Öğrenmeyle Geliştirilmiş Arama Ağaçları (Learning-Augmented Search Trees)

I. Öğrenmeyle Geliştirilmiş Arama Ağaçları Nedir?

Bilgisayar bilimlerinin temel taşlarından olan geleneksel arama ağaçları, verileri sıralı bir düzende saklayarak hızlı ve verimli sorgulamalara olanak tanır. Kırmızı-siyah ağaçlar, AVL ağaçları, 2-3-4 ağaçları veya bunların pratikte yaygın bir uygulaması olan B-Ağaçları gibi yapılar, anahtar ekleme, silme ve belirli bir anahtarı arama gibi temel işlemleri, genellikle hem ortalama hem de en kötü durumda logaritmik zamanda ($\mathcal{O}(\log n)$) gerçekleştirme konusunda kuvvetli teorik garantiler sunar. Bu garantiyi, ağacın yüksekliğini sıkı bir şekilde kontrol altında tutan, karmaşık ve katı dengeleme kuralları (örneğin, AVL ve kırmızı-siyah ağaçlarındaki rotasyonlar, B-Ağaçlarındaki düğüm bölme ve birleştirme işlemleri) uygulayarak sağlarlar. Ancak, bu sağlam teorik temelin ve yaygın kullanımın getirdiği güvenilirliğe rağmen, klasik yaklaşımların doğasında önemli bir kısıtlama bulunur: Verinin kendisinde bulunan istatistiksel özelliklerden veya zamanla ortaya çıkan erişim örüntülerinden genellikle faydalanmazlar ve bunlardan bağımsız olarak çalışırlar [1], [3]. Diğer bir deyişle, veri kümesindeki anahtarların %90'ına çok az erişilirken kalan %10'unun neredeyse tüm sorguları oluşturması veya verilerin belirli bir sayısal aralıkta yoğun bir şekilde kümelenmesi gibi durumlar, klasik bir B-Ağacının veya AVL ağacının temel logaritmik erişim maliyetini değiştirmez. Ağacın yapısı, bu tür dağılımsal bilgilere karşı kördür ve dengeleme mekanizmaları genel garanti sağlamak adına bu potansiyel optimizasyonları göz ardı eder.

İşte bu noktada, son yıllarda giderek artan bir ilgiyle araştırılan **Öğrenmeyle Geliştirilmiş Arama Ağaçları (Learning-Augmented Search Trees)** kavramı devreye girer. Bu yeni nesil veri yapıları, klasik arama ağaçlarının katı, kural tabanlı doğasını, makine öğrenmesi modellerinin veri odaklı ve adaptif yetenekleriyle birleştirerek mevcut kısıtlamaları aşmayı hedefler. Temel fikir oldukça basittir: Geleneksel ağaç yapılarının içine veya yerine, verinin dağılımını, erişim sıklığını veya diğer ilgili özelliklerini "öğrenebilen" makine öğrenmesi modellerini (veya daha genel bir terimle "öngörücüler") entegre etmek [2], [3]. Bu entegrasyon, ağacın hem yapısını hem de üzerinde çalışan algoritmaları veri ve erişim kalıplarına duyarlı hale getirerek performansı (hız ve/veya bellek kullanımı açısından) iyileştirmeyi amaçlar. Bu yaklaşımın arkasındaki ana motivasyonlar şu şekilde özetlenebilir:

- **Erişim Süresinin Azaltılması (Daha Hızlı Erişim):** Öğrenme modelleri, klasik yapıların sunamadığı optimizasyonları mümkün kılabilir. Örneğin, sık erişilen anahtarların ağacın köküne daha yakın yerleştirilmesini sağlayarak [2], [3] veya aranan bir anahtarın sıralı veri içindeki konumunu doğrudan tahmin ederek [1] arama işlemini hızlandırabilirler. Özellikle Kümülatif Dağılım Fonksiyonu (CDF) tahmini kullanan yaklaşımlarda, modelin tahmini, arama yapılacak bölgeyi önemli ölçüde küçültebilir. Mükemmel bir tahmin, arama süresini idealde $\mathcal{O}(1)$'e indirebilir. Tahmin hatalı olsa bile, arama uzayını daraltıp ardından küçük bir yerel arama (örneğin, ikili arama veya üstel arama [5]) yapmak, klasik $\mathcal{O}(\log n)$ seviyesindeki ağaç gezinmesinden daha hızlı olabilir. Bu, özellikle "son mil araması" (last-mile search) olarak adlandırılan, yaprak seviyesine yakın yerlerdeki arama maliyetini düşürmede etkilidir.
- **Bellek Verimliliğinin Artırılması (Daha Az Bellek Kullanımı):** Geleneksel B-Ağaçları gibi yapılarda, iç düğümler anahtarları ve alt düğümlere işaretçileri saklamak için önemli miktarda bellek kullanabilir. Özellikle büyük veri kümelerinde bu durum ciddi bir yük oluşturur. Öğrenilmiş modeller, özellikle basit parametrik modeller (örneğin, sadece eğim ve kesişim parametrelerini saklayan lineer regresyon modelleri), bu iç düğümlerin yerini alarak veya onların işlevini daha az veriyle yerine getirerek çok daha kompakt endeksler oluşturma potansiyeline sahiptir [1]. Daha küçük bir bellek ayak izi, aynı miktarda ana belleğe daha fazla verinin veya daha fazla sayıda endeksin sığdırılabilmesi anlamına gelir, bu da genel sistem performansını olumlu etkiler.
- **Değişen Koşullara Uyum Sağlama (Adaptasyon Yeteneği):** Gerçek dünya sistemlerinde veri dağılımları veya kullanıcıların sorgu kalıpları zaman içinde değişebilir (bu durum "concept drift" olarak da bilinir). Klasik ağaçların katı dengeleme kuralları bu tür değişimlere yalnızca yapısal dengeyi koruyarak yanıt verirken, öğrenme modelleri bu değişimlere daha akıllıca uyum sağlama potansiyeli taşır. Modeller, gelen yeni verilere göre anlık olarak güncellenebilir veya periyodik olarak yeniden eğitilebilir [3], [5]. Bu adaptasyon yeteneği, ağacın performansının zamanla düşmesini engelleyerek dinamik ortamlarda sürdürülebilir bir verimlilik sunabilir.

