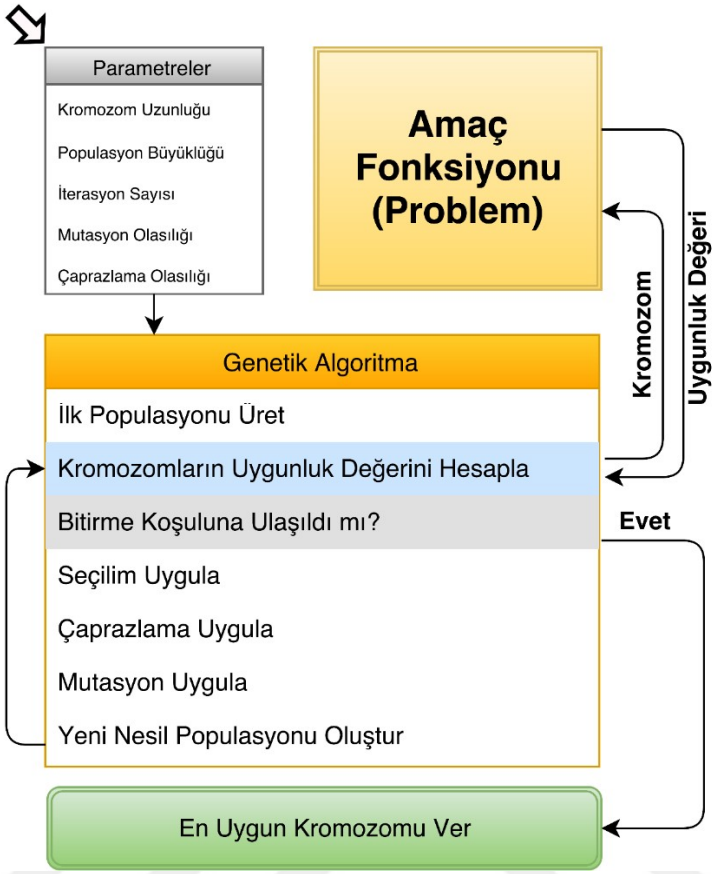
Şekil 2.12: 2 Genden Oluşan Bir Kromozom Örneği.

Tablo 2.6: 2 Genli Kromozomun Değere Dönüştürülmesi.

Gen Adı	Değişken	Gen	10 Tabanında Değeri	Uyarlanmış Değeri
Gen 1	x_1	1011001011	715	13,9784946237
Gen 2	x_2	0101110010	370	7,2336265885

Örnek olarak seçilen kromozom, $x_1 = 13{,}98; x_2 = 7{,}23$ değerlerini taşımaktadır.

Klasik GA aramasının işleyişi Şekil 2.13'te verilmiştir.



Şekil 2.13: Klasik GA İşleyişi.

Genetik arama öncesi belirlenen kromozom uzunluğu ve topluluk büyüklüğü parametrelerine göre ilk topluluk üretilir. Topluluk, her bir nesilde yer alan kromozomları ifade etmektedir. Oluşturulan topluluktaki tüm kromozomların başarı ya da maliyet değerleri, amaç fonksiyonu kullanılarak hesaplanır. Bitirme koşulu sağlanıyorsa genetik arama durdurularak en iyi çözüm elde edilir. Bitirme koşulu maksimum nesil sayısı, ulaşılmak istenilen hedef değer ya da probleme özgü olarak belirlenen başka bir kriter olabilir. Bitirme koşulu sağlanmadığı sürece, GA adımları tekrarlanır ve arama sonuçlanmaz. Bitirme koşulu sağlandığında elde edilmiş olan en iyi çözüm, genetik aramanın sonucu olarak kabul edilir.

Çözülmesi gereken problem karmaşık hale geldikçe kromozom yapısının da bu karmaşıklığa uygun tasarlanması gerekmektedir. Örneğin, bir labirent çözme problemi için kromozom yapısı, her iki bit bir gen olarak kabul edilerek oluşturulabilir. Tablo 2.7'de, böyle bir problem için kullanılabilecek olası tüm genler ve anlamları sunulmaktadır.

Tablo 2.7: Bir Labirent Problemi İçin Kullanılacak Genlerin Anlamları.

Gen	Anlamı
00	Yukarı / İleri
01	Sağa
10	Sola
11	Aşağı / Geri

Bu durumda kromozom, sayısal anlamda değil, mantıksal anlamında değerler içerecektir. Örnek olarak 20 bit ve 10 gene sahip

10000100000111100101

kromozomu, amaç fonksiyonu içerisinde "sol-ileri-sağ-ileri-ileri-sağ-geri-sol-sağ-sağ" değerini alacaktır. Amaç fonksiyonu, kromozomu bu anlama dönüştürdükten sonra çözülmesi amaçlanan labirent üzerinde test etmekte ve ulaşması beklenen duruma uzaklığına göre bir başarı ya da ceza puanı döndürmektedir. Ele alınan problem için hazırlanan amaç fonksiyonu başarı puanı döndürüyorsa maksimizasyon, ceza puanı döndürüyorsa minimizasyon işlemi gerçekleştiriyor denilebilir.

Kromozom yapısı olarak {0,1} alfabesinin kullanılması GA'nın çözebileceği problem çeşidini arttırmaktadır. Amaç fonksiyonu hazırlanabilen (kromozoma/çözüm önerisine karşılık bir başarı ya da maliyet puanı belirlenebilen) her türlü problem, GA ile çözülebilmektedir. Bir uygulama örneği olarak Haupt ve Haupt (2004), "Mary Had a Little Lamb" şarkısının notalarının GA ile belirlenmesi üzerinde durmuştur. Her bir nota 3 bit kullanılarak oluşturulmuş gen ile ifade edilmektedir (Tablo 2.8).

Tablo 2.8: Notaları İfade Eden Genler ve Karşılık Gelen Anlamları.

Gen	Anlam (Nota)
000	Sus
001	A
010	B
011	C
100	D
101	E
110	F
111	G

Tablo 2.8'de genlerin ifade ettiği notalar verilmiştir. GA, bu genlerden oluşan kromozomları, bir çözüm önerisi olarak sunmaktadır. Örnekte amaç fonksiyonu kullanıcının kendisidir. Birey, her bir kromozom sayesinde elde edilen nota dizisini dinler ve bir başarı puanı verir. İlgili çalışma içerisinde problemin başarıyla çözüme ulaştığı bilgisi ve ayrıntıları yer almaktadır.

2.2.1. Genetik Algoritma Operatörleri

GA'nın doğal seçilim sürecini taklit ederek sonuca ulaştığından daha önce bahsedilmişti. Klasik bir GA aramasında bu taklidin gerçekleştirilebilmesi için 3 operatör uygulanmaktadır. Bunlar; seçilim, çaprazlama ve mutasyon operatörleridir (Mitchell, 1995). Seçilim operatörü, hangi kromozomların sonraki nesillerde yer alacağının uygunluk değerine bağlı olarak belirlenmesini; çaprazlama ve mutasyon operatörleri ise yeni kromozom üretme sürecini gerçekleştirmektedir (Herrera, Lozano ve Verdegay, 1998). Bahsedilen temel operatörler alt başlıklar içerisinde incelenmiştir.

2.2.1.1. Seçilim Operatörü

Seçilim operatörü; doğadaki, ortama uyum sağlayanın hayatta kalması prensibinin GA içerisindeki işleyişini sağlamaktadır (Srinivas ve Patnaik, 1994). Klasik GA'da yeni nesil üretimi (reproduction), kromozomların uygunluk değerlerine göre belirlenen ağırlıklarıyla oluşturulmuş slotlara sahip bir rulet tekeri üzerinde doğrusal arama yapan seçilim fonksiyonu ile sağlanır (Goldberg, 1989). Seçilim operatörü, yeni kromozomlar oluşturabilmek için, topluluk içerisinden seçilen bir eşleşme havuzundan iki kromozomun seçilmesini sağlamaktadır (Haupt ve Haupt, 2004).

Literatürde birçok seçilim yöntemi bulunmaktadır. Bu yöntemler;

- Rulet tekeri (Roulette Wheel) ve Olasılıksal Evrensel (Stochastic Universal) seçme ile uygunluk-orantılı seçilim,
- Sigma derecelendirmesi (Sigma scaling),
- Boltzmann seçilimi (Boltzman Selection),
- Sıralandırma seçilimi (Rank selection),
- Turnuva seçilimi (Tournament selection),
- Sabit durum seçilimi (Steady-state selection)

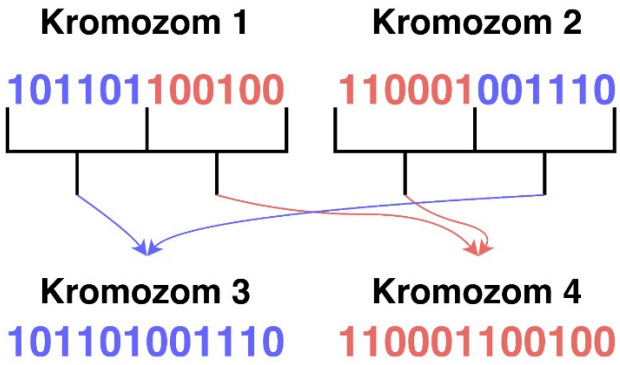
olarak sıralanmaktadır (Michelle, 1998).

Rulet tekeri ve turnuva seçilimi sık kullanılan iki seçilim yöntemidir ve en iyisinin hangisi olduğunu söylemek zordur (Haupt ve Haupt, 2004). Rulet tekeri seçiliminde tüm kromozomlar, uygunluk değerlerine oranla bir seçilme olasılıklarına sahiptir. Her bir kromozomun seçilme olasılığı, kromozomun uygunluk değerinin topluluktaki tüm kromozomların uygunluk değerleri toplamına oranıyla elde edilir. Seçilim, bu olasılıklar dahilinde rastgele olarak gerçekleştirilmektedir. Turnuva seçilimi ise topluluk içerisinden sabit sayıda kromozomun rastgele ele alınarak, uygunluk değeri en iyi olanın seçilmesi yöntemidir. İki yöntemde de toplulukta yer alan kötü uygunluk değerine sahip kromozomların da yeni nesilde bulunma olasılığı mevcuttur ancak düşüktür. Bu, çeşitliliğin oluşmasına fayda sağlayan önemli bir özelliktir.

Michelle (1998) tarafından, seçilim yöntemleri arasında gösterilen elitizm (elitism), güncel uygulamalarda seçilim operatöründen bağımsız bir operatör olarak ele alınmış ve uygulanmıştır. Elitizm operatörü, bir nesildeki en iyi çözümün sonraki nesile hiçbir değişikliğe uğramadan aktarılmasıyla çalışmaktadır. Elitizm operatörünün kullanılması tüm problemler için gerekli olmasa da bazı problemler için çok ciddi performans artışı sağlayabilmektedir (Simon, Ergezer ve Du, 2009).

2.2.1.2. Çaprazlama Operatörü

Çaprazlama, seçilim operatöründen sonra uygulanmaktadır. İşlem, topluluk içerisindeki kromozomların rastgele seçilen bir noktadan bölünerek çapraz olarak tekrar birleştirilmesiyle gerçekleştirilmektedir. Çaprazlama operatörünün uygulanma olasılığı genelde 0,5 ile 1 arasında bir değer almaktadır (Srinivas ve Patnaik, 1994). Süreç, ebeveyn kromozomlardan bir ya da daha fazla yavru kromozom üretmeyi sağlamaktadır (Haupt ve Haupt, 2004). Şekil 2.14 bir noktadan bölünme örneği sunulmaktadır.



Şekil 2.14: Bir Noktadan Çaprazlama Örneği.

Şekil 2.14'te, Kromozom 1 ve Kromozom 2 ebeveyn kromozomları ifade etmektedir. Ebeveyn kromozomlar, 6. karakter sonrasında bölünerek, iki yeni kromozom oluşturmuşlardır (Kromozom 3 ve 4). Ebeveyn kromozomların ve oluşturdukları yeni kromozomların onluk tabanda değerleri Tablo 2.9'da sunulmaktadır.

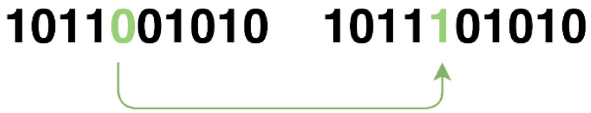
Tablo 2.9: Çaprazlama Sonrası Kromozomların 10 Tabanında Değerleri.

Kromozom Adı	Kromozom	10 Tabanında Değeri
Kromozom 1	101101100100	2916
Kromozom 2	110001001110	3150
Kromozom 3	101101001110	2894
Kromozom 4	110001100100	3172

Tablo 2.9'da görüldüğü gibi, çaprazlama öncesinde 2916 ve 3150 değerinde iki kromozom mevcutken, çaprazlama sayesinde 2894 ve 3172 değerlerine sahip iki yeni kromozom elde edilmiş olmaktadır. Bu sayede topluluk içerisinde daha önce ulaşılmayan değerde bir kromozom elde edilmiştir. Çaprazlama işlemi, kromozomun birden fazla noktadan bölünmesiyle de gerçekleştirilebilmektedir.

2.2.1.3. Mutasyon Operatörü

Mutasyon operatörü, kromozumu oluşturan bitlerin belirlenen oranda değiştirilmesiyle çalışmaktadır (Haupt ve Haupt, 2004). Değiştirmekten kasıt 0 ise 1, 1 ise 0 yapmaktır. Mutasyon operatörünün rolü aday çözümlerin, çözüm uzayının bir bölgesinde yoğunlaşmasını engelleyebilmek ve çözüm uzayının keşfedilmemiş kısmına da ulaşılmasını sağlamaktır (Srinivas ve Patnaik, 1994). Şekil 2.15'te bir kromozomun mutasyon operatörü uygulanmadan önceki ve sonraki anları gösterilmektedir.



Şekil 2.15: Mutasyon Örneği.

Şekilde kromozomun 5. biti, 0 iken 1 olarak değişmektedir. Böylece kromozomun 10 tabanındaki değeri 714 iken 746 olmuştur.

Mutasyon operatörünün uygulanma olasılığı genellikle 0,005 ile 0,05 değerleri arasında seçilmektedir. Bu oranın 0 olması bir sorunu beraberinde getirebilir (Srinivas ve Patnaik, 1994). Bu sorun, topluluğun çözüm uzayına yeterince iyi dağılamaması ya da n nesil sonra uzayın bir bölgesinde yoğunlaşmasıdır. Bu duruma yerel optimuma takılma (get stuck on a local optimum) adı verilir (Yuret ve Maza, 1993; Srinivas ve Patnaik, 1994). Böyle durumlara karşı, çözüm uzayının farklı yerlerine sıçrayış gerçekleştirmek gerekmektedir. Bu görev mutasyon operatörüne aittir. Mutasyon operatörünün uygulanma olasılığı 1'e yaklaştığında çok fazla bit terse döneceği için süreç, rastgele çözüm arama sürecine benzeyecektir (Srinivas ve Patnaik, 1994). Bu da GA'nın yarattığı avantajların kaybolması anlamına gelmektedir. Bu sebeple mutasyon olasılığı genelde oldukça düşük tutulmaktadır.

GA'nın karmaşık problemleri çözebilmesinde mutasyon operatörünün rolü bulunmaktadır. Ayrıca GA'nın başarısında seçilim, çaprazlama ve mutasyon operatörlerinin dengesi oldukça önem taşımaktadır (Mitchell, 1995).

Seçilim operatörü, çaprazlama ve mutasyon operatörleri sebebiyle değişken bir dengeye sahiptir. Bu dengeye keşif/faydalanma dengesi (the exploration/exploitation balance) adı verilmektedir (Mitchell, 1995). Tüm arama algoritmalarında arama uzayı üzerinde keşif/faydalanma belirlemesi yapılmalıdır. Keşif (exploration), bir arama uzayındaki yeni alanların tamamının ziyaret edilmesi süreci; faydalanma (exploitation) ise daha önce ziyaret edilmiş noktalara yakın bölgelerin ziyaret edilmesi sürecidir (Crepinsek, Liu ve Mernik, 2013).

2.2.2. Amaç Fonksiyonunun Oluşturulması

Önceki başlıklar GA'nın bir probleme çözüm araması sürecinde kullanılan GA bileşenlerini açıklamaktadır. Bu başlık altında ise problemin GA'ya tanıtılması işlemine

değinilecektir. Problem, GA'ya amaç fonksiyonu olarak tanımlanmalıdır. Optimize edilmek istenilen amaç fonksiyonu, her bir kromozomun değerlendirilmesinin sağlandığı mekanizmadır (Srinivas ve Patnaik, 1994).

GA tarafından üretilen her bir kromozom, aday çözüm niteliği taşımaktadır. Bu aday çözümlerin hangisinin ulaşılmak istenilen çözüm ya da çözüme en yakın olduğu amaç fonksiyonu ile belirlenebilmektedir. Amaç fonksiyonu, GA araması başlatılmadan önce optimize edilmek istenilen probleme göre yapılandırılır. Girdi parametresi olarak bir kromozom alarak, aldığı kromozom için ürettiği başarı ya da maliyet değerini döndürmektedir. Amaç fonksiyonu, her bir nesilde, toplulukta bulunan tüm kromozomlar için uygulanır. Optimizasyon işlemi, her bir kromozom için elde edilen değerin minimize ya da maksimize edilmeye çalışılmasıyla gerçekleştirilmektedir.

GA ile optimize edilmek istenilen problemlere farklı isimler verilebilmektedir. Amaç (objective) fonksiyonu, uygunluk (utility / profit) fonksiyonundan farklı olarak, çoğu problem için genellikle bir maliyet (cost) fonksiyonunun minimize edilmesi anlamına gelmektedir. Minimizasyon ve maksimizasyon problemleri arasında dönüşüm, değerin -1 ile çarpılmasıyla kolayca gerçekleştirilebilmektedir (Goldberg, 1989).

2.2.3. Kısıtların Belirlenmesi ve Uygulanması

Klasik yöntemde kısıtların kontrol edilememesi, GA için büyük bir zorluk oluşturmuştur (Michalewicz ve Janikow, 1991; Kowalczyk, 1997). Kısıtların belirlenmesi, çözüme ulaştırılmak istenilen problemin çözüm kümesinin doğru belirlenebilmesi için önemlidir. Kısıtlar sayesinde, çözüm uzayının sınırları belirlenebileceği gibi; çözüm uzayı içerisinde kalan ancak arama yapılmaması gereken bölgeler de tanımlanabilir.

Kısıt altındaki bir amaç fonksiyonu;

$$\min z = f(\vec{x}) \tag{2.9}$$

Kısıtlar: (2.10)

$$g_j(\vec{x}) \geq 0, \qquad j = 1, \ldots J,$$

$$h_k(\vec{x}) = 0, \qquad k = 1, \ldots, K,$$

$$x_i^l \leq x_i \leq x_i^u, \qquad i = 1, \ldots, n.$$

olarak tanımlanır. Burada $\vec{x}$, n boyutlu karar değişkeni vektörü, J bilinmeyenler vektörü sayısı, K ise eşitlik kısıtlarının sayısını göstermektedir. $f(\vec{x})$ amaç fonksiyonunu, $g_j(\vec{x})$, j. eşitsizlik kısıtını, $h_k(\vec{x})$ ise k. eşitlik kısıtını göstermektedir. i. değişken $[x_i^l, x_i^u]$ aralığında değer almaktadır (Deb, 2000).

Michalewicz ve Janikow (1991), çalışmasında kısıt altındaki $f(x_1, x_2, \ldots, x_q)$ fonksiyonunun sahip olabileceği tüm kısıtları;

1. **Alan kısıtları:** $i = 1,2, \ldots, q$ için $l_i \leq x_i \leq u_i$ kısıtı; $\vec{l} = \langle l_1, \ldots, l_q \rangle$, $\vec{u} = \langle u_1, \ldots, u_q \rangle$, $\vec{x} = \langle x_1, \ldots, x_q \rangle$ olmak üzere $\overrightarrow{l_i} \leq \overrightarrow{x_i} \leq \overrightarrow{u_i}$
2. **Eşitlikler:** $\vec{x} = \langle x_1, \ldots, x_q \rangle$, A= (a_{ij}), $\vec{b} = \langle b_1, \ldots, b_p \rangle$, $1 \leq i \leq p$, $1 \leq j \leq q$ ve p eşitsizlik sayısı olmak üzere $A\vec{x} = \vec{b}$
3. **Eşitsizlikler:** $\vec{x} = \langle x_1, \ldots, x_q \rangle$, $C = (c_{ij})$, $\vec{d} = \langle d_1, \ldots, d_m \rangle$, $1 \leq i \leq m$, $1 \leq j \leq q$ ve m eşitsizlik sayısı olmak üzere $C\vec{x} \leq \vec{d}$

olarak tanımlamıştır.

Kısıtlar, topluluğun uygun çözüm kümesi içerisinde kalmasını sağlayabilmektedir. Kısıtlı problemlerde optimum nokta genelde kısıtlarla oluşturulan sınırlar yakınındadır (Le Riche, Knopf-Lenoir ve Haftka, 1995). Dolayısıyla kısıt içeren problemlerde bu kısıtların tanımlanması, çözüme ulaşmada önem taşıyabilmektedir.

Kısıtların uygulanması ve kısıtlara uymayan kromozomlara uygulanacak işlemlerle ilgili çeşitli çalışmalar bulunmaktadır (Michalewicz ve Janikow, 1991; Kowalczyk, 1997; Deb, 2000; Coello ve Montes; 2002; Chootinan ve Chen, 2006; Li, Wang, Yang ve Cai, 2016).

Kısıtların uygulanmasında yaygın olarak; zorlama (Hard Constraint) ve cezalandırma (Soft Constraint) yöntemleri kullanılmaktadır (Arenas ve diğ., 2015). Zorlama yönteminde, uygun çözüm kümesi dışına çıkan bir aday çözümün, tekrar uygun çözüm

kümesi içerisine döndürülmesi sağlanmaktadır. Böylece topluluktaki kromozomlar tüm nesillerde uygun çözüm kümesi içinde kalmaktadır. Cezalandırma yöntemi ise bu konuda biraz daha esnektir. Bu yöntemde uygun çözüm kümesi dışına çıkan aday çözümler, bir ceza puanı alırlar (Orvosh ve Davis, 1994). Ceza puanı, aday çözümün sonraki nesillerde yer alabilme olasılığını oldukça düşürmektedir. Bu sayede doğal olarak uygun çözüm kümesi dışına çıkan aday çözümler genellikle topluluktan elenmektedir. Ancak düşük bir olasılıkla bu çözümler de yeni nesilde yer alabilir, hatta topluluk içerisinde daha başarılı aday çözümlerin elde edilmesine olanak sağlayabilirler. Bu sebeple aday çözümlerin, uygun çözüm kümesi dışına çıkmasına izin vermek, sonucun başarısına yardımcı olabilir (Orvosh ve Davis, 1994). Cezalandırma yöntemi, kısıtların uygulanmasında daha iyi bir çözüm gibi görülmektedir. Genel yaklaşım olarak da cezalandırmanın kullanıldığı görülmektedir (Le Riche, Knopf-Lenoir ve Haftka, 1995). Ayrıca zorlama yöntemini cezalandırma yöntemine dönüştürmek de mümkündür (Martinez ve Fages, 2015).

Arama işlemine başlamadan önce problem ile ilgili bilinen tüm kısıtların uygulanması zaman ve bilgisayar gücü maliyetini azaltacaktır. Ayrıca problemin modellenmesi esnasında belirlenemeyen kısıtlar, çözüm aranırken akıllı kısıt tanımlamasıyla belirlenebilir. Akıllı kısıt ile çözüm uzayında uygunluk değeri düşük bölgelerin belirlenmesi sağlanabilmektedir. Bu sayede uygulama esnasında çözüm uzayında bulunan başarısız alanlar kısıtlanabilir (Vipin, 1992).

2.2.4. Çok Amaçlı Genetik Algoritmalar

Bu bölüme kadar GA'nın tek bir amaç fonksiyonunu optimize etmesiyle ilgili bilgiler sunuldu. Fakat birleştirilerek bir fonksiyon olarak ifade edilemeyen birden fazla amaç fonksiyonuna sahip problemlerle karşılaşılabilmesi de mümkündür. Bu tarz problemlere çok amaçlı (multiobjective) veya çok kriterli (multicriteria) problem adı verilmektedir (Goldberg, 1989). Çok amaçlı optimizasyon problemlerinde, amaç fonksiyonları ayrı ayrı ele alındıklarında gerçekçi bir çözüme ulaşılamamaktadır (Ghosh ve Dehuri, 2004). Bu sebeple çok amaçlı optimizasyon problemlerinde farklı yaklaşımlar kullanılması gerekmektedir.

Kısıtlar;

$$g_j(x) \geq 0, \qquad j = 1,2,\dots,J;$$

$$h_k(x) = 0, \qquad k = 1,2,\dots,K; \tag{2.11}$$

$$x_i^{(L)} \leq x_i \leq x_i^{(U)}, \qquad i = 1,2,\dots,n$$

olarak belirlenen bir çok amaçlı optimizasyon probleminde amaç fonksiyonu;

$$\min \vee \max f_m(x_i), \qquad m = 1,2,\dots,M; \tag{2.12}$$

olarak gösterilmektedir. Burada x_i, $x \in R^n$ olmak üzere bir $x_i = (x_1, x_2, \dots, x_n)^T$ vektörünü ifade etmektedir (Deb, 2011).

Optimize edilmek istenilen problemin çok amaçlı optimizasyon problemi olması durumunda tüm amaç fonksiyonlarını aynı anda optimize eden çözümün aranması gerekmektedir. Amaç fonksiyonlarının birleştirilerek tek bir amaç fonksiyonu olarak ele alınabilmesi, problemin daha kolay çözülebilmesini sağlayabilir. Ancak bazı problemlerde tüm amaç fonksiyonları aynı çözüm önerisiyle optimize olmayabilir. Tüm amaç fonksiyonlarının aynı anda optimum çözüme ulaşmasına izin veren bir çözüm olmaması durumuna amaçların çatışması (objective conflict) adı verilmektedir (Hans, 1988). Bir amaç fonksiyonu için en iyiye ulaşırken başka bir amaç fonksiyonu daha kötü sonuç üretiyorsa burada amaçların çatışmasından söz edilebilir. Böyle durumlarda optimum çözümü bulmak daha karmaşık hale gelmektedir. Literatürde bu gibi durumlarda çözüme ulaşılabilmesi için önerilmiş yöntemler bulunmaktadır. Sık kullanılan çok amaçlı genetik algoritma yöntemleri;

- VEGA (Vector Evaluated Genetic Algorithm)
- Amaçları Ağırlıklandırarak Toplama Yöntemi (Weighted Sum Method)
- MOGA (Multi Objective Genetic Algorithm)
- NPGA (Niched-Pareto Genetic Algorithm)
- NSGA (Non-Dominated Sorting Genetic Algorithm)
- SPEA (Strengh Pareto Evolutionary Algorithm)
- NPGA 2
- NSGA-II

şeklinde sıralanabilir (Marler ve Arora, 2004; Taboada ve Coit, 2008).

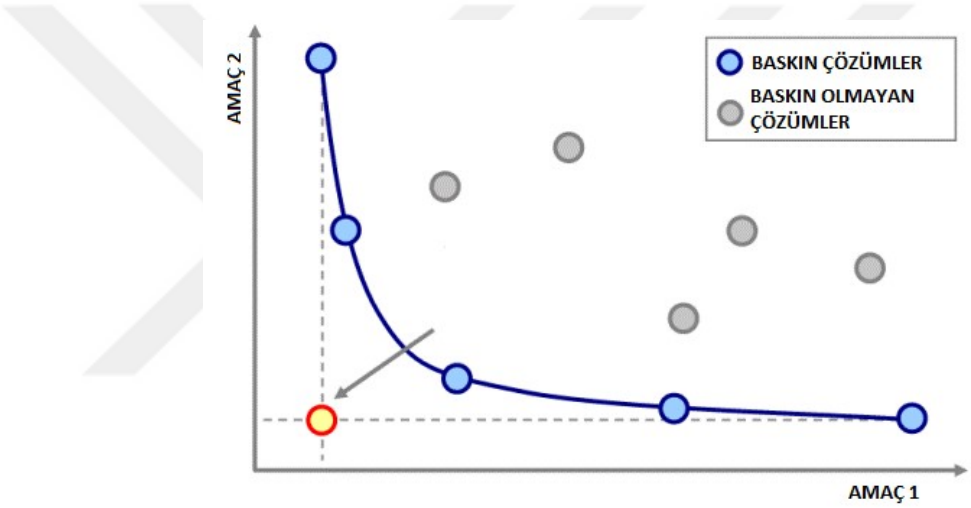
Literatürde karşılaşılan ilk çok amaçlı problem çözme yöntemi, Schaffer (1985) tarafından geliştirilen VEGA'dır. Schaffer, VEGA ile her amaç fonksiyonu için ayrı ayrı seçilim uygulayan bir model sunmuştur. Yöntemde, eşleşme havuzu n eşit parçaya bölünmektedir. i. parçası, i. amaç fonksiyonuna göre topluluktan rastgele seçilen kromozomlardan oluşur. Ardından, eşleşme havuzu birleştirilerek çaprazlama ve mutasyon uygulanır. Schaffer bu yöntemi uygunluk oranlı seçilim ile birlikte uygulamıştır (Zitzler ve Thiele, 1998). Topluluk boyutu N ise, her bir amaç için ele alınan alt toplulukların boyutu N/n kadar olmaktadır (Fonseca ve Fleming, 1993).

Ağırlıklandırarak toplama yöntemi (Weighted sum method), çok amaçlı optimizasyonda en temel, basit ve yaygın yöntemdir (Srinivas ve Deb, 1994; Marler ve Arora, 2004). Hajela ve Lin (1992) tarafından ortaya konulan bu yöntem, her bir amaç fonksiyonundan elde edilen değerin, bir katsayı ile çarpıldıktan sonra toplanarak skaler bir değer elde edilmesini kapsamaktadır. Elde edilmek istenilen değer, i tane amaç fonksiyonu olması durumunda; $0 < w_i < 1$ ve $\sum w_i = 1$ olmak üzere, $\sum w_i f_i(x)$ eşitliği ile hesaplanabilir (Zitzler ve Thiele, 1998). Bu, bir nevi amaç fonksiyonu sayısı indirgeme olarak görülebilir. Her bir amaç fonksiyonunun birbirinden bağımsız skaler değeri mevcutken ele alınan yöntemle amaç fonksiyonları birleştirilerek tek bir değer elde edilebilmektedir. Böylece GA, amaç fonksiyonlarının ağırlıklı toplamından oluşan tek bir skaler değeri optimize etmektedir. Ağırlıklandırarak toplama yöntemi, birbirlerine üstünlüğü olan amaç fonksiyonları içeren optimizasyon problemlerinde avantajlı bir yöntemdir (Srinivas ve Deb, 1994).

Ağırlıklandırarak toplama yönteminin bazı dezavantajları da bulunmaktadır. Öne çıkan 3 dezavantaj Srinivas ve Deb (1994) tarafından vurgulanmıştır. Birinci dezavantaj, ağırlıkların kesin olarak belirlenmesinin zor olmasıdır. Farklı ağırlıklar belirleyen bir kişi, aynı problem için çok daha farklı bir çözüme ulaşabilecektir. İkinci dezavantaj, bu yöntem ile başarılı olması muhtemel bir çözüm kümesi değil, tek bir çözüm elde edilebilmesidir. Ancak bu durum, bir çözüm kümesi yerine tek bir çözüme ulaşılmak istenildiğinde avantaj olarak görülebilir. Üçüncü dezavantaj ise, ağırlıkları tutarlı ve sürekli olarak değiştirmenin, Pareto optimal noktaların düzgün bir dağılımını ve Pareto

optimal kümenin isabetli/doğru ve tam bir temsilini mutlaka vereceği anlamına gelmemesidir.

Fonseca ve Fleming (1993), çok amaçlı GA'lar için MOGA adında, sıralama temelli bir uygunluk değerlendirme yöntemi önermiştir. Uygunluk değerlendirme yönteminin, harici bir karar verici gibi araya girerek değişiklik yapabilmesine izin verilmiştir. MOGA ile çözüm tek bir değer olarak değil, birbirinden daha iyi olmayan değerler kümesi olarak elde edilir. Çözüm önerileri birbirleriyle düşük, yüksek ve düşük değil olarak kıyaslanmaktadır. Her biri diğerinden daha kötü olmayan çözümler kümesi, Pareto-optimal küme olarak adlandırılmaktadır.



Şekil 2.16: Pareto Optimal Kümesi.

Robinson (2014) tarafından sunulan şeklin referans alınmasıyla oluşturulmuş olan Şekil 2.16, iki amaç fonksiyonlu bir minimizasyon problemi için ele alınan aday çözümleri sunmaktadır. Çizgiyle birleştirilmiş çözüm önerileri, elemanlarından hiçbiri birbirinden daha iyi olmayan bir pareto-optimal kümeyi göstermektedir. Bu küme içerisindeki tüm aday çözümler uygun çözüm olarak görülmekte, amaç fonksiyonlarının önceliği dikkate alınarak uygun çözüm seçimi yapılması gerekmektedir.

NPGA adı verilen yöntem, seçilim operatörü üzerinde iyileştirme yapılarak gerçekleştirilmiştir. Turnuva seçiliminde ele alınacak kromozom sayısı, bir parametre değeri olarak tanımlanmaktadır. Bu sayı en az 2 olarak seçilir. Turnuva seçilimi çok amaçlı optimizasyon probleminde kullanıldığında pareto baskın turnuvalar (Pareto

Dominated Tournaments) ya da baskın olmayan turnuva (Non-Dominated Tournament) ortaya çıkmaktadır (Horn, Nafpliotis ve Goldberg, 1994).

