ANADOLU ÜNİVERSİTESİ BİLİM VE TEKNOLOJİ DERGİSİ
ANADOLU UNIVERSITY JOURNAL OF SCIENCE AND TECHNOLOGY
Cilt/Vol.:10-Sayı/No: 2 : 495-513 (2009)

ARAŞTIRMA MAKALESİ /**RESEARCH ARTICLE**

MIDI MÜZİK VERİLERİNİN SONEK AĞACIYLA DİZİNLENMESİ

Gıyasettin ÖZCAN [1], Adil ALPKOÇAK [2]

ÖZ

Bu çalışmada, müziksel verilerin dizinlenmesi araştırılmıştır. Bu amaçla müzik verilerinin temsili ve sonek ağaçları ile dizinlenmesi konularında araştırmalar yapılmıştır. Mevcut sonek ağacı algoritmaları daha çok DNA gibi biyolojik verilerin dizinlenmesi için tasarlanmışlardır. Oysa bu algoritmalar, müziğin kendine has özelliklerini hesaba katmamıştır. Bu eksikliği gidermek amacıyla, sonek ağacının fiziksel yapısı, müzik veri tabanlarını dizinleyecek şekilde yeniden tasarlanmıştır. Bunun yanı sıra, müzik eserlerini sınıflandırmanın ve her sınıfta yer alan veri setini ayrı bir sonek ağacında dizinlemenin performansı arttırdığı görülmüştür. Önerdiğimiz yaklaşımları test etmek için ise MIDI müzik dosyalarından oluşan veri tabanları kullanılmıştır. Deney sonuçları yaklaşımlarımızın önceki çalışmalardan daha iyi sonuçlar verdiğini göstermiştir.

Anahtar Kelimeler : Müziksel bilgi erişim, MIDI, Çok sesli müzik, Sonek ağaçları, Sayfa erişim.

INDEXING MIDI MUSIC FILES WITH SUFFIX TREES

ABSTRACT

This study is based on indexing music sequences. In this respect, we consider representation of music information and indexing music sequences with suffix trees. Current suffix tree algorithms generally aim at indexing biological sequences such as DNA. Nevertheless, the algorithms do not consider the properties of music sequences. In order to fulfill such leak, we modify the physical structure of the suffix trees. Moreover, we present that clustering the music sequences and indexing each cluster by a different suffix tree enhance the performance. We have tested our approaches on MIDI music files and made comparisons with early approaches. Performance experiments show that our approach outperforms the earlier approaches in the literature.

Keywords: Music information retrieval, MIDI, Polyphonic music, Suffix tree, Page access.

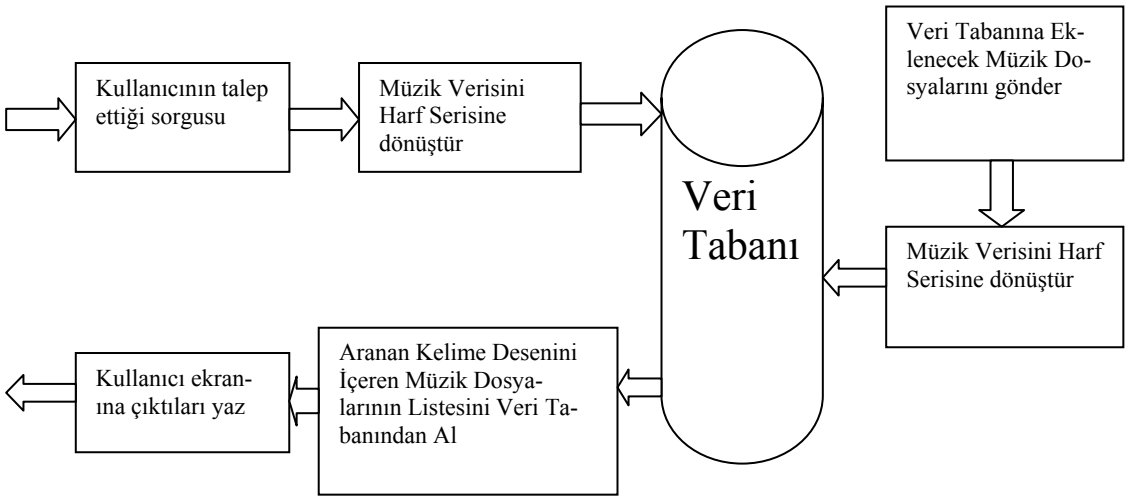
1. GİRİŞ

Müzik verilerinin dijital ortamda yaygınlaşması yeni sonuçlar ve yeni ihtiyaçlar doğurmuştur. Bilgisayarlar ve internet müzik üreticileri ve dinleyiciler arasındaki etkileşimi sağlayan esas arabulucu konumuna gelmiştir. Artık müzik eserleri çoğu dijital veri tabanlarında saklanır ve erişilir olmuştur.

