Kural Örüntülerinin Düzenlenmesi ve Ontolojik Verinin İndekslenmesi ile Rete Tabanlı Bir Çıkarsama Motorunun Eniyilenmesi

T. Özacar, Ö. Öztürk, M.O. Ünalır
Ege Üniversitesi Bilgisayar Müh. Bölümü, İzmir
tugba.ozacar, ovunc.ozturk, murat.osman.unalir@ege.edu.tr

Özetçe

Bu çalışmada Rete tabanlı bir OWL çıkarsama motorunun gerçekleştirimi ve bir eniyileme buluşsalı ile bir indeksleme mekanizması kullanarak, çıkarsama motorunun performansının arttırılması anlatılmaktadır. Eniyileme buluşsalı, bilinen bazı eniyileme buluşsallarının değiştirilip, birleştirilmesi sonucu oluşmuş melez bir buluşsaldır. Ayrıca, ontolojik verinin Piramit Tekniği olarak bilinen bir indeksleme mekanizması ile verimli olarak çalışabilmesi için verinin düzenlenmesi ve Piramit Tekniği'nin eniyilenmesi sunulmaktadır.

Abstract

This work implements a Rete-based OWL inference engine and improves the performance of this inference engine using an optimization heuristic and an indexing mechanism. The optimization heuristic is a hybrid one that modifies and mixes some well known optimization heuristics. Also, we reorganize ontological data and optimize the Pyramid Technique in order to make ontological data to work more efficiently with the Pyramid Technique.

1. Giriş

Büyük boyutta ve birleşik ontolojileri işleyebilme yeteneği, bütün anlamsal web araçları için olduğu gibi çıkarsama araçları için de çok gereklidir. Orta ölçekte ontolojileri iyi sayılabilecek performanslarla işleyebilen kural tabanlı çıkarsama araçları olmasına rağmen, büyük ölçekte ontolojiler için aynı şeyi söylemek mümkün değildir[1]. Bunun nedeni çıkarsama işleminin zaman ve performans gereksiniminin yüksek olmasıdır. Bir başka sorun, çıkarsama motorlarının, veri değişikliklerinde çıkarsama sürecini en baştan başlatmalarıdır. Verilerin sıklıkla değiştiği durumlarda Rete [2][3] gibi algoritmaları kullanmak mantıklıdır. Rete, önceden bulunmuş sonuçları hatırlayarak, bunların tekrar hesaplanmasını önleyen eniyilenmiş bir algoritmadır. Rete yalnızca silinen ve yeni eklenen olguları kurallarla test ederek performansı önemli ölçüde arttırır.

Rete, eniyilenmiş bir algoritma olmasına karşın, bu algoritma üzerinde yapılabilecek daha ileri eniyilemeler vardır. Bu çalışmada, bilinen bazı eniyileme buluşsalları[4] anlamsal web'in ihtiyaçları doğrultusunda güncellenmiş ve biraraya getirilerek, melez bir buluşsal ortaya konulmuştur. Bu melez buluşsalın, Rete üzerinde sağladığı performans artışını daha da arttırmak için Rete ağı içerisinde bazı indeksleme mekanizmaları kullanılmıştır.

Fakat bu indeksleme mekanizmaları, bellek gereksinimi yüksek olan Rete algoritmasının, bellek gereksinimini bir kat daha arttırmıştır. Bu nedenle, çalışma da bu indeksleme mekanizmalarının yerine geçecek ve bellek gereksinimini azaltacak bir alternatif aranmış ve Piramit Tekniği[5] olarak bilinen bir indeksleme mekanizmasının kullanılmasına karar verilmiştir.

Bir sonraki bölümde Rete tabanlı çıkarsama süreci ayrıntılı olarak anlatılmaktadır. Üçüncü bölümde,

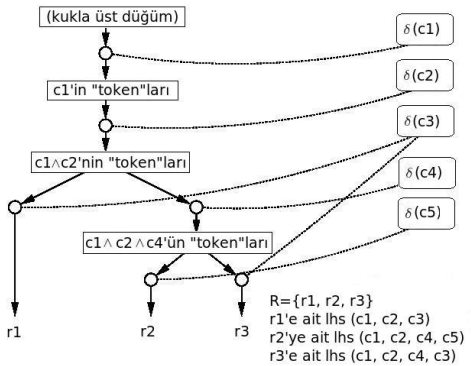
kural örüntülerinin düzenlenmesi ile ilgili eniyilemeler ve bunların melezlenmesi sunulur. Dördüncü bölüm, Piramit Tekniği'nin, ontolojik veriyi indekslemek üzere adapte edilmesini anlatır. Bu başlık üç alt başlık içermektedir. İlki, Piramit Tekniği'nin gerektirdiği girdi formatını sağlayabilmek için anlamsal web kaynaklarının nümerik değerlere nasıl atanacağını sunar. İkincisi Piramit Tekniği'ne ait sorgulama mekanizmasının bir altkümesini kullanarak ontolojik verinin nasıl sorgulanacağını anlatır. Sonuncusu daha iyi sorgu performansı sağlayabilmek için Piramit Tekniği üzerinde bir eniyileme sunar. Beşinci bölümde sunulan eniyilemelerin ve Piramit Tekniği'nin, büyük ölçekli ontolojilerle değerlendirilmesi anlatılmaktadır. Son olarak altıncı bölüm bildiriyi sonlandırır ve geleceğe yönelik çalışmaları sunar.

2. Rete Tabanlı bir Çıkarsama Motoru

Rete, eniyilenmiş bir ileriye doğru çıkarsama algoritmasıdır. Verimli olmayan bir ileriye doğru çıkarsama algoritması, kuralları uygulayarak yeni olgular elde eder ve ontolojiye bir olgu eklendiği veya ontolojiden bir olgu çıkartıldığı anda, algoritma neredeyse bir önceki döngüdekiler ile aynı sonuçları üretmek üzere en baştan başlar. Rete, önceden bulunmuş sonuçları hatırlayarak, bunların tekrar hesaplanmasını önleyen eniyilenmiş bir algoritmadır. Rete yalnızca silinen ve yeni eklenen olguları kurallarla test ederek performansı önemli ölçüde arttırır.

