AĞAÇ VE ÇİZGE VERİTABANLARINDA HASSAS BİLGİ GİZLEME

HARUN GÖKÇE

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

KASIM 2010

ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Ünver KAYNAK
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Doç. Dr. Erdoğan DOĞDU
Anabilim Dalı Başkanı

Harun GÖKÇE tarafından hazırlanan AĞAÇ VE ÇİZGE VERİTABANLARINDA HASSAS BİLGİ GİZLEME adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Yrd. Doç. Dr. Osman ABUL
Tez Danışmanı

Tez Jüri Üyeleri

Başkan	:	Prof. Dr. İsmail Hakkı TOROSLU	_____
Üye	:	Doç. Dr. Erdoğan DOĞDU	_____
Üye	:	Yrd. Doç. Dr. Osman ABUL	_____

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Harun GÖKÇE

Üniversitesi : **TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü : **Fen Bilimleri**
Anabilim Dalı : **Bilgisayar Mühendisliği**
Tez Danışmanı : **Yrd. Doç. Dr. Osman ABUL**
Tez Türü ve Tarihi : **Yüksek Lisans - Kasım 2010**

Harun GÖKÇE

AĞAÇ VE ÇİZGE VERİTABANLARINDA HASSAS BİLGİ GİZLEME

ÖZET

Veritabanı yayınlama kuruluşların bazen ihtiyaç duyduğu yararlı bir işlemdir. Fakat bu her ne kadar iyi bir işlem olsa da, hassas bilgileri açığa çıkarmak suretiyle tehdit edici olabilmektedir. Bugünlerde, birçok ileri seviye veri madenciliği uygulaması geliştirildiğinden, bu veri madenciliği uygulamalarının yayınlanan veritabanı üzerinde uygulanmasıyla, veritabanında saklı olan hassas bilgiler açığa çıkabilir. Dolayısıyla olduğu gibi veritabanı yayınlamak güvenli bir veritabanı yayınlama değildir. Bu yüzden veritabanındaki hassas bilgiler ilk önce tanımlanmalı ve sonra da elenmelidir. Bu işlem sterilize etme işlemi olarak adlandırılır. Hassas bilgi gizleme daha çok hareket tipi veritabanları bağlamında oldukça çalışılmıştır. Fakat aynı zamanda hassas bilgi gizlemenin ağaç ve çizge tipi yapısal veritabanları için de çalışılması gerekmektedir. Bu tezde, hassas bilgi gizleme ağaç ve çizge tipindeki veritabanlarını da içerek şekilde genişletilmiştir. Bu çalışma her iki veritabanında hassas bilgi gizleme problemini tanımlamakta ve çözümler geliştirmektedir. Bunun yanı sıra FISHER adında, işlemler, dizgiler ve zaman-mekân izleri gibi diğer veritabanlarında da hassas bilgi gizleme yapabilecek bütüncül bir uygulama geliştirilmiştir.

Anahtar Kelimeler: Bilgi gizleme, Ağaç veritabanları, Çizge veritabanları, Veri mahremiyeti, Veri Madenciliği

University	**:**	**TOBB University of Economics and Technology**
Institute	**:**	**Institute of Natural and Applied Sciences**
Science Programme	**:**	**Computer Engineering**
Supervisor	**:**	**Assistant Professor Dr. Osman ABUL**
Degree Awarded and Date	**:**	**M.Sc. - November 2010**

Harun GÖKÇE

SENSITIVE KNOWLEDGE HIDING IN TREE AND GRAPH DATABASES

ABSTRACT

Database sharing is a beneficial process which organizations sometimes need to do. Although it is a good practice, it may threaten the database security through disclosing sensitive knowledge. This is because sophisticated data mining tools nowadays are so developed that running any of the tools on published database may disclose the sensitive knowledge implied by the database. As a result, *as is* database publishing is not a secure way of database sharing. Hence, we reason that the sensitive knowledge in database must be firstly identified then it must be removed. The process is called the sanitization. Sensitive knowledge hiding is extensively studied mostly in the context of transactions. However, it needs to be studied for tree and graph structured databases as well. In this thesis, the sensitive knowledge hiding is extended for tree and graph databases. This work defines respective problems and develops solutions for both of them. Moreover, a framework, called FISHER, is developed for sensitive knowledge hiding which is able to hide sensitive knowledge from various kinds of other databases as well, including transactions, sequences, and spatio-temporal databases.

Keywords: Knowledge hiding, Tree databases, Graph databases, Data privacy, Data mining

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren hocam Yrd. Doç. Dr. Osman ABUL'a, kıymetli tecrübelerinden faydalandığım TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine ve 108E016 numaralı proje kapsamında destek aldığım TÜBİTAK'a teşekkürü bir borç bilirim.

İÇİNDEKİLER

ÇİZELGELERİN LİSTESİ

ŞEKİLLERİN LİSTESİ

1 GİRİŞ

Farklı kuruluşlar, kendi iş alanları ile ilgili zamanla edindikleri verileri kendi veritabanlarında toplarlar. Toplanan bu veriler bilimsel ve sosyal açıdan yararlı bilgiler içerebilirler. Bazı kuruluşlar bundan hareketle, sahip oldukları veritabanlarını böyle amaçlar için yayınlama eğiliminde olurlar. İyi niyetli bu eğilimden kaynaklanan veritabanı yayınlama, beraberinde bazı riskleri de getirir. Bu riskler veritabanında bulunan, veritabanı sahibi için hassas olabilecek bazı bilgilerin yayınlama ile beraber başkalarınca öğrenilmesi olabilir. Veritabanını elde eden üçüncü kişiler, gelişmiş veri madenciliği tekniklerini kullanarak bu hassas bilgileri açığa çıkarabilirler. Böylesi bir durum veritabanı yayınlayan kuruluşlar için arzu edilmeyen bir durum olduğundan, veritabanlarını yayınlayanlar veritabanlarını olduğu gibi değil de, kendileri için hassas olarak gördükleri bilgileri veritabanından eleyerek yayınlamayı tercih ederler. Bu şekilde bir yayınlama literatürde *mahremiyet kaygılı veri yayınlama* olarak geçmektedir ve ilk olarak 1991'de ortaya atılmıştır [31]. 1991'den bugüne bu kapsamda birçok çalışma yapılmış ve konu etkin bir çalışma alanı olmuştur [39, 9, 3, 24].

Mahremiyet kaygılı veritabanı yayıncılığında, ilk olarak veritabanında veri madenciliği teknikleri uygulanır. Veri madenciliği sonucunda elde edilen bilgiler arasında hassas olan bilgiler tespit edilir. Bu hassas bilgilerin yayınlanacak veritabanında, benzer veri madenciliği teknikleriyle elde edilmemesi için veritabanı için uygun bir dönüştürme işlemi tanımlanır. Dönüştürme işleminin belirlenmesinde dikkat edilen iki parametre vardır: Birincisi dönüştürme işlemi, hassas bilgileri elediğini garanti etmeli ve ikinci olarak da veritabanının orijinalliğini olabildiğince korumalıdır. İlk parametre dönüştürme işleminin doğru bir işlem olup olmadığını ikinci parametre de dönüştürme işleminin kalitesini gösterir. Kaliteden kasıt veritabanının orijinal halinin ne kadar korunduğudur. Dönüştürme işlemi çeşitli operasyonlar kullanılarak yapılabilmektedir. Çoğu zaman bu, veritabanındaki bazı verilerin hassas olarak belirlenen bilgiyi içermemesini sağlayacak şekilde değiştirilmesi veya silinmesidir. Değiştirme işlemi veritabanında, ilk durumda mevcut olmayan ve dolayısıyla gerçek olmayan bazı bilgilerin oluşmasına neden olabilmektedir. Silme işlemi ise değiştirmenin aksine gerçek olma-

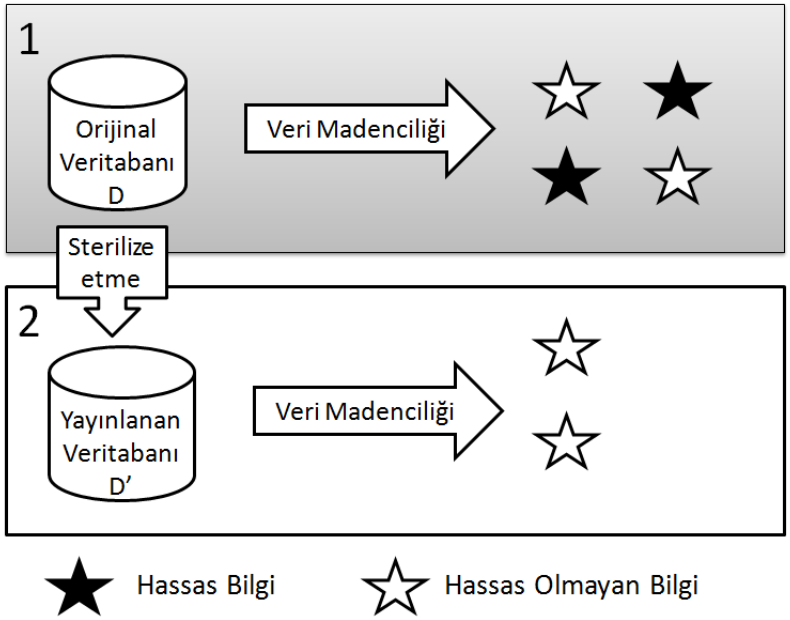
yan bilgiler üretmediği için genellikle tercih edilen bir yöntemdir.

Geliştirilen veri madenciliği teknikleri veritabanlarından istatistiksel olarak çok sayıda geçen ve aralarında istatistiksel ilişki bulunan örüntüleri keşfetmektedir. Dolayısıyla bu tezde gizlenen örüntüler mevcut veri madenciliği uygulamalarıyla sık olarak bulunan örüntülerdir.

Olduğu gibi veritabanı yayıncılığı beraberinde içerdiği hassas bilgilerin başkalarınca öğrenilmesi riskini doğurur. Şekil 1.1'de bir veritabanı yayınlama operasyonu betimlenmiştir. 1 numaralı bölüm veritabanının olduğu gibi yayınlandığı seçenektir. Veritabanı bu şekilde orijinal haliyle yayınlanırsa, veritabanını edinen kişiler uygun bir veri madenciliği uygulamasıyla veritabanında saklı olan bilgileri açığa çıkarabilir. Açığa çıkan bilgiler arasından bir kısmı hassas olabilir. Bu da veritabanı sahipleri tarafından arzu edilmeyen bir seçenektir. Diğer bir seçenek ise veritabanının orijinali üzerinde uygun sterilizasyon işlemi yapıldıktan sonra yayınlanmasıdır (şekilde 2 numaralı bölüm). Bu şekilde yayınlanan veritabanı üzerinde veri madenciliği uygulamasıyla bilgiler çıkarıldığında, bu bilgiler arasında hassas olan bilgiler olmayacaktır. Çünkü sterilizasyon işlemi ile veritabanı hassas bilgilerden arındırılmıştır. Böylece sterilize edilmiş veritabanı güvenle yayınlanabilir.

1.1 Motivasyon

Gerçek hayatta, içerisindeki hassas bilgilerin saklanması gereken veritabanları içerdikleri verilerin yapısına göre değişmektedir. Bu veritabanlarında ilk ortaya çıkan tür öge kümeleri veritabanlarıdır. Öge kümeleri veritabanları işlemlerden (*transaction*) oluşurlar. Yani, bir işlem ögelerini sonlu bir ögeler kümesinden alan bir kümedir. Bu tip veritabanlarına verilen klasik örnek market-sepeti tipi veritabanlarıdır. Marketlerde yapılan her alışveriş bir işlem, alışverişte satın alınan ürünler ise işlemin ögeleri olarak düşünülebilir. Zamanla birçok alışverişin kaydı tutularak oluşturulmuş böyle bir veritabanında veri madenciliği yapılarak hangi ürünlerin beraber çok sattığı öğrenilebilir. Bunlar arasında bazı bilgiler market için hassas olabilir ve bu bilgilerin başkalarınca



Şekil 1.1: Veritabanı yayınlama seçenekleri

öğrenilmesi istenmeyebilir. Bu durumda veri gizlemeye ihtiyaç duyulur.

Hassas bilgi gizleme yapılan veritabanlarından biri de dizgi (*sequence*) tipindeki veriler içeren veritabanlarıdır. Dizgi tipindeki bir veri örnek olarak GPS aracıyla herhangi bir kişinin gittiği yerlerin sırası olabilir. Farklı kişilerin farklı zamanlarda uğradıkları yerlerin kaydı tutularak dizgi tipinde veriler, bu verilerle de veritabanı oluşturulabilir. Böylece oluşturulmuş veritabanında bazı yerlerden bazı yerlere gidilmesi birçok gezintide geçiyorsa ve bu bilgi hassas ise bunun başkalarınca öğrenilmesi sakıncalı olabilir. Bu durumda veritabanının yayınlanması için veri gizleme işleminin yapılması gerekli olur.

Dizgi tipindeki verilerde her kayıt birden çok bilgiden oluşabilir. Mesela, yukarıda anlatılan dizgi tipindeki bir veride gidilen yer bilgisi ile zaman bilgisi beraber tutulabilir. Bu şekilde veriler çeşitlendirilebilir. Bu tip verilerden oluşan veritabanlarında gizleme de ihtiyaç duyulan bir operasyon olabilir.

Dizgi ve öge kümeleri dışında farklı yapıda olan veritabanları da zamanla oluşmaktadır. Buna bir örnek olarak ağaç tipindeki verilerden oluşan veritabanları verilebilir.

Ağaç tipindeki verilerin nasıl olacağı ve hassas bilgi gizlemenin ağaçlar üzerinde de uygulanabileceğine şu şekilde bir örnek verilebilir: Kategorik olarak oluşturulmuş, her kategorinin alt kategorilere sahip olduğu, her kategoriye kendine has bir sayfanın ayrıldığı ve herhangi bir kategori sayfasından o kategorinin sadece üst kategorisine ve alt kategorilerine bağlantıların yer aldığı bir web sitesi düşünelim. Bu web sitesindeki bir sayfaya erişen herhangi bir kullanıcı, sadece o sayfanın üst kategorisine ya da alt kategorilerine erişebilecektir. Bu durum erişilen tüm sayfalar için geçerlidir. Bu yapıdaki bir web sitesinde, bir kullanıcının bir gezintisi, art arda gezilen sayfalardan oluşur ki bu gerçekte ağaç tipindeki bir yapının gezilmesine karşılık gelir. Belirli bir süre içerisinde, web sitesine yapılan her erişimde, gezilen sayfaların kaydı tutularak bir veritabanı oluşturulabilir. Bu veritabanındaki her kayıt bir erişim ve o erişime ait gezilen sayfalardan oluşur. Böyle bir veritabanında, bazı gezinti örüntülerinin - mesela herhangi bir X kategorisinden onun üst kategorisine gidilmesi, oradan da gelinen kategori dışındaki bir kategoriye gidilmesi ve böyle bir hareketin çok sayıda kayıtta tekrar etmesi durumunun hassas olduğu düşünülebilir. Bu durumda veritabanının saf haliyle yayınlanması, veritabanı üzerinde üçüncü kişilerce veri madenciliği tekniklerinin kullanılmasıyla, hassas olduğu düşünülen gezintilerin açığa çıkmasına neden olur. Bu nedenle, ağaç tipi veriler de üzerinde hassas bilgi gizleme yapılabilecek veriler olabilmektedir. Benzer şekilde çizge tipi veriler içeren veritabanlarında da hassas bilgi gizleme yapılabilir. Örnek olarak sosyal ağlar çizge tipinde bir yapıya sahiptir ve hassas bilgi gizlemeye konu olabilmektedir.

1.2 Bilimsel Katkı

Bu tez kapsamında yapılan çalışmaların literatüre katkısı şu şekilde sıralanabilir:

- Hassas bilgi gizleme problemi daha önce öge kümeleri ve dizgi yapısındaki veritabanları için çalışılmıştır. Yapılan çalışmalarda birçok algoritma geliştirilmiştir. Fakat bu algoritmaların tümüne aynı anda ulaşmak mümkün değildir. Geliştirilen FISHER isimli uygulamada bu algoritmaların birçoğu tek bir uygulama altında toplanmıştır.

- FISHER uygulaması kullanılarak sık öge kümesi gizleme algoritmalarının performansları ölçülmüş ve hızlı olan algoritmanın hassas bilgiyi fazla yan etkiyle gizlediği; en yavaş olan algoritmanın ise en az yan etkiyle gizleme yaptığı tespit edilmiştir [5]. Bundan hareketle sık öge kümeleri tipindeki veritabanlarında hem hızlı olan hem de az yan etkiyle hassas bilgi gizleyen bir algoritma geliştirilmiştir. Geliştirilen algoritma FISHER uygulamasına da eklenmiştir.

- Daha önce üzerinde çalışılmamış olan ağaç tipi veritabanlarında hassas bilgi gizleme problemi üzerine çalışılmış, problem tanımı yapılmış ve problemin çözümü için algoritma geliştirilmiştir.

- Yine daha önce çalışılmamış olan çizge tipi veritabanlarında da hassas bilgi gizleme problemi tanımlanarak, problem çözümü için algoritma geliştirilmiştir.

- Ağaç ve çizgeler için geliştirilen algoritmalar hızlı, etkin ve olabildiğince az yan etkiye sahip algoritmalardır. Bu algoritmalar da FISHER uygulamasına eklenerek uygulamanın bütüncül bir veri gizleme uygulaması olması sağlanmıştır.

1.3 Döküman Yapısı

Tezin devamında sırasıyla Bölüm 2'de veri madenciliği ve bilgi gizleme, Bölüm 3'te sık öge kümeleri için tez kapsamında geliştirilmiş olan bir algoritma, Bölüm 4'te ağaç yapıları ve ağaçlarda bilgi gizleme algoritması, Bölüm 5'te çizge yapıları, çizgelerde bilgi gizleme algoritması, Bölüm 6'de farklı türdeki veritabanlarında hassas bilgi gizleme yapmak üzere geliştirilen FISHER uygulaması anlatılmaktadır. Bölüm 7 ise bu teze sonuç olarak yer almaktadır.

2 VERİ MADENCİLİĞİ VE HASSAS BİLGİ GİZLEME

Artan teknolojik gelişmelerle ve araçlarla günlük hayatta birbirinden farklı birçok veri elde edilmektedir. Çoğu zaman bu veriler toplanmakta ve veritabanlarında veya veri ambarlarında depolanmaktadır. Elde edilen verilerin gerçekten yararlı olup olmadığının, herhangi bir anlam taşıyıp taşımadığının insanlar tarafından çıplak gözle anlaşılmaları imkânsızdır. Bunun için verilerin araçlar ve uygulamalar yardımıyla incelenmesi gereklidir. Bu durumda da veri madenciliğinden bahsedilebilir. Tanım olarak veri madenciliği farklı veri ambarlarındaki verilerin birçok kritere göre geliştirilmiş uygulamalarla analiz edilmesi ve faydalı olabilecek sonuçlar elde etme işlemidir [46]. Faydalı olabilecek sonuçlardan kasıt, gerçek hayatta toplumsal, bilimsel ve sosyal faydaları olan sonuçlardır.

En yaygın veri madenciliği tekniği veritabanlarında gizli olan bilgilerin istatistiksel yönden açığa çıkarılmasıdır. Yani hangi örüntüler veritabanlarında sık geçmektedir, hangilerinin arasında ilişki vardır gibi sorulara yanıt vermektedir. Bu şekilde elde edilen örüntüler arasından bazılarının hassas olması muhtemeldir. Bunun için bu durum veritabanı yayıncılığında kuruluşların çekinceli kalmasına neden olur. Bu nedenle bu kuruluşların çekincelerinin giderilerek veritabanı yayınlamalarını sağlamak gerekir. Bu ise veritabanlarından hassas bilgilerin arındırılmasını gerektirir.

Bu bölüm, sırasıyla veri madenciliği, veri madenciliğinin uygulandığı veri tipleri, mahremiyet kaygılı veri madenciliği ve mevcut olan veri gizleme yöntemleri ve uygulandıkları veri tipleri hakkında bilgi vermektedir.

2.1 Veri Madenciliği

Bu bölümde veri madenciliği kapsamında sık öge küme madenciliği anlatılmaktadır ve benzer yaklaşımın uygulandığı veritabanları hakkında bilgi verilmektedir.

$\mathcal{I}$ = $\{i_1, i_2, \ldots, i_n\}$ bir semboller kümesi olsun. Bir T *işlemi (transaction)* $\mathcal{I}$'nın boş olmayan bir alt kümesidir: $T \in 2^{\mathcal{I}} - \emptyset$'dir. Bir $\mathcal{D}$ veritabanı birçok işlem içeren bir koleksiyondur: $\mathcal{D}$ = $\{T_1, T_2, \ldots, T_m\}$. Bir *X öge kümesi (itemset)* ise k adet öge içeren bir *k-öge kümesi*dir.

Tanım 1. *(Destek Değeri)* X bir öge kümesi olsun, $\mathcal{D}$ veritabanında X'in *destek değeri (support)* X'i altküme olarak içeren (X'i destekleyen) işlemlerin sayısıdır:
$sup_{\mathcal{D}}(X) = |\{T : X \subseteq T \wedge T \in \mathcal{D}\}|$.

Tanım 2. *(Sık Öge Küme Madenciliği [7])* σ kullanıcı tanımlı negatif olmayan bir tamsayı olsun ve *eşik değeri (disclosure threshold)* olarak adlandırılsın. Belirli bir $\mathcal{D}$ veritabanında, σ'dan küçük destek değerine sahip olmayan tüm öge kümelerine *sık öge kümeleri (frequent itemsets)* denir: $F_{(\mathcal{D},\sigma)} = \{X : X \subseteq I, X \neq \emptyset, sup_D(X) \geq \sigma\}$. Sık öge kümesi madenciliği ise $F_{(D,\sigma)}$'yı, verilen $\mathcal{D}$ veritabanında σ eşik değerine göre bulma problemidir. Tamsayı olarak verilen eşik değerine mutlak eşik değeri denir. Bazı durumlarda da verilen eşik değeri tamsayı değil de 0 ile 1 arasında oran gösteren bir sayı olabilir. Bu şekilde verilen eşik değerine göreceli eşik değeri denir ve veritabanındaki işlem sayısı ile çarpılarak mutlak eşik değeri karşılığı hesaplanabilir.

Tanım 3. *(İlişki Kuralı Madenciliği)* σ_1 ve σ_2 kullanıcı tanımlı negatif olmayan tamsayılar olsun. Belirli bir $\mathcal{D}$ veritabanında, σ_1'dan küçük destek değerine sahip olmayan $X \cup Y$ kümesi için ayrıca $sup_{\mathcal{D}}(X \cup Y) / sup_{\mathcal{D}}(X)$ değeri (*güven değeri*) σ_2'den büyükse X ile Y bir ilişki kuralının öge kümeleri olurlar ve bu durum şu şekilde gösterilir: $X \Rightarrow Y$. İlişki kuralı madenciliği problemi $\mathcal{D}$ veritabanından eşik değerlerine göre tüm $X \Rightarrow Y$ ilişki kurallarını bulma problemidir.

[39]'den alınmış örnek bir veritabanı Çizelge 2.1(a)'da verilmiştir. Bu veritabanı 9 işlem içermektedir ve her işlem elemanlarını $\mathcal{I} = \{a, b, c, d, e, f, g, h\}$ kümesinden almaktadır. Eşik değeri $\sigma = 3$ olarak düşünülürse, sık olan öge kümeleri Çizelge 2.1(b)'dekiler gibi olur. Çizelge 2.1(b)'nin her satırı *k-öge kümesi*'nden oluşmaktadır, 1. satırda 1 elemanlı öge kümeleri, 2. satırda 2 elemanlı öge kümeleri, v.s.

Tanım gereği sık öge kümelerini bulmak kolay görünse de, 1993'te ilk tanıtılmasından [7] beri etkili hesaplama konusunda önem çeken bir alan olmuştur. **Apriori** adında bir

Çizelge 2.1: (a) Örnek bir sık öge kümesi veritabanı, (b) $\sigma = 3$ eşik değerine göre bulunmuş sık öge kümeleri ve destek değerleri

(a)

Tid	Items
1	*abcde*
2	*acd*
3	*abdfg*
4	*bcde*
5	*abd*
6	*bcdfh*
7	*abcg*
8	*acde*
9	*acdh*

(b)

a :7, b :6, c :7, d :8, e :3
ab :4, ac :5, ad :6, bc :4, bd :5, cd :6, ce :3, de :3
abd :3, acd :4, bcd :3, cde :3

özellik sık öge kümelerini hızlı bulmak için önemli bir özellik olarak öne çıkmıştır. Bu özellik sık olmayan bir öge kümesini altküme olarak içerebilen tüm öge kümelerinin sık olmayacağını ifade eder. Apriori üstten-alta (*top-down*) ve genişlik-öncelikli (*breadth-first*) bir kafes (*lattice*) yapısı kullanır. Kafesin kökü tüm işlemlerde bulunan $\emptyset$'tir. Bu kafesin her düğümü öge kümesinden oluşur ve her düğüm ebeveyn düğümlere yeni bir öge eklenerek oluşturulur. Yani k elemanlı bir öge kümesine farklı ögeler eklenerek $k + 1$ elemanlı öge kümeleri oluşturulur. Yeni öge kümeleri oluşturulurken aynı zamanda oluşturulan öge kümesinin veritabanındaki destek değeri de hesaplanır. Eğer hesaplanan destek değeri belirlenen eşik değerinden küçükse (öge kümesi sık değilse) bu öge kümesinden artık yeni öge kümeleri oluşturulmaz. Bu yöntemle oluşturulan kafes tüm sık öge kümelerini destek değerleriyle tutar ve kafesteki tüm düğümler sık öge kümelerine karşılık gelir. Apriori özelliğinin en büyük sorunu aday öge kümelerinin oluşturulması ve destek değerlerinin sayılmasıdır. Aprioriden yararlanan fakat aday öge kümelerini üretmeyi ve destek değeri hesaplamayı farklı yaklaşımlar kullanarak yapan birçok algoritma geliştirilmiştir [6, 7, 19, 34].

Bu şekildeki veri madenciliği sadece öge kümeleri tipindeki veritabanları üzerinde uygulanmamaktadır. Benzer şekilde sık dizgi madenciliği de dizgi şeklinde veriler içeren veritabanları üzerinde uygulanabilmektedir [8].

Ağaç tipi verilerde ise bu şekildeki veri madenciliği sık altağaçları bulmak şeklinde olmaktadır. Ağaç veritabanları *orman (forest)* olarak adlandırılırlar. Altağaç madenleme algoritmaları genellikle küçük ağaçlardan başlayarak sık alt ağaçları bulurlar ve bu sık altağaçlara yeni düğümler eklemek yoluyla daha büyük altağaçları üretirler. Bir altağacın destek değeri verilen eşik değerinden küçük olduğu anda o ağacı daha da büyütmezler. Bu şekilde olabilecek tüm altağaçlar üretilmekte ve destek değerleri hesaplanmaktadır. Bu algoritmaların hızlılığını etkileyen en önemli unsur altağaçların üretilmesi ve destek değerlerinin sayılmasıdır. Ayrıca bir altağacın birden fazla üretilmemesi de hızlılık açısından gereklidir. En hızlı algoritmalardan biri [48]'de tanıtılmıştır. Bu algoritmada düğümler etiketlerinin sözlük sırasına ve altağaçların sırasına göre özel bir sıralamada tutulurlar. Ayrıca yeni bir ağaç mevcut bir altağacın kökten en sağdaki yaprağa olan yol üzerindeki düğümlerine düğüm eklemek (yaprak düğümler için çocuk eklemek, diğerleri için kardeş eklemek) suretiyle üretilir. Bu iki kriter beraber uygulanarak aynı ağacın birden fazla üretilmemesi sağlanır. Üretilen ağaçların etkin destek sayımı için ise veritabanı dikey bir yapıda oluşturulur. İlk önce veritabanında her farklı etiket için dikey bir sütun ayrılır ve bu sütunda ilgili etiketin hangi ağaçlarda ve düğümlerde geçtiği tutulur. Bu dikey yapıdaki veritabanı aynı anda hem ağaç oluşturmayı hem de ağaç oluşturulurken oluşturulan ağacın destek değerinin sayılmasını da kolaylaştırır.

Çizge tipi veriler içeren veritabanlarında da sık çizge madenciliği kapsamında çalışmalar yapılmıştır. Burada da sık ağaç madenciliğine benzer olan bir sistematik vardır. İlk önce en küçük çizgeler üretilmekte ve destek değerleri hesaplanmakta, sonra da bu çizgelere yeni düğümler eklenerek üretilen çizgelerin destek değerleri sayılmaktadır. Eşik değerinden büyük olan çizgeler sık çizgeler olarak hesaplanmaktadır. Çizge madenciliği için de etkili algoritmalar geliştirilmiştir [29, 30, 47, 23].

Ağaç ve çizgeler için bir önemli problem de bir örüntünün bir veride olup olmadığının yoklanmasıdır. Bu örüntü içerme, örüntü eşleme adlarıyla da literatürde geçmektedir. Örüntünün eşleme yapısına göre yapılmış birçok çalışma mevcuttur. Ağaç arama konusunda [37], değişen yapılardaki ağaç eşleme konusunda [25, 13, 11, 41, 43, 14, 20] ve eşbiçimlilik hesaplanması konusunda [42, 17, 15, 27, 28] birçok çalışma yapılmıştır.

2.2 Mahremiyet Korumalı Veri Madenciliği

Mahremiyet kaygılı veri yayıncılığı mahremiyet korumalı veri madenciliği altında incelenmektedir ve ilk olarak O'Leary tarafından üzerinde çalışılan bir konu olmuştur [31]. Bu çalışmada veri madenciliğinin veri yayıncılığı için bir risk unsuru olabileceği belirtilmiştir. Bunun nedeni gelişmiş veri madenleme uygulamalarıyla hassas ve mahrem bilgilerin açığa çıkarılabilir olmasıdır. Bu yüzden veri yayıncılığı önemle ele alınması gereken bir işlem olmalıdır. Bunun için öncelikle veri madenciliği ile hassas bilgilerin neler olduğu tanımlanmalı, ardından hassas bilgi gizleme uygulamaları aracılığıyla veritabanlarından bu hassas bilgiler arındırılmalıdır. Bu işlem veritabanı yayıcılarının mahremiyet kaygısından hareketle oluşan bir yöntemdir.