Müziksel Bilgi Erişim Sistemleri (MBES), müziksel veri tabanlarından hızlı ve güvenli bilgi erişimini amaçlar. MBES sistem sorgusunun adımları Şekil 1'de gösterilmiştir. Yapısı gereği tatminkar MBES yazılım üretmek, bilgisayar bilimci ve müzikoloji uzmanlarının ortak çalışmaları ile meydana gelir. Bir yandan bilgisayar bilimci verilerin disk üzerinde yerleşim organizasyonunu belirlerken, müzikoloji uzmanı müziğin algısal özelliklerinin organizasyona katkısını belirler.

[1] Dumlupınar Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Kütahya.
E-mail: ozcan@dumlupinar.edu.tr
[2] Dokuz Eylül Üniversitesi, Bilgisayar Mühendisliği Bölümü, Tınaztepe 35100 İzmir.
Tel: (232) 412 74 01 E-mail: adil.alpkocak@deu.edu.tr

Geliş: 3 Mart 2008; **Düzeltme:** 4 Aralık 2008 **Kabul:** 7 Nisan 2009



Şekil 1. Müzik Veri Tabanının Oluşumu ve Kullanıcı Sorgu Süreci

Günümüzde MBES konusunda kullanıcı ihtiyaçları giderek artmaktadır. Bu ihtiyacı doğuran sebepler ticari, sanatsal, eğitsel ve etiktir. Örneğin, plak şirketleri tüketicilere ürün portföyünü tanıtmak için kapsamlı sorgulama imkânları sunmak istemektedir. Öte yandan, yapısal müzik analizinin bilgisayar yardımı ile yapılması sanatçıların yeni eser üretimine yardımcı olmaktadır. Aynı sebeplerle, MBES müzik eğitimi için de gereksinim konumundadır. En son olarak MBES, etik sorunların çözümünde de etkin olarak kullanılabilecek bir kaynaktır. Bu sayede, yeni bestelendiği iddia edilen bir müzik eserinin veri tabanındaki müzik eserlerinden herhangi birine olan benzerliği belirlenmektedir.

Bu çalışmada MBES ile ilgili ihtiyaçlar ve araştırma konuları özetlenecektir. Bu sebeple, müziğin kendine has özelliklerinden ve ezgi çıkarım yöntemlerinden bahsedeceğiz. En son olarak MIDI müzik dosyalarının sonek ağacı kullanarak dizinlenmesinin önemi, zorlukları anlatılacak ve yeni dizinleme yaklaşımımız açıklanacaktır.

2. MÜZİKSEL BİLGİ ERİŞİM SİSTEMLERİNİ SINIFLANDIRILMASI

Literatürde Müziksel Bilgi Erişim Sistemlerini konu alan çok sayıda araştırma mevcuttur. Bu alanda yapılan araştırma konularını dört sınıfa ayırabiliriz [Fingerhut]:

1-Ön işleme

2- Müzik eserlerinin dizinlemesi

3- Müzik eserlerinden özellik çıkarma

4-Eserlerin bilgisayara kaydedilmesi için organizasyon geliştirme.

2.1 Ön İşleme

Müzik dosyalarında kaydetme ya da okuma işlemlerinden önce dosyanın taranmasına ön işleme denir. Örneğin verilen müzik dosyasında hangi çalgı aletlerinin kullanıldığını bilmek gerekebilir. Böyle bir durumda verilen parçada sadece kullanılan çalgılar taranır.