Turnuva seçiliminde ele alınan aday çözümler içerisinden, uygunluk/maliyet değeri optimuma daha yakın olan kromozom seçilmektedir. Ancak bu durum, amaç ve topluluktaki kromozom sayısının artması durumunda başarılı bir yöntem olmaktan çıkmaktadır. Bunun yerine, değerlendirilmek üzere seçilen iki kromozomun yanı sıra karşılaştırılabilmeleri için rastgele bir kromozom kümesi de referans olarak seçilmektedir. Değerlendirilecek iki kromozom, bu setteki kromozomlarla karşılaştırılmaktadır. Ele alınan kromozomlardan biri karşılaştırma kümesinden iyi değil, diğer kromozom iyi ise; ikinci kromozom sonraki nesile ulaşmaya hak kazanmaktadır. İkisinin de iyi olması ya da ikisinin de iyi olmaması durumunda paylaşım sürecine başvurulmaktadır (Horn, Nafpliotis ve Goldberg, 1994). Uygunluk paylaşımının amacı, topluluğu arama uzayındaki farklı yüksek bölgelere dağıtmaktır. Böylece her bir yüksek bölge, yüksekliğiyle orantılı miktarda topluluğa sahip olabilmektedir. Her bir yükselti, niş (niche) olarak da anılmaktadır. Turnuva seçiliminde ele alınan iki kromozomun ikisi de karşılaştırma kümesinden iyi ya da ikisi de kötüyse kromozomların nişleri kontrol edilir. Değerlendirmedeki iki kromozomdan hangisinin bulunduğu nişte daha az kromozom varsa, o kromozom turnuvayı kazanmış sayılmaktadır (Zitzler ve Thiele, 1998). Algoritma geliştirilerek NPGA 2 adıyla literatürde yer almıştır (Erickson, Mayer ve Horn, 2002; Taboada ve Coit, 2008).

Zitzler ve Thiele (1999) tarafından ortaya atılan SPEA yöntemi, topluluğun (P) dışında pareto-optimal çözümlerin bulunacağı harici bir topluluk (P') daha bulundurmak üzerine kuruludur. P'deki pareto-optimal çözümler, P' içerisine kopyalanmaktadırlar. Yöntemde uygunluk değeri, topluluk büyüklüğüyle orantılı bir kuvvet değeri yardımıyla hesaplanmaktadır. Zitzler, Laumanns ve Thiele (2001), SPEA yöntemini geliştirerek SPEA2 yöntemini literatüre katmıştır.

NSGA yöntemi, yalnızca seçilim operatörünün klasik GA seçilim operatöründen farklı çalışması ile ayrılmaktadır (Srinivas ve Deb, 1994). Bu yöntemde değerlendirme, birden fazla adımdan oluşmaktadır. Öncelikle pareto yüzeyindeki (pareto front) çözümler belirlenmekte, temsili olarak eşit uygunluk değeri almaktadır. Bu kromozomlar sürecin devamında dikkate alınmaz. Atanan temsili uygunluk değeri, oluşan yeni pareto

yüzeyinde bulunan en küçük uygunluk değerinden daha az olarak belirlenmektedir. Bu döngü, topluluktaki tüm bireylere uygulanana kadar devam etmekte ve bu sayede tüm topluluk sınıflandırılmış olmaktadır (Zitzler ve Thiele, 1998).

Deb (2002), NSGA'nın dezavantajlarına vurgu yaparak NSGA-II algoritmasını önermiştir. NSGA-II algoritmasında, başlangıç olarak rastgele bir P_0 topluluğu üretilir. Kromozomlar, Pareto-optimal durumlarına göre sınıflandırılır. Ardından kromozomlara, sınıflandırma seviyeleri ile orantılı bir sıra değeri, uygunluk değeri olarak atanır (en iyi: 1, daha az iyi: 2, ...). İkili turnuva seçilimi, çaprazlama ve mutasyon operatörleri uygulanarak N boyutlu Q_0 topluluğu oluşturulur. Elitizm uygulanır. t. jenerasyonda, P_t ve Q_t toplulukları birleştirilerek R_t topluluğu oluşturulur. Böylece R_t topluluğu $2N$ boyutuna sahip olur. R_t, pareto-optimal duruma göre sıralanır. En iyi pareto-optimal çözümleri kümesi, P_{t+1}. topluluğu oluşturur (Deb ve diğ., 2002).

3. MALZEME VE YÖNTEM

3.1. TEZİN AMACI VE ÖNEMİ

Ele alınan bir veri setine ait veri yapısının, ilişkisel veri modeline uygun hale getirilmesi veri yapısının sırasıyla normal formlara uygun hale getirilmesiyle gerçekleştirilebilmektedir (Başlık 2.1.4). Bir veri yapısının, seçilen bir normal formda sayılabilmesi için normal form ile belirlenen kurallara uygun olması gerekmektedir. Bu kurallar genellikle veritabanı tasarımcısı tarafından karar verilmesi gereken müdahaleler ile uygulanmaktadır. Bu durum, işlemin öznel bir süreç olmasına sebep olmakla birlikte, farklı tasarımcılar tarafından aynı veri seti için farklı tasarımlar elde edilmesine de sebep olabilmektedir. Sürecin tasarımcıdan bağımsız, tamamen kurallara bağlı olabilmesi, normalizasyon sürecinin otomatik hale getirilmesiyle mümkün olabilir.

Gerçekleştirilen literatür incelemesinde, otomatik normalizasyon çalışmalarıyla karşılaşılmıştır (Başlık 2.1.5). Bu çalışmalarda görülen ortak eksiklik, otomatik normalizasyon sürecinin gerçekleşebilmesi için fonksiyonel bağımlılıkların kullanıcı tarafından tanımlanmış olmasıdır. Bu durumda normalizasyon süreci; tanımlanan fonksiyonel bağımlılıklara, dolayısıyla tasarımcının kararlarına bağlı olarak gerçekleşecektir.

Gerçekleştirilen tez çalışmasında, tasarımcının etkisinin tamamen ortadan kaldırılması, bu sayede otomatik normalizasyonun yalnızca ele alınan ham veri setine bağlı olarak gerçekleştirilebilmesi amaçlanmıştır. Tez çalışması kapsamında önerilen algoritma, klasik yöntem olan normal formlarla değil, normalizasyon sürecinin bir optimizasyon problemi olarak ele alınmasıyla gerçekleştirilmiştir. Çalışmanın önemi; literatürde karşılaşılan otomatik normalizasyon çalışmaları göz önüne alındığında, tasarımcının karar ve etkisinin ortadan kaldırılması ve sürecin tam otomatik hale getirilmesi ile ortaya çıkmaktadır.

3.2. ÖNERİLEN ALGORİTMA

3.2.1. Problem

Bu doktora tezinde ele alınan problemin amacı, ilişkisel veritabanı hazırlama sürecinde normalizasyon işleminin öznel olarak gerçekleştirilmesinden kaynaklanabilecek hataları

yok edebilmektir. Bunu gerçekleştirebilmek için izlenilen yol, ilişkisel veritabanı normalizasyonunu bir optimizasyon problemi olarak ele almaktır. Eldeki bu optimizasyon problemi, birden fazla amaç fonksiyonunun aynı anda optimize edilmesini gerektirdiğinden çok amaçlı optimizasyon problemi olarak görülmüştür. Optimizasyon problemini oluşturan alt amaç fonksiyonları, Başlık 2.3.2 içerisinde ayrıntılı olarak sunulmuştur.

3.2.2. Problemin Alt Amaç Fonksiyonları

Problemin alt amaç fonksiyonları, literatürdeki normalizasyon tanımına ve deneyimsel çalışmalara dayalı olarak belirlenmiş ve alt başlıklarla sunulmuştur.

3.2.2.1. Hücre Sayısının En Az Olması

Birinci alt amaç fonksiyonu, ilişkisel veritabanı tasarımındaki toplam hücre sayısını minimize etmeye çalışmaktadır. Hücre sayısının minimize edilebilmesi için amaç fonksiyonu;

$$f_1 = \frac{\sum_{i=1}^{n} R_i A_i}{R_r A_r} \qquad 2.13$$

şeklinde tanımlanmıştır. Burada, $i = 1, \dots, n \in N$ olmak üzere n, aday çözümle oluşturulan ilişkisel veritabanı tasarımındaki toplam varlık sayısını; R_i, $i.$ varlıktaki toplam satır (kayıt) sayısını; R_r, ham veri setindeki toplam satır sayısını, A_i, $i.$ varlıktaki toplam nitelik sayısını; A_r, ham veri setindeki toplam nitelik sayısını belirtmektedir.

İfadenin pay kısmında, ele alınan aday çözümün içerdiği tüm varlıklardaki hücre sayılarının toplamı; paydada ise ham verideki toplam hücre sayısı hesaplanmaktadır. İfadenin payda kısmı, amaç fonksiyonuyla elde edilecek değerin ham veri seti büyüklüğünden bağımsız olabilmesi için kullanılmıştır. Aday çözümdeki toplam hücre sayısı, ele alınan veriye göre çeşitlilik gösterebilir. Ancak aday çözümdeki toplam hücre sayısının ham verideki toplam hücre sayısına oranı, elde edilecek değerin 1 civarında olmasını sağlayabilmektedir.

3.2.2.2. Tekrarlı Verinin En Az Olması

İkinci alt amaç fonksiyonu, gereksiz tekrarlı verinin en aza indirilmesini amaçlamaktadır. Alt amaç fonksiyonu;

$$f_2 = \sum_{i=1}^{n} \frac{R_i - (R_d)_i}{R_i} \qquad 2.14$$

olarak tanımlanmıştır. Burada; $i = 1,..,n \in N$ olmak üzere n, aday çözümle oluşturulan ilişkisel veritabanı tasarımındaki toplam varlık sayısını; R_i, i. varlık içerisindeki toplam satır sayısını; $(R_d)_i$ ise i. varlıktaki toplam tekrarsız/benzersiz (distinct) satır sayısını vermektedir. Ele alınan bir j varlığının tamamen benzersiz kayıtlardan oluşması durumunda;

$$R_j = (R_d)_j \qquad 2.15$$

olacağı için,

$$\frac{R_j - (R_d)_j}{R_j} = 0 \qquad 2.16$$

olarak hesaplanacak, bu durumda da j varlığının f_2 eşitliğine kattığı değer 0 olacaktır. Eğer j varlığının içerisindeki kayıtların tamamı tek bir değer alıyorsa;

$$(R_d)_j = 1 \qquad 2.17$$

olacağı için,

$$\frac{R_j - 1}{R_j} \qquad 2.18$$

eşitliği elde edilecektir. Böyle bir durumda j varlığı için yapılan maliyet hesaplamasının 1'e yakın bir değer alması beklenebilir. Bu amaç fonksiyonundan dönecek değerin $[0, n)$ aralığında olması beklenmektedir.

3.2.2.3. Varlık Sayısının En Az Olması

İlk iki alt amaç fonksiyonunun etkisiyle, aday çözümler her bir niteliğin ayrı bir varlık içerisinde yer alması çözümünü içerebilmektedir. Bu durum, pratikte geçerli bir çözüm olamayabileceği için toplam varlık sayısını azaltmaya yönelik bir alt amaç fonksiyonu kullanılmıştır. Üçüncü amaç fonksiyonu,

$$f_3 = \frac{n}{2^m} \tag{2.19}$$

olarak tanımlanmıştır. Burada n, aday çözüm içerisindeki toplam varlık sayısını; m ise ham veri setindeki toplam nitelik sayısına bağlı olarak hesaplanan bir değeri göstermektedir. m değerinin hesaplanması başlık 2.3.4.1 içerisinde ayrıntılı olarak ele alınmıştır. 2^m ifadesi, ele alınan ham veri setine karşılık oluşturulabilecek en fazla varlık sayısını göstermektedir. Bu alt amaç fonksiyonu minimize edilmeye çalışılmaktadır.

3.2.2.4. Sayısal Varlık Sayısının En Az Olması

Sayısal değerler işaret edici nitelikte de kullanılabildiği için her durumda kategorik değişken olarak görülmesi hatalı sayılabilir. Örneğin 1000 kişinin bulunduğu bir varlık içerisinde yaş niteliği tekrarlı değerler almak zorundadır. Ancak bu niteliğin yeni bir varlık içerisine eklenip birincil anahtar özelliğinde bir ID niteliğiyle ifade edilmesi mantıklı bir çözüm değildir. Bu alt amaç fonksiyonu, sayısal varlık sayısını minimize etmeye çalışmaktadır. Tüm niteliklerine ait kayıtları sayısal veri içeren varlıklara sayısal varlık adı verilmiştir. Dördüncü amaç fonksiyonu f_4,

$$f_4 = \frac{n_N}{n} \tag{2.20}$$

olarak tanımlanmıştır. Burada n_N ifadesi aday çözüm içerisindeki sayısal varlık sayısını; n ise aday çözüm içerisindeki toplam varlık sayısını göstermektedir. Sayısal varlık sayısının minimize edilmek istenmesi sebebiyle bu amaç fonksiyonu da minimize edilmeye çalışılmaktadır.

3.2.2.5. Amaç Fonksiyonu

Problemin amaç fonksiyonu, alt amaçların ağırlıklandırılmış toplamlarıyla elde edilmektedir. Buna göre *F* amaç fonksiyonu,

$$0 < w_i < 1 \text{ ve } \sum_{i=1}^{4} w_i = 1 \qquad 2.21$$

olmak üzere,

$$F = \sum_{i=1}^{4} w_i f_i \qquad 2.22$$

olarak tanımlanmıştır. w_i, $i.$ amaç fonksiyonunun aldığı ağırlık değerini; f_i, $i.$ amaç fonksiyonundan dönen değeri ifade etmektedir.

3.2.3. Ağırlık Katsayılarının Belirlenmesi

Önerilen F amaç fonksiyonunu oluşturan alt amaç fonksiyonları eşit önemde olmayabileceği için eşit ağırlıkla toplamak hatalı çözüme sebep olabilecektir. Bu sebeple her bir alt amaç fonksiyonu için ağırlıkların belirlenmesi ihtiyacı ortaya çıkmıştır.

Çalışma içerisinde katsayıların belirlenebilmesine yönelik bir genetik arama süreci gerçekleştirilmiştir. Katsayıların aranması sürecinde aşağıda sunulan adımlar takip edilmiştir.

1. Tüm w_i ağırlıkları eşit kabul edilerek F fonksiyonu minimize edilmiştir. Hesaplanan F değeri için f_1, f_2, f_3 ve f_4 fonksiyonlarından dönen değerler kaydedilmiştir. F fonksiyonunun eşit ağırlıklı alt amaç fonksiyonları ile hesaplanması F_a, alt amaç fonksiyonları ise sırasıyla f_{1a}, f_{2a}, f_{3a} ve f_{4a} olarak ifade edilmektedir.
2. Ulaşılmak istenilen veritabanı tasarımı, bir aday çözüm gibi amaç fonksiyonu ile değerlendirilmiş ve f_1, f_2, f_3 ve f_4 fonksiyonlarından dönen değerler kaydedilmiştir. Bu değerler; amaç fonksiyonu için F_b, alt amaç fonksiyonları için sırasıyla f_{1b}, f_{2b}, f_{3b} ve f_{4b} olarak ifade edilmektedir.
3. Seçilen 20 veritabanı için F_a ve F_b değerleri hesaplanmıştır.
4. Tüm veritabanları için $F_a \geq F_b$ koşulunu sağlayacak, $i \in \{1,2,3,4\}$ için w_i değerleri aranmıştır.

Katsayıların aranması sürecinde en fazla sayıda veritabanı için $F_a \geq F_b$ koşulunu sağlayan w_i değerleri belirlenmeye çalışılmıştır. Katsayı arama işlemi GA ile gerçekleştirilmiştir. Aramayla ilgili ayrıntı ve bulgular başlık 4.1.1 altında sunulmuştur.

GA araması sonucunda elde edilen en uygun w_i değerleri;

$$w_1 = 0{,}33960899$$
$$w_2 = 0{,}02030116$$
$$w_3 = 0{,}60869628$$
$$w_4 = 0{,}03139356$$

olarak belirlenmiştir. Buna göre F amaç fonksiyonu;

$$F = 0{,}33960899 \times f_1 + 0{,}02030116 \times f_2 + 0{,}60869628 \times f_3 + 0{,}03139356 \times f_4 \quad (2.23)$$

olarak güncellenmiştir. Katsayılar, çalışma kapsamında ele alınan 20 farklı ilişkisel veritabanının kullanılmasıyla belirlenmiştir. Herhangi bir veri setinin ilişkisel veritabanına dönüştürülmesinde de bu katsayılar kullanılabilir. Ancak farklı veri setleri için katsayıların güncellenmesi gerekebilir.

3.2.4. GA Parametrelerinin Ayarlanması

3.2.4.1. Kromozom Uzunluğu

GA araması başlatılmadan önce bazı parametrelerinin ayarlanmasını gerekmektedir. Bu parametrelerden en önemlisi kromozom uzunluğudur. Her bir kromozomun, ele alınan problem için bir ilişkisel veritabanı tasarımı örneği (aday çözüm) içerdiği belirtilmişti. Bu sebeple kromozom yapısı bir ilişkisel veritabanı tasarımını temsil edebilecek nitelikte olmalıdır. Önerilen algoritmanın farklı ham veri setleri tarafından kullanılabilir olabilmesi için kromozom uzunluğunun parametrik olarak belirlenmesi gerekmektedir.

İlişkisel veritabanı tasarımı temelde niteliklerin varlıklara dağılımını gösterdiği için kromozomun tüm niteliklerin yer alacağı varlıkları ifade etmesi beklenmiştir. Bu sebeple kromozom uzunluğu, ham veri setindeki toplam nitelik sayısı ile orantılı olarak belirlenmektedir.

Kromozom, ham veri setindeki nitelik sayısı kadar genden oluşmalıdır. Her bir gen, sırasıyla bir niteliği; gen içerisinde taşınan bilgi ise niteliğin yer alacağı varlığı ifade etmektedir. Kromozomun içerdiği gen sayısı, ham veri setindeki nitelik sayısı kadardır. Ele alınan nitelik sayısı arttıkça ilişkisel veritabanı tasarımındaki varlık sayısı da artış gösterebileceği için geni oluşturan bit sayısı değişkendir. Bu sayı hesaplanan m değeri ile belirlenmektedir. Bu ayrıntılar gözetilerek tanımlanan kromozom uzunluğu hesaplaması aşağıdaki gibidir.

HVS, ham veri setindeki nitelik sayısı olmak üzere m değişkeni;

$$HVS = 2^m$$
$$m = \log_2 HVS \qquad 2.24$$

olarak hesaplanmaktadır. Buna göre kromozom uzunluğu (KU);

$$KU = \lceil m \rceil \; HVS \qquad 2.25$$

olarak tanımlanmıştır. Eşitlikte m değişkeninin karşılığı hesaba katıldığında;

$$KU = \lceil log_2 HVS \rceil \; HVS \qquad 2.26$$

eşitliği elde edilmektedir. Daha önceden bahsedildiği ve eşitlikten de görüleceği üzere kromozom uzunluğu, HVS'ye bağlı olarak hesaplanan bir değere sahiptir.

3.2.4.2. Topluluk Büyüklüğü

Çalışmada gerçekleştirilen testlerde topluluk büyüklüğü 100 olarak seçilmiştir ancak ham verideki nitelik sayısının artması durumunda çözüm uzayına iyi dağılım yapılabilmesi için topluluk sayısının arttırılması gerekebilir.

3.2.4.3. Nesil Sayısı

Yapılan testlerde en iyi çözüme 1000'den az nesilde erişildiği için maksimum nesil sayısı 1000 olarak belirlenmiştir. Nesil sayısı, ele alınan ham veri setindeki nitelik sayısıyla doğru orantılı olarak artmaktadır. Bu sebeple nesil sayısını nitelik sayısına bağlı olarak

tayin etmek sonuca ulaşmada fayda sağlayabilir. Nitelik sayısına göre nesil sayısının değişimi başlık 4.1.2 altında sunulmuştur.

3.2.4.4. Mutasyon Olasılığı

Çalışmada standart GA mutasyon operatörü yetersiz geldiği için yeni bir akıllı mutasyon operatörü tasarlanmıştır. Operatörün çalışma olasılığı $p_m = 0{,}1$ seçilmiştir. Geliştirilen akıllı mutasyon operatörünün algoritması ve performansı ile ilgili bilgiler Başlık 2.3.5 altında sunulmuştur.

3.2.4.5. Çaprazlama Olasılığı

Çaprazlama olasılığı için R'nin (R Core Team, 2016) GA paketinin (Scrucca, 2013; Scrucca, 2016) varsayılan çaprazlama olasılığı değeri olan $p_c = 0{,}8$ kullanılmıştır.

3.2.5. Akıllı Mutasyon Operatörü

Standart GA mutasyon operatörü, bir kromozomdaki bir bitin belirli olasılıklar dahilinde tersine çevrilmesi işlemidir. Ancak bu mutasyon operatörü, önerilen algoritma için yeterli görülmemiştir. Algoritmayla oluşturulan kromozom yapısında bitler yerine genlerde anlamlı bilgi taşınmaktadır. Bu sebeple bir bitin değiştirilmesi yüksek olasılıkla daha kötü bir kromozom üreteceği için bir akıllı mutasyon operatörü oluşturma ihtiyacı duyulmuştur.

Geliştirilen mutasyon operatörü kendi içerisinde, her biri yüzde 20 olasılıkla çalışabilecek 5 kırılım içermektedir. Ayrıca GA parametrelerinde mutasyon operatörünün uygulanma olasılığı $p_m = 0{,}1$ olarak ayarlanmıştır. Bu sayede mutasyonun uygulanmama durumu da dikkate alındığında her seferinde karşılaşılması olası 6 durum mevcuttur. Bu 6 durum ve karşılaşıldığında uygulanan işlemler aşağıdaki gibidir:

1. **Uygulanmama durumu:** Bu durumda mutasyon operatörü uygulanmaz.
2. **Standart mutasyon operatörünün uygulanması:** Bu durumda GA, standart mutasyon operatörünü uygulayarak bir biti tersine çevirir. Bu sayede standart mutasyon operatöründen de faydalanılmaktadır.
3. **Bir genin rastgele değiştirilmesi:** Kromozom genlere ayrıldıktan sonra rastgele seçilen bir gen silinerek yerine rastgele oluşturulan bir gen yerleştirilir. Bu sayede

veritabanı tasarımı sabit tutularak bir nitelik rastgele başka bir varlık içerisine eklenmektedir.

4. **İki genin yer değiştirmesi:** Bir kromozom içerisinde rastgele belirlenen iki gen yer değiştirilir. Bu sayede veritabanı tasarımında mevcut olan ve rastgele seçilen iki nitelik yer değiştirmektedir.
5. **Bir genin diğeri üzerine kopyalanması:** Bir kromozom içerisinde rastgele seçilen bir gen kopyalanarak, yine rastgele seçilen bir gen üzerine yazılır. Bu sayede bir nitelik, rastgele seçilen başka bir niteliğin bulunduğu varlığa taşınmış olmaktadır.
6. **Gen havuzundan atama:** Bir kromozom içerisindeki tüm genler alınır ve tekilleştirilir (distinct). Bu sayede elde edilen gen havuzu kullanılarak rastgele seçilen genler ile yeni bir kromozom elde edilir. Böylece mevcut varlık sayısı değişmeden niteliklerin yerleri rastgele değiştirilmiş olmaktadır.

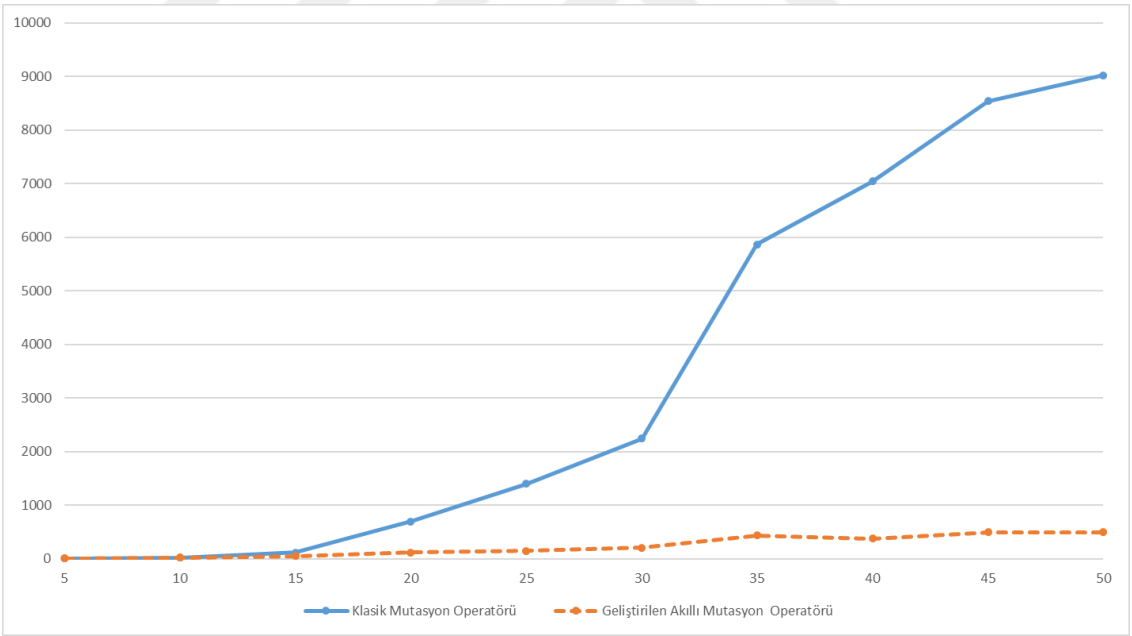
Geliştirilen akıllı mutasyon operatörü, performansının değerlendirilebilmesi için klasik GA mutasyon operatörüyle karşılaştırılmıştır. Karşılaştırma çalışması çeşitli miktarda nitelik sayısı kullanılarak gerçekleştirilmiştir. Her bir nitelik sayısı için 30 kez arama yapılmıştır. Çalışma sonuçları Tablo 2.10'da verilmiştir.

Tablo 3.1: Geliştirilen Akıllı Mutasyon Operatörünün Performansı.

HVS	Klasik Mutasyon Operatörü			Geliştirilen Akıllı Mutasyon Operatörü		
	Çözüme Ulaşma Sayısı	Nesil Sayısı Medyan Değeri	Nesil Sayısı MAD Değeri	Çözüme Ulaşma Sayısı	Nesil Sayısı Medyan Değeri	Nesil Sayısı MAD Değeri
5	30	5	1,4826	30	5	1,4826
10	30	24	14,0847	30	24,5	12,6021
15	30	115,5	71,9061	30	49	22,9803
20	30	698	316,5351	30	119	51,1497
25	30	1401	583,4031	30	150	40,7715
30	30	2244	482,5863	30	210,5	77,8365

35	29	5868	1282,449	30	438	123,0558
40	25	7048	1656,064	30	379,5	113,4189
45	23	8546	1530,043	30	494	135,6579
50	24	9026	1140,861	30	499	140,847

Tablo 2.10'da HVS, ham veri setindeki sütun sayısını belirtmektedir. Çözüme ulaşma sayısı, 30 denemenin kaç tanesinde başarılı çözüme ulaşıldığını göstermektedir. Nesil sayısının medyan değeri, başarılı şekilde çözüme ulaşan aramaların, kaçıncı nesilde çözüme ulaştıklarının medyanını sunmakta; nesil sayısının mad değeri sütunu ise, bahsedilen nesil sayılarının mad değerini vermektedir. Elde edilen sonuçların grafiği Şekil 2.17 ile sunulmaktadır.



Şekil 3.1: Geliştirilen Akıllı Mutasyon Operatörünün Performansının Grafiği

Şekil 2.17'de yatay eksen, arama yapılan HVS sayısını; düşey eksen ise başarıya ulaşılan nesil sayısını göstermektedir. Klasik mutasyon operatörünün performansı düz çizgiyle, geliştirilen akıllı mutasyon operatörünün performansı ise kesikli çizgiyle sunulmuştur.

Performans testinin sonuçlarından görüleceği üzere, geliştirilen akıllı mutasyon operatörü çalışma için klasik mutasyon operatörüne göre daha başarılı performans göstermiştir. Ayrıca yüksek HVS değerleri için klasik mutasyon operatörünün 10000 nesil ile yapılan aramalarda bile çözüme ulaşamadığı durumların olduğu görülebilmektedir (Tablo 2.10).

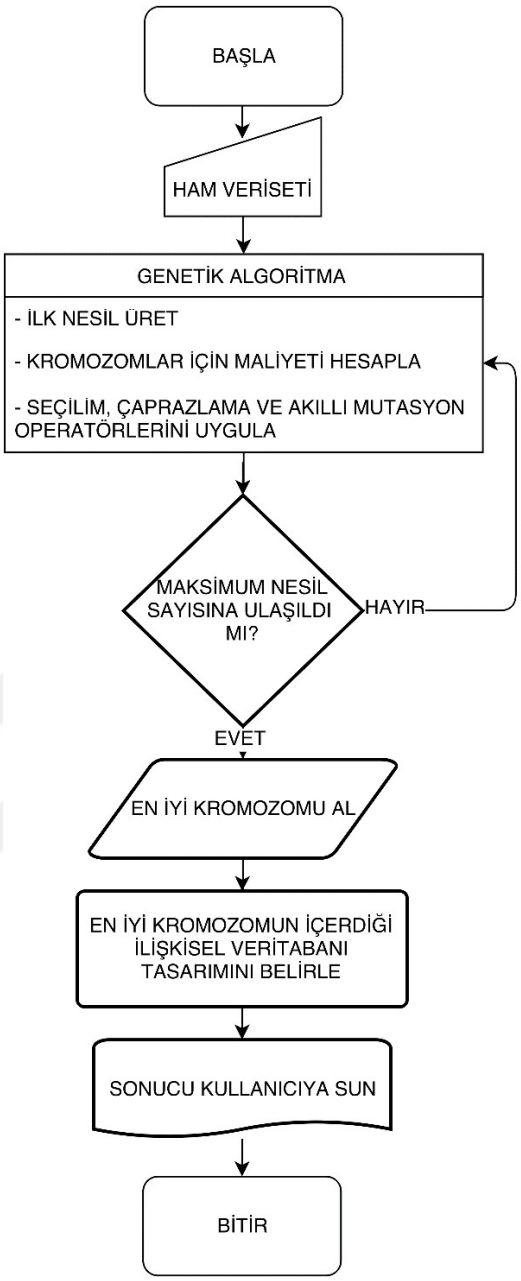
3.2.6. Algoritma

Tez çalışmasında, veritabanı normalizasyonunu otomatik olarak gerçekleştirebilecek bir algoritma önerilmektedir. Problemin çözümünde, bir sezgisel optimizasyon yöntemi olan GA kullanılmıştır. GA'nın temel kromozom tipi olan ikili (binary) kromozom yapısı tercih edilmiştir.

Algoritma adımları

1. Ham veri seti girdi olarak alınır,
2. Ham veri setindeki nitelik sayısına bağlı olarak GA parametreleri belirlenir,
3. Optimizasyon işlemini gerçekleştirmek için GA araması başlatılır.
    a. İlk nesil üretilir,
    b. Amaç fonksiyonu ile topluluktaki tüm kromozomlar için maliyet (cost) hesaplaması gerçekleştirilir,
    c. Seçilim, çaprazlama ve akıllı mutasyon operatörleri uygulanır,
    d. Maksimum nesil sayısına ulaşılmadıysa b adımına dönülür,
4. GA tarafından belirlenen, amaç fonksiyonunu minimize eden kromozom alınır,
5. Kromozom analiz edilerek içerdiği ilişkisel veritabanı önerisi belirlenir,
6. Sonuç kullanıcıya sunulur

olarak belirlenmiştir. Şekil 2.18'de sürecin işleyişi sunulmuştur.



Şekil 3.2: Otomatik Normalizasyon Süreci.

3.2.7. Örnek Uygulama

Bu başlık altında, geliştirici tarafından hazırlanan örnek bir ham veri setinin, önerilen algoritma ile ilişkisel veritabanına dönüştürülmesi süreci tüm ayrıntılarıyla birlikte sunulmuştur. Bu örnek uygulama, önerilen algoritmanın test edilmesinden çok nasıl çalıştığını uygulamalı anlatmak için gerçekleştirilmiştir.

3.2.7.1. Veri Seti

Kullanılacak ham veri seti Tablo 2.11'de sunulmaktadır.

Tablo 3.2: Örnek Ham Veri Seti.

Ad Soyad	Telefon	Kitap Adı	Sayfa Sayısı	Basım Yılı	Yayınevi
Emre Akadal	201	PHP	300	2000	Alfa
Emre Akadal	201	Java	400	2010	Kodlab
Emre Akadal	201	R	287	2009	Turkoglu
Emre Akadal	201	.NET	220	2012	Kodlab
Inayet Akadal	202	C++	376	2008	Papatya
Inayet Akadal	202	İstatistik	410	2010	Alfa
Inayet Akadal	202	R	287	2009	Turkoglu
Serra Celik	203	R	287	2009	Turkoglu
Serra Celik	203	İstatistik	410	2010	Alfa
Serra Celik	203	R	287	2009	Turkoglu
Serra Celik	203	Java	400	2010	Kodlab
M. Hakan Satman	204	İstatistik	410	2010	Alfa
M. Hakan Satman	204	R	287	2009	Turkoglu
M. Hakan Satman	204	Java	400	2010	Kodlab
Ahmet Olgun	205	Java	400	2010	Kodlab
Ahmet Olgun	205	.NET	220	2012	Kodlab
Oguz Saltik	206	PHP	300	2000	Alfa
Oguz Saltik	206	Java	400	2010	Kodlab
Oguz Saltik	206	R	287	2009	Turkoglu
Oguz Saltik	206	Istatistik	410	2010	Alfa
Sibel Senturk	207	IBE	107	2015	Papatya
Sibel Senturk	207	Istatistik	410	2010	Alfa
Sibel Senturk	207	PHP	300	2000	Alfa
Hakan Umut Akadal	208	PHP	300	2000	Alfa
Hakan Umut Akadal	208	R	287	2009	Turkoglu
Hakan Umut Akadal	208	Java	400	2010	Kodlab
Gokhan Satman	209	.NET	220	2012	Kodlab
Sefa Saylan	210	PHP	300	2000	Alfa
Sefa Saylan	210	Java	400	2010	Kodlab
Sefa Saylan	210	C++	376	2008	Papatya

Veri seti; 6 nitelik, 30 kayıttan oluşmaktadır. Nitelikler, kolaylık olması açısından Tablo 2.12'de gösterilen belirteçlerle ifade edilmiştir.

Tablo 3.3: Nitelik Belirteçleri.

Nitelik Adı	Kullanılacak Belirteç
Ad Soyad	v1
Telefon	v2

Kitap Adı	v3
Sayfa Sayısı	v4
Basım Yılı	v5
Yayınevi	v6

Önerilen algoritma, girdi olarak yalnızca ham veri setini gerektirmektedir. Ham veri seti girdi olarak belirlendikten sonraki adım GA'nın kullanacağı kromozom yapısının belirlenmesidir.

3.2.7.2. Kromozom Yapısı

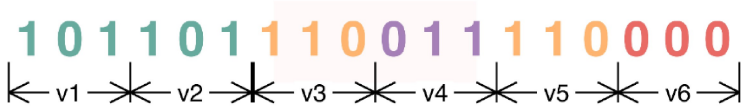
Kromozom uzunluğu, başlık 2.3.4.1'de tanımlanan eşitlik kullanıldığında;

$$\lceil log_2 HVS \rceil \; HVS = \lceil log_2 6 \rceil \; 6 = \; 18 \qquad 2.27$$

olarak hesaplanmaktadır. Eşitliğin sonucu, $HVS = 6$ olan bir veri seti için kromozom uzunluğunun 18 bit olarak seçilmesi gerektiğini göstermektedir. Her bir niteliğin yer alacağı varlığı belirten gen;

$$m = \lceil log_2 HVS \rceil = \lceil log_2 6 \rceil = 3 \qquad 2.28$$

bit ile temsil edilir. Şekil 2.19'da veri seti için bir kromozom örneği verilmektedir.



Şekil 3.3: Kromozomun Taşıdığı Anlam.

Şekil 2.19'daki kromozom, ele alınan veri seti için bir ilişkisel veritabanı tasarımı çözüm önerisini ifade etmektedir. Şekildeki kromozoma göre niteliklerin yer alacakları varlıklar Tablo 2.13'te sunulmuştur. Genin 10 tabanındaki değeri, niteliğin yer alacağı varlığı işaret etmektedir.

Tablo 3.4: Kromozoma Göre Niteliklerin Yer Alacakları Varlıklar

Niteliğin Belirteci	Gen	Genin 10 Tabanındaki Değeri
v1	101	5
v2	101	5
v3	110	6

v4	011	3
v5	110	6
v6	000	0

Kromozomun içerdiği ilişkisel veritabanı tasarımı çözümü 4 varlıktan oluşmaktadır (0, 3, 5, 6). Kromozoma göre 0 numaralı varlık v6'yı; 3 numaralı varlık v4'ü; 5 numaralı varlık v1 ve v2'yi; 6 numaralı varlık v3 ve v5'i içermektedir.

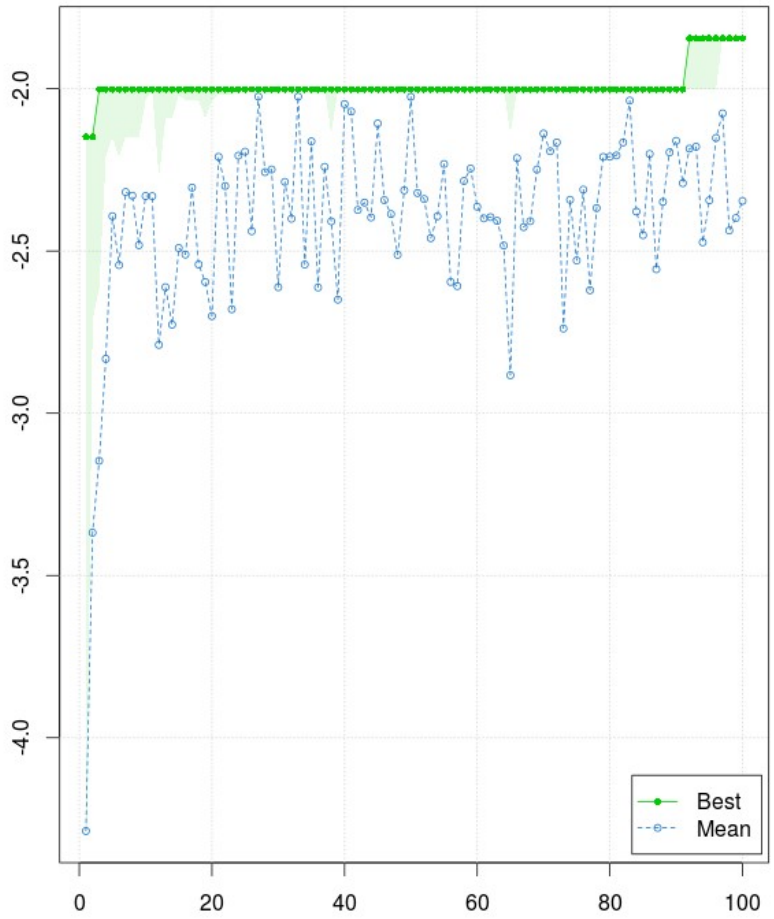
3.2.7.3. Algoritmanın Uygulanması

Örnek olarak ele alınan ham veri seti için bir otomatik normalizasyon uygulaması gerçekleştirilmiştir. Uygulama için kromozom uzunluğu 18, topluluk büyüklüğü 20, nesil sayısı 100, mutasyon olasılığı $p_m = 0{,}1$ ve çaprazlama olasılığı $p_c = 0{,}8$ olarak tanımlanmıştır.

Genetik arama sonucunda en iyi çözümü veren ve uygunluk değeri 1,844841 olan,

000000100100100011

kromozomu elde edilmiştir. Uygulamada kullanılan R paketi GA, optimizasyon tipini varsayılan olarak maksimizasyon kabul etmektedir. Bu sebeple minimizasyon tipli amaç fonksiyonunun negatifi kullanılmıştır. Kromozomların amaç fonksiyonu ile elde edilen maliyet değerlerinin nesil sayısına göre değişimi Şekil 2.20'de verilmektedir.



Şekil 3.4: Örnek Veri İçin Uygunluk Değeri Değişimi.

Elde edilen en uygun kromozom incelenmek istenildiğinde; $m = 3$ olması sebebiyle kromozom, 3 bitlik genlere ayrılarak her bir değişkenin yer alacağı varlıklar belirlenmelidir. Kromozomun içerdiği bilgi Tablo 2.14 ile sunulmuştur.

$$Kromozom: 000 - 000 - 100 - 100 - 100 - 011$$

Tablo 3.5: Elde Edilen Kromozomun Taşıdığı Anlam.

Niteliğin Belirteci	Gen	Genin 10 Tabanındaki Değeri
v1	000	0
v2	000	0
v3	100	4
v4	100	4
v5	100	4
v6	011	3

Buna göre tablo yapılanması Tablo 2.15'teki gibi olacaktır.

Tablo 3.6: Çözüm Önerisi – Belirteçlerle.

Varlık 0	v1, v2
Varlık 4	v3, v4, v5
Varlık 3	v6

Belirteçler kullanıldığında yapılanma Tablo 2.16'da verildiği gibi olmaktadır.

Tablo 3.7: Çözüm Önerisi.

Varlık 0	Ad Soyad, Telefon
Varlık 4	Kitap Adı, Sayfa Sayısı, Basım Yılı
Varlık 3	Yayınevi

Elde edilen çözüm önerisinin diyagramı Şekil 2.21 ile sunulmuştur.

Şekil 3.5: Çözüm Önerisinin Diyagramı.

Buna göre, örnek ham veriye karşılık elde edilen çözüm önerisi 3 varlıktan oluşmaktadır. Çözüm incelendiğinde gerçekten de ham veri için oluşturulabilecek en uygun çözüm olduğu gözlenebilmektedir.

3.3. GELİŞTİRME ORTAMI

3.3.1. Donanım

Kullanılan bilgisayar Intel Core i7-2600 CPU @ 3.40 GHz x 8 işlemciye ve 8 GB RAM belleğe sahiptir. Ayrıca NVIDIA Corporation GT216 (GeForce 210) ekran kartı bulunmaktadır.

3.3.2. Yazılım

Uygulama Linux işletim sisteminin Ubuntu dağıtımının 14.04 LTS x64 sürümü üzerinde gerçekleştirilmiştir. Uygulama için geliştirme dili olarak R tercih edilmiştir. R, istatistiksel hesaplama ve görselleştirme için kullanılan bir özgür yazılım geliştirme ortamıdır. Unix, Windows ve MacOS gibi çeşitli ortamlarda derlenebilir ve çalıştırılabilir (R Core Team, 2016). R'nin, GA için yazılmış paketler bulundurması, kullanım kolaylığı ve bilimsel çalışmalarda yaygın olarak kullanılması tercih edilmesinde rol oynamıştır.

Geliştirme ortamı olarak, R için özel bir editör olan RStudio'nun 0.99.441 versiyonu kullanılmıştır (RStudio Team, 2015). Uygulama gerçekleştirilirken R dili için geliştirilmiş olan *GA*, *doParallel* ve *sqldf* paketleri kullanılmıştır. *GA* paketi (Scrucca, 2013; Scrucca, 2016) GA aramaları gerçekleştirilebilmesini, *doParallel* (Revolution Analytics ve Wetson, 2015) paketi hesaplamanın işlemcinin çekirdeklerine dağıtılmasıyla hesaplama sürecini hızlandırılabilmesini, *sqldf* paketi (Grothendieck, 2014) ise herhangi bir veritabanı entegrasyonu gerektirmeden veri üzerinde doğrudan SQL komutlarıyla işlem yapılabilmesini sağlamaktadır.

3.4. YÖNTEM

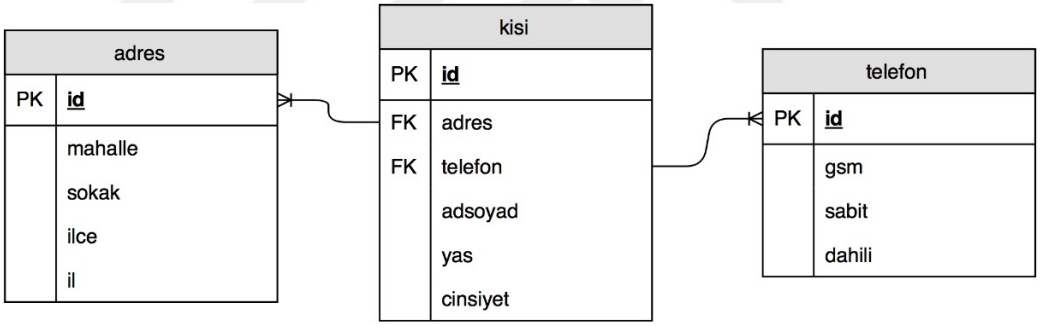
Önerilen algoritmanın başarısının ölçülebilmesi için bir ham veri setine karşılık gelen en iyi ilişkisel veritabanı tasarımının bilinmesi gerekmektedir. Başarı değerlendirmesi, algoritma tarafından sunulan çözümün en iyi çözüme benzerliği ile gerçekleştirilebilir. Ancak gerçekleştirilen literatür taramasında bir ham veri setine karşılık gelen en iyi ilişkisel veritabanı tasarımını gösteren bir çalışma ile karşılaşılmamıştır. Bu sebeple algoritmanın değerlendirilebilmesi için ilişkisel veritabanı tasarımları ele alınarak veri üretme süreci gerçekleştirilmiştir. Değerlendirme, önerilen algoritmanın, üretilen ham veri setine karşılık ilk durumdaki ilişkisel veritabanı tasarımını oluşturma başarısı ile sağlanmıştır. Süreç, alt başlıklar ile ayrıntılı olarak sunulmuştur.

3.4.1. Veritabanı Tasarımlarının Seçimi

Önerilen algoritmanın test edilebilmesi için 20 adet ilişkisel veritabanı tasarımı seçilmiştir. Tasarımların ham veri setine dönüşüm sürecinin kolay, hatasız ve gerçekçi olabilmesi için yıldız şema (star schema) tasarımlar tercih edilmiştir. Tasarımlar creately.com alan adlı internet sitesinden alınmıştır. İnternet sitesinde sunulan ilişkisel veritabanı tasarımları çeşitli akademik çalışmalarda kullanılmıştır (Satish, Sheeba ve Rangarajan, 2013; Kollar ve Sterbinszky, 2014; Ortuno, 2014; Khan, 2015; Herradi ve diğ., 2015).

3.4.2. Veri Üretme Süreci

Algoritmanın test edilmesi sürecinde kullanılacak ham veri setlerinin elde edilebilmesi için uygulanan ilk adım, ele alınan veritabanı tasarımlarına uygun veritabanlarının oluşturulması olmuştur. Tasarımlara uygun veritabanları hazırlandıktan sonra üretilen veri, varlıklara kayıt olarak eklenmiştir. Elde edilen varlıklar, tasarımda bulunan ilişkiler kullanılarak sorgulanmıştır.



Şekil 3.6: Veri Üretme Süreci İçin Örnek Bir İlişkisel Veritabanı Tasarımı.

Şekil 3.1'de veri üretme sürecine örnek gösterebilmek amacıyla bir ilişkisel veritabanı tasarımı sunulmuştur. Ele alınan veritabanı için;

```
SELECT adsoyad, yas, cinsiyet, mahalle, sokak, ilce, il,
gsm, sabit, dahili
FROM kisi, adres, telefon
WHERE kisi.adres = adres.id
AND kisi.telefon = telefon.id
```

SQL sorgusu uygulandığında çıktı Tablo 3.1'deki gibi olmaktadır.

Tablo 3.8: Veri Üretme Süreciyle Elde Edilen Bir Örnek Ham Veri Seti.

adsoyad	yas	cinsiyet	Mahalle	sokak	ilce	il	gsm	sabit	dahili
..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..

Sorgu sonucunda veritabanında bulunan tüm varlıklar içerisindeki nitelikler, tek bir elektronik tabloya dönüşmüştür. Elde edilen dosya, çalışma için ham veri seti niteliği taşımaktadır. Böylece Tablo 3.1'deki ham veri seti, ilişkisel veritabanı formuna ulaştırılmak istenildiğinde, ulaşılması gereken tasarımın Şekil 3.1.'deki gibi olması gerektiği savunulabilir.

Ele alınan 20 ilişkisel veritabanı tasarımı için bu işlem gerçekleştirilerek 20 ham veri seti elde edilmiştir. Başlangıçta ele alınan tasarımlar da algoritmayla ulaşılması gereken hedef olarak kabul edilmiştir.

3.4.3. Alt Amaç Fonksiyonlarının Ağırlıklarının Belirlenmesi

Amaç fonksiyonunu oluşturan 4 alt amaç, farklı ağırlıklarda öneme sahip olabilecekleri için her bir alt amaç fonksiyonu için katsayılar tanımlanmıştır. Ağırlıkların belirlenmesi için gerçekleştirilen uygulama Başlık 2.3.3 içerisinde, ağırlıkların hesaplanması sürecinde elde edilen sayısal bulgular Başlık 4.1.1 içerisinde ayrıntılı olarak incelenmiştir.

3.4.4. Önerilen Algoritmanın Başarısının Değerlendirilmesi

Çalışmanın uygulama kısmında, önerilen algoritmaya girdi olarak Başlık 3.2.2'de gösterilen şekilde hazırlanan ham veri seti sunulmakta ve çıktı olarak niteliklerin varlıklara dağılımını gösteren bir ilişkisel veritabanı tasarımı önerisi elde edilmektedir. Elde edilen çözümün başarısının test edilmesi, genetik arama sonucu elde edilen tasarım ile başlangıçta belirli olan tasarımla karşılaştırılması sayesinde gerçekleştirilmiştir. İki tasarımın birebir aynı olması doğrudan en başarılı olarak kabul edilmiş, ufak farklılıklara sahip olan çözümlerde ise hatanın sebebi yorumlanmıştır. Başlık 4.2, tüm uygulamalar için değerlendirmeleri sunmakta, alt başlıkları ayrıntılı bilgi içermektedir.

3.5. R KÜTÜPHANESİ

Önerilen algoritmanın uygulanabilmesini sağlayan bir R kütüphanesi hazırlanmıştır. Kütüphanenin yüklenebilmesi için;

```
source("autonormlib.R");
```

R komutunun kullanılması gerekmektedir. Optimizasyon işlemini gerçekleştirebilmek için *autonorm* fonksiyonunu çağırmak yeterlidir. Kütüphane, *autonorm* fonksiyonuna yardımcı birçok fonksiyon içermektedir. *autonorm* fonksiyonu, ham veri setini içeren tek bir zorunlu parametre ile çalıştırılabilmektedir. Aşağıda basit bir örnek sunulmaktadır. Ham veri setini okuyarak bir değişkene yüklemek için;

```
veridosyam <- read.csv("f.csv", sep = ",", header = FALSE)
```

komutunun çalıştırılması gerekmektedir. Bu komutla birlikte ham veri seti, *veridosyam* değişkenine yüklenmektedir. Otomatik normalizasyon işlemini gerçekleştirmek için *autonorm* fonksiyonunu;

```
sonuc <- autonorm(veridosyam)
```

şeklinde çağırmak yeterlidir. Burada *sonuc* değişkeni, arama sonucunu alabilmek için kullanılmaktadır.

autonorm fonksiyonu, kullanılması zorunlu olmayan başka parametrelere de sahiptir. Bu parametreler ve sağladıkları faydalar aşağıda belirtilmiştir.

quick: Verilen ham veri seti içerisinden örneklem alarak bu örneklem ile veritabanı tasarımı önermektedir. Daha az veri ile çalışıldığından daha hızlıdır ancak seçilecek verinin rastgele olması, dolayısıyla şans faktörünün etkisi hatalı sonuç alınmasına sebep olabilir. Bu parametrenin varsayılan değeri FALSE olarak belirlenmiştir.

popsize: GA'nın çalışması için gerekli bir parametre olan topluluk büyüklüğüdür. Varsayılan olarak 100 seçilmiştir.

maxiter: GA'nın gerektirdiği maxiter parametresidir. Varsayılan olarak 1000 seçilmiştir. Farklı bir nesil sayısı seçilmek istenildiğinde bu parametre kullanılabilir.

caching: Bu parametre önbellekleme işlemini sağlamaktadır. Parametre TRUE değeri aldığında, her kromozom değerlendirmesi sonrasında kromozomu ve değerini kaydeder. Bu sayede arama işlemi bir kromozoma tekrar denk geldiğinde yeniden işlem yapmak

yerine eldeki değeri kullanır. Parametrenin varsayılan değeri FALSE olarak belirlenmiştir.

w: Önerilen algoritmanın hangi alt amacı hangi ağırlıkla kullanacağını belirlemektedir. 4 elemanlı bir vektördür. Varsayılan olarak Başlık 2.3.3'te sunulan değerleri almaktadır.

monitor: Geliştirilen yazılım çalışırken kullanıcıyı bilgilendirmek üzere ekrana bilgi basan fonksiyondur. Tez için özel bir monitör fonksiyonu hazırlanmış ve varsayılan olarak parametre değeri olarak atanmıştır.

mutation: Algoritma çalışırken kullanılacak mutasyon operatörüdür. Algoritmaya özel akıllı bir mutasyon operatörü geliştirildiği için bu operatöre ait fonksiyon kütüphaneye eklenmiş ve varsayılan olarak atanmıştır.

4. BULGULAR

Bu başlık altında, önerilen algoritmanın hazırlanması, iyileştirilmesi ve test edilmesi aşamalarında elde edilen bulgular iki alt başlık ile sunulmaktır. Birinci başlık altında önerilen algoritma ve alt amaç katsayıları belirlenmesi aşamasında elde edilen değerler gösterilmekte, ayrıca GA'da arama süresine büyük etkisi olan nesil sayısının optimum olarak seçilmesi gereken aralığın belirlenmesine ilişkin veriler sunulmaktadır. İkinci başlık ise algoritmanın seçilen veritabanlarına uygulanmasını ve bunun sonucunda elde edilen verileri içermektedir.

4.1. ÖNERİLEN ALGORİTMANIN İYİLEŞTİRİLMESİ

4.1.1. Katsayıların Hesaplanması

Çalışmada önerilen algoritma, çok amaçlı bir optimizasyon problemi olup 4 alt amaçtan meydana gelmektedir. Ancak tüm alt amaçların aynı ağırlıkla sonuca etki etmesi çözüme olumsuz etkide bulunabilmektedir. Bazı alt amaçların çözüm üzerinde diğerlerinden daha baskın etkiye sahip olması gerekmektedir. Çok amaçlı optimizasyon problemlerinin çözümü için kullanılan yöntemlerden biri olan alt amaçların ağırlıklandırılması çözüm üretilmesine yardımcı olmaktadır. Amaç fonksiyonu Başlık 2.3.2.5 içerisinde;

$$\sum_{i=1}^{4} w_i = 1 \tag{4.1}$$

olmak üzere,

$$F = \sum_{i=1}^{4} w_i f_i \tag{4.2}$$

şeklinde tanımlanmıştı. Alt amaç fonksiyonu ağırlıklarını belirten w_i için belirleme yöntemi başlık 2.3.3 altında ayrıntılı olarak verilmektedir. Katsayıların belirlenmesi sürecinde GA araması kullanılmış ve başlık 2.3.3 altında sunulan adımlar 500 kez tekrar

edilmiştir. Gerçekleştirilen çalışmanın sonucunda ulaşılan, katsayılara ait istatistikler Tablo 4.1'de sunulmuştur.

Tablo 4.1: Katsayı Hesaplama Sonuçları.

	w_1	w_2	w_3	w_4
En düşük	,01253	,005474	,3994	,00709
En yüksek	,54620	,054120	,9354	,05658
Ortalama	,33960	,020300	,6087	,03139
Medyan	,36880	,016980	,5849	,03178
Standart Sapma	,13590	,011575	,1299	,01330

Algoritma içerisinde, test sonucunda elde edilen değerlerin ortalamaları kullanılmıştır. Ayrıca hesaplamalardaki standart sapma değerleri dikkate alındığında katsayıların;

$$\overline{w_i} - w_i^{sd} < w_i < \overline{w_i} + w_i^{sd} \tag{4.3}$$

ifadesinde w_i, i. katsayı, $\overline{w_i}$, i. katsayının ortalaması, w_i^{sd} ise i. katsayının standart sapması olmak üzere;

$$\begin{aligned} &,203702885 < w_1 < ,47551510 \\ &,008725754 < w_2 < ,03187657 \\ &,478729044 < w_3 < ,73866351 \\ &,018088280 < w_4 < ,04469885 \end{aligned} \tag{4.4}$$

aralığında seçilmesi de uygun görülebilir.

Hesaplanan katsayıların ortalamasına göre F_3 yüzde 61 ile en büyük etkiye sahiptir. F_1, yüzde 34 ile ikinci büyük etkiye sahipken F_2 ve F_4 sırasıyla yüzde 2 ve yüzde 3 etkiye sahiptir. Standart sapma değerleri dikkate alındığında katkı oranları daha geniş bir alana sahip olmakla birlikte sıralama aynı kalmaktadır.

4.1.2. Optimum Nesil Sayısının Hesaplanması

Nesil sayısı algoritmanın hızını ve başarısını etkileyen önemli faktörlerdendir. Nesil sayısının gerektiğinden daha az seçilmesi optimum sonuca ulaşma olasılığını düşürürken

gerektiğinden fazla seçilmesi de çalışma süresini arttıracaktır. Bu sebeple optimum nesil sayısının belirlenebilmesi için de bir çalışma gerçekleştirilmiştir.

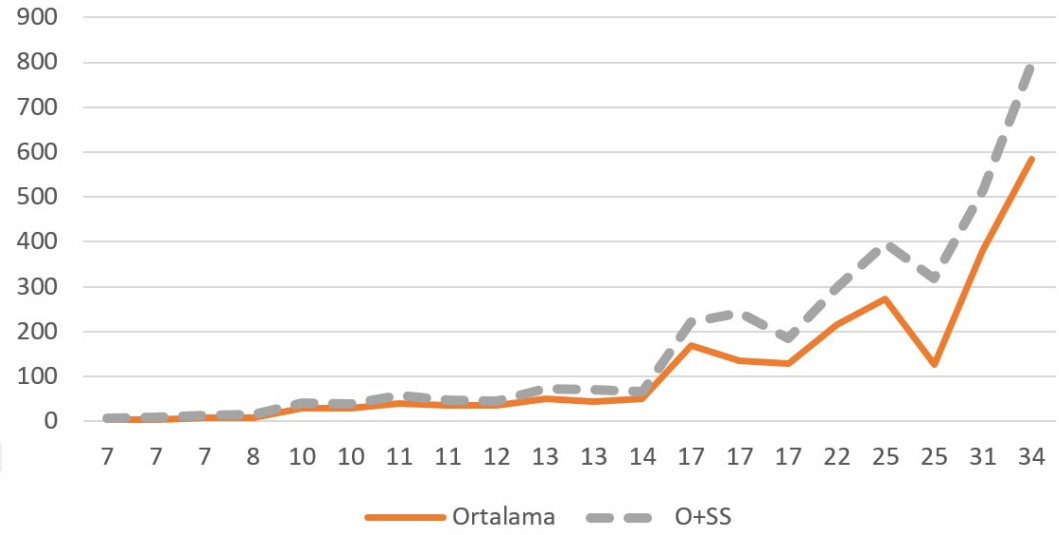
Uygulama için ele alınan veritabanlarından üretilen veri setleri, algoritma içerisinde 1000 nesil ile çözüme ulaştırılmaya çalışılmış ve en iyi çözüme ulaştıkları nesil sayısı kaydedilmiştir. Algoritma girdi olarak yalnızca ham veri setini aldığı için nesil sayısına karşılık nitelik sayısı ile değerlendirme yapılmıştır.

Uygulamalar ile elde edilen, en iyi sonuca ulaşılan nesil sayıları Tablo 4.2 ile sunulmaktadır.

Tablo 4.2: En İyi Sonuca Ulaşılan İterasyon Sayıları.

Nitelik	En Düşük	En Yüksek	Ortalama (O)	Standart Sapma (SS)	O+SS	O-SS
7	0	24	3,9	3,68	7,58	0,22
7	1	23	4,52	4,52	9,04	0
7	1	21	8,71	4,49	13,2	4,22
8	1	54	9,12	6,26	15,38	2,86
10	5	70	28,85	13,05	41,9	15,8
10	11	55	29,85	9,67	39,52	20,18
11	10	91	39,82	17,03	56,85	22,79
11	16	75	35,63	12,4	48,03	23,23
12	13	58	35,02	9,94	44,96	25,08
13	22	153	51,44	21,69	73,13	29,75
13	12	154	45,15	25,72	70,87	19,43
14	25	107	50,6	15,73	66,33	34,87
17	65	314	168,8	53	221,8	115,8
17	37	752	134,78	107,04	241,82	27,74
17	61	387	128,73	56,05	184,78	72,68
22	97	489	216,3	82,23	298,53	134,07
25	105	837	272,9	124	396,9	148,9
25	26	920	125,9	191,88	317,78	< 0
31	169	844	380,9	135,33	516,23	245,57
34	243	999	583,2	210,62	793,82	372,58

Şekil 4.1, nitelik sayısına karşılık gelen nesil sayılarını göstermektedir. Düşey eksen nesil sayısını, yatay eksen ham veri setindeki nitelik sayısını göstermektedir. Kesikli çizgi, Tablo 4.2'de sunulan ortalamayı, sürekli çizgi ise ortalama ile standart sapma değerlerinin toplamını ifade etmektedir.



Şekil 4.1: Nitelik Sayısına Göre Nesil Sayısı Değişimi.

Şekil 4.1'e göre ham veri setinde bulunan nitelik sayısı, en iyi çözüme ulaşılan nesil sayısını doğru orantılı olarak etkilemektedir. Daha kesin değerler için çok sayıda test yapılması gerekmekle birlikte, belirlenen bir nitelik sayısı için seçilebilecek nitelik sayısı Şekil 4.1 ve Tablo 4.2 ile yaklaşık olarak tayin edilebilmektedir.

4.2. ALGORİTMANIN UYGULANMASI

Bu doktora tezi kapsamında önerilen algoritma, bu algoritma baz alınarak gerçekleştirilmiş R dilindeki yazılım ile 20 veritabanından üretilen 20 yapay veri seti kullanılarak test edilmiştir. Veritabanı seçimi ile ilgili ayrıntılı bilgi Başlık 3.2.1 altında sunulmuştur. Bu başlık altında yapılan testlerin bulguları sunulacaktır. Ele alınan veritabanlarıyla ilgili özet bilgi Tablo 4.3'te sunulmuştur.

Tablo 4.3: Ele Alınan Veritabanlarının Özet Bilgisi

Veritabanı	Nitelik Sayısı	Varlık Sayısı
1	10	4
2	25	4
3	7	4
4	7	4
5	17	4
6	11	4
7	12	3
8	22	4
9	7	3

10	13	4
11	14	4
12	13	4
13	31	5
14	14	5
15	10	3
16	17	4
17	11	4
18	13	4
19	34	8
20	8	4

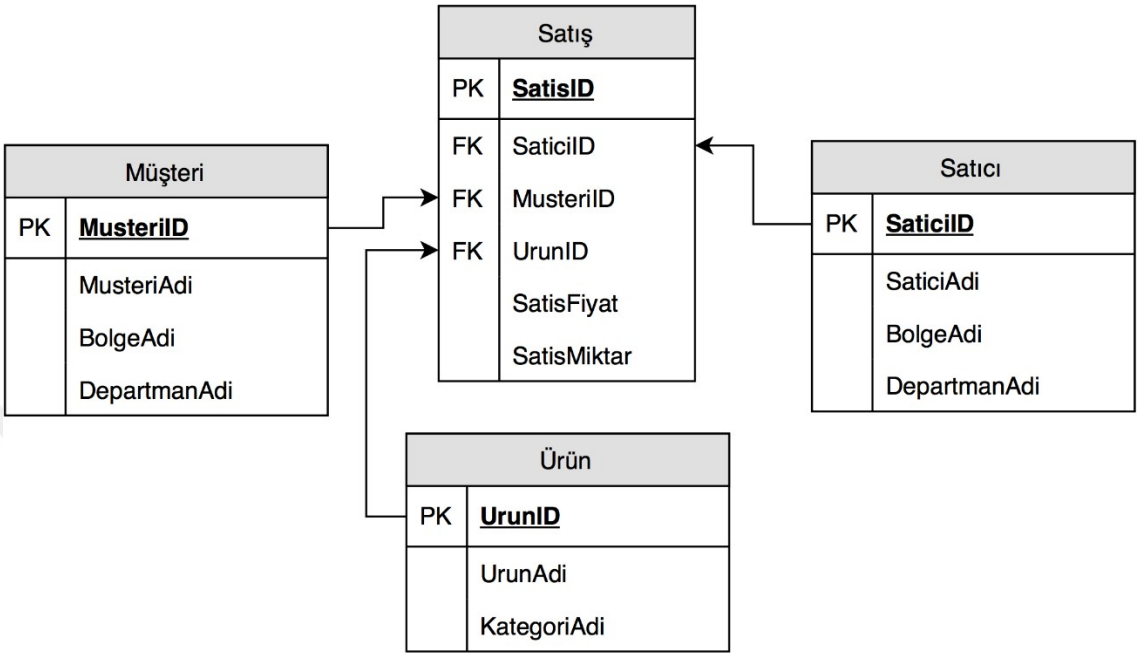
Yapılan uygulamalarda, elde edilen 20 yapay veri setinin büyük çoğunluğu birebir olarak beklenen veritabanı tasarımına ulaşırken, diğerlerinin büyük çoğunluğu da beklenen tasarıma çok yaklaşmıştır.

Önerilen algoritma, ele alınan 20 veritabanı içerisinden 11 tanesi (1, 2, 3, 6, 7, 8, 10, 11, 15, 16, 17) için beklenilen veritabanı tasarımının birebir aynısını önermiştir. 6 tanesi (4, 5, 12, 13, 14, 18) için ise ulaşılmak istenen tasarıma çok yakın tasarımlar elde edilmiştir. Yakın tasarımlar, bir niteliğin tekrarlı değer içermesi ancak farklı bir varlık ile temsil edilmesinin gereksiz olduğu durumlarda görülmektedir. Örneğin cinsiyet niteliği 2 değer alması sebebiyle çok fazla veri tekrarı oluşturuyor gibi görünebilmektedir. Algoritma da bu sebeple cinsiyet niteliğini hatalı varlık içerisine yerleştirebilmektedir. Geriye kalan 3 veritabanı için ise, beklenen veritabanı tasarımından çok uzak olmayan ancak yapıda değişiklik öneren tasarımlar ortaya çıkmıştır. Algoritma tarafından önerilen tasarımlar, ulaşılmak istenilen tasarımlarla karşılaştırılarak incelendiğinde, normalizasyon kurallarına göre mantıklı sayılabilecek düzenlemeler içermektedir. Bu sebeple bu tasarım önerilerine de tamamıyla hatalı demek doğru olmayacaktır.

Tüm veritabanları için gerçekleştirilen uygulamalar ve ayrıntıları alt başlıklar şeklinde sunulmuştur.

4.2.1. Veritabanı 1

Veritabanı 1 için varlık-ilişki diyagramı Şekil 4.2'de gösterilmektedir.



Şekil 4.2: Veritabanı 1'in Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.4'teki gibidir.

Tablo 4.4: Veritabanı 1 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
MusteriAdi	V1
BolgeAdi	V2
DepartmanAdi	V3
SaticiAdi	V4
BolgeAdi	V5
DepartmanAdi	V6
UrunAdi	V7
KategoriAdi	V8
SatisFiyat	V9
SatisMiktar	V10

Veri seti, 10 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 110 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.5}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 10$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 10 \rceil \, 10 = 30 \tag{4.6}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 10 \rceil = 3 \tag{4.7}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1442 \\ f_2 w_2 &= 0{,}000406 \\ f_3 w_3 &= 0{,}1522 \\ f_4 w_4 &= 0{,}0222 \end{aligned} \tag{4.8}$$

Buna göre;

$$F = 0{,}319 \tag{4.9}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.5'teki gibidir.

Tablo 4.5: Veritabanı 1 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	5
En yüksek	70
Medyan	29
MAD	13,3434

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

1001100110010100010001001110111010101010

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.3'te sunulmaktadır.

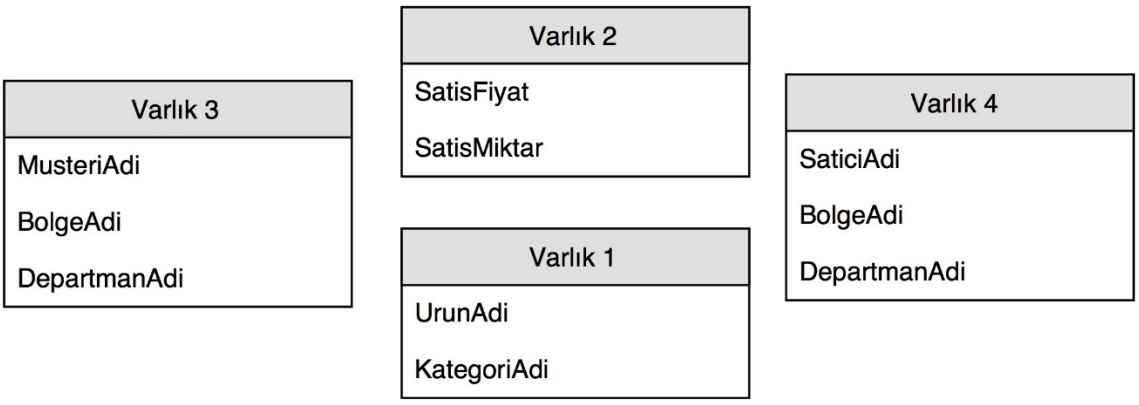
```
> anStructure(mych,rawdata)
[1] 7 8
[1]  9 10
[1] 1 2 3
[1] 4 5 6
```

Şekil 4.3: Veritabanı 1 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.3'te elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 7 ve 8 bir varlık içerisinde, nitelik 9 ve 10 ikinci varlık içerisinde, nitelik 1, 2 ve 3 üçüncü varlık içerisinde, nitelik 4, 5 ve 6 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.6'da nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.6: Veritabanı 1 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	7, 8	UrunAdi, KategoriAdi
Varlık 2	9, 10	SatisFiyat, SatisMiktar
Varlık 3	1, 2, 3	MusteriAdi, BolgeAdi, DepartmanAdi
Varlık 4	4, 5, 6	SaticiAdi, BolgeAdi, DepartmanAdi

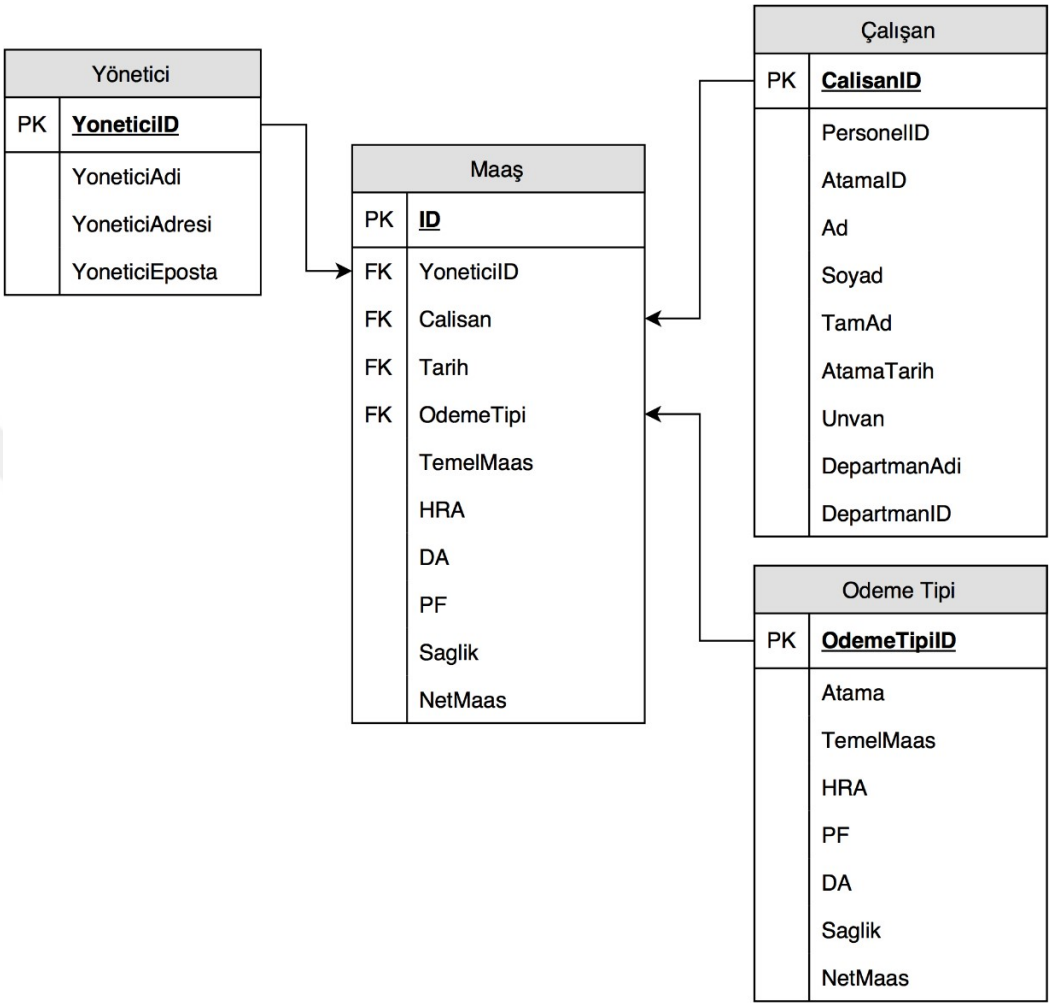


Şekil 4.4: Veritabanı 1 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.4'te diyagram olarak sunulmuştur.

4.2.2. Veritabanı 2

Veritabanı 2 için varlık-ilişki diyagramı Şekil 4.5'te gösterilmektedir.



Şekil 4.5: Veritabanı 2'nin Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.7'deki gibidir.

Tablo 4.7: Veritabanı 2 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
YoneticiAdi	V1
YoneticiAdresi	V2
YoneticiEposta	V3
PersonelID	V4
AtamaID	V5

Ad	V6
Soyad	V7
TamAd	V8
AtamaTarih	V9
Unvan	V10
DepartmanAdi	V11
DepartmanID	V12
Atama	V13
TemelMaas	V14
HRA	V15
DA	V16
PF	V17
Saglik	V18
NetMaas	V19
TemelMaas	V20
HRA	V21
DA	V22
PF	V23
Saglik	V24
NetMaas	V25

Veri seti, 25 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 105 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.10}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 25$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 25 \rceil \, 25 = 125 \tag{4.11}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 25 \rceil = 5 \tag{4.12}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$f_1 w_1 = 0{,}1655933$$

$$f_2 w_2 = 0{,}009550988$$

$$f_3 w_3 = 0{,}07608703 \qquad 4.13$$

$$f_4 w_4 = 0{,}0221986$$

Buna göre;

$$F = 0{,}27343 \qquad 4.14$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.8'deki gibidir.

Tablo 4.8: Veritabanı 2 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	105
En yüksek	837
Medyan	242,5
MAD	85,9908

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

1000110001100011111011110111101111011110111101111011110111101111011110011
1001110011100111001110011100111000100001000010000100001000010000100

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.6'da sunulmaktadır.

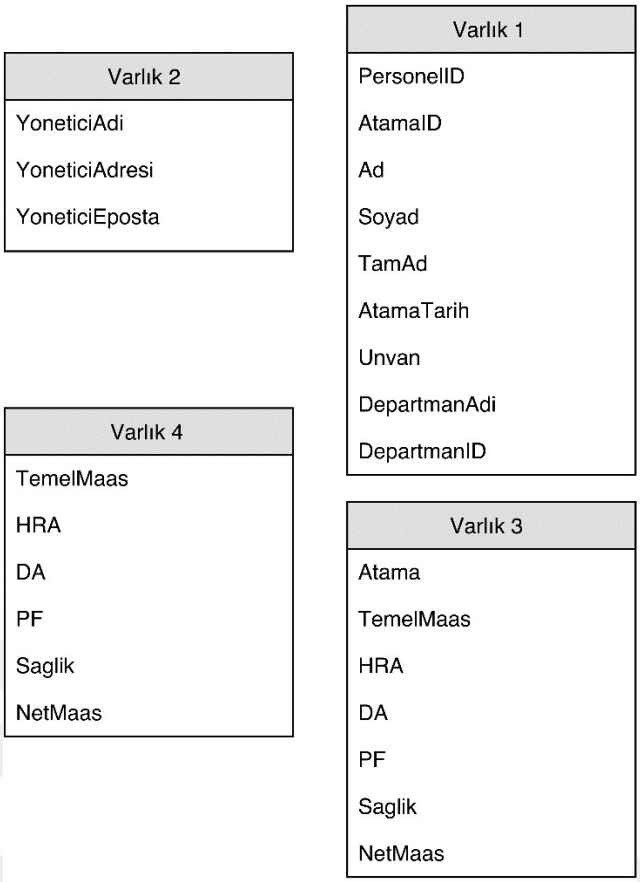
```
> anStructure(mych,rawdata)
[1]  4  5  6  7  8  9 10 11 12
[1] 1 2 3
[1] 13 14 15 16 17 18 19
[1] 20 21 22 23 24 25
```

Şekil 4.6: Veritabanı 2 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.6'da elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 4, 5, 6, 7, 8, 9, 10, 11 ve 12 birinci varlık içerisinde, nitelik 1, 2 ve 3 ikinci varlık içerisinde, nitelik 13, 14, 15, 16, 17, 18 ve 19 üçüncü varlık içerisinde, nitelik 20, 21, 22, 23, 24 ve 25 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.9'da nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.9: Veritabanı 2 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	4,5,6,7,8,9,10,11,12	PersonelID, AtamaID, Ad, Soyad, TamAd, AtamaTarih, Unvan, DepartmanAdi, DepartmanID
Varlık 2	1,2,3	YoneticiAdi, YoneticiAdresi, YoneticiEposta
Varlık 3	13,14,15,16,17,18,19	Atama, TemelMaas, HRA, DA, PF, Saglik, NetMaas
Varlık 4	20,21,22,23,24,25	TemelMaas, HRA, DA, PF, Saglik, NetMaas

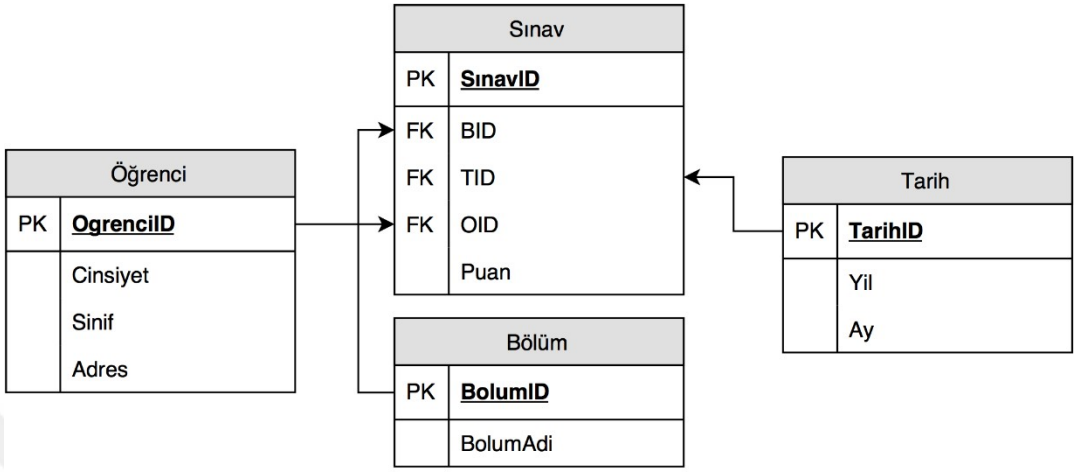


Şekil 4.7: Veritabanı 2 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.7'de diyagram olarak sunulmuştur.

4.2.3. Veritabanı 3

Veritabanı 3 için varlık-ilişki diyagramı Şekil 4.8'de gösterilmektedir.



Şekil 4.8: Veritabanı 3'ün Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.10'daki gibidir.

Tablo 4.10: Veritabanı 3 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Cinsiyet	V1
Sinif	V2
Adres	V3
Yil	V4
Ay	V5
BolumAdi	V6
Puan	V7

Veri seti, 7 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 41 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.15}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 7$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 7 \rceil \, 7 = 21 \tag{4.16}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 7 \rceil = 3 \tag{4.17}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}06607821 \\ f_2 w_2 &= 0{,}04964951 \\ f_3 w_3 &= 0{,}3043481 \\ f_4 w_4 &= 0{,}0221986 \end{aligned} \tag{4.18}$$

Buna göre;

$$F = 0{,}4422745 \tag{4.19}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.11'deki gibidir.

Tablo 4.11: Veritabanı 3 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	0
En yüksek	24
Medyan	3
MAD	2,9652

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım

için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

00000000101101001011

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.9'da sunulmaktadır.

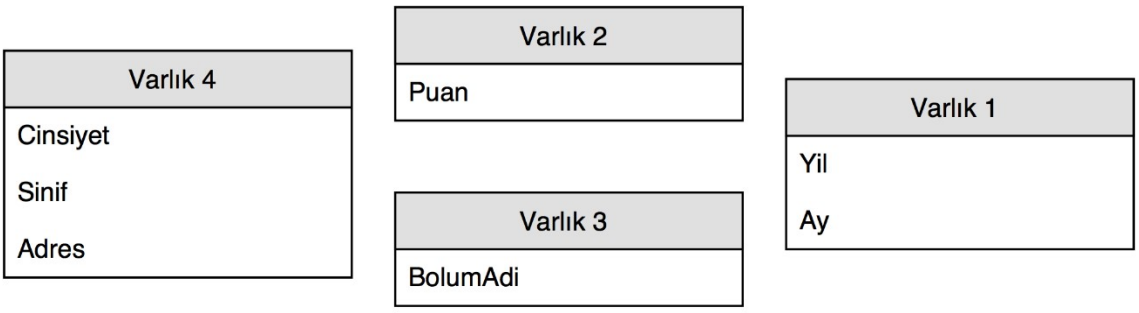
```
> anStructure(mych,rawdata)
[1] 4 5
[1] 7
[1] 6
[1] 1 2 3
```

Şekil 4.9: Veritabanı 3 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.9'da elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 4 ve 5 birinci varlık içerisinde, nitelik 7 ikinci varlık içerisinde, nitelik 6 üçüncü varlık içerisinde, nitelik 1, 2 ve 3 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.12'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.12: Veritabanı 3 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	4, 5	Yil, Ay
Varlık 2	7	Puan
Varlık 3	6	BolumAdi
Varlık 4	1, 2, 3	Cinsiyet, Sinif, Adres

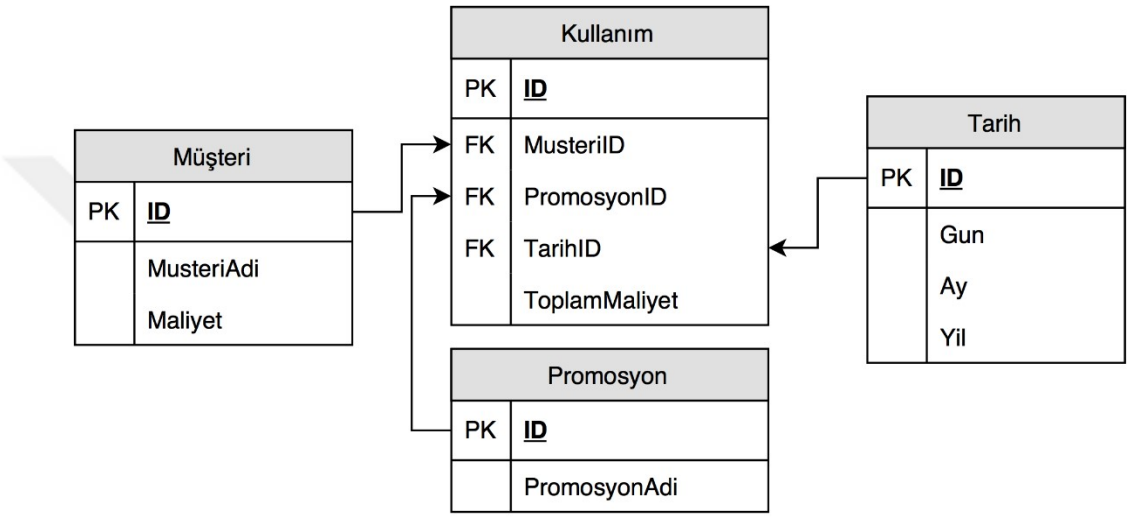


Şekil 4.10: Veritabanı 3 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.10'da diyagram olarak sunulmuştur.

4.2.4. Veritabanı 4

Veritabanı 4 için varlık-ilişki diyagramı Şekil 4.11'de gösterilmektedir.



Şekil 4.11: Veritabanı 4'ün Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.13'teki gibidir.

Tablo 4.13: Veritabanı 4 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri setindeki Adı
MusteriAdi	V1
Maliyet	V2
PromosyonAdi	V3
Gun	V4
Ay	V5
Yil	V6
ToplamMaliyet	V7

Veri seti, 7 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 25 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.20}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 7$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 7 \rceil \, 7 = 21 \tag{4.21}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 7 \rceil = 3 \tag{4.22}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1699986 \\ f_2 w_2 &= 0{,}06905131 \\ f_3 w_3 &= 0{,}2282611 \\ f_4 w_4 &= 0{,}02176708 \end{aligned} \tag{4.23}$$

Buna göre;

$$F = 0{,}489078 \tag{4.24}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.14'teki gibidir.

Tablo 4.14: Veritabanı 4 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	1
En yüksek	23
Medyan	6
MAD	2,9652

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

11111100000100100 1000

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.12'de sunulmaktadır.

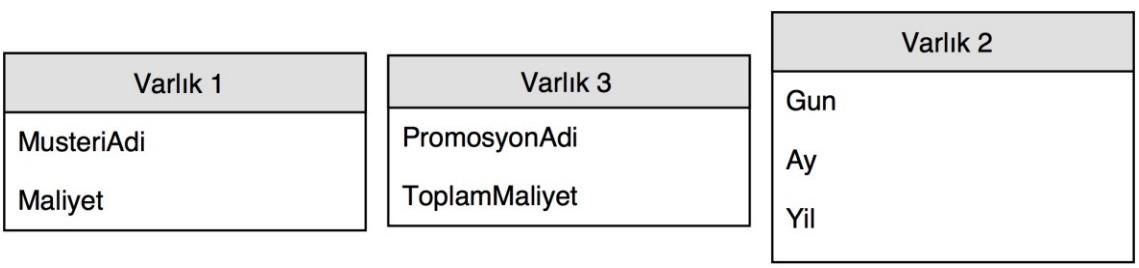
```
> anStructure(mych,rawdata)
[1] 1 2
[1] 4 5 6
[1] 3 7
```

Şekil 4.12: Veritabanı 4 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.12'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 1 ve 2 birinci varlık içerisinde, nitelik 4, 5 ve 6 ikinci varlık içerisinde, nitelik 3 ve 7 üçüncü varlık içerisinde bulunmalıdır. Tablo 4.15'te nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.15: Veritabanı 4 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	1, 2	MusteriAdi, Maliyet
Varlık 2	4, 5, 6	Gun, Ay, Yil
Varlık 3	3, 7	PromosyonAdi, ToplamMaliyet



Şekil 4.13: Veritabanı 4 İçin Elde Edilen Çözümün Diyagramı.

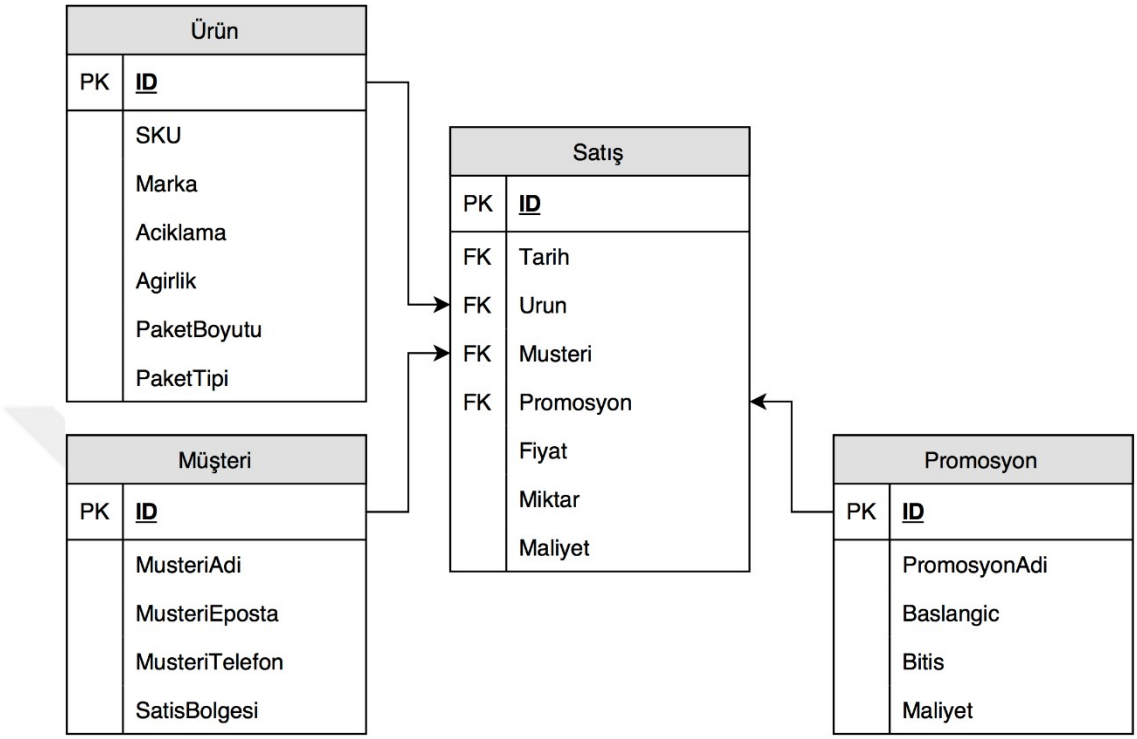
Algoritmanın önerdiği veritabanı tasarımı incelendiğinde başlangıçta ele alınan veritabanı tasarımı ile farklılıklar taşıdığı görülmektedir. Önerilen veritabanı tasarımında (Tablo 4.12) Varlık 1 ve Varlık 2 doğru belirlenmişken Varlık 3 içerisinde yer alan nitelik 3 ve nitelik 7, gerçek veritabanı tasarımına göre farklı varlıklar içerisinde yer almalıdır.

Bu hataya sebep olabilecek iki durum söz konusudur. Birincisi, alt amaçlardan birinin tablo sayısını en aza indirmeye çalışmasıdır. Bu alt amacın etkisiyle algoritma, birer niteliğe sahip iki varlık oluşturmak yerine iki niteliği tek bir varlık içerisinde temsil etme eğitiminde olmuştur. İkinci durum, sayısal varlık sayısının en az olması alt amacından kaynaklanmaktadır. Algoritma, tamamen sayısal değerlerden oluşan niteliklerin bir araya gelerek varlık oluşturmasını engelleyecek şekilde çalışmaktadır. ToplamMaliyet niteliğinin sayısal verilerden oluşması sebebiyle bir varlık oluşturması mümkün olamamıştır.

Alt amaçların yönlendirmeleri sebebiyle algoritma tarafından önerilen veritabanı tasarımı, gerçek tasarıma oldukça yakın ancak aynı değildir. Elde edilen çözüm Şekil 4.13'te diyagram olarak sunulmuştur.

4.2.5. Veritabanı 5

Veritabanı 5 için varlık-ilişki diyagramı Şekil 4.14'te gösterilmektedir.



Şekil 4.14: Veritabanı 5'in Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.16'daki gibidir.

Tablo 4.16: Veritabanı 5 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri setindeki Adı
MusteriAdi	V1
MusteriEposta	V2
MusteriTelefon	V3
SatisBolgesi	V4
SKU	V5
Marka	V6
Aciklama	V7
Agirlik	V8
PaketBoyutu	V9
PaketTipi	V10
PromosyonAdi	V11
Baslangic	V12

Bitis	V13
Maliyet	V14
Fiyat	V15
Miktar	V16
Maliyet	V17

Veri seti, 17 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 91 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.25}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 17$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 17 \rceil \, 17 = 85 \tag{4.26}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 17 \rceil = 5 \tag{4.27}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1278928 \\ f_2 w_2 &= 0{,}04081456 \\ f_3 w_3 &= 0{,}07608703 \\ f_4 w_4 &= 0 \end{aligned} \tag{4.28}$$

Buna göre;

$$F = 0{,}2447943 \qquad 4.29$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.17'deki gibidir.

Tablo 4.17: Veritabanı 5 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	65
En yüksek	314
Medyan	169,5
MAD	54,1149

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

000000000000000000000101101011010110101100010010111111011110111011110111100001000010000010

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.15'te sunulmaktadır.

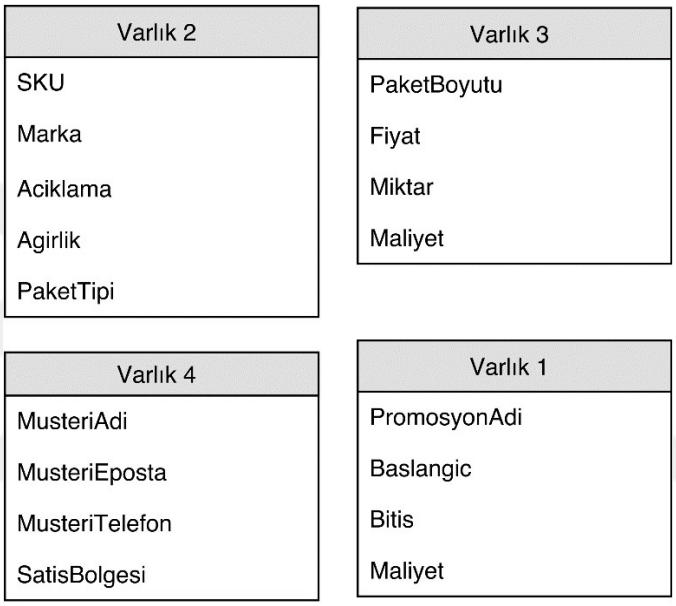
```
> anStructure(mych,rawdata)
[1] 11 12 13 14
[1]  5  6  7  8 10
[1]  9 15 16 17
[1] 1 2 3 4
```

Şekil 4.15: Veritabanı 5 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.15'te elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 11, 12, 13 ve 14 birinci varlık içerisinde, nitelik 5, 6, 7, 8 ve 10 ikinci varlık içerisinde, nitelik 9, 15, 16 ve 17 üçüncü varlık içerisinde, nitelik 1, 2, 3 ve 4 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.18'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.18: Veritabanı 5 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	11, 12, 13, 14	PromosyonAdi, Baslangic, Bitis, Maliyet
Varlık 2	5, 6, 7, 8, 10	SKU, Marka, Aciklama, Agirlik, PaketTipi
Varlık 3	9, 15, 16, 17	PaketBoyutu, Fiyat, Miktar, Maliyet
Varlık 4	1, 2, 3, 4	MusteriAdi, MusteriEposta, MusteriTelefon, SatisBolgesi



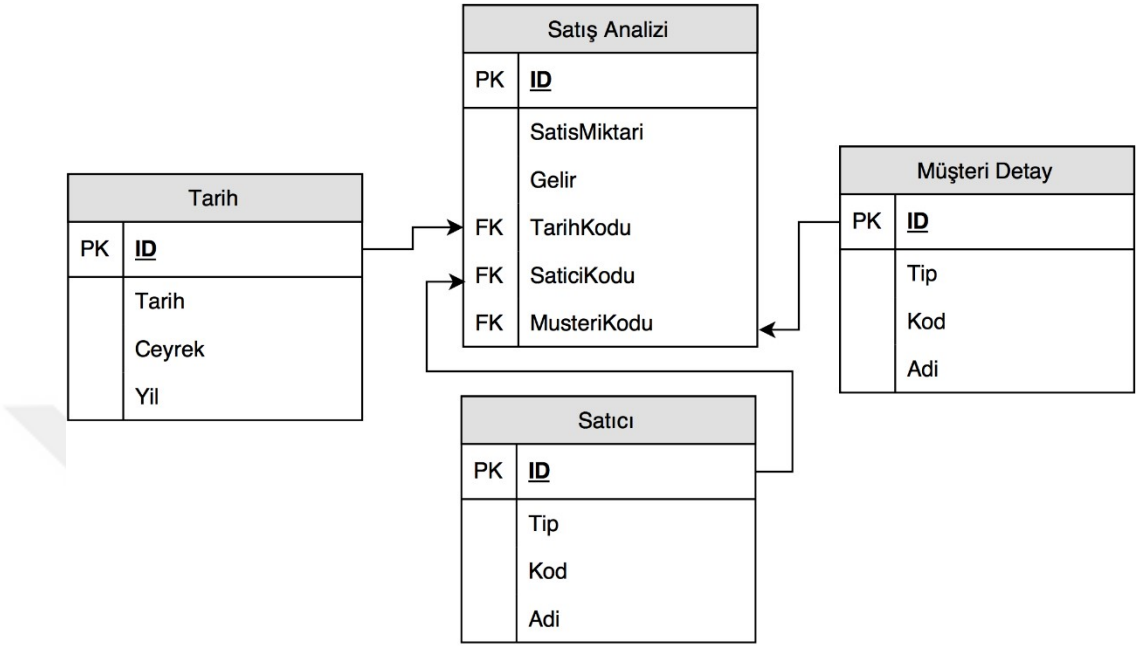
Şekil 4.16: Veritabanı 5 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımında gerçek veritabanı tasarımına göre tek bir nitelik yanlış varlık içerisinde yer almıştır. Nitelik 9 (PaketBoyutu)'un, Varlık 2 içerisinde yer alması gerekirken Varlık 3 içerisinde yer almıştır.

Bu hatanın sebebi Nitelik 9'un yapısıdır. Nitelik 9, "xxs", "xs", "s", "m", "l", "xl", "xxl" değerlerini almaktadır. Alınabilecek değerlerin kısıtlı olması, veri setinde çok fazla tekrara sebep olmaktadır. Algoritmanın veri tekrarını en aza indirme çabası, bu niteliğin yanlış varlık içerisinde yer almasına sebep olmuştur. Böylece algoritma gerçek veritabanı tasarımına çok yakın ancak aynı olmayan bir çözüme ulaşmıştır. Elde edilen çözüm Şekil 4.16'da diyagram olarak sunulmuştur.

4.2.6. Veritabanı 6

Veritabanı 6 için varlık-ilişki diyagramı Şekil 4.17'de gösterilmektedir.



Şekil 4.17: Veritabanı 6'nın Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.19'daki gibidir.

Tablo 4.19: Veritabanı 6 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Tip	V1
Kod	V2
Adi	V3
Tip	V4
Kod	V5
Adi	V6
Tarih	V7
Ceyrek	V8
Yil	V9
SatisMiktari	V10
Gelir	V11

Veri seti, 11 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 45 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.30}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 11$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 11 \rceil \, 11 = 44 \tag{4.31}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 11 \rceil = 4 \tag{4.32}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1410304 \\ f_2 w_2 &= 0{,}04280485 \\ f_3 w_3 &= 0{,}1521741 \\ f_4 w_4 &= 0{,}0221986 \end{aligned} \tag{4.33}$$

Buna göre;

$$F = 0{,}3582079 \tag{4.34}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.20'deki gibidir.

Tablo 4.20: Veritabanı 6 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	10
En yüksek	91
Medyan	38,5
MAD	17,0499

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

00100010001011011101110111111111111101110111

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.18'de sunulmaktadır.

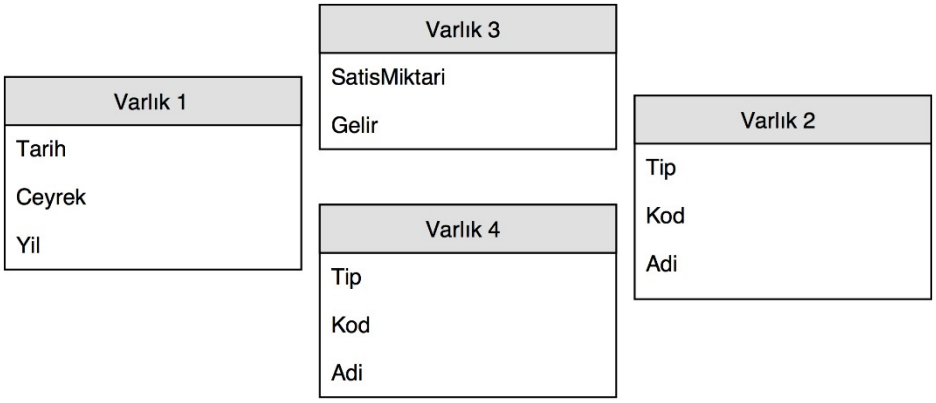
```
> anStructure(mych,rawdata)
[1] 7 8 9
[1] 4 5 6
[1] 10 11
[1] 1 2 3
```

Şekil 4.18: Veritabanı 6 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.18'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 7, 8 ve 9 birinci varlık içerisinde, nitelik 4, 5 ve 6 ikinci varlık içerisinde, nitelik 10 ve 11 üçüncü varlık içerisinde, nitelik 1, 2 ve 3 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.21'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.21: Veritabanı 6 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	7, 8, 9	Tarih, Ceyrek, Yil
Varlık 2	4, 5, 6	Tip, Kod, Adi
Varlık 3	10, 11	SatisMiktari, Gelir
Varlık 4	1, 2, 3	Tip, Kod, Adi

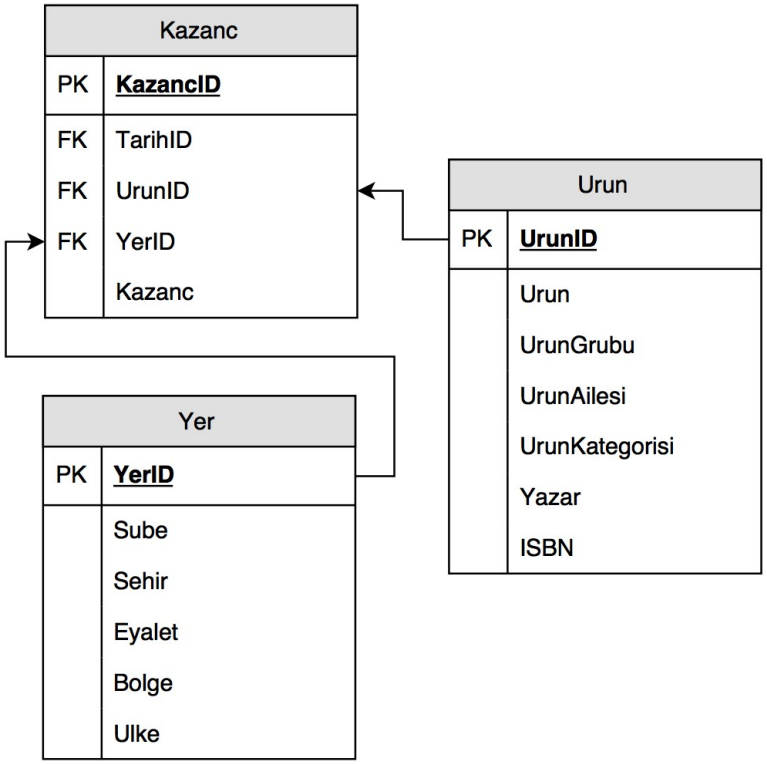


Şekil 4.19: Veritabanı 6 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.19'da diyagram olarak sunulmuştur.

4.2.7. Veritabanı 7

Veritabanı 7 için varlık-ilişki diyagramı Şekil 4.20'de gösterilmektedir.



Şekil 4.20: Veritabanı 7'nin Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.22'deki gibidir.

Tablo 4.22: Veritabanı 7 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Sube	V1
Sehir	V2
Eyalet	V3
Bolge	V4
Ulke	V5
Urun	V6
UrunGrubu	V7
UrunAilesi	V8
UrunKategorisi	V9
Yazar	V10
ISBN	V11

Kazanc	V12

Veri seti, 12 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 49 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.35}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 12$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 12 \rceil \, 12 = 48 \tag{4.36}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 12 \rceil = 4 \tag{4.37}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1333531 \\ f_2 w_2 &= 0{,}009443448 \\ f_3 w_3 &= 0{,}1141306 \\ f_4 w_4 &= 0{,}02176708 \end{aligned} \tag{4.38}$$

Buna göre;

$$F = 0{,}2786942 \tag{4.39}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.23'teki gibidir.

Tablo 4.23: Veritabanı 7 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	13

En yüksek	58
Medyan	34,5
MAD	9,6369

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

1001100110011001100101010101010101010101010101011100

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.21'de sunulmaktadır.

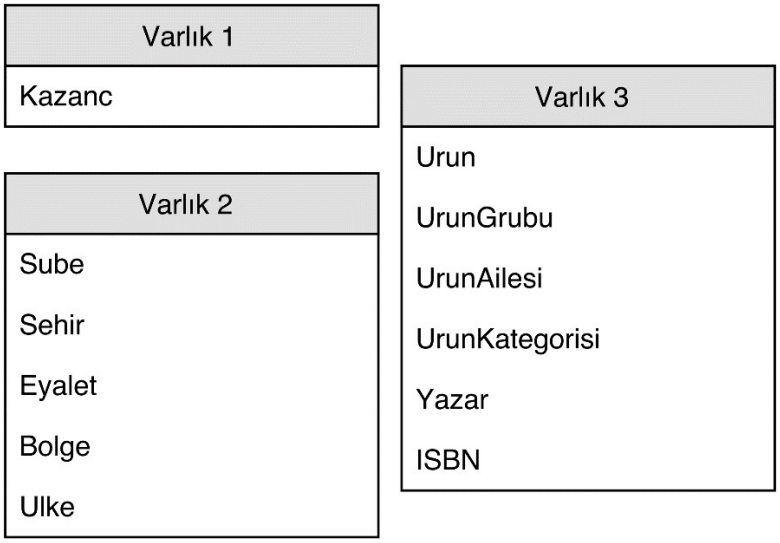
```
> anStructure(mych,rawdata)
[1] 12
[1] 1 2 3 4 5
[1]  6  7  8  9 10 11
```

Şekil 4.21: Veritabanı 7 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.21'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 12 birinci varlık içerisinde, nitelik 1, 2, 3, 4 ve 5 ikinci varlık içerisinde, nitelik 6, 7, 8, 9, 10, 11 ve 12 üçüncü varlık içerisinde bulunmalıdır. Tablo 4.24'te nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.24: Veritabanı 7 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	12	Kazanc
Varlık 2	1, 2, 3, 4, 5	Sube, Sehir, Eyalet, Bolge, Ulke
Varlık 3	6, 7, 8, 9, 10, 11	Urun, UrunGrubu, UrunAilesi, UrunKategorisi, Yazar, ISBN

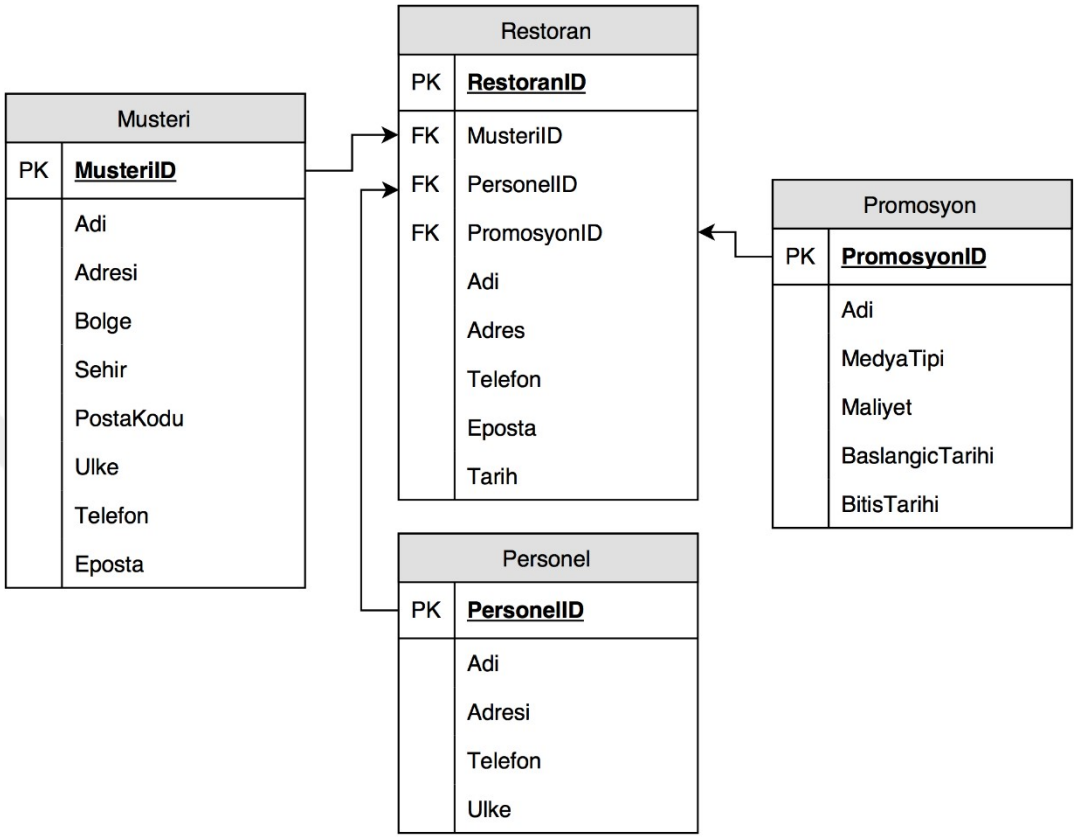


Şekil 4.22: Veritabanı 7 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.22'de diyagram olarak sunulmuştur.

4.2.8. Veritabanı 8

Veritabanı 8 için varlık-ilişki diyagramı Şekil 4.23'te gösterilmektedir.



Şekil 4.23: Veritabanı 8'in Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.25'teki gibidir.

Tablo 4.25: Veritabanı 8 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Adi	V1
Adresi	V2
Bolge	V3
Sehir	V4
PostaKodu	V5
Ulke	V6
Telefon	V7
Eposta	V8
Adi	V9
Adresi	V10

Telefon	V11
Ulke	V12
Adi	V13
MedyaTipi	V14
Maliyet	V15
BaslangicTarihi	V16
BitisTarihi	V17
Adi	V18
Adres	V19
Telefon	V20
Eposta	V21
Tarih	V22

Veri seti, 22 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 169 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.40}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 22$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 22 \rceil \, 22 = 110 \tag{4.41}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 22 \rceil = 5 \tag{4.42}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$f_1 w_1 = 0{,}1469889$$

$$f_2 w_2 = 0{,}02714425 \tag{4.43}$$

$$f_3 w_3 = 0{,}07608703$$

$$f_4 w_4 = 0$$

Buna göre;

$$F = 0{,}2502202 \qquad 4.44$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.26'daki gibidir.

Tablo 4.26: Veritabanı 8 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	97
En yüksek	489
Medyan	196,5
MAD	74,8713

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

101101011010110101101011010110101101011011111111111111
111110001100011000110001100011101111011110111011110111

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.24'te sunulmaktadır.

```
> anStructure(mych,rawdata)
[1]  9 10 11 12
[1] 18 19 20 21 22
[1] 1 2 3 4 5 6 7 8
[1] 13 14 15 16 17
```

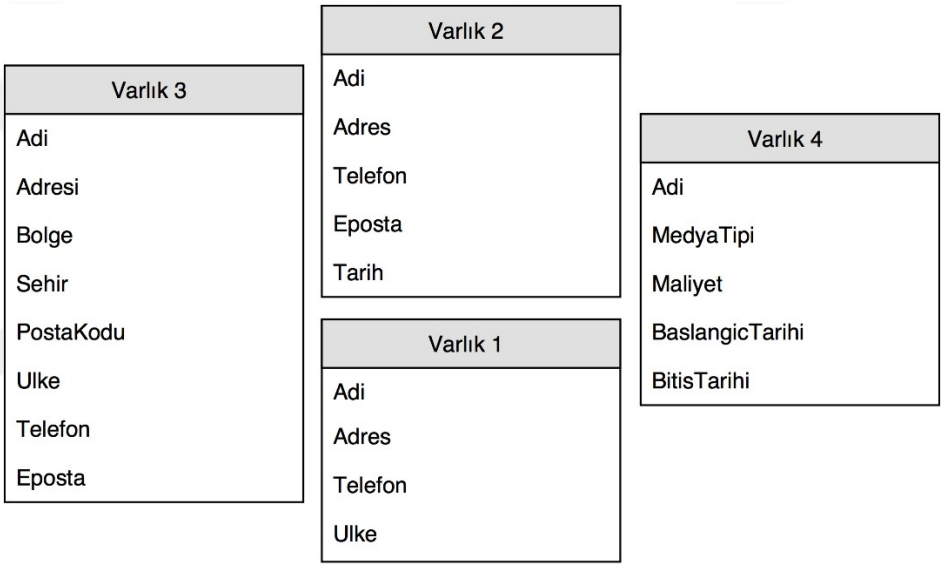
Şekil 4.24: Veritabanı 8 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.24'te elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 9, 10, 11 ve 12 birinci varlık içerisinde, nitelik 18, 19, 20, 21 ve 22 ikinci varlık içerisinde, nitelik 1, 2, 3, 4, 5, 6, 7 ve 8 üçüncü varlık içerisinde, nitelik 13, 14, 15, 16 ve 17 dördüncü varlık

içerisinde bulunmalıdır. Tablo 4.27'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.27: Veritabanı 8 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	9, 10, 11, 12	Adi, Adres, Telefon, Ulke
Varlık 2	18, 19, 20, 21, 22	Adi, Adres, Telefon, Eposta, Tarih
Varlık 3	1, 2, 3, 4, 5, 6, 7, 8	Adi, Adresi, Bolge, Sehir, PostaKodu, Ulke, Telefon, Eposta
Varlık 4	13, 14, 15, 16, 17	Adi, MedyaTipi, Maliyet, BaslangicTarihi, BitisTarihi

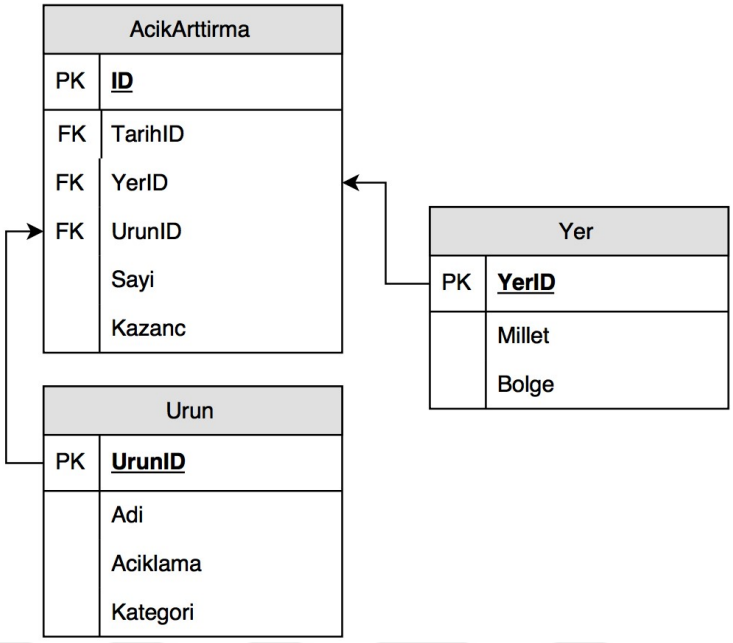


Şekil 4.25: Veritabanı 8 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.25'te diyagram olarak sunulmuştur.

4.2.9. Veritabanı 9

Veritabanı 9 için varlık-ilişki diyagramı Şekil 4.26'da gösterilmektedir.



Şekil 4.26: Veritabanı 9'un Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.28'deki gibidir.

Tablo 4.28: Veritabanı 9 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Adi	V1
Aciklama	V2
Kategori	V3
Millet	V4
Bolge	V5
Sayi	V6
Kazanc	V7

Veri seti, 7 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 37 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.45}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 7$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 7 \rceil \, 7 = 21 \tag{4.46}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 7 \rceil = 3 \tag{4.47}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}2484967 \\ f_2 w_2 &= 0{,}04389545 \\ f_3 w_3 &= 0{,}1521741 \\ f_4 w_4 &= 0 \end{aligned} \tag{4.48}$$

Buna göre;

$$F = 0{,}4445663 \tag{4.49}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.29'daki gibidir.

Tablo 4.29: Veritabanı 9 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	1
En yüksek	21
Medyan	8
MAD	4,4478

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

00000000001001001001

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.27'de sunulmaktadır.

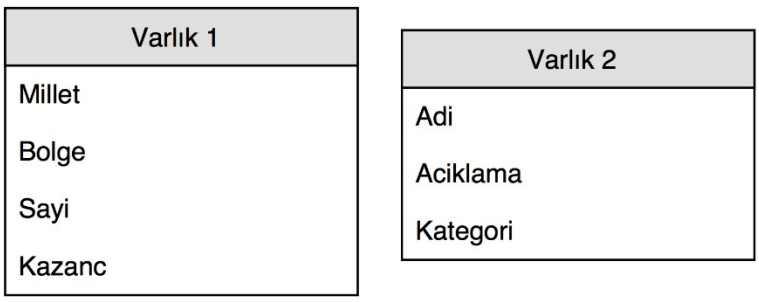
```
> anStructure(mych,rawdata)
[1] 4 5 6 7
[1] 1 2 3
```

Şekil 4.27: Veritabanı 9 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.27'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 4, 5, 6 ve 7 birinci varlık içerisinde, nitelik 1, 2 ve 3 ikinci varlık içerisinde bulunmalıdır. Tablo 4.30'da nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.30: Veritabanı 9 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	4, 5, 6, 7	Millet, Bolge, Sayi, Kazanc
Varlık 2	1, 2, 3	Adi, Aciklama, Kategori



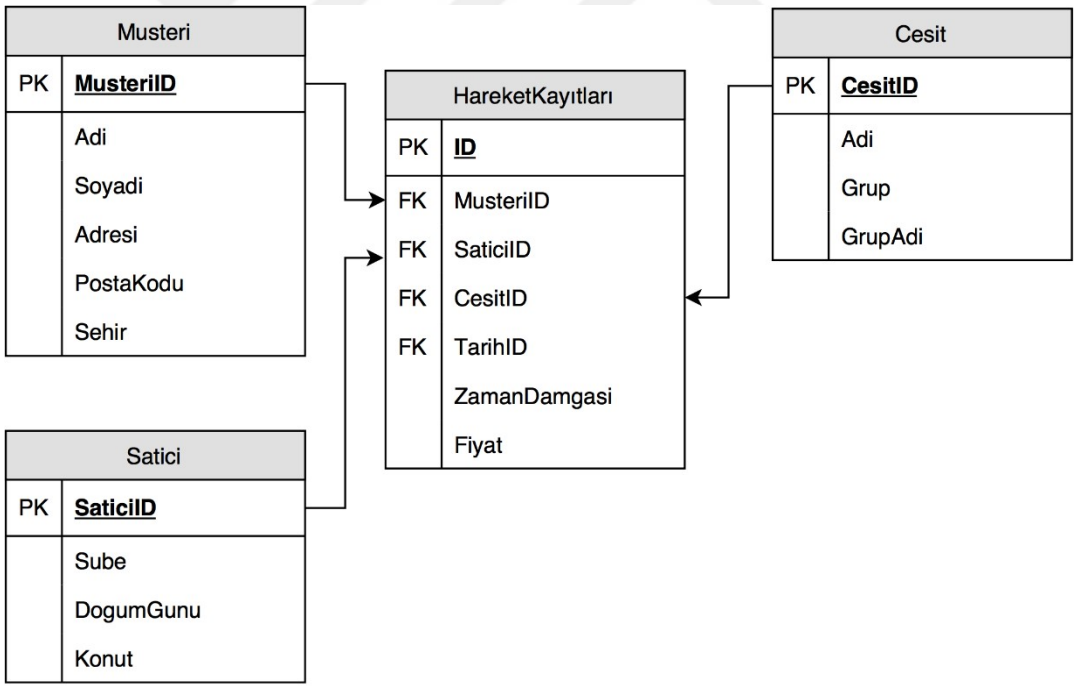
Şekil 4.28: Veritabanı 9 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı, başlangıçta ele alınan tasarıma göre farklılıklar taşımaktadır. 3 varlık oluşması beklenirken algoritma nitelikleri 2 varlık içerisinde bir araya getirmiştir. Önerilen varlıklardan Varlık 2, Urun ile birebir aynı nitelikleri içermektedir. Varlık 1 ise tasarımdaki Yer ve AcikArttirma varlıklarına ait nitelikleri içermektedir.

Bu hata, algoritmanın tablo sayısını en aza indirme ve sayısal varlık sayısını en aza indirme alt amaçlarının bir sonucu olabilir. Sayi ve Kazanc nitelikleri tamamen sayısal değerler içerdiği için algoritma, kategorik değer içeren Millet ve Bolge niteliklerini de kapsayan tek bir varlık oluşturmuştur. Elde edilen çözüm Şekil 4.28'de diyagram olarak sunulmuştur.

4.2.10. Veritabanı 10

Veritabanı 10 için varlık-ilişki diyagramı Şekil 4.29'da gösterilmektedir.



Şekil 4.29: Veritabanı 10'un Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.31'deki gibidir.

Tablo 4.31: Veritabanı 10 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Adi	V1
Soyadi	V2
Adresi	V3
PostaKodu	V4
Sehir	V5
Sube	V6
DogumGunu	V7
Konut	V8
Adi	V9
Grup	V10
GrupAdi	V11
ZamanDamgasi	V12
Fiyat	V13

Veri seti, 13 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 65 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.50}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 13$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 13 \rceil \, 13 = 52 \tag{4.51}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 13 \rceil = 4 \tag{4.52}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$f_1 w_1 = 0{,}1035546 \tag{4.53}$$

$$f_2 w_2 = 0{,}00913665$$

$$f_3 w_3 = 0{,}1521741$$

$$f_4 w_4 = 0$$

Buna göre;

$$F = 0{,}2648653 \qquad 4.54$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.32'deki gibidir.

Tablo 4.32: Veritabanı 10 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	22
En yüksek	153
Medyan	46.5
MAD	14,0847

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

0010001000100010001000010001000111001100110011011101

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.30'da sunulmaktadır.

```
> anStructure(mych,rawdata)
[1] 12 13
[1]  9 10 11
[1] 1 2 3 4 5
[1] 6 7 8
```

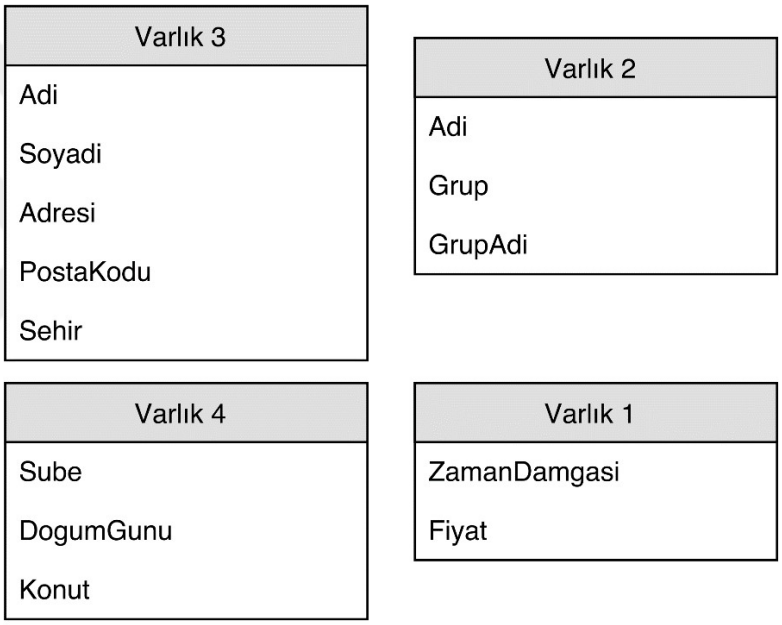
Şekil 4.30: Veritabanı 10 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.30'da elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 12 ve 13

birinci varlık içerisinde, nitelik 9, 10 ve 11 ikinci varlık içerisinde, nitelik 1, 2, 3, 4 ve 5 üçüncü varlık içerisinde, nitelik 6, 7 ve 8 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.33'te nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.33: Veritabanı 10 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	12, 13	ZamanDamgasi, Fiyat
Varlık 2	9, 10, 11	Adi, Grup, GrupAdi
Varlık 3	1, 2, 3, 4, 5	Adi, Soyadi, Adresi, PostaKodu, Sehir
Varlık 4	6, 7, 8	Sube, DogumGunu, Konut

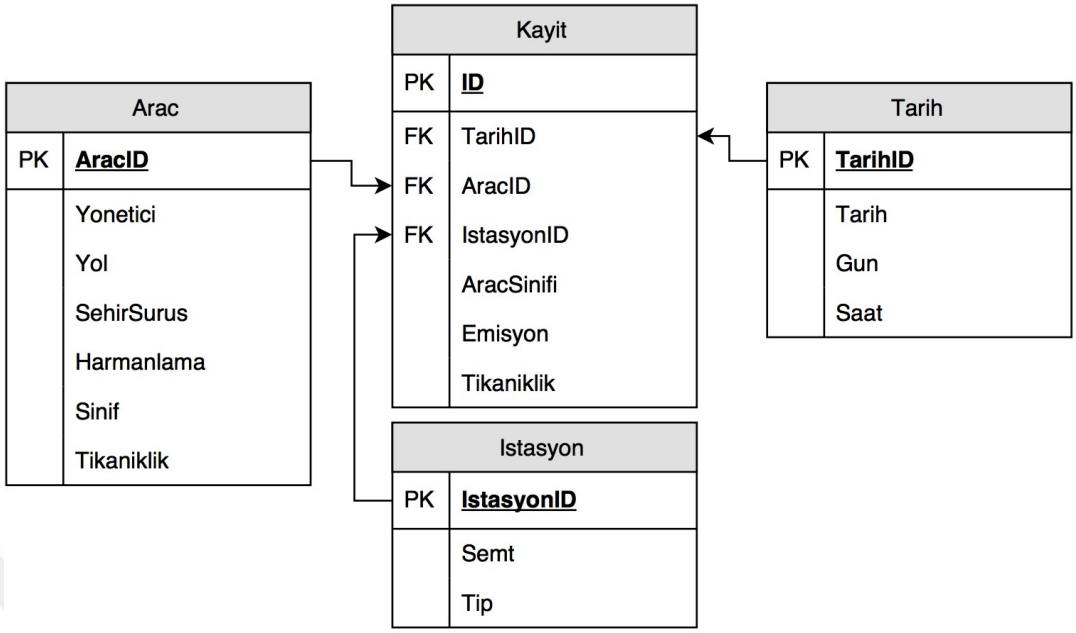


Şekil 4.31: Veritabanı 10 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.31'de diyagram olarak sunulmuştur.

4.2.11. Veritabanı 11

Veritabanı 11 için varlık-ilişki diyagramı Şekil 4.32'de gösterilmektedir.



Şekil 4.32: Veritabanı 11'in Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.34'teki gibidir.

Tablo 4.34: Veritabanı 11 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Yonetici	V1
Yol	V2
SehirSurus	V3
Harmanlama	V4
Sinif	V5
Tikaniklik	V6
Semt	V7
Tip	V8
Tarih	V9
Gun	V10
Saat	V11
AracSinifi	V12
Emisyon	V13
Tikaniklik	V14

Veri seti, 14 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 43 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.55}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 14$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 14 \rceil \, 14 = 56 \tag{4.56}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 14 \rceil = 4 \tag{4.57}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$f_1 w_1 = 0{,}1611258$$

$$f_2 w_2 = 0{,}01048489$$

$$f_3 w_3 = 0{,}1521741 \tag{4.58}$$

$$f_4 w_4 = 0$$

Buna göre;

$$F = 0{,}3237847 \tag{4.59}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.35'teki gibidir.

Tablo 4.35: Veritabanı 11 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	25
En yüksek	107
Medyan	51
MAD	16,3086

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

10101010101010101010100001000111111111111010101010101

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.33'te sunulmaktadır.

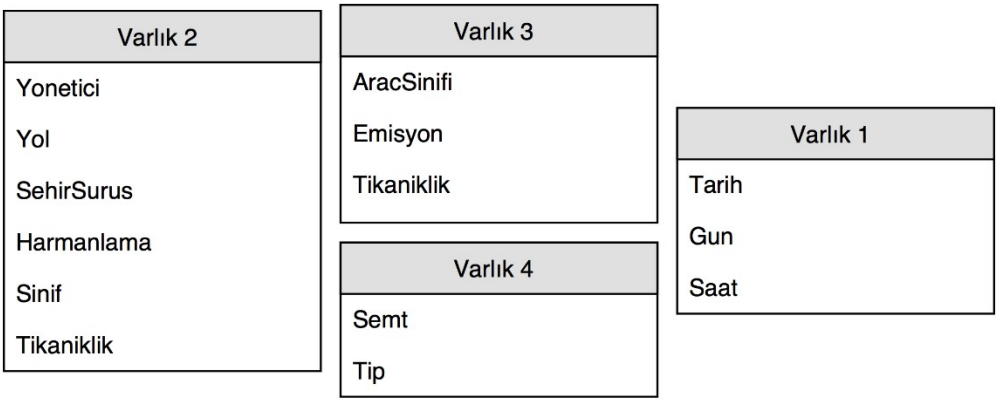
```
> anStructure(mych,rawdata)
[1]  9 10 11
[1] 1 2 3 4 5 6
[1] 12 13 14
[1] 7 8
```

Şekil 4.33: Veritabanı 11 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.33'te elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 9, 10 ve 11 birinci varlık içerisinde, nitelik 1, 2, 3, 4, 5 ve 6 ikinci varlık içerisinde, nitelik 12, 13 ve 14 üçüncü varlık içerisinde, nitelik 7 ve 8 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.36'da nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.36: Veritabanı 11 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	9, 10, 11	Tarih, Gun, Saat
Varlık 2	1, 2, 3, 4, 5, 6	Yonetici, Yol, SehirSurus, Harmanlama, Sinif, Tikaniklik
Varlık 3	12, 13, 14	AracSinifi, Emisyon, Tikaniklik
Varlık 4	7, 8	Semt, Tip

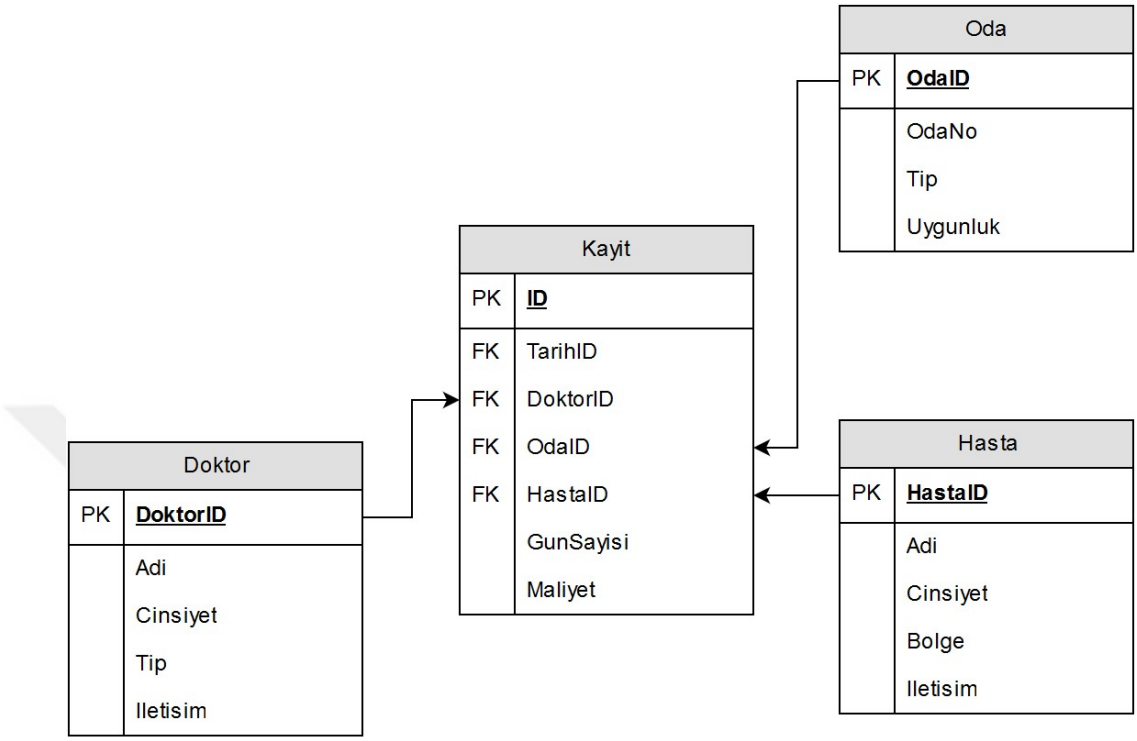


Şekil 4.34: Veritabanı 11 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.34'te diyagram olarak sunulmuştur.

4.2.12. Veritabanı 12

Veritabanı 12 için varlık-ilişki diyagramı Şekil 4.35'te gösterilmektedir.



Şekil 4.35: Veritabanı 12'nin Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.37'deki gibidir.

Tablo 4.37: Veritabanı 12 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
OdaNo	V1
Tip	V2
Uygunluk	V3
Adi	V4
Cinsiyet	V5
Tip	V6
Iletisim	V7
Adi	V8
Cinsiyet	V9
Bolge	V10
Iletisim	V11
GunSayisi	V12

Maliyet	V13

Veri seti, 13 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 52 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.60}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 13$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 13 \rceil \, 13 = 52 \tag{4.61}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 13 \rceil = 4 \tag{4.62}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1508909 \\ f_2 w_2 &= 0{,}07512464 \\ f_3 w_3 &= 0{,}1521741 \\ f_4 w_4 &= 0 \end{aligned} \tag{4.63}$$

Buna göre;

$$F = 0{,}3781896 \tag{4.64}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.38'deki gibidir.

Tablo 4.38: Veritabanı 12 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	37
En yüksek	752
Medyan	108,5
MAD	69,6822

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

101010101010011101110111011111100010111011100010010

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.36'da sunulmaktadır.

```
> anStructure(mych,rawdata)
[1]  8 10 11
[1] 1 2 3
[1] 4 5 6 7
[1]  9 12 13
```

Şekil 4.36: Veritabanı 12 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.36'da elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 8, 10 ve 11 birinci varlık içerisinde, nitelik 1, 2 ve 3 ikinci varlık içerisinde, nitelik 4, 5, 6 ve 7 üçüncü varlık içerisinde, nitelik 9, 12 ve 13 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.39'da nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.39: Veritabanı 12 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	8, 10, 11	Adi, Bolge, Iletisim
Varlık 2	1, 2, 3	OdaNo, Tip, Uygunluk
Varlık 3	4, 5, 6, 7	Adi, Cinsiyet, Tip, Iletisim
Varlık 4	9, 12, 13	Cinsiyet, GunSayisi, Maliyet

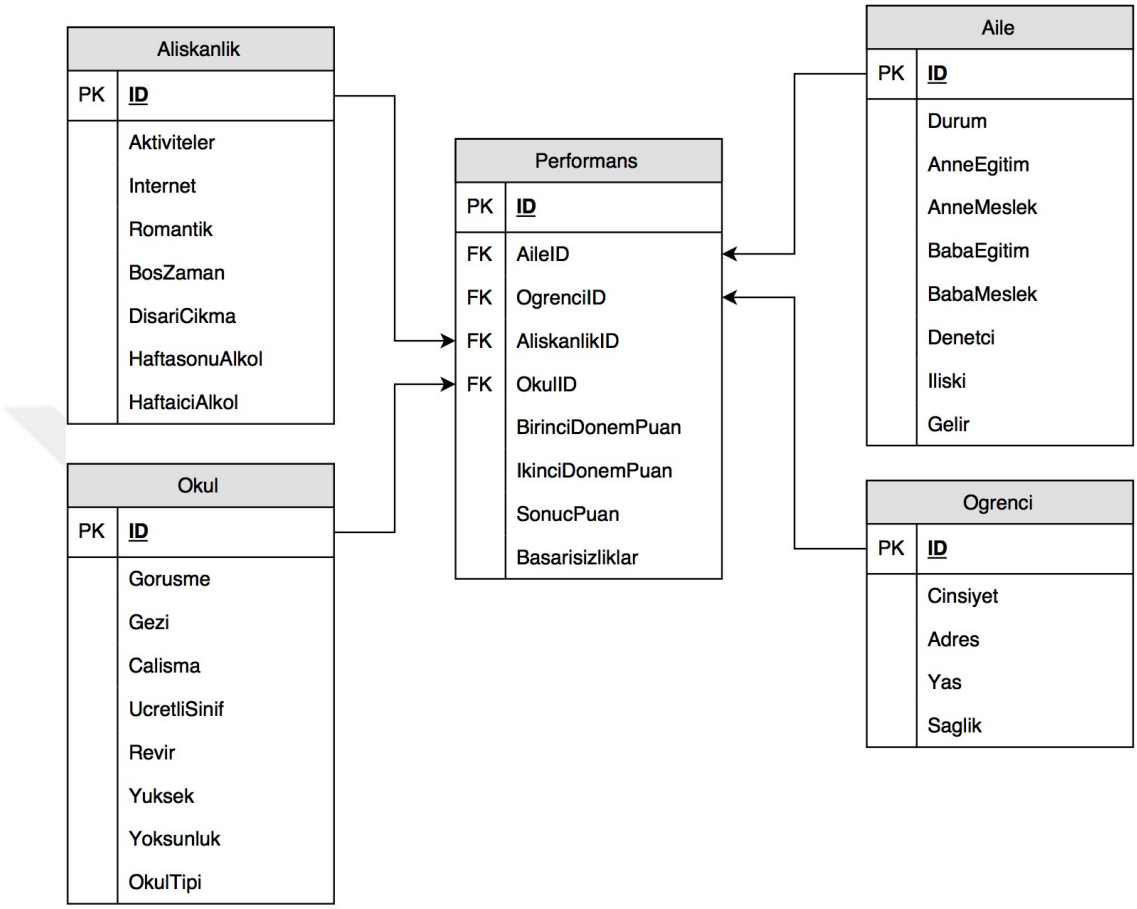
Şekil 4.37: Veritabanı 12 İçin Elde Edilen Çözümün Diyagramı.

Bu uygulama için algoritma, başlangıçta ele alınan veritabanı tasarımına çok yakın ancak birebir aynı olmayan bir tasarım önermiştir. nitelik 9, Varlık 1 içerisinde yer almalıyken algoritma Varlık 4 içerisinde sunmuştur. Bunun sebebi nitelik 9'un içerdiği verinin yapısıdır. Nitelik 9, cinsiyeti belirttiği için yalnızca 2 farklı değer alabilmekte, bu da veri tekrarını çok büyük oranda arttırmaktadır.

Algoritma, çok fazla veri tekrarı olduğunu düşündüğü bu niteliği, sayısal değerler içeren varlıklar olan GunSayisi ve Maliyet nitelikleri ile aynı varlık içerisine eklemiştir. Bu sayede Varlık 4'ün bir sayısal varlık olması durumunun da önüne geçerek optimizasyonu tamamlamıştır. Elde edilen çözüm Şekil 4.37'de diyagram olarak sunulmuştur.

4.2.13. Veritabanı 13

Veritabanı 13 için varlık-ilişki diyagramı Şekil 4.38'de gösterilmektedir.



Şekil 4.38: Veritabanı 13'ün Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.40'taki gibidir.

Tablo 4.40: Veritabanı 13 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Aktiviteler	V1
Internet	V2
Romantik	V3
BosZaman	V4
DisariCikma	V5
HaftasonuAlkol	V6
HaftaiciAlkol	V7
Durum	V8

AnneEgitim	V9
AnneMeslek	V10
BabaEgitim	V11
BabaMeslek	V12
Denetci	V13
Iliski	V14
Gelir	V15
Gorusme	V16
Gezi	V17
Calisma	V18
UcretliSinif	V19
Revir	V20
Yuksek	V21
Yoksunluk	V22
OkulTipi	V23
Cinsiyet	V24
Adres	V25
Yas	V26
Saglik	V27
BirinciDonemPuan	V28
IkinciDonemPuan	V29
SonucPuan	V30
Basarisizliklar	V31

Veri seti, 31 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 71 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.65}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 31$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 31 \rceil \, 31 = 155 \tag{4.66}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 31 \rceil = 5 \tag{4.67}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$f_1 w_1 = 0{,}08834916$$
$$f_2 w_2 = 0{,}2149003$$
$$f_3 w_3 = 0{,}1141306$$
$$f_4 w_4 = 0{,}02796848 \qquad 4.68$$

Buna göre;

$$F = 0{,}4453485 \qquad 4.69$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.41'deki gibidir.

Tablo 4.41: Veritabanı 13 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	169
En yüksek	844
Medyan	350,5
MAD	131,9514

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

101001010010100101001010010100101001100111001110011100111
001110011100111001110011101111011110111011110111011101111101
111011110111001111101111011110110110101101011010110

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.39'da sunulmaktadır.

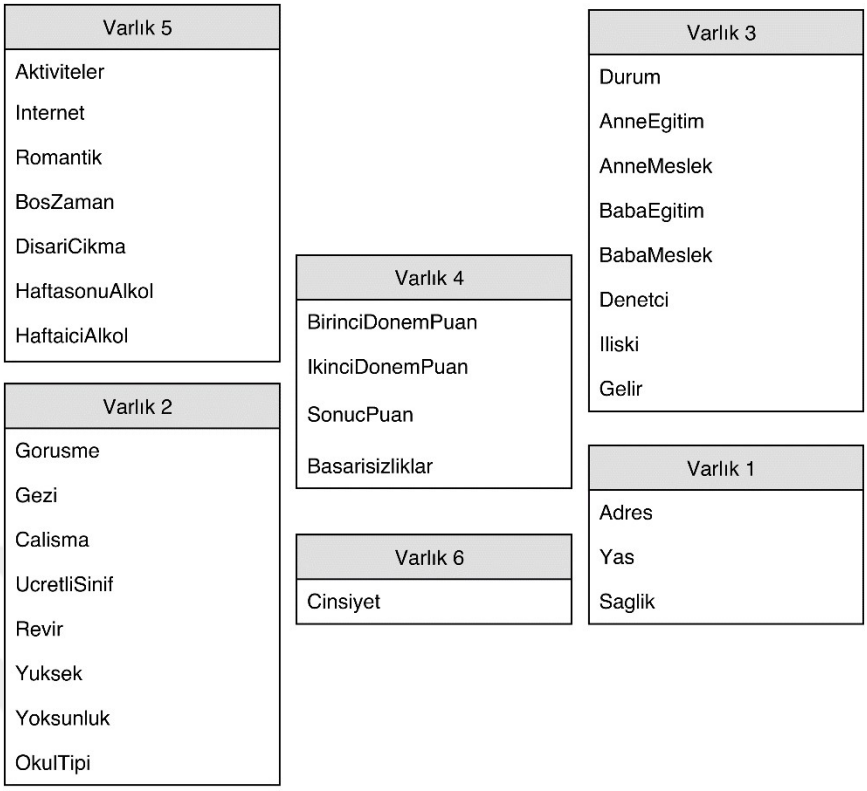
```
> anStructure(mych,rawdata)
[1] 25 26 27
[1] 16 17 18 19 20 21 22 23
[1]  8  9 10 11 12 13 14 15
[1] 28 29 30 31
[1] 1 2 3 4 5 6 7
[1] 24
```

Şekil 4.39: Veritabanı 13 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.39'da elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 25, 26 ve 27 birinci varlık içerisinde, nitelik 16, 17, 18, 19, 20, 21, 22 ve 23 ikinci varlık içerisinde, nitelik 8, 9, 10, 11, 12, 13, 14 ve 15 üçüncü varlık içerisinde, nitelik 28, 29, 30 ve 31 dördüncü varlık içerisinde, nitelik 1, 2, 3, 4, 5, 6 ve 7 beşinci varlık içerisinde, nitelik 24 ise altıncı varlık içerisinde bulunmalıdır. Tablo 4.42'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.42: Veritabanı 13 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	25, 26, 27	Adres, Yas, Saglik
Varlık 2	16, 17, 18, 19, 20, 21, 22, 23	Gorusme, Gezi, Calisma, UcretliSinif, Revir, Yuksek, Yoksunluk, OkulTipi
Varlık 3	8, 9, 10, 11, 12, 13, 14, 15	Durum, AnneEgitim, AnneMeslek, BabaEgitim, BabaMeslek, Denetci, Iliski, Gelir
Varlık 4	28, 29, 30, 31	BirinciDonemPuan, IkinciDonemPuan, SonucPuan, Basarisizliklar
Varlık 5	1, 2, 3, 4, 5, 6, 7	Aktiviteler, Internet, Romantik, BosZaman, DisariCikma, HaftasonuAlkol, HaftaiciAlkol
Varlık 6	24	Cinsiyet

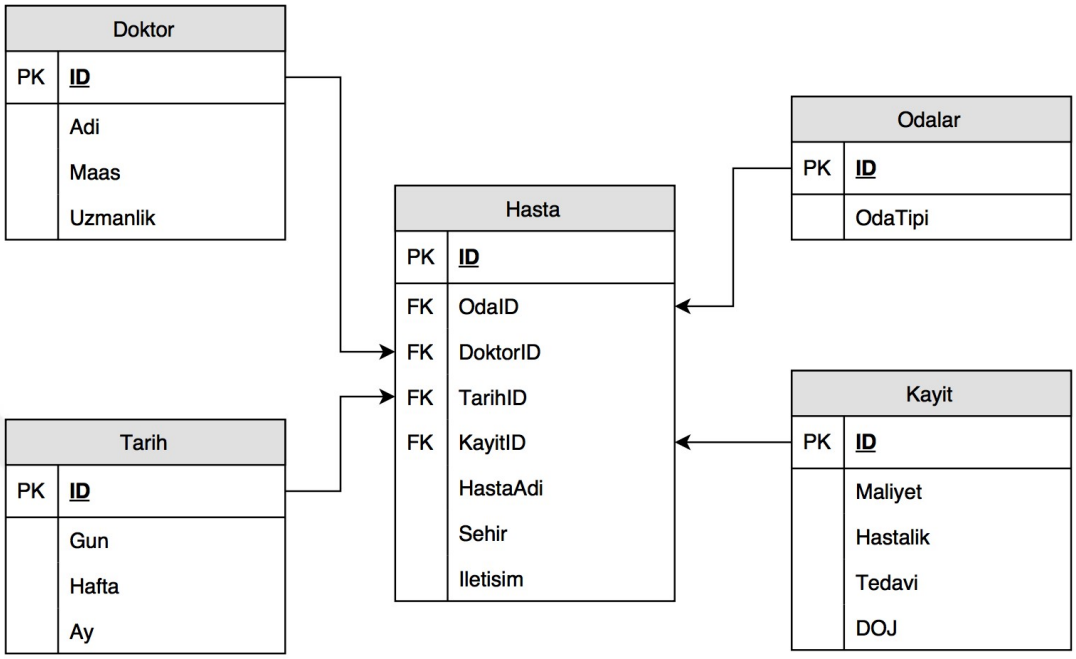


Şekil 4.40: Veritabanı 13 İçin Elde Edilen Çözümün Diyagramı.

Bu örnekte algoritma, başlangıçta ele alınan veritabanı tasarımına çok yakın ancak aynı olmayan bir tasarım önerisi sunmuştur. Veritabanı 12'deki duruma benzer olarak Cinsiyet (24) niteliği, çok fazla veri tekrarı barındırması sebebiyle Varlık 1 içerisinde yer alması gerekirken algoritma tarafından yeni bir varlık olarak ele alınmıştır. Elde edilen çözüm Şekil 4.40'ta diyagram olarak sunulmuştur.

4.2.14. Veritabanı 14

Veritabanı 14 için varlık-ilişki diyagramı Şekil 4.41'de gösterilmektedir.



Şekil 4.41: Veritabanı 14'ün Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.43'teki gibidir.

Tablo 4.43: Veritabanı 14 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Adi	V1
Maas	V2
Uzmanlik	V3
OdaTipi	V4
Gun	V5
Hafta	V6
Ay	V7
Maliyet	V8
Hastalik	V9
Tedavi	V10
DOJ	V11
HastaAdi	V12
Sehir	V13
Iletisim	V14

Veri seti, 14 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 78 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.70}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 14$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 14 \rceil \, 14 = 56 \tag{4.71}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 14 \rceil = 4 \tag{4.72}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve 94 tanesinde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1601499 \\ f_2 w_2 &= 0{,}0665374 \\ f_3 w_3 &= 0{,}1521741 \\ f_4 w_4 &= 0{,}0221986 \end{aligned} \tag{4.73}$$

Buna göre;

$$F = 0{,}40106 \tag{4.74}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.44'teki gibidir.

Tablo 4.44: Veritabanı 14 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	26
En yüksek	920
Medyan	62,5
MAD	25,2042

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

0110011001100000100110011001001000100010001000000000000000

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.42'de sunulmaktadır.

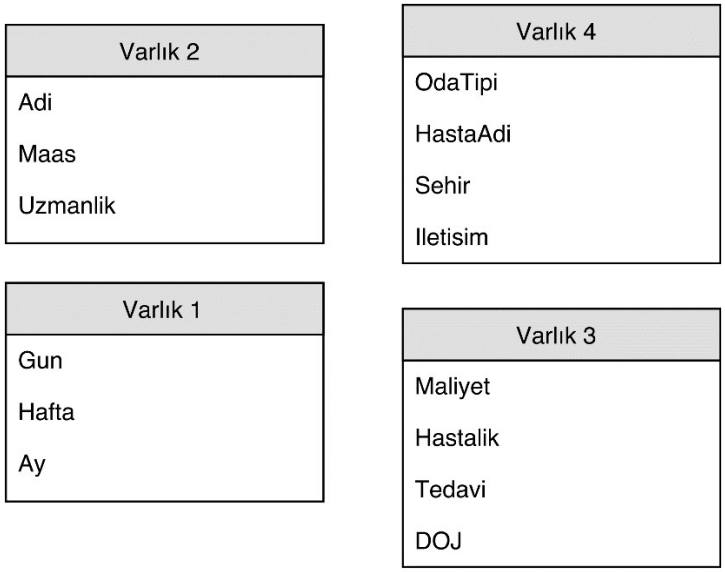
```
> anStructure(mych,myfile)
[1] 5 6 7
[1] 1 2 3
[1]  8  9 10 11
[1]  4 12 13 14
```

Şekil 4.42: Veritabanı 14 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.42'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 5, 6 ve 7 birinci varlık içerisinde, nitelik 1, 2 ve 3 ikinci varlık içerisinde, nitelik 8, 9, 10 ve 11 üçüncü varlık içerisinde, nitelik 4, 12, 13 ve 14 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.45'te nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.45: Veritabanı 14 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	5, 6, 7	Gun, Hafta, Ay
Varlık 2	1, 2, 3	Adi, Maas, Uzmanlik
Varlık 3	8, 9, 10, 11	Maliyet, Hastalik, Tedavi, DOJ
Varlık 4	4, 12, 13, 14	OdaTipi, HastaAdi, Sehir, Iletisim



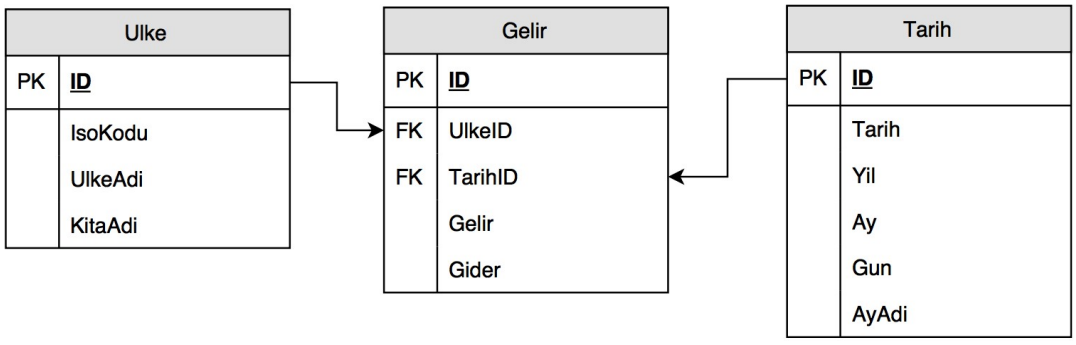
Şekil 4.43: Veritabanı 14 İçin Elde Edilen Çözümün Diyagramı.

Algoritma, başlangıçta ele alınan veritabanı tasarımına çok yakın ancak farklı bir tasarım önerisi sunmuştur. Nitelik 4, farklı bir varlık içerisinde yer almalıyken algoritma içerisinde Varlık 4 içerisinde önerilmiştir.

Hata sebebi, nitelik 4'ün içerdiği değerlerin veri tekrarı yapması ve tablo sayısının azaltılmaya çalışılması olarak görülebilir. Elde edilen çözüm Şekil 4.43'te diyagram olarak sunulmuştur.

4.2.15. Veritabanı 15

Veritabanı 15 için varlık-ilişki diyagramı Şekil 4.44'te gösterilmektedir.



Şekil 4.44: Veritabanı 15'in Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT

SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.46'daki gibidir.

Tablo 4.46: Veritabanı 15 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	**Niteliğin Veri Setindeki Adı**
IsoKodu	V1
UlkeAdi	V2
KitaAdi	V3
Tarih	V4
Yil	V5
Ay	V6
Gun	V7
AyAdi	V8
Gelir	V9
Gider	V10

Veri seti, 10 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 38 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.75}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 10$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 10 \rceil \, 10 = 30 \tag{4.76}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 10 \rceil = 3 \tag{4.77}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$f_1 w_1 = 0{,}1117314 \tag{4.78}$$

$$f_2 w_2 = 0{,}06109624$$

$$f_3 w_3 = 0{,}1141306$$

$$f_4 w_4 = 0{,}02176708$$

Buna göre;

$$F = 0{,}3087252 \qquad 4.79$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.47'deki gibidir.

Tablo 4.47: Veritabanı 15 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	11
En yüksek	55
Medyan	28
MAD	9,6369

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

$$10101010101001100110011001100110011001110111$$

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.45'te sunulmaktadır.

```
> anStructure(mych,myfile)
[1] 1 2 3
[1]  9 10
[1] 4 5 6 7 8
```

Şekil 4.45: Veritabanı 15 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.45'te elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 1, 2 ve 3 birinci varlık içerisinde, nitelik 9 ve 10 ikinci varlık içerisinde, nitelik 4, 5, 6, 7 ve 8

üçüncü varlık içerisinde bulunmalıdır. Tablo 4.48'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.48: Veritabanı 15 İçin Elde Edilen Çözüm.

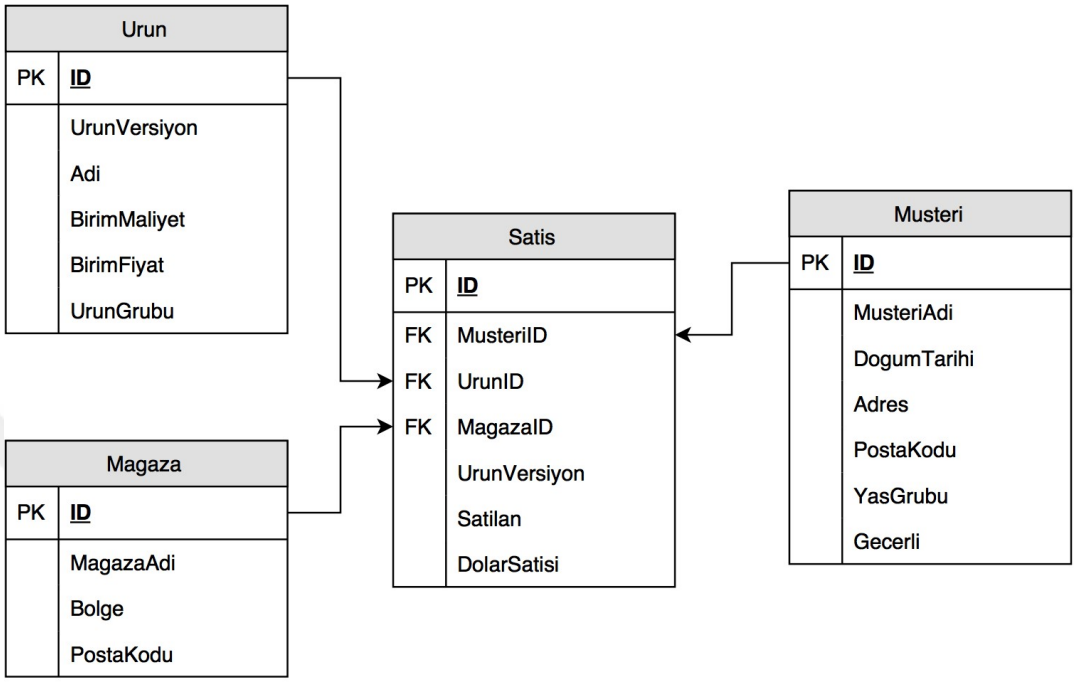
Varlık	Nitelikler	Etiketleri
Varlık 1	1, 2, 3	IsoKodu, UlkeAdi, KitaAdi
Varlık 2	9, 10	Gelir, Gider
Varlık 3	4, 5, 6, 7, 8	Tarih, Yil, Ay, Gun, AyAdi

Şekil 4.46: Veritabanı 15 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.46'da diyagram olarak sunulmuştur.

4.2.16. Veritabanı 16

Veritabanı 16 için varlık-ilişki diyagramı Şekil 4.47'de gösterilmektedir.



Şekil 4.47: Veritabanı 16'nın Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.49'daki gibidir.

Tablo 4.49: Veritabanı 16 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
UrunVersiyon	V1
Adi	V2
BirimMaliyet	V3
BirimFiyat	V4
UrunGrubu	V5
MusteriAdi	V6
DogumTarihi	V7
Adres	V8
PostaKodu	V9
YasGrubu	V10
Gecerli	V11
MagazaAdi	V12
Bolge	V13
PostaKodu	V14

UrunVersiyon	V15
Satilan	V16
DolarSatisi	V17

Veri seti, 17 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 67 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.80}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 17$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 17 \rceil \, 17 = 85 \tag{4.81}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 17 \rceil = 5 \tag{4.82}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1112719 \\ f_2 w_2 &= 0{,}0157103 \\ f_3 w_3 &= 0{,}07608703 \\ f_4 w_4 &= 0{,}0221986 \end{aligned} \tag{4.83}$$

Buna göre;

$$F = 0{,}2252678 \tag{4.84}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.50'deki gibidir.

Tablo 4.50: Veritabanı 16 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	61
En yüksek	387
Medyan	113,5
MAD	39,2889

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

01110011100111001110011101101011010110110
10110101101000001000010000100111001110011 1

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.48'de sunulmaktadır.

```
> anStructure(mych,myfile)
[1]  6  7  8  9 10 11
[1] 1 2 3 4 5
[1] 15 16 17
[1] 12 13 14
```

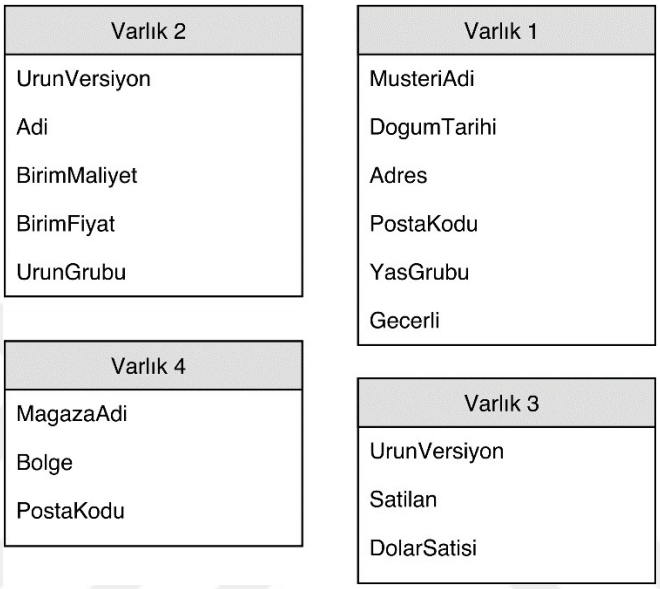
Şekil 4.48: Veritabanı 16 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.48'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 6, 7, 8, 9, 10 ve 11 birinci varlık içerisinde, nitelik 1, 2, 3, 4 ve 5 ikinci varlık içerisinde, nitelik 15, 16 ve 17 üçüncü varlık içerisinde, nitelik 12, 13 ve 14 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.51'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.51: Veritabanı 16 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	6, 7, 8, 9, 10, 11	MusteriAdi, DogumTarihi, Adres, PostaKodu, YasGrubu, Gecerli

Varlık 2	1, 2, 3, 4, 5	UrunVersiyon, Adi, BirimMaliyet, BirimFiyat, UrunGrubu
Varlık 3	15, 16, 17	UrunVersiyon, Satilan, DolarSatisi
Varlık 4	12, 13, 14	MagazaAdi, Bolge, PostaKodu

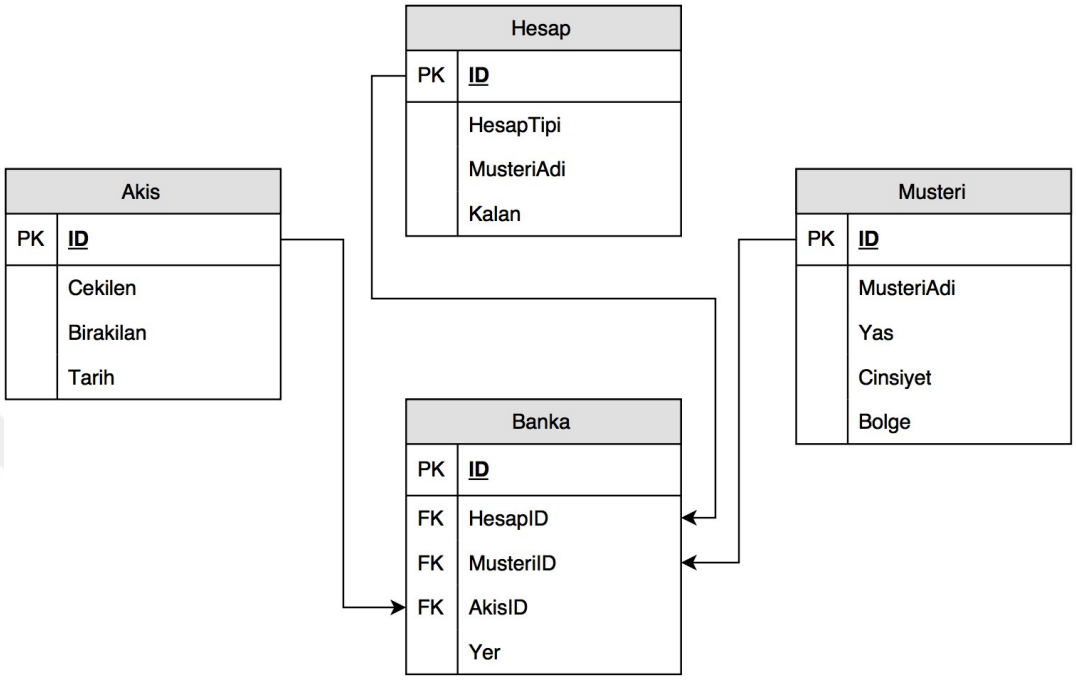


Şekil 4.49: Veritabanı 16 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.49'da diyagram olarak sunulmuştur.

4.2.17. Veritabanı 17

Veritabanı 17 için varlık-ilişki diyagramı Şekil 4.50'de gösterilmektedir.



Şekil 4.50: Veritabanı 17'nin Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.52'deki gibidir.

Tablo 4.52: Veritabanı 17 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
HesapTipi	V1
MusteriAdi	V2
Kalan	V3
Cekilen	V4
Birakilan	V5
Tarih	V6
MusteriAdi	V7
Yas	V8
Cinsiyet	V9
Bolge	V10
Yer	V11

Veri seti, 11 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 64 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.85}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 11$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 11 \rceil \, 11 = 44 \tag{4.86}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 11 \rceil = 4 \tag{4.87}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}08996551 \\ f_2 w_2 &= 0{,}03166981 \\ f_3 w_3 &= 0{,}1521741 \\ f_4 w_4 &= 0 \end{aligned} \tag{4.88}$$

Buna göre;

$$F = 0{,}2738094 \tag{4.89}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.53'teki gibidir.

Tablo 4.53: Veritabanı 17 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	16
En yüksek	75

Medyan	33,5
MAD	12,6021

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

00110011001111101110111010111011101110110000

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.51'de sunulmaktadır.

```
> anStructure(mych,myfile)
[1] 4 5 6
[1]  7  8  9 10
[1] 1 2 3
[1] 11
```

Şekil 4.51: Veritabanı 17 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.51'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 4, 5 ve 6 birinci varlık içerisinde, nitelik 7, 8, 9 ve 10 ikinci varlık içerisinde, nitelik 1, 2 ve 3 üçüncü varlık içerisinde, nitelik 11 dördüncü varlık içerisinde bulunmalıdır. Tablo 4.54'te nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.54: Veritabanı 17 İçin Elde Edilen Çözüm.

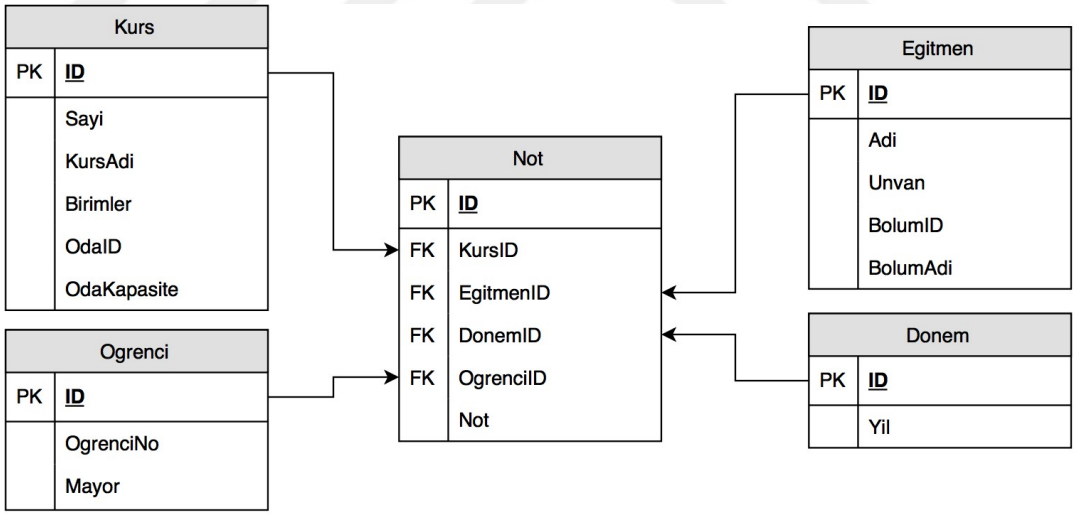
Varlık	Nitelikler	Etiketleri
Varlık 1	4, 5, 6	Cekilen, Birakilan, Tarih
Varlık 2	7, 8, 9, 10	MusteriAdi, Yas, Cinsiyet, Bolge
Varlık 3	1, 2, 3	HesapTipi, MusteriAdi, Kalan
Varlık 4	11	Yer

Şekil 4.52: Veritabanı 17 İçin Elde Edilen Çözümün Diyagramı.

Algoritma tarafından önerilen veritabanı tasarımı incelendiğinde, başlangıçta ele alınan veritabanı tasarımındaki varlık-nitelik dizilimine uygun olduğu görülmektedir. Bu örnekte, algoritma ham veri setini oluşturmak için kullanılan veritabanı tasarımına ulaşmıştır. Elde edilen çözüm Şekil 4.52'de diyagram olarak sunulmuştur.

4.2.18. Veritabanı 18

Veritabanı 18 için varlık-ilişki diyagramı Şekil 4.53'te gösterilmektedir.



Şekil 4.53: Veritabanı 18'in Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.55'teki gibidir.

Tablo 4.55: Veritabanı 18 İçin Varlık-İlişki Diyagramı.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Sayi	V1
KursAdi	V2
Birimler	V3
OdaID	V4
OdaKapasite	V5
Adi	V6
Unvan	V7
BolumID	V8
BolumAdi	V9
OgrenciNo	V10
Mayor	V11
Yil	V12
Not	V13

Veri seti, 13 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 48 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.90}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 13$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 13 \rceil \, 13 = 52 \tag{4.91}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 13 \rceil = 4 \tag{4.92}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$f_1 w_1 = 0{,}05668858 \tag{4.93}$$

$$f_2 w_2 = 0{,}02559185$$

$$f_3 w_3 = 0{,}2282611$$

$$f_4 w_4 = 0{,}02614089$$

Buna göre;

$$F = 0{,}3366824 \qquad 4.94$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.56'daki gibidir.

Tablo 4.56: Veritabanı 18 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	12
En yüksek	154
Medyan	38,5
MAD	14,0847

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

$$0101010101010101010111011101110011001011101101101000$$

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.54'te sunulmaktadır.

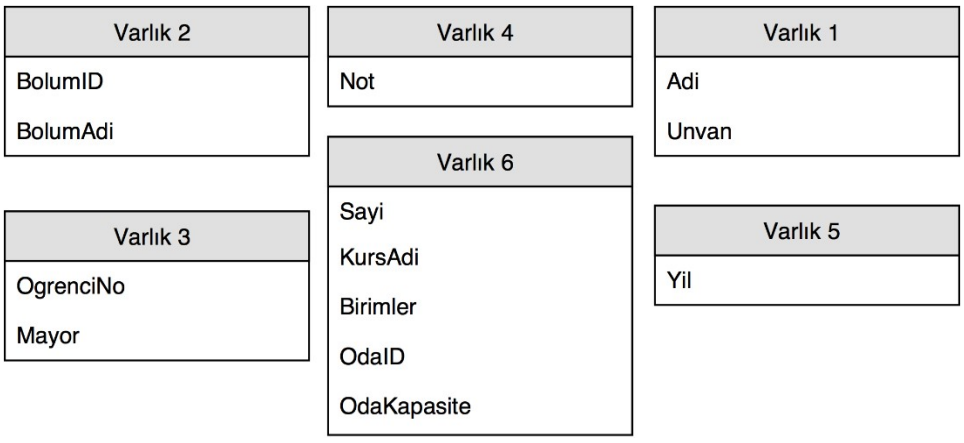
```
> anStructure(mych,myfile)
[1] 6 7
[1] 8 9
[1] 10 11
[1] 13
[1] 12
[1] 1 2 3 4 5
```

Şekil 4.54: Veritabanı 18 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.54'te elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 6 ve 7 birinci varlık içerisinde, nitelik 8 ve 9 ikinci varlık içerisinde, nitelik 10 ve 11 üçüncü varlık içerisinde, nitelik 13 dördüncü varlık içerisinde, nitelik 12 beşinci varlık içerisinde, nitelik 1, 2, 3, 4 ve 5 ise altıncı varlık içerisinde bulunmalıdır. Tablo 4.57'de nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.57: Veritabanı 18 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	6, 7	Adi, Unvan
Varlık 2	8, 9	BolumID, BolumAdi
Varlık 3	10, 11	OgrenciNo, Mayor
Varlık 4	13	Not
Varlık 5	12	Yil
Varlık 6	1, 2, 3, 4, 5	Sayi, KursAdi, Birimler, OdaID, OdaKapasite



Şekil 4.55: Veritabanı 18 İçin Elde Edilen Çözümün Diyagramı.

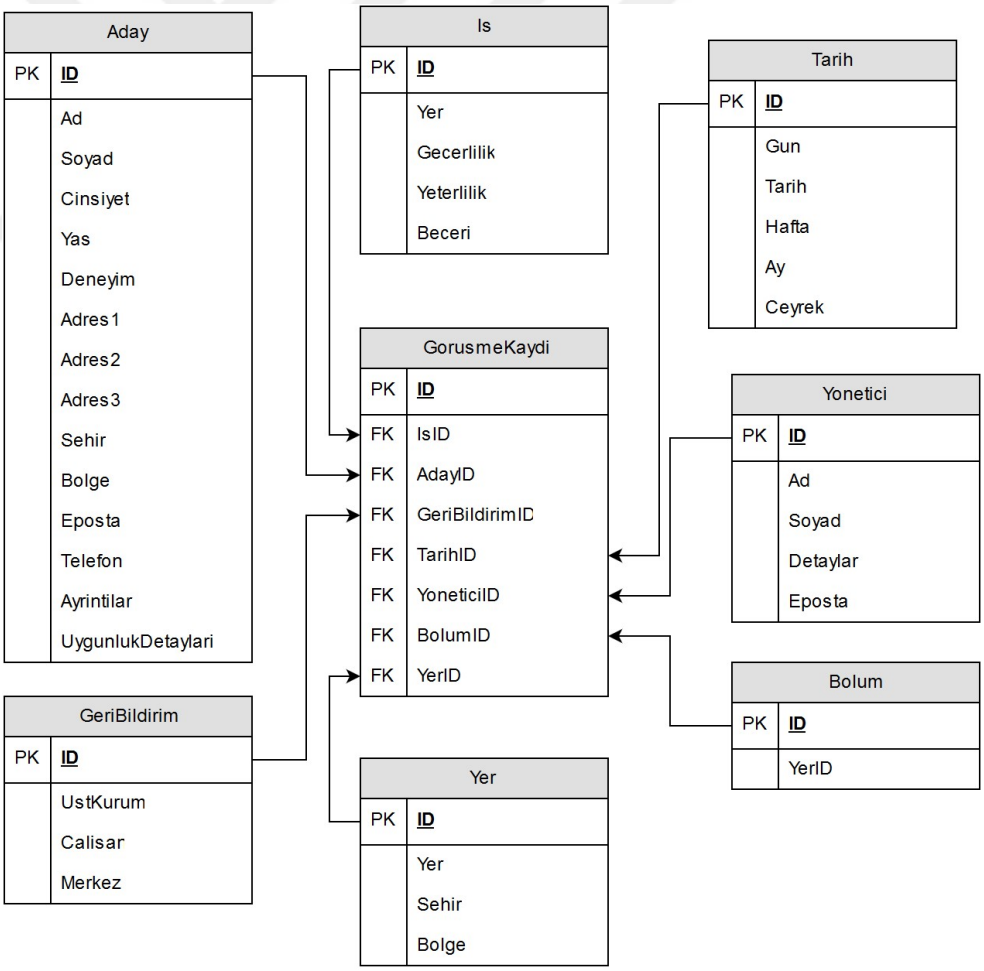
Bu örnek için algoritma, başlangıçtaki veritabanı tasarımına çok yakın ancak aynı olmayan bir tasarım önerisinde bulunmuştur. Algoritma, nitelik 6, 7, 8 ve 9'un aynı varlık

içerisinde yer alması beklenirken nitelik 6 ve 7'yi bir varlık içerisinde, nitelik 8 ve 9'u farklı bir nitelik içerisinde sunmuştur. Diğer varlıklar birebir aynı önerilmiştir.

Hatanın sebebi niteliklerin içerdiği değerlerden kaynaklanmaktadır. Nitelik 8 ve 9 bir bölümün ID'sini ve adını içermektedir. Verilerin de tekrarın da benzeşmesi sebebiyle algoritma bu iki niteliği farklı bir varlık yapma eğiliminde olmuştur. Bu açıdan değerlendirildiğinde aslında algoritmanın da hatalı bir karar verdiği söylenememektedir. BolumID ve BolumAdi, mantıksal olarak da bir varlık oluşturabilecek niteliktedirler. Elde edilen çözüm Şekil 4.55'te diyagram olarak sunulmuştur.

4.2.19. Veritabanı 19

Veritabanı 19 için varlık-ilişki diyagramı Şekil 4.56'da gösterilmektedir.



Şekil 4.56: Veritabanı 19'un Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT

SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.58'deki gibidir.

Tablo 4.58: Veritabanı 19 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
Ad	V1
Soyad	V2
Cinsiyet	V3
Yas	V4
Deneyim	V5
Adres1	V6
Adres2	V7
Adres3	V8
Sehir	V9
Bolge	V10
Eposta	V11
Telefon	V12
Ayrintilar	V13
UygunlukDetaylari	V14
Yer	V15
Gecerlilik	V16
Yeterlilik	V17
Beceri	V18
Yer	V19
Sehir	V20
Bolge	V21
UstKurum	V22
Calisan	V23
Merkez	V24
Gun	V25
Tarih	V26
Hafta	V27
Ay	V28
Ceyrek	V29
Ad	V30
Soyad	V31
Detaylar	V32
Eposta	V33
YerID	V34

Veri seti, 34 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 253 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.95}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 34$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 34 \rceil \, 34 = 204 \tag{4.96}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 34 \rceil = 6 \tag{4.97}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve 72 kere aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}04041347 \\ f_2 w_2 &= 0{,}01780702 \\ f_3 w_3 &= 0{,}1141306 \\ f_4 w_4 &= 0{,}02918477 \end{aligned} \tag{4.98}$$

Buna göre;

$$F = 0{,}2015358 \tag{4.99}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.59'daki gibidir.

Tablo 4.59: Veritabanı 19 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	243
En yüksek	999
Medyan	575
MAD	240,9225

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

1000110001100011111011110111101111011110111101111011110111101111011110011
1001110011100111001110011100111000100001000010000100001000010000100

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.57'de sunulmaktadır.

```
> anStructure(mych,myfile)
[1] 27
[1] 19 20 21
 [1]  1  2  4  5  6  7  8  9 10 11 12 13 14
[1] 22 23 24
[1] 25
[1] 29
[1] 28
[1] 3
[1] 30 31 32 33
[1] 15 16 17 18
[1] 26
[1] 34
```

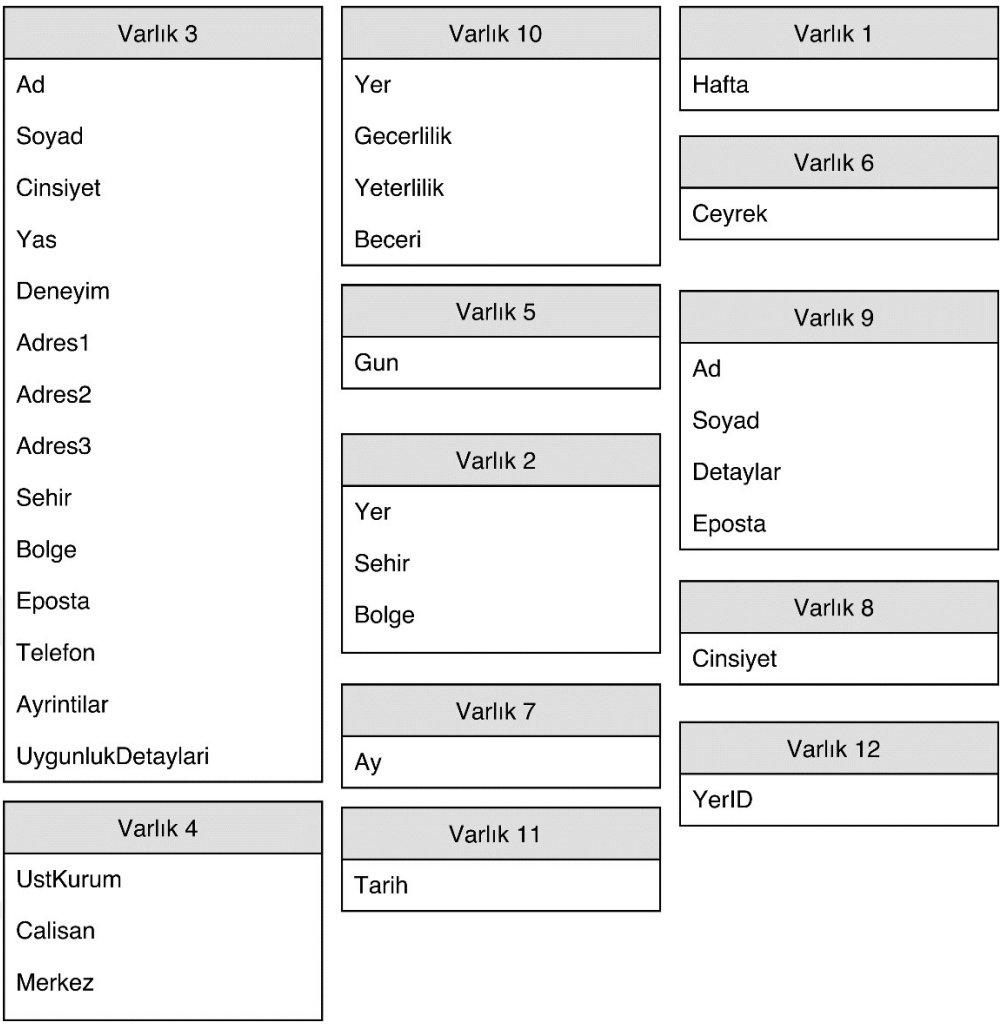
Şekil 4.57: Veritabanı 19 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.57'de elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 27 birinci varlık içerisinde, nitelik 19, 20 ve 21 ikinci varlık içerisinde, nitelik 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 ve 14 üçüncü varlık içerisinde, nitelik 22, 23 ve 24 dördüncü varlık içerisinde, nitelik 25 beşinci varlık içerisinde, nitelik 29 altıncı varlık içerisinde, nitelik 28 yedinci varlık içerisinde, nitelik 3 sekizinci varlık içerisinde, nitelik 30, 31, 32 ve 33 dokuzuncu varlık içerisinde, nitelik 15, 16, 17 ve 18 onuncu varlık içerisinde, nitelik 26 onbirinci varlık içerisinde, nitelik 34 ise onikinci varlık içerisinde bulunmalıdır. Tablo 4.60'ta nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.60: Veritabanı 19 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri

Varlık 1	27	Hafta
Varlık 2	19, 20, 21	Yer, Sehir, Bolge
Varlık 3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	Ad, Soyad, Cinsiyet, Yas, Deneyim, Adres1, Adres2, Adres3, Sehir, Bolge, Eposta, Telefon, Ayrintilar, UygunlukDetaylari
Varlık 4	22, 23, 24	UstKurum, Calisan, Merkez
Varlık 5	25	Gun
Varlık 6	29	Ceyrek
Varlık 7	28	Ay
Varlık 8	3	Cinsiyet
Varlık 9	30, 31, 32, 33	Ad, Soyad, Detaylar, Eposta
Varlık 10	15, 16, 17, 18	Yer, Gecerlilik, Yeterlilik, Beceri
Varlık 11	26	Tarih
Varlık 12	34	YerID



Şekil 4.58: Veritabanı 19 İçin Elde Edilen Çözümün Diyagramı.

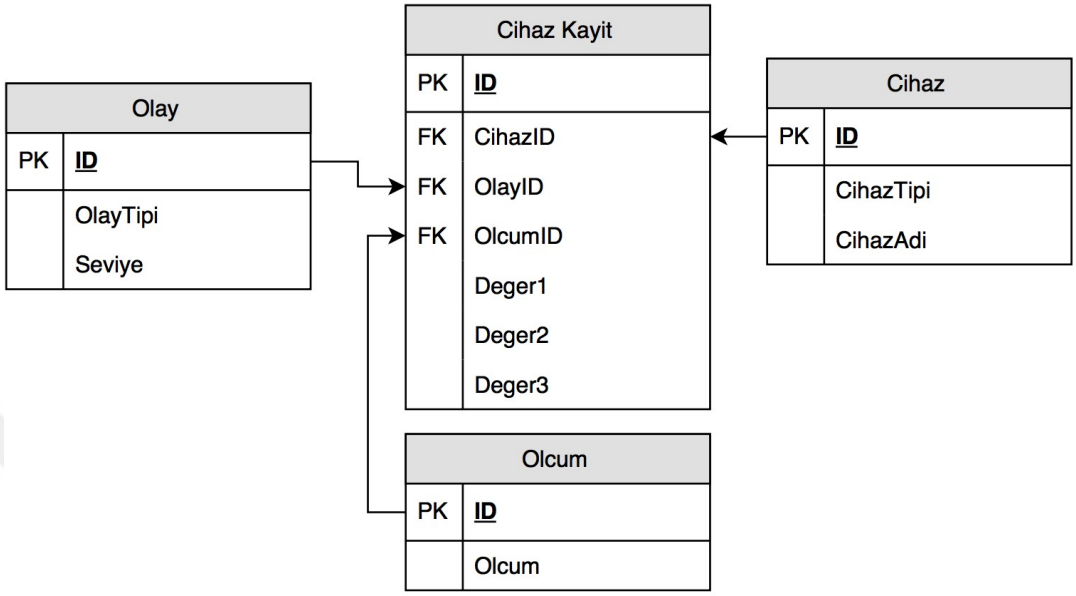
Bu uygulama için algoritma, başlangıçta elde bulunan veritabanı tasarımına göre farklı bir tasarım önerisinde bulunmuştur. Farklılıklar iki durum olarak incelenebilir.

Birinci durum Tarih varlığının oluşturulamamasıdır. Algoritma, Tarih varlığındaki veri tekrarlarından dolayı nitelikleri yeni varlık olarak ele alma eğiliminde olmuştur. Bu sayede veri tekrarını çok daha azaltabilmektedir. Tablo sayısının artma maliyeti, veri tekrarından daha kabul edilebilir düzeyde olduğundan algoritma bu nitelikleri birer varlık içerisine yerleştirerek veri tekrarından kurtulmuştur.

İkinci durum Cinsiyet (3) niteliğidir. Niteliğin içerdiği veri yapısı sebebiyle algoritma bu niteliği de yeni bir varlık olarak değerlendirmiştir. Elde edilen çözüm Şekil 4.58'de diyagram olarak sunulmuştur.

4.2.20. Veritabanı 20

Veritabanı 20 için varlık-ilişki diyagramı Şekil 4.59'da gösterilmektedir.



Şekil 4.59: Veritabanı 20'nin Varlık-İlişki Diyagramı.

Şekilde verilen veritabanı tasarımına uygun olarak varlıklar oluşturulmuş ve rassal veriler eklenmiştir. Elde edilen yapay varlıklar, tasarımda verilen ilişkiler kullanılarak SELECT SQL sorgusu kullanılarak ham veri setine dönüştürülmüştür. Niteliklerin ham veri seti için dönüştürülen isimleri Tablo 4.61'deki gibidir.

Tablo 4.61: Veritabanı 20 İçin Nitelik İsimleri Dönüşümü.

Niteliğin Tasarımdaki Adı	Niteliğin Veri Setindeki Adı
OlayTipi	V1
Seviye	V2
CihazTipi	V3
CihazAdi	V4
Olcum	V5
Deger1	V6
Deger2	V7
Deger3	V8

Veri seti, 8 nitelik ve 500 kayıttan oluşmaktadır. Ayrıca dosya boyutu 28 kilobayttır.

Kromozom yapısının belirlenebilmesi ve uygulamanın gerçekleştirilebilmesi için gerekli parametreler, aşağıdaki şekilde hesaplanmıştır. Başlık 2.3.4.1 içerisinde kromozom uzunluğunu (KU) hesaplamak için;

$$KU = \lceil \log_2 HVS \rceil \, HVS \tag{4.100}$$

eşitliği verilmişti. Buna göre bu veritabanı için kromozom uzunluğu $HVS = 8$ için;

$$KU = \lceil \log_2 HVS \rceil \, HVS = \lceil \log_2 8 \rceil \, 8 = 24 \tag{4.101}$$

olarak hesaplanmaktadır. Kromozomdaki her bir gen uzunluğu ise;

$$m = \lceil \log_2 HVS \rceil = \lceil \log_2 8 \rceil = 3 \tag{4.102}$$

olarak hesaplanmaktadır. Bu parametrelerle 100 kez arama gerçekleştirilmiş ve tümünde aynı sonuca ulaşılmıştır. Ulaşılan değerler aşağıdaki gibidir.

$$\begin{aligned} f_1 w_1 &= 0{,}1952752 \\ f_2 w_2 &= 0{,}01668755 \\ f_3 w_3 &= 0{,}2282611 \\ f_4 w_4 &= 0 \end{aligned} \tag{4.103}$$

Buna göre;

$$F = 0{,}4402238 \tag{4.104}$$

olmuştur. Ayrıca en iyi sonuca erişilen nesil sayısı incelendiğinde elde edilen sonuçlar Tablo 4.62'deki gibidir.

Tablo 4.62: Veritabanı 20 İçin En İyi Çözüme Ulaştıran İterasyon Sayıları.

En düşük	1
En yüksek	54
Medyan	9
MAD	4,4478

Her bir kromozom çözüm önerisi olarak bir veritabanı tasarımı sunmaktadır ancak farklı kromozomlar da aynı tasarımı işaret ediyor olabilir. Kromozom değişse de aynı tasarım için elde edilecek amaç fonksiyonundan dönen değer değişmemektedir. Ulaşılan çözüm önerilerinden biri

101101001001011011011011

kromozomu olarak elde edilmiştir.

Kromozomun önerdiği veritabanı tasarımı Şekil 4.60'ta sunulmaktadır.

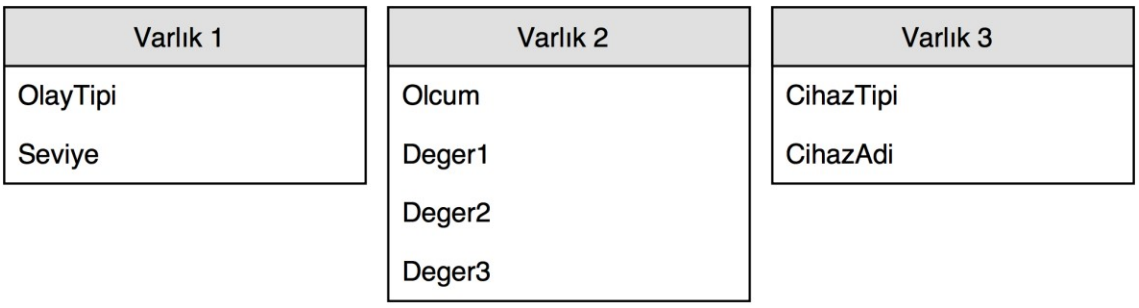
```
> anStructure(mych,myfile)
[1] 1 2
[1] 5 6 7 8
[1] 3 4
```

Şekil 4.60: Veritabanı 20 İçin Ulaşılan Veritabanı Tasarımı Önerisi.

Şekil 4.60'ta elde edilen çözüm önerisinde her bir satır bir varlığı, her bir sayı ise o varlık içerisinde yer alması gereken niteliği ifade etmektedir. Bu örnek için nitelik 1 ve 2 birinci varlık içerisinde, nitelik 5, 6, 7 ve 8 ikinci varlık içerisinde, nitelik 3 ve 4 üçüncü varlık içerisinde bulunmalıdır. Tablo 4.63'te nitelik etiketleriyle birlikte veritabanı tasarımı tekrar sunulmaktadır.

Tablo 4.63: Veritabanı 20 İçin Elde Edilen Çözüm.

Varlık	Nitelikler	Etiketleri
Varlık 1	1, 2	OlayTipi, Seviye
Varlık 2	5, 6, 7, 8	Olcum, Deger1, Deger2, Deger3
Varlık 3	3, 4	CihazTipi, CihazAdi



Şekil 4.61: Veritabanı 20 İçin Elde Edilen Çözümün Diyagramı.

Bu örnek için algoritma, eldeki veritabanı tasarımına çok yakın ancak farklı bir tasarım önerisi sunmuştur. Eldeki veritabanı tasarımında nitelik 5 (Olcum) tek başına bir varlık içerisinde yer alırken algoritma bu niteliği Varlık 2 içerisine eklemiştir. Bu farklılığın sebebi olarak algoritmanın tablo sayısını en aza indirmeye çalışması durumu görülebilir. Elde edilen çözüm Şekil 4.61'de diyagram olarak sunulmuştur.

5. TARTIŞMA VE SONUÇ

Bu doktora tezi çalışmasında, elektronik tablo biçiminde saklanan bir ham veri setinin, ilişkisel veritabanı yapısına dönüşebilmesi için gereken normalizasyon işleminin otomatik hale getirilebilmesi üzerine çalışılmıştır.

Literatürde otomatik normalizasyon adı altında kısıtlı sayıda çalışmaya rastlanılmıştır. Çalışmalar araç, eğitim ve yöntem olarak üç kategoriye ayrılarak Başlık 2.1.5 altında ayrıntılı olarak incelenmiştir. Araç, normalizasyon işlemini gerçekleştiren tasarımcıya yardımcı yazılımı, eğitim ise normalizasyonun öğretilmesinde ele alınan yaklaşımları ifade ettiği için bu çalışma ile aynı tabanda sayılamazlar. Yöntem kategorisi altında incelenen çalışmalar, bu tez çalışmasıyla benzer amaç taşıyarak gerçekleştirilmiştir. Çalışmalarda, normalizasyon işlemine farklı yaklaşımlarda bulunarak "en uygun" çözüm arayışı görülmektedir.

Otomatik veritabanı normalizasyonu konusunda yapılan çalışmaların ortak noktası nitelikler arasındaki bazı bağlantı ve ilişkilerin kullanıcıdan isteniyor olmasıdır. Yöntemler, ham veri seti ve kullanıcıdan gelen niteliklerin birbirleriyle ilişkisi bilgisi ışığında normalizasyon işlemini gerçekleştirebilmektedir. Bu durum iki sonuca yol açmaktadır. Birincisi, nitelikler arasındaki ilişkilerin kullanıcı tarafından hatalı tanımlanması, sürecin tamamen hatalı gerçekleşmesine sebep olabilir. Bu da elde edilen çözümün hatalı olabilmesine ve hatta elde edilen çözümün hatalı olduğunun fark edilememesine sebep olabilir. İkinci durum ise, "otomatik" adı verildiği halde işlemin kullanıcı dokunuşuna ihtiyaç duymasıdır. Otomatik olarak adlandırılan bir sürecin, tamamiyle otomatik gerçekleştirilmesi beklenmektedir. Bu durum, literatürdeki çalışmaların "Yarı-otomatik" olarak görülmesi gerektiğini göstermektedir.

Yapılan literatür taramasında normalizasyon konusunda "tam otomatik" (ham veri seti dışında hiçbir girdiye ihtiyaç duymayan) bir yöntem ile karşılaşılmamıştır. Tez çalışmasının konusu bu eksikliği giderebilecek bir yöntemin önerilmesini ve uygulamalar ile sunulmasını içermektedir.

Çalışmada veritabanı normalizasyonu, tanımından yola çıkılarak bir optimizasyon problemi olarak ele alınmıştır. Problem, 4 amacın aynı anda optimize edilmesi olarak tasarlanmıştır. Alt amaçlar şu şekildedir:

1. Hücre sayısının minimize edilmesi
2. Tekrarlı veri miktarının minimize edilmesi
3. Varlık sayısının minimize edilmesi
4. Sayısal varlık sayısının minimize edilmesi

Alt amaçlar, kapsamları ve gereklilikleri ile ilgili ayrıntıları ile birlikte Başlık 2.3.2 içerisinde incelenmiştir. Çok amaçlı olarak tasarlanan bu optimizasyon probleminin çözümünde, sezgisel bir arama yöntemi olan Genetik Algoritmalar (GA) tercih edilmiştir.

Gerçekleştirilen otomatik normalizasyon işleminde girdi dosyası olarak alınan ham veri setinin toplam nitelik sayısına bağlı olarak GA'nın kromozom yapısı oluşturulmaktadır. Kromozom, bir çözüm önerisi özelliği taşıyarak, hangi niteliğin hangi varlık içerisinde yer alması gerektiği bilgisini taşımaktadır. Her bir aday çözüm için, önerilen amaç fonksiyonu sayesinde maliyet hesaplaması gerçekleştirilmektedir. Minimum maliyet, algoritma için en iyi çözüm anlamı taşımaktadır.

Standart GA, aday çözümleri 1 ve 0'lardan oluşan karakter dizilerini kullanarak ifade etmektedir. Belirli olasılıklar dahilinde çözüm uzayının farklı bölgelerine de ulaşabilmek için herhangi bir karakter ters çevrilebilir (1 ise 0, 0 ise 1). Bu operatöre mutasyon adı verilmektedir. Çalışmada standart mutasyon operatörü geliştirilerek akıllı bir mutasyon operatörü önerilmiş ve uygulanmıştır. Standart mutasyon operatörü yalnızca bir karakteri ters çevirmekteyken geliştirilen akıllı mutasyon operatörü kromozomu gen bazında inceleyerek işlem gerçekleştirmektedir. Bu sayede çok farklı bir aday çözüm üretmek yerine bir niteliğin farklı bir varlık içerisinde ifade edilmesi gibi yeni çözümler elde edilmektedir. Operatör, aşağıdaki adımlardan birini her bir işlem 1/6 olasılıkla gerçekleştirilecek şekilde rastgele uygulamaktadır.

1. Standart mutasyon operatörü uygulaması gerçekleştirilir.
2. Rastgele bir gen seçilir ve rastgele herhangi bir gen ile değiştirilir.
3. Rastgele seçilen iki gen yer değiştirilir.
4. Rastgele seçilen bir gen, yine rastgele seçilen başka bir genin üzerine kopyalanır.

5. Kromozomdaki genlerle bir havuz oluşturulur. Kromozom, havuzdaki genlerden rastgele seçimler yapılarak yeniden oluşturulur.
6. Mutasyon operatörü uygulanmaz.

Geliştirilen akıllı mutasyon operatörü, niteliklerin varlıklar içerisindeki yerleşimlerini daha iyi ifade ettiği için bu problemin çözümünde standart mutasyon operatörüne göre daha iyi performans göstermektedir.

GA aramasında seçilecek nesil sayısı, girdi dosyası olarak seçilen ham veri setindeki nitelik sayısı ile doğru orantılıdır. Nitelik sayısı arttıkça GA'nın en iyi çözüme ulaşması için ihtiyaç duyduğu nesil sayısında artış olmaktadır. Başlık 4.1.2 içerisinde nitelik sayısına denk gelen nesil sayısı değişimleri ayrıntılı olarak sunulmuştur.

Bir ham veri setine karşılık oluşturulacak ilişkisel veritabanı yapısındaki toplam varlık sayısı en fazla ham veri setindeki nitelik sayısı kadar olabilir. Genetik aramada kromozom yapısı ham veri setindeki nitelik sayısına bağlı olarak belirlenmekte; bu sayede kromozom, aday çözümde elde edilebilecek toplam varlık sayısı kısıtını kendiliğinden yaratmaktadır. Optimizasyon hesabında kısıt olarak yalnızca bu durum mevcuttur. Çözüm uzayı ya da amaç fonksiyonu içerisinde başka bir kısıt uygulanmamıştır.

Amaç fonksiyonu içerisinde alt amaçların belirli ağırlıklarla değerlendirilebilmesi için her alt amacın ağırlığının belirlenmesi ihtiyacı doğmuştur. Uygulama için ele alınan veritabanları, alt amaç ağırlıkları eşit seçilerek önerilen sistem ile otomatik normalize edilmiştir. Elde edilen sonuçlar, veritabanlarının diyagramları ile karşılaştırılmış, diyagramdaki tasarıma ulaşılabilmesi için alt amaçların hangi ağırlık ile kullanılması gerektiği hesaplanmıştır. Buna göre alt amaçlar için ağırlıklar önerilmiştir. Ancak bu ağırlıklar eldeki veritabanlarına göre belirlendiğinden farklı çalışmalarda farklı ağırlık seçimi gerektirebilmektedir.

Algoritma, 20 farklı veritabanı ile test edilmiştir. Veritabanlarının varlık-ilişki diyagramına göre hazırlanan yapay veri, algoritma ile normalize edilmiş; elde edilen sonuç gerçek varlık-ilişki diyagramı ile karşılaştırılmıştır. Yapılan karşılaştırmada, 11 uygulama için önerilen algoritma normalizasyon işlemini ulaşılması beklenen durumla birebir aynı şekilde gerçekleştirmiştir. 6 uygulamada ise olması beklenen varlık-ilişki diyagramı ile algoritma tarafından önerilen normalize yapı arasında çok az sayıda

farklılık bulunmaktadır. 3 uygulamada yakın sayılamayacak ancak anlamsız da olmayan ilişkisel veritabanı tasarımı önerisi oluşmuştur. Önerilen algoritma uygulamaların yarısından fazlasında tam olarak beklenen sonucu üretmiş, kalan uygulamaların da çoğunluğunda veri yapısına bağlı olarak basit farklılıklar içeren yakın önerilerde bulunmuştur.

Önerilen algoritma, girdi olarak belirlenen ham veri seti için oluşturulabilecek en iyi normalizasyon işlemini gerçekleştirebilmekte ya da beklenene çok yakın sonuç üretebilmektedir. Yakın sonuçlar genelde verinin yapısı sebebiyle oluşmaktadır. Örnek olarak bireylere ait bir varlık içerisinde cinsiyet niteliğinin yalnızca 2 değer alabilmesi, algoritma tarafından varlık içerisinde veri tekrarı oluşturduğu yönünde yorumlanmaktadır. Bu şekilde az sayıda değer alabilen niteliklerin algoritma tarafından otomatik olarak tek bir niteliğe sahip yeni bir varlık şeklinde tanımlanması sayısal varlık sayısının en aza indirilmesi alt amacıyla engellenmeye çalışılmıştır. Bu sayede tüm kayıtları sayısal değer alan varlıklar başarı puanını etkilemektedir. Buna rağmen algoritma benzer özellikteki nitelikleri hatalı varlıklar içerisine yerleştirebilmektedir. Kategorik değer alan niteliklerin değerlerinin sayısal seçilmesi algoritma başarısının artmasında rol oynayabilecektir.

Normalizasyon işlemi, geleneksel olarak normal formların gerçekleştirilmesi ile sezgisel olarak gerçekleştirildiği için ve literatürde herhangi bir tam otomatik normalizasyon algoritmasına rastlanmadığı için önerilen algoritma başka herhangi bir algoritmayla karşılaştırılmamıştır. Uygulama için seçilen 20 veritabanı da uzman kişiler tarafından tasarlanmış olsa da aynı veri seti için oluşturulabilecek en iyi (normalize) diyagram oldukları tartışmalıdır. Bu sebeple bu algoritmanın tamamiyle nesnel olarak değerlendirilmesi mümkün olamamıştır. Ele alınan veritabanı sayısının fazla tutulması ile tutarlılık arttırılmaya çalışılmıştır. Ayrıca eldeki sonuçlar veritabanı tasarımcısı olarak ve normal formlar gözetilerek öznel olarak değerlendirildiğinde sonuçların kabul edilebilir olduğu düşünülmüştür.

Algoritma, normalizasyon işlemi sonucunda ham veri seti için hangi sayıda varlık tanımlanması gerektiğini ve hangi niteliğin hangi varlık içerisinde yer alması gerektiğini bir diğer deyişle fonksiyonel bağımlılıkları belirleyebilmektedir ancak hangi varlıklar arasında ilişki bulunduğunu ve ilişkinin türünü belirleyememektedir. Bu sebeple

kullanıma hazır bir ilişkisel veritabanı elde etmek mümkün değildir. Yine de tasarımcı için başarılı bir karar destek sistemi niteliğindedir.

Çalışma, uluslararası literatürdeki otomatik normalizasyon çalışmalarına göre iyileştirme göstermekte; sezgisel ve deneyime dayalı gerçekleştirilen normalizasyon işleminin bir optimizasyon problemi olarak ele alınması açısından da özgün olma niteliği taşımaktadır. Önerilen algoritmanın geliştirilmesi ve çalışmanın ilerletilebilmesi için yapılabilecek çalışmalardan bazıları aşağıda sunulmuştur.

- Normalizasyon işleminin başarısını arttırabilmek için etkisi bulunan yeni alt amaçların belirlenmesi ve entegre edilmesi
- Alt amaç katsayılarının daha hassas bir şekilde belirlenmesi
- Farklı optimizasyon algoritmaları ile problemin daha hızlı çözülebilmesi
- Otomatik oluşturulan varlıklar arasındaki ilişkilerin ve ilişki tiplerinin belirlenebilmesi

KAYNAKLAR

Ahmad, R., Saknakosnak, P. ve Hooi, Y., 2014, Excel-Database Converting System Using Data Normalization Technique, *Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013),* Springer, 23-30.

Ahmedi, L., Jakupi, N. ve Jajaga, E., 2012, NORMALDB--A Logic-Based Interactive e-Learning Tool for Database Normalization and Denormalization, *eLmL 2012: The Fourth International Conference on Mobile, Hybrid, and On-line Learning,* 44-50.

Akehurst, D., Bordbar, B., Rodgers, P. ve Dalgliesh, N., 2002, Automatic normalisation via metamodelling, *Declarative Meta Programming to Support Software Development Workshop Proceeding,* 45-48.

Arenas, D., Chevrier, R., Hanafi, S. ve Rodriguez, J., 2015, Solving the train timetabling problem, a mathematical model and a genetic algorithm solution approach, *6th International Conference on Railway Operations Modelling and Analysis (RailTokyo2015),* Tokyo, 1-12.

Bachman, C., 1969, *Data structure diagrams* (Cilt 1), New York, NY, USA, ACM, 4-10.

Bahmani, A., Naghibzadeh, M. ve Bahmani, B., 2008, Automatic database normalization and primary key generation, *Electrical and Computer Engineering CCECE, Canadian Conference on,* IEEE, 11-16.

Bahmani, A., Shekofteh, S., Naghibzadeh, M. ve Deldari, H., 2010, Parallel algorithms for automatic database normalization, *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on,* Cilt 2, IEEE, 157-161.

Bekiroğlu, S., Dede, T. ve Ayvaz, Y., 2009, Imlementation of different encoding types on structural optimization based adaptive genetic algorithm, *Finite Elements in Analysis and Design, 45*, 826-835.

Bernstein, P., 1976, Synthesizing third normal form relations from functional dependencies, *ACM Transactions on Database Systems,* 1(4), 277-298.

Chen, P., 1976, The entity-relationship model—toward a unified view of data, *ACM Transactions on Database Systems (TODS),* 1(1), 9-36.

Chootinan, P. ve Chen, A., 2006, Constraint handling in genetic algorithms using a gradient-based repair method. *Computers & operations research,* 33(8), 2263-2281.

Codd, E. F., 1969, Redundancy and consistency of relations stored in large data banks. *SIGMOD Rec.*, 17-36.

Codd, E. F., 1970, A relational model of data for large shared data banks. *Communications of the ACM. 13.6*, 377-387.

Codd, E. F., 1981, Data models in database management. *ACM Sigmod Record 11.2*, 112-114.

Codd, E. F., 1982, Relational database: a practical foundation for productivity. *Communications of the ACM 25.2*, 109-117.

Coello, C. ve Montes, E., 2002, Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics,* 16(3), 193-203.

Crepinsek, M., Liu, S.-H. ve Mernik, M., 2013, Exploration and exploitation in evolutionary algorithms: a survey. *ACM Computing Surveys,* 45(3), 35:1-35:33.

Çağıltay, N. ve Tokdemir, G., 2010, *Veritabanı Sistemleri Dersi.* Ada Matbaacılık.

Deb, K., 2000, An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 311-338.

Deb, K., 2011, Mult-objective optimization using evolutionary algorithms: an introduction. *Multi-Objective Evolutionary Optimisation for Product Design and Manufacturing*, 3-35, London, UK, Springer.

Deb, K., Pratap, A., Agarwal, S. ve Meyarivan, T., 2002, A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on,* 6(2), 182-197.

Demba, M., 2013, An Algorithmic Approach to Database Normalization. *International Journal of Digital Information and Wireless Communications (IJDIWC),* 3(2), 57-65.

Dongare, Y., Dhabe, P. ve Deshbukh, S., 2011, RDBNorma:-A semi-automated tool for relational database schema normalization up to third normal form. *International Journal of Database Management Systems (IJDMS),* 3(1), 133-154.

Du, H. ve Wery, L., 1999, Micro: A normalization tool for relational database designers. *Journal of Network and Computer applications,* 22(4), 215-232.

Duggal, K., Srivastav, A. ve Kaur, S., 2014, Gamified Approach to Database Normalization. *International Journal of Computer Applications,* 93(4), 47-53.

Dunkel, B. ve Soparkar, N., 1999, Data organization and access for efficient data mining. *Data Engineering, 1999. Proceedings., 15th International Conference on,* 522-529.

Elmasri, R. ve Navathe, S., 2003, *Fundamentals of database systems 4th edition.* Boston: Pearson Education, Inc.

Erickson, M., Mayer, A. ve Horn, J., 2002, Multi-objective optimal design of groundwater remediation systems: application of the niched Pareto genetic algorithm (NPGA). *Advances in Water Resources,* 25(1), 51-65.

Fagin, R., 1977, Multivalued dependencies and a new normal form for relational databases. *ACM Transactions on Database Systems,* 2(3), 262-278.

Fagin, R., 1979, Normal forms and relational database operators. *ACM-SIGMOD*, 153-160.

Fagin, R., 1981, A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems*, 6(3), 387-415.

Fanguy, R. ve Kleen, B., 2005, Normalization Shootout: a competitive game that impacts student learning. *Issues in Information Systems,* 6(1), 21-27.

Fayyad, U., Piatetsky-Shapiro, G. ve Smyth, P., 1996, From data mining to knowledge discovery in databases. *AI Magazine,* 17(3), 37-54.

Fonseca, C. ve Fleming, P., 1993, Genetic Algorithms for Multiobjective Optimization: Formulation Discussion and Generalization. *ICGA,* Citeseer, 416-423.

Ghosh, A. ve Dehuri, S., 2004, Evolutionary algorithms for multi-criterion optimization: a survey. *International Journal of Computing & Information Sciences,* 2(1), 38-57.

Gogolla, M., 1991, Towards a semantic view of an extended entity-relationship model. *ACM Transactions on Database Systems,* 16(3), 369-416.

Goldberg, D., 1989, *Genetic algorithms in search, optimization, and machine learning.* Addion Wesley.

Grothendieck, G., 2014, sqldf: Perform SQL selects on r data frames. https://CRAN.R-project.org/package=sqldf adresinden alındı.

Hajela P. ve Lin C. Y., 1992, Genetic search strategies in multicriterion optimal design. *Structural optimization 4.2,* 99-107.

Hans, A., 1988, Multicriteria optimization for highly accurate systems. *Multicriteria Optimization in Engineering and Sciences,* 19, 309-352.

Haupt, R. L. ve Haupt, S. E., 2004, Practical genetic algorithms. John Willey & Sons.

Henry, W., 1969, Hierarchical structure for data management . *IBM Systems Journal 8.1*, 2-15.

Herradi, N., Hamdi, F., Metais, E., Ghorbel, F. ve Soukane, A., 2015, *PersonLink: An ontology representing family relationships for the CAPTAIN MEMO memory prosthesis,* Springer.

Herrera, F., Lozano, M. ve Verdegay, J., 1998, Tackling real-coded genetic algorithms: operators and tools for behavioural analysis. *Artificial Intelligence Review,* 12, 265-319.

Hoffer, J., Ramesh, V. ve Topi, H., 2016, *Modern database management.* Pearson Education Limited.

Holland, J., 1975, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* Michigan Press.

Horn, J., Nafpliotis, N. ve Goldberg, D., 1994, A niched Pareto genetic algorithm for multiobjective optimization. *Evolutionary Computation, 1994, IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on,* IEEE, 82-87.

Jones, G., Willett, P., Glen, R., Leach, A. ve Taylor, R., 1997, Development and validation of a genetic algorithm for flexible docking. *Journal of molecular biology*, 727-748.

Kent, W., 1981, A simple guide to five normal forms in relational database theory. *Communications of the ACM,* 26(2), 120-125.

Khan, A., 2015, *Scenario based software testing using UML activity diagram.*

Kollar, L. ve Sterbinszky, N., 2014, *Case study in system development - notes.*

Kowalczyk, R., 1997, Constraint consistent genetic algorithms. *IEEE*, 343-348.

Köseoğlu, K., 2007, Veritabanı mantığı. Pusula Yayıncılık.

Le Riche, R., Knopf-Lenoir, C. ve Haftka, R., 1995, A Segregated Genetic Algorithm for Constrained Structural Optimization. *ICGA,* Citeseer, 558-565.

Lee, H., 1994, Justifying database normalization: a cost/benefit model. *Information Processing & Management,* 31(1), 59-67.

Li, J., Wang, Y., Yang, S. ve Cai, Z., 2016, *A comparative study of constraint-handling techniques in constrained multiobjective evolutionary optimization,* IEEE Press.

Marler, R. T. ve Arora, J. S., 2004, Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization 26.6,* 369-395.

Martinez, T. ve Fages, F., 2015, On translating MiniZinc constraint models into fitness function for evolutionary algorithms: application to continuous placement

problems. *Proceedings of the Sixth Workshop on Bin Packing and Placement Constraints, BPPC*.

Menzel, C. ve Mayer, R., 1998, The IDEF family of languages, *Handbook on Architectures of Information Systems,* Berlin Heidelberg: Springer, 209-241.

Michalewicz, Z. ve Janikow, C., 1991, Handling constraints in genetic algorithms, *ICGA*, 151-157.

Mitchell, M., 1995, Genetic algorithms: an overview, *Complexity,* 1(1), 31-39.

Mitchelle, M., 1998, *An introduction to genetic algorithms* MIT Press: London.

Mitrovic, A., 2002, NORMIT: A web-enabled tutor for database normalization. *Computers in Education, 2002. Proceedings. International Conference on,* IEEE, 1276-1280.

Nizam, A., 2011, *Veritabanı Tasarımı: İlişkisel Veri Modeli ve Uygulamalar,* Papatya Yayıncılık Eğitim.

Ortuno, D., 2014, *Recruiting and assessment functions in samper head hunting.*

Orvosh, D. ve Davis, L., 1994, Using a genetic algorithm to optimize problems with feasibility constraints, *Evolutionary Computation, 1994, IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on,* IEEE, 548-553.

Purchase, H., Welland, R., McGill, M. ve Colpoys, L., 2004, Comprehension of diagram syntax: an empirical study of Entity Relationship notations, *International Journal of Human-Computer Studies, 61*(2), 187-203.

R Core Team., 2016, R: a language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/ adresinden alındı.

Raghu, R. ve Johannes, G., 2000, *Database management systems.* Singapur: McGraw Hill.

Ramakrishnan, R. ve Gehrke, J., 2007, *Database management systems.*

Read, R., Fussell, D. ve Silberschatz, A., 1992, *A multi-resolution relational data model.* Austin, Texas: Department of Computer Science, University of Texas at Austin.

Revolution Analytics ve Wetson, S., 2015, doParallel: Foreach parallel adaptor for the parallel package, https://CRAN.R-project.org/package=doParallel adresinden alındı.

Riordan, R., 2005, *Designing effective database systems.* Addison-Wesley.

Robinson, K., 2014, *The regulated market and cost-benefit analysis,* 01 11, 2017 tarihinde Exploring Environmental Ethics and Policy: https://krobinsonphil4302.wordpress.com/2014/10/21/the-regulated-market-and-cost-benefit-analysis/ adresinden alındı.

Rowley, J., 2007, The wisdom hierarchy: representations of the DIKW hierarchy, *Journal of Information Science,* 33(2), doi:10.1177/0165551506070706, 163-180.

RStudio Team, 2015, RStudio: Integrated development for R. *RStudio, Inc.* http://www.rstudio.com/ adresinden alındı.

Satish, P., Sheeba, K. ve Rangarajan, K., 2013, Deriving combinatorial test design model from UML activity diagram, *IEEE Sixth International Conference on Software Testing*, 331-337.

Satman, M. H., 2008, Ekonometrik Yöntem ve Sorunlara Genetik Algoritma Yaklaşımları ve İktisadi Uygulamalar, *Doktora Tezi,* İstanbul Üniversitesi Sosyal Bilimler Enstitüsü.

Satman, M. H., 2013, Machine Coded Genetic Algorithms For Real Parameter Optimization Problems, *Gazi University Journal of Science, 26*(1), 85-95.

Satman, M. H., 2016, Genetik Algoritmalar. Türkmen Kitabevi.

Schaffer, J., 1985, Multiple objective optimization with vector evaluated genetic algorithms, *Proceedings of the 1st International Conference on Genetic Algorithms*, Pittsburgh, PA, USA, 93-100.

Scrucca, L., 2013, GA: a package for genetic algorithms in R. *Journal of Statistical Software, 53*(4), 1-37. http://www.jstatsoft.org/v53/i04/ adresinden alındı.

Scrucca, L., 2016, On some extensions to GA package: hybrid optimisation, parallelisation and island evolution. *R Journal.*

Simon, D., Ergezer, M. ve Du, D., 2009, Population distributions in bibgeography-based optimization algorithms with elitism. *Systems, Man and Cybemetics*, IEEE International Conference (SCM).

Soler, J., Boada, I., Prados, F. ve Poch, J., 2006, A web-based Problem-Solving Environment for database Normalization, *Simposio Internacional de Informatica Educativa. SIIE.*

Srinivas, M. ve Patnaik, L., 1994, Genetic algorithms: a survey, *Evolutionary Computation, 27*(6), 17-26.

Srinivas, N. ve Deb, K., 1994, Muiltiobjective optimization using nondominated sorting in genetic algorithms, *Evolutionary computation,* 2(3), 221-248.

Sumathi, S. ve Esakkirajan, S., 2007, *Fundamentals of relational database management systems,* Springer.

Sunitha, G. ve Jaya, A., 2013, A knowledge based approach for automatic database normalization, *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 2*(5), 1816.

Taboada, H. ve Coit, D., 2008, Multi-objective scheduling problems: determination of pruned Pareto sets, *IIE Transactions,* 40(5), 552-564.

Teorey, T., Lightstone, S., Nadeau, T. ve Jagadish, H., 2011, *Database modelling and design - Logical design,* Elsevier.

Teorey, T., Yang, D. ve Fry, J., 1986, A logical design methodology for relational databases using the extended entity-relationship model, *ACM Computing Surveys (CSUR) 18.2*, 197-222.

Vardi, M., 1987, Fundamentals of dependency theory, *Trends in Theoretical Computer Science,* Rockville, MD: Computer Science Press, 171-224.

Verma, S., 2012, Comparing manual and automatic normalization techniques for relational database, *IJREAS*.

Vipin, K., 1992, Algorithms for constraint-satisfaction problems: A survey, *AI magazine,* 13(1), 32.

Wolfram Research, Inc., 2016, *Mathematica.* Champaign, Illinois: Wolfram Research, Inc.

Wright, A., 1991, Genetic algorithms for real parameter optimization, *Foundations of Genetic Algorithms*, 205-218.

Yadav, P. ve Goyal, A., 2013, *Concepts of Database System,* S. K. Kataria and Sons.

Yazıcı, A. ve Karakaya, Z., 2007, JMathNorm: A database normalization tool using mathematica, *Computational Science - ICCS 2007,* Springer, 186-193.

Yuret, D. ve Maza, M., 1993, Dynamic hill climbing: overcoming the limitations of optimization techniques, *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*, 208-212.

Zhu, X.-h., Zeng, Q.-l. ve Cao, Q.-h., 2010, A Complex XML Schema to Map the XML Documents of Distance Education Technical Specifications into Relational Database. *JDCTA,* 4(8), 182-192.

Zitzler, E. ve Thiele, L., 1998, Multiobjective optimization using evolutionary algorithms—a comparative case study. *Parallel problem solving from nature—PPSN V*, Springer, 292-301.

Zitzler, E. ve Thiele, L., 1999, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, *evolutionary computation, IEEE transactions on,* 3(4), 257-271.

Zitzler, E., Laumanns, M. ve Thiele, L., 2001, SPEA2: Improving the strength Pareto evolutionary algorithm, Eidgenossische Technische Hochschule Zurich (ETH), Institut fur Technische Informatik und Kommunikationsnetze (TIK).

EKLER

EK-A: ÇALIŞTIRILABİLİR KODLAR

autonorm Fonksiyonu

```
autonorm <- function(anRawdata,
                     quick = FALSE,
                     popsize = 100,
                     maxiter = 1000,
                     caching = FALSE,
                     w = c(0.33960899, 0.02030116, 0.60869628, 0.03139356),
                     monitor = if (interactive()) anMonitor else FALSE,
                     mutation = anMutation,
                     pmutation = 0.1) {


  if (quick == TRUE) {
    anRawdata <- anSampling(anRawdata)
  }


  if (caching == TRUE) {
    anCacheCreate()
  }


  hvs <<- anHVS(anRawdata)
  if (hvs < 2) {
```

```
  stop ("FATAL ERROR: You need least 2 attributes.")
}

m <- anM(hvs)

tab <- anMaxTable(m)

chsize <- anChsize(hvs, m)

cl <- makeCluster(detectCores(), type='PSOCK')
registerDoParallel(cl)

myresult <- ga(
    type="binary",
    nBits=chsize,
    fitness=anFitnessFunction,
    anRawdata=anRawdata,
    caching=caching,
    hvs = hvs,
    m = m,
    tab = tab,
    w = w,
    maxiter=maxiter,
    popSize=popsize,
    pmutation = pmutation,
```

```
      elitism = 2,
      mutation = anMutation,
      monitor= anMonitor,
      parallel = detectCores()
      )

  stopCluster(cl)

  return(myresult)
}
```

anMutation Fonksiyonu

```
anMutation <- function (object, parent, ...)  {
  mutate <- parent <- as.vector(object@population[parent, ])
  mymnumber <- anM(hvs)
  mycoin <- round(runif(1,1,5))
  if (mycoin == 1) {
    n <- length(parent)
    j <- sample(1:n, size = 1)
    mutate[j] <- abs(mutate[j] - 1)
  } else if (mycoin == 2) {
    chmatrix <- matrix(mutate,ncol = mymnumber,byrow = TRUE)
    chmatrix[round(runif(1,1,nrow(chmatrix))),] <- round(runif(mymnumber,0,1))
    mutate <- as.vector(t(chmatrix))
  } else if (mycoin == 3) {
    chmatrix <- matrix(mutate,ncol = mymnumber,byrow = TRUE)
```

```
    chmatrix2 <- chmatrix
    gene1 <- round(runif(1,1,nrow(chmatrix)))
    gene2 <- round(runif(1,1,nrow(chmatrix)))
    chmatrix[gene1,] <- chmatrix2[gene2,]
    chmatrix[gene2,] <- chmatrix2[gene1,]
    mutate <- as.vector(t(chmatrix))
  } else if (mycoin == 4) {
    chmatrix <- matrix(mutate,ncol = mymnumber,byrow = TRUE)
    chmatrix[round(runif(1,1,nrow(chmatrix))),] <-
chmatrix[round(runif(1,1,nrow(chmatrix))),]
    mutate <- as.vector(t(chmatrix))
  } else if (mycoin == 5) {
    chmatrix <- matrix(mutate,ncol = mymnumber,byrow = TRUE)
    genepool <- unique(chmatrix)
    for (i in 1:nrow(chmatrix)) {
      chmatrix[i,] <- genepool[round(runif(1,1,nrow(genepool))),]
    }
    mutate <- as.vector(t(chmatrix))
  }
  return(mutate)
}
```

anMonitor Fonksiyonu

```
anMonitor <- function (object, digits = getOption("digits"), ...) {
  fitness <- na.exclude(object@fitness)
  sumryStat <- c(mean(fitness), max(fitness))
  sumryStat <- format(sumryStat, digits = digits)
```

```
  replicate(2, clearConsoleLine())

  cat(paste("\rAUTONORM | autonorm.org | Completed: ", round((object@iter /
object@maxiter)*100) ,"% | Best Fitness Value =", sumryStat[2],sep = ""))

  flush.console()

}
```

anFitnessFunction Fonksiyonu

```
anFitnessFunction <- function (ch, anRawdata, caching, hvs, m, tab, w) {

  if (caching == TRUE) {

    controlValue <- anCacheControl(ch)

    if (controlValue != FALSE) {

      return (controlValue)

    }

  }


  v <- anAttributeVector(ch, hvs, m)


  tables <- anGetTables(tab,v)


  anCreateTables(tab,v,anRawdata)


  obj1 <- w[1] * anObjective1(tables, anRawdata)


  obj2 <- w[2] * anObjective2(v)


  obj3 <- w[3] * anObjective3(v, tab)
```

```
  obj4 <- w[4] * anObjective4(tables, v, anRawdata)


  fitnessvalue <- sum(  obj1 +
                        obj2 +
                        obj3 +
                        obj4,
                        na.rm = TRUE  )


  if (caching == TRUE) {
    anCaching(ch, fitnessvalue, obj1, obj2, obj3, obj4)
  }


  return (-fitnessvalue)
}
```

anSampling Fonksiyonu

```
anSampling <- function (anrawdata) {
  totalrow <- NROW(anrawdata)
  if (totalrow > 250) {
    startpoints <- round(runif(10,1,totalrow))


    myblocks <- c(
      startpoints[1]:(startpoints[1]+9),
      startpoints[2]:(startpoints[2]+9),
      startpoints[3]:(startpoints[3]+9),
      startpoints[4]:(startpoints[4]+9),
```

```
      startpoints[5]:(startpoints[5]+9),
      startpoints[6]:(startpoints[6]+9),
      startpoints[7]:(startpoints[7]+9),
      startpoints[8]:(startpoints[8]+9),
      startpoints[9]:(startpoints[9]+9),
      startpoints[10]:(startpoints[10]+9)
    )
    anrawdata <- anrawdata[myblocks,]
  } else {
    print("Quick mode cannot run. It must be bigger than 250 rows")
  }
  return(anrawdata)
}
```

anCacheCreate Fonksiyonu

```
anCacheCreate <- function () {
  anCacheFrame <<- matrix (rep(NA, 6),ncol = 6)
}
```

anCacheDrop Fonksiyonu

```
anCacheDrop <- function () {
  remove(anCacheFrame)
}
```

anCaching Fonksiyonu

```
anCaching <- function(ch, fitnessvalue, obj1, obj2, obj3, obj4) {
  anCacheFrame <<- rbind(anCacheFrame, c(binary2decimal(ch), fitnessvalue,
obj1, obj2, obj3, obj4))
}
```

anCacheControl Fonksiyonu

```
anCacheControl <- function (ch) {

  cachecontrol <- as.data.frame(anCacheFrame)

  cacheresult <- sqldf(paste("select * from cachecontrol where V1 =
'",binary2decimal(ch),"' ",sep=""))

  if (!is.na(cacheresult[1,2])) {

    return (-(cacheresult[1,2]))

  } else {

    return (FALSE)

  }

}
```

anHVS Fonksiyonu

```
anHVS <- function (anDataset) {

  return (ncol(anDataset))

}
```

anM Fonksiyonu

```
anM <- function (hvs) {

  m <- ceiling(log2(hvs))

  if (m == 1) {

    m <- 2

  }

  return (m)

}
```

anChsize Fonksiyonu

```
anChsize <- function (hvs, m) {

  return (m * hvs)
```

```
}
```

anMaxTable Fonksiyonu

```
anMaxTable <- function (m) {

  tab <- (2^(m))

  return (tab)

}
```

anAttributeVector Fonksiyonu

```
anAttributeVector <- function (ch, hvs, m) {

  index <- 1

  v <- c()

  for (i in 1:hvs) {

    tableid <- binary2decimal(ch[index:(index+m-1)]) + 1

    v <- append(v,tableid)

    index <- index + m

  }

  return (v)

}
```

anGetTables Fonksiyonu

```
anGetTables <- function(tab,v) {

  tables <- c()

  for (i in 1:tab) {

    if (length(which (v == i)) != 0) {

      tables <- append(i,tables)

    }

  }

  return (tables)
```

```
}
```

anCreateTables Fonksiyonu

```
anCreateTables <- function (tab, v, anRawdata) {
  for (i in 1:tab) {
    if (length(which (v == i)) != 0) {
      temptable <- unique(anRawdata[(which (v == i))])
      assign(paste("table",i,sep=""), temptable,envir = globalenv())
    }
  }
}
```

anObjective1 Fonksiyonu

```
anObjective1 <- function(tables, anRawdata) {
  totalcell <- 0
  for (i in tables) {
    temptable <- get(paste("table",i,sep=""))
    totalcell <- totalcell + ((ncol(temptable)) * nrow(temptable))
  }
  return (totalcell / (nrow(anRawdata)*ncol(anRawdata)))
}
```

anObjective2 Fonksiyonu

```
anObjective2 <- function(v) {
  dupprates <- c()
  for (i in 1:length(v)) {
    mytable <- get(paste("table",v[i],sep=""))
    thecol <- mytable[,paste("V",i,sep="")]
    allrows <- length(thecol)
```

```
    distinctrows <- length(unique(thecol))
    myvalue <- ((allrows - distinctrows) / allrows)
    dupprates <- append(dupprates,myvalue)
  }
  return (sum(dupprates,na.rm = TRUE))
}
```

anObjective3 Fonksiyonu

```
anObjective3 <- function(v, tab) {
  return (length(unique(v)) / tab)
}
```

anObjective4 Fonksiyonu

```
anObjective4 <- function(tables,v, anRawdata) {
  numericaltable <- 0
  for (i in tables) {
    if (is.numeric(as.vector(as.matrix(anRawdata[which(v == i)])))) {
      numericaltable <- numericaltable + 1
    }
  }
  return((numericaltable / length(unique(v))) ^ (1/length(unique(v))))
}
```

anStructure Fonksiyonu

```
anStructure <- function(ch, anRawdata) {
  hvs <- anHVS(anRawdata)

  m <- anM(hvs)
```

```
  tab <- anMaxTable(m)

  chsize <- anChsize(hvs,m)

  v <- anAttributeVector(ch,hvs,m)

  tables <- anGetTables(tab,v)

  for (i in tables) {
    print(which (v == i))
  }
}
```

anFitnessList Fonksiyonu

```
anFitnessList <- function(ch, anRawdata, w = c(0.33960899, 0.02030116,
0.60869628, 0.03139356)) {

  hvs <- anHVS(anRawdata)

  m <- anM(hvs)

  tab <- anMaxTable(m)

  chsize <- anChsize(hvs,m)

  v <- anAttributeVector(ch, hvs, m)
```

```
  tables <- anGetTables(tab,v)

  anCreateTables(tab,v,anRawdata)

  obj1 <- w[1] * anObjective1(tables, anRawdata)

  obj2 <- w[2] * anObjective2(v)

  obj3 <- w[3] * anObjective3(v, tab)

  obj4 <- w[4] * anObjective4(tables, v, anRawdata)

  return( c(obj1, obj2, obj3, obj4,
            sum(  obj1 +
                  obj2 +
                  obj3 +
                  obj4,
                  na.rm = TRUE)  ))
}
```

ÖZGEÇMİŞ

Kişisel Bilgiler	
Adı Soyadı	Emre Akadal
Doğum Yeri	Fatih
Doğum Tarihi	01.01.1988
Uyruğu	☑ T.C. ❒ Diğer:
Telefon	0532 161 87 54
E-Posta Adresi	emreakadal@gmail.com
Web Adresi	www.emreakadal.com

Eğitim Bilgileri	
Lisans	
Üniversite	İstanbul Üniversitesi
Fakülte	Fen Fakültesi
Bölümü	Fizik
Mezuniyet Yılı	2010

Yüksek Lisans	
Üniversite	İstanbul Üniversitesi
Enstitü Adı	Fen Bilimleri Enstitüsü
Anabilim Dalı	Enformatik Anabilim Dalı
Programı	Enformatik Programı
Mezuniyet Tarihi	2013

Doktora	
Üniversite	İstanbul Üniversitesi
Enstitü Adı	Fen Bilimleri Enstitüsü
Anabilim Dalı	Enformatik Anabilim Dalı
Programı	Enformatik Programı
Mezuniyet Tarihi	2017

Makale ve Bildiriler
Satman M.H., Akadal E., "mcga: R implementation of the machine-coded genetic algorithms", British Journal of Mathematics & Computer Science, 20 (2), 2017.
Satman M.H., Akadal E., "ARIMA forecasting as a genetic inheritance operator in floating-point genetic algorithms", Journal of Mathematical and Computational Science , no.3, pp.360-376, 2016.
Selçukcan Erol Ç., Özdemir Ş., Özen Z., Akadal E., Ayvaz Reis Z., "Fibonacci Spiral In Sunflower With Geogebra ", JOURNAL OF EDUCATIONAL AND INSTRUCTIONAL STUDIES IN THE WORLD, vol.25, pp.190-201, 2012.
Satman M.H., Akadal E., "Reel sayılı genetik algoritmalarda makine kodlamalı genetik operatörlerin performanslarının diğer bazı genetik operatörlerle

karşılaştırılması", 17. ULUSLARARASI EKONOMETRİ, İSTATİSTİK VE YÖNEYLEM ARAŞTIRMASI SEMPOZYUMU, SİVAS, TÜRKIYE, 2-4 Haziran 2016, ss.1-1.

Çelik S., Akadal E., "Mobil Ticaret Uygulamaları Arayüzleri Üzerine Bir Araştırma", XIX. Türkiye'de İnternet Konferansı, İZMİR, TÜRKIYE, 27-29 Kasım 2014, ss.66-66.

Gülseçen S., Özdemir Ş., Gezer M., Akadal E., Özen Z., "The Good Reader Of Digital World, Digital Natives: Are They Good Writer Of That World?", 6th World Conference on Educational Sciences, VALLETTA, MALTA, 6-9 Şubat 2014.

Akadal E., Dönmez B., "Observation Of The Young Age Individual'S Approach To A New Computer Fact", 5th International Future-Learning Conference on Innovations in Learning for the Future 2014: e-Learning, İSTANBUL, TÜRKİYE, 5-7 Mayıs 2014.

Özdemir Ş., Akadal E., Çelik S., Ayvaz Reis Z., "Uygulama Marketlerinin Eğitim Kategorisi Altındaki Uygulamalarının İncelenmesi", Akademik Bilişim 2013, ANTALYA, TÜRKİYE, 23-25 Ocak 2013.

Akadal E., Özdemir Ş., Ayvaz Reis Z., "Gnu Özgür Belgeleme Lisansı (Gfdl) Kapsamındaki Dokümanlar İçin Bir Çevrimiçi Arşiv Geliştirilmesi", Akademik Bilişim 2013, ANTALYA, TÜRKİYE, 23-25 Ocak 2013, cilt.1, no.1.

Selçukcan Erol Ç., Akadal E., Olgun A., Ayvaz Reis Z., "New Address Of On-Campus Communication: A Pilot Study From Istanbul University", World Conference on Information Technology, TÜRKİYE, Kasım 2011, pp.1347-1351.

Muhammad B., Akadal E., Alsadi M., Alshaikh M., Iqbal M., Sabimbona S., Gülseçen S., "An Overview On Ict Development In Turkey, Pakistan, Middle East And Africa", International Development Informatics Association, TÜRKİYE, Eylül 2012, vol.1, no.1.

Akadal E., Özdemir Ş., Ayvaz Reis Z., "The Analyze Of Education Applications Category In Online Mobile Application Markets", Future-Learning 2012, TÜRKİYE, 14-16 Kasım 2012, pp.545-562.

Akadal E., Saylan S., "Gerçek Dünya Haritası Üzerinde Genetik Algoritmalarla Gezgin Satıcı Problemi Uygulaması", R ile Veri Madenciliği Uygulamaları, Balaban M. E., Kartal E., Ed., Çağlayan Kitabevi, İstanbul, ss.165-186, 2016.