Rete algoritması, Rete ağının çıkarsama sürecine dayalıdır. İzleyen tanımlar, Rete ağındaki kavramların biçimsel gösterimlerini sunmaktadır. U sabitler kümesi, W={w| w = (s, p, o) $\wedge$ s, p, o$\in$ U} ontolojideki tüm olguların kümesi, R ontolojiye ait tüm kurallar kümesi ve O = (W, R) ontoloji ve w $\in$ W olmak üzere her bir w, diğer bir deyişle her bir olgu, bir özne (s), bir yüklem (p) ve bir nesneden (o) oluşur. r$\in$R ve lhs ve rhs atom listeleri olmak üzere, r = (lhs, rhs) 'dir. Bir atom, at = (a, i, v), bir öznitelik a, bir tanıtıcı i ve bir değişkenden v oluşur ve a,i,v$\in$T ve V değişkenler kümesi olmak üzere T = U $\cup$ V'dir. Bir lhs atomu koşul olarak adlandırılır. C, tüm kurallara ait tüm koşulların bulunduğu kümedir. Bir ontolojiye ait Rete ağında $\Omega(O) = (\alpha, \beta)$, α "alpha" ağını, β ise "beta" ağını belirtir. $\alpha = \{\delta(c) \mid c \in C\}$ ve $D = \{x \mid x \subseteq W\}$ olmak üzere, $\delta : C \rightarrow D$ bir fonksiyondur. $\delta(c)$, c koşulu ile eşleşen tüm olguları içeren kümeyi, diğer bir deyişle $\{w \mid w = (s, p, o) \wedge c = (a, i, v) \wedge ((s = a) \vee (a \in V)) \wedge ((p = i) \vee (i \in V)) \wedge ((o = v) \vee (v \in V))\}$ kümesini döndürür. β ağı "beta" belleklerden ve "join" düğümlerden oluşur. “Beta” bellekler "token" adı verilen, kuralların kısmi gerçeklenmelerini saklarken, "join" düğümler kural koşulları arasındaki değişken bağlamalarının tutarlılığını test ederler. Şekil-1[3], ağdaki kavramlar arası ilişkileri göstermektedir. Şekilde, dikdörtgenler "beta" bellek düğümlerini, ovaller "alpha" bellekleri ve daireler "join" düğümleri göstermektedir.

$S = \{x \mid x=(c_1 \wedge ... \wedge c_n) \wedge r = (lhs, rhs) \wedge lhs = (c_1 ,..., c_t) \wedge (1 \leq n \leq t) \wedge r \in R\}$, I = {x | x, w $\in$ W elemanlarından oluşmuş bir tümel-evetlemeli küme} ve $\varphi : S \rightarrow I$ bir fonksiyon olmak üzere, $PI = \{\varphi(s) \mid s \in S\}$ bütün "beta" bellekleri içeren kümedir. $\varphi(s)$, s ile eşleşen olguların bağlamalı setlerini döndürür. Rete ağının sonundaki gerçeklenmeler, "production" düğümleri, diğer bir deyişle, "p-node", olarak ele alınırlar. $P=\{\varphi(s) \mid s = (c_1 \wedge ... \wedge c_t) \wedge r = (lhs, rhs) \wedge lhs = (c_1, ..., c_t) \wedge r \in R\}$ kümesi, tüm “p-node”ları içeren kümedir. Bir yayılımın Rete ağının sonuna varması, diğer bir deyişle bir "p-node"un aktive edilmesi, bir kuralın tüm koşullarının tamamıyla eşleştiğini ve kuralın sağ el tarafındaki (rhs) atomların ontolojiye eklenecek yeni bir olgu ürettiğini gösterir. Ontolojiye yeni bir olgunun eklenmesi, Rete ağını tetikler ve önceden bulunmuş olguların tekrar hesaplanmasını gerektirmeden, bu yeni eklenen olgulara bağlı yeni olgular çıkarsanır.



Şekil-1. Örnek bir RETE Ağı.

Bu çalışmada kullanılan çıkarsama motoru, OWL ontolojilerindeki örtük bilginin, kurallar aracılığı ile belirlik hale getirilmesi için Rete algoritmasını kullanmaktadır. Rete tabanlı çıkarsama motoru, kural belirtimi için OWL Rules Language[6], sorgu belirtimi için OWL-QL [7] kullanır. Sorgular, kuralların özel bir altkümesidirler. Çıkarsama aracına bir sorgu gönderildiğinde, bu sorgu Rete ağına, sağ el tarafı (lhs) boş olan bir "p-node" olarak eklenir. Bu "production" düğümünün elementleri, kullanıcıya sorgu sonucu olarak döndürülürler. Sonuç döndürüldükten sonra, bu "production" düğümü ve "production" düğümünün ebeveyn düğümleri Rete ağından silinir.

3. Eniyileme Buluşsalları ve Bunların Melez Kullanımı

Yüksek performanslı çıkarsama, anlamsal web vizyonu açısından büyük önem taşımaktadır. Rete bir performans eniyileme algoritması olmasına karşın, Rete üzerinde yapılabilecek daha ileri eniyilemeler sözkonusudur. Bu eniyileme buluşsallarının bir kısmının çok önceden biliniyor olmasına karşın, bu çalışma kapsamında en iyiye yakın performansa ulaşabilmek için bilinen eniyilemeler yeniden ele alınmış, bunların güncellenmesi ve birleştirilmesi sonucu melez bir eniyileme ortaya konulmuştur. Aşağıda her bir eniyileme tekniği ve bu tekniğin performansa etkisi anlatılmaktadır. Daha sonra bu buluşsalların birleştirilmesi sonucu oluşan melez eniyileme buluşsalı ayrıntılı olarak anlatılmaktadır.

Şunu da belirtmek gerekir ki, bu melez eniyilemeye ek performans artışı sağlamak için, Rete ağının "Alpha" ve "Beta" kısımlarında bazı indeksleme mekanizmaları da kullanılmıştır.

Buluşsal 1: Kısıtlayıcı Koşulları Öncelikli Olarak Yerleştirmek

Bu optimizasyon, kısıtlayıcı koşulları öncelikli olarak yerleştirerek, Rete ağının orta kısmında oluşan yükü azaltır. Rete algoritması, kuralları, ontoloji üçlüleri ile karşılaştırarak, bir kuralın bütün gerçeklenmelerini bulur. $\{c_1, ...,c_n\}$ bir kuralın sol el tarafı olmak üzere, $L=((c_1), (c_1 \wedge c_2), ..., (c_1 \wedge ... \wedge c_{n-1}), (c_1 \wedge ... \wedge c_n))$ bir sıralı liste, $K = \{k_1, ..., k_n\}$ kümesi r'nin tüm gerçeklenmelerinin kümesi ve k_x L'nin x. elementine ait tüm eşleşmelerin olduğu kümedir. k_n, K'nın n. elementi olmak üzere, $E(k_n)$ k_n'nın boyutunu gösterir;

$$E(k_0)=1$$

$$E(k_n) \subseteq E(k_{n-1}) \times \delta(c_n)$$

Bu nedenle kısıtlayıcı, diğer bir deyişle minimum "alpha" belleğe sahip, koşulları önce dizmek izleyen gerçeklenmelerin boyutlarını azaltır. Hangi koşulun daha kısıtlayıcı olduğuna karar vermek için aşağıdaki üç yöntem kullanılır:

Yöntem 1: Bu yöntem daha az sayıda ontoloji üçlüsüyle eşleşen koşulları, daha öncelikli olarak sıralar. t anında minimum "alpha" belleğe sahip koşulu (m), bulur, fakat m koşulunun bir dizi ekleme ve silme işleminden sonra da minimum belleğe sahip koşul olacağını garanti etmez.

Yöntem 2: Bu buluşsal daha fazla değişken içeren koşulların, daha büyük "alpha" belleğe sahip olacağı varsayımını kabullenir. Bu nedenle daha az değişkene sahip koşulları öncelikli olarak sıralar.

Yöntem 3: Anlamsal web ontolojilerinde kompleks yüklemlerin kullanımı minimum düzeydedir[8]. Ontolojiler genel olarak kapsama ve örneği olma ilişkilerinden oluşur. Bununla birlikte, kapsama ilişkileri, çıkarsama sırasında döngüsel olarak tekrarlayan bir hesaplamaya neden olurlar ve bu hesaplama sonucunda kapsama[1] yüklemi içeren koşulların sayısı daha da

1 Bu çalışmada yalnızca kapsama ilişkisi dikkate alınmıştır. Fakat, çıkarsama işlemi sırasında döngüsel olarak tekrarlayan hesaplamaya neden

fazlalaşır[9]. Bu nedenlerden dolayı, bu buluşsal, kompleks yüklemlere sahip koşullara ait "alpha" belleklerin, kapsama veya örneği olma yüklemine sahip koşulların "alpha" belleklerine göre daha küçük boyutlarda olacağı varsayımını kabullenir. Sonuç olarak, sıklıkla kullanılan yüklemlere sahip koşulları, kural koşulları dizisinin en sonuna yerleştirir. Aynı zamanda, bu buluşsal, ontolojiye yapılan ekleme ve silme işlemlerinin performansını da arttırır. Çünkü sıklıkla kullanılan bir yükleme sahip koşulun değişme olasılığı, diğer koşullara göre daha fazladır ve eğer bu koşul, n tane koşulun en sonunda yer alırsa, koşul yeni yaratılan (veya silinen) bir olguyla eşleştiğinde yalnızca bir "join" işlemi gerçekleşir. Fakat bu koşul, n tane koşulun en başında yer alırsa, n-1 tane "join" işlemi gerçekleşir [4].

Buluşsal 2: Ortak Değişkene Sahip Koşulları Ardışık Olarak Dizmek

Bu optimizasyon, Rete ağının orta kısımlarında oluşan yükü hafifletmek için, ortak değişkene sahip koşulları ardışık olarak dizer. Eğer bir kuralın n. koşulu, n-1. koşul ile ortak bir değişkene (x), sahip ise n-1. gerçeklemede, x üzerinde oluşmuş kısıtlar sonucu $E(k_n)$'in boyutları azalır.

Melez Buluşsal

Bu iki buluşsal, birbiri ile çelişmeden ve birarada kullanıldıklarında ayrı ayrı kullanılmalarına göre daha iyi performans sağlayacakları garantisini sağlayarak melezlenebilirler. r eniyilenecek kural, C(r), r'ye ait tüm koşulların sıralı listesi, r^1 eniyilemenin sonucu ve l, $C(r^1)$ listesinin en son elemanı olmak üzere, eniyileme aşağıdaki sırada çalışır:

Adım 1:

– $C(r^1) \leftarrow$ null
– C(r)'ye ait en kısıtlayıcı koşulu[2] (x), bul ve x'i C(r)'den silerek, $C(r^1)$'ye ekle

Adım 2:

– eğer $C(r) \neq \emptyset$ ise
- $x \in C(r)$ olmak üzere, l ile en fazla sayıda ortak değişkene sahip koşullar içinde en kısıtlayıcı olanını (x), belirle ve x'i C(r)'den silerek, $C(r^1)$'ye ekle
- Adım 2

– değilse
- r'yi döndür

En kısıtlayıcı koşulun belirlenmesi, kurallar ve sorgular için değişir. Yöntem 1, sorguları eniyilerken minimum "alpha" belleğe sahip olguları bulmayı garanti eder fakat kuralların eniyilenmesi sırasında işe yaramaz hale gelir. Rete ağının oluşumu henüz tam olarak bitmediği için, kuralların eniyilenmesi sırasında bir koşul ile eşleşen üçlülerin sayısı hesaplanamaz. Bu nedenle, kuralların eniyilenmesi sırasında yalnızca iki yöntem, şu öncelik sırasında kullanılır: Method 2, Method 3. Bu, minimum değişkene sahip koşullar içerisinde kompleks bir yükleme sahip ilk koşul en kısıtlayıcı olandır, anlamına gelir. Metod 2'nin, Metod 3'ten daha yüksek önceliğe sahip olmasının nedeni, kurallar içerisinde üç değişkenli ve ontolojideki tüm üçlülerle eşleşen çok fazla sayıda koşulun olmasıdır. Bu koşullar en büyük "alpha" belleğe sahiptirler ve Metod 2 bunların en sona dizileceğini garanti eder.

4. Ontolojik Verinin Piramit Tekniği ile İndekslenmesi

Ontolojik veri, her biri bir özne, bir yüklem ve bir nesne içeren bir üçlüye karşılık gelen olgulardan oluşur. Ontolojik sorgular, üçlünün bu üç parçasını temel alır, diğer bir deyişle ontolojik veri üç boyutludur. Yapılan analizler, melez eniyilemenin sağladığı performans artışını daha da arttırmak için, üç boyut üzerinde yaratılan indekslerin bellek gereksinimini önemli ölçüde arttırdığını göstermektedir. Kabul edilebilir ölçülerde zaman artışını göze alarak, bellek gereksinimini düşürebilmek için, Piramit Tekniği olarak bilinen yeni bir indeksleme mekanizması kullanılmasına karar verilmiştir.

4.1. Anlamsal Web Kaynaklarının Nümerik Değerlerle Eşleştirilmesi

Piramit Tekniği d-boyutlu veriyi indekslemek için d-boyutlu noktaları girdi olarak kullanır. Her bir boyut kendine özgü bir URI[3] ile belirtilen bir

olacak diğer geçişken yüklemler de, sık kullanılan yüklemler olarak sınıflandırılabilirler.

2 Eğer sonuç birden fazla koşul içeriyorsa, ilk koşul seçilir.

3 Ontolojilerdeki her bir anonim düğüme ve literale programatik olarak kendilerine özgü sayılar atanmıştır.

anlamsal web kaynağı olmak üzere, ontolojik verinin üç boyutu (özne, yüklem ve nesne) vardır. Piramit Tekniği tarafından kullanılan girdi formatını destekleyebilmek için, üç boyutlu ontolojik verinin üç boyutlu bir noktaya çevrilmesi gerekmektedir. Asıl olarak Piramit Tekniği, 0-1 arası nümerik boyut değerlerini kullanır fakat kesirli sayıların yol açtığı sorunları önlemek amacı ile bu çalışmada her bir URI 16-rakamdan oluşan nümerik değerlerle ifade edilmektedir. Kesirli sayılar kullanıldığında, virgülden sonra 16-rakama kadar bir hassasiyete ulaşılamamaktadır.

Bu çalışmada, [10]'da belirtilen eşleme mekanizması, anlamsal web kaynaklarını eşleyecek şekilde değiştirilmiştir. Eşleme mekanizması, ontolojideki her anlamsal web kaynağına 16-rakamlı ve bu kaynağa özgü bir sayı atar. Bu sayının ilk altı rakamı isim uzayını belirtir. Kalan on rakamsa referans adını belirtir. Diğer bir deyişle, eşleme mekanizması, 10^6 isim uzayı ve 10^{10} referans adı içeren bir ontolojiyi bile destekleyebilmektedir.

Bu eşleme mekanizmasında, her bir isim uzayı 0'dan başlayarak ardışık olarak numaralandırılır (en fazla 10^6'ya kadar). Aynı zamanda her bir referans adı da 0'dan başlayarak numaralandırılır (en fazla 10^{10}'a kadar). u eşlenecek bir URI, n u'nun isim uzayının aldığı değer, r u'nun referans adının aldığı değer ve x eşleme sonucu oluşan 16 rakamlı sayı olmak üzere:

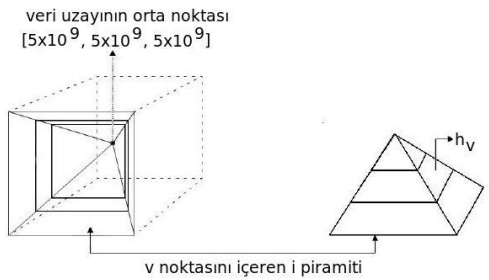
$$x = (n \times 10^{10}) + r$$

Anlamsal web kaynaklarının, sayılarla ifade edilmesi, Piramit Tekniği'nin gerektirdiği girdi formatını sağlamasının yanında ek getiriler sağlar. Çıkarsama sürecinde, karakter dizilerinin üzerinde değerlendirme ve hesaplama yapmak yerine bu işlemleri sayı değerleri üzerlerinde gerçekleştirmenin, bu işlemlerin çok fazla miktada yapıldığı da gözönünde tutulduğunda, önemli ölçüde zaman ve bellek performansı artışı sağlayacağı görülmektedir.

4.2. Piramit Tekniği'nin Ontolojik Veriye Adapte Edilmesi

Piramit Tekniği'nin temel prensibi d-boyutlu veri noktalarının, tek boyutlu değerlere indirgenmesi ve bu değerlerin, B^+-tree gibi verimli indeks yapıları kullanılarak saklanması ve erişilmesidir. Bizim durumumuzda, bu teknik üç boyutlu noktayı tek boyutlu değere dönüştürür. Piramit Tekniği veri uzayını, veri uzayının orta noktasını tepe noktaları olarak kabul eden 2d piramite (bizim durumumuzda 6 piramite) böler. Daha sonra altı piramitin her biri, her biri B^+-tree'nin bir veri sayfasına karşılık gelen birçok parçaya bölünür. i karşılık gelen piramitin (p_i) indeksi, h_v p_i içerisindeki v'nin yüksekliği olmak üzere, Piramit Tekniği üç boyutlu ontolojik veriyi tek boyutlu bir değere ($i + h_v$) dönüştürür (Şekil-2). Ontolojik veriye erişmek istendiğinde, öncelikle piramit değeri hesaplanır ve bu değer anahtar olarak kullanılarak B^+-tree sorgulanır. Sonuç olarak döndürülen B^+-tree sayfaları, sorgu sonucunu oluşturan noktaları içerir. Daha sonra bu veri sayfalarının sıralı olarak aranıp, sorgu sonuçlarına tam olarak karşılık gelen noktaların bulunması gerekir.

Piramit Tekniği çok boyutlu verinin yönetilmesine yönelik bir indeks mekanizmasıdır. Az boyutlu (üç boyut) veriyi indekslemek için bu mekanizmayı seçmemizdeki amaç potansiyel indeks gereksinimlerini karşılayabilmektir. Piramit Tekniği performanstan ödün vermeden boyut sayısını esnek bir biçimde arttırmayı sağlar.



Şekil-2. Piramit Tekniği ile üç boyutlu verinin indekslenmesi.

4.3. İndekslenmiş Ontolojik Verinin Sorgulanması

Dönüştürülmüş ontolojik sorgular, Piramit Tekniği'nin kullandığı sorguların bir alt kümesini oluşturur. Piramit Tekniği sorguları, nokta sorgular ile r = $[q_0^{min}, q_0^{max}]$,..., $[q_{d-1}^{min}, q_{d-1}^{max}]$, (($q_x^{min} = q_x^{max}$) veya ($q_x^{min}$=0 ve $q_x^{max} = 10^{10}$)) ve $0 \leq x \leq d-1$ olmak üzere r ile gösterilen bir d-boyutlu aralıklar kümesi olan aralık sorguların bir alt kümesinden

oluşur.

Ontolojik sorguların iki türü aşağıda tanımlanmıştır ve Tablo 1 ontolojik veri üzerinde tanımlanabilecek olası tüm aralık sorgular ile bu sorguların dönüşümlerini göstermektedir:

Nokta Sorgular: Sorgu yanıtı evet/hayır şeklindedir. Bu tip sorgular ontolojik verinin aranması ve düzenlenmesi amacı ile kullanılırlar. Böyle bir durumda, bir nokta verilir ve döndürülen yanıt bu noktanın veri uzayı içerisinde olup olmadığını gösterir. Noktanın her bir boyutu sorguda belirtilir. Bu problemin çözümü için öncelikle noktanın piramit değeri hesaplanır, daha sonra bu değeri anahtar olarak kullanarak B^+-tree sorgulanır.

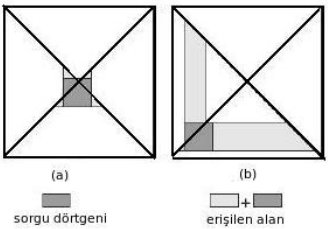
Aralık Sorgular: Bu tip sorgular bir grup ontolojik veri döndürür. Yanıt kümesi, veri uzayında bir noktadan çok bir aralığı belirtir. Bu durumda, üç boyutlu bir aralık verilir, sonuç yanıt kümesini içeren bir veri aralığıdır. Tam olarak yanıta ulaşabilmek için, bu veri aralığının sıralı olarak aranması gerekmektedir.

Tablo 1. Aralık sorguların olası tüm biçimleri.

Ontolojik Sorgu	Dönüştürülmüş Ontolojik Sorgu
*, *, *	$[0,10^{10}]$, $[0,10^{10}]$, $[0,10^{10}]$
pv_s, *, *	$[pv_s, pv_s]$, $[0,10^{10}]$, $[0,10^{10}]$
*, pv_p, *	$[0,10^{10}]$, $[pv_p, pv_p]$, $[0,10^{10}]$
*, *, pv_o	$[0,10^{10}]$, $[0,10^{10}]$, $[pv_o, pv_o]$
pv_s, pv_p, *	$[pv_s, pv_s]$, $[pv_p, pv_p]$, $[0,10^{10}]$
pv_s, *, pv_o	$[pv_s, pv_s]$, $[0,10^{10}]$, $[pv_o, pv_o]$
*, pv_p, pv_o	$[0,10^{10}]$, $[pv_p, pv_p]$, $[pv_o, pv_o]$

4.4 Piramit Tekniğinin Eniyilenmesi

Piramit Tekniği'nin bazı durumlarda sıralı aramadan daha kötü performans gösterdiği kanıtlanmıştır [11]. Şekil-3 piramitin merkezine yakın bir veri aralığı ile piramitin kenarına yakın bir veri aralığını sorgulamak arasındaki farkı göstermektedir. Sorgular kenara yakın bir veri aralığını döndürdüğünde, yanıt kümesi bu veri aralığının oldukça küçük bir kısmını kaplar. Sonuç olarak, ulaşılan veri, yanıt setinden çok daha büyüktür ve bu veri içerisinde yanıt setinin sıralı olarak aranması Piramit Tekniği'ni verimsiz kılar.



Şekil-3. Piramitin merkezine yakın bir veri aralığı ile piramitin kenarına yakın bir veri aralığını sorgulamak arasındaki fark.

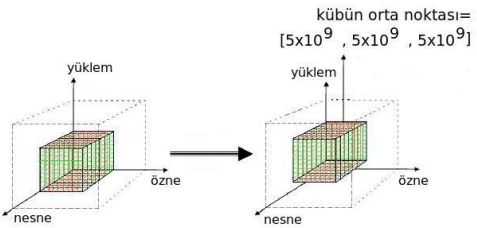
Bu nedenle, verilerin merkeze doğru kaydırılması ile kenar kısımların boşaltılması mantıklı bir yaklaşımdır. Böyle bir kaydırmayı yapabilmek için, eşleme mekanizması üzerinde bazı değişiklikler yapılır. Değiştirilmiş eşleme mekanizmasında (Şekil-4), her bir referans adı 0'dan başlayarak ardışık olarak numaralandırılır (10^{10}'a kadar) fakat isim uzayı numaralandırması aşağıdaki şekilde güncellenmiştir:

s_i i. isim uzayına atanmış sayı, $s_0 = 5 \times 10^5$ ve $s_{x+1} = \delta(s_x)$ olmak üzere,

$$\delta(x) = \begin{cases} 5x10^5 - |5x10^5 - s_i|, & eğer\ s_i < 5x10^5\ ise \\ 5x10^5 + |5x10^5 - s_i| + 1, & değilse \end{cases}$$

u eşlenecek bir URI, n u'nun isim uzayına atanmış sayı, r u'nun referans adına atanmış sayı ve x eşleme sonucu olan 16-rakamlı sayı olmak üzere:

$$x = (n \times 10^{10}) + r$$



Şekil-4. İsim uzayı numaralandırma mekanizması üzerinde yapılan değişikliğin veri dağılımına etkisi.

5. Performans Analizi

Bu çalışmada, çıkarsama motorunun performansını değerlendirmek ve gerçekleştirilen eniyilemelerin ve indeksleme tekniğinin performans üzerindeki etkisini görmek için Lehigh Üniversitesi Karşılaştırmalı Değerlendirmesi[12] kullanılmıştır. Lehigh Üniversitesi Karşılaştırmalı Değerlendirmesi, anlamsal web havuzlarının büyük veri setleri ile sergiledikleri performansı değerlendirmek için geliştirilmiştir. Veri seti, "univ-bench" adındaki ontolojiye uygun olarak ve karşılaştırmalı değerlendirme ile birlikte sağlanan sentetik veri üretme aracını kullanarak üretilir. Havuzların performansları, veri yükleme zamanını, havuz boyutunu, sorguya tepki süresini ve sorgu yanıtlarının tamlığını ve geçerliliğini içeren bir dizi metriğe göre değerlendirilir. Karşılaştırmalı değerlendirme takımı, belirli sorgu tiplerine karşılık gelen 14 örnek sorguyu kapsar.

Piramit Tekniği'ni uygulamadan önce, değerlendirme aşamasında yalnızca LUBM(1,0) veri seti kullanılmıştır. Bellek gereksinimlerinin yüksek olması nedeniyle daha büyük veri setleri kullanılamamaktadır. Testler, aşağıda özellikleri verilmiş olan kişisel bilgisayarda gerçekleştirilmiştir:

- AMD Athlon 64 3500 2200 Ghz CPU; 2 GB RAM;320 GB sabit disk
- Windows XP Professional İşletim Sistemi, .NET Framework 1.1

Karşılaştırmalı değerlendirme içerisindeki ilk metrik, veri setinin yüklenme süresidir. 103,074 örnek içeren LUBM(1,0) veri setinin yüklenmesi 6 dakika 41 saniye sürmektedir. Bu sürenin, diğer havuzlar ile karşılaştırıldığında, özellikle yükleme sürecinin Rete algoritmasının en çok zaman alan kısmı olduğu dikkate alınırsa başarılı bir sonuç olduğu görülmektedir.

Karşılaştırmalı değerlendirme içerisindeki diğer metrikler, sorgularla ilgilidir. Daha kesin sonuçlar elde edebilmek için, sorgular on kez çalıştırılıp bütün sonuçların ortalaması alınmıştır. Sistemde, tüm sorgular yüzde yüz tam ve geçerli olarak yanıtlanmıştır. Sistemin çıkarsama düzeyi, OWL Lite ve OWL DL arasındadır. Bu çıkarsama düzeyi, OWL Entailment [13] testlerine göre belirlenmiş 30 kural ile sağlanmaktadır.

Test kapsamında, eniyileme işleminden sonra tüm sorgular çalıştırılmıştır. Fakat eniyileme olmaksızın tüm sorguların çalıştırılmamasının nedeni, sorguların eniyileme yapılmadığında aşırı derecede yavaş çalışmasıdır. Sistemdeki performans artışının, kural örüntülerinin yeniden düzenlenmesi dışında bir nedeni daha olduğu unutulmamalıdır. Çıkarsamayı hızlandırmak için, Rete ağının "alpha" ve "beta" kısımlarında ek indeksleme mekanizmaları kullanılmıştır. Tablo 2, tüm sorguların çalışma sürelerini göstermektedir.

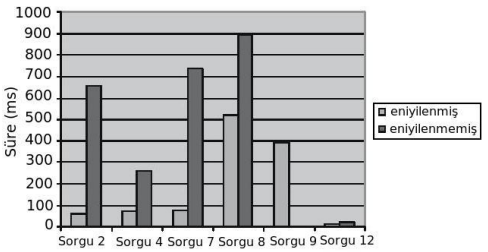
Tablo 2. Sorguların çalışma zamanları.

Sorgu	Metrik		Sorgu	Metrik	
1	Süre(ms)	13	8	Süre(ms)	521
	Yanıt sayısı	4		Yanıt sayısı	7790
	Tamlık	100		Tamlık	100
2	Süre(ms)	60	9	Süre(ms)	390
	Yanıt sayısı	0		Yanıt sayısı	208
	Tamlık	100		Tamlık	100
3	Süre(ms)	23	10	Süre(ms)	10
	Yanıt sayısı	6		Yanıt sayısı	4
	Tamlık	100		Tamlık	100
4	Süre(ms)	70	11	Süre(ms)	140
	Yanıt sayısı	34		Yanıt sayısı	224
	Tamlık	100		Tamlık	100
5	Süre(ms)	532	12	Süre(ms)	13
	Yanıt sayısı	719		Yanıt sayısı	15
	Tamlık	100		Tamlık	100
6	Süre(ms)	477	13	Süre(ms)	11
	Yanıt sayısı	7790		Yanıt sayısı	1
	Tamlık	100		Tamlık	100
7	Süre(ms)	76	14	Süre(ms)	546
	Yanıt sayısı	67		Yanıt sayısı	5916
	Tamlık	100		Tamlık	100

Çalışmada sunulan eniyileme buluşsallarının ana amacı kısmi gerçeklenmelerin boyutlarını azaltmak, diğer bir deyişle sorgu çalıştırılması

sırasında oluşan "token" sayısını azaltmaktır. Sorgu içerisinde tek bir koşul olduğunda, yanıtlama süreci oldukça basittir. Oluşturulan tüm "token"lar, yanıt kümesinin bir elemanıdır. Koşul sayısı ikiye çıktığında, ikinci koşuldaki "token"lar, birinci koşuldaki "token"larla denetlenir ve bu "token"ların yanıt kümesine eklenip eklenmeyeceğine karar verilir. İki koşula kadar, koşul diziliminin performans üzerinde bir etkisi yoktur. Fakat üç veya daha fazla koşul olduğunda, koşul dizilimi önem kazanmaktadır.