Bu alandaki araştırmalar, çeşitli öğrenme paradigmalarını ve ağaç entegrasyon yöntemlerini keşfetmektedir. Bazı çalışmalar, Kraska vd. [1] gibi, anahtarın konumunu tahmin etmek için CDF modellemeye odaklanırken ve bunu RMI gibi hiyerarşik yapılarla gerçekleştirirken, diğerleri Lin vd. [2] ve Cao vd. [3]

gibi, erişim sıklığı veya anahtarın göreceli önemi gibi ölçütleri öğrenerek Treap veya B-Ağacı gibi yapılardaki düğüm önceliklerini veya yapısal kararları yönlendirir.

Ancak, öğrenmeyle geliştirilmiş arama ağaçları yaklaşımı, sunduğu potansiyel faydaların yanı sıra çözülmesi gereken önemli zorlukları da beraberinde getirir. Makine öğrenmesi modellerinin doğası gereği olasılıksal olması ve mükemmel tahmin garantisi sunmaması, **model hatalarının** performans üzerindeki etkisini kritik bir konu haline getirir. Tahminler ne kadar hatalı olabilir ve bu hatalara rağmen yapı ne kadar "gürbüz" (robust) kalabilir? **Dinamik veri güncellemeleri** (ekleme, silme) sadece ağaç yapısını değil, aynı zamanda modelin temel aldığı veri dağılımını da değiştirir; bu güncellemelerin verimli bir şekilde nasıl yönetileceği ve modellerin ne zaman, ne sıklıkla ve ne kadar maliyetle **yeniden eğitileceği** veya güncelleneceği önemli mühendislik problemleridir. Klasik ağaçların sunduğu katı **teorik garantiler** (özellikle en kötü durum garantileri), öğrenilmiş yapılar için genellikle daha zayıf veya ortalama durum odaklıdır; bu garantilerin güçlendirilmesi veya yeni analiz yöntemleri geliştirilmesi gerekmektedir. Son olarak, karmaşık modellerin eğitimi ve çıkarımı için **modern donanımlardan (GPU/TPU, SIMD)** yararlanma potansiyeli olsa da, bunun getirdiği ek karmaşıklık ve gecikme (latency) sorunlarının nasıl aşılacağı da aktif araştırma konularıdır [1], [3], [5]. Bu inceleme, öğrenmeyle geliştirilmiş arama ağaçları alanındaki bu temel fikirleri, öncü çalışmaları, elde edilen sonuçları ve karşılaşılan zorlukları daha detaylı bir şekilde ele alacaktır.

II. Öğrenilmiş Endeks Yapıları: "The Case for Learned Index Structures" (Kraska vd., 2018)

Tim Kraska ve arkadaşları tarafından 2018'de sunulan "The Case for Learned Index Structures" [1] başlıklı çalışma, geleneksel veritabanı endeksleme tekniklerine radikal bir alternatif önererek öğrenmeyle geliştirilmiş veri yapıları alanının temellerini atmıştır. Çalışmanın ana fikri, on yıllardır kullanılan B-Ağaçları, Hash tabloları ve Bloom filtreleri gibi endeks yapılarının, aslında belirli bir görevi yerine getiren "modeller" olarak görülebileceği ve bu modellerin yerine makine öğrenmesi ile eğitilmiş modellerin kullanılabileceğidir.

A. Endekslerin Modeller Olarak Yeniden Düşünülmesi

Kraska vd. [1], endeks yapılarının temel işlevini bir modelleme problemi olarak yeniden çerçevelendirmiştir:

- **B-Ağacı Modeli:** Sıralı bir veri kümesindeki bir anahtarın konumunu (örneğin, diskteki sayfa numarası veya bellekteki dizi indeksi) bulmak için kullanılır. B-Ağacının kendisi, anahtarı girdi olarak alıp konumu çıktı olarak veren, karmaşık, kural tabanlı bir model olarak düşünülebilir.
- **Hash Tablosu Modeli:** Bir anahtarı, genellikle tekdüze (uniform) bir dağılıma sahip olacak şekilde, belirli bir bellek adresine eşleyen bir fonksiyondur. Hash fonksiyonu bu modelin özünü oluşturur.
- **Bloom Filtresi Modeli:** Bir anahtarın belirli bir kümede var olup olmadığını (olasılıksal olarak) belirleyen bir ikili sınıflandırıcı modelidir. "Var" veya "Yok/Muhtemelen Var" tahmini yapar.