Ön işleme süreci, dosyanın sıkıştırılması ya da çıkarılması sürecini de içerir. Veri sıkıştırmaktan amaç, tekrar eden nota kalıplarının diskte kapladığı alanın azaltılmasıdır. Böylece müzik verileri disk üzerinde daha az yer kaplar. Bilindiği üzere veriyi sıkıştırmak için birçok yöntem mevcuttur (Leyshon, 2001). Ancak unutulmamalıdır ki, önerilecek sıkıştırma algoritmaları müziğin kendine has yapısını dikkate almak zorundadır. Buna mukabil, sıkıştırılmış dosyaların da yine ön işlemeye tabii tutulup çözümlenmesi gerekir. Sıkıştırılan dosyanın çözümleme işleminde de iki nokta önemlidir. Bunlardan birincisi veri güvenliği, ikincisi ise hızdır.

2.2 Müzik Eserlerinin Dizinlenmesi

Gündelik yaşamda karşılaşılan en önemli MBES problemi, aranan müzik eserlerinin veri tabanından bulunup getirilmesidir. Arama işlemi çoğunlukla müzik parçasının adına, bestecisine, ya da plak şirketine göre yapılır. Ayrıca kullanıcı aklından geçen nota kalıbına göre arama yapmak isteyebilir. MBES, kullanıcıların isteklerini öngörüp kayıtlı müzik dosyalarını doğru şekilde sınıflandırmak zorundadır. Veri sınıflandırma ve dizinleme konusunda çok say-

ıda algoritma literatürde bulunabilir. (Bainbridge, 1999) (Lemström, 2000).

Müziksel veri dizinlemesinde temel bir konu ise müziğin kendine has yapısıdır. Bu gerçeğin doğal sonucu olarak klasik veri dizinleme algoritmalarının en üst performansı vereceği şüphelidir. Zira bu algoritmalar tasarlanma esnasında müziğin özgün koşullarını dikkat almaz. Örneğin Sonek Dizinleme Ağaçları genellikle DNA ve genleri dizinlemek üzere geliştirilmektedir.

2.3 Müzik Eserlerinden Özellik Çıkarılması

Melodi, anahtar, harmoni, ritim gibi öğeler müzik eserlerinin kimliğini belirler(Temperley, 2001). Verilen MIDI müzik dosyasından bu özelliklerin çıkarılması dizinlenme sırasında önemlidir. Böylece benzer anahtar, harmoni ya da ritme sahip müzik dosyaları sınıflandırılabilir.

Her müzik parçasının kendine has bir melodisi vardır. Melodi, dinlenen müzik parçasının hafızada kalan kısmıdır. Bu gerçekten dolayı, kullanıcılar daha çok müziğin melodisini sorgularlar. Öte yandan melodinin oluşumuna ait bir matematiksel formül yoktur; tamamen insan algı ve zekâsı tarafından belirlenir. Dolayısıyla, sadece insan zekâsını hesaba katarak çıkarsamalar yapan melodi ayıklama algoritmaları önerilebilir (Uitdenbogert ve Zobel, 1999,)(Özcan, v.d., 2005). Ancak bu algoritmaların hatalı sonuçlar üretmesi de mümkündür.

MIDI müzik eserlerinin yapısal analizini yapmak güncel bir araştırma konusudur. Parça içinde tekrar eden nota ya da kalıpların bulunması, müzik eserleri arasında benzer kalıpların belirlenmesi yapısal analiz kategorisine girmektedir. Bir müzik dosyasına ait yapısal özelliklerin özetinin çıkarılması da bu kategori içindedir. Böylece yapısal özellik açısından benzer özellikteki dosyaları belirleyip kümelendirmek mümkün olacaktır.

2.4 Organizasyon

Sınıflandırılan müzik dosyalarının saklama cihazına hangi mantığa göre kaydedilmesi gerektiğine ilişkin süreç organizasyon olarak tanımlar. Eğer veri tabanı kullanılacaksa hangi hiyerarşik yapının sorgulama sürecini en aza indirebileceği organizasyonla ilgili bir araştırma konusudur. Bunun yanı sıra, müzik dosyaları bir ağ üzerindeki birden çok bilgisayara kaydedilebilir. Dolayısıyla dosya organizasyonu dağıtık sistem de olabilir.