Veri yayınlama ile oluşabilecek mahremiyet ifşasının önüne geçebilmek için farklı yaklaşımlar mevcuttur.

2.2.1 Veri Bozmak

Veri bozmak veritabanında var olan bazı özniteliklerin hassas olduğu kabulü altında, özniteliklerden bazılarının değiştirilmesi ya da bozulmasıdır. Veri bozma esnasında veritabanının orijinal haliyle aynı kalmasına dikkat edilir, böylece orijinal veritabanı ile bozulmuş veritabanı neredeyse aynı özellikleri göstermesi amaçlanır. Bu amacın gerçekleşip gerçekleşmediği veritabanları üzerinde veri madenleme uygulamaları çalıştırılıp sonuçlarına bakılarak öğrenilebilir.

2.2.2 K-Anonimlik

K-anonimlik (k-anonymity) yayınlanan veritabanında her bir kayıt için en az k-1 (k>1) tane ilgili kayıt ile ayırt edilemeyen kayıtların olmasını gerektirir [40]. Böylece herhangi bir kayıt en az k-1 adet kayıtla ayırt edilemeyecek şekilde aynı özelliği gösterir. $\mathcal{D}$ veritabanında ayırt edici öznitelikler varsa k-anonimlikten söz edilemez ve dola-

yısıyla bu özniteliklerin yayınlanmadan önce çıkarılması gerekir. Fakat bu işlem de her zaman çözüm olmayabilir. Çünkü veritabanı içerisinde anahtar gibi davranan gizli anahtarlar (*quasi-identifier*) olabilir. Bu anahtarlar daha önce yayınlanmış farklı veritabanları ile bitiştirilince (*join*) gizli öznitelikler ayırt edici olabilir. Bu da mahremiyetin açığa çıkmasına neden olur.

Veritabanını yayınlayan kurum, daha önce yayınlanmış tüm veritabanlarını kontrol edemeyeceği için kendi veritabanını gizli anahtarlardan arındırmalı ve o şekilde yayınlamalıdır. Bunun için gizli anahtarlar tespit edilmeli ve bu anahtarların veritabanında her kayıt için en az k-1 adet aynı gizli anahtar olmalıdır. Gizli anahtarların ortadan kaldırılması daha çok gizli anahtarların üst nitelikleriyle temsil edilecekleri hale getirilmesi şeklinde olur.

2.3 Hassas Bilgi Gizleme

Yayınlanan veritabanında, yayınlayıcının başkalarınca bilinmesini istemediği bazı ilişkiler ve örüntüler olabilir. Bilgi gizleme ile hedeflenen bilinmesi istenmeyen ilişki ve örüntülerin veritabanından yok edilmesi ve ardından veritabanının yayınlanmasıdır. Bu şekildeki veritabanında veri madenleme yapılsa bile hassas bilgiler öğrenilmeyecektir.

Veritabanları ve hassas bilgiler çok farklı yapıda ve boyutta olabileceğinden, mahremiyet kaygısını giderecek etkin algoritmalara ihtiyaç vardır. Geliştirilen algoritmalar uygulandıkları veri türüne göre farklılık gösterirler ve hassas bilgini gizlenmesi problemi de böylece veri tipi bağımlı olur. Hassas bilgi gizleme problemi sık öge kümeleri üzerinde uygulanıyorsa, sık öge kümesi gizleme [9]; dizgiler üzerinde uygulanıyorsa dizgi gizleme [2, 1, 3]; ağaçlar üzerinde uygulanıyorsa ağaç gizleme ve çizgeler üzerinde uygulanıyorsa çizge gizleme problemi olur.

2.3.1 Sık Öge Kümesi ve İlişki Kuralı Gizleme

Sık öge madenciliği sonucunda elde edilen örüntülerin bir kısmı bilgi içerebilir. Bu bilginin bir kısmı hassas olabilir ve bu bilginin yayınlanacak veritabanında bulunması veritabanı sahibini veritabanını yayınlamaktan alıkoyabilir. Dolayısıyla veritabanının yayınlanması için veritabanı sahibi için hassas olan bilginin korunması gerekir. Hangi bilginin hassas olup olmadığı ise veritabanı sahibi tarafından karar verilecek bir durumdur.

Tanım 4. *(Hassas Öge Kümesi Gizleme)* $P_h = \left\{X_i \mid X_i \in 2^{\mathcal{I}} \wedge i = 1, 2, \ldots, n\right\}$ n tane hassas öge kümesi olsun. Verilen bir ψ eşik değerine göre *sık öge kümesi gizleme* problemi $\mathcal{D}$ veritabanını $\mathcal{D}'$ veritabanına dönüştürmeyi gerektirir, öyle ki:

- $\forall X_i \in P_h : sup_{\mathcal{D}'}(X_i) < \psi$.
- $\sum_{X \in (2^{\mathcal{I}}) \setminus \mathcal{P}_h} \mid sup_{\mathcal{D}}(X) - sup_{\mathcal{D}'}(X) \mid$ minimum olmalıdır.

Dönüştürme işlemi sterilize etmek (*sanitization*) olarak da geçmektedir. Dönüştürme ile elde edilen $\mathcal{D}'$, $\mathcal{D}$'nin yayınlanabilir versiyonudur, çünkü hassas bilgileri artık içermemektedir. Problemin birinci koşulu tüm hassas öge kümelerinin destek değerlerinin verilen eşik değerinin altına indirilmesini gerektirir, böylece bu hassas öge kümeleri ilgili eşik değerine göre artık hassas değillerdir. Yani dönüştürmeden sonra $P_h \bigcap F_{(\mathcal{D}', \psi)} = \emptyset$'dir. Problemin ikinci koşulu dönüştürülmüş veritabanının olabildiğince orijinal haline benzemesini gerektirir. İkinci koşula riayet edilmeyen en sıradan çözüm $\mathcal{D}' = \emptyset$ olmasıdır. Ama veritabanı yayıncılığı belli bir amaçla yapıldığından, ikinci koşul ne kadar yerine getirilirse bu amaca o kadar yaklaşılmış olunur.

Çizelge 2.1(a)'daki veritabanında örnek gizlemeler için hassas öge kümeleri $\mathcal{P}_h = \{\{a, c, d\}, \{a, d\}, \{b, c, d\}\}$ olsun. Buna uygun iki farklı gizlenmiş veritabanı Çizelge 2.2'de verilmiştir. Eşik değeri 3 olarak yapılan gizleme sonucu oluşan veritabanlarının ikisi de artık hassas öge kümelerini verilen eşik değerinden daha az destekleyecek şekilde içermektedir. Böylece hassas öge kümesi gizleme probleminin ilk gereği ye-

Çizelge 2.2: $\psi = 3$ eşik değerine göre farklı iki gizleme

(a) $\mathcal{D}_1^{'}$

Tid	Items
1	*bcde*
2	*cd*
3	*bdfg*
4	*cde*
5	*bd*
6	*bcdfh*
7	*abcg*
8	*acde*
9	*acdh*

(b) $\mathcal{D}_2^{'}$

Tid	Items
1	*abce*
2	*cd*
3	*abdfg*
4	*bcde*
5	*ab*
6	*bcdfh*
7	*abcg*
8	*ace*
9	*acdh*

Çizelge 2.3: Çizelge 2.2'deki veritabanlarından elde edilen sık öge kümeleri ($\psi = 3$)

(a) $F_{(D_1^{'},3)}$

a :3, b :5, c :7, d :8, e :3
ac :3, bc :3, bd :3, cd :6, ce :3, de :3
cde :3

(b) $F_{(D_2^{'},3)}$

a :6, b :6, c :7, d :5, e :3
ab :4, ac :4, bc :4, bd :3, cd :4, ce :3

rine getirilmiştir. Bu iki sterilize edilmiş veritabanında veri madenciliği sonucuna göre eşik değeri 3 için sık öge kümeleri Çizelge 2.3'de verilmiştir. Bu sonuçlara da bakıldığında sterilize edilmiş ilk veritabanında toplam 12 sık öge kümesi varken, ikinci veritabanında 11 sık öge kümesi vardır. Bu sonuç veritabanlarının farklı şekilde sterilize edilebileceğini fakat her sterilize işleminin veritabanını farklı bir hale getirdiği göstermektedir. Hassas olmayan sık öge kümelerinin korunması problemin ikinci gereği olduğu için bu iki gizleme işleminden birincisinin daha etkili olduğu açıktır.

Literatürde var olan hassas gizleme algoritmalarının çoğu problemin ilk gereğini sağlarken bu örnekte olduğu gibi ikinci gereği farklı biçimlerde sağlamaktadır.

Tanım 5 (İlişki Kuralı Gizleme)**.** $\mathcal{P}_h = \{X_1 \Rightarrow Y_1, X_2 \Rightarrow Y_2, \ldots, X_n \Rightarrow Y_n\}$ $\mathcal{D}$'den gizlenecek olan ilişki kurallarının kümesi olsun. Verilen (ψ_1, ψ_2) eşik değeri çifti için, İlişki Kuralı Gizleme problemi $\mathcal{D}$'nin $\mathcal{D}'$'ye dönüştürülmesini gerektirir, öyle ki:

1. $\mathcal{D}$'de her hassas ilişki kuralının destek değeri ψ_1'den küçük olmalıdır: $\forall X_i \Rightarrow Y_i \in \mathcal{P}_h : sup_{\mathcal{D}'}(X_i \bigcup Y_i) < \psi_1 \vee \frac{sup_{\mathcal{D}'}(X_i \bigcup Y_i)}{sup_{\mathcal{D}'}(X_i)} < \psi_2$;

2. $\sum_{X \in (2^{\mathcal{I}}) \setminus \mathcal{S}} |sup_{\mathcal{D}}(X) - sup_{\mathcal{D}'}(X)|$ minimum olmalıdır. $\mathcal{S} = \{X \bigcup Y : X \Rightarrow Y \in \mathcal{P}_h\}$.

İlk koşul hassas bilgi gizlemeyi garanti ederken, ikinci koşul veritabanı bütünlüğünü korumayı amaçlar.

İlişki kuralı gizleme probleminin en makul yolu hassas ilişki kurallarının destek ve/veya güven değerlerini tanımlı eşik değerlerinin altına düşürmektir. Bunun için hassas öge kümesi gizleme için geliştirilmiş algoritmalar kullanılabilir. Güven değeri düşürülerek de ilişki kuralı gizleme yapılabileceğinden, bu yönüyle hassas öge kümesi gizlemeden farklılaşır ve güven düşürmeyi temel alan algoritmalar kullanılabilir.

Bilgi gizleme ilk olarak ilişkisel veritabanları için çalışılan bir alan olmuştur [9]. Bu çalışmada ilişki kurallarının destek ve güven değerleri düşürülerek gizleme yapılmıştır. Gizleme yapılırken hassas olmayan bilgilerin korunması temel amaçtır. Fakat bu problem NP-Hard olduğundan yapılan çalışmalarda etkin sezgiseller önerilmiştir.

[16] numaralı çalışmada da hem hassas ilişki kurallarında yer alan öge kümelerinin destek değerlerinin hem de ilişkilerin güven değerlerinin düşürülmesi ile gizleme işlemi yapılmaktadır. Bir ilişki kuralının güven değerinin düşürülmesi için ya kuraldaki öncül öge kümesinin destek değerinin sadece bu öge kümesinin içeren işlemler arasından arttırılması ya da ilişkinin sağındaki öge kümesinin destek değerinin ilişkinin her iki öge kümesini içeren işlemler arasından düşürülmesi gerekir. [44] numaralı çalışmada da benzer bir mantık takip edilmektedir.

[35, 36]'da "*bilinmeyenler*" kavramı kullanılmıştır. Buradaki amaç tanımlanabilir hassas ilişki kurallarının işlemlerdeki bilinen değerlerin bilinmeyen değerlerle değiştirilerek gizlenmesini sağlamaktır.

[39]'da *sınır* kavramından hareketle işlemler kaybolan sık öge kümeleri dikkate alınarak ele alınmıştır. Bunun için öge kümesi latisi kullanılmıştır ve *pozitif* ile *negatif* sınırların hesaplanmasının kolaylaşması amaçlanmıştır. Önerilen yöntem sınır değerle-

rin olabildiğince korunmasıdır. Böylece veritabanı da olabildiğince korunmuş olmaktadır. Sınır kavramından yararlanan bir diğer çalışmada da tamsayı programlama mantığı kullanılmış ve en az sayıda işlemin sterilize edilmesiye gizleme işleminin başarılması amaçlanmıştır [26]. Hassas öge kümesi gizleme konusunda yapılan çalışmalara [33, 24] numaralı çalışmalar da örnek olarak verilebilir.

2.3.2 Dizgi Örüntüleri Gizleme

Sık öge kümelerinde ögelerin sırası önemsizdir. Fakat sık öge kümesine benzer şekilde işlemlerden oluşan, her işlemin de ögeleri sıralı olan veritabanlarında da hassas bilgi gizleme geçerli olmaktadır. Çünkü bu tip veritabanlarından sık örüntüleri bulan veri madenciliği algoritmaları mevcuttur. Bu durum da sık olan örüntüler arasında yer alan hassas örüntülerin anlaşılması sonucunu doğurur.

$\mathcal{D}$ bir dizgiler veritabanı olsun. Bu durumda $T \in \mathcal{D}$ler Σ: $T = \langle t_1, \ldots, t_{T_n} \rangle$ şeklinde ifade edilir, burada $t_i \subseteq \Sigma, \forall i \in \{1, \ldots, T_n\}$'dir. Tüm dizgilerin kümesi $\{2^\Sigma - \emptyset\}^*$ ile gösterilir. Bir $U \in \Sigma^*$ dizgisi $V \in \Sigma^*$'nin $U \sqsubseteq V$ ile gösterilen bir alt dizgisidir. U V'den bazı semboller silinerek elde edilir. Örneğin $U = \langle u_1, \ldots, u_m \rangle$ $V = \langle v_1, \ldots, v_n \rangle$'nin bir alt dizgisidir, bunun için ise m adet $i_1 < \ldots < i_m$ indisi olmalıdır ve ek olarak da $u_1 \subseteq v_{i_1}, \ldots, u_m \subseteq v_{i_m}$ olmalıdır. Dizgi veritabanlarına benzer olarak *dizgisel örüntüler* de sık öge kümelerinin sıralısı olarak tanımlanırlar.

Bir S dizgisinin destek değeri $\mathcal{D}$'de bu dizgiyi alt dizgi olarak içeren işlem sayısıdır: $sup_{\mathcal{D}}(S) = |\{T \in \mathcal{D} \,|\, S \sqsubseteq T\}|$. Sık dizgi madenleme algoritması [8], sık öge kümesi madenleme algoritmasına benzer olarak verilen bir $\mathcal{D}$ veritabanından verilen σ eşik değerinden büyük destek değerine sahip olan tüm dizgileri bulmaktadır: $\mathcal{F}(\mathcal{D}, \sigma) = \{S \in \{2^\Sigma - \emptyset\}^* \,|\, sup_{\mathcal{D}}(S) \geq \sigma\}$.

Dizgi gizleme problemi, sık dizgi madenleme algoritmalarıyla öğrenilebilecek hassas dizgileri gizlemeyi gerektirir [2].

Tanım 6 (Dizgi Gizleme Problemi)**.** $\mathcal{P}_h = \{S_1, S_2, \ldots, S_n\}$, $S_i$lerin $S_i \in \{2^\Sigma - \emptyset\}^*, \forall i \in \{1, \ldots, n\}$ olduğu durumda hassas olan dizgiler kümesi olsun ve $\mathcal{D}$'den gizlenmesi gereksin. Verilen ψ eşik değerine göre Dizgi Gizleme Problemi $\mathcal{D}$ veritabanını $\mathcal{D}'$ veritabanına dönüştürmeyi gerektirir, öyle ki:

1. $\forall S_i \in \mathcal{P}_h, sup_{\mathcal{D}'}(S_i) < \psi$;
2. $\sum_{S \in \{2^\Sigma - \emptyset\}^* \setminus \mathcal{P}_h} |sup_{\mathcal{D}}(S) - sup_{\mathcal{D}'}(S)|$ minimum olmalıdır.

Dizgi gizleme problemi farklı boyutlardaki veriler üzerinde uygulanabilir. Dizgiler tek boyutlu da olabilirler, iki boyutlu (zaman-mekân izleri) da olabilirler. Hem tek boyutlu hem de iki boyutlu dizgilerde gizleme konusu yakın zamanda çalışılmıştır [2, 1, 3].

3 DENGELİ SIK ÖGE KÜMESİ GİZLEME

[5] numaralı çalışma kapsamında hassas bilgi gizleme işlemini farklı veritabanları üzerinde yapan ve farklı algoritmalar içeren *FISHER* isimli bir uygulama geliştirilmiştir. Bu uygulama hassas öge kümesi gizleme için aralarında Cyclic Hiding Algoritması (CHA) [9] ve Border Based Hiding Algoritmasının (BBHA) [39] da aralarında bulunduğu farklı 5 algoritma içermektedir. Bu uygulamayla `http://fimi.cs.helsinki.fi/data/` adresinden edinilen `Retail` veritabanı üzerinde birtakım performans testleri yapılmıştır. Bunun için veritabanından ortalama destek değerleri 977,95 olan 20 tane sık öge kümesi hassas olarak belirlenmiş ve bu hassas sık öge kümeleri 500, 400, 300, 200, 100 eşik değerleri için BBHA ve CHA kullanılarak gizlenmiştir. Test için bu iki algoritmanın seçilmesinin nedeni CHA'nın çok basit bir yapıda olması ve son derece hızlı çalışması, BBHA'nın da veritabanı bütünlüğünü korumak için oldukça ileri seviye olmasıdır.

CHA ve BBHA aşağıda sırasıyla Algoritma 1 ve Algoritma 2'de verilmiş ve devamında anlatılmıştır.

CHA ilk kez [9] numaralı çalışmada önerilmiştir. Bu algoritma sırasıyla her hassas sık öge kümesini saklar. Herhangi bir hassas sık öge kümesi için 4. satırda o hassas öge kümesini destekleyen işlemi bulur ve bu işlemden hassas öge kümesi ögelerinden birini silerek ilgili hassas öge kümesinin destek değerini bir düşürmüş olur. Bu şekilde yeterince işlemden öge silmesi yapılırsa ilgili hassas öge kümesi gizlenmiş olur.

BBHA [39] algoritması ilk olarak 1. satırda sık öge kümelerinin pozitif sınırını, 2. satırda da hassas öge kümelerinin negatif sınırını hesaplar. 3. satırda hassas öge kümeleri öge sayılarının azalan sırasında ve destek değerlerinin artan sırasında sıralanırlar. Sıralanmış hassas öge kümeleri 4 ile 11 satırları arasında teker teker gizlenirler. Herhangi bir X hassas öge kümesinin destek değeri 7 ile 10. satırlar arasında 1 azaltılır. Bu satırlar X eşik değerinden küçük destek değerine sahip oluncaya kadar devam eder. $i \in X \subseteq T$ olduğu bir işlem ve öge çifti (T, i) $HidingCandidates$ metoduyla hesap-

Algoritma 1 Cyclic Hiding Algorithm (*CHA*)

Girdi: $\mathcal{D}$, $\mathcal{P}_h$, ψ
Çıktı: $\mathcal{D}'$

for all $X \in \mathcal{P}_h$ **do**
  $Sup^X \leftarrow sup_{\mathcal{D}}(X)$
  while $Sup^X \geq \psi$ **do**
    $X \subseteq T$ koşulunu sağlayan sıradaki $T \in \mathcal{D}$ yi bul
    Sıradaki $i \in X$i T'den sil
    $Sup^X \leftarrow Sup^X - 1$
  end while
end for
$\mathcal{D}' \leftarrow \mathcal{D}$

Algoritma 2 Border-Based Hiding Algorithm (*BBHA*)

Girdi: $\mathcal{D}$, $\mathcal{P}_h$, ψ, λ
Çıktı: $\mathcal{D}'$

$Bd^+ \leftarrow PositiveBorder(F_{(\mathcal{D},\psi)})$
$\mathcal{P}_h \leftarrow LowerBorder(\mathcal{P}_h)$
$\mathcal{P}_h$ ı boyutlarının azalan, destek değerlerinin artan sırasına göre sırala
for all $X \in \mathcal{P}_h$ **do**
  $V \leftarrow \emptyset$
  $C \leftarrow HidingCandidates(X, \mathcal{D}, Bd^+)$
  while $sup_{\mathcal{D}}(X) \geq \psi$ **do**
    $c \leftarrow SelectCandidate(C, X, \mathcal{D}, Bd^+, \psi, \lambda)$
    $C \leftarrow C \setminus c$
    $V \leftarrow V \bigcup c$
  end while
  $\mathcal{D} \leftarrow Update(\mathcal{D}, V)$
end for
$\mathcal{D}' \leftarrow \mathcal{D}$

lanan gizleme aday işlemleri ve ögelerindendir. $SelectCandidate$ metodu bu adaylar arasından hassas olmayan sık öge kümelerinin pozitif sınırını en az bozacak işlem ve ögeyi seçer. Seçilen aday işlemden seçilen aday öge silinir.

Algoritma oldukça karmaşık ve hızlılık konusunda etkisizdir. Algoritmanın amacı öge silme işlemi ile sık öge kümelerinin pozitif sınırını en az değiştirmek, böylece hassas olmayan sık öge kümelerinin gizleme işleminden sonra da sık olmasını sağlamaktır.

FISHER uygulamasındaki bu iki algoritma için yapılan testte değişen eşik değerlerine göre çalışma zamanları Çizelge 3.1'de gösterilmiştir. Bu çizelgede çalışma zamanları saniye türünden verilmiştir. Görülen şudur ki, CHA, BBHA'dan çok daha hızlıdır. Hız farkı o kadar fazladır ki, bu fark bazen 1000 kattan daha fazla olabilmektedir. Çizelge 3.2 ise hassas olmayan sık öge kümelerinin ne kadarının kaybolduğunu göstermektedir. Hız açısından oldukça kötü olan BBHA burada da CHA'ya göre oldukça iyi sonuçlar vermiştir. Fakat hız konusunda olan farkın aksine buradaki sonuçlarda görülen oranlar en fazla birkaç kattır.

Çizelge 3.1: CHA ve BBHA çalışma zamanları.

Eşik Değeri	**500**	**400**	**300**	**200**	**100**
BBHA (sn)	1531.8	2221.6	3599.6	6704.2	20785.4
CHA (sn)	1,607	1,716	1,856	1,935	2,231

Çizelge 3.2: Kaybolan sık öge kümeleri

Eşik Değeri	**500**	**400**	**300**	**200**	**100**
Hassas Olmayan sık öge kümeleri	472	609	1158	2242	6645
BBHA.	56	80	141	262	977
CHA.	88	140	257	466	1515

Bu iki sonuçtan anlaşılmaktadır ki veritabanı bütünlüğünün korunması ne kadar amaçlanırsa çalışma zamanı o kadar artmaktadır. Bu durumda akla ilk gelen konu hem çalışma zamanı açısından hem de veritabanı bütünlüğünü koruma açısından iyi bir algoritmanın mümkün olup olmadığıdır. Bundan hareketle çalışma zamanı olarak CHA'ya yaklaşan, veritabanı bütünlüğünü koruma açısından da BBHA'ya yaklaşan bir algoritma (Dengeli Gizleme Algoritması (DGA)) geliştirilmiştir. Bu algoritma da 3'de tanıtılmıştır.

3.1 Dengeli Gizleme Algoritması

Algoritma 3 Dengeli Gizleme Algoritması
Girdi : D, P_h, φ
Çıktı : D'

1. Tüm $\forall T \in D$'leri uzunluklarının artan sırasına göre sırala
2. *for all* $X \in P_h$
   (a) $Sup^X \leftarrow supp_D(X)$
   (b) *while* $Sup^X \geqslant \varphi$
      i. X'i destekleyen sıradaki $T \in D$'yi al.
      ii. $SS_{P_h}(T) \leftarrow \{Y : Y \in P_h \wedge Y \subseteq T\}$
      iii. *kurban* $\leftarrow SS_{P_h}(T)$ de en çok sayıda geçen X ögesi
      iv. T'den kurban ögeyi sil.
      v. $Sup^X \leftarrow Sup^X - 1$
3. $D' \leftarrow D$

CHA hızlı çalışmasına karşın, sildiği ögelerin seçimini veritabanı bütünlüğünü koruma açısından etkili bir şekilde yapmaz. Dengeli Gizleme Algoritmasının geliştirilmesinin nedeni, CHA'ya yakın hızda çalışan ama sildiği ögeleri daha etkin ve karmaşık bir şekilde seçerek veritabanı bütünlüğünü korumayı amaçlanan bir algoritmaya olan ihtiyaçtır.

Algoritma 3 ilk önce veritabanındaki tüm işlemleri içerdikleri eleman sayısına göre artan sırada sıralar. Bundaki amaç, boyutu az olan bir işlemin daha az sayıda hassas olmayan öge kümesini desteklemesidir. Böylece hassas olmayan birçok öge kümesine ait ögelerin silinmesinden kaçınılmış olur. Hassas öge kümeleri sırasıyla saklanırlar. Her sık öge kümesi için algoritma ilgili sık öge kümesini destekleyen sıradaki işlemi alır. Bu işlemin desteklediği tüm hassas öge kümeleri bulunur ve bunlar $SS_{P_h}(T)$ ile gösterilir. Desteklenen hassas öge kümeleri içerisinde en çok sayıda bulunan ortak öge kurban olarak seçilir. Seçilen kurban öge mevcut işlemden silinir. Kurban ögenin bu şekilde bulunması ve silinmesiyle birden çok hassas öge kümesinin destek değeri azaltılmış olur. Her işlem seçme ve kurban silme işleminden sonra en az bir öge kümesinin destek değeri azalır. Bu şekilde tüm hassas öge kümelerinin gizlenmesi garanti edilir.

Bunun yanında da veritabanı bütünlüğü korunmuş olur.

Çizelge 3.3: Örnek veritabanı.

	X	$supp_D(X)$	$SS_{P_h}(T)$	T	$victim$
1	*acd*	*4*	*{acd, ad}*	*acd*	*a*
2	*acd*	*3*	*{acd,ad}*	*acde*	*a*
3	*ad*	*4*	*{ad}*	*abd*	*a*
4	*ad*	*3*	*{a}*	*acdh*	*a*
5	*bcd*	*3*	*{bcd}*	*bcde*	*b*

Çizelge 3.3'teki örnekte silinecek aday ögeler gösterilmiştir. Çizelgedekidaki her satırda saklanan hassas öge kümesi, bu öge kümesinin anlık destek değeri, onu destekleyen sıradaki ilk işlem ve silinecek kurban öge görülmektedir. Bir satırdan diğerine geçilirken işlemden kurban öge silinmektedir ve hassas sık öge kümesinin destek değeri de bir azalmaktadır. Eğer bu öge kümesinin destek değeri artık sık değilse aynı işlem sıradaki hassas sık öge kümesi için yapılmaktadır. Sonuç veritabanının her silme işleminden sonraki şekli Çizelge 3.4'de verilmiştir. İçerisinde silme işlemi yapılan işlemlerin son hali koyu olarak gösterilmiştir. Son sütunda ise dönüştürülmüş veritabanının son hali görülmektedir.

Çizelge 3.4: Veritabanının gizleme işlemi boyuncaki hâli

D	*i*.silme işleminden sonrası					**D'**
Initial	*i=1*	*i=2*	*i=3*	*i=4*	*i=5*	Final
abd	*abd*	*abd*	***bd***	*bd*	*bd*	***bd***
acd	***cd***	*cd*	*cd*	*cd*	*cd*	***cd***
abcg	*abcg*	*abcg*	*abcg*	*abcg*	*abcg*	***abcg***
acde	*acde*	***cde***	*cde*	*cde*	*cde*	***cde***
acdh	*acdh*	*acdh*	*acdh*	***cdh***	*cdh*	***cdh***
bcde	*bcde*	*bcde*	*bcde*	*bcde*	***cde***	***cde***
abcde	*abcde*	*abcde*	*abcde*	*abcde*	*abcde*	***abcde***
abdfg	*abdfg*	*abdfg*	*abdfg*	*abdfg*	*abdfg*	***abdfg***
bcdfh	*bcdfh*	*bcdfh*	*bcdfh*	*bcdfh*	*bcdfh*	***bcdfh***

3.2 Algoritma Performans Testleri

Bu bölümde algoritmanın BBHA ve CHA'ya göre performans testleri verilmektedir. Test veritabanları olarak IBM sentetik veriseti üreteci kullanılarak iki tane veritabanı

oluşturulmuştur. Bu veritabanları `T5I250K` ve `T10I450K` olarak isimlendirilmiştir. Her iki veritabanı da 500 farklı öge içermekte ve veritabanı özellikleri isimlerinden anlaşılabilmektedir. İlk veritabanı ortalama 5 öge içeren işlemlerden oluşmaktadır. Bu veritabanında bulunan sık öge kümeleri ortalama 2 elemanlıdır ve veritabanı toplam 50000 işlem içermektedir.

`http://fimi.cs.helsinki.fi/data/`'den edinilen `Retail` [10] veritabanı da performans testleri çin kullanılmıştır. Bu veritabanında 16470 farklı öge ve 88163 işlem bulunmaktadır. Her işlem ortalama 13 ögelidir. Her veritabanı için 20 farklı öge kümesi hassas olarak seçilmiştir. Seçilen hassas öge kümelerinin karekteristiği Çizelge 3.5'de betimlenmiştir.

Çizelge 3.5: Hassas öge kümeleri özellikleri

	T5I250K	T10I450K	Retail
Hassas Öge Kümelerinin Ortalama Boyutu	2,3	2	2,3
Hassas Öge Küme Sayısı	20	20	20
Hassas Öge Kümelerinin Ortalam Destek Değeri	121,8	249,8	977,95

3.2.1 Değerlendirme Metrikleri

Performans testlerinde şu değerlendirme metrikleri kullanılmıştır:

- Çalışma zamanı : Saniye cinsinden toplam çalışma zamanı
- Veri Bozulması: (M0) = $\sum_{T \in D} |T| - \sum_{T \in D'} |T|$.
- Bilgi Kaybı: (M1) [32] = $\frac{\sum_{i \in I} sup_D(\{i\}) - sup_{D'}(\{i\})}{\sum_{i \in I} sup_D(\{i\})}$.
- Kalite: (M2) = $\frac{\left|F_{(D',\varphi)}\right|}{\left|F_{(D,\varphi)} - P_h\right|}$.

- Sık Destek Bozulması: (M3) [2] = $\frac{1}{\left|F_{(D',\varphi)}\right|} \sum_{X \in F_{(D',\varphi)}} \frac{sup_D(X) - sup_{D'}(X)}{sup_D(X)}$.
- Sık Örüntü Bozulması: (M4) [2] = $\frac{\left|F_{(D,\varphi)}\right| - \left|F_{(D',\varphi)}\right|}{\left|F_{(D,\varphi)}\right|}$.

Bu metriklerden M2 dışındaki hepsi için "ne kadar az o kadar iyi" özelliği geçerli iken, M2 için "ne kadar çok o kadar iyi" özelliği geçerlidir.

3.2.2 Performans Sonuçları

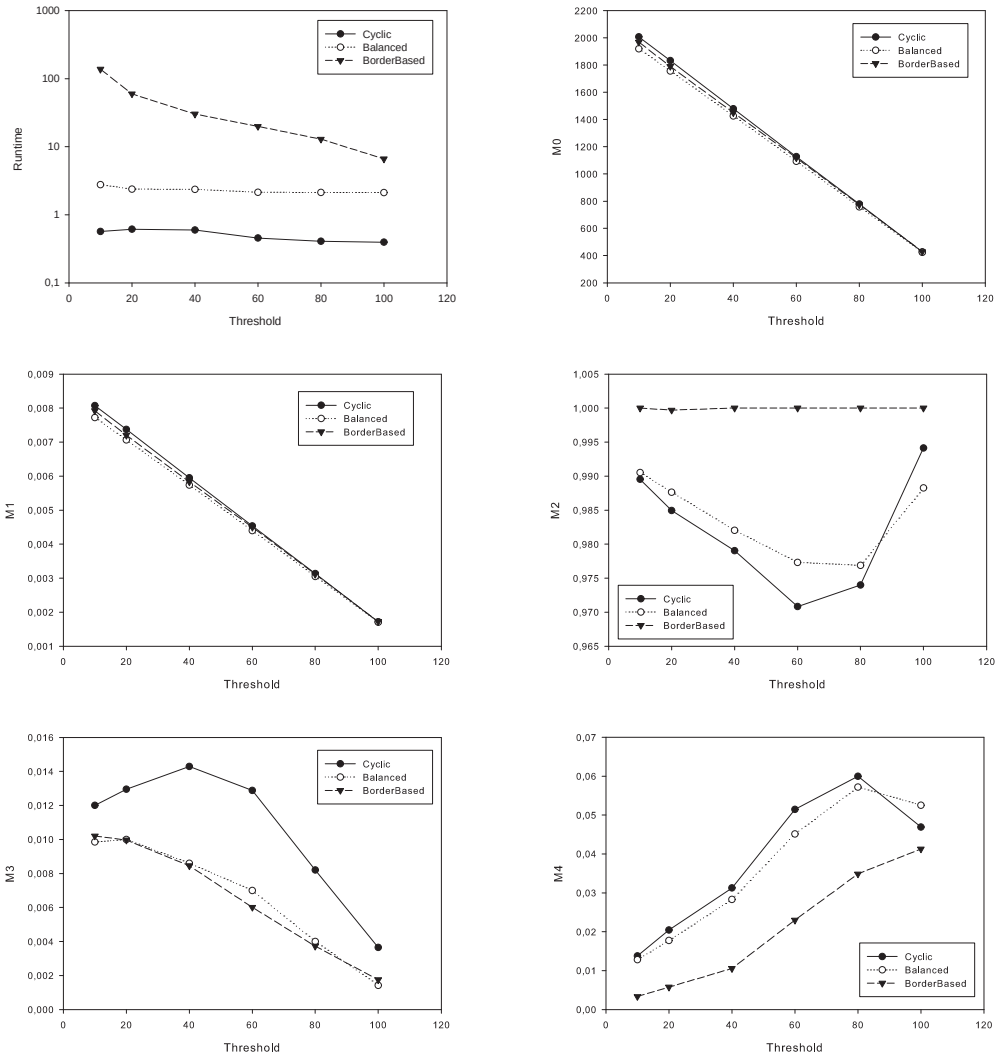
Her üç veritabanı için performans sonuçları Şekil 3.1, Şekil 3.2 ve Şekil 3.3'te gösterilmiştir.

`T5I250K` veritabanı için kalite metriklerine bakıldığında en iyi sonuç veren BBHA'dır. En kötü sonuç veren ise CHA'dır. Dengeli Gizleme algoritması ise BBHA'ya yakın sonuçlar vermiştir. Çalışma zamanı olarak bakıldığında ise en kötüsünün BBHA en iyisinin CHA olduğu görülmektedir. Dengeli Gizleme Algoritması ise bu yönden CHA' ya oldukça yaklaşmakta gizleme işlemini kısa sürede gayet kaliteli yapan bir algoritma olmaktadır.

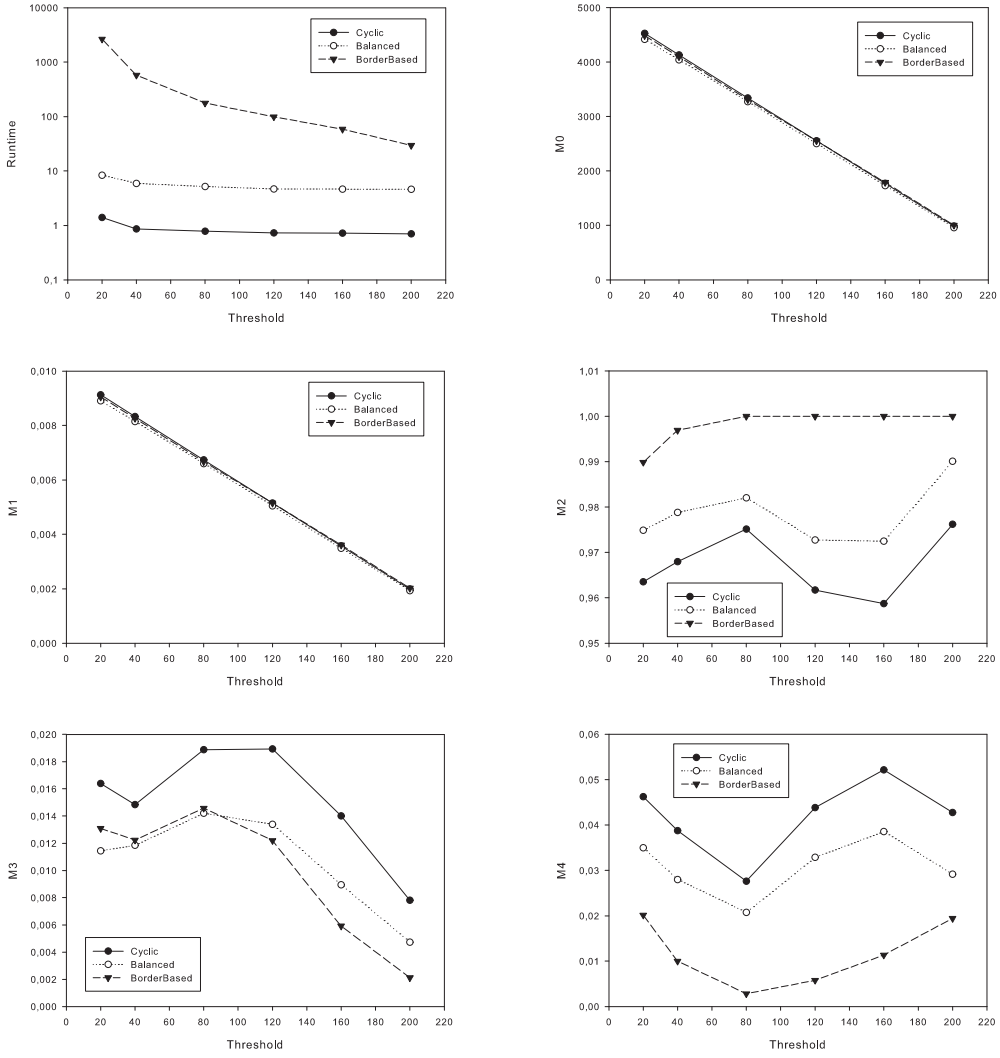
`T10I450K` veritabanı için ise benzer sonuçlar görülmektedir. Burada da CHA çalışma zamanı olarak oldukça iyi iken kalite metriklerinde kötü bir performans göstermektedir. BBHA ise aksi sonuçlar vermektedir. Dengeli Gizleme Algoritmasının hızlı ve etkili olduğu bu veritabanındaki performans sonuçlarından da görülebilmektedir.

`Retail` veritabanı için ise diğer iki veritabanındakine benzer sonuçlar elde edilmiştir.

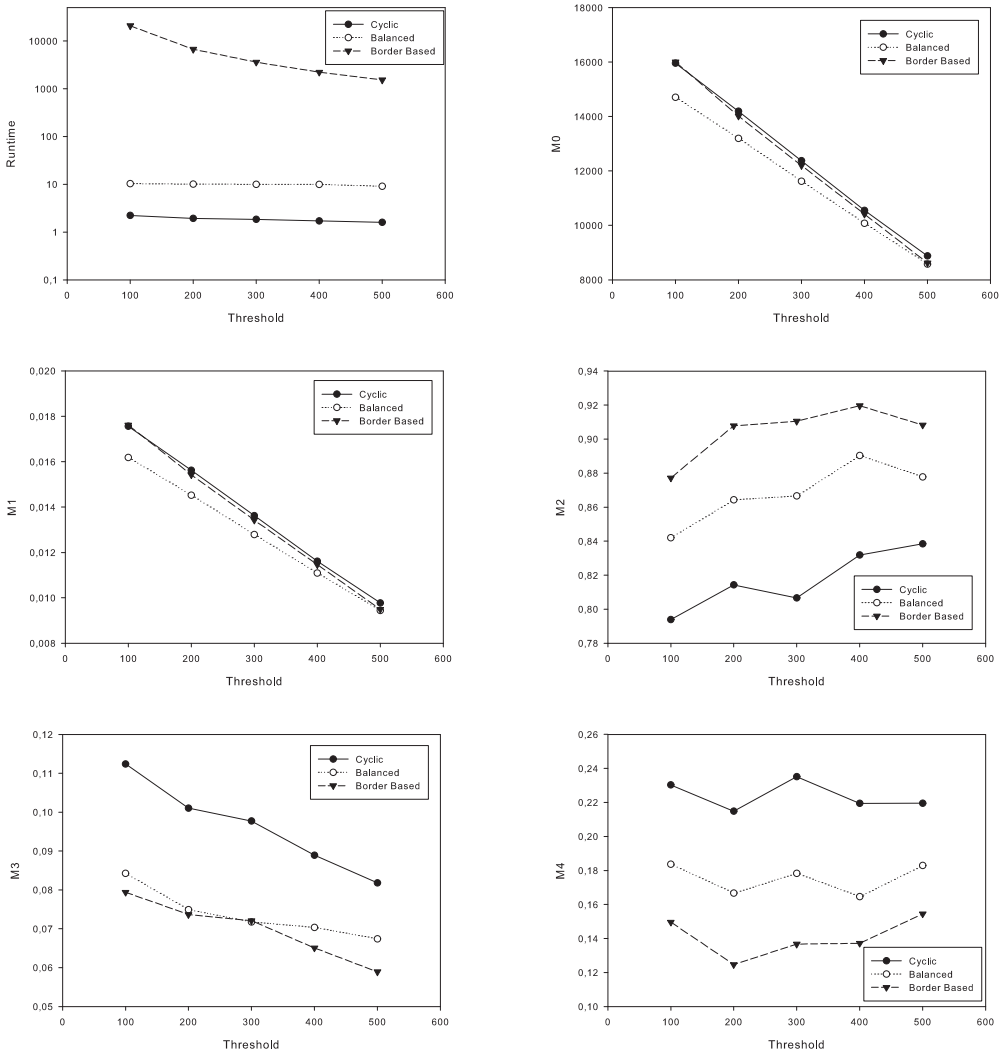
Sonuç olarak hızlı ve etkili bir gizleme için Dengeli Gizleme Algoritması çok uygun bir algoritma olmuştur.



Şekil 3.1: `T5I250K` veritabanı için performans sonuçları



Şekil 3.2: `T10I450K` veritabanı için performans sonuçları



Şekil 3.3: `Retail` veritabanı için performans sonuçları

4 AĞAÇ GİZLEME

Bu bölümde, ağaçlarla ilgili ön bilgiler verilmekte ve devamında da bilgi gizleme problemlerine konu olan ağaç örüntülerinin bir kısmı tanımlanmaktadır. Bir kısım ağaç örüntülerinden kasıt, literatürde, veri madenciliği tekniklerinin, geliştirilmiş uygulama ve araçlarla ağaçlar üzerinde uygulanmasıyla elde edilebilen ağaç örüntüleridir.

4.1 Tanımlamalar

Tanım 7 (Etiketli Çizge)**.** Bir etiketli ve yönlü G çizgesi $G = (V, E, \Sigma_V, \phi_V, \Sigma_E, \phi_E)$ şeklinde gösterilen bir 6 çok-ögelidir öyleki;

- V düğümler kümesidir,
- E ($E \subseteq V \times V$) kenarlar kümesidir,
- Σ_V düğüm etiketleri alfabesidir,
- ϕ_V ($V \rightarrow \Sigma_V$) düğüm etiketleri atama fonksiyonu,
- Σ_E kenar etiketleri alfabesi ve
- ϕ_E ($E \rightarrow \Sigma_E$) de kenar etiketleri atama fonksiyonudur.

Çizgeler bazen kenar etiketlerine sahip olmazlar. Bazen de kenar etiketlerinin varlığı önemsiz olur. Bu her iki durumda da çizgeler 4 çok-ögeli olurlar ve $G = (V, E, \Sigma_V, \phi_V)$ ile gösterilirler. Gösterim kolaylığı için çizgelerin $G = (V, E)$ şeklindeki gösterimi daha yaygındır.

Tanım 8 (Etiketli Köklü Ağaç)**.** Bir $T = (V, E)$ çizgesi veriliyor olsun. r diye bir düğüm kök olarak seçilsin. Eğer bu T çizgesi şu şartları sağlar ise T bu durumda bir köklü etiketli ağaç olur:

- T düğümleri arasında döngü içermez,

- T ağacının kökü $r \in V$ düğümüdür ve bu $r = root(T)$ ile gösterilir,
- Tüm $\forall v \in V$ düğümleri için kökten (r) v düğümlerine eşsiz tek bir yol bulunur.

Bir T ağacı için, $root(T)$ ile gösterilen ve seçilmiş bir $v \in V$ düğümü T ağacının kök düğümü olarak atanır. Herhangi iki $x, y \in V$ düğümü için, eğer $root(T)$'den başlayan ve y'de sonlanan bir yol varsa ve x de bu yol üzerinde yer alıyorsa, x düğümü y düğümünün *atasıdır*. Bununla beraber, y düğümü de x düğümünü *torunudur*. Eğer x ve y düğümleri $root(T)$'den y'ye olan yol üzerinde yer alan ardışık düğümler ise, x düğümü y düğümünün *ebeveyni*; y düğümü de x düğümünün *çocuğu* olur. $x, y \in V$ aynı ebeveynin çocukları ise *kardeş* olarak adlandırılırlar. Bu ilişkler şu notasyonlarla gösterilirler: $parent(v)$ $v \in V$'nin ebeveynini, $child(v)$ v'nin çocuklarını, $desc(v)$ v'nin torunlarını ve $ancs(v)$ de v'nin atalarını belirtir. $child(v)$, $desc(v)$ and $ancs(v)$ birden çok elemanlı düğümler kümesini ifade edebilirken, $parent(v)$ ise en çok tek elemanlı bir düğüm kümesini ifade eder. $v \in V$ düğümünü kök olarak kabul eden bir ağacı $T[v]$ notasyonu temsil eder. Bu kökü T'nin v düğümü olan bir ağaç demektir. Bir F *ormanı* ise m adet köklü ağaç içeren, $F = \{T_1, T_2, \ldots, T_m\}$ şeklinde ifade edilen *m* elemanlı bir küme olarak tanımlanır.

Ağaçlar genel olarak *kök-önce (pre-order)* ve *kök-sonra (post-order)* olarak gezilirler. Kök-önce gezintide herhangi bir düğüm için ilk önce düğümün kendisi, sonra da sırasıyla çocukları soldan sağa ziyaret edilir. Kök-sonra gezinti de ise herhangi bir düğüm için ilk önce sırasıyla soldan sağa çocukları ziyaret edilir sonra da düğümün kendisi ziyaret edilir. Bu iki gezintiden herhangi birisi için, $v \in V$ düğümünün, ağacın ilgili gezintisindeki sırasını verdiği de farz edilir. $post(v)$ ağacın kök-sonra gezintisinde v'nin sıra numarasını; $pre(v)$ ise ağacın kök-önce gezintisinde v'nin sıra numarasını gösterir. Örnek olarak Şekil 4.1'deki ağaçta *D* etiketli düğümün kök-önce gezintideki sıra numarası 4 iken, kök-sonra gezintideki sıra numarası ise 2'dir.

4.2 Ağaç Örüntüleri

Bir ağacın diğer bir ağacı içerip içermediği 4.4. bölümde tanıtılacak olan agaç gizleme probleminin esasını oluşturmaktadır. Bu bir P örüntü ağacının diğer bir T veri ağacı tarafından içerilip içerilmediğinin ya da T veri ağacının altağaçlarından biriyle eşlenip eşlenmediğinin tespit edilmesini gerektirir. Devamda, ağaç örüntülerinin veri madenciliğinde en fazla kullanılan iki sınıfı tanımlanmaktadır. Bunlar *birebir (induced)* ve *gömülü (embedded)* alt ağaç içerme sınıflarıdır.

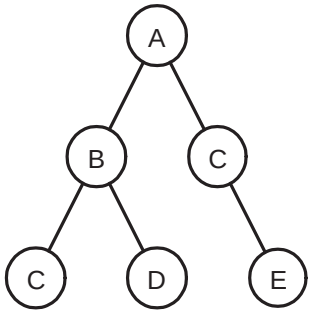
Tanım 9 (Birebir Altağaç [48])**.** $P = (W, F)$ ve $T = (V, E)$ ağaçları veriliyor olsun. Eğer (i) $\forall v \in W.\phi_W(v) = \phi_V(\varphi(v))$, (ii) $(u, v) \in F$ için ancak ve ancak $(\varphi(u), \varphi(v)) \in E$ ve $\phi_F(u, v) = \phi_E(\varphi(u), \varphi(v))$ koşullarını sağlayan birebir bir $\varphi : W \to V$ fonksiyonu varsa, P ağacı T'nin bir *birebir altağacıdır* ve bu ifade $P \preceq_i T$ ile gösterilir. φ belirteci hem kenar ve düğüm etiketlerinin aynı olmasını gerektirir, hem de düğümler arasındaki ebeveyn-çocuk ilişkisini korur. Yani aralarında ebeveyn-çocuk ilişkisi bulunan iki P örüntü ağacı düğümünün eşlendikleri T ağacı düğümleri arasında da ebeveyn-çocuk ilişkisi vardır.

Tanım 10 (Gömülü Altağaç [48])**.** $P = (W, F)$ ve $T = (V, E)$ ağaçları veriliyor olsun. (i) $\forall v \in W.\phi_W(v) = \phi_V(\varphi(v))$ ve (ii) $(u, v) \in F$ için ancak ve ancak T'de $root(T)$'den başlayan, $\varphi(u)$'dan geçen ve $\varphi(v)$'de sonlanan bir yol olmalıdır. Bu koşulları sağlayan birebir bir$\varphi : W \to V$ fonksiyonu varsa, P ağacı T'nin bir *gömülü altağacıdır* ve bu ifade $P \preceq_e T$ ile gösterilir. Burada φ belirteci hem kenar ve düğüm etiketlerinin aynı olmasını gerektirir hem de düğümler arasında ata-torun ilişkisini korur. Yani aralarında ata-torun ilişkisi bulunan iki P örüntü ağacı düğümünün eşlendikleri T ağacı düğümleri arasında da ata-torun ilişkisi vardır.

Çoğu zaman ağaç eşleme sınıfı (birebir ya da gömülü) söz konusu bağlamdan anlaşılmaktadır. Bu durumda ağaç eşleme $P \preceq T$ şeklinde gösterilir. Gözden kaçırılmaması gereken, hem birebir ağaç eşlemede hem de gömülü ağaç eşlemede, eşleme fonksiyonunun kardeş düğümler arasındaki sıraya dikkat edip etmediğidir. Bu durumda da ağaç eşlemenin düğüm eşleme sırasından kaynaklanan iki eşleme sınıfı daha söz konusu olmaktadır.

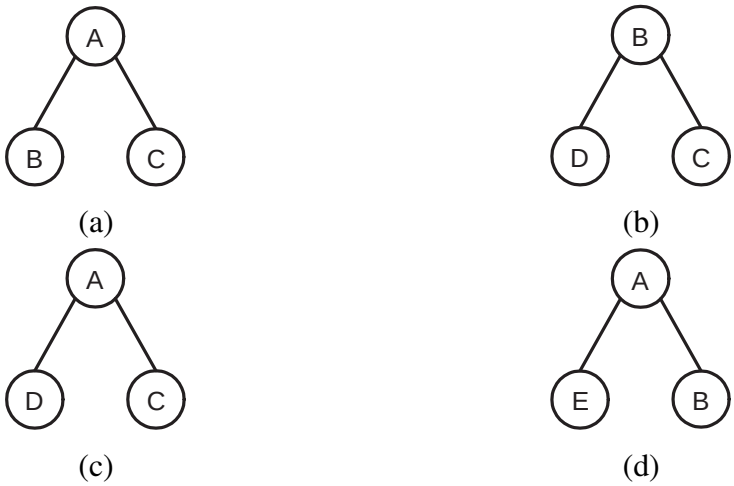
Tanım 11 (Sıralı ve Sırasız Altağaçlar)**.** $P = (W, F)$ ve $T = (V, E)$ ağaçları verilmiş ve Tanım 9'daki ya da Tanım 10'daki φ fonksiyonu sağlanmış olsun. Eğer φ fonksiyonu düğümleri eşlerken hem P hem de T ağaçlarının kök-önce gezintilerine riayet ediyorsa, φ aynı zamanda *sıralı* bir eşleme yapıyor demektir. Yani $u, v \in W$ düğümleri için $pre(u) < pre(v)$ ise $\varphi(u), \varphi(v) \in V$ için de $pre(\varphi(u)) < pre(\varphi(v))$ olmalıdır. Eğer φ eşleme yaparken ağaçların kök-önce gezintilerine riayet etmiyorsa yapılan eşleme *sırasız* bir eşlemedir. Böylece, eşleme sıralı ise P ağacı T ağacının sıralı alt ağacı, eşleme sırasız ise P de T'nin sırasız altağacı olur. Açıkçası, her sıralı alt ağaç aynı zamanda sırasız bir altağaç olduğundan sırasız altağaçlar sıralı altağaçların daha genel bir halidir.

Tanım 9'daki birebir eşleme ile Tanım 10'daki gömülü eşleme, Tanım 11'deki sıralı ve sırasız eşlemeler beraber düşünüldüğünde örüntü altağaçların 4 farklı sınıftan biri şeklinde olduğu ortaya çıkar. Bunlar *birebir-sıralı (induced-ordered)*, *birebir-sırasız (induced-unordered)*, *gömülü-sıralı (embedded-ordered)* and *gömülü-sırasız (embedded-unordered)* altağaç sınıflarıdır. Altağaç sınıfı, ağaç eşleme sınıfı, ağaç içerme sınıfı aynı kavramı ifade eden kelimeler olarak birbirlerinin yerlerine kullanılmaktadır. Şekil 4.1'de sadece düğüm etiketli olan bir ağaç görülmektedir. Bu ağacın, 4 farklı altağaç sınıfına ait altağaçları ise Şekil 4.2'de gösterilmiştir. Bir T ağacının birçok altağacı olabilir fakat $root(T)$'nin çocuklarının kök olduğu altağaçlar diğer altağaçlardan önem bakımından farklılaşırlar. Bu altağaçlara *anlık altağaçlar (immediate subtrees)* denir.



Şekil 4.1: Örnek ağaç

Tanım 12 (Örüntü Ağacı Destek Değeri)**.** Verilen $T = (V, E)$ veri ağacı ve $P = (W, F)$ örüntü ağacı için $P \preceq T$ gösterimi geçerliyse T'ye P'yi destekliyor denir.



Şekil 4.2: Şekil 4.1'de verilen ağaç için 4 farklı altağaç sınıfı: (a) birebir-sıralı, (b) birebir-sırasız, (c) gömülü-sıralı ve (d) gömülü-sırasız

Ağaç eşleme sınıfı biliniyor olsun. Bu durumda P örüntü ağacının bir $\mathcal{D}$ veritabanındaki destek değeri P'yi bilinen eşleme sınıfına göre altağaç olarak içeren ağaçların sayısı kadardır, yani $sup_{\mathcal{D}}(P) = |\{T \in \mathcal{D} : P \preceq T\}|$. Bir T veri ağacı P örüntü ağacını birçok şekilde ve sayıda destekleyebileceğinden (P'nin T'nin birden fazla altağacıyla eşlenmesi durumu), tüm bu farklı desteklemelerin sayısı $c_T(P)$ ile gösterilir. T veri ağacı P örüntü ağacını en fazla düğüm sayısı kadar içerebileceğinden, $0 \leq c_T(P) \leq |T|$ dir. Bundan hareketle ağırlıklı destek değeri kavramından bahsedilebilir. Buna göre verilen bir örüntü ağaçları kümesi $\mathcal{P}$ için destek ve ağırlıklı destek değerleri sırasıyla şu şekillerde gösterilir ve hesaplanır: $sup_{\mathcal{D}}(\mathcal{P}) = |\bigcup_{P \in \mathcal{P}} \{T \in \mathcal{D} : P \preceq T\}|$ ve $wsup_{\mathcal{D}}(\mathcal{P}) = \sum_{P \in \mathcal{P}} wsup_{\mathcal{D}}(P)$. Bu destek değeri mutlak destek değeridir. Ağaç destek değeri bazen de göreceli olarak tanımlanabilir. Göreceli destek değeri tamsayı olmayan ve destekleyen ağaç sayılarının tüm ağaçların sayısına oranını belirtir ve toplam ağaç sayısıyla çarpılarak mutlak destek değerine dönüştürülür.

Destek değeri hesaplanması için, örüntü ağacının belirlenmiş olan eşleme sınıfına göre hangi ağaçlarda içerildiğinin bulunması ve bu ağaçların sayılması gerekir. Ağırlıklı destek değerinin hesaplanması için ise her ağacın örüntü ağacını kaç kere içerdiğinin bulunması ve tüm ağaçlardaki içerilme sayılarının toplanması gerekir.

4.3 Altağaç Eşleme

$P = (W, F)$ ve $T = (V, E)$ veriliyor olsun. P'nin T ile eşlenebilmesi bir diğer ifadeyle T'nin P'yi destekleyebilmesi için, T'nin $T[v]$ ile gösterilen ve P ile eşleşen bir altağacı olmalıdır.

P ile $T[v]$'yi eşleme kuralı birebir-sıralı eşleme sınıfı için özyineli olarak aşağıdaki gibi tanımlanmıştır. Diğer eşleme sınıfları için de eşleme kuralları benzer şekildedir.

Tanım 13 (Birebir-Sıralı Altağaç Eşleme)**.** Verilen $P = (W, F)$ örüntü ağacı şu şartları sağlıyorsa, $T = (V, E)$ veri ağacının $T[v]$ altağacına eşlenebiliyor demektir:

- $|W| = 0$, yani P boş bir ağaç ise,
- $|W| > 0$, yani P boş bir ağaç değilse ve devamdaki iki şart sağlanıyorsa,
  - $\varphi_W(root(P)) = \varphi_V(root(T[v]))$ ve
  - $(c_1, c_2, \ldots, c_k)$, P'nin k adet çocuğunun sıralı bir dizisi olsun. $(c_{n1}, c_{n2}, \ldots, c_{nk})$ de $root(T[v]$'nin k adet seçilmiş çocuğunun dizisi olsun. $\forall i = 1, 2 \ldots k$: (i) $\varphi_F(root(P), c_i) = \varphi_E(root(T[v]), c_{ni})$ olmalı, ve (ii) $P[c_i]$ de $T[c_{ni}]$ ile eşlenebilmelidir.