Bu analojiye dayanarak, yazarlar, eğer verinin kendi istatistiksel özelliklerinden (dağılımı, sıklığı vb.) öğrenen makine öğrenmesi modelleri kullanılırsa, bu görevlerin klasik yapılardan daha verimli (daha hızlı ve/veya daha az yer kaplayan) bir şekilde yerine getirilebileceğini öne sürmüşlerdir.

B. CDF Tahmini ve RMI (Recursive Model Index)

Özellikle sıralı veriler için B-Ağacı yerine kullanılabilecek ana modelleme yaklaşımı olarak Kümülatif Dağılım Fonksiyonu (CDF) tahminini önermişlerdir. Bir veri kümesinin CDF'si, belirli bir k anahtarından küçük veya ona eşit olan elemanların oranını verir. Eğer CDF ($F(k)$) doğru bir şekilde öğrenilebilirse, N elemanlı sıralı bir dizide k anahtarının konumu $p(k) \approx F(k) \times N$ formülüyle doğrudan tahmin edilebilir.

Ancak, tek bir modelin (özellikle basit modellerin) karmaşık veri dağılımlarını yeterince doğru temsil edemeyeceğini gözlemleyen yazarlar, **RMI (Recursive Model Index - Özyinelemeli Model İndeksi)** mimarisini geliştirmişlerdir. RMI, bir model hiyerarşisidir:

- **Katmanlı Yapı:** En üst katmandaki model, tüm veri aralığı için kaba bir konum tahmini yapar ve girdiyi bir sonraki katmandaki ilgili alt modele yönlendirir.
- **Uzmanlaşma:** Her alt model, kendisine atanan daha dar bir anahtar aralığına odaklanır ve bu aralık için daha hassas bir tahmin yapar. Bu yapı, genellikle 2 veya 3 katmandan oluşur.
- **Model Seçimi:** Farklı katmanlarda farklı model türleri (örneğin, kök katmanda küçük bir sinir ağı, alt katmanlarda daha hızlı olan lineer regresyon modelleri) kullanılabilir.
- **Hata Yönetimi:** Yaprak katmandaki model son konum tahminini yapar. Model hatasını yönetmek için, her yaprak modelinin tahminiyle birlikte bir minimum ve maksimum hata sınırı (min-max error bound) saklanır. Son konum, bu sınırlar içinde yapılan yerel bir arama (örneğin, ikili arama) ile bulunur.

Bu katmanlı yapı, hem model karmaşıklığını yönetmeyi hem de tahmin doğruluğunu artırmayı hedefler.

C. Arama Stratejileri ve Diğer Endeks Türleri

Model tahminlerindeki hataları telafi etmek ve performansı artırmak için çeşitli stratejiler önerilmiştir:

- **Model Eğimli Arama (Model-Biased Search):** Yerel arama (örn. ikili arama) sırasında ilk kontrol edilen nokta olarak modelin tahmin ettiği konum kullanılır.
- **Eğimli Dördül Arama (Biased Quaternary Search):** Önbellek verimliliğini artırmak için, ilk adımda tahmin edilen konum ve etrafındaki birkaç nokta kontrol edilir.

Ayrıca, çalışma Hash tabloları ve Bloom filtreleri için de öğrenilmiş yaklaşımlar önermektedir:

- **Öğrenilmiş Hash Fonksiyonları:** Veri dağılımını (CDF) öğrenen bir model, anahtarları kovalara daha dengeli dağıtarak çarpışmaları azaltabilir.

- **Öğrenilmiş Bloom Filtreleri:** Bloom filtresi problemini bir sınıflandırma problemi olarak ele alarak (öğe kümede mi, değil mi?) veya özel öğrenilmiş hash fonksiyonları kullanarak daha az bellekle aynı hata oranını elde etmeyi hedefler.

D. Bulgular ve Zorluklar

Kraska vd. [1], yaptıkları ilk deneylerde, öğrenilmiş indekslerin geleneksel B-Ağaçlarına göre arama süresinde 3 kata kadar hızlanma ve bellek kullanımında ise birkaç kat mertebesinde azalma sağlayabileceğini göstermişlerdir. Modern donanımların (SIMD, GPU/TPU) öğrenilmiş modelleri çalıştırmadaki potansiyel avantajlarına da dikkat çekmişlerdir.

Ancak çalışma, önemli zorlukları da kabul etmektedir:

- **Statik Veri Odaklılık:** Önerilen yapılar ve deneyler büyük ölçüde statik, yani üzerinde ekleme/silme/güncelleme yapılmayan veri kümelerine odaklanmıştır. Dinamik güncellemelerin verimli yönetimi açık bir problem olarak bırakılmıştır.
- **Model Eğitimi ve Güncellemesi:** Modellerin nasıl eğitileceği, ne zaman güncelleneceği ve bu işlemlerin maliyeti detaylandırılmamıştır.
- **Garantiler:** Klasik yapıların sunduğu kesin kötü durum garantileri yerine, öğrenilmiş indekslerin performansı genellikle ortalama duruma ve modelin doğruluğuna bağlıdır.

Bu çalışma, endeksleme alanında yeni bir araştırma yönü açmış ve sonraki birçok çalışmaya (ALEX, öğrenilmiş BST'ler vb.) ilham kaynağı olmuştur.

III. Öğrenmeyle Geliştirilmiş İkili Arama Ağaçları (Lin, Luo, Woodruff, 2022)

Honghao Lin, Tian Luo ve David Woodruff tarafından 2022 yılında sunulan "Learning Augmented Binary Search Trees" [2] başlıklı çalışma, öğrenmeyle geliştirilmiş veri yapıları alanını ikili arama ağaçlarına (BST) taşıyan önemli bir adımdır. Kraska vd. [1] çalışmasının genel model tabanlı indeks fikrinden farklı olarak, Lin vd. daha spesifik bir şekilde klasik, dengeli bir BST olan Treap veri yapısını ele almış ve bu yapıya öğrenme yetenekleri kazandırmayı hedeflemiştir. Çalışmanın temel motivasyonu, klasik Treap'lerin rastgele öncelikler kullanarak beklenen logaritmik performansı sağlamasına rağmen, verinin erişim sıklığı gibi değerli bilgilerden yararlanmamasıdır.

A. Treap Yapısı ve Öğrenme Entegrasyonu

Treap (Tree + Heap), her düğümde hem bir anahtar (key) hem de bir öncelik (priority) değeri tutan bir ikili arama ağacıdır. Ağaç yapısı, iki temel özelliği aynı anda sağlamak zorundadır:

1) **Arama Ağacı Özelliği:** Herhangi bir düğüm için, sol alt ağacındaki tüm anahtarlar o düğümün anahtarından küçük, sağ alt ağacındaki tüm anahtarlar ise büyüktür.
2) **Yığın (Heap) Özelliği:** Herhangi bir düğümün önceliği, alt düğümlerinin (çocuklarının) önceliklerinden daha büyüktür (veya eşittir). Genellikle maksimum yığın özelliği kullanılır.

Klasik Treap'lerde, öncelikler genellikle [0, 1] aralığından rastgele ve sürekli bir dağılımdan seçilir. Bu rastgelelik, ağacın beklenen yüksekliğinin $\mathcal{O}(\log n)$ olmasını sağlar ve yapıyı dengeli tutar.

Lin vd. [2], bu rastgele öncelik atamasını öğrenmeyle değiştirme fikrini ortaya atmıştır. Eğer bir "öngörücü" (örneğin, bir sıklık tahmincisi) her bir anahtarın erişim sıklığını (f_x) veya göreceli sıralamasını (rank, r_x) tahmin edebiliyorsa, bu tahmin doğrudan Treap önceliği olarak kullanılabilir. Yani, bir x anahtarının önceliği $priority_x = f_x$ (veya $-r_x$, çünkü daha yüksek rank düşük sayıya karşılık gelir ama Treap'te yüksek öncelik istenir) olarak atanır. Bu yaklaşımın sezgisel mantığı şudur: Daha sık erişilen (veya daha üst sırada yer alan) anahtarlar daha yüksek önceliğe sahip olacak ve yığın özelliği gereği ağacın köküne daha yakın konumlanacaktır. Bu da ortalama arama derinliğini azaltacaktır.

B. Teorik Analiz ve Zipfian Dağılımı Odaklılık

Lin vd. [2], bu öğrenmeyle geliştirilmiş Treap yapısının teorik performansını analiz etmişlerdir. Analizlerinin odak noktası, özellikle **Zipfian dağılımı** olarak bilinen ve birçok gerçek dünya senaryosunda (örneğin, kelime sıklıkları, web sayfası ziyaretleri) gözlemlenen çarpık (skewed) bir dağılım türüdür. Zipfian dağılımında, en sık erişilen elemanlar, daha az sıklıkta erişilenlerden kat kat daha fazla erişilir.

Çalışmanın ana teorik sonucu, öğrenmeyle geliştirilmiş Treap'in, erişim sıklıkları Zipfian dağılımına uyduğu takdirde, **statik optimaliteye** yakınsadığıdır. Yani, her bir x anahtarı için beklenen arama derinliği $\mathrm{O}(\log(1/p_x))$ olur (burada $p_x = f_x/m$, x'in göreceli erişim sıklığıdır). Bu, sık erişilen elemanların (p_x büyük) çok sığ bir derinliğe ($\log(1/p_x)$ küçük), nadiren erişilen elemanların ise daha derinlere sahip olacağı anlamına gelir. Tüm erişim dizisi üzerindeki toplam beklenen maliyet, $\sum_x f_x \log(1/p_x)$'e, yani verinin entropisine (bilgi teorik alt sınırına) orantılıdır. Bu, öğrenilmiş önceliklerin, Zipfian dağılımı altında, teorik olarak mümkün olan en iyi statik BST performansına sabit bir çarpanla yaklaştığını gösterir. Bu sonuç, klasik rastgele Treap'in beklenen $\mathcal{O}(\log n)$ derinliğinden önemli bir iyileşmedir, çünkü rastgele Treap çarpık dağılımlardan bu şekilde faydalanamaz.