3. MIDI MÜZİK FORMATI VE DİZİNLEME

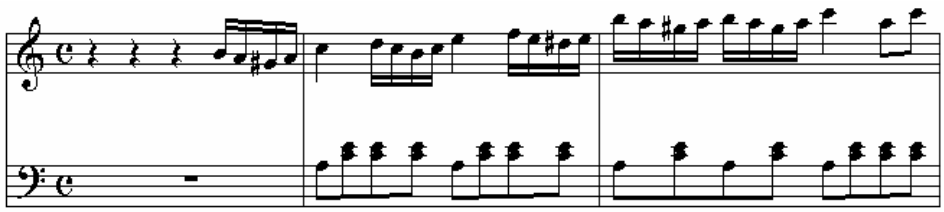
Bu çalışmada müzik dosyalarını işlemek için MIDI müzik dosyalarının dizinlenmesinden ve MIDI formatının öneminden bahsedeceğiz. Daha sonra dokümanlarda benzerlik bulma algoritmalarını tanıtacağız. En son olarak ise MIDI müzik dokümanlarının sonek ağaçları kullanılarak dizinlenmesinin faydalarını ve zorluklarını açıklayacağız.

3.1 MIDI Müzik Formatı

MIDI, elektronik müzik aygıtlarının bilgisayarlar ile iletişim, eşleme ve kontrolünü sağlayan dijital bir müzik formatıdır. Bu formatta müzik perdeleri ve süreleri en çok 128 tam sayı değerini alabilen rakamlarla temsil edilir (General MIDI). Dolayısıyla MIDI notaları, ASCII kodları yardımıyla karakter verisine dönüşebilir. Yapısı gereği MIDI, sinyal bilgisi içeren diğer müzik formatlarına göre çok daha sade bir yapıya sahiptir. Bu özelliğinden dolayı MIDI notaları arasındaki benzerlikler daha hızlı tespit edilebilir.

MIDI formatındaki dosya her bir notaya ait çalgı aleti, gürlük, perdesi ve notanın çalınma aralığı bilgilerini tutar. Standart olarak, MIDI müzik formatı 16 farklı çalgı aletini tanımlar. Notaya ait gürlük verisi ise ses düzeyini belirler. Ayrıca her bir notanın hangi aralıkta basıldığı da belirtilir ve notalar çalınma anına göre sıralanır. Şekil 2'de bir müzik eserinin notaları gösterilmektedir. İki farklı kanalda yer alan notalar eş zamanlı olarak çalınmaktadır. Buna göre üst satırdaki notalar, MIDI formatında 1 numaralı kanalda, alt satırdaki notalar ise 2 numaralı kanalda saklanmaktadır. Bu veriler ışığındaki müzik notalarının perdelerini "Tablo 1"de belirtilen şekilde temsil edebilmekteyiz.

Müzik verilerinin "Tablo 1"deki gibi ardışık perde serileri olarak ifade edilebilmesi dizinleme açısından önem arz eder. Bu sayede müzik verileri üzerinde kelime işleme algoritmalarının uygulanması mümkün olur(Gusfield, 1997). Örneğin müzik parçaları arasındaki benzerliği bulmak için KMP (Knuth vd., 1997) ya da BM (Boyer ve Moore, 1977) kelime benzerlik algoritmaları kullanılabilir. Daha da önemlisi sonek dizileri (Manber ve Myers, 1993) ve sonek ağaçları (Ukkonen, 1995) yardımıyla müzik verileri dizinlenebilir.



Şekil 2. Mozart'ın Alla Turka eserinden bir bölüm.

Tablo 1. MIDI nota perdeleri. Yüksek perde değeri daha tiz sesi temsil etmektedir. Parantez içindeki notalar eşzamanlı çalınmaktadır.

Kanal 1	{ 71, 69, 68, 69, 72, 74, 72, 71, 72, 76, 77, 76, 75, 76, 83, 81, 80, 81, 83, 81, 80 }
Kanal 2	{ 57,(60, 64), (60, 64), (60, 64), 57,(60, 64), (60, 64),(60, 64), 57,(60, 64), 57, (60, 64), 57, (60, 64), (60, 64), (60, 64) }

3.2 Çok Sesli Müzik Eserleri

Müziğin yapısı gereği birden fazla nota aynı anda çalınabilir. Bu tür müziğe polifonik yani çok sesli müzik denir (Feridunoğlu, 2004). Bu durumu Şekil 2 ve Tablo 1'de ifade edilen Alla Turka parçasında da görebiliriz. Ancak polifonik müziğin bu gerçeği dizinleme açısından temel bir soruna yol açar. Eş zamanlı nota perdelerini seri olarak temsil etmek dizinleme açısından önemli bir güçlüğü beraberinde getirir. Çünkü perde serisi çok boyutludur.

Çok sesli müziğin tersi monofonik, yani tek sesli müziktir. Bu tür müzik eserlerinde birden fazla nota asla aynı anda çalmaz. Örneğimizin birinci kanalında hiçbir nota eş zamanlı çalmamaktadır. Bu gerçeği Tablo 1 birinci kanaldan takip edebiliriz. Çünkü parantez içinde toplanan nota perdeleri yoktur.

Aynı anda birden fazla notanın çalması nedeniyle çok sesli müziğin dizinlenmesi çok daha zordur. Örneğin BM (Boyer ve Moore, 1977), KMP(Knuth vd., 1997) ve Sonek Ağaçları (Ukkonen, 1995) tek boyutlu karakter serilerinde dizinleme ve benzerlik bulmak için tasarlanmışlardır. Öte yandan çok sesli müziği dizinlemeleri mümkün değildir. Bu algoritmaların çok sesli müzik için işlev görebilmesi için bazı dönüşümler gerekir.

3.3 Çok Sesli Müzikten Tek Sesli Müziğe Dönüşüm

Müziğin kendine has özellikleri kullanılarak çok sesli müzik eserlerini tek sesli müzik eserine dönüştürmek mümkündür (Uitdenbogert ve Zobel, 1999) (Özcan v.d., 2005). Bu yöntemler müzik eserindeki eşlik notalarını ayırt etmeye çalışır. Zira eşlik notaları müziğin melodisine doğrudan etki etmez. Eşlik notaları atıldıktan sonra müziğin asıl temasını ifade eden melodi notaları saklanır. Kalan notalar tek sesli müziğe dönüştürülerek tek boyutlu bir seri elde edilir. Örnek verecek olursak, Şekil 2'de bahsi geçen müzik eserinin ikinci kanalı sadece eşlik notalarını içerir. Dolayısıyla ikinci kanala ait notaları veri setinden çıkardığımız takdirde müziğin ana temasında bir kayıp olmaz. Bu durum tesadüften öte, müziksel bir gerçektir (Uitdenbogert ve Zobel, 1999) (Özcan v.d., 2005).

Müzik eserlerinde esas temayı anlatan melodi, çoğu kez tiz perdeli notalardan oluşur ve yüksek perde değerleri ile temsil edilir. Dolayısıyla birden çok nota aynı anda tınladığı takdirde, en tiz perdeye sahip olan notanın melodiyi temsil ettiği, geriye kalan notaların ise eşlik olduğu öngörülür. Bu gerçeği Tablo 2'de görmek mümkündür. Alla Turka parçasında melodi kanalında bulunan notalar, eşlik kanalında yer alan notalara göre daha tiz perde değerlerine sahiptir.

Açıkça belirtmek gerekir ki, melodiyi her zaman en yüksek perde değerine sahip notalarda aramak bazen yanılgıya sebep olur. Müzik notalarının farklı çalgı aletleri tarafından çalındığı esnada, düşük perde değerli notalar bazen melodiyi temsil edebilir. Bu esnada sadece yüksek perdeli notayı saklamak melodinin kaybına sebep olur. Bu sebeple tek sesli müziğe dönüşüm esnasında veri güvenliği sorunları hala mevcuttur (Özcan v.d., 2005).

4. KELİME İŞLEME VE SONEK AĞACI

Bu bölümde kelime işleme algoritmalarından olan sonek ağacını ve önemini anlatacağız. Daha sonra sonek ağacının yapısını ve özellik-

lerini tanıtacağız. En son olarak, yeni tasarladığımız sonek ağacı yaklaşımlarımızı açıklayacağız.

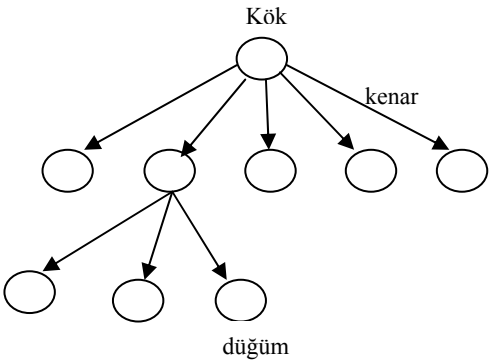
Karakter veri setlerinin özelliklerini tanımak 1970'li yıllardan günümüze önemli bir araştırma konusudur. Bu gelişim sürecini tetikleyen ana unsur organizmaların genetik yapısını belirleyen DNA analizidir (Gusfield, 1997). Sadece bir tek DNA dosyasının gigabyte yer kaplamasından dolayı DNA analizinin elle yapılması mümkün değildir. Daha ötesi, DNA dosyalarının bilgisayarda işlenmesi bile uzun zaman almaktadır. Bu nedenle yeni kelime işlem algoritmalarına ihtiyaç duyulmaktadır.

Günümüzde kelime işleme ve dizinleme algoritmalarının kullanım alanı sadece biyolojik veriler ile sınırlı değildir. Bu algoritmalar arama motorlarında da kullanılır. Bunun yanı sıra, derleyici tasarımı, işletim sistemi ve veri tabanı uygulamalarında da aynı algoritmalara ihtiyaç duyulur. Elbette ki müzik verilerinin dizinlenmesi için de kelime işleme algoritmalarına ihtiyaç vardır. Çünkü müzik sorgulama işlemi de bir çeşit kelime işleme problemidir.

Bu bölümde, kelime işleme algoritmalarından birisi olan sonek ağacını irdeleyeceğiz. Bu ağaç yapısının müzik verilerini dizinleme açısından yerini tanıtacağız. Son olarak ağacın fiziksel yapısını müzik verilerini dizinleyecek şekilde düzenleyeceğiz.

4.1 Sonek Kavramı ve Sonek Ağacı

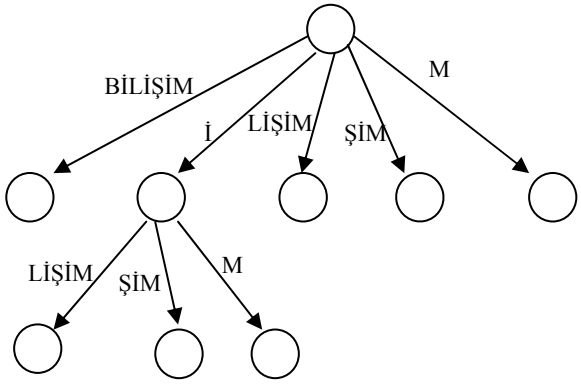
Bilgisayar Biliminde en yüksek erişim hızını sağlayan veri yapıları, ağaçlardır ve ağaçlar bu özelliklerini hiyerarşik yapılarına borçludur (Cormen, 1989). Şekil 3'de ağaç yapısı gösterilmiştir. Örnek ağaçta oval şekiller düğümleri, oklar ise kenarları temsil eder. Ağaçta bir adet kök düğüm vardır ve arama işlemleri her defasında kök düğümden başlar.



Şekil 3. Ağacın genel yapısı. Oval şekiller düğümleri; oklar ise düğümleri birbirine bağlayan kenarları temsil eder.

Sonek ağacının yapısını anlatmadan önce sonek kavramını açıklamakta fayda vardır. Dilbilgisinde sonek, kelimelerin kök veya gövdesinin sonunda yer alan ektir (Gusfield, 1997). Örneğin "bilişim" kelimesi için tanımlanabilecek sonekler şunlardır: "bilişim, ilişim, lişim, işim, şim, im, ve m". Dolayısıyla bir kelimenin harf sayısı kadar soneki olur. "Bilişim" kelimesine ait sonekler Şekil 4'teki ağaçta dizinlenmiştir. Bütün ağaç veri yapılarında olduğu gibi Sonek Ağacı da bir tane kök düğüm içerir ve bütün arama işlemleri kök düğümden başlar. Kökten yaprak düğüme giden yolda karşılaşacağımız alt harf serilerinin birleşimi bir soneki temsil eder. Boyutu ne olursa olsun, her sonek için ağaçta özel bir yol oluşturur. Sonek bilgisinin veri tabanında hangi adreste saklandığını ise yaprak düğümler belirler. Eğer aranan kelime ya da desen yaprak düğüme gelene kadar bulunduysa desen veri tabanında var manasına gelir.

Sonek ağacı, verilen bir kelimeye ait tüm sonekleri dizinleyen ağaca denir. Sonek ağacında erişim süresi sadece sorgunun boyutuna bağlıdır. Örneğin "bilişim" kelimesi yedi harften oluşmaktadır ve sonek ağacında "bilişim" kelimesini aradığımızda en çok yedi adım sonunda bilgi erişimi garanti edilir. Veri tabanının büyüklüğünün terabyte mertebesinde olması dahi arama süresini değiştirmez.



Şekil 4. Sonek Ağacı ile "BİLİŞİM" kelimesinin dizinlenmesi. Kök düğümden itibaren yaprak düğüme ulaşan her yol üzerindeki harflerin birleşimi kelimeye ait bir soneki ortaya çıkarmakta.

Şekil 4'ten de anlaşılabileceği üzere ağaçta sorgu ya da desen arama süreci hiçbir zaman desenin boyutundan fazla olamaz. Çünkü ağaç üzerinde kökten yaprak düğüme doğru hareket ederken, her düğüm erişiminde en az bir karakter okunur. Eğer ağaçta aranılacak desen *m* karakterden oluşuyorsa, en çok *m* düğüm erişimi sonunda aranılan desene ağaçta ulaşılır.

Sonek ağacı, en hızlı veri erişimini sağladığı için etkileyici bir dizinleme tekniğidir. Öte yandan, ağacın önemli dezavantajları da vardır. Bunlar kötü bellek yerleşimi, ağacın çok fazla yer kaplaması ve ağacın dengesiz yapısıdır. Bu sebeplerden dolayı ağacın oluşturulma süreci çok uzun sürebilir. Sonek ağacı konusundaki araştırmalar, bu sorunları çözmeyi amaçlar.

4.2 Tanımlamalar

Ağaç tarafından dizinlenmesi amacıyla S setinde k tane farklı harf serisi olduğunu farz edelim, öyle ki

$$S = \{S^1, S^2, ..., S^k\} \quad (1)$$

Veri setinde yer alan her bir harf serisi, S^J, Σ alfabesinde yer alan harflerden oluşsun. Formül olarak harf serisini şöyle tanımlayabiliriz;

$$S^J = \{s_1^j, s_2^j, ..., s_n^j\} \quad (2)$$

Yukarıdaki formülde bahsi geçen s_i^j, S^J harf serisine ait n sonekten herhangi birini simgeler. Normalde her harf serisinin aynı uzunlukta olmasını beklememek gerekir. Dolayısıyla kelime uzunluğunu temsil eden n sabit bir sayı olmayabilir.

Öte yandan alfabemiz, Σ,

$$\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_\Upsilon\} \quad (3)$$

şeklinde tanımlanmış olsun.

Sonek ağacı, S veri setinde yer alan tüm harf serilerinin soneklerini dizinleyen ağaçtır. Oluşan ağaçta her bir düğüm Υ adet alt düğüme sahip olabilir. Ancak ağaçların alt düğüm sayısı Υ değerinden daha az olabilir. Bu durumu Şekil 4'te ifade edilen ağaçta görebiliriz. Kök düğüm haricindeki düğümlerin alt düğüm sayısı Υ değerinden küçüktür. Ancak yeni sonekler eklenmesi sonucu, bir düğümün alt düğüm sayısı artabilir.

Ağaçta sonekler yaprak düğümlerle temsil edilmektedir. Öte yandan, içsel düğümler ortak önekleri temsil eder. Ağaca yeni bir sonek eklendiğinde iki farklı sonekin ortak öneki ortaya çıkabilir. Bunun sonucunda ortak önek için yeni bir düğüm oluşturulur. Şekil 4'te "işim" ve "im" soneklerinin ağaca eklendiği durumu düşünelim. Her iki sonekin ortak bir öneki vardır. Bu yüzden yeni bir düğüm "i" önekini temsil edecek şekilde oluşturulur.

Ağaç üzerinde ortak önek oluşumu eklenen soneklere bağlıdır yani veri setine göre ağacın yapısı şekillenir. Dolayısıyla ağacın yapısını sonekleri bilmeden tahmin etmek imkânsızdır. Aynı şekilde ağaç oluştuğunda her bir düğümün kaç adet alt düğümü içereceğini de tahmin edemeyiz. Önceden kestirilebilinen tek şey alt düğüm sayısının Υ değerinden daha fazla olamayacağıdır.

Sonek ağacına eklenen her bir sonek bