XML Veritabanı için Tavlama Benzetimi ile Sorgu Optimizasyonu

Yaşar GÖZÜDELİ [1] M.Ali AKCAYOL [2]

[1,2] Gazi Üniversitesi, Mühendislik Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü, Maltepe, 06570, Ankara
ygozudeli@verivizyon.com, akcayol@gazi.edu.tr

Özet— Bu çalışmada tavlama benzetimi kullanılarak XML sorgu optimizasyonu gerçekleştirilmiştir. XML sorgulamada, sorgu ağacında yer alan düğümlerin birleştirilmesi gerekmektedir. Düğümlerin birleştirilme sıralaması sorgu maliyetini belirleyen en önemli etkendir. Bu çalışmada, düğümlerin sıralaması tavlama benzetimi ile gerçekleştirilmiştir. Uygulama C#.NET 2.0 dili ile hazırlanmış ve açık kaynak Timber XML veritabanı yönetim sistemi kullanılarak deneysel sonuçlar alınmıştır. Deneysel sonuçlar klasik yöntemlere göre daha kolay uygulanabilen tavlama benzetiminin XML sorgu optimizasyonunda başarılı olduğunu göstermiştir.

Anahtar kelimeler— XQuery, XML, sorgu optimizasyonu, tavlama benzetimi

XML Database Query Optimization using Simulated Annealing

Abstract— In this study, XML query optimization has been realized using simulated annealing. The nodes in query tree need to be joined for XML querying. The joining order of nodes is most important for the cost of query. In this study, the joining order has been done using simulated annealing. An application has been developed using C#.NET 2.0 programming language and experimental results have been obtained from open source Timber XML database management system. The experimental results show that simulated annealing, which is easier to apply than classical methods, has been successfully applied for XML query optimization.

Keywords— XQuery, XML, query optimization, simulated annealing

1. GİRİŞ

EXtensible Markup Language (XML) her ne kadar bir veri aktarım standardı olarak tasarlanmış[1] olsa da, aktarılan verinin kendi tanımını da içermesi nedeniyle bir veri saklama formatı olarak da kabul görmüş Standart Generalized Markup Language (SGML)[2] alt dilidir. Özellikle yarı yapılanmış veri saklama standardı olarak, İlişkisel Veritabanı Yönetim Sistemleri (İVTYS) tarafından da kullanılmaya başlanmıştır[3,4].

XML verinin sorgulanma ihtiyacı, XML'in veri aktarım formatı olarak kullanıldığı zamanlara dayanmaktadır. Bu çerçevede XML'in düğümler halinde sorgulanmasını sağlayan XPath[5] standart olarak World Wide Web Consortium(W3C) tarafından kabul edilmiştir. Ancak XML, zaman içerisinde veri aktarım dışında, veri depolama standardı olarak da kullanılmıştır. Özellikle yarı yapılanmış verilerin saklanmasında XML'in veri kullanılabilmesi için birçok sorgulama şekli önerilmiştir[5–8]. Bunlar arasında en yaygın kullanılan XPath'i de içine alan XML Query Language(XQuery) olmuştur[9]. XQuery henüz bir W3C standardı olma yolunda 'Candidate Recommendation' statüsünde bir standart taslağı olmasına rağmen, birçok akademik araştırmada [10,11] ve önde gelen VTYS üreticileri [3,12] tarafından tercih edilen bir XML sorgulama yöntemidir.

Öte yandan sorgu sürelerinin kestirimi ve sorgu iyileştirmesi, ilişkisel veritabanı yönetim sistemleri fikrinin ortaya çıktığı 1970'li yıllardan günümüze, ticari ve akademik araştırmalar için ilgi odağı olmuştur[13]. Bu konudaki tekniklerin gelişim süreci ve temel dayanakları Chaudhuri tarafından detaylı olarak incelenmiştir[14]. XQuery temelli olarak geliştirilen XVTYS'lerde İVTYS'ler için kullanılan bu yaklaşımlardan yararlanılmıştır[15].

İVTYS ortamındaki birçok akademik çalışma, XML ortamına da başarıyla aktarılmıştır. Ama XML ortamı yönlü grafların özel bir şekli olması nedeniyle ilişkisel ortamdaki bazı yöntemler XML sorgulama iyileştirilmesine başarı ile uygulanamamıştır. Bundan dolayı birçok yeni sorgu iyileştirme yöntemlerinin geliştirilmesi de gerekmiştir[10, 11, 15, 16]. İlişkisel ortam için Bennett ve arkadaşları tarafından İVTYS için iyi bir iyileştirme tekniği olarak sunulan en-soldan birleştirme(Left-Deep join veya Sistem-R) yaklaşımı, XQuery iyileştirmesinde başarılı olmamıştır[17]. Ancak XQuery optimizasyonu hakkında meta sezgisel yaklaşımların kullanıldığı çalışmalar bulunmaktadır. İVTYS sorgu iyileştirmesi ile ilgili yapılan bazı çalışmalar, meta sezgisel yaklaşım ile ilişkisel sorgu optimizasyonu yapılabileceğini gösteren örnekler olmuştur[15]. Steinbrunn ve arkadaşları tarafından yapılan çalışmada rastsal yöntemler ile birlikte tavlama benzetimi ve genetik algoritma kullanarak birleştirme sıralaması yapılmış ve özellikle çok sayıdaki tablolar için, deterministik yöntemlere göre daha başarılı sorgulama sonuçları elde edilmiştir[18].

Bu makalede, İVTYS için başarıyla uygulanmış olan tavlama benzetimi ile birleşim sıralaması (Join Order) sorununun XML temelli ortamda çözümü gerçekleştirilmiştir. Birleşim Sıralaması, XML ortamın düğümlerden oluşan yapısının depolanıp veri alınması sırasında tekrar bir araya getirilme zorunluluğu nedeniyle, İVTYS'ye göre çok daha fazla öneme sahiptir[19-22]. Çünkü artan sayıda birleşim gereksinimi ile birlikte deterministik yaklaşımlar yeterli performansı sağlayamamaktadır[17].

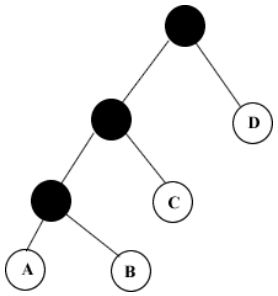
Bu çalışmada, XQuery sorgu iyileştirmesinde, ilk kez tavlama benzetimi kullanılmıştır. Yapılan çalışmada, sorguda kullanılacak düğümlerin öncelik sıralamaları ile ilgili iyileştirmeler için mantıksal sorgu planı tavlama benzetimi kullanılarak yapılmıştır.

2. SORGU OPTİMİZASYONUNDA BİRLEŞTİRME SIRALAMASI PROBLEMİ

Birden fazla tabloyu bir arada sorgulamak gerektiğinde tabloların birleşme işlemine katılım süreleri, sorgu iyileştirmesinde çok önemlidir. Birden fazla tabloyu birlikte sorgulamak için farklı birkaç birleştirme yaklaşımı kullanılmaktadır.