C. Katkılar ve Sınırlılıklar

Lin vd. [2] çalışmasının temel katkıları şunlardır:

- Treap veri yapısına doğrudan öğrenilmiş bilgiyi (sıklık/rank) öncelik olarak entegre eden ilk çalışmalardan biridir.
- Özellikle Zipfian dağılımları altında, öğrenmeyle geliştirilmiş Treap'in statik optimaliteye ulaştığını teorik olarak göstermiştir.
- Öğrenmenin, klasik rastgeleliğe dayalı yapılara göre (belirli dağılımlar altında) nasıl performans avantajı sağlayabileceğini ortaya koymuştur.

Ancak, çalışmanın önemli sınırlılıkları da bulunmaktadır:

- **Zipfian Varsayımı:** Teorik analiz ve optimalite sonuçları büyük ölçüde verinin Zipfian dağılımına uyduğu varsayımına dayanmaktadır. Bu varsayımın geçerli olmadığı genel dağılımlar için performans garantileri belirsizdir.
- **Genel Dağılımlar İçin Potansiyel Sorunlar:** Daha sonraki çalışmalar (örneğin, Cao vd. [3]), sıklığı doğrudan öncelik olarak kullanmanın bazı dağılımlarda (örneğin, önceliklerin çok dar bir aralıkta yoğunlaştığı durumlar) ağacın derinliğinin $\Omega(n)$ olmasına yol açabileceğini göstermiştir. Bu durum, Lin vd. yaklaşımının genel dağılımlar için gürbüz (robust) olmayabileceğini ima eder.
- **Gürbüzlük ve Hata Toleransı:** Çalışma, model tahminlerindeki hataların etkisini sınırlı ölçüde ele almaktadır. Doğrudan sıklık kullanmak yerine, Cao vd. [3] tarafından önerilen kompozit (karma) öncelikler gibi yaklaşımlar, hem genel dağılımlara hem de tahmin hatalarına karşı daha fazla dayanıklılık sunabilir.

Sonuç olarak, Lin, Luo ve Woodruff'un çalışması, öğrenmeyle geliştirilmiş BST'ler alanında önemli bir temel taşıdır. Treap'lere öğrenilmiş önceliklerin entegrasyonunu basit ve etkili bir şekilde göstermiş ve belirli (Zipfian) koşullar altında optimaliteyi kanıtlamıştır. Ancak, yaklaşımın genelliği ve gürbüzlüğü konusundaki sınırlılıklar, sonraki çalışmalar için önemli motivasyon kaynakları olmuştur.

IV. Kompozit Öncelikli Öğrenmeyle Geliştirilmiş B-Ağaçları (Cao vd., 2023)

Xinyuan Cao ve arkadaşları tarafından sunulan "Learning-Augmented B-Trees" [3] ve "On the Power of Learning-Augmented Search Trees" [4] başlıklı çalışmalar, öğrenmeyle geliştirilmiş arama ağaçları alanında önemli genellemeler ve iyileştirmeler sunmaktadır. Bu çalışmalar, Lin vd. [2] tarafından başlatılan öğrenilmiş öncelikli Treap fikrini temel alarak, bu yaklaşımı hem keyfi veri dağılımlarına uyumlu hale getirmiş, hem ikili arama ağaçlarından (BST) B-Ağaçlarına genişletmiş, hem de dinamik erişim kalıplarına uyum sağlayan ve hatalara karşı dayanıklı yapılar önermiştir.

A. Kompozit Öncelik Fonksiyonu

Çalışmanın merkezinde, öğrenilmiş bilgiyi rastgelelikle birleştiren "kompozit öncelik" (composite priority) fonksiyonu yer alır. Doğrudan öğrenilmiş sıklığı veya rank'ı öncelik olarak kullanmanın potansiyel dezavantajlarını (örneğin, belirli dağılımlarda dengesizlik veya hatalara karşı hassasiyet) gidermek için tasarlanmıştır. Her bir x öğesi için bir "ağırlık" (w_x) belirlenir (bu ağırlık, statik durumda göreceli sıklık, dinamik durumda ise örneğin bir sonraki erişime kadar geçecek süreye dayalı bir tahmin olabilir). Kompozit öncelik, bu ağırlık kullanılarak şöyle hesaplanır:

$$priority_x = -\left\lfloor \log_B \log_4\left(\tfrac{1}{w_x}\right) \right\rfloor + \delta_x, \quad \delta_x \sim U(0,1)$$

Burada:

- B, B-Ağacının dallanma faktörüdür (BST için genellikle B=2 veya B=4 alınabilir).
- $\log_B \log_4(1/w_x)$ terimi, ağırlığın etkisini logaritmik olarak ölçekler. Çift logaritma kullanımı, özellikle çok farklı büyüklükteki ağırlıklar arasındaki farkı yumuşatır ve öncelik aralığını daraltır. Yüksek ağırlık (w_x büyük), daha yüksek önceliğe (negatif işaret nedeniyle sayısal olarak daha küçük değere) yol açar.
- $\lfloor \cdot \rfloor$ taban fonksiyonu, önceliğin tamsayı kısmını belirler ("katman" veya "tier" olarak düşünülebilir).
- δ_x, [0, 1] aralığından bağımsız ve tekdüze (uniform) olarak seçilen rastgele bir sayıdır. Bu bileşen, aynı veya çok yakın tamsayı önceliğe sahip öğeler arasında rastgele bir sıralama sağlar, bağları çözer ve Treap benzeri bir dengeleme mekanizması sunar.

Bu kompozit öncelik, öğrenilmiş bilgiden (ağırlıktan) faydalanarak sık erişilen öğeleri köke yaklaştırırken, rastgele bileşen sayesinde yapının genel dağılımlar altında ve gürültülü tahminlerle bile dengeli kalmasını ve en kötü durumda $\mathcal{O}(\log_B n)$ erişim garantisini korumasını sağlar. Bu yaklaşım, Lin vd. [2] çalışmasının Zipfian varsayımını ortadan kaldırır.

B. Statik Optimalite

Erişim sıklıkları (f_x) bilinen statik bir veri kümesi için, ağırlıklar $w_x = f_x/m$ (toplam erişim m) olarak seçildiğinde, bu kompozit önceliklerle inşa edilen öğrenmeyle geliştirilmiş B-Ağacı (veya BST), statik optimaliteye ulaşır [3]:

- Her x öğesinin beklenen derinliği $\mathrm{O}(\log_B(m/f_x))$ olur. Bu, sonucun sadece Zipfian değil, **herhangi** bir sıklık dağılımı için geçerli olduğunu gösterir.
- Toplam beklenen erişim maliyeti ($\sum_x f_x \mathrm{E}[\mathrm{depth}(x)]$), teorik alt sınır olan $\sum_x f_x \log_B(m/f_x)$ ifadesine sabit bir çarpanla ($\mathcal{O}(1)$) yakınsar. Bu, yapının bilgi teorik olarak optimal olduğunu gösterir.

C. Dinamik (Öz-Örgütlenen) B-Ağaçlar ve Çalışma Seti Bağı

Cao vd. [3], [4], yapının dinamik erişim kalıplarına uyum sağlaması için özel bir önceliklendirme şeması önerir. Bu şema, erişimlerin zamansal yerelliğinden (temporal locality) faydalanmayı hedefler:

- **Interval-Set Priority:** Bir x öğesinin i. erişim anındaki önceliği, x'in bir önceki erişimi ile bir sonraki erişimi arasında geçen sürede erişilen **farklı** öğelerin sayısına ($\mathrm{interval}(i, x)$) göre belirlenir. Özellikle, öncelik $\frac{1}{(1+\mathrm{interval}(i,x))^2}$ ile orantılıdır. Yani, bir öğe sık sık (araya az farklı öğe girerek) erişiliyorsa, interval değeri küçük, önceliği ise yüksek olur.
- **Working-Set Bound (Çalışma Seti Bağı):** Bu interval-set önceliği kullanıldığında, öğrenmeyle geliştirilmiş B-Ağacı (ve Treap), toplam erişim maliyetinin $\sum_{i=1}^{m} \log_B(1 + \mathrm{interval}(i, x_i))$ ile sınırlı olduğunu garanti eder. Bu, Splay ağaçları gibi klasik öz-örgütlenen yapıların hedeflediği ve zamansal yerelliği yakalayan güçlü bir teorik bağdır. Cao vd. çalışması, bu bağı öğrenme yoluyla ve B-Ağaçları için kanıtlayan ilk çalışmalardan biridir.
- **Güncelleme Maliyeti:** Dinamik öncelik güncellemeleri, B-Treap mekanizması ile logaritmik maliyetle

($O(|\log_B(w_x^{(i)}/w_x^{(i-1)})|)$) gerçekleştirilir. Interval-set önceliğinin güzel bir yanı, her erişimde sadece erişilen öğenin önceliğinin değişmesi ve güncelleme maliyetinin kontrol altında tutulmasıdır.

D. Hata Dayanıklılığı (Robustness)

Model tahminlerinin (ağırlıkların) hatalı olması durumunda yapının performansı nasıl etkilenir? Cao vd. [3], yapının hatalara karşı dayanıklı olduğunu göstermiştir:

- **Statik Durumda Hata:** Gerçek sıklıklar p iken hatalı tahminler q kullanılırsa, toplam maliyete eklenen yük $m \cdot D_{KL}(p\|q)/\ln B$ ile orantılıdır. Yani, hata (KL ayrışması ile ölçülen) ne kadar küçükse, performans kaybı da o kadar az olur.
- **Dinamik Durumda Hata:** Dinamik ağırlık tahminlerindeki ortalama mutlak hata (MAE) ε ise, toplam maliyete eklenen ek yük $O(m \log_B(1 + n\varepsilon/m))$ ile sınırlıdır. Bu, hatanın etkisinin logaritmik olduğunu ve toplam erişim sayısına (m) göre normalize edildiğini gösterir.

Bu sonuçlar, kompozit öncelik mekanizmasının, öğrenilmiş ağırlıklardaki hatalara karşı yapıyı koruduğunu ve performansın kademeli olarak düştüğünü (graceful degradation) ortaya koymaktadır.

Özetle, Cao vd. çalışması, öğrenmeyle geliştirilmiş arama ağaçlarını daha genel, dinamik ve gürbüz hale getirmiş, teorik olarak güçlü garantiler (statik ve dinamik optimalite, hata dayanıklılığı) sunmuş ve bu fikirleri B-Ağaçları gibi pratik önemi yüksek yapılara başarıyla genişletmiştir.

V. ALEX: Uyarlanabilir ve Güncellenebilir Öğrenilmiş İndeks (Ding vd., 2020)

Jialin Ding ve arkadaşları tarafından 2020 yılında sunulan ALEX (Adaptive Learned Index - Uyarlanabilir Öğrenilmiş İndeks) [5], Kraska vd. [1] tarafından önerilen orijinal "Learned Index" yapısının önemli bir sınırlamasını gidermek üzere tasarlanmıştır: dinamik güncellemelerin (ekleme, silme, güncelleme) desteklenmemesi. Orijinal yapı, statik, salt okunur veri kümelerinde etkileyici performans gösterse de, gerçek dünya veritabanı iş yüklerinin çoğunda karşılaşılan dinamik değişiklikleri yönetemiyordu. ALEX, öğrenilmiş indekslerin temel fikirlerini (veri dağılımından öğrenme, model tabanlı tahminler) pratik, kanıtlanmış depolama ve indeksleme teknikleriyle birleştirerek, dinamik iş yükleri için yüksek performanslı ve düşük bellek ayak izine sahip bir indeks yapısı sunmayı hedefler.

A. ALEX Mimarisi ve Temel Fikirler

ALEX, ana bellekte (in-memory) çalışan, tamamen dinamik bir arama ağacı yapısıdır ve birkaç temel yenilik üzerine kuruludur:

- **Uyarlanabilir RMI (Adaptive RMI):** Orijinal RMI'nin statik (genellikle 2-3 katmanlı sabit) yapısının aksine, ALEX'in RMI benzeri iç düğüm hiyerarşisi dinamiktir. Ağacın derinliği, her düğümdeki model sayısı (fanout) ve hatta düğümlerin türü (iç düğüm mü, veri düğümü mü olacağı) veri dağılımına ve gözlemlenen iş yüküne göre zamanla değişebilir ve uyarlanabilir. Bu adaptasyon, basit maliyet modelleri tarafından yönlendirilir.
- **Gapped Array (Boşluklu Dizi) Veri Düğümleri:** ALEX'in yaprak düğümleri ("data nodes") verileri depolar. Orijinal öğrenilmiş indeksin kullandığı yoğun paketlenmiş (densely packed) sıralı dizi veya B+Ağacının yaprak düğümlerindeki (genellikle sonunda boşluk bırakan) yapının yerine ALEX, "Gapped Array" adı verilen bir düzen kullanır. Bu düzende, anahtarlar arasında stratejik olarak boşluklar ("gaps") bırakılır. Bu boşlukların iki temel faydası vardır:
  1) **Amortize Edilmiş Ekleme Maliyeti:** Yeni bir anahtar eklendiğinde, eğer uygun bir boşluk varsa, anahtar doğrudan bu boşluğa yerleştirilebilir ve büyük veri kaydırmalarına gerek kalmaz. Bu, ekleme işlemlerinin ortalama maliyetini düşürür.
  2) **Model Doğruluğunun Korunması:** ALEX, anahtarları modelin tahmin ettiği konuma yakın yerleştirmeye çalışır ("model-based insertion"). Gapped Array, bu hedefe ulaşmak için daha fazla esneklik sunar.

  Boşlukları verimli bir şekilde atlamak için her veri düğümü, hangi konumların dolu hangilerinin boş olduğunu gösteren bir bitmap tutar.
- **Esnek İç Düğümler:** RMI gibi, ALEX'in iç düğümleri de anahtarı bir sonraki seviyedeki doğru alt düğüme yönlendirmek için modeller (genellikle basit ve hızlı lineer regresyon modelleri) kullanır. Ancak ALEX, iç düğümlerin anahtar uzayını daha esnek bir şekilde bölmesine olanak tanır. Bir iç düğüm, anahtar uzayının bazı kısımlarını doğrudan veri düğümlerine, bazılarını ise daha fazla çözünürlük gerektiren başka iç düğümlere yönlendirebilir. Kolay bölünmeyi sağlamak için iç düğümlerin işaretçi (çocuk) sayısı genellikle 2'nin kuvveti olarak kısıtlanır.
- **Üstel Arama ve Model Tabanlı Ekleme:** Tahmin edilen konumu bulduktan sonra, ALEX anahtarı bulmak için Kraska vd.'nin önerdiği hata sınırları içinde ikili arama yerine, tahmin edilen konumdan başlayan **üstel arama (exponential search)** kullanır. Bu, modelin hata sınırlarını saklama ihtiyacını ortadan kaldırır ve model tahminleri doğru olduğunda ikili aramadan daha hızlıdır. Ayrıca, yeni anahtarlar eklenirken de modelin tahmin ettiği konuma yerleştirilirler ("model-based insertion"). Bu, verilerin modele uygun şekilde konumlanmasını sağlayarak modelin doğruluğunun zamanla korunmasına yardımcı olur.

B. Dinamik Operasyonlar ve Adaptasyon Mekanizmaları

ALEX'in dinamizmi, ekleme ve silme işlemleri sırasında düğümlerin nasıl yönetildiğine ve RMI yapısının nasıl adapte edildiğine dayanır:

- **Ekleme (Non-full Node):** Bir veri düğümü dolu değilse (belirlenen yoğunluk sınırının altındaysa), yeni anahtar

model tahmini ve gerekirse üstel arama ile doğru konuma yerleştirilir. Eğer konumda boşluk varsa doğrudan yazılır, yoksa en yakın boşluğa doğru asgari sayıda eleman kaydırılarak yer açılır ve anahtar eklenir.

- **Ekleme (Full Node) ve Adaptasyon:** Bir veri düğümü belirlenen üst yoğunluk sınırına (d_u) ulaştığında, ALEX'in adaptasyon mekanizması devreye girer. Düğümün gelecekteki operasyon maliyetini (hem arama hem de ekleme için) tahmin etmek üzere basit, lineer **maliyet modelleri** kullanılır. Bu modeller, düğümde gözlemlenen ortalama üstel arama iterasyonu ve ortalama kaydırma sayısı gibi istatistiklere dayanır. ALEX, aşağıdaki aksiyonlardan hangisinin beklenen maliyeti en aza indireceğini bu maliyet modelleriyle hesaplar ve onu uygular:
  1) **Genişletme (Expansion):** Düğüm için daha büyük bir Gapped Array alanı ayrılır, elemanlar yeni alana model tabanlı olarak yeniden yerleştirilir. Eğer gözlemlenen operasyon maliyeti, düğüm oluşturulurken beklenen maliyetten önemli ölçüde sapmışsa, düğümün modeli yeniden eğitilir (retrain); aksi takdirde mevcut model yeni boyuta göre ölçeklenir (scale).
  2) **Yanlara Bölme (Splitting Sideways):** Düğüm iki kardeş düğüme ayrılır. Bu işlem, B+Ağaçlarındaki gibi yukarı doğru (ebeveyn düğümlere) yayılabilir ve gerekirse ebeveyn düğümlerin de bölünmesine yol açabilir.
  3) **Aşağı Bölme (Splitting Downwards):** Mevcut veri düğümü bir iç düğüme dönüştürülür ve veriler bu yeni iç düğümün işaret ettiği iki yeni veri düğümüne dağıtılır. Bu, RMI'nin derinliğini yerel olarak artırır.

  Bu maliyet modeli tabanlı karar verme mekanizması, ALEX'in farklı veri dağılımlarına ve iş yüklerine (örneğin, okuma ağırlıklı vs. yazma ağırlıklı) otomatik olarak uyum sağlamasını ve elle parametre ayarı gerektirmemesini sağlar.
- **Silme:** Basitçe anahtar ve ilgili veri (payload) kaldırılır. Eğer düğümün yoğunluğu alt sınırın (d_l) altına düşerse, düğüm daraltılabilir (contraction) veya komşu düğümlerle birleştirilebilir (merge), ancak makalede birleştirme mekanizması basitlik adına uygulanmamıştır.
- **Sınır Dışı Eklemeler (Out-of-Bounds Inserts):** Mevcut anahtar aralığının dışına taşan eklemeler için, kök düğümün kapsadığı anahtar aralığı genişletilir, gerekirse yeni bir kök düğüm oluşturulur.

C. Performans ve Katkılar

Ding vd. [5], yaptıkları kapsamlı deneylerde ALEX'in aşağıdaki sonuçları elde ettiğini göstermiştir:

- Salt okunur iş yüklerinde, orijinal Öğrenilmiş İndeks'ten 2.2 kata kadar daha yüksek performans ve 15 kata kadar daha küçük indeks boyutu.
- Okuma/yazma içeren dinamik iş yüklerinde, geleneksel B+Ağaçlarından 4.1 kata kadar daha yüksek performans ve 2000 kata kadar daha küçük indeks boyutu.
- Model B+Ağacı ve Adaptive Radix Tree (ART) gibi diğer modern ana bellek indekslerinden de genellikle daha iyi performans.
- Veri dağılımı kaymalarına ve sıralı ekleme gibi zorlu senaryolara karşı gürbüzlük (robustness).
- Büyük veri kümelerine iyi ölçeklenebilirlik.

ALEX'in temel katkıları, öğrenilmiş indeks fikrini pratik ve dinamik hale getirmesi, Gapped Array ve model tabanlı ekleme ile hem arama hem de ekleme performansını optimize etmesi ve maliyet modeli tabanlı adaptasyon ile parametre ayarı ihtiyacını ortadan kaldırmasıdır. Bu çalışma, öğrenilmiş indekslerin statik sınırlamalarını aşarak daha geniş bir uygulama alanına sahip olabileceğini göstermiştir.

KAYNAKLAR

[1] Kraska, T., Beutel, A., Chi, E. H., Dean, J., & Polyzotis, N. (2018, May). The case for learned index structures. In Proceedings of the 2018 international conference on management of data (pp. 489-504).

[2] Lin, H., Luo, T., & Woodruff, D. (2022, June). Learning augmented binary search trees. In International Conference on Machine Learning (pp. 13431-13440). PMLR.

[3] Cao, X., Chen, J., Chen, L., Lambert, C., Peng, R., & Sleator, D. (2022). Learning-augmented b-trees. arXiv preprint arXiv:2211.09251.

[4] Cao, X., Chen, J., Stepin, A., & Chen, L. On the Power of Learning-Augmented Search Trees.

[5] Ding, J., Minhas, U. F., Yu, J., Wang, C., Do, J., Li, Y., ... & Kraska, T. (2020, June). ALEX: an updatable adaptive learned index. In Proceedings of the 2020 ACM SIGMOD international conference on management of data (pp. 969-984).