Ağaç eşlemede en sıradan durum boş bir örüntü ağacının olabilecek tüm veri ağaçları ile eşlenmesi durumudur. Boş olmayan örüntü ağaçları için, ilk önce köklerin etiketleri aynı olmalıdır, sonra örüntü ağacının kökünün anlık altağaçları verilen eşleme sınıfına göre veri ağacının bazı altağaçlarıyla eşlenmelidir, en son olarak da varsa karşılıklı kenar etiketleri de eşlenmelidir. Tanım 13'den hareketle hem veri ağacının altağaçları hem de örüntü ağacının altağaçları üzerinde uygulanan ve eşleme olup olmadığını yoklayan dinamik programlama tabanlı bir algoritma geliştirilmiştir. Bu algoritmada, hesaplama etkinliği açısından en büyük zorluk veri ağacının çocuklarının k boyutundaki kombinasyonlarının üretilmesidir. Bu kombinasyonların üretilmesi en kötü durum olarak üsseldir. Örnek olarak kardeş olan 4 örüntü ağacı düğümünün yine kardeş olan

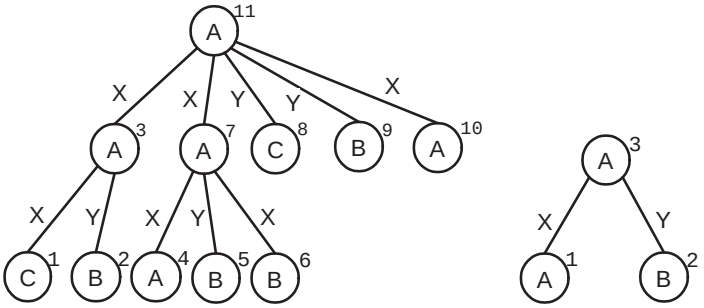
8 veri ağacı düğümüyle eşlenip eşlenmediğinin bulunması için 70 farklı muhtemel eşleme durumun bulunması ve bunlardan hangilerinin geçerli olup olmadıklarının tespit edilmesi gerekir. Her veri ağacı düğümü için bu hesaplamanın yapılması gerektiği düşünülürse hesaplama maliyetinin oldukça fazla olduğu görülebilir.

Önerilen dinamik programlamalı algoritma ile sadece eşleme olup olmadığı değil, farklı eşleme biçimlerinin sayılması da mümkündür. Buradaki farklı eşleme biçimlerinden kasıt ağırlıklı destek değeri hesaplanmasındakinden farklıdır. Ağırlıklı destek değeri hesaplanmasında veri ağacı düğümlerinden kaçının örüntü ağacının köküyle eşlenebildiği farklı eşleme sayılarını oluşturur. Buradaki farklı eşleme sayısı ise bir veri ağacı düğümünde var olan eşlemenin eşlenen düğümlerin altağaçlarının farklı farklı eşlenmesiyle oluşmaktadır. Dolayısıyla bir ağaçtaki farklı eşleme sayılarıyla, bir düğümdeki eşleme sayısı aynı sayıyı ifade etmemektedirler. Bu amaçla her düğümdeki farklı eşleme sayısını saklayan M eşleme sayacı tablosu kullanılmıştır.

n düğümlü bir T veri ağacı ve m düğümlü bir P örüntü ağacı verilmiş olsun. Buna göre M eşleme sayacı tablosu n satırdan ve m sütundan oluşan ve negatif olmayan tamsayılar içeren bir tablo olarak tasarlanır. Tablodaki her satır indisi veri ağacının kök-sonra gezilmesindeki düğümlerin sıra numarasına karşılık gelir. Her sütun indisi ise örüntü ağacının kök-sonra gezilmesindeki düğümlerin sıra numarasına karşılık gelir. Şekil 4.3'de örnek bir veri ağacı ve örüntü ağacı verilmiştir. Bu şekildeki ağaçların düğümlerinin yanındaki numaralar düğümlerin kök-sonra gezilmesindeki düğümlerin sıra numarasını göstermektedir. M eşleme sayacı tablosundaki her $M(i, j)$ hücresi P'nin j kök-sonra numaralı düğümünü kök olarak alan alt ağacın kaç farklı şekilde T'nin i kök-sonra numaralı düğümünü kök olarak alan altağaçla kaç farklı şekilde eşlendiğini gösterir. cc verilen bir düğümün çocuk sayısını veren bir fonksiyon olsun. $Comb(i, cc(j))$ de T'nin i kök-sonra numaralı düğümün çocuklarının $|cc(j)|$ boyutundaki muhtemel eşleme kombinasyonlarını veriyor olsun. Ek olarak $MComb(i, cc(j)) \subseteq Comb(i, cc(j))$ de $Comb(i, cc(j))$'deki geçerli eşleme kombinasyonlarının sayısını veren fonksiyon olsun. Buna verilenlere göre M eşleme tablosunun hücreleri şu şekilde doldurulur:

$$M(i,j) = \begin{cases} 0 & j \text{ düğümü yaprak ve } i \text{ ile } j \text{ farklı etiketlere sahipse,} \\ 1 & j \text{ düğümü yaprak ve } i \text{ ile } j \text{ aynı etiketlere sahipse,} \\ 0 & j \text{ düğümü yaprak değilse fakat } i \text{ düğümü yapraksa,} \\ 0 & i \text{ ve } j \text{ düğümleri yaprak değillerse ve } |cc(i)| < |cc(j)| \text{ ise,} \\ X & i \text{ and } j \text{ düğümleri yaprak değillerse ve } |cc(i)| \geq |cc(j)| \text{ ise} \end{cases} \tag{4.1}$$

Formüldeki $X = \sum_{p \in MComb(i,cc(j))} \prod_{l \in cc(j)} M(p(l), l)$ ve $p(l)$ de p kombinasyonunda veri ağacında l'ye karşılık gelen düğümdür.



Şekil 4.3: Veri ve örüntü ağaçları: (a) bir veri ağacı, (b) bir örüntü ağacı.

M eşleme sayacı tablosunun doldurulmasından sonra, son sütunda 0 olmayan değerler örüntü ağacının ilgili satıra karşılık gelen veri ağacını düğümünü kök olarak kabul eden altağaçla kaç farklı şekilde eşlenebildiğini göstermektedir. Çizelge 4.1(a)'da, Şekil 4.3'daki veri ve örüntü ağaçları için birebir-sıralı eşeleme sınıfına göre oluşturulmuş olan M eşleme sayacı tablosu görülmektedir. Tablodan görülebileceği gibi 7 nolu düğümde sadece bir tane eşleme, 11 nolu düğümde ise 2 farklı eşleme vardır.

Bazı uygulamalardaki ağaçlarda kenarlar etiketlerden yoksun olabilir veya tamamen ihmal edilebilir. Çizelge 4.1(b)'de M eşleme sayacı tablosunun kenar düğümleri dikkate alınmaksızın oluşturulmuş şeklini göstermektedir. Bu durumda da hem 7. hem de 11. düğümlerde ikişer eşleme oluşmaktadır.

Çizelge 4.1: Eşleme tabloları: (a) kenar etiketli birebir-sıralı altağaç için M tablosu, (b) kenar etiketsiz birebir-sıralı altağaç için M tablosu.

(a)

	1	2	3
1	0	0	0
2	0	1	0
3	1	0	0
4	1	0	0
5	0	1	0
6	0	1	0
7	1	0	1
8	0	0	0
9	0	1	0
10	1	0	0
11	1	0	2

(b)

	1	2	3
1	0	0	0
2	0	1	0
3	1	0	0
4	1	0	0
5	0	1	0
6	0	1	0
7	1	0	2
8	0	0	0
9	0	1	0
10	1	0	0
11	1	0	2

4.3.1 Diğer İçerme Sınıflarında Altağaç Eşleme

Birebir-sırasız ağaç eşleme tanımı için Tanım 13'deki sıra kısıtının ortadan kaldırılması gerekir. Bu ise kombinasyon hesabı yerine permütasyon hesabının yapılmasını gerektirir. Permütasyon hesaplaması kombinasyon hesaplamasından daha maliyetli olduğundan birebir-sırasız ağaç eşleme birebir-sıralı ağaç eşlemeden çalışma zamanı olarak daha kötüdür. Bunun yanında birebir-sırasız eğleme için Tanım 13'deki diğer koşullar aynen geçerlidir. Aynı dinamik programlama mantığı ile birebir-sırasız eşleme bu şekilde mümkündür. Çizelge 4.2(a)'da kenar etiketlerinin önemli olduğu eşleme sayacı tablosu, Çizelge 4.2(b)'de de kenar etiketleri göz ardı edilerek oluşturulan eşleme sayacı tablosu görülmektedir.

Gömülü-sıralı ve gömülü-sırasız eşleme sınıfları için, eşleme tabloları Tanım 13 kısmen değiştirilerek elde edilebilir. Gömülü-sıralı sınıfı için Tanım 13'deki koşullar sonuncusu haricinde aynen korunmalıdır. Son koşulun ise şu şekilde değiştirilmesi gerekir:

- $k \geq 1$ olmak üzere P ağacı k tane çocuğa sahip olsun. P'nin çocuklarının sıralı dizisi $(c_1, c_2, \ldots, c_k)$ olsun. $root(T[v])$'nin öyle bir torunlar altdizisi (kök-önce

Çizelge 4.2: Eşleme tabloları: (a) kenar etiketli birebir-sırasız altağaç için M tablosu, (b) kenar etiketsiz birebir-sırasız altağaç için M tablosu.

(a)

	1	2	3
1	0	0	0
2	0	1	0
3	1	0	0
4	1	0	0
5	0	1	0
6	0	1	0
7	1	0	1
8	0	0	0
9	0	1	0
10	1	0	0
11	1	0	3

(b)

	1	2	3
1	0	0	0
2	0	1	0
3	1	0	0
4	1	0	0
5	0	1	0
6	0	1	0
7	1	0	2
8	0	0	0
9	0	1	0
10	1	0	0
11	1	0	3

gezinti numaralarına göre) $(c_{n1}, c_{n2}, \ldots, c_{nk})$ olmalıdır ki, $\forall i = 1, 2 \ldots k : P[c_i]$ ile $T[c_{ni}]$ eşlenmelidir.

Açıkça, bu koşuldaki sıra kısıtı da ortadan kaldırılırsa gömülü-sırasız sınıf için eşleme tanımı da elde edilir. Her iki gömülü sınıf için yukarıdaki sayma işlemi kullanılarak oluşturulmuş eşleme tablosu Çizelge 4.3'de gösterilmiştir. Dikkat edileceği gibi, gömülü sınıf birebir sınıfa göre daha fazla kombinasyon hesaplaması gerektirmektedir bu yüzden daha maliyetlidir.

4.4 Ağaç Gizleme Problemi

Bölüm 4.3'de altağaç eşleme sınıfları tanımlanmıştır. Bu bölümde ise, ağaç gizleme problemi tanımlanmaktadır. Ağaç gizleme problemi en yalın ifadesiyle var olan eşlemeleri yok ederek örüntü ağaçlarının veri ağaçlarındaki varlığını ortadan kaldırmayı amaçlamaktadır. Problemin formal tanımı şu şekildedir:

Tanım 14 (Ağaç Gizleme Problemi)**.** Bir $\mathcal{D}$ veri ağaçları veritabanı (orman), bir $\mathcal{P}_h$ hassas örüntü ağaçları kümesi ve bir ψ eşik değeri verilmiş olsun. $\mathcal{D}$ veritabanı $\mathcal{D}'$ veritabanına öyle dönüştürülmelidir ki, dönüştürme sonucunda şu şartlar sağlanmalıdır:

Çizelge 4.3: Eşleme tabloları: (a) kenar etiketsiz gömülü-sırasız altağaç için M tablosu, (b) kenar etiketsiz gömülü-sıralı altağaç için M tablosu.

(a)

	1	2	3
1	0	0	0
2	0	1	0
3	1	1	0
4	1	0	0
5	0	1	0
6	0	1	0
7	2	2	2
8	0	0	0
9	0	1	0
10	1	0	0
11	5	4	13

(b)

	1	2	3
1	0	0	0
2	0	1	0
3	1	1	0
4	1	0	0
5	0	1	0
6	0	1	0
7	2	2	2
8	0	0	0
9	0	1	0
10	1	0	0
11	5	4	7

- $\forall P \in \mathcal{P}_h . sup_{\mathcal{D}}(P) < \psi$ olmalıdır ve
- $\mathcal{D}$ ile $\mathcal{D}'$ olabildiğince aynı olmalıdır.

Öge kümesi gizleme problemine benzer olarak, tanımın ilk koşulu dönüştürme işleminden sonra, $\mathcal{D}'$ üzerinde uygulanacak sık altağaç madenciliği sonuç verileri arasında hassas örüntü ağaçlarının yer almamasını garanti eder. İkinci koşul ise dönüştürülmüş veritabanı ile orijinal halinin mümkün olduğunca aynı kalmasını, yani bütünlüğünün korunmasını gerektirir. Dönüştürme işlemi sterilize etmek olarak de kullanılabilir.

Tanım 14, ağaç gizleme probleminin genel ifadesidir. Belirginleştirmek gerekirse, dönüştürme işleci ve $\mathcal{D}$ ile $\mathcal{D}'$ arasında bütünlük, benzerlik ölçüleri de tanımlanmalıdır. Tez kapsamında yapılan çalışmalarda, ağaçlarda sahte örüntüler oluşmaması için düğümlerin etiketlerini, düğüm etiketlerinin ait olduğu alfabeden olmayan sembollerle değiştirmek dönüştürme işleci olarak seçilmiştir. Düğüm etiketleri alfabesi Σ_V, kenar etiketleri alfabesi Σ_E iken, dönüştürme işleci sembolü ise $\Delta \notin \Sigma_V \bigcup \Sigma_E$ olarak seçilmiştir. Bu şekildeki bir işlece etiket maskeleme de denir. Böylece dönüştürülmüş $\mathcal{D}'$ veritabanı bir yönüyle $\mathcal{D}$'nin bir altkümesidir. Benzerlik ölçüsü ise maskelenen düğüm sayısı ya da $\mathcal{D}'$'deki Δ sembolünün sayısıdır. Bu sayının az olması veritabanı bütünlüğünün korunduğunu gösterir.

Teorem 1. Ağaç gizleme problemi NP-Harddır. **İspat**. Ağaçlar her düğümün en fazla bir çocuk içerdiği durumda dizgi şekline gelirler. Abul [2], aynı dönüştürme işleci ve benzerlik ölçüsüne göre dizgilerde gizleme probleminin NP-hard bir problem olduğunu göstermiştir. Kısıtlı problemin NP-Hard oluşu problemin orijinalinin de NP-Hard olduğu anlamına gelir [18].

Tanım 14'deki ağaç gizleme problemi şu iki altprobleme ayrılarak incelenebilir. Çünkü üzerinde gizleme yapılabilecek birçok ağaç olabilir. Bu ağaçlardan hangilerinin seçilmesi gerektiği global ölçekli çözümün problemine karşılık gelir. Global ölçekli çözüm ile gizleme yapılmak üzere seçilen bir ağaç içinde birden sayıda eşleme olabilir. Bu durumda ise seçilen bu ağaçtaki eşlemeleri ortadan kaldıracak şekilde hangi düğümün seçilmesine karar verilmelidir. Bu ise yerel ölçekli çözümün problemine karşılık gelir.

4.4.1 Global Ölçekli Çözüm

Problem tanımı dönüştürülmüş veritabanının örüntü ağaçlarını $\psi - 1$ den fazla destekleyen veri ağacını bulundurmamayı zorunlu kılar. Yani örüntü ağaçlarının her birinin dönüştürülmüş veritabanındaki destek değerleri en fazla $\psi - 1$ kadar olmalıdır. Dolayısıyla her P hassas örüntü ağacı için eğer $sup_{\mathcal{D}}(P) \geq \psi$ ise veritabanından $sup_{\mathcal{D}}(P) - \psi + 1$ kadar veri ağacı seçilmeli ve bu ağaçlarda gizleme işlemi yapılmalıdır. $sup_{\mathcal{D}}(P) - \psi + 1$ kadar veri ağacının tüm ağaçlar arasından belirlenip seçilmesi global ölçekli çözüme karşılık gelir. Seçilecek ağaçlar, öyle seçilmelidir ki problem tanımının ikinci gereğini sağlamalıdır ve veritabanı bütünlüğü korunmuş olmalıdır.

Global çözüm için en sıradan durum, herhangi bir P örüntü ağacının verilen eşik değerinden daha küçük destek değerine sahip olduğu durumdur ve bu durum herhangi bir sterilize etme işlemi gerektirmez. Sıradan olmayan durumlar ise P'nin eşik değerinden daha küçük destek değerine sahip olmadığı durumlardır. Bu durumda P'yi destekleyen $sup_{\mathcal{D}}(P)$ kadar ağaç arasından $sup_{\mathcal{D}}(P) - \psi + 1$ kadarı gizleme yapılmak için seçilmelidir. Seçim işleminin en naif (*naive*) yöntemi ağaçların rastgele seçilmesidir. Fakat bu durum her ne kadar hızlı olsa da etkili bir yöntem değildir, çünkü problem tanımının

ikinci gereği olan veritabanı bütünlüğünü korunmasına riayet ettiği pek de söylenemez. Bunun için $sup_{\mathcal{D}}(P) - \psi + 1$ kadar veri ağacının bu şartı sağlayacak şekilde seçilmesi gerekir. Bu tezde önerilen yöntem şu şekildedir: Üzerinde gizleme yapılacak ağaç öyle seçilmelidir ki, yapılacak bir düğüm maskeleme diğer bazı hassas öge kümelerinin de gizlenme ihtimalini doğurmalıdır.

4.4.2 Yerel Ölçekli Çözüm

Global ölçekli çözümle seçilmiş bir ağaçta var olan gizlemenin yok edilmesi için örüntü ağaçı düğümlerinden birisinin maskelenmesi gerekir. Her ne kadar bu çözüm uygun bir çözüm olsa da eşlemenin bozulduğunu garanti etmez. Çünkü maskelenen düğümün aynı etiketli ve mevcut eşlemede herhangi bir örüntü ağacı düğümüyle eşlenmemiş bir kardeşi varsa, maskelemeden sonra bu kardeş düğüm maskelenen düğümün yerine eşlemeye katkıda bulunacaktır. Bu durum aynı düğümde farklı birçok eşlemenin olduğu durumdur. Dolayısıyla silinecek düğüm öyle seçilmelidir ki, bu düğümün maskelenmesiyle o düğümde olabilecek tüm farklı eşlemeler ortadan kalkmalıdır. Bu ise seçilecek düğümün örüntü ağacının kökünün eşlendiği düğüm olmasını gerektirir. Çünkü örüntü ağacının kökünün eşlendiği düğüm maskelenince, o düğümde olabilecek tüm eşlemeler böylece ortadan kaldırılmış olur. Bu nedenle örüntü ağacının köküne karşılık gelen düğümün maskelenmesi yerel ölçekli çözümün bir parçası olarak seçilmiştir.

Global ölçekli çözümde seçilen veri ağaçları örüntü ağaçlarını birçok sayıda ve şekilde destekleyebilir (örüntü ağacının kökü birden çok veri ağacı düğümüyle eşlenebilir). Yani örüntü ağacının bu ağaçlardaki ağırlıklı destek değerleri 1'den büyük olabilir. Bu ağaçlarda gizleme yapılabilmesi için var olan tüm eşlemelerin ortadan kaldırılması gerekir.

Yerel ölçekli çözümde akla gelen ilk naif yöntem eşlemelerde örüntü ağacının köküne karşılık gelen veri ağacı düğümlerinin tamamının maskelenmesidir. Bu şekilde veri ağacının örüntü ağacını desteklememesi sağlanır. Her ne kadar naif yöntem umulduğu

gibi çalışsa da, kendi içerisinde etkililik açısından bir sorun içermektedir. Bu sorun ise var olan çoğu eşlemelerin çakışabildiği düşünülürse, çok sayıda maskelemenin yapılacak olmasıdır. Bu durumun açık bir örneği Şekil 4.3'de birebir-sırasız eşleme sınıfı için görülmektedir. Görülebileceği gibi örüntü ağacı veri ağacının 7 ve 11 numaralı düğümleriyle eşlenmektedir. Ayrıca 7 numaralı düğüm 11 numaralı düğümdeki eşlemeye de altağaç olarak katkıda bulunmaktadır. Naif yöntemde bu veri ağacında gizleme yapılabilmesi için 7 ve 11 numaralı her iki düğümün de maskelenmesi gerekir. Fakat sadece 7 numaralı düğümün maskelenmesiyle 11 numaralı düğümdeki eşlemenin de bozulacağı ve bu ağaçta gizleme işleminin başarılmış olabileceği açıktır. Dolayısıyla naif yöntemin pek de etkili bir yöntem olmadığı açıktır. Bunun yerine en az maskelemeyle en çok eşlemenin ortadan kaldırılması yerel ölçekli çözüm için daha uygundur. Bu ise Bölüm 4.5'de anlatılmış ve algoritmada kullanılmıştır.

Ağaç gizleme problemi NP-Hard bir problem olmasına rağmen hızlı ve etkili sezgiseller mevcuttur. Devamda yerel ölçekli çözümü naif çözümden daha etkili olan alttan-üste artırımlı çalışan bir algoritma yer almaktadır. Bu algoritma, global ölçekte daha etkin çalışmak için ağırlıklı destek değerlerinden faydalanmaktadır. Var olan birçok uygulamada ağaçlar kenar etiketsiz olduklarından ya da kenar etiketleri önemsiz olduğundan geliştirilen algoritma da sadece düğüm etiketli ağaç gizlemek için tasarlanmıştır.

4.5 Alttan-Üste Ağaç Gizleme

Ağaç gizleme algoritmasının sözde kodu Algoritma 4'de verilmiştir. Algoritma 4 girdiye ihtiyaç duymaktadır: (i) veri ağaçları veritabanı $\mathcal{D}$, (ii) hassas örüntü ağaçları kümesi $\mathcal{P}_h$, (iii) eşik değeri ψ ve (iv) eşleme sınıfı (birebir-sıralı, birebir-sırasız, gömülü-sıralı, gömülü-sırasız) MC. Algoritma çıktı olarak dönüştürülmüş veritabanını vermektedir.

Algoritmadaki global ölçekli çözüm sezgiseli veri ağaçlarının ve örüntü ağaçlarının sıralanmasını gerektirmektedir (2. ile 5. satırlar arası). Her veri ağacı için kaç farklı

örüntüyle eşlendiği hesaplanmakta (2. satır) ve veri ağaçları bu değerin artan sırasına göre sıralanmaktadırlar (3. satır). Bu sıralama 7. satırda dikkate alınmaktadır. Böylece sezgisel az sayıda eşleme içeren veri ağaçlarının daha az maskelenmesi önsezisi üzerinde oluşmaktadır. Sezgisel ayrıca örüntü ağaçlarını da ağırlıklı destek değerlerine göre sıralamakta (6. satır) ve ilk önce en az en az ağırlıklı destek değerine sahip olanı saklamaktadır. Bu sezgiselin benzeri öge kümeleri ve dizgi gizleme için de kullanılmıştır [2] ve etkili bir sezgiseldir.

Algoritma 4 Alttan-Üste Ağaç Gizleme Algoritması

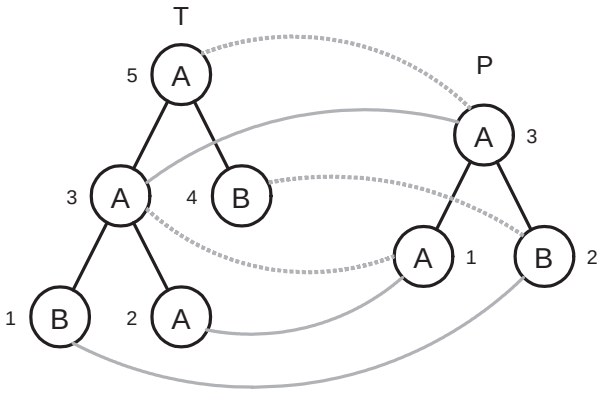
Girdi: $\mathcal{D}$ veri ağaçları veritabanı, $\mathcal{P}_h$ hassas örüntü ağaçları kümesi, ψ eşik değeri, MC eşleme sınıfı

Çıktı: $\mathcal{D}'$ sterilize edilmiş veri ağaçları veritabanı

1: $\mathcal{D}' \leftarrow \mathcal{D}$
2: $c_T(\mathcal{P}_h) = \Sigma_{P \in \mathcal{P}_h} c_T(P)$' yi hesapla
3: $\mathcal{D}'$ yi $c_T(\mathcal{P}_h)$'in artan sırasına göre sırala
4: $wsup_{\mathcal{D}'}(P), \forall P \in \mathcal{P}_h$'yi hesapla
5: $\mathcal{P}_h$'yi $wsup_{\mathcal{D}'}(\mathcal{P}_h)$'nin artan sırasına göre sırala
6: **for all** $P \in \mathcal{P}_h$ **do**
7: **while** $sup_{\mathcal{D}'}(P) \geq \psi$ **do**
8: $P \preceq T'$ 'yi sağlayan sıradaki $T' \in \mathcal{D}'$'yi MC ye göre bul
9: $T' \leftarrow LocalSanitize(T', P)$
10: $sup_{\mathcal{D}'}(P) \leftarrow sup_{\mathcal{D}'}(P) - 1$
11: **end while**
12: **end for**

9. satır yerel ölçekli çözümü içermekte ve T' veri ağacı ile P örüntü ağacı için $P \preceq T'$ ifadesinin geçerli olmadığını garanti etmektedir. *LocalSanitize* prosedürü yerel ölçekli sezgiseli alttan-üste kök-sonra gezintiyi esas alarak gerçekleştirmektedir. Prosedür T''nü kök-sonra gezmekte ve ziyaret ettiği $v \in T'$ düğümünde verilen eşleme sınıfına uygun olan bir eşeleme olup olmadığını kontrol etmektedir. Eğer bir eşleme varsa v düğümünü Δ ile maskelemektedir. Alttan-üste olan gezinti esnasında yapılan bu maskeleme, maskelenen düğümün üst kısımlarda oluşabilecek eşlemelere olan katkısını ortadan kaldırmaktadır. Yani sırada ziyaret edilecek tüm v'lerde başlayan ve maskelenen düğümü de içeren $T'[v]$'ler olmayacaktır.

Şekil 4.4'de örnek bir veri ağacı ve örüntü ağacı görülmektedir. Düğüm numaraları kök-sonra gezintiye göre atanmıştır. MC eşleme sınıfının birebir-sırasız olduğu düşü-



Şekil 4.4: Örnek bir veri ağacı ile gizlenecek olan örüntü ağacı

nülürse örüntü ağacı iki farklı şekilde veri ağacıyla eşlenmektedir ve 3 ile 5. veri ağacı düğümlerinde eşlemeler başlamaktadır. Noktalı çizgiler 5 numaralı düğümde başlayan eşlemeyi, düz çizgiler ise 3 numaralı düğümde başlayan eşlemeyi göstermektedir. Dikkat edilirse 3 numaralı düğümden hem düz çizgi hem de noktalı çizgi çıkmaktadır. Bu 3 numaralı düğümün her iki eşlemede de bulunduğunu göstermektedir. Seçilen yerel ölçekli çözüm gereği ilk önce 3 numaralı düğümün etiketi Δ ile değiştirilerek maskelenmektedir. Bu düğümün maskelenmesiyle de alttan-üste gezintide ilk durumda 5 numaralı düğümde var olan eşleme artık olmayacaktır. Dolayısıyla tek maskeleme ile 2 eşleme birden ortadan kaldırılmıştır. Naif metodun bu iki düğümü de maskeleyeceği düşünülürse, seçilen yerel ölçekli çözümün naif çözümden veritabanının bütünlüğünü koruması açısından daha etkili olduğu açıktır. Benzer bir durum Şekil 4.3'de de vardır.

Algoritma veri ağaçları düğümlerini gezerken ağaç içerme kontrolü yaptığından, ağaç içerme kontrolü de Bölüm 4.3'de anlatıldığı gibi üssel olduğundan, algoritmanın hızlı çalışması ağaç içerme probleminin hızlı bir şekilde çözülmesine doğrudan bağlıdır. Dolayısıyla daha hızlı ağaç içerme çözümlerine ihtiyaç vardır.

4.6 Etkili Altağaç Eşleme

Bu bölümde dört ayrı eşleme sınıfı ayrı ayrı incelenmektedir. Literatürde Kilpelainen [22] tarafından dört ayrı eşleme sınıfı için de algoritmalar önerilmiştir. Önerilen bu

algoritmalar sırasıyla tanıtılmaktadır. Bu algoritmalar Algoritma 4'de yerel ölçekli çözümün daha hızlı olması için kullanılmıştır.

4.6.1 Birebir-Sırasız Altağaç Eşleme

P bir örüntü ağacı ve $\langle P[1], P[2], \ldots, P[k]\rangle$ de P'nin çocuklarını kök kabul eden anlık altağaçlar olsun. Yine benzer biçimde T de bir veri ağacı ve $\langle T[1], T[2], \ldots, T[l]\rangle$ de T'nin çocuklarını kök kabul eden anlık altağaçlar olsun. Kök düğümlerin aynı etiketleri taşıdıkları varsayılsın. Bu durumda bir eşleme olması için $k \leq l$ olmalıdır ve veri ile örüntü ağaçlarının anlık altağaçlarının da eşlenmesi gerekir. Anlık altağaçların eşlenmesi ikili çizge eşlemesi gibi ele şu şekilde ele alınabilir. İkili çizgeler düğümleri iki farklı kümeye ayrılan, var olan kenarların da farklı kümelerdeki düğümleri birbirine bağladığı çizgelerdir. Benzer şekilde $\langle T[1], T[2], \ldots, T[l]\rangle$'yi bir ikili çizgenin sol düğümler kümesi, $\langle P[1], P[2], \ldots, P[k]\rangle$'yi de ikili çizgenin sağ düğümler kümesi olarak düşünmek mümkündür. Sağ ve sol kümeler arasındaki kenarlar da eşleme durumlarının varlığı olsun. Bu durumda eğer böyle bir ikili çizgede maksimum eşleme sayısı k kadar ise her $\langle P[1], P[2], \ldots, P[k]\rangle$ farklı $\langle T[1], T[2], \ldots, T[l]\rangle$'lerle eşleniyor demektir. Dolayısıyla da P ve T'nin köklerinden başlayan en az bir eşleme vardır.

Algoritma 5 yukarıda verilen eşleme mantığı ile dinamik programlamanın ağaçların kök-sonra gezilerek uygulanmasıyla oluşturulmuştur. Maksimum ikili çizge eşleme polinomsal zaman karmaşıklığına sahip olduğundan, algoritma da Bölüm 4.3'de anlatılan eşleme yönteminin aksine polinomsal bir zaman karmaşıklığına sahiptir. Maksimum ikili çizge eşleme için Hopcroft ile Karp [21] tarafından geliştirilen ve $O(n^{5/2})$ zaman karmaşıklığına sahip olan algoritma kullanılmıştır. $n = |T|$ ve $m = |P|$ olarak düşünülürse, Algoritma 5'nin toplam zaman karmaşıklığı $O(nm * n^{5/2})$ kadar olur.

4.6.2 Gömülü-Sırasız Altağaç Eşleme

Kilpelainen'ın gömülü-sırasız ağaç eşleme algoritması örüntü ağacının düğümlerinin altkümelerinden oluşan eşleme sistemlerinden ($S(v)$) yararlanır. Dolayısıyla en kötü

Algoritma 5 Birebir-sırasız Ağaç Eşleme

Girdi: $T = (V, E)$ veri ağacı, $P = (W, F)$ örüntü ağacı
Çıktı: P ile eşlenen T'nin altağaçları

```
1:  M ← |T| x |P| boyutunda bir mantıksal tablo
2:  M ← false
3:  for i = 1 to |T| do
4:      v ← T[i];
5:      v_1, v_2, ..., v_l, l > 0 v'nin çocukları
6:      for j = 1 to |P| do
7:          w ← P[j];
8:          w_1, w_2, ..., w_k, k > 0 w'nun çocukları
9:          G = (X ∪ Y, BE)
10:         X = {v_1, v_2, ..., v_l}, Y = {w_1, w_2, ..., w_k} ve
11:         BE = {(x, y)|x ∈ X, y ∈ Y, M[x][y] = true};
12:         if label(w) = label(v) ve G'de maksimum eşleme boyutu k ise then
13:             M[i][j] ← true;
14:         end if
15:     end for
16:     if M[i][root(P)] = true then
17:         i'den başlayan bir eşleme vardır
18:     end if
19: end for
```

durumda zaman karmaşıklığı örüntü ağacının düğüm sayısına bağlı olarak üsseldir. Bir veri ağacı düğümü v için eşleme sistemi örüntü ağacının $\{w_1, w_2, \ldots, w_k\}$ düğümlerinin altkümelerinden oluşur, öyle ki, $\langle P[w_1], P[w_2], \ldots, P[w_k]\rangle$ $T[v]$'de gömülü olmalıdır. Eğer bir v' düğümü v'nin atası ise, $T[v']$'nin $T[v]$'yi içereceği açıktır, dolayısıyla $S(v') \supseteq S(v)$'dir. Ayrıca bir örüntü ağacı düğümü w için, $\{w\} \in S(v)$ olması için $children(w) \in S(v)$ olmalıdır. Algoritma 6'da $v \in T$'ler için eşleme sistemleri hesaplanması verilmiştir. Algoritmada en dıştaki döngüde veri ağacı kök-sonra olarak gezilmektedir. 6. satırda her veri ağacı düğümü için eşleme sistemi, çocuklarının eşleme sisteminden faydalanılarak hesaplanmakta, 17. satırda da güncellenerek saklanmaktadır. Gezilen düğümde başlayan eşleme olup olmadığı da 15. satırda belirtilmektedir. Algoritma $|P|$'ye bağlı olarak üssel olmasına rağmen, çoğu uygulamada $|P|$ küçük olduğundan kabul edilebilir bir performansa sahiptir.

Şekil 4.4'deki veri ve örüntü ağaçları için Algoritma 6'ye göre oluşturulmuş örnek eşleme sistemleri Çizelge 4.4'de örneklendirilmiştir. Buna göre algoritmada, 14.satırdaki

Algoritma 6 Gömülü-sırasız Ağaç Eşleme

Girdi: $T = (V, E)$ veri ağacı, $P = (W, F)$ örüntü ağacı
Çıktı: P ile eşlenen T düğümleri

```
1:  for i = 1 to |T| do
2:     v ← T[i];
3:     S ← {∅};
4:     v_1, v_2 ..., v_l, l > 0 v'nin çocukları;
5:     for p = 1 to l do
6:        S = {A ∪ B|A ∈ S, B ∈ S(v_p)};
7:     end for
8:     SΔ ← ∅;
9:     for all w ∈ W ve label(w) = label(v) olmalı do
10:       if children(w) ∈ S then
11:          SΔ ← SΔ ∪ {{w}}.
12:       end if
13:    end for
14:    if root(P) ∈ SΔ then
15:       T'nin i. düğümünde eşleme bulundu.
16:    end if
17:    S(v) = S ∪ SΔ.
18: end for
```

koşul çizelgede 3. ve 5. satırlarda doğru olmaktadır ve algoritmada 15. satırda da buna uygun olarak eşlenme bulunduğu bildirilmektedir.

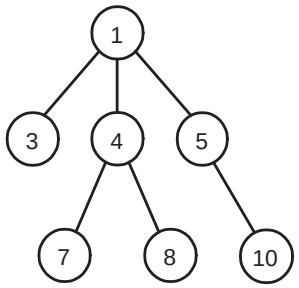
Çizelge 4.4: Gömülü-sırasız eşleme için örnek eşleme sistemleri.

i	S	$S\Delta$	$S(i)$
1	$\{\emptyset\}$	$\{\{2\}\}$	$\{\emptyset, \{2\}\}$
2	$\{\emptyset\}$	$\{\{1\}\}$	$\{\emptyset, \{1\}\}$
3	$\{\emptyset, \{\emptyset, 1\}, \{\emptyset, 2\}, \{\emptyset, 1, 2\}\}$	$\{\{1\},\{3\}\}$	$\{\emptyset, \{\emptyset, 1\}, \{\emptyset, 2\}, \{\emptyset, 1, 2\}\}$
4	$\{\emptyset\}$	$\{\{2\}\}$	$\{\emptyset, \{2\}\}$
5	$\{\emptyset, \{\emptyset, 1\}, \{\emptyset, 2\}, \{\emptyset, 1, 2\}\}$	$\{\{1\},\{3\}\}$	$\{\emptyset, \{\emptyset, 1\}, \{\emptyset, 2\}, \{\emptyset, 1, 2\}\}$

4.6.3 Gömülü-Sıralı Altağaç Eşleme

Kilpelainen'in gömülü-sıralı altağaç eşleme algoritması yapısal olarak yukarıda verilen birebir-sırasız ve gömülü-sırasız eşleme algoritmalarına benzemektedir. Bu algoritma P örüntü ağacı düğümlerine aralık numaraları atamayı gerektirmektedir. Bu atama işlemi şu şekilde olur: İlk önce $root(P)$'ye 1 atanır. Daha sonra tüm düğümler yukarıdan aşağı düzey düzey gezilir. Her düzeyde de kardeş düğümler soldan sağa

gezilir. Herhangi bir düzeyden başka bir düzeye geçildiğinde en son atanan numara 2 arttırılır. Kardeş düğümlere ise artan sırada aralık numaraları atanır. Örnek bir aralık numarası atama Şekil 4.5'de görülmektedir.



Şekil 4.5: Örnek bir ağacın düğümlerine atanmış aralık değerleri

Herhangi bir $v \in T$ düğümü için $\langle v \rangle$ gösterimi o düğümün *düğüm aralığını* gösterir. Ek olarak, v'nin *çocuk düğümleri* ise çocukların aralık numaralarının dizisinden oluşur. Sadelik açısından çocuk aralıkları biri başlangıç biri de bitiş olmak üzere bir çift aralık numarasından oluşur. Örnek olarak Şekil 4.5'de, kök düğümün aralığı $\langle 1 \rangle$ ve çocuklarının aralığı ise $\langle 3, 4, 5 \rangle$ ya da kısaca $\langle 3, 5 \rangle$'dir. Ortak gösterim için düğüm aralıkları da bir çift sayıdan oluşabilir, bu durumda başlangıç ve bitiş aralıklarının ikisi de o düğümün aralığı olur, yani başlangıç ve bitiş aralıkları aynıdır. Buna örnek olarak Şekil 4.5'deki ağacın kök düğümünün aralığı $\langle 1, 1 \rangle$ şeklinde gösterilir.

Gömülü-sıralı ağaç eşleme algoritması düğüm aralıkları ve çocuk aralıklarından oluşan eşleme kümeleri kullanır. Eşleme kümeleri sıralı bir yapıdır. Sıralama aralıkların başlangıç noktalarının artan sırasındadır. Bir $v \in T$ düğümünün eşleme kümesi P'nin çocuk aralıklarından oluşan öyle bir kümedir ki, bu küme $T[v]$'de gömülüdür. Algoritmanın sözde kodu Algoritma 7'de verilmiştir. Gömülü-sırasız ağaç eşleme algoritmasının biçimsel benzerliğine karşın zaman karmaşıklığı olarak polinomsal bir zaman karmaşıklığına $O(nm)$ sahiptir. Bunun sebebi de aralıkların sıralı tutulması ve 6. satırdaki aralık kümeleri birleştirme işleminin polinomsal zamanda yapılabilmesidir. 6. satırdaki aralık birleştirme işlemi, $\bigoplus$, şu şekilde olmaktadır (Algoritma 8): Eşleme kümeleri aynı anda ilk elemandan son elemana doğru gezilirken, her iki eşleme kümesinden alınan sıradaki aralıklar için ($l = \langle l.a, l.b \rangle$ ve $r = \langle r.a, r.b \rangle$), eğer çakışma varsa $l \bigoplus r = \langle min(l.a, r.a), max(l.b, r.b) \rangle$ şeklinde yeni bir aralık oluşmaktadır ve

bu aralık da sonuç eşleme kümesine eklenmektedir.

Algoritma 7 Gömülü-sıralı Ağaç Eşleme

Girdi: $T = (V, E)$ veri ağacı, $P = (W, F)$ örüntü ağacı
Çıktı: T'nin P ile eşlenen düğümleri

1: **for** $i = 1$ to $|T|$ **do**
2: $v \leftarrow T[i]$;
3: $M \leftarrow \emptyset$;
4: $v_1, \ldots, v_l$, $l > 0$ v'nin çocukları;
5: **for** $p = 1$ to l **do**
6: $M \leftarrow M \bigoplus M(v_p)$;
7: **end for**
8: $M\Delta \leftarrow \emptyset$;
9: **for all** $w \in W$ ve $label(w) = label(v)$ **do**
10: **if** $\langle children(w) \rangle \in M$ **ya da** $children(w) = \emptyset$ **then**
11: $M\Delta \leftarrow M\Delta \cup \{\langle w \rangle\}$
12: **end if**
13: **end for**
14: **if** $\langle root(P) \rangle \in M\Delta$ **then**
15: T'nin i. düğümünde bir eşleme bulundu.
16: **end if**
17: $M(v) = M \cup M\Delta$.
18: **end for**

Çizelge 4.5: Gömülü-sıralı eşleme için örnek eşleme kümeleri.

i	M	$M\Delta$	$M(i)$
1	$\{\emptyset\}$	$\{\langle 4,4 \rangle\}$	$\{\langle 4,4 \rangle\}$
2	$\{\emptyset\}$	$\{\langle 3,3 \rangle\}$	$\{\langle 3,3 \rangle\}$
3	$\{\langle 3,3 \rangle, \langle 4,4 \rangle\}$	$\{\emptyset\}$	$\{\langle 3,3 \rangle, \langle 4,4 \rangle\}$
4	$\{\emptyset\}$	$\{\langle 4,4 \rangle\}$	$\{\langle 4,4 \rangle\}$
5	$\{\langle 3,4 \rangle\}$	$\{\langle 1,1 \rangle\}$	$\{\langle 1,1 \rangle, \langle 3,4 \rangle\}$

Şekil 4.4'deki veri ve örüntü ağaçları için Algoritma 7'ye göre oluşturulmuş örnek eşleme kümeleri Çizelge 4.5'de örneklendirilmiştir. Buna göre algoritmada 14.satırdaki koşul sadece 5. düğüm için doğru olmaktadır ve çizelgede 5. satır için buna uygun olarak eşlenme bulunduğu bildirilmektedir.

4.6.4 Birebir-Sıralı Altağaç Eşleme

Birebir-sıralı altağaç eşleme algoritması, gömülü-sıralı altağaç eşleme algoritmasıyla aynı yöntemi izlemekte ve zaman karmaşıklığı olarak da benzer zaman karmaşıklığına

Algoritma 8 Aralık Birleştirme

Girdi:Sıralı $L = l_1, l_2, \ldots, l_m$ ve $R = r_1, r_2, \ldots, r_n$ aralık listeleri
Çıktı:Sıralı $R \bigoplus L$ birleştirilmiş aralık listesi.

$i = 1$; $j = 1$
$l_{m+1} = [\infty, \infty]$; $r_{n+1} = [\infty, \infty]$;
while $i \leq m$ or $j \leq n$ **do**
  if $l_i.b + 1 < r_j.a$ **then**
    Output l_i;
    $outb \leftarrow l_i.b$ ▷ l_i ile r_j birleştirilemez
  else
    ▷$l_i.b + 1 > r_j.a$;
  end if
  if $l_i.a < r_j.a$ **then**
    while $l_i.b + 1 > r_j.a$ **do**
      ▷ l_i ile r_j birleştirilemez
      $x = r_j.b$;
      $j = j + 1$;
    end while
    $outb = max(l_i.b, x)$;
    Output $\langle l_i.a, outb \rangle$;
  else
    ▷$l_i.a > r_j.a$;
    Output r_j;
    $outb = r_j.b$;
    $j = j + 1$;
  end if
  while $l_i.b \leq outb$ **ve** $l_i.b + 1 < r_j.a$ **do**
    ▷ Sonuç aralıklarında yer alan ve r_j ile birleştirilemeyen l_i'leri atla;
    $i = i + 1$;
  end while
end while

sahiptir. Fakat, gömülü-sıralı altağaç eşleme algoritmasından, 17. satırda farklılaşmaktadır. Gömülü-sıralı altağaç eşlemede 17. satırda $M(v) = M \cup M\Delta$ hesaplanırken, birebir-sıralı eşleme için $M(v) = M\Delta$'nın hesaplanması yeterli olmaktadır. Çünkü birebir-sıralı eşleme için torunların değil, sadece çocukların eşlemeye katkısı olmaktadır. Dolayısıyla ağaç kök-sonra gezilirken torun düğümlerin eşlemeye muhtemel katkısı yukarı taşınmaz, sadece çocukların eşlemeye katkısı dikkate alınmaktadır.

4.7 Ağaç Gizleme Performans Sonuçları

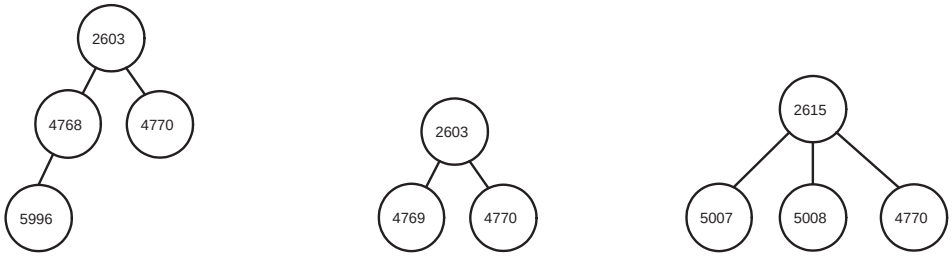
Bu bölümde geliştirilen ağaç gizleme algoritmasının örnek veritabanları üzerinde test edilmesiyle elde edilen performans sonuçları anlatılmaktadır. Ağaç gizleme algoritması Java ile gerçekleştirilmiştir. Devamda verilen her test 5 kere tekrarlanmış ve ortalama sonuçlar burada gösterilmiştir.

Testlerde üç farklı veritabanı kullanılmıştır. Bunlardan birincisi [48]'de değinilmiş olan `CSLOGS` veritabanı, ikincisi `http://sourceforge.net/projects/dmtl/` adresinden edinilen `WEBLOG` veritabanı ve üçüncüsü de sentetik `T50M50N10` veritabanıdır. İlk iki veritabanı gerçek veritabanlarıdır sonuncusu ise yapaydır. Sonuncu veritabanı [48]'den tanıtılan ağaç oluşturucu kullanılarak oluşturulmuştur. Her üç veritabanı için de tüm ağaçlar kenar etiketsizdir.

`CSLOGS` veritabanı 59691 kullanıcı etkileşimini içeren ve RPI bilgisayar bilimi departmanındaki (`http://www.cs.rpi.edu/`) kategorik olarak bulunan web sayfaları erişiminden oluşmaktadır. Veritabanında 13361 farklı web sayfası bulunmaktadır. Her kullanıcı etkileşimi bir web sayfaları gezintisine karşılık gelmektedir ve ortalama 23.3 sayfa içermektedir. `T50M50N10` veritabanı 50000 ağaç içermektedir. Her ağaç düğüm etiketlerini 10 elemanlık bir alfabeden almakta ve ortalama 50 düğüm içermektedir. `WEBLOG` veritabanı ise ortalama her ağacın 7 düğüm içerdiği ve düğüm alfabesinin 9060 farklı elemandan oluştuğu 8074 ağaç içermektedir.

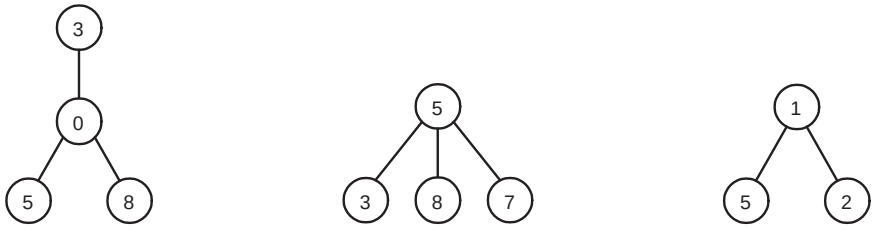
Hassas örüntülerin belirlenmesi için ilk önce `CSLOGS` ve `T50M50N10` veritabanları üzerinde veri madenleme algoritması kullanılmış ve bulunan sık altağaçlardan 3'er tanesi hassas olarak seçilmiştir. Örüntüler her iki veritabanı için *örüntü1*, *örüntü2*, and *örüntü3* olarak isimlendirilmiştir. Bu örüntüler Şekil 4.6 ve Şekil 4.7'lerde görülmektedir. `WEBLOG` veritabanı ise üç farklı örüntü seçilmiştir. Bu örüntüler Şekil 4.8'de verilmiştir. Bu şeklide numaralarla belirtilen düğümler gerçekte Çizelge 4.6'deki web sayfalarına karşılık gelmektedir. Bu veritabanı için, *örüntü1* Madonna ile ilgili sayfalardan, *örüntü2* Nazım Hikmet ile ilgili sayfalardan ve *örüntü3* de Orhan Veli Kanık ile ilgili sayfalardan oluşmaktadır.

Performans testlerinde 3 farklı metrik kullanılmıştır. Her metrik tüm altağaç içerme sınıfları için değişen eşik değerlerine göre ölçülmüştür. 3 farklı metrikten ilki çalışma zamanı, ikincisi ($M0$) maskelenen düğüm sayısı, üçüncüsü de dönüştürme işleminden sonraki sık alt ağaçların dönüştürme öncesindeki sık altağaçlara oranıdır ($M1$). $M1$ metriğinin ölçülmesi için alt ağaç madenleme uygulaması kullanılmıştır. Bu uygulama ile farklı eşik değerleri için gizlemeden önce ve sonraki sık altağaçlar bulunmuş ve sayıları oranlanmıştır.



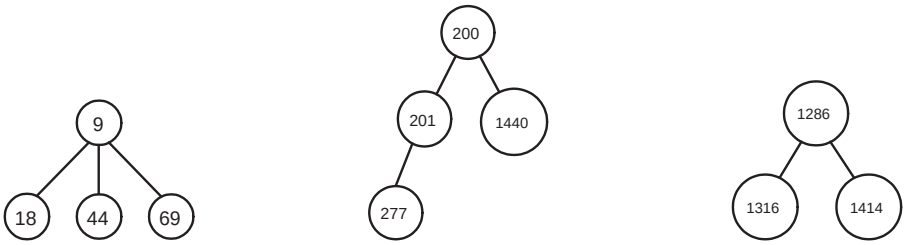
örüntü1 *örüntü2* *örüntü3*

Şekil 4.6: `CSLOGS`'dan gizlenmek üzere seçilmiş örüntüler



örüntü1 *örüntü2* *örüntü3*

Şekil 4.7: `T50M50N10`'dan gizlenmek üzere seçilmiş örüntüler



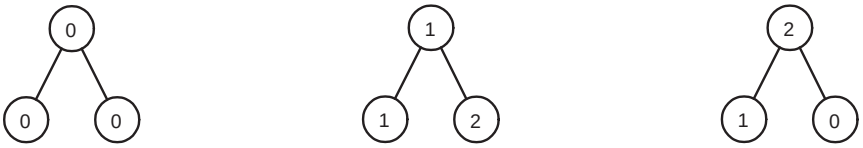
örüntü1 *örüntü2* *örüntü3*

Şekil 4.8: WEBLOG'dan gizlenmek üzere seçilmiş örüntüler

Çizelge 4.6: WEBLOG veritabanında etiketlere karşılık gelen URL adresleri

ID	URL
9	http://www.cs.rpi.edu/∼kennyz/madonna_lyrics/HomePage.html
18	http://www.cs.rpi.edu/∼kennyz/madonna_lyrics/like.a.virgin.html
44	http://www.cs.rpi.edu/∼kennyz/madonna_lyrics/true.blue.html
69	http://www.cs.rpi.edu/∼kennyz/madonna_lyrics/like.a.prayer.html
200	http://www.cs.rpi.edu/∼sibel/poetry/nazim_hikmet.html
201	http://www.cs.rpi.edu/∼sibel/poetry/frames/nazim_hikmet_1.html
277	http://www.cs.rpi.edu/∼sibel/poetry/poems/nazim_hikmet/turkce.html
1440	http://www.cs.rpi.edu/∼sibel/poetry/frames/nazim_hikmet_2.html
1286	http://www.cs.rpi.edu/∼sibel/poetry/orhan_veli.html
1316	http://www.cs.rpi.edu/∼sibel/poetry/frames/orhan_veli_1.html
1414	http://www.cs.rpi.edu/∼sibel/poetry/frames/orhan_veli_2.html

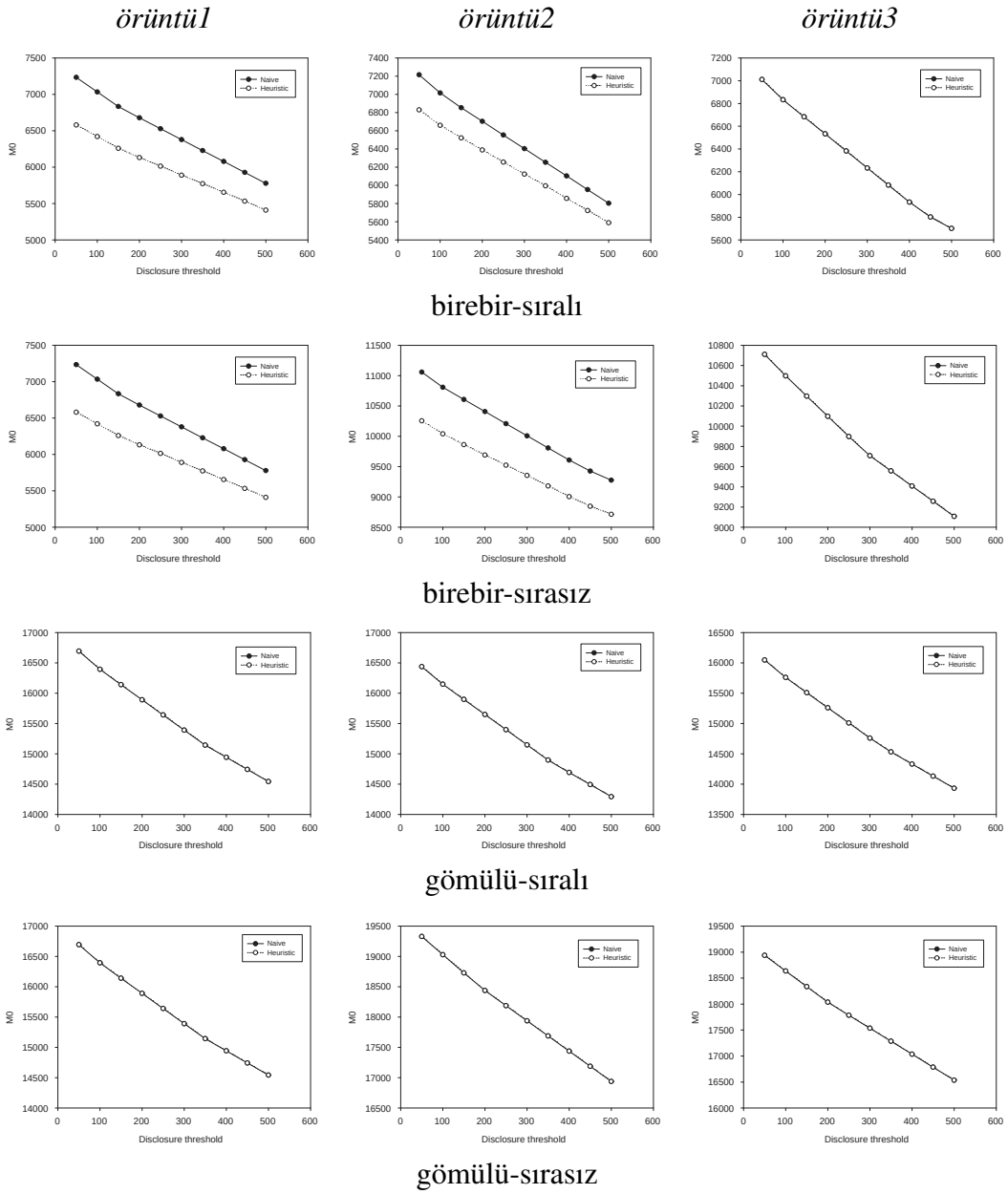
3 farklı veritabanı üzerinde performans deneylerinin yanı sıra T10M30N3 adında her ağacın ortalama 30 düğüm içerdiği, her düğümün 3 farklı etiketten birine sahip olduğu ve 10000 ağaçtan oluşan yapay bir veritabanı oluşturulmuştur. Bu veritabanının oluşturulmasındaki amaç ağaç gizlemede uygulanan yerel sezgiselin naif yöntemle karşılaştırılmasıdır. Naif yöntem ile yerel sezgiselin kullanıldığı algoritmalardaki global sezgisel aynıdır. Bu veritabanı için seçilmiş 3 örüntü Şekil 4.9'de görülmektedir. Örüntüler küçük boyutlarda seçilerek destek değerlerinin fazla olması amaçlanmıştır. Bu veritabanı için performans sonuçları Şekil 4.10 ve Şekil 4.11'de görülmektedir.



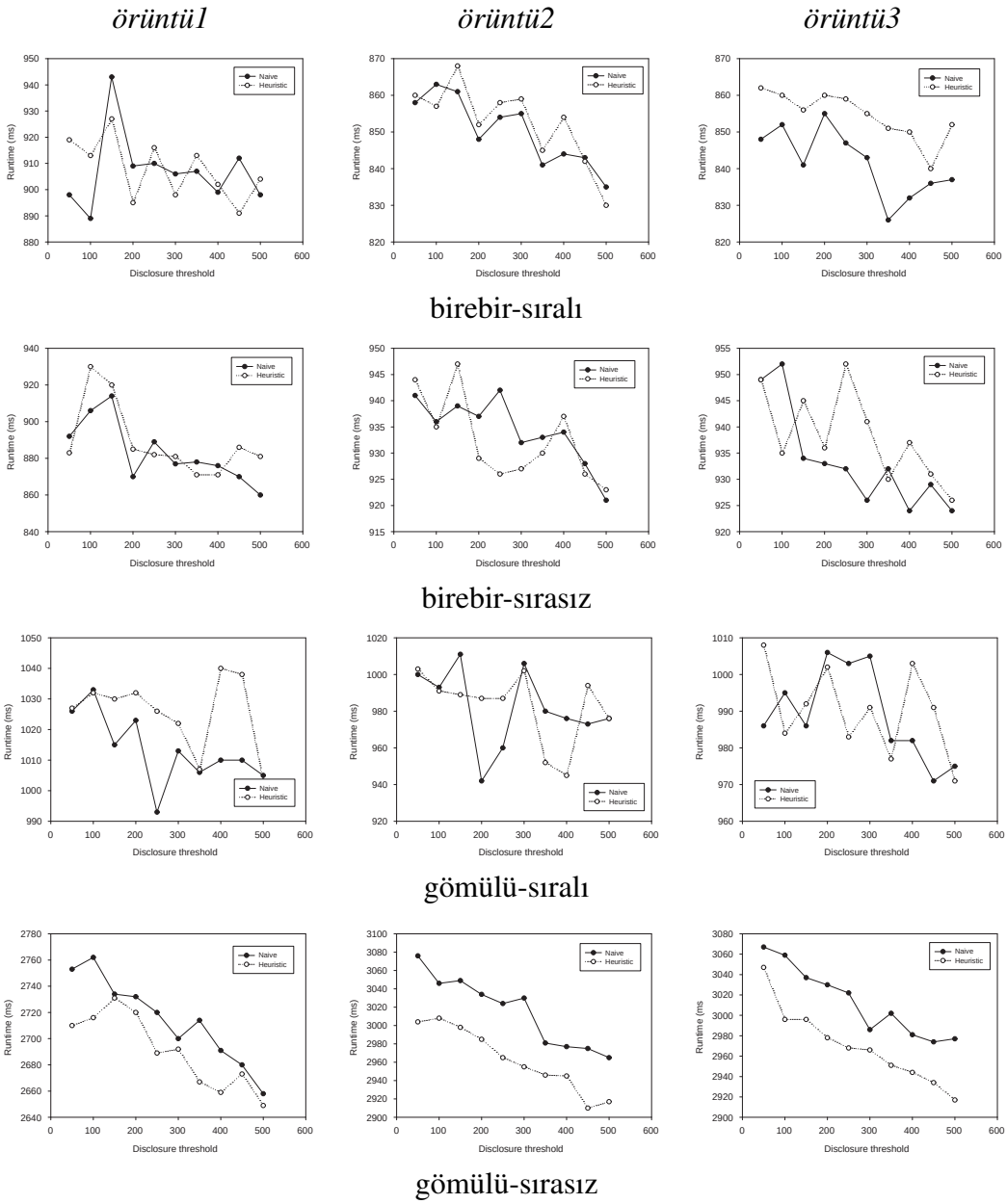
örüntü1 *örüntü2* *örüntü3*

Şekil 4.9: T10M30N3'ten gizlenmek üzere seçilmiş örüntüler

Şekil 4.10'de T10M30N3'te tüm eşleme sınıfları ve 3 farklı örüntü için gizleme yapılmış ve değişen eşik değerlerine göre ölçülen $M0$ metriklerinin grafiği verilmiştir. Burada birebir-sıralı eşleme için *örüntü1* ve *örüntü2*'nin gizlenmesinde yerel sezgisel naif çözümden daha etkili olmuştur. Yerel sezgisel daha az maskeleme ile gizleme işlemini



Şekil 4.10: `T10M30N3` örüntüleri için naif yöntemle yerel sezgiselin performans sonuçları.

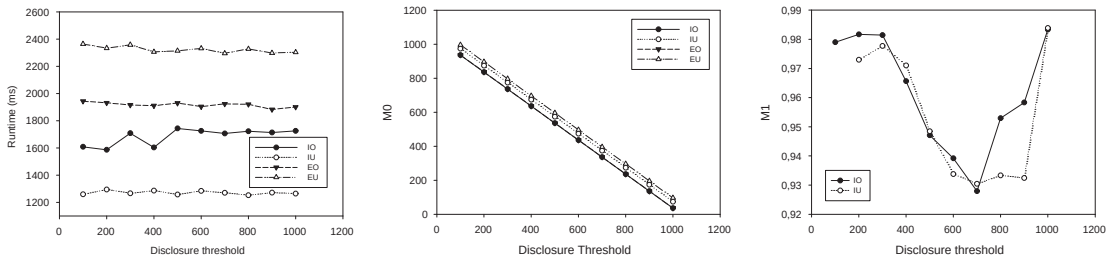


Şekil 4.11: `T10M30N3` örüntüleri için naif yöntemle yerel sezgiselin performans sonuçları.

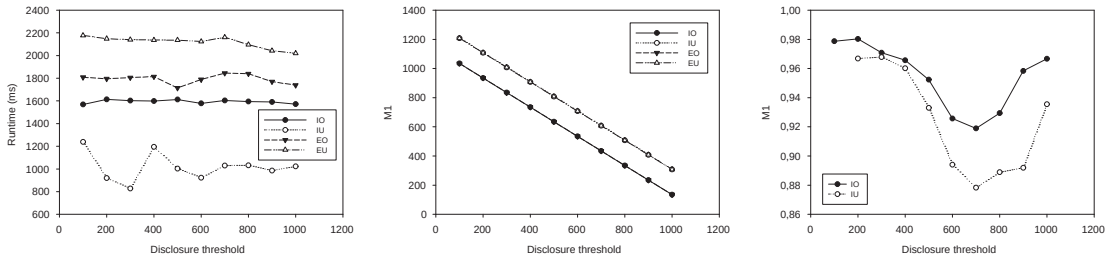
başarıyla yaparken, naif yöntem daha fazla maskeleme yapmıştır. *örüntü3* için yerel sezgisel ile naif yöntemin farklı sonuçlar vermemesi normal bir durumdur. Çünkü yerel sezgisel çakışan eşlemeler olduğu zaman iyi bir yöntem olmaktadır. Fakat *örüntü3*'de aralarında ata-torun ilişkisi bulunan ve etiketleri aynı olan düğümler olmadığından bu örüntünün ağaçlarda var olan eşlemelerinin çakışması mümkün değildir. *örüntü1* ve *örüntü2* için ise bahsedilen durum mevcuttur, dolayısıyla çakışan eşlemeler olmuştur ve yerel sezgisel de en az maskelemeyle gizleme yapmıştır. Aynı durum birebir-sırasız gizleme için de geçerlidir. Fakat gömülü-sıralı ve gömülü-sırasız eşleme için geçerli değildir. Bu iki sınıf için de naif yöntem ile yerel sezgisel aynı başarıya sahiptir. Bunun da sebebi kök maskelemenin seçilmiş olması ve kökün anlık alt ağaçlarının, veri ağaçlarının en alt kısımlarında yer alan alt ağaçlarla eşlenmelerinin hiçbir şekilde bozulmuyor olmasıdır. Veri ağacının alt düzeylerinde yer alan ve örüntü ağacı kökünün anlık altağaçlarıyla eşlenen altağaçlar, veri ağacı kök-sonra gezildikçe olabilecek tüm eşlemelere katkıda bulunurlar. Sadece kök maskelemeyle bunun önüne geçilemeyeceğinden yerel sezgisel ile naif yöntem aynı başarıya sahiptirler.

Şekil 4.11'de de naif yöntem ile yerel sezgiselin çalışma zamanları karşılaştırılmıştır. Esasında yerel sezgisel ile naif yöntem arasında çalışma zamanı bakımından bir üstünlük beklenmemektedir. Çünkü her iki durumda da ağaçların gezilmesi gerekmektedir. Aradaki tek fark yerel sezgiselde eşlemeler bulundukları anda düğüm maskelemesi yapılmakta, naif yöntemde ise bulunan eşlemeler not edilmekte ve ağaç gezintisi bittikten sonra ilgili düğümler maskelenmektedir. Grafiklerden de görülebileceği gibi çalışma zamanı açısından dikkat çekici bir fark bulunmamaktadır.

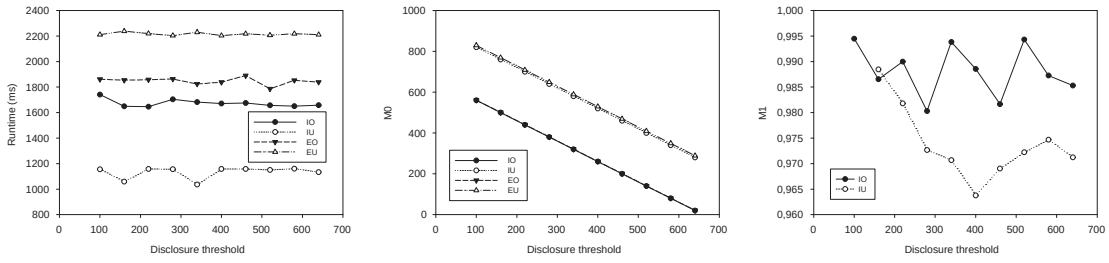
`CSLOGS` veritabanında seçilen örüntülerin gizleme sonuçlarının performans grafikleri Şekil 4.12'de verilmiştir. Çalışma zamanı açısından eşleme sınıfları iyiden kötüye şu şekilde sıralanmışlardır: birebir-sırasız, birebir-sıralı, gömülü-sıralı ve gömülü-sırasız. Bu birebir-sırasız ile birebir-sıralı eşleme sınıfları dışında beklenen bir sonuçtur. Çünkü gömülü eşlemenin birebir eşlemeye; sırasız eşlemenin de sıralı eşlemeye göre sayıca fazla olması en doğal durumdur. Burada birebir-sıralı eşlemenin birebir-sırasız eşlemeden daha yavaş olmasının sebebi tamamen Algoritma 7'nin yapısından ve ağaçların fazla sıkışık olmamasından kaynaklanmaktadır. Buradaki zaman farkı örüntü ağacı-



örüntü1

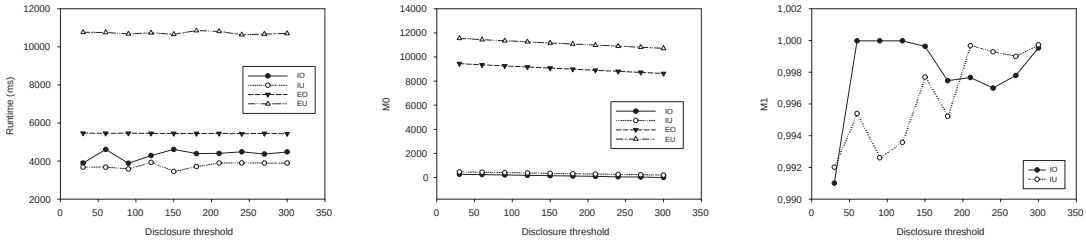


örüntü2

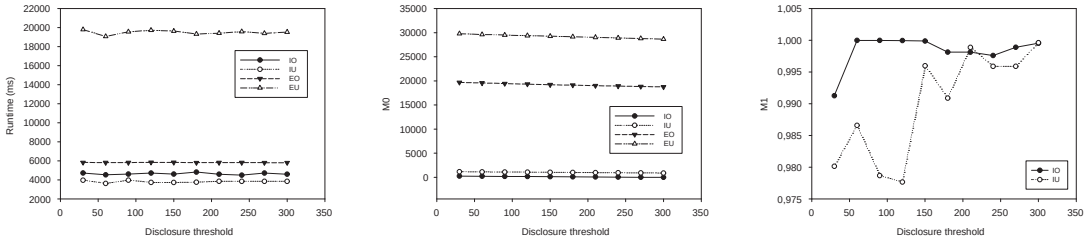


örüntü3

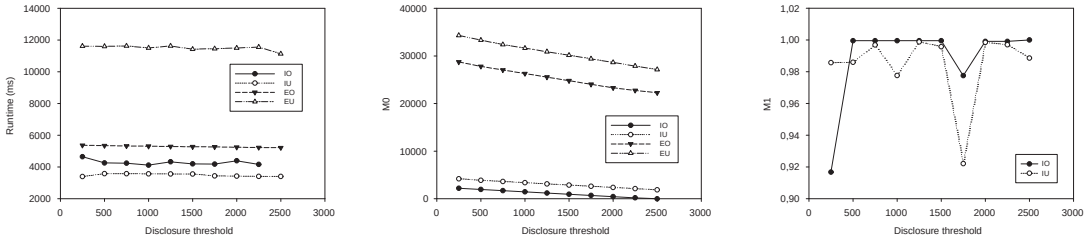
Şekil 4.12: `CSLOGS` performans sonuçları



örüntü1



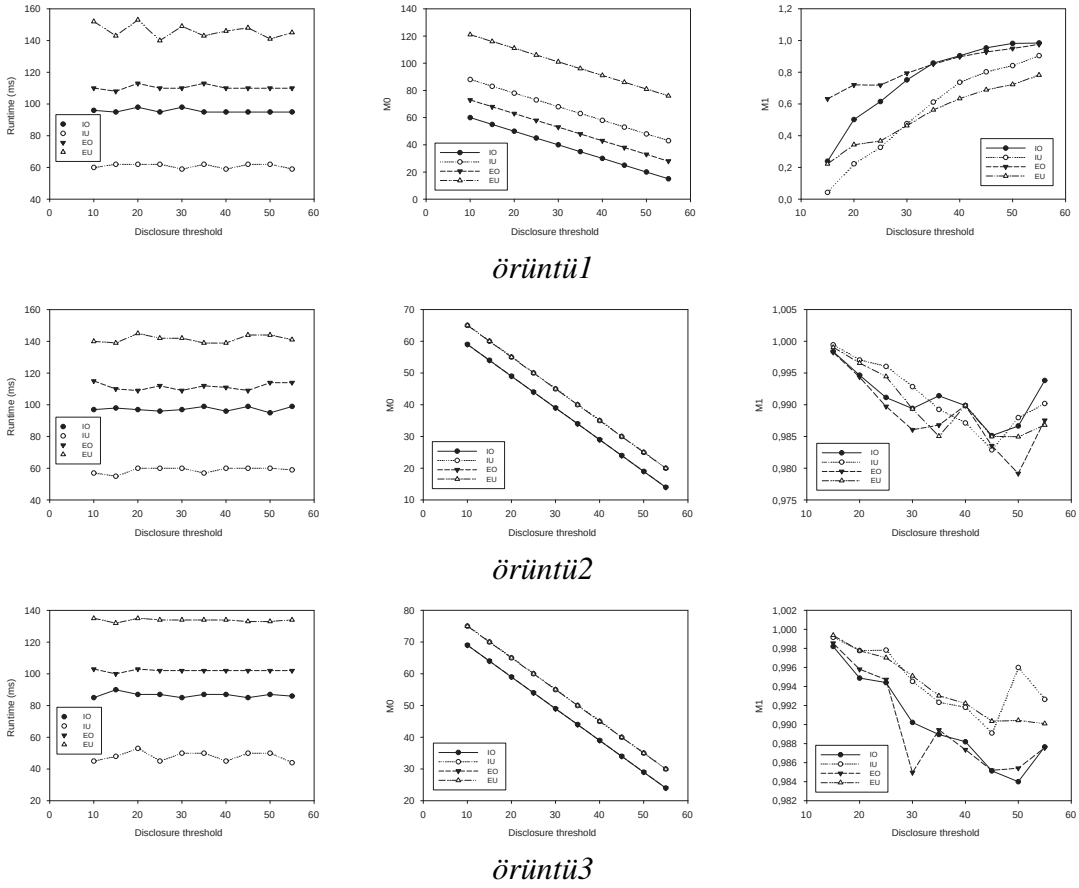
örüntü2



örüntü3

Şekil 4.13: `T50M50N10` performans sonuçları

nın aralık numarası atılması nedeniyle bir fazla gezilmesi, aralık birleştirme işleminin sıralı yapılması ve birleştirilen aralıkların tekrar sıralanmasından kaynaklanmaktadır. Bu veritabanı için $M1$ metriği sadece birebir-sıralı ve birebir-sırasız eşleme için hesaplanabilmiştir. Bu metriğin değerce fazla olması veritabanı bütünlüğünün korunduğunu gösterir. Grafikten de görülebileceği gibi tüm örüntüler için birebir-sıralı eşleme birebir-sırasız eşlemede veritabanı daha fazla korunmuştur. Bu doğal bir sonuçtur, çünkü birebir-sırasız eşleme sayısının he zaman birebir-sıralı eşleme sayısından fazla olması gerekir. Bu da daha fazla düğümün maskelenmesi ihtimalini doğurur, böylece veritabanı daha fazla bozulmuş olur. *örüntü3* ve *örüntü2* için $M0$ metriğinin bazı sınıflar için aynı çıkmasının sebebi gizleme yapılan ağaçların çoğunun gizlenen örüntüleri sadece bir kez desteklemesidir.



Şekil 4.14: `WEBLOG` performans sonuçları

`T50M50N10` veritabanında seçilen örüntülerin gizleme sonuçlarının performans grafikleri Şekil 4.13'de verilmiştir. Bu veritabanındaki gizleme performans sonuçları çalışma zamanı ve $M1$ metrikleri için Şekil 4.12'deki sonuçlarla benzerdir. $M0$ metriği için ise daha belirgin sonuçlar elde edilmiştir. Gömülü sınıflar için çok sayıda eşleme olduğu ve daha çok düğümün maskelendiği grafikten anlaşılmaktadır. Bu metriğe göre eşleme sayılarının çokluğu ise farklı sınıflar için artan sırada şu şekildedir: birebir-sıralı, birebir-sırasız, gömülü-sıralı ve gömülü-sırasız.

`WEBLOG` veritabanında seçilen örüntülerin gizleme sonuçlarının performans grafikleri Şekil 4.14'de verilmiştir. Bu veritabanındaki performans sonuçları çalışma zamanı ve $M0$ metrikleri için Şekil 4.12'deki sonuçlarla benzer sonuçları vermiştir. $M1$ metriği için ise farklı örüntüler için ayırt edici olmayan farklı sonuçlar elde edilmiştir. Bunun da sebebi veritabanının ve seçilen örüntülerin destek değerlerinin küçük olmasıdır.

5 ÇİZGE GİZLEME

5.1 Çizge Örüntüleri

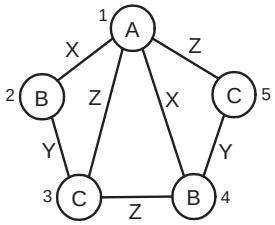
Çizge örüntüleri, *çizge eşbiçimliliğinden (graph isomorphism)* yararlanılarak tanımlanan örüntülerdir. Aşağıda tanımı verilen çizge eşbiçimliliğinde H örüntü çizgesini, G de veri çizgesini göstermektedir.

Tanım 15 (Gömülü Altçizge Eşbiçimliliği)**.** $H = (V_1, E_1, \Sigma_V, \phi_{V_1}, \Sigma_E, \phi_{E_1})$ ve $G = (V_2, E_2, \Sigma_V, \phi_{V_2}, \Sigma_E, \phi_{E_2})$ çizgeleri veriliyor olsun. Eğer (i) $\forall v \in V_1 . \phi_{V_1}(v) = \phi_{V_2}(\varphi(v))$, (ii) $\forall (u,v) \in E_1$ $(\varphi(u), \varphi(v)) \in E_2$ ve $\phi_{E_1}(u,v) = \phi_{E_2}(\varphi(u), \varphi(v))$ koşullarını sağlayan birebir bir $\varphi : V_1 \rightarrow V_2$ fonksiyonu varsa H çizgesi G çizgesinin bir *gömülü altçizgesidir* (ya da kısaca *altçizgesidir*). Bu durum $H \preceq_e G$ ile gösterilir. φ belirteci hem düğüm ve kenar etiketlerini hem de kenar varlığı ilişkisini korur fakat örüntüde olmayan kenarların veri çizgesinde de olmamasını gerekli kılmaz.

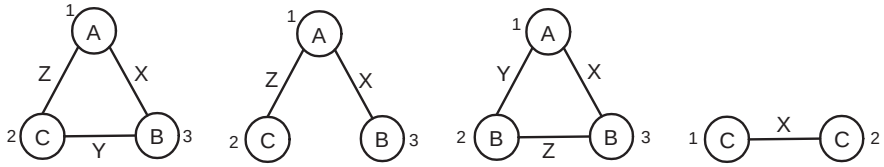
Tanım 16 (Birebir Altçizge Eşbiçimliliği)**.** $H = (V_1, E_1, \Sigma_V, \phi_{V_1}, \Sigma_E, \phi_{E_1})$ ve $G = (V_2, E_2, \Sigma_V, \phi_{V_2}, \Sigma_E, \phi_{E_2})$ çizgeleri veriliyor olsun. Eğer (i) $\forall v \in V_1 . \phi_{V_1}(v) = \phi_{V_2}(\varphi(v))$, (ii) $\forall (u,v) \in E_1 | (\varphi(u), \varphi(v)) \in E_2 \wedge \phi_{E_1}(u,v) = \phi_{E_2}(\varphi(u), \varphi(v))$, (iii) $\forall (u,v) \notin E_1 | (\varphi(u), \varphi(v)) \in E_2$ şartlarını sağlayan birebir bir $\varphi : V_1 \rightarrow V_2$ fonksiyonu varsa, H çizgesi G'nin *birebir altçizgesidir* ve bu durum $H \preceq_i G$ şeklinde gösterilir. Burada φ belirteci hem kenar ve düğüm etiketlerini hem de kenar ilişkisinin olup olmadığı bilgisini korur. Yani örüntüde iki düğüm arasında olmayan bir kenar bu iki düğümün eşlendiği veri çizgesi düğümleri arasında da olmamalıdır.

Bu iki çizge eşbiçimliliğinden hareketle iki farklı çizge örüntüsü sınıfı tanımlanır: Gömülü altçizge sınıfı ve birebir altçizge sınıfı. Eğer çizgeler, kenar etiketlerinden yoksun ise $\phi_{E_1}(u,v) = \phi_{E_2}(\varphi(u), \varphi(v))$ ifadesi herhangi (u,v) kenar için doğru bir tanımlama olur ve kenar varlığını (etiketler dikkate alınmaksızın) ifade eder.

Şekil 5.1'de kenarları ve düğümleri etiketli olan örnek bir çizge görülmektedir. Bu çizgenin düğümleri numaralandırılmış ve numaralar düğümlerin yanında gösterilmiştir.



Şekil 5.1: Örnek kenar ve düğüm etiketli çizge



Şekil 5.2: Örnek altçizgeler

Şekil 5.2(a)'deki örnek altağaç Şekil 5.1'nin hem gömülü hem de birebir bir altçzgesidir. Şekil 5.2(b)'deki altçizge gömülü bir altçizgedir fakat birebir değildir. Şekil 5.2(c) ve Şekil 5.2(d)'deki altçizgeler ise ne gömülüdür ne de birebirdir.

Literatürdeki çizge verilerini madenleme uygulamaları sadece gömülü altçizgeler üzerinde yoğunlaştığından, bu tezde çizge örüntü sınıfı olarak sadece gömülü altçizge eşbiçimliliği üzerinde çalışılmıştır.

Eğer bir H örüntü çizgesi bir G veri çizgesinin gömülü bir altçizgesi ise, bu durum $H \preceq G$ biçiminde belirtilir. Hatta G'nin H'yi içerdiği ve desteklediği de bu durumdan anlaşılır. Bir veri çizgesi, bir örüntü çizgesini birden çok sayı ve şekilde destekleyebilir. Çünkü farklı eşbiçimlilikler, farklı çizge içerme durumlarını oluştururlar, bu da farklı içerme sayısının artmasına neden olur. Ağaçlardaki destek değerine benzer olarak çizgeler için de ağırlıklı destek değerinden bahsedilebilir. Örnek olarak Şekil 5.1'deki çizge (G) ile ve Şekil 5.2(b)'deki altçizgeyi (H) verebiliriz. Şekillerden görülebileceği üzere, G H'yi desteklemekte ve dört farklı destekleme biçimi olduğundan H'nin ağırlıklı destek değeri dörttür.

Tanım 17 (Çizge Gizleme Problemi)**.** $\mathcal{D}$ diye bir çizgeler veritabanı, $\mathcal{P}_h$ diye hassas örüntü çizgeler veritabanı ve ψ diye de bir eşik değeri verilmiş olsun. Çizge gizleme

problemi $\mathcal{D}$'nin $\mathcal{D}'$ dönüştürülmesini gerektirir, öyle ki:

- $\forall P \in \mathcal{P}_h . sup_{\mathcal{D}}(P) < \psi$ olmalı ve
- $\mathcal{D}$ ile $\mathcal{D}'$ mümkün olduğunca benzer olmalıdır.

Daha önce tanımlanan ağaç gizleme problemine benzer olarak Tanım de iki koşulun sağlanmasını gerektirmektedir. İlk olarak dönüştürme sonucunda $\mathcal{D}'$'de örüntü çizgelerini içeren veri çizgelerinin sayısının ψ değerinden küçük olması gerekir, böylece gizleme gerekliliği sağlanmış olur. İkinci olarak da dönüştürme işlemi o kadar etkin olmalıdır ki, veritabanı neredeyse orijinali ile aynı olmalıdır.

Teorem 2. Çizge gizleme problemi NP-Hard bir problemdir. **İspat:** Çizgeler belirli kısıtlamalar altında ağaç biçiminde veriler olabilmektedir (döngüsel olmadıkları zaman). Teorem 1 ağaç gizleme probleminin NP-Hard bir problem olduğunu ispatlamaktadır. Dolayısıyla çizge gizleme probleminin NP-Hard oluşu, ağaç gizleme probleminin NP-Hard oluşundan gelmektedir [18].

Çizge gizleme probleminin çözülmüş olması için G veri çizgesi ile H örüntü çizgesi arasında altçizge içerme durumunun olmaması gerekir. Bu nedenle çizge gizleme problemi çizge içerme durumunun test edilmesini gerektirdiği için, altçizge içerme problemi çözümüne ihtiyaç duymaktadır. Bunun için, bu tezde Ullman [42] tarafından geliştirilmiş olan en kötü durumda da üssel bir çalışma zamanına sahip olan algoritmadan faydalanılmıştır. Devamda Ullman'ın algoritması anlatılmaktadır.

5.2 Ullman'ın Çizge İçerme Algoritması

Ullman'ın [42] algoritması *geri-izlemeli (backtracking)* bir tasarıma sahiptir ve hem çizge eşbiçimliliği hem de altçizge eşbiçimliliği problemlerini çözebilmektedir. Çizge gizleme problemi de altçizgelerin incelenmesini gerektirdiğinden, Ullman'ın algoritması altçizge içerme testlerinin yapılması için yeterli olmaktadır. Ullman'ın algoritması girdi olarak bir G veri çizgesi ve bir P örüntü çizgesi almaktadır. Girdi çizge-

leri *bitişiklilik matrisleri (adjacency matrix)* biçimindedir ve tüm işlemler bu matrisler üzerinden yapılmaktadır. Bir çizge düğümlerini farklı dizilmesinden kaynaklanacak şekilde birden çok bitişiklilik matrisine sahip olabilir. Algoritma da bu durumdan hareketle, bir permütasyon matrisinin yardımıyla G'nin olabilecek tüm farklı bitişiklilik matrislerini üretmekte ve H örüntü çizgesinin bitişiklilik matrisiyle karşılaştırmaktadır. G'nin H'yi içermesi en başta $|G| > |H|$ koşulunu gerektirdiğinden karşılaştırma G'nin oluşturulan farklı bitişiklilik matrislerinin $|H|$ boyutundaki alt matrisleri ile örüntü matrisinin tümü arasında olmaktadır.

Ullman algoritmasının temeli permütasyon matrislerinin üretilmesidir. Bir permütasyon matrisi elemanları 0 ya da 1'lerden oluşan ve her satır ve sütunun en fazla bir adet 1 içerebildiği bir matristir. G veri çizgesinin bitişiklilik matrisi $n \times n$ boyutunda bir M_G matrisi; H örüntü çizgesinin bitişiklilik matrisi de $m \times m$ boyutunda bir M_H matrisi; P de M_G'nin permütasyon matrisi olsun. Eğer M_G ve M_H aynı satır ve sütun sayılarına sahip iseler $M_H = P * M_G * P^T$ eşitlik koşulu M_G ile M_H arasındaki bir eşbiçimlilik olduğunu ifade eder. Altçizge eşbiçimliliğinin denetlenmesi için ise M_G'nin alt çizgelerinin belirlenmesi gerekir. M_G'nin altçizgeleri M_G'nin ilk $|H|$ kadar satırlarından ve ilk $|H|$ kadar sütunlarından oluşan matrisler olarak seçilip ve $M_{G[1..|H|],[1..|H|]}$ ile gösterilirler. $M_H = P * M_G * P^T$ eşitlik koşulu $M_H = (P * M_G * P^T)_{[1..|H|],[1..|H|]}$ biçiminde yeniden düzenlenir ve koşul sağlanırsa P permütasyon matrisi ile H'nin G'de bir altçizge eşbiçimliliği bulunmuş olur. $(P * M_G * P^T)_{[1..|H|],[1..|H|]} = P_{[1..|H|],[1..|G|]} * M_G * (P_{[1..|H|],[1..|G|]})^T$ eşitliği matris çarpımı düşünüldüğünde doğru olan bir ifadedir. Algoritmada daha hızlı hesaplama için P'nin $P_{[1..|H|],[1..|G|]}$ kısmı üzerinden hesaplamalar yapmak yeterli olmaktadır.

Algoritma 9, Ullman'ın algoritmasını özyineli bir şekilde kabataslak vermiştir. En başta P permütasyon matrisinin tüm değerlerine 0 ve k'ye de 1 atanır. Algoritma k boyutunda parçalı eşlemeler buldukça özyineli olarak kendini çağırır (8. satır) ve m boyutunda parçalı eşleme bulunca da bulunan parçalı eşlemenin bir altçizge içerme durumu olduğunu anlayarak anlık permütasyonu en başta olan $\mathcal{P}$ kümesine ekler (2. satır). Bu permütasyon bir eşleme olduğuna karşılık gelir. 7. satır çizge içerme testini k kadar düğüm için yapar. Burada dikkat edilmesi gereken karşılaştırma işlecinin

tam bir eşitlik ($=$) işleci olmayabileceğidir. Birebir çizge eşleme için bu karşılaştırma işleci tam bir eşitlik ($=$) iken, gömülü çizge eşlemede ise $\preceq$ halini alır. Çünkü gömülü eşlemede eşitliğin sol tarafında yer alan ve kenar olmama durumunu belirten '-' sembolünün eşitliğin sağ tarafında karşılık gelen öğeyle birebir aynı olması beklenmez. Dikkat edilmesi gereken bir diğer durum da P matrisinin elemanları sayılardan (0 ya da 1) oluşmakta iken, M_G matrisinin sembollerden (kenar ve düğüm etiketleri) oluştuğudur. Dolayısıyla burada bilindiği anlamıyla bir matris çarpımı değil şu şekilde çalışan farklı bir matris çarpımı yer almaktadır: 0 ile çarpılan herhangi bir sembol sonuç vermiyor kabul edilir, '1' ile çarpılan sembolün ise kendisini verdiği kabul edilir. Her satır sadece bir tane '1' içerdiğinden herhangi bir toplamaya gerek yoktur, çarpım matrisinin ilgili elemanı doğrudan 1 ile çarpılan sembol olur.

7. satırdaki eşitlik koşulu tam bir eşitlik işleci olarak ele alınırsa, Ullman'ın algoritması birebir çizge içerme problemini de çözebilmektedir. Bu tezde sadece gömülü ağaç eşlemeler dikkate alındığından bu durum ihmal edilmiştir.

Algoritma 9 Çizge İçerme(M_H, M_G, P, k)

Girdi: M_H ve M_G bitişiklilik matrisleri, P permütasyon matrisi, k anlık satırı
Çıktı: $\mathcal{P}$ permütasyon listesi.

1: **if** $k > m$ **then**
2: $\mathcal{P} \leftarrow \mathcal{P} \bigcup P$ ▷ bir çizge içerme bulundu.
3: **return**
4: **end if**
5: **for** $i \leftarrow 1$ to n **do**
6: $P_{ki} \leftarrow 1$ and $P_{kj} \leftarrow 0. \forall j \neq i$
7: **if** $M_H \preceq (P_{[1..k],[1..n]} * M_G * (P_{[1..k],[1..n]})^T)$ **then**
8: **Çizge İçerme**(M_H, M_G, P, $k+1$) ▷ parçalı eşlemeden devam et.
9: **end if**
10: **end for**

Çizelge 5.1 Şekil 5.1'daki veri çizgesinin ve Şekil 5.2(b)'deki örüntü çizgesinin bitişiklilik matrislerini göstermektedir. Düğümler koyu etiketlerle, düğümler arası kenar olmaması durumu ise '-' sembolü ile gösterilmiştir. Diğer semboller ise kenar etiketini göstermektedir. Bu çizgelerden veri çizgesinin örüntü çizgesiyle eşlenen dört farklı permütasyonu vardır. Bu dört eşleme Çizelge 5.2'de görülmektedir. Burada solda yer alan matrisler örüntü çizge ile eşlenen veri ağacı altçizgesinin bitişiklilik matrisini,

sağdakiler de ilgili permütasyon matrisinin eşleme anındaki durumunu göstermektedir. Dikkat edilirse veri çizgesinin altçizgelerinin bitişiklilik matrisleri örüntü çizgesinin bitişiklilik matrisine tamamen eşit değildir. Çünkü burada ele alınan gömülü çizge eşlemedir dolayısıyla örüntü çizge bitişiklilik matrisindeki '-' sembollerinin karşılık gelen altçizge bitişiklilik matrisi sembolüyle karşılaştırılması gereksizdir. Eğer karşılaştırılsaydı, sadece birebir altçizge eşlemeleri bulunurdu ki burada da sadece 1 tane mevcuttur.

Çizelge 5.1: Bitişiklilik matrisleri: (a) Şekil 5.1'da verilen çizgenin bitişiklilik matrisi, (b) Şekil 5.2(b)'de verilen çizgenin bitişiklilik matrisi

(a)

	1(A)	2(B)	3(C)	4(B)	5(C)
1(A)	**A**	X	Z	X	Z
2(B)	X	**B**	Y	-	-
3(C)	Z	Y	**C**	Z	-
2(B)	X	-	Z	**B**	Y
3(C)	Z	-	-	Y	**C**

(b)

	1(A)	2(C)	3(B)
1(A)	**A**	Z	X
2(C)	Z	**C**	-
3(B)	X	-	**B**

5.3 Altçizge Gizleme

Çizge gizlemesinin yapılmış olması için örüntü çizgelerinin destek değerlerinin eşik değerinden küçük olması gerekir. Gizlenecek örüntüler ($P \in \mathcal{P}_h$) genellikle eşik değerinden daha büyük destek değerine sahip olduğundan herhangi bir P çizgesi için P'yi destekleyen veri çizgeleri arasından $sup_{\mathcal{D}'}(P) - \psi + 1$ kadarının seçilmesi gerekir (global çözüm). En sıradan durum rastgele seçimdir fakat bu durum problemin ikinci gereğini pek de karşılamayabilir. Bu yüzden seçilecek veri çizgelerinde en az maskeleme ile en çok hassas çizge gizlemek global çözüm olarak uygundur.

Seçilen çizgeler eğer örüntü çizgeyi bir kez içeriyorsa herhangi bir düğümün maskelenmesi gizleme için yeterlidir. Fakat çoğu zaman bir veri çizgesi bir örüntü çizgesini çok sayıda içerir, dahası çoğu düğüm birden fazla içerilme durumlarına katkıda bulunur. Tüm bu içerilme durumlarına katkıda bulunan hangi düğümün seçileceği lokal çözüme karşılık gelir.

Çizelge 5.2: Şekil 5.1'deki çizgenin Şekil 5.2(b)'deki altçizgeyi 4 farklı şekilde içerme durumu

(a)

	1(A)	3(C)	2(B)
1(A)	**A**	Z	X
3(C)	Z	**C**	Y
2(B)	X	Y	**B**

1	0	0	0	0
0	0	1	0	0
0	1	0	0	0

(b)

	1(A)	3(C)	4(B)
1(A)	**A**	Z	X
3(C)	Z	**C**	Z
4(B)	X	Z	**B**

1	0	0	0	0
0	0	1	0	0
0	0	0	1	0

(c)

	1(A)	5(C)	2(B)
1(A)	**A**	Z	X
5(C)	Z	**C**	-
2(B)	X	-	**B**

1	0	0	0	0
0	0	0	0	1
0	1	0	0	0

(d)

	1(A)	5(C)	4(B)
1(A)	**A**	Z	X
5(C)	Z	**C**	Y
4(B)	X	Y	**B**

1	0	0	0	0
0	0	0	0	1
0	0	0	1	0

Geliştirilen çizge gizleme algoritması ağaç gizleme algoritması ile yapısal olarak neredeyse aynıdır ve Algoritma 10'da sözde kodu verilmiştir.

Ağaç gizleme de olduğu gibi, çizge gizlemede de global ölçekte seçilen sezgisel önce en az sayıda eşleme içeren çizgelerin sterilize etme işlemine tabi tutulmasıdır. Fakat yerel ölçek için geliştirilen ve aşağıda açıklanan 3 farklı sezgisel mevcuttur.

Çizge gizleme algoritması ağaç gizleme algoritmasından sadece 9. satırda farklılaşmaktadır. Burada mevcut çizgede yerel ölçekli çözüm gerçekleştirilmektedir. Yerel ölçekli çözüm olarak 3 farklı sezgisel geliştirilmiştir.

Algoritma 10 Çizge Gizleme Algoritması

Girdi:$\mathcal{D}$ çizgeler veritabanı, $\mathcal{P}_h$ hassas çizgeler kümesi, ψ eşik değeri
Çıktı:$\mathcal{D}'$ sterilize edilmiş çizgeler veritabanı

1: $\mathcal{D}' \leftarrow \mathcal{D}$
2: $c_G(\mathcal{P}_h) = \Sigma_{P \in \mathcal{P}_h} c_G(P)$' yi hesapla
3: $\mathcal{D}'$ yi $c_G(\mathcal{P}_h)$'in artan sırasına göre sırala
4: $wsup_{\mathcal{D}'}(P), \forall P \in \mathcal{P}_h$'yi hesapla
5: $\mathcal{P}_h$'yi $wsup_{\mathcal{D}'}(\mathcal{P}_h)$'nin artan sırasına göre sırala
6: **for all** $P \in \mathcal{P}_h$ **do**
7: **while** $sup_{\mathcal{D}'}(P) \geq \psi$ **do**
8: $P \preceq G'$ 'yi sağlayan sıradaki $G' \in \mathcal{D}'$'yi al
9: $G' \leftarrow LocalHeuristic(G', P)$
10: $sup_{\mathcal{D}'}(P) \leftarrow sup_{\mathcal{D}'}(P) - 1$
11: **end while**
12: **end for**

5.3.1 Yerel Sezgisel 1

Algoritma 11 Yerel Sezgisel 1

Girdi:G veri çizgesi, H örüntü çizgesi.

1: V_H'nin V_G'de eşlenebilecekleri düğüm sayılarını ayrı ayrı bul.
2: $lbl \leftarrow minFrequent(V_H)$'yi hesapla.
3: **for all** $v \in V_G$ **do**
4: **if** $label(v) = label(lbl)$ **then**
5: v'yi maskele.
6: **end if**
7: **end for**

Yerel Sezgisel 1, Algoritma 11'de verilmiştir. Bu sezgisel en basit olan sezgiseldir ve ağırlıklı destek değerine bakmaksızın sadece altçizge içerme durumunu dikkate almaktadır. Eğer G veri çizgesi, H örüntü çizgesini destekliyorsa, sezgisel ilk olarak H çizgesindeki farklı etiketli düğümleri bulur ve bu düğümlerle aynı etiketi taşıyan düğümleri ayrı ayrı her düğüm için sayar (1. satır). Örnek olarak G de en az sayıda bulunan P düğümü lbl olsun (2. satır). Buna göre sezgisel lbl ile aynı etikete sahip olan tüm G düğümlerini maskeler (5. satır). Maskeleme işlemi sonucunda G'nin H'yi hiçbir şekilde desteklemediği açıktır. Çünkü G de lbl ile eşlenebilecek herhangi bir düğüm artık kalmamıştır. Bu sezgisel oldukça etkindir, çünkü Algoritma 9'da 2. satıra ilk geldiğinde durur ve tüm $\mathcal{P}$'nin hesaplanmasını gerekli kılmaz. Ek olarak etiket sayma işlemi ve maskeleme işlemi de G'deki düğüm sayısıyla orantılı olarak doğrusal

zamanda yapılabilir.

5.3.2 Yerel Sezgisel 2

Algoritma 12 Yerel Sezgisel 2

Girdi:$\mathcal{P}$

1: **while** $\mathcal{P} \neq \emptyset$ **do**
2: $kurban \leftarrow \mathcal{P}$'de en çok tekrar eden düğüm.
3: $kurban$'ı maskele.
4: $\mathcal{P}$'den $kurban$ı içeren eşlemeleri sil.
5: **end while**

Yerel Sezgisel 2 Algoritma 12'de verilmiştir. Bu sezgisel tüm farklı altçizge eşlemelerin listesi yani $P \in \mathcal{P}$' leri esas almaktadır. Bulunan $\mathcal{P}$ eşleme kümesinden yararlanan sezgisel ilk olarak $P \in \mathcal{P}$ lerde en çok sayıda geçen düğümü bulur (2. satır). Bir an için bu düğümün v olduğunu düşünelim. Sonra G'de bu düğüm maskelenir (3. satır) ve devamında v'yi içeren tüm permütasyonlar (P) $\mathcal{P}$'den atılır (4. satır). Sonra aynı işlem güncel $\mathcal{P}$ için sırasıyla $\mathcal{P}$ boş küme olana kadar tekrar edilir (1.-5. satırlar). Bu sezgisel doğru bir biçimde çalışır çünkü her iterasyondan sonra en az bir eşleme elenmiş olur. Bu sezgiselin etkinliğini azaltan faktör $\mathcal{P}$'nin tamamen hesaplanmasıdır. Bunun dışında diğer hesaplama adımları etkin bir şekilde gerçekleştirilebilmektedir.

5.3.3 Yerel Sezgisel 3

Algoritma 13 Yerel Sezgisel 3

Girdi:$\mathcal{P}$

1: **while** $\mathcal{P} \neq \emptyset$ **do**
2: $freqList \leftarrow \mathcal{P}$'deki düğümlerin listesi.
3: $freqList$'deki düğümleri $\mathcal{P}$'deki frekanslarına göre azalan sırada sırala.
4: $kurbanListesi \leftarrow \emptyset$
5: **while** $|\mathcal{P}| > kurbanListesi$'ndeki ögelerin frekans toplamı **do**
6: $freqList$'ten sıradaki düğümü $kurbanListesi$'ne at.
7: **end while**
8: $kurbanListesi$'ndeki tüm düğümleri maskele.
9: $\mathcal{P} \leftarrow$ Algoritma 9
10: **end while**

Yerel Sezgisel 3 Algoritma 13'de verilmiştir. Sezgisel 2'ye benzer olarak bu sezgisel

de tüm farklı altçizge eşlemelerinin kümesini esas alır. Hesaplanan $\mathcal{P}$ kümesinden yararlanarak, permütasyonlarda yer alan her farklı düğümün freakansları hesaplanır (2. satır). Tüm düğümler frekanslarının azalan sırasına uygun olarak sıralanırlar (3. satır). Sıralanmış düğümlerden en tepedeki i kadar düğüm G'de maskelenmek üzere seçilir. i değeri öyle seçilmelidir ki, seçilen düğümlerin frekansları toplamı en az $|\mathcal{P}|$ kadar olmalıdır (4.-7. satırlar). Buradaki önsezi, amaç $|\mathcal{P}| = 0$ yapılmak istendiğinden, seçilen i kadar düğümün maskelenmesiyle amacın gerçekleştirilebilmiş olma ihtimalidir. Seçilen i kadar düğüm G'de maskelenir (8. satır) ve $\mathcal{P}$ tekrardan hesaplanarak (9.satır) aynı prosedür yeni $\mathcal{P}$'ye uygulanır. $\mathcal{P}$ boş küme olana dek bu durum devam ettirilir. Burada $\mathcal{P}$ nin tekrar tekrar hesaplanması, her ne kadar, seçilen i kadar düğümün maskelenmesiyle veri çizgesinin örüntü çizgesini içermemesi umulmuş olsa da bunun garanti edilebilir olmayışından kaynaklanır. Görüldüğü gibi bu sezgisel Ullman algoritmasının birden çok defa çağırılmasını gerektirebilmektedir ki bu durum da olumsuz anlamda etkinlik açısından önemli bir durumdur. Aslında bu sezgisel i kadar düğüm seçilerek değil de her seferinda ilk düğüm seçilerek de tasarlanabilirdi, fakat bu durum da Ullman algoritmasının daha da fazla çalışmasına neden olurdu ki etkinlik açısından tercih edilebilir bir durum değildir. Diğer yandan, eğer böyle tasarlansaydı muhtemelen daha etkili bir sezgisel olurdu. Dolayısıyla i kadar düğümün seçilmesiyle hızlılık/etkililik dengesi gözetilmiştir.

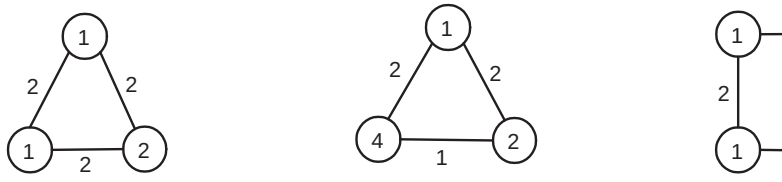
5.4 Çizge Gizleme Performans Sonuçları

Çizge gizleme algoritmsı da ağaç gizleme algoritması gibi Java ile gerçekleştirilmiştir. Yine ağaçlarda olduğu gibi performans metrikleri olarak çalışma zamanı, maskelenen düğüm sayısı $M0$ ve veritabanı gizlemeden sonraki sık çizgelerin veri gizlemeden önceki sık çizgelere oranı $M1$ metrikleri kullanılmıştır. Algoritmanın performans testleri için iki farklı veritabanı belirlenmiştir. Bunlardan birincisi `SYNTHETIC` veritabanı ve ikincisi de `CHEMICAL` [38] veritabanıdır. İlk veritabanı yapay bir veritabanıdır ve *GraphGen* [12] isimli çizge oluşturucu kullanılarak üretilmiştir. İkinci veritabanı ise kimyasal bileşenlerden oluşan gerçek bir veritabanıdır. *SYNTHETIC* veritabanı 1000 tane çizge içermektedir ve her çizgede ortalama 10 düğüm 20 de kenar bulunmakta-

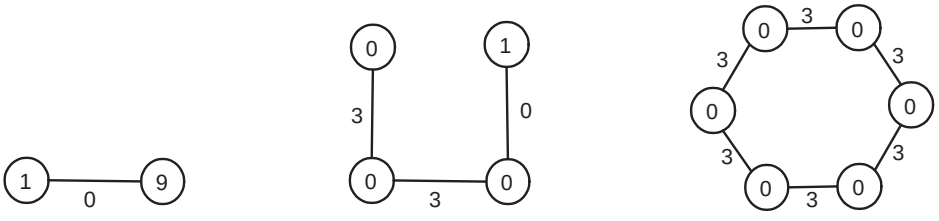
dır. Düğümler etiketlerini 5 elemanlı bir alfabeden, kenarlar da etiketlerini 4 elemanlı bir alfabeden almaktadırlar. CHEMICAL veritabanı 340 tane çizge içermektedir. Her çizgenin ortalama 27 düğümü ve 27,4 de kenarı bulunmaktadır. Düğüm etiketleri 66 elemandan oluşan bir alfabeden, kenar etiketleri de 4 elemanlı bir alfabeden gelmektedir. CHEMICAL veritabanı her ne kadar bilgi gizleme için uygun bir veritabanı olmasa da, üzerinde veri madenciliği yapılabiliyor olması nedeniyle ve geliştirilen çizge algoritmasının performansını ölçmek amacıyla tercih edilmiştir.

Bu iki veritabanı için gizlenecek örüntüler, sık çizge madenciliği yapılarak, sık altçizgeler arasından seçilmiştir. Her veritabanı için de 3'er tane örüntü seçilmiştir ve sırasıyla *örüntü1*, *örüntü2* ve *örüntü3* olarak adlandırılmıştır. Bu örüntüler Şekil 5.3 ve 5.4'de görülmektedir. Sık altçizge madenleme için ise *Gaston* [30] isimli araç kullanılmıştır.

Performans testlerinde her örüntü için değişen eşik değerlerine karşın kullanılan üç farklı yerel sezgiselin performansı ölçülmüştür.

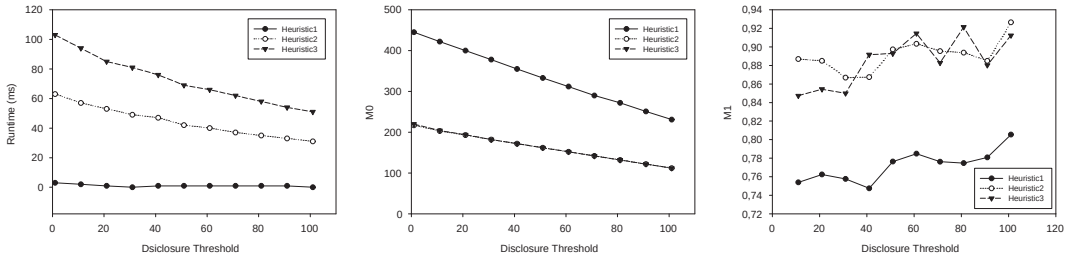


Şekil 5.3: SYNTHETIC veritabanında gizlenmek için seçilmiş örüntüler

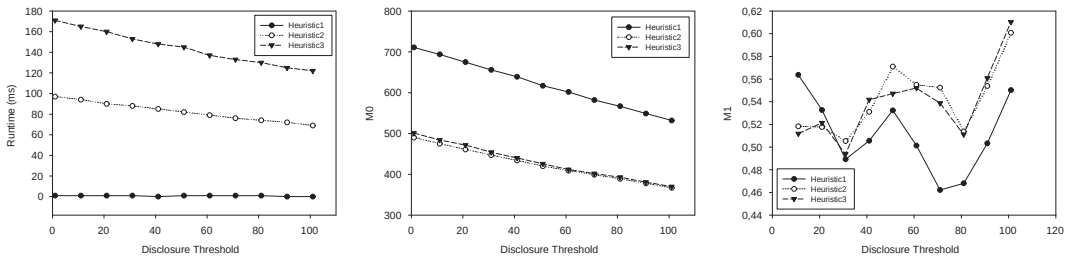


Şekil 5.4: CHEMICAL veritabanında gizlenmek için seçilmiş örüntüler

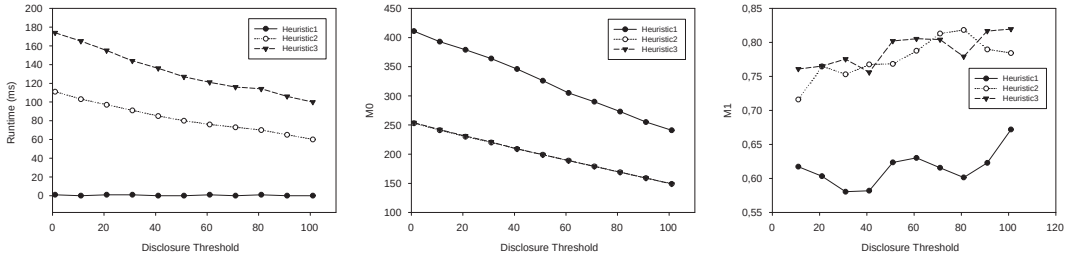
Şekil 5.5'de SYNTHETIC veritabanında seçilen 3 farklı örüntü çizgelerin gizlenmesiyle elde edilen sonuçlar görülmektedir. Yerel sezgisellerin çalışma zamanlarına bakıldığında Yerel Sezgisel 1 daha hızlı çalışmaktadır. Bu olağan bir sonuçtur, çünkü bu



örüntü1

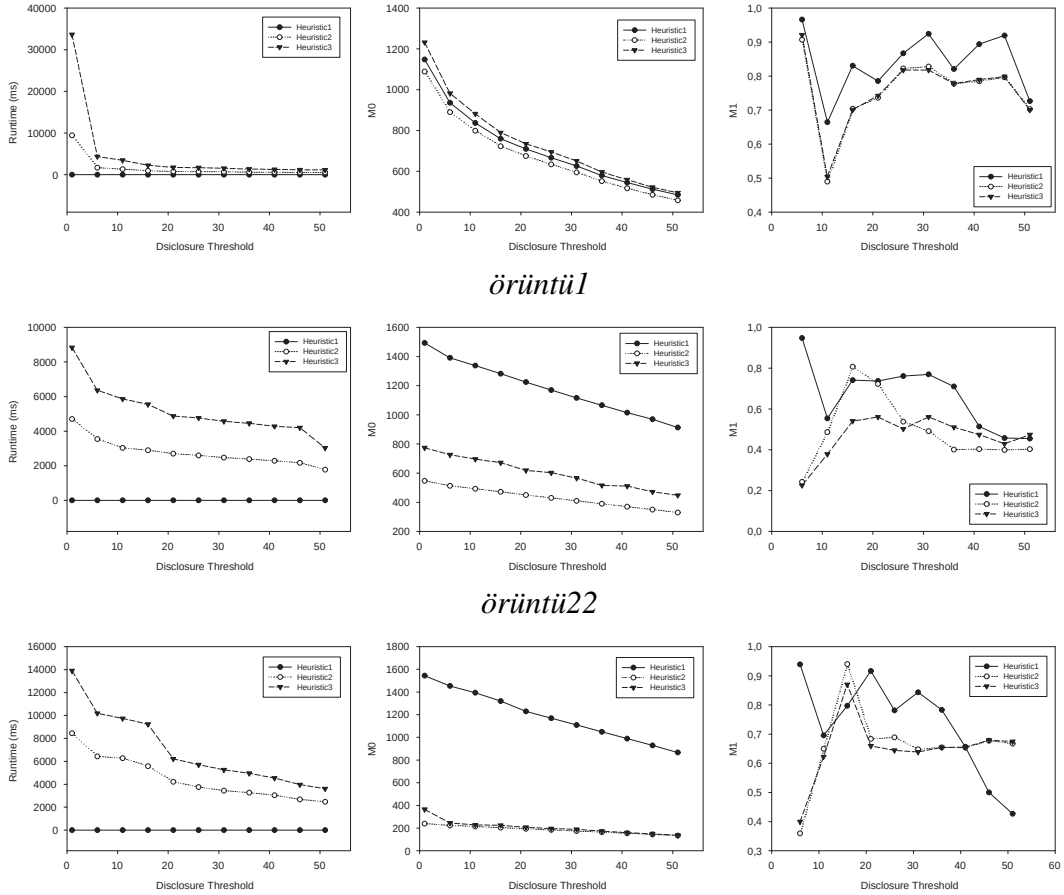


örüntü2



örüntü3

Şekil 5.5: SYNTHETIC veritabanının performans sonuçları



Şekil 5.6: CHEMICAL veritabanının performans sonuçları

sezgisel gizleme esnasında Ullman algoritmasına ihtiyaç duymamaktadır. Daha sonra en hızlı olan ise Yerel Sezgisel 2'dir. Bu sezgiselin Yerel Sezgisel 3'ten daha hızlı olması da olağandır. Çünkü burada Ullman algoritması sadece bir kere çağırılırken, Yerel Sezgisel 3'te daha fazla çağrılabilmektedir. $M0$ metriğine bakıldığında en başarısız olan Yerel Sezgisel 1; en başarılı olan ise Yerel Sezgisel 2'dir. Yerel Sezgisel 3 ise Yerel Sezgisel 2'den kötüdür fakat fazla da bir fark yoktur.$M1$ metriğine göre ise Yerel Sezgisel 2 ve Yerel Sezgisel 3 yakın sonuçlar verirken yerel Sezgisel 1 kötü sonuçlar vermektedir.

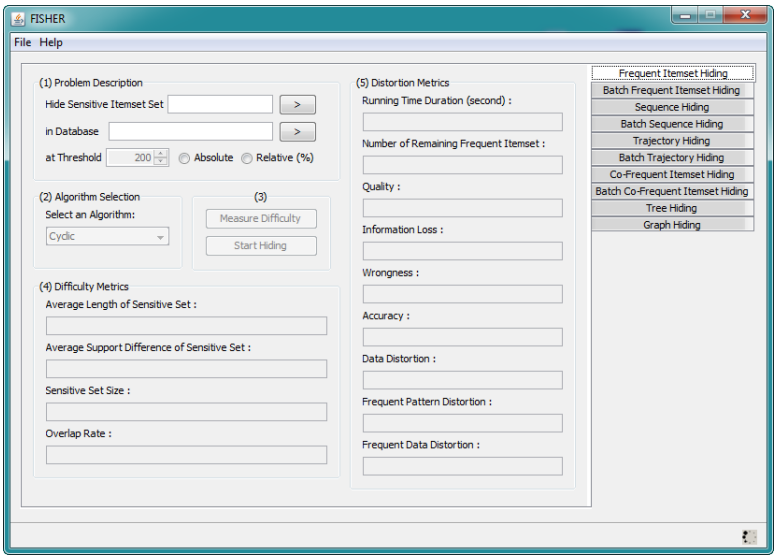
CHEMICAL veritabanı için performans testleri ise Şekil 5.6'de gösterilmiştir. Bu veritabanının performans sonuçları da SYNTHETIC veritabanının gösterdiği sonuçlarla örtüşmektedir.

6 FISHER UYGULAMASI

Tez kapsamında sık öge kümeleri gizleme algoritmaları, dizgi gizleme algoritmaları ve geliştirilen ağaç gizleme algoritması ile çizge gizleme algoritmasının yer aldığı bütüncül bir veri gizleme uygulaması (FISHER) geliştirilmiştir. Her veri tipindeki gizleme için ayrı sekmeler ayrılmış bu sekmelerin ait olduğu veri türleri için veri gizleme ekranları tasarlanmıştır.

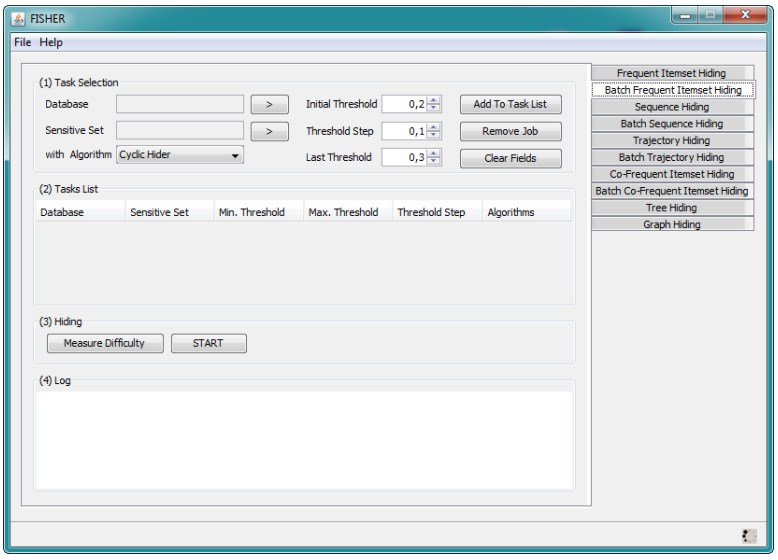
FISHER uygulaması Java programlama dili ile Netbeans IDE ile geliştirilmiştir. Jar dosyası haline getirilerek taşınması ve çalıştırılması kolaylaştırılmıştır. Hassas bilgi gizleme amacıyla Java 5 ve 6 kurulu olan tüm platformlarda çalışabilmektedir. Yalnız uygulama, problem tiplerine göre tanımlı olan metriklerin ölçümü için başkalarınca yazılmış veri madenciliği uygulamalarına ihtiyaç duymaktadır. Bu veri madenciliği uygulamaları ise Windows ortamında derlenmişlerdir. Bu yüzden metrik ölçümlerinin yapılması için bu veri madenciliği uygulamalarının çalıştırılabilir dosyaları FISHER uygulamasıyla aynı dizinde yer almalıdır ve uygulama Windows ortamında çalıştırılmalıdır.

Devamda FISHER uygulaması ekran ekran anlatılmaktadır.



Şekil 6.1: Sık öge kümesi gizleme ekranı

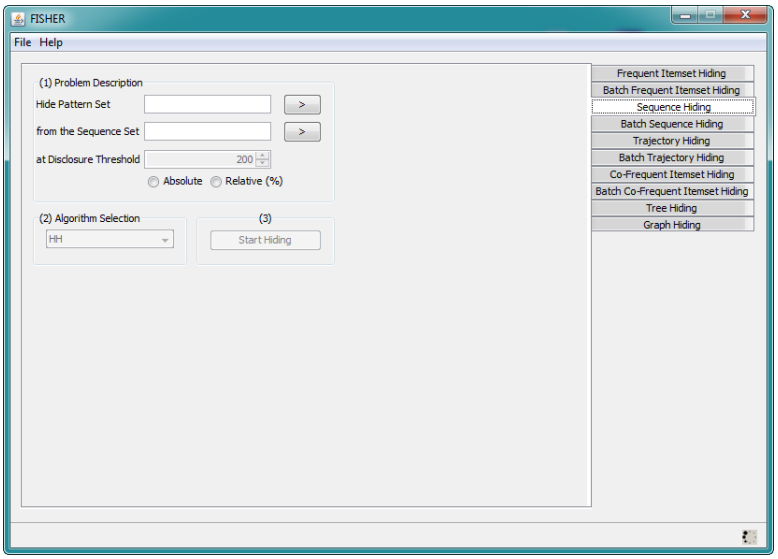
Şekil 6.1'de sık öge gizleme ekranı görülmektedir. Bu ekranda gizleme yapmak için gizleme yapılacak veritabanı dosyası, gizlenecek hassas örüntü kümesi dosyası ve eşik değeri seçilmektedir. Eşik değeri iki farklı şekilde seçilebilir: İlki mutlak eşik değeri (bir tamsayı), ikincisi ise göreceli eşik değeridir. Göreceli eşik değeri bir yüzde oranıdır ve mutlak olarak hangi değere karşılık geldiği veritabanındaki işlem sayısıyla çarpılarak hesaplanmaktadır. Bu ekranda gizlemede kullanılacak algoritmanın da seçilmesi gerekmektedir. Seçilebilecek algoritmalar: Cyclic Hider [9], FHFSI [45], Border Based Hider [39], Matrix Hider [24] ve geliştirilen Dengeli Gizleme Algoritmasıdır. Gizleme işlemi yapılmadan gizleme işleminin zorluğuna dair bilgi almak mümkündür. Gizlenecek örüntülerin ortalama uzunluğu, ortalama destek değeri ve toplam örüntü sayısının fazlalığı ile örüntülerde ortak olan ögelerin azlığı problemin zorluğunu arttırır. Gizleme sonucunda Bölüm 3.2.1'de tanımlı olan metrik sonuçları ekrana yazdırılır. Gizleme sonucunda sonuç veritabanı, girdi veritabanının bulunduğu dizine ayırt edici bir isimle kaydedilmektedir.



Şekil 6.2: Toplu sık öge kümeleri gizleme ekranı

Şekil 6.2'de toplu öge kümesi gizleme ekranı verilmiştir. Bu ekranda her bir gizleme işlemi bir görev olarak yer almaktadır. Bir görev gizleme yapılacak veritabanı dosyasına, gizlenecek hassas öge kümeleri dosyasına, başlangıç eşik değerine, bitiş eşik değerine, eşik değeri artış miktarına ve kullanılacak algoritmaya ihtiyaç duyar. Bunlar sağlanarak oluşturulmuş görevler görev listesine atılır ve bu görevler sonra teker teker yerine

getirilirler. Burada da her görevin zorluğu ölçülebilir ve uygulama tarafından ayrı bir dosyaya yazılır. Ekranın altında ise anlık olarak hangi görevlerin çalıştırıldığına dair bilgi verilmektedir.

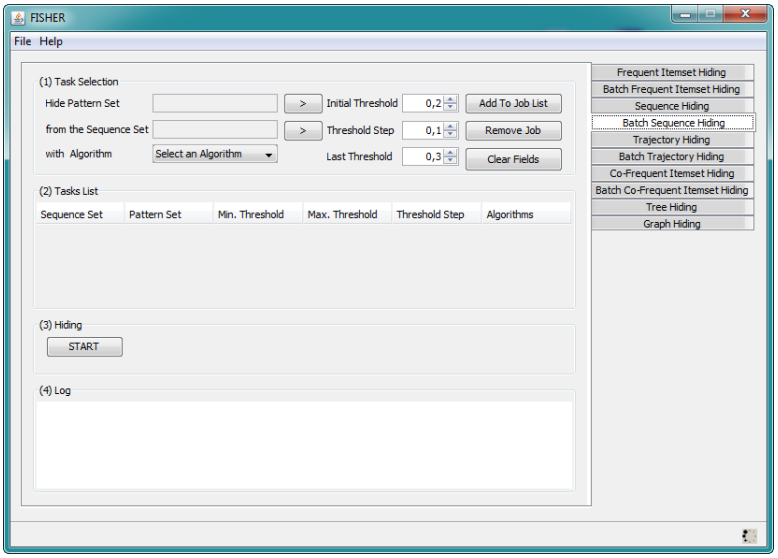


Şekil 6.3: Dizgi gizleme ekranı

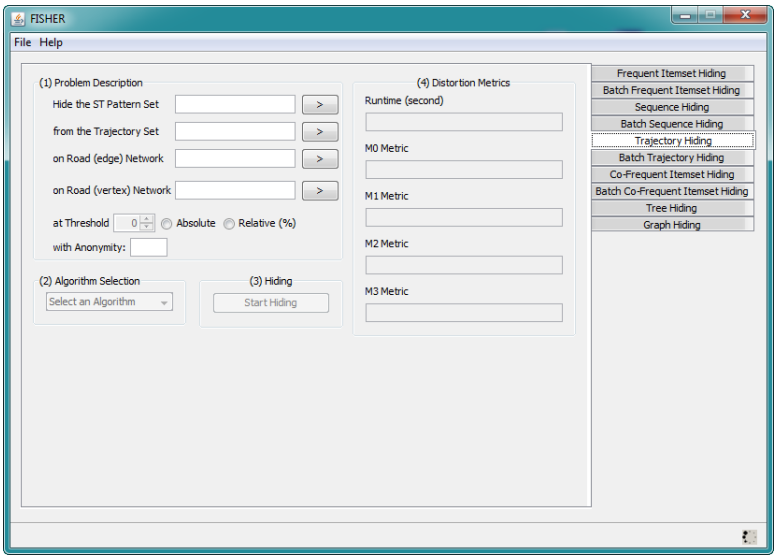
Dizgi gizleme ekranı Şekil 6.3'de gösterilmiştir. Bu ekranda gizlenecek örüntü dizgilerinin dosyası, gizleme yapılacak dizgiler dosyası, eşik değeri ve algoritmanın seçilmesi gerekir. Eşik değeri öge kümesi gizleme ekranındakine benzer olarak mutlak ya da göreceli olarak seçilebilmektedir. Algoritma olarak da [2, 3] numaralı çalışmalardaki HH, HR, RH, RR algoritmaları seçilir ve gizleme yapılır. Gizleme sonucunda oluşan veritabanı otomatik olarak kaydedilir.

Şekil 6.4'de hassas dizgi gizleme işleminin topluca yapılabilmesine olanak sağlayan ekran görüntüsü yer almaktadır. Her gizleme işlemi bir görev olarak tanımlanmaktadır. Her bir görev için gizleme yapılacak dizgiler veritabanı dosyası, gizlenecek örüntü dizgiler kümesi dosyası, kullanılacak algoritma, başlangıç eşik değeri, bitiş eşik değeri ve eşik değeri artış miktarı seçilmektedir. Oluşturulan görevler görev kuyruğuna atılmakta ve *gizle* komutu ile görevler çalıştırılarak sonuçları dosyalara kaydedilmektedir.

Şekil 6.5'de zaman-mekan izleri taşıyan iki boyutlu dizgi tipindeki verileri gizleme ekranı verilmiştir. Diğer ekranlarda olduğu gibi gizleme yapılacak veritabanı, gizlenecek hassas dizgiler dosyası, gizlemenin üzerinde yapılacağı çizge tipindeki yörüb-



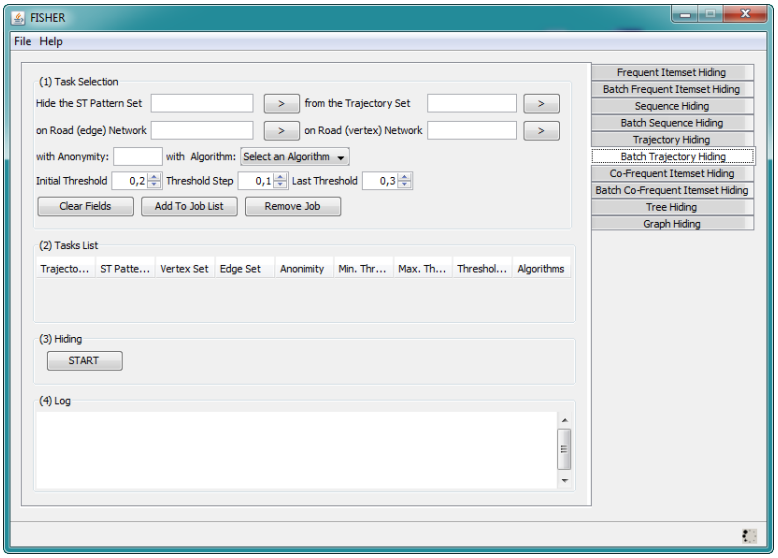
Şekil 6.4: Toplu dizgi gizleme ekranı



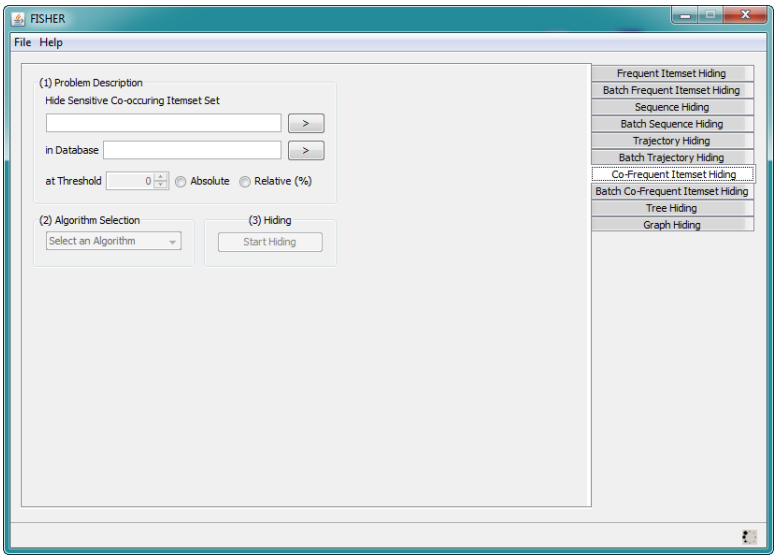
Şekil 6.5: Zaman-mekan izleri gizleme ekranı

gelerin düğümler kümesi ve kenarlar kümesi ile anonimlik değeri seçilmektedir. Eşik değeri hem mutlak hem de göreceli olarak girilebilmektedir. Seçilebilen 4 tane algoritma vardır: HH, HR, RH, RR [3].Gizleme yapıldıktan sonra ölçülen metrikler ekrana yazdırılmaktadır. Şekil 6.6'de ise zaman-mekan izleri taşıyan veritabanlarında gizleme topluca yapılabilmektedir. Aynen dizgi gizlemede olduğu gibi her gizleme işlemi bir görev olarak görev listesine atılmakta sonra da görevler teker teker çalıştırılmaktadır.

[4] numaralı çalışmada beraberce sık olan öge kümelerinin hassas olanlarının gizlen-

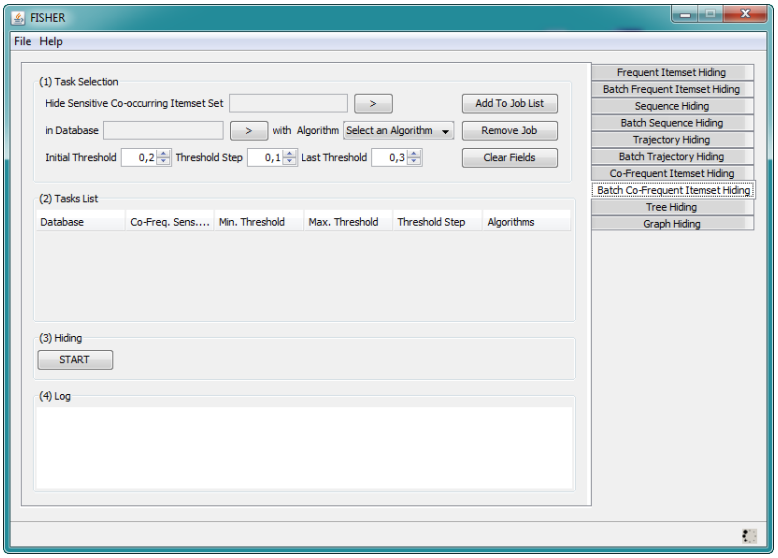


Şekil 6.6: Zaman-mekan izleri toplu gizleme ekranı

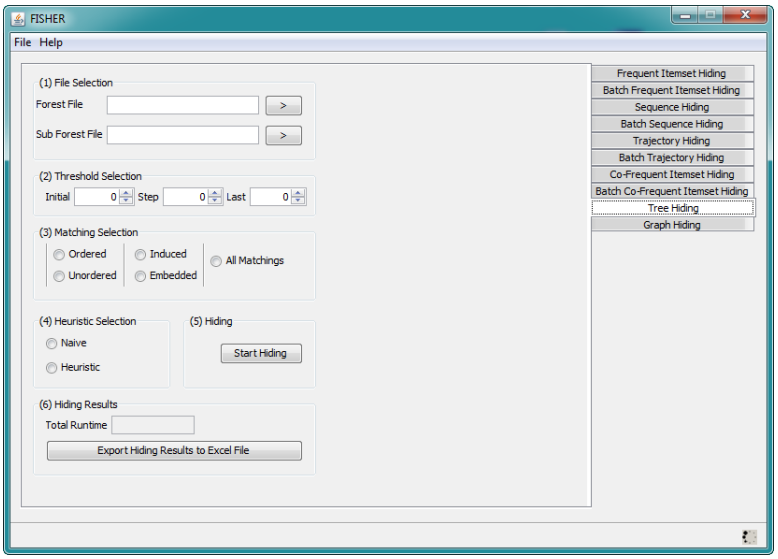


Şekil 6.7: Birlikte sık hassas öge kümelerini gizleme ekranı

mesi hakkında algoritmalar geliştirilmiştir. Bu algoritmalar da FISHER uygulamasına eklenmiştir ve örnek ekran görüntüsü Şekil 6.7'de gösterilmiştir. Gizleme işlemi için beraber sık olan sık öge kümeleri dosyası, veritabanı dosyası ve eşik değeri (mutlak ya da göreceli) seçilerek gizleme probleminin girdileri sağlanmaktadır. Algoritma olarak şu dört algoritmadan biri seçilebilmektedir: *HittingSetHide*, *PickMinHide*, *PickMinSupHide* ve *UnitAllHide*. Şekil 6.8'de ise beraber sık olan öge kümelerinin gizlenmesi topluca yapılmaktadır. Her gizleme işlemleri görev olarak oluşturulup görev listesine atılmaktadır. Sonra da görev listesindeki görevler teker teker çalıştırılmaktadır.

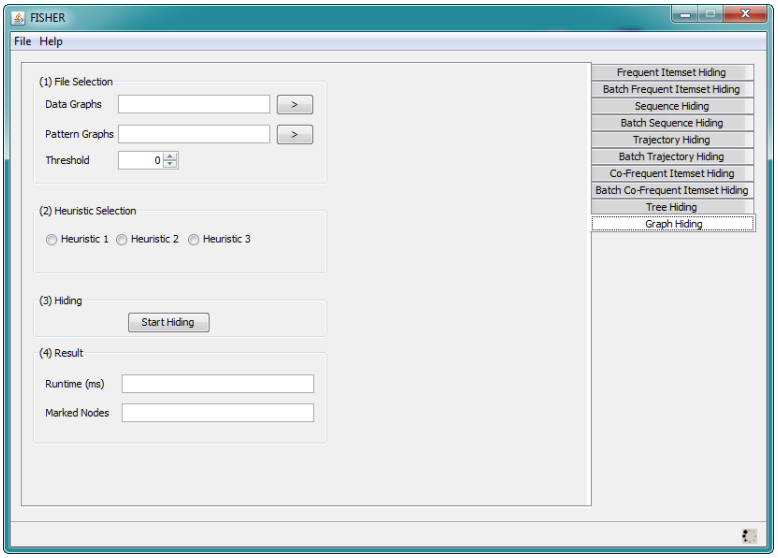


Şekil 6.8: Birlikte sık hassas öge kümelerini toplu gizleme ekranı



Şekil 6.9: Ağaç gizleme ekranı

FISHER uygulamasının ağaç gizleme ekranı Şekil 6.9'de görülmektedir. Ağaç gizleme işlemi yapmak için gerekli olan orman dosyası, örüntü ağaçlar dosyası, başlangıç eşik değeri, bitiş eşik değeri, eşik değeri artış miktarı, gizlemenin yapılacağı ağaç eşleme sınıfı ve gizleme sezgiseli seçilmelidir. Gerekli girdiler seçildikten sonra gizleme işlemi yapılır, sonuçlar dosyaya otomatik olarak kaydedilir. Gizlemeden sonra değişen eşik değerlerine ve eşleme sınıfına göre otomatik olarak hesaplanan çalışma zamanları ve $M0$ metrikleri bir Excel dosyasına kaydedilebilir.



Şekil 6.10: Çizge gizleme ekranı

Şekil 6.10'de FISHER uygulamasının çizge gizleme ekranı görülmektedir. Bu ekranda da diğer ekranlara benzer olarak çizgeler veritabanı dosyası, gizlenecek örüntüler dosyası, eşik değeri ve kullanılacak olan sezgisel seçilerek gizleme işleminin girdileri belirlenir. Gizleme yapıldıktan sonra sonuçlar otomatik olarak kaydedilir.

7 SONUÇ

Veritabanı yayıncılığı toplumsal ve bilimsel fayda açısından olması gerekli olan bir tutumdur. Bu amaçla kuruluşlar sahip oldukları veritabanlarını iyi niyetle yayınlarlar. Yayınlanan veritabanları üzerinde veri madenciliği teknikleri uygulandığında istatistikî olarak bazı bilgiler elde edilebilmektedir. Elde edilen bilgilerin bazıları veritabanı sahibi için, açığa çıkması pek de arzu edilmeyen bilgiler olabilir. Bu yüzden veritabanı sahipleri bir yandan veri yayıncılığının faydasıyla veritabanı yayınlama eğiliminde olurlar, diğer yandan da hassas bilgilerinin açığa çıkma ihtimali ile veritabanı yayınlamadan vazgeçme eğiliminde olabilirler. Bu durumda yapılacak olan veritabanlarından, veritabanı sahipleri tarafından açığa çıkmasından korkulan hassas bilgilerin elenmesidir. Böylece yayınlanan veritabanlarından veri madenciliği uygulamaları ile hassas olduğu düşünülen bilgilere ulaşma ihtimali ortadan kaldırılmış olur. Veritabanı sahipleri de kaygısızca veritabanlarını yayınlayabilirler.

Veritabanları yapıları itibariyle farklı biçimlerde bulunurlar. Bu tez kapsamında hassas bilgi gizleme yapısal veritabanları üzerinden düşünülerek anlatılmıştır. Yapısal veritabanlarından ilki öge kümesi biçiminde olan veritabanlarıdır. Bu tip veritabanları üzerinde hassas bilgi gizlemesinin ilk yapıldığı veritabanı türü olmuştur. Bunun için birçok uygulama geliştirilmiştir. Bu uygulamalar farklı değerlendirme metriklerine göre farklı sonuçlar vermektedir. Bu tezde de bu tip veritabanlarından hassas bilgi gizleme için hızlı ve etkili bir algoritma geliştirilmiştir.

Sık öge kümelerinden sonra çalışılan veritabanı türü dizgi tipindeki veritabanlarıdır. Bu veritabanı için hassas bilgi gizleme sık öge kümesi kadar çalışılmamış olsa da, hızlı ve etkili uygulamalar geliştirilmiştir. Dizgi tipindeki veritabanlarının çok boyutlu olabileceğinden hareketle de uygulamalar çeşitlendirilmiştir.

Literatürde ağaç ve çizge yapısındaki veritabanlarından hassas bilgi gizleme konusu henüz çalışılmamıştır. Fakat bu iki tip veritabanı için sık veri madenciliği uygulamaları geliştirilmiştir. Geliştirilen veri madenciliği uygulamalarının bu veritabanları üzerinde

çalıştırılmasıyla elde edilecek bilgilerin hassas olabileceği gerçeğinden hareketle bu tezde bu iki tip veritabanları için hassas gizleme problemi tanımlanmıştır. Gizlenecek hassas bilgiler ise veri madenciliği ile elde edilebilecek bilgiler olmuştur. Hassas bilgi gizlemesi için hem ağaç hem de çizge tipi veritabanlarından hassas bilgi gizleme algoritmaları geliştirilmiştir. Bu algoritmaların hızlı olduğu kadar etkili olmaları amaçlanmıştır. Yapılan performans testlerinde de görülmüştür ki, geliştirilen algoritmalar performans açısından gayet tatminkârdır.

Tez çalışması boyunca yapısal veritabanlarında hassas bilgi gizleme algoritmaları gerçekleştirilmiştir. Her farklı veritabanı türü için hassas bilgi gizleme yapabilecek bütüncül bir uygulama geliştirilmiştir. Bu uygulama Java ile geliştirilmiş ve FISHER olarak adlandırılmıştır. Var olan veri madenciliği uygulamaları edinilmiş ve FISHER tarafından kullanılmak üzere kılıflandırılmıştır.

Zamanla farklı tipte yapısal veritabanları oluşacağından ve bu veritabanlarından veri madenciliği ile hassas bilgiler açığa çıkarılabileceğinden hassas bilgi gizlemenin bu yeni veritabanların üzerinde uygulanması söz konusu olacaktır. Dolayısıyla ilerde yapılabilecek çalışmalar arasında yeni oluşan veritabanlarında hassas bilgi gizleme algoritmalarının geliştirilmesi sayılabilir. Örnek olarak 3 boyutlu veritabanlarından hassas bilgi gizleme algoritmaları geliştirilebilir.

Yapılabilecek bir diğer çalışma ise zaten tanımlı olan veritabanlarından hassas bilgi gizleme yapabilecek daha farklı yaklaşımlar içeren etkili algoritmalar geliştirmektir.

Yeni algoritmalar geliştirildikçe, bu algoritmalar FISHER uygulamasına eklenebilir ve FISHER'ın bütüncül bilgi gizleme uygulaması olarak güncel tutulması sağlanabilir.

KAYNAKLAR

[1] O. Abul, M. Atzori, F. Bonchi, and F. Giannotti. Hiding sensitive trajectory patterns. In *6th Int. Workshop on Privacy Aspects of Data Mining (PADM'07), in conjunction with ICDM'07*.

[2] O. Abul, M. Atzori, F. Bonchi, and F. Giannotti. Hiding sequences. In *Third ICDE Int. Workshop on Privacy Data Management (PDM'07), in conjunction with ICDE'07*.

[3] O. Abul, F. Bonchi, and F. Giannotti. Hiding sequential and spatiotemporal patterns. *IEEE Transactions on Knowledge and Data Engineering*, 22(12):1709–1723, 2010.

[4] Osman Abul. Hiding co-occurring frequent itemsets. In *Proceedings of the 2009 EDBT/ICDT Workshops*, EDBT/ICDT '09, pages 117–125, New York, NY, USA, 2009. ACM.

[5] Osman Abul, Harun Gökçe, and Yagmur Sengez. Frequent itemsets hiding: A performance evaluation framework. In *ISCIS*, pages 668–673. IEEE, 2009.

[6] R.C. Agarwal, C.C. Aggarwal, and V.V. Prasad. A tree projection algorithm for generation of frequent itemsets. *Journal of Parallel and Distributed Computing*, 61:350–371, 2000.

[7] R. Agrawal, T. Imielienski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD'93)*, pages 207–216, 1993.

[8] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Eleventh International Conference on Data Engineering (ICDE'95)*, pages 3–14, Taipei, Taiwan, 1995.

[9] M. Atallah, E. Bertino, A. Elmagarmid, M. Ibrahim, and V. S. Verykios. Disclosure limitation of sensitive rules. In *KDEX'99*, pages 45–52, 1999.

[10] T. Brijs, G. Swinnen, K. Vanhoof, and G. Wets. Using association rules for product assortment decisions: A case study. In *Knowledge Discovery and Data Mining*, pages 254–260, 1999.

[11] Li Chen, Amarnath Gupta, and M. Erdem Kurul. Stack-based algorithms for pattern matching on dags. In *Proceedings of the 31th International Conference on Very Large Databases (VLDB 2005)*, pages 493–504, 2005.

[12] James Cheng, Yiping Ke, Wilfred Ng, and An Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD Conference*, pages 857–872, 2007.

[13] Jiefeng Cheng, Jeffrey Xu Yu, Bolin Ding, Philip S. Yu, and Haixun Wang. Fast graph pattern matching. In *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE 2008)*, pages 913–922, 2008.

[14] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *IJPRAI*, pages 265–298, 2004.

[15] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. Performance evaluation of the vf graph matching algorithm. In *ICIAP '99: Proceedings of the 10th International Conference on Image Analysis and Processing*, page 1172. IEEE Computer Society, 1999.

[16] E. Dasseni, V. S. Verykios, A. K. Elmagarmid, and E. Bertino. Hiding association rules by using confidence and support. In *Proceedings of the 4th International Workshop on Information Hiding*, pages 369–383, 2001.

[17] David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 632–640, 1995.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman, January 1979.

[19] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.

[20] Christoph M. Hoffmann and Michael J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[21] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

[22] Pekka Kilpelainen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, University of Helsinki, 1992.

[23] Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the First IEEE International Conference on Data Mining (ICDM 2001)*, 2001.

[24] G. Lee, C.-Y. Chang, and A. L. P. Chen. Hiding sensitive patterns in association rules mining. In *COMPSAC'04*.

[25] Hsiao-Tzu Lu and Wuu Yang. A simple tree pattern-matching algorithm. In *In Proceedings of the Workshop on Algorithms and Theory of Computation*, 2000.

[26] S. Menon, S. Sarkar, and S. Mukherjee. Maximizing accuracy of shared databases when concealing sensitive patterns. *Information Systems Research*, 16(3):256–270, 2005.

[27] B. T. Messmer and H. Bunke. Subgraph isomorphism in polynomial time. Technical report, University of Bern, 1995.

[28] Bruno T. Messmer and Horst Bunke. Efficient subgraph isomorphism detection: A decomposition approach. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):307–323, 2000.

[29] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the Tenth ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining (KDD 2004), pages 647–652, 2004.

[30] Siegfried Nijssen and Joost N. Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1):77 – 87, 2005. Proceedings of the International Workshop on Graph-Based Tools (GraBaTs 2004).

[31] D. E. O'Leary. Knowledge discovery as a threat to database security. In Gregory Piatetsky-Shapiro and William J. Frawley, editors, *Knowledge Discovery in Databases*, pages 507–516. AAAI/MIT Press, 1991.

[32] S. R. M. Oliveira and O. R. Zaïane. Protecting sensitive knowledge by data sanitization. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM 2003)*, pages 211–218, 2003.

[33] E. D. Pontikakis, A. A. Tsitsonis, and V. S. Verykios. An experimental study of distortion-based techniques for association rule hiding. In *Proceedings of the 18th Conference on Database Security (DBSEC 2004)*, pages 325–339, 2004.

[34] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB'95*, pages 432–444, 1995.

[35] Y. Saygin, V. S. Verykios, and C. Clifton. Using unknowns to prevent discovery of association rules. *ACM SIGMOD Record*, 30(4):45–54, 2001.

[36] Y. Saygin, V. S. Verykios, and A. K. Elmagarmid. Privacy preserving association rule mining. In *Proceedings of the 2002 International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE 2002)*, 2002.

[37] Dennis Shasha, Jason T. L. Wang, and Rosalba Giugno. Algorithmics and applications of tree and graph searching. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–52, 2002.

[38] A. Srinivasan, R.D. King, S.H. Muggleton, and M. Sternberg. The predictive toxicology evaluation challenge. In *IJCAI'95*, pages 1–6, 1997.

[39] X. Sun and P. S. Yu. A border-based approach for hiding sensitive frequent itemsets. In *ICDM'05*, pages 426–433, 2005.

[40] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty Fuzziness and Knowledge-based Systems*, 10(5), 2002.

[41] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE 2008)*, 2008.

[42] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[43] Gabriel Valiente and Conrado Martinez. An algorithm for graph pattern-matching. In *Proc. 4th South American Workshop on String Processing*, volume 8 of *International Informatics Series*, pages 180–197, 1997.

[44] V. S. Verykios, A. K. Elmagarmid, E. Bertino, Y. Saygin, and E. Dasseni. Association rule hiding. *IEEE Transactions on Knowledge and Data Engineering*, 16/4:434–447, 2004.

[45] C-C. Weng, S-T. Chen, and Y-C. Chang. A novel algorithm for hiding sensitive frequent itemsets. In *8th Int. Symposium on Advanced Intelligent Systems (ISIS'07)*, 2007.

[46] Hongquan Xu. Discovering knowledge in data: An introduction to data mining. *Journal of Statistical Software, Book Reviews*, 16(1):1–2, 5 2006.

[47] Xifeng Yan and Jiawei Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the Second IEEE International Conference on Data Mining (ICDM 2002)*, pages 721–724, Washington, DC, USA, 2002.

[48] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035, 2005.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı	:	GÖKÇE, Harun
Uyruğu	:	T.C.
Doğum tarihi ve yeri	:	01.10.1985 Malatya
Medeni Hali	:	Bekar
Telefon	:	05055919222
Email	:	hgokce@etu.edu.tr

Eğitim

Derece	**Eğitim Birimi**	**Mezuniyet Tarihi**
Lisans	TOBB ETÜ Bilgisayar Mühendisliği	2008

Yabancı Dil

İngilizce

Yayınlar

- Ö. Ö. Işıkman, M. Özdemir, H. Gökçe, B. G. Yalım, T. Özyer, R. Alhajj. "Adaptive Weighted Multi-Criteria Fuzzy Query Processing for Web Based Real Estate Applications", 23rd International ACM Symposium on Applied Computing (ACM SAC'08) - DTTA Track, Fortaleza, Brazil, March 2008.
- O. Abul, H. Gökçe, and Y. Şengez. "Frequent Itemsets Hiding: A Performance Evaluation Framework", 24th International Symposium on Computer and Information Sciences (ISCIS'09), September 2009, Cyprus.
- H. Gökçe and O. Abul. "A Tradeoff Balancing Algorithm for Hiding Sensitive Frequent Itemsets", Int. Conf. on Knowlede Discovery and Information Retrieval (KDIR'10), October 2010, Valencia.
- H. Gökçe and O. Abul. "Hassas Bilgi Gizleme Uygulaması", ELECO 2010, Aralık 2010, December 2010, Bursa, Türkiye.