2.1. Sistem R iyileştirmesi veya left-deep join yaklaşımı

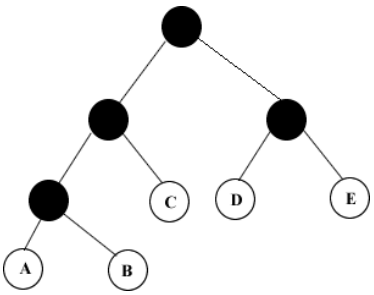
Sistem-R[23] iyileştirmesinde, çok fazla sayıda sorgu planı için kestirim yapmak yerine sadece soldan devam eden plan(left-deep-join) üstünden devam edilir. Bu yöntemin en büyük avantajı, bir önceki birleştirmeden elde edilen sonuçların bir sonraki birleştirmede doğrudan alınabilmesi nedeniyle geçici bir tabloya yazma ve tablodan okuma gerektirmemesidir. Dezavantajı ise, 16'dan fazla tablo içeren sorgularda tepki süresinin aşırı şekilde yavaşlayarak I/O maliyeti nedeniyle sonsuza yaklaşmasıdır[17]. Az sayıda tablo sorgularken belirgin bir I/O işlemi azalması ve budama nedeniyle de daha az olasılık üstünden daha kısa sürede sonuca ulaşılmasını sağlayan bir algoritmadır. Bir diğer faydası, birleştirmeye giren tablolar arasında kartezyen çarpımından kaçınılmasına olanak vermesidir. Şekil 1'de örnek Sistem-R sorgu için birleştirme görülmektedir.



Şekil 1. Örnek bir sorgu için Sistem-R birleştirme

2.2. Çalı ağacı çözüm uzayı

Çalı Ağacı birleştirmesinin Sistem-R'den en büyük farkı bir önceki sonuçların tutulabileceği bir geçici hafıza kullanımı gerektirmesidir. Sistem-R veya Left-Deep[23] yaklaşımı, iyi sonuç gelme olanağı olacak şekilde en soldan her seferinde bir tablo birleştirerek bütün tabloları eklemekte ve daha kötü sonuçlarla birlikte daha iyi sonuçların da gözden kaçırılmasına neden olabilen bir birleştirme yöntemidir. Şekil 2'de örnek bir çalı ağacı sorgu birleşimi görülmektedir.



Şekil 2. Örnek bir sorgu için çalı ağacı birleştirme

Olasılıklar da değerlendirildiğinde daha iyi birleştirme sıralamaları elde edilebileceği Ono ve arkadaşları tarafından ispat edilmiştir[24]. Ono ve arkadaşları tarafından yapılan araştırmaya göre, n birleşmeye katılacak tablo sayılarını göstermek üzere, $(n^3-n)/6$ farklı çalı ağacı birleştirmesi yapılabilmektedir. Ancak bu sayı Sistem-R için sadece $(n-1)^2$ ile sınırlıdır. Öte yandan bu çalışmada yapılan beşgen yıldız grafların sorgu maliyetlerinin en kötü durumunda bile kabul edilebilir olduğu gözlemlenmiştir. Beşgen yıldız ile taranabilen birleştirme uzayı ise $(n-1)2^{(n-2)}$ olarak hesaplanmıştır ki bu da Sistem-R'ye göre oldukça büyük bir tarama uzayının varlığına işaret etmektedir.

Vance ve arkadaşları tarafından yapılan, çalı ağacı birleşim sıralamasında kartezyen çarpımı yönteminin kullanılması bu alanda yapılmış deterministik bir çalışma olarak verilebilir[25].
Birleştirme uzaylarının oldukça fazla olması, deterministik ve deterministik olmayan çeşitli birleştirme uzayı tarama algoritmalarının geliştirilmesine neden olmuştur.

2.3. Birleştirme sıralaması problemine deterministik çözümler

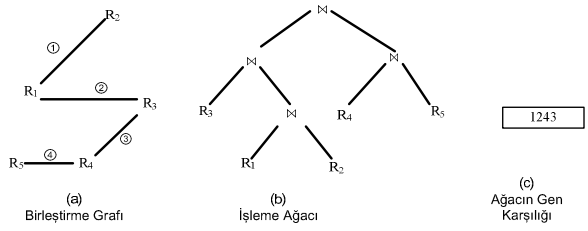
Bu alanda yapılan çalışmalardan en ünlüsü dinamik programlama[26], Sistem-R'deki ağaç yapısını soldan başlayarak üretir. Her seferinde bir yeni tablo ekleyerek çözüme ilerler. Bir başka yaklaşım olarak, Wong ve arkadaşları tarafından Sistem-R'nin daha basit bir uygulaması ortaya atılmış ve minimum seçim yöntemi olarak kabul görmüştür [27]. Buna göre orta seviyedeki düğümleri mümkün oldukça küçük tutmak hedeflenir. En az sayıda seçim yapılacak tablo en başa alınarak, daha sonraki aşamalarda daha az kaydın taranması esasına dayanır.

Bu yöntemlerin dışında Krishanmurthy, Boral ve Zaniolo tarafından KBZ Algoritması geliştirilmiştir[28]. KBZ algoritması en iyi sıralamayı polinom zamanda $O(n^2)$ yapabilmektedir.

2.4. Birleştirme sıralaması problemine deterministik olmayan çözümler

İVTYS'lerde meta sezgisel yöntemlerin sorgu iyileştirmesinde kullanıldığı çalışmada tavlama benzetimi başarıyla kullanılmıştır[29]. GoldBerg tarafından yapılan çalışmada, genetik algoritmanın arama optimizasyonu ve makine öğrenmesinde kullanılabileceği gösterilmiştir[30]. Ioannidis ve Kang, çok sayıda birleşim gerektiren sorgularda rastsal algoritmaların kullanılabileceğini önermişlerdir[31].

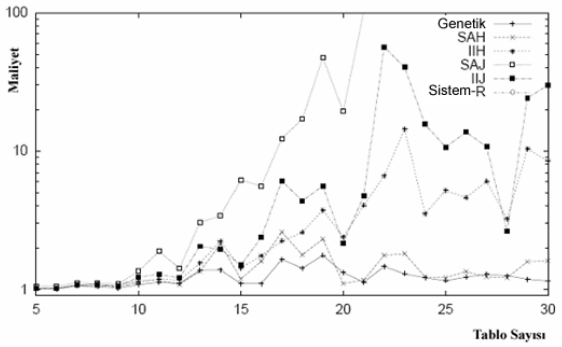
1991 yılında, Bennettt ve arkadaşları, İVTYS'leri için çok sayıda birleşim gerektiren sorgularda birleşim sırası belirlemede genetik algoritmayı kullanan Left-Deep ve Bushy yöntemlerine dayalı alternatif araştırmalar yapmış, özellikle artan sayıda girdiden oluşan birleştirmelerde performans artışı sağlamayı başarmıştır. Bu araştırmada kullanılan bir tablo sıralamasının kodlama dönüşümü ve sıralamaları Şekil 3'te verilmiştir.



Şekil 3. Tabloların birleşim sıralamalarının kodlanması

1992 yılında yapılan çalışmada [11] bugüne kadar yapılan rastsal ve sezgisel birleştirme sırası belirleme algoritmalarının bir genel değerlendirmesi yapılmıştır. Diğer sezgisel tekniklerin yanı sıra tavlama benzetiminin de birleştirme sırasının belirlenmesinde kullanımının başarılı bir örneğini ortaya koymuşlardır. Yaptıkları çalışmada tavlama benzetimi iki farklı şekilde kullanılmış olup araştırmacılar tarafından SAH(Simulated Annealing-H) olarak anılan yöntem geliştirilmiştir.

Araştırma sonuçları Şekil 4'te görülmektedir. SAH, diğer kullanılan yöntemlere göre artan tablo sayılarında daha başarılı sonuçlar verebilmektedir.



Şekil 4. Farklı algoritmaların tablo sayısına bağlı maliyet

3. XML SORGU OPTİMİZASYON YAKLAŞIMLARI

XML'in ilk yıllarında, OQL(Object Query Language), XML-QL, XPath, XQL(XML Query Language)[6], ve benzeri genel kabul görmüş sorgulama yöntemleri kullanılmıştır. XML verinin kullanımının yaygınlaşmasının ardından, bu veriyi sorgulama ihtiyacı bu yöntemler tarafından tam olarak karşılanamaz hale gelmiştir. 1999 yılında verileri sorgulamak için genel kabul görmüş bir standart geliştirmek amacıyla W3C tarafından oluşturulan çalışma grubu IBM, Microsoft, Oracle ile diğer küçük üreticiler ve bazı saygın akademisyenlerin de desteğiyle 2002 yılında XQuery[5] standart taslağı yayınlandı. Bu yayının ardından, XML sorgulama için bu taslak kullanılmaya başlamış ve bu alanda birçok araştırma projesi başlatılmış veya bu standart taslağı ile uyumlu uygulamalara yönelik araştırmalar yapılmıştır[10, 11].

3.1.XML sorgu optimizasyon yaklaşımları

XQuery sorgularının optimize edilmesi iki farklı eksen etrafında şekillenmektedir. Bunlardan ilki, sorguyu öncelikle SQL ifade domainine çevirip daha sonra da SQL ifadesi olarak optimize etmektir[32]. Aşağıda bir XQuery ifadesi ve bu ifadenin SQL'e çevrilmiş şekli görülmektedir.

```
for $d in //yazar, $m in //kitap
where $d/yazarKod = $m/kitapKod
return  <KitapYazar
```

```
yazarKod="{$d/yazarKod}"
kitapKod="{$m/mad}"/>
```

```
SELECT d.yazarKod, m.KitapKod
FROM Yazar d, Kitap m
WHERE d.yazarKod= m.yazarKod
```

Zaman içerisinde daha çok kabul gören yöntem, doğrudan XML cebiri çerçevesinde yapılan sorgu iyileştirmesidir. Doğrudan sorgunun XML cebirine göre planları oluşturulur. XML cebirine göre operatör eşdeğerler kullanılır ve XML veriye erişim-işleme maliyeti hesaplanır. Ancak bu noktada birçok farklı yaklaşım mevcuttur ve yaygın bir XML-XQuery cebiri henüz bulunmamaktadır. En çok kabul gören XQuery cebirlerinden biri XAM(XML Access Module) olup Timber'da kullanılan cebirin de temelini oluşturmaktadır. Bu alanda yapılan iki önemli araştırma olarak Galax[10] ve Timber[11] gösterilmektedir.

XML ile ilişkisel sorguların optimize edilmesindeki temel yaklaşımlar paralellik göstermektedir. Öte yandan yapılan bazı çalışmalarda [33] XML cebiri ile yapılan iyileştirmelerin, ilişkisel yöntemlere göre daha iyi veya daha kötü sonuçlar verdiği görülmüştür.

XQuery ile iyi yapılandırılmış veriler sorgulandığı için, bazen şema üstünden sorgu optimizasyonu yapmak mümkündür. Bu tür bir örnek Şekil 5'te verilmiştir[33].

```
<!ELEMENT Students (Student*)>
<!ELEMENT Student (Name, Address,
Birthday)>
<!ELEMENT Address (Street, City, Zip,
(Tel|Email))>

//Student[Birthday]/Address[Tel|Email]

//Student/Address
```

Şekil 5. XML tip tanımı

Şekil 5'te bir DTD(Document Type Definition) tanımı ile tipi belirtilen bir XML dökümanı ve bu döküman üstünde verilen bir sorgunun şemadan yararlanılarak basitleştirilmiş şekli görülmektedir. Bu basitleştirme, her bir XML elemanının Tel ve Email değerlerinin olmasının zorunluluk şeklinde tanımlanmasına dayanmaktadır. Çünkü şemaya göre her bir öğrencinin adresi, telefonu ve e-mail bilgisi olmak zorundadır.

3.2. TAX XML cebir sistemi

TAX, ilişkisel veri cebirinden daha bağımsız bir veri modeli sunar ve bu veri modeli üstüne kurulu bir ilişkisel cebir önerir[32].

TAX cebirine göre, bir veri ve bu veri üstündeki sorgu, ağaç modeli ile Şekil 6'da verildiği gibi düzenlenmektedir.

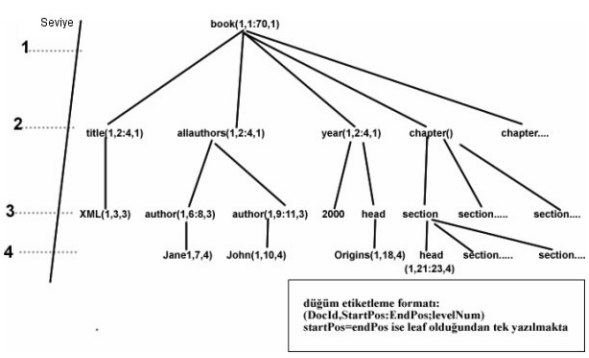
```
<book>
       <title> XML </title>
       <allauthors>

       <author>Jane</author><author>John</a
uthor>
       </allauthors><year>2000</year>
       <chapter>
              <head> Origins</head>
              <section>
                     <head>...</head>

       <section>...</section><section>...</
section>
              </section>
       </chapter>
       <chapter>
              <head>Axes</head>
              <section>
                     <head>...</head>
                     <section>...</section>
                     <section>...</section>
              </section>
       </chapter>
...
</book>
```

Şekil 6. Bir XML dökümanı

Şekil 6'da verilen dökümanın veritabanına aktarım formatı Şekil 7'de gösterilmiştir. Bu şekle göre her bir düğümde, döküman tekil tanımlayıcısı, başlama pozisyonu ve bitiş pozisyonu ile derinliği gibi ek tanımlayıcılara yer verilmektedir.



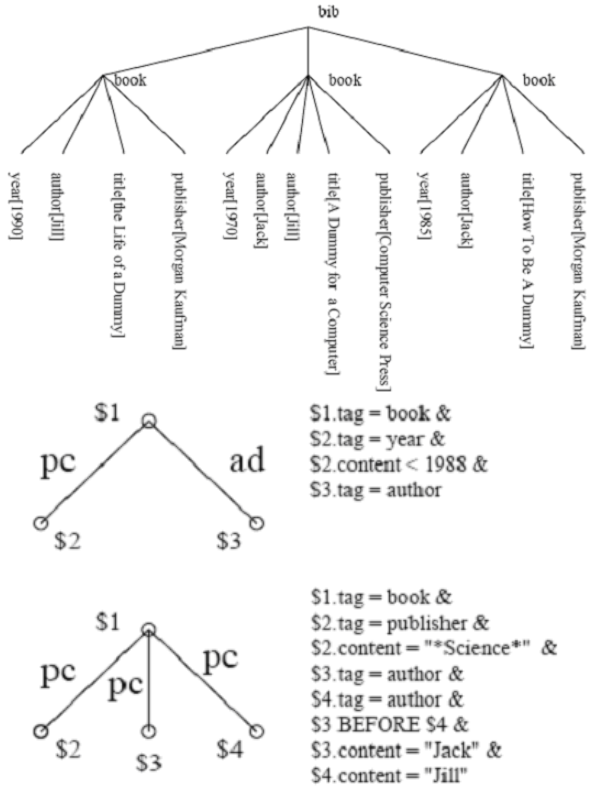
Şekil 7. XML dökümanını veritabanına aktarma formatı

Verilerin sorgulanması aşamasında ise, pattern tree adı verilen yapılardan yararlanılmaktadır. Bu tür bir örnek Şekil 8'de gösterilmiştir. Bu şekle göre pattern tree'ler bir ağaç ve ağaçta yer alan alt-üst ilişkisi tanımından oluşur. Bir alt-üst düğüm ilişkisi eğer ata-soy türünden ise, ara düğüm sayısı önemsizdir ve ad işareti ile veya çift çizgi ile gösterilmektedir. Tersine alt-üst ilişkisi sadece anne-çocuk ilişkisi ise, bu durum sadece bir sonraki düğüm ile bir önceki düğüm şartını aramaktadır ve pc etiketi ile işaretlenmekte veya tek çizgi ile gösterilmektedir.

Şekil 8'de verilen gösterim, pattern tree'lerde yer alan ad ve pc etiketlerinin eşdeğeri düğümlerin daha kolay

sorgulanması amacıyla eklenmiştir. İlgili şartları sağlayan etiket eşleşim formülleri Khalifa ve arkadaşları tarafından oluşturulmuştur[34].

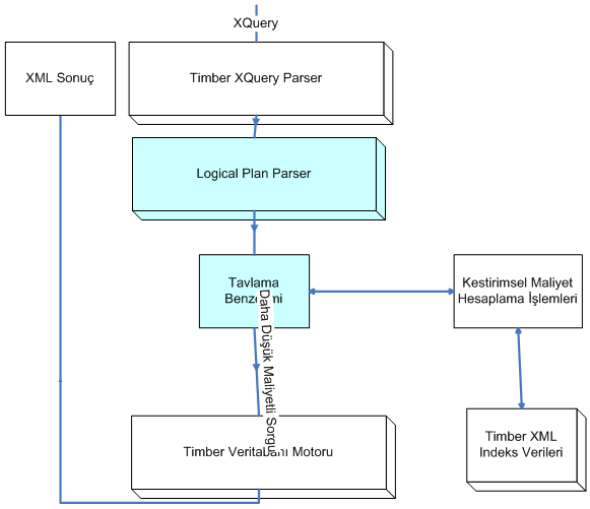
Burada gösterilen pattern tree'lerin içerdiği veriler, veritabanından düğümlerin uygun şartlarda birleştirilmeleri ile bulunur. Bu yaklaşıma Structural Join[34] denmektedir. Bu nedenle, TAX cebiri ile yapılan bir sorguda çok fazla sayıda birleştirme gereksinimi oluşmaktadır. Bu yaklaşımda da sorgulamaya girecek düğümlerin büyüklükleri ile ilgili bir sıralama sorunu mevcuttur [35].



Şekil 8. Bir XML veritabanı ağacı ve bu veritabanındaki verilerden üretilen iki pattern tree

4. GELİŞTİRİLEN YAZILIM

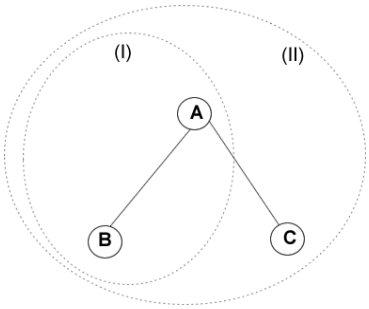
Yapılan çalışmada, Structural Join algoritması ile sorgu birleştirme sıralaması tavlama benzetimi algoritmasıyla gerçekleştirilmiştir. Yapılan çalışma için geliştirilen uygulamanın blok diyagramı Şekil 9'da gösterilmiştir.



Şekil 9. Geliştirilen uygulamanın blok diyagramı

Çözümler arasındaki farklılık, düğümlerin farklı birleşim sıralanmasıyla ortaya çıkmaktadır. Her bir çözümün başarı oranını gösterecek bir fonksiyona gerek vardır. Bu fonksiyon probleme özgüdür ve modeldeki her bir çözümün uygunluk değerini hesaplamaktadır. Bu fonksiyona, uygunluk fonksiyonu denmektedir. Bu fonksiyondan elde edilen sonuca çözümün uygunluk değeri denmektedir. Bu çalışmada, bir çalıştırma planının toplam maliyeti ceza puanı olarak kullanılmıştır. Böylece Structural Join [34] yaklaşımında yer alan maliyet fonksiyonu, tavlama benzetimi için ceza temelli uygunluk fonksiyonu olarak kullanılmıştır.

A düğümü B düğümünün atası olmak üzere her seferinde sadece bir düğüm birleştirerek ilerleyen bir sorgu çalıştırma yöntemi kullanılmıştır. Yöntemin çalışması Şekil 10'da ayrıntılı olarak gösterilmiştir.



Şekil 10. A düğümüne ait B ve C çocuklarının birleştirilme sırası

Şekil 10'da gösterilen A ve B şeklindeki iki düğümün maliyet fonksiyonu Khalifa ve arkadaşları tarafından yapılan araştırmada aşağıdaki gibi hesaplanmıştır [34];

$$2*|AB|*f\_io+2*|A|*f\_st$$

Burada A düğümü B düğümünün atası olmak üzere,

$|A|$: A düğümünün kardinalitisi,

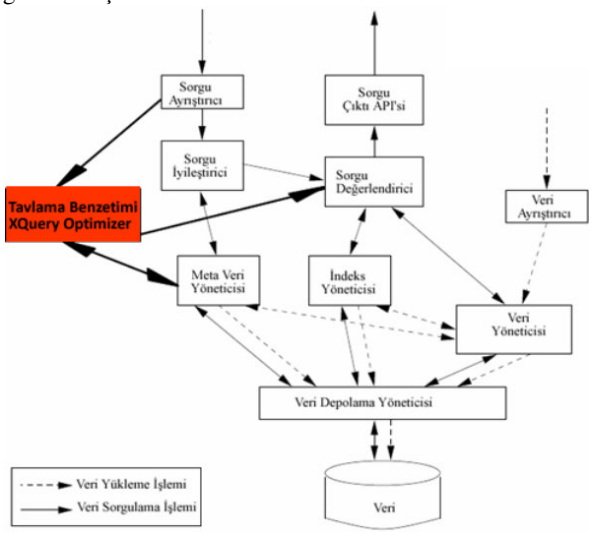
$|AB|$: A-B birleştirmesinin değerinin kardinalitisi,

f_io : I/O işlemleri maliyeti

f_st : Sıralama maliyeti

değerlerini ifade etmektedir.

Bu çalışmada tavlama benzetimi ile problemin çözümünün gerçekleştirilmesi için C#.NET 2.0 programlama dili ile açık kaynak Timber veritabanı motoru program bileşenleri kullanılmıştır. Şekil 11'de Timber veritabanının, XML veriyi ayrıştırması ve sorgulama esnasında işlemler yapan bileşenlerinin blok diyagramları gösterilmiştir[37]. Şekil üstünde, yapılan çalışmanın işlevsel konumunun anlaşılabilmesi için eklenen modül koyu renkli dolu dikdörtgen olarak gösterilmiştir.

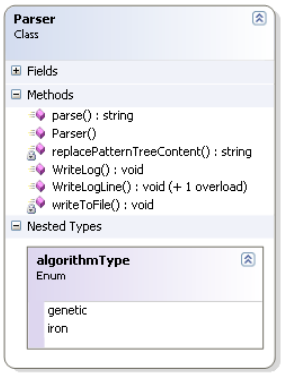


Şekil 11. Timber veritabanı bileşenleri ve yapılan çalışmanın işlevsel konumu

Şekil 11'e göre, gelen XML sorguları, Timber projesinde yer alan ayrıştırıcı tarafından ayrıştırıldıktan sonra, geliştirilen tavlama benzetimi XQuery optimizer tarafından ele alınarak yeniden optimize edilmektedir.

Geliştirilen uygulama, Timber tarafından ayrıştırılmış XQuery sorguların çalıştırma planını almakta ve bu sorgulardan yeni bir ağaç yapısı üretmektedir. Arkasından bu ağaç yapısındaki düğümlerden bir uygun başlangıç çözümü elde edilmekte ve arkasından bu başlangıç çözümünün komşuluklarında yeni çözümler elde edilerek en uygun olan çözüme yaklaşmaya çalışmaktadır. Sorgu penceresine yazılan bir XQuery ifadesi, tavlama benzetimi algoritması ve klasik Timber yöntemi ile optimize edilebilmektedir.

Bu çalışmada geliştirilen yazılımın Parser sınıfının diyagramı Şekil 12'de gösterildiği gibidir. Bu sınıf, kullanıcıdan gelen parametrelerin okunması işini yapar. Ayrıca Timber projesinde yer alan XQueryParser uygulamasının ürettiği ve mantıksal sorgu adı verilen dilin ifadelerini yorumlayarak yeni mantıksal sorgular üretmektedir.

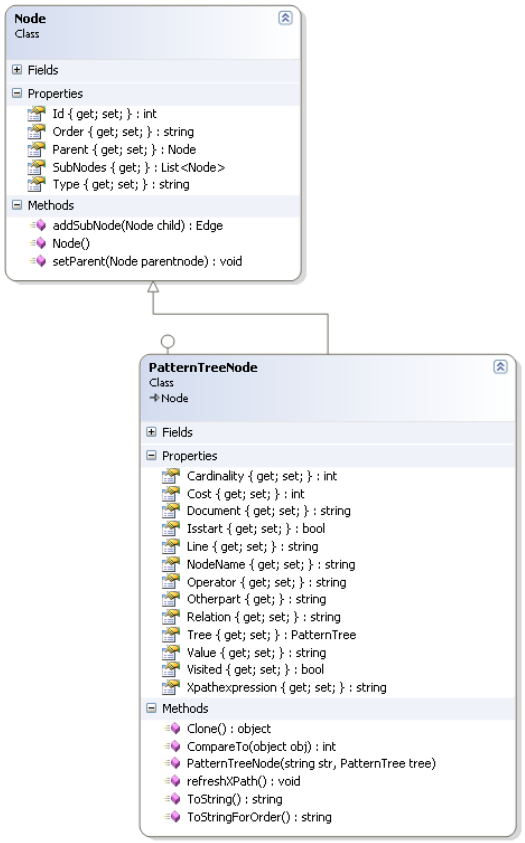


Şekil 12. Parser sınıfı için UML Class diyagramı

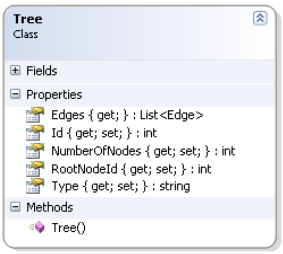
Şekil 13'te UML Class diyagramı gösterilen Node sınıfı, bir temel sınıf kütüphanesidir. PatternTreeNode sınıfının bazı temel özelliklerini özet olarak bulundurmaktadır. Şekil 13'de görülen PatternTreeNode sınıfı, mantıksal ifadede bulunan desen ağacının her bir düğümünü tutmak için kullanılır. Node sınıfından bazı özellikleri miras alır. Bu sınıf ICloneable arayüzünü gerçekler. Bu sayede PatternTreeNode'un yeni kopyaları daha pratik oluşturulabilir.

PatternTreeNode sınıfı aynı zamanda IComparable arayüzünü de gerçekler. Bu sayede farklı pattern tree'ler maliyet değerlerine göre kıyaslanabilir hale getirilebilirler ve azalan-artan sıralama işlemi kolaylaştırılmış olur.

PatternTreeNode sınıfı düğümleri hiyerarşik yapıda tutmak için kullanılan sınıftır. UML sınıf diyagramı Şekil 14'te gösterildiği gibidir.

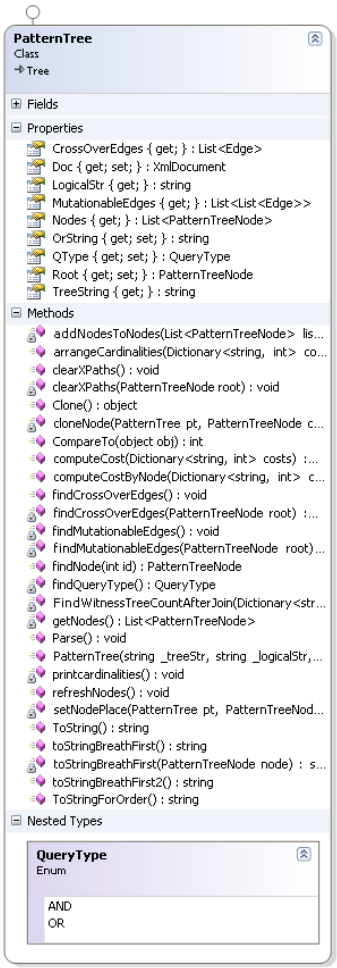


Şekil 13. Node ve PatternTreeNode sınıflarının UML diyagramları



Şekil 14. Tree Sınıfı UML Class diyagramı

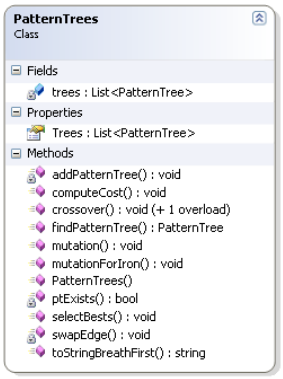
Tree sınıfı mantıksal ifadede bulunan PatternTree'yi ayrıştırma işleminde ve sonrasında tutmak için kullanılır. İçerisinde pattern tree düğümleri hiyerarşik sırada bulunur. Mantıksal ifadede bir PatternTree tüm sorguyu ifade etmek için kullanılır. Sorguda geçen ifadeler PatternTree'de bir düğüm olarak bulunur. Şekil 15'te PatternTree sınıfı UML diyagramı görülmektedir.



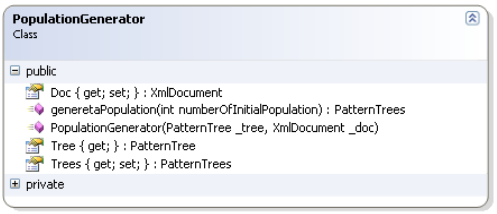
Şekil 15. PatternTree sınıfı UML Class diyagramı

PatternTree sınıfı mantıksal ifadede bulunan tüm pattern ağaçlarını tutan sınıftır. İçerisinde genel olarak sorgunun şekline göre bir veya daha fazla pattern tree içerir. PatternTrees sınıfı UML Class diyagramı Şekil 16'da verilmiştir.

PopulationGenerator sınıfı ilk populasyonu üretmek için kullanılır. Yaptığı iş bir pattern ağacından rastgele istenilen sayıda ağaç üretmektir. PopulationGenerator sınıfına ait UML Class diyagramı Şekil 17'de gösterilmiştir.



Şekil 16. PatternTrees sınıfı UML Class diyagramı



Şekil 17. PopulationGenerator sınıfı UML diyagramı

5. TAVLAMA BENZETİMİ İLE XML SORGU OPTİMİZASYONU

Tavlama benzetimi ilişkisel veritabanı sorgularında sıralama belirlenmesinde kullanılmış bir yöntemdir [14, 17]. Tavlama benzetimi, genetik algoritmaya göre biraz daha karmaşıklığı az ve uygulaması basit bir yöntem olması nedeni ile bu çalışmada tercih edilmiştir.

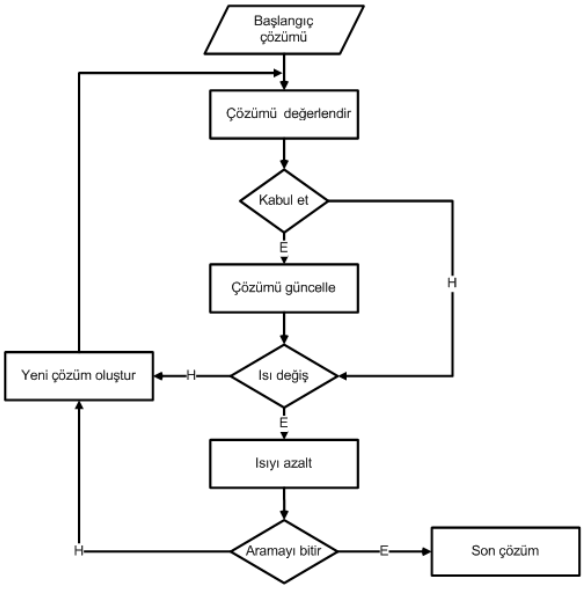
1983 yılında Kirkpatrick ve arkadaşları tarafından önerilen Tavlama Benzetimi, iyileştirme problemleri için iyi çözümler veren bir tarama tekniği olarak kabul görmüştür[36].

Tavlama Benzetimi, katıların ısıtılması ve sonra kristalleşmeye kadar yavaş yavaş soğutulması esasına dayalı bir yaklaşımla çözüm alanını tarar. Bu benzetime göre, sıcaklık değeri, elde edilen en iyi çözümden daha kötü çözümlerin kabul edilme olasılığını belirlemede kullanılır. Yüksek bir sıcaklık değeri ile başlatılır ve her bir adımda sıcaklık değeri düşürülmeden önce belli sayıda çözüm üretilir. Yeni çözümler belirlenen kriterlere göre kabul edilir veya reddedilir. Düşen her sıcaklık, eldeki çözümün bırakılıp yeni bir çözüme geçme ihtimalinin azalmasına etki eder. Sıcaklık minimum değere ulaştığında veya algoritma istenen iterasyon kadar çalıştığında veya istenen çözüme ulaşıldığında algoritma sonlandırılır. Algoritma pseudo-kod olarak Şekil 18'de görülmektedir.

```
Begin
Başlangıç çözümü seç
Başlangıç sıcaklığı seç (t=100)
Sıcaklık azaltma fonksiyonu belirle
repeat
repeat
  Yeni bir komşu çözüm üret
  if (yeni - eski) < 0 then yeni çözümü seç
  else begin
     [0,1] aralığında rassal sayı üret (r)
     if exp(-(yeni-eski)/t) > r then yeni
       çözümü seç
  end
 until iterasyon sayısına kadar
 t = f(t);
until (t<0) veya uygun çözüm bulunana kadar
End
```

Şekil 18. Tavlama benzetimi algoritması

Veri erişimi iyileştirmesinde, bazen sorgu optimizasyonu, sorgu çalıştırma süresinden daha fazla süre gerektirebilir. Bu istenmeyen bir durumdur. Tavlama benzetimi, çözüm sahasını tarama işlemini belli bir iterasyon ile kısıtladığından, sorgu optimizasyonu için sadece belli bir süre çalışıp elde ettiğinin en iyisini döndürmesinden dolayı sorgu optimizasyonuna kolay uygulanabilir bir algoritmadır. Şekil 19'da tavlama benzetimi algoritmasının akış şeması görülmektedir.



Şekil 19. Tavlama benzetimi algoritmasının akış şeması

6. DENEYSEL SONUÇLAR

Bu bölümde, incelenen ve gerçekleştirilen sorgu iyileştirme yöntemlerinin deneysel sonuçlarına yer verilmektedir. Bütün deneyler, çift çekirdekli AMD Turion64 1.6GHz mobile işlemci, 896GB RAM ve 100GB 5400rpm SATA mobil disk'den oluşan sistem üstünde gerçekleştirilmiştir. Bütün deneylerde sorgu çalıştırma ortamı olarak Timber [10] kullanılmıştır.

Deneylerde, verileri test etmek üzere oluşturulan deneme.xml verileri kullanılmıştır. Dosyanın veri boyutu 288KB'dır. İçerisinde 6400 adet XML düğümü içermektedir.

Sorgulama yönteminin etkinliğini test etmek için farklı karmaşıklık seviyesinde sorgular kullanılmıştır.

Sorguların zorlukları ve kolaylıkları, mantıksal seviyede Timber tarafından elde edilen sorgu işleme ara desenlerin aşağıdaki karakteristiklerine göre belirlenmiştir;

- Her bir yükleme ait eşdeğer düğüm sayısı
- Ara desen ağacı derinliği
- Her bir düğüm altında yer alan alt düğüm sayısı

Deneylerde kullanılan sorgular aşağıda verilmiştir. Bu sorgulardan Şekil 20'de yer alan Q1 basit seviye XQuery sorgusu olarak seçilmiştir. Şekil 21'de yer alan Q2 orta zorlukta sorgu ve Şekil 22'de yer alan Q3 sorgusu da ileri seviyede zor sorgu olarak seçilmiştir.

```
for $b in
document("deneme.xml")/countries/country
      for $c in $b//location
      for $d in $c//employee
         for $e in $c//job
where
$b//COUNTRY_ID = UK and
$d/JOB_ID = SA_REP and
$d/MANAGER_ID = 146
         return
         <a>{$d}</a>
```

Şekil 20. Q1 sorgusu

```
for $b in
document("deneme.xml")/countries/country
        for $c in $b//location
        for $d in $c//employee
where
$b//COUNTRY_ID = UK and
$d/JOB_ID = SA_REP and
$d/DEPARTMENT_ID = 80 and
$d/SALARY < 9000 and
$d/SALARY > 6000 and
         return
         <a>{$d}</a>
```

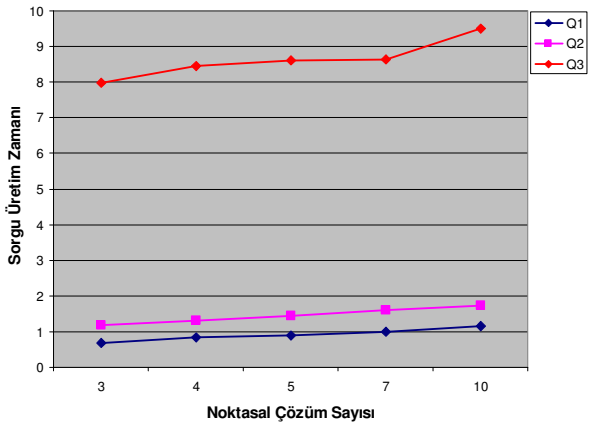
Şekil 21. Q2 sorgusu

```
for $b in
document("deneme.xml")/countries/country
        for $c in $b//location
        for $d in $c//employee
          for $e in $c//job
where
$b//COUNTRY_ID = UK and
$d/JOB_ID = SA_REP and
$d/MANAGER_ID = 146 and
$d/DEPARTMENT_ID = 80 and
$d/SALARY < 9000 and
$d/SALARY > 6000 and
$c/LOCATION_ID > 1000 and
$c/LOCATION_ID < 4000 and
$c/COUNTRY_ID = UK and
$e/MIN_SALARY>6000
         return  <a>{$e}</a>
```

Şekil 22. Q3 sorgusu

Yukarıda yer alan sorgular, Timber üstünde tavlama benzetimi ile birleştirme sıralaması belirleme işleminde kullanılmış ve optimizasyon için harcanan zamanlar aşağıda verilmiştir.

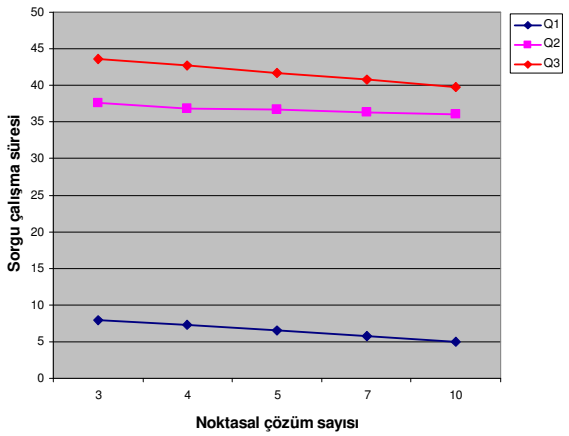
Tavlama benzetimi kullanılarak birleştirme sıralaması belirlemek için yapılan çalışmaların sonuçları Şekil 23'te verilmiştir. Q1, Q2 ve Q3 sorguları için birleştirme sıralaması için geçen süre, tavlama benzetiminin bir noktada artan sayıda çözüm üretmesine bağlı olarak incelenmiştir.



Şekil 23. Tavlama benzetimi ile Q1, Q2 ve Q3 sorgularının oluşturulma süreleri

Şekil 23'te görüldüğü gibi farklı deneylerde bir noktada denenen çözüm sayısı artırılarak sonuçlar alınmıştır. Tavlama benzetiminde, sıcaklık 100 dereceden başlatılarak her iterasyonda 1 derece düşürülmüş ve lineer olarak her noktada sabit sayıda çözüm denenmiştir.

Tavlama benzetimi ile yapılan deneylerin sonucuna göre sorgu karmaşıklığı arttıkça çözüm için harcanan zaman da artmaktadır. Basit seviye bir sorgu için gerekli üretim zamanına göre sorgu karmaşıklaştıkça çok daha fazla üretim zamanı gerekmektedir. Bu fark, birim çözümlerin üretim maliyeti ile doğrudan ilişkilidir. Tavlama benzetimi algoritması ile yapılan eşdeğer deneylerin sonuçları Şekil 24'te görülmektedir.



Şekil 24. Q1, Q2 ve Q3 için tavlama benzetimi ile her iterasyonda değişen birey sayısına karşılık sorgu süresi

7. SONUÇLAR

Bu çalışmada, tavlama benzetimi algoritması İVTYS ortamında olduğu gibi XVTYS ortamındada birleştirme sıralaması belirlenmesinde başarıyla kullanılmıştır. Deneysel çalışmalarda basit seviye XML sorgularda tavlama benzetimi algoritmalarının uygulanabilirliğinin azaldığı, buna karşın özellikle karmaşık sorgularda tavlama benzetiminin daha iyi çalıştırma planları elde edebilmek için güçlü bir seçenek olduğu görülmüştür.

KAYNAKLAR

[1] Internet: "Extensible Markup Language (XML)", http://www.w3.org/TR/REC-xml/, World Wide Web Consortium, 2004.

[2] Internet: "Standardized General Markup Language (SGML)", http://www.w3.org/MarkUp/SGML/, World Wide Web Consortium, 1996.

[3] S. Pal, I. Cseri, O. Seeliger, G. Schaller, L. Giakoumakis, V. Zolotov, "Indexing XML Data Stored in a Relational Database", ***VLDB Conference,*** 1147-1157, 2004.

[4] D. Florescu, D. Kossmann, "Storing and Querying XML Data using an RDMBS", ***IEEE Data Engineering Bulletin,*** 27-34, 1999.

[5] Internet: "XPath", http://www.w3.org/TR/xpath, World Wide Web Consortium, 1999.

[6] Internet: "XML Query Language (XQL)", http://www.w3.org/TandS/QL/QL98/pp/xql.html, World Wide Web Consortium, 1998.

[7] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, D. Maier, D. Suciu, "Querying XML Data", ***IEEE Data Engineering Bulletin***, 22(3), 10-18, 1999.

[8] Ceri, S., Comai, S., Damiani, E., Fraternali P, "A Graphical Language for Querying and Restructuring XML Documents(XML-GL)", ***Computer Networks: The International Journal of Computer and Telecommunications Networking***, 31(11-16), 1171-1187, 1999.

[9] Internet: "XQuery (Proposed Recommendation)", http://www.w3.org/TR/xquery/, World Wide Web Consortium, 2006.

[10] S. Paparizos, S. Al-Khalifa, A. Chapman, H. V. Jagadish, V. S. Lakshmanan, A. Nierman, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, C. Yu, "TIMBER: A Native System for Querying XML", ***SIGMOD Conference*****,** California, 672-672, 2003.

[11] A. Vyas, M. F. Fernández, J. Siméon, "The Simplest XML Storage Manager Ever", ***Very Large Data Bases 32nd international conference*****,** Seoul-Korea, 1215 – 1218, 2006.

[12] Internet: "BerkeleyDB", http://www.oracle.com/technology/products/berkeley-db/index.html, 2006.

[13] E. F. Codd, "A relational model of data for large shared data banks", ***ACM***, 13(6), 377-378, 1970.

[14] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems", ***Symposium on Principles of Database Systems***, Seattle-Washington-ABD, 34-43, 1998.

[15] Y. Wu, J. Patel, H.V. Jagadish, "Structural Join Order Selection for XML Query Optimization", ***ICDE Conference***, Bangalore-Hindistan, 443-454, 2003.

[16] M. Fernandez, J. Simeon, C. Suciu, P. Wadler, "A Data Model and Algebra for XML Query", FST TCS, 2000.

[17] K. Bennett, M.C. Ferris, Y.E. Ioannidis, "A Genetic Algorithm for Database Query Optimization", ***4th International Conference on Genetic Algorithms***, San Mateo-Kanada, 400-407, 1991.

[18] D. Florescu ve M. Kossmann, "Storing and Querying XML data using RDBMS", ***IEEE Data Engineering Bulettin***, 22(3), 27-34, 1999.

[19] M. Fernandez, W. C. Tan ve D. Sucio, "SilkRoute: Trading between relational and XML", ***ACM Transactions on Database Systems***, 27(4), 438-493, New York-ABD, 2002.

[20]J. Shanmugasundaram, J. Kiernan, E. Shekita, C. J. Funderburk, "Querying XML Views of Relational Data", ***26th Very Large Data Bases Conference***, Egypt, 65-76, 2000.

[21] Y. Gözüdeli, "Yazılımcılar için SQL Server 2005 ve Veritabanı Programlama", ***Seçkin Yayınları***, Ankara, 392-292, 2006.

[22] X. Meng, Y. Wang, D. Luo, S. Lu, J. An, Y. Chen, J. Ou, Y. Jiangi, "OrientX : A Schemabased Native XML Database System", ***Proceedings of the VLDB Conference***, Berlin-Almanya, 1057–1060, 2003.

[23] M. M. Astarahan, "System-R: A relational approach to data management", ***ACM Transactions on Database Systems***, 1(2), 97-137, 1976.

[24] K. Ono, G.M. Lohman, "Measuring the complexity of join enumaration in query optimization", ***VLDB Conference***, Brisbane-Avustralya,314-325, 1990.

[25] B. Vance, D. Maier, "Rapid Bush Join-order Optimization with Cartesian Products", ***SIGMOD***,35-46, 1996.

[26] P. G. Sellinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price, "Access path selection in a relational database management system", ***ACM- SIGMOD***, Boston-ABD, 23-34, 1979.

[27] E.Wong, K. Youssefi, "A strategy for query proccessing", ***ACM Transactions on Database Systems***, 1(3), 223-241, 1976.

[28] R. Krisnamurth, H. Boral, C. Zaniolo, "Optimization of non-recursive queries", ***VLDB Konferansı***, Japonya, 128-137, 1986.

[29] Y. E. Ionnidis, E.Wong, "Query optimization by simulated anneling", ***In Proc. of the ACM SIGMOD Conf. on Management of Data***, San Fransisco-ABD, 9-22, 1987.

[30] D. E. Goldberg, "Genetic Algorithms in Search Optimization and Machine Learning", ***Addison Wesley***, Alabama, 217-223, 1989.

[31] Y. E. Ioannidis, Y. C. Kang, "Randomized algorithms for optimizing large join queries", ***In Proc. of the ACM SIGMOD Conf. on Management of Data***, Denver-ABD, 168-177, 1990.

[32] H. V. Jagadish, L. Lakshmanan , V. D. Srivastava, K. Thompson, "TAX: A Tree Algebra for XML", ***The 8th International Workshop on Database Programming Languages***, Frascati, Italya,149-164, 2002.

[33] M. Philippe,"XQuery optimization", ***VLDB 2003 PhD Workshop*****,** Berlin-Almanya, 12-13, 2003.

[34] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, D. Srivastava, "Structural Joins: A Primitive for Efficient XML Query Pattern Matching", ***ICDE***, 141-152 , 2002.

[35] S. Paparizos, S. Al-Khalifa, A. Chapman, H. V. L. Jagadish, V. S. Lakshmanan, A. Nierman, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, C. Yu, "TIMBER: A native system for querying XML", ***In Proc. SIGMOD Conf***., San Diego-California-ABD, 274-291, 2003.

[36] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing", ***Science***, 220(4598), 671-680, 1983.

[37] M. V. Mannino, P. Chu, T. Sagger, "Statistical profile estimation in database systems", ***ACM Computing Surveys***, 20(3), 192-221, 1988.