Bu nedenle, üç ya da daha fazla koşula sahip, 2, 4, 7, 8, 9 ve 12 numaralı sorguları test etmek anlamlı olmaktadır. Test sonucunda tüm sorguların çalışma zamanlarında düşüş gözlemlenmiştir. Fakat, Şekil-5'te de görüldüğü gibi, 2, 7, 9 ve 4 numaralı sorguların çalışma zamanlarındaki düşüş daha da belirgindir.



Şekil-5. Sorgu performanslarındaki artış.

Bunun nedeni, bu sorgulardaki koşulların tipleridir. Bu sorguların ortak değişkenli koşulları yoktur ve koşullara ait "alpha" bellekler büyük boyutlardadır. Koşulların sırası değiştirilmediğinde, artçı koşullarda yapılacak denetleme sayısı artmaktadır. Örnek olarak, koşulları Tablo-3'te verilen 2 numaralı sorguyu inceleyelim.

Tablo 3. İkinci sorguya ait koşulların ve bu koşullara karşılık gelen "alpha" belleklerin boyutları.

Etiket	Koşul	Boyut
a	?x rdf:type ub:GraduateStudent	1874
b	?y rdf:type ub:University	979
c	?z rdf:type ub:Department	15
d	?x ub:memberOf ?z	8330
e	?z ub:subOrganizationOf ?y	463
f	?x ub:undergraduateDegreeFrom ?y	2414

Eniyileme işleminden sonra, sorgu koşullarının sırası c, e, b, f, a, d olacak şekilde değişir. Sorguyu yeniden düzenlemek için en kısıtlayıcı koşuldan başlanır. En kısıtlayıcı koşul, en küçük "alpha" belleğe sahip c koşuludur. Daha sonra c ile en fazla sayıda ortak değişkene sahip d ve e koşulları seçilir. e koşulu, d'den daha kısıtlayıcı olduğu için ikinci koşul olarak e seçilir. Üçüncü adımda, e ile en fazla sayıda ortak değişkene sahip b, d ve f belirlenir. Bunların içinde, en kısıtlayıcı koşul olan b, üçüncü koşuldur. f koşulu, kalan koşullar içerisinde b ile en fazla sayıda ortak değişkene sahip tek koşul olduğu için, dördüncü koşul olarak seçilir. Geri kalan a ve d koşullarının, her ikisi de b koşulu ile eşit sayıda ortak değişkene sahiptir. Fakat a koşuluna ait "alpha" bellek daha küçük olduğu için, a beşinci koşuldur. Geriye kalan d koşulu sıranın en sonuna yerleştirilir. Tablo 4, eniyilenmiş sıra ve eniyilenmemiş sıranın çalıştırılması sonucu oluşan "token" sayısını göstermektedir.

Tablo 4. Beta bellekler ve oluşturulan "token"ların sayıları.

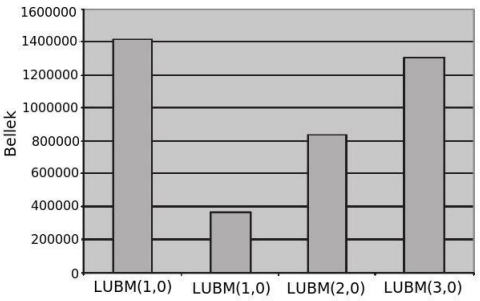
Koşullar	Token Sayısı	Koşullar	Token Sayısı
a	1874	c	15
a∧b	1834646	c∧e	15
a∧b∧c	27519690	c∧e∧b	15
a∧b∧c∧d	0	c∧e∧b∧f	0
a∧b∧c∧d∧e	0	c∧e∧b∧f∧a	0
a∧b∧c∧d∧e∧f	0	c∧e∧b∧f∧a∧d	0

Sorgu 2'nin eniyilenmiş ve eniyilenmemiş olarak çalıştırılması sonucu oluşan kısmi gerçeklenmelerin boyutlarındaki fark, sorgunun çalışma zamanındaki düşüşü açıklamaktadır. Rete ağına bir sorgu eklenmesi sırasında, asıl zaman alan kısım ilgili düğümlerde yeni "token"ların oluşturulmasıdır. Koşul dizisi güncellenmeden önce, Sorgu 2 ağa eklendiğindinde 27519690 "token" oluşturulmuştur. Fakat dizi eniyilendiğinde yalnızca 15 "token" oluşturulmaktadır. Kısacası eniyilenmeden sonra çalışma zamanında düşüş olmasının nedeni oluşturulan "token" sayısındaki düşüştür.

Sistemde, sorgular çıkarsama motoruna eklenmeden önce eniyilenmektedir. Eniyilemeyi gerçekleştirmek için ek bir süreye gerek duyulsa

da, bu ek süre sağlanan performans artışı ile karşılaştırıldığında göz ardı edilebilecek kadar kısadır.

Sistemdeki performans artışının bir nedeninin de Rete ağında kullanılan indeks mekanizmaları olduğu belirtilmişti. Fakat bu indeks mekanizmalarının, bellek gereksinimini arttırdığı gözlemlenmiştir. Bu nedenle bu indeks mekanizmalarını kaldırıp, Piramit Tekniği adı verilen yeni bir indeksleme mekanizması kullanmaya karar verilmiştir. Bu yeni teknik kabul edilebilir ölçüde bir performans kaybı yaratmasına rağmen, bellek gereksinimini %70 oranında düşürerek, zaman ve bellek gereksinimleri arasında optimum dengeyi sağlamıştır (Şekil-6). Şekildeki ilk satır Piramit Tekniği'nin kullanılmasından önce çıkarsama motorunun bellek gereksinimini göstermektedir. İkinci, üçüncü ve dördüncü satırlarda ise Piramit Tekniği'nin değişik veri setleri ile bellek performansı gösterilmektedir. Tekniğin uygulanmasından sonra, aynı kapasitede bellek ile açılabilen veri seti büyüklüğünde üç kat artış görülmektedir (LUBM(3,0), LUBM(1,0)'ın üç katı büyüklüğündedir). Tablo 5 ise, Piramit Tekniği'nin sorguların çalışma performansları üzerindeki etkisini göstermektedir.



Şekil-6. Piramit Tekniği'nin uygulanmasından sonra çıkarsama motorunun bellek gereksinimi.

Tablo 5. Piramit Tekniği'nin uygulanmasından sonra sorguların çalışma zamanları.

Sorgu	Metrik	Lubm(1,0)	Lubm(3,0)	Lubm(5,0)
1	Süre(ms)	762.0	540.0	440.2
	Yanıt sayısı	4	4	4
	Tamlık	100	100	100
2	Süre(ms)	705.8	434.0	433.8
	Yanıt sayısı	0	2	9
	Tamlık	100	100	100
3	Süre(ms)	868.6	340.2	421.6
	Yanıt sayısı	6	6	6
	Tamlık	100	100	100
4	Süre(ms)	40.0	646.4	105.6
	Yanıt sayısı	34	34	34
	Tamlık	100	100	100
5	Süre(ms)	121.8	502.6	665.2
	Yanıt sayısı	719	719	719
	Tamlık	100	100	100
6	Süre(ms)	315.0	499.6	596.6
	Yanıt sayısı	7790	26258	48562
	Tamlık	100	100	100
7	Süre(ms)	874.8	577.6	577.6
	Yanıt sayısı	67	67	67
	Tamlık	100	100	100
8	Süre(ms)	90.2	718.0	674.4
	Yanıt sayısı	7790	7790	7790
	Tamlık	100	100	100
9	Süre(ms)	705.8	693.4	634.0
	Yanıt sayısı	208	692	1245
	Tamlık	100	100	100
10	Süre(ms)	462.0	634.0	374.6
	Yanıt sayısı	4	4	4
	Tamlık	100	100	100
11	Süre(ms)	899.6	765.0	655.4
	Yanıt sayısı	224	224	224
	Tamlık	100	100	100
12	Süre(ms)	727.6	868.4	340.0
	Yanıt sayısı	15	15	15

	Tamlık	100	100	100
13	Süre(ms)	687.2	530.8	777.6
	Yanıt sayısı	1	4	8
	Tamlık	100	100	100
14	Süre(ms)	537.0	280.6	633.6
	Yanıt sayısı	5916	13559	19868
	Tamlık	100	100	100

6. Sonuçlar

Bu çalışmada, Rete tabanlı bir çıkarsama motorunun zaman ve bellek gereksinimlerini azaltmak amacıyla, bu çıkarsama motoru üzerinde, sorgu örüntülerini yeniden düzenleyen melez bir eniyileme ve Piramit Tekniği'ne dayalı bir indeksleme mekanizması uygulanmıştır. Eniyileme buluşsalı beklendiği gibi sorgu performansını arttırmıştır. Piramit Tekniği de aracın bellek ihtiyacını %70 oranında azaltarak, daha önce açılamayan veri setlerinin açılabilmesini sağlamıştır. Fakat sistem hala gelişime açıktır. Daha fazla ve daha çeşitli eniyilemelerle performans daha da arttırılabilir. Son olarak, bazı çıkarsama araçları daha iyi performans gösterse de, anlatılan çıkarsama motorunun veri değişikliklerine daha dayanıklı olduğu, diğer bir deyişle çıkarsama sürecini yeniden başlatmadığı unutulmamalıdır.

Kaynaklar

[1] Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large OWL Datasets, International Semantic Web Conference (2004) 274-288

[2] Forgy, C.: Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. Artificial Intelligence, 19 (1982) 17–37

[3] Doorenbos, R.B.: Production matching for large learning systems, Technical report, Pittsburgh, PA, USA (2001)

[4] Ishida, T.: Optimizing rules in production system programs, National Conference on Artificial Intelligence. (1988) 699–704

[5] Berchtold, S., Böhm, C., Kriegel, H.P.: The pyramid-technique: Towards breaking the curse of dimensionality. In Haas, L.M., Tiwary, A., eds.: SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA, ACM Press (1998) 142–153

[6] Horrocks, I., Patel-Schneider, P. F.: A Proposal for an OWL Rules Language Semantics and Abstract Syntax, Draft Version of 16 October 2003

[7] Fikes, R., Hayes, P., Horrocks, I.: OWL-QL: A Language for Deductive Query Answering on the Semantic Web, Technical Report KSL 03-14, Stanford University, Stanford, CA (2003)

[8] Staab, S.: Ontologies' Kisses in Standardization, IEEE Intelligent Systems 17(2002) 70–79

[9] Zhang, L., Yu, Y., Lu, J., Lin, C., Tu, K., Guo, M., Zhang, Z., Xie, G., Su, Z., Pan, Y.: Orient: Integrate Ontology Engineering Into Industry Tooling Environment, International Semantic Web Conference. (2004) 823–838

[10] Jagadish, H.V., Koudas, N., Srivastava, D.: On effective multi-dimensional indexing for strings. In: SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (2000) 403–414

[11] Zhang, R., Ooi, B.C., Tan, K.L.: Making the Pyramid Technique Robust to Query Types and Workloads, ICDE. (2004) 313–324

[12] Guo, Y., Pan, Z., Heflin, J.: An Evaluation of Knowledge Base Systems for Large Owl Datasets, International Semantic Web Conference. (2004) 274–288

[13] Carroll, J.J., Roo, J.D.: Owl Web Ontology Language Test Cases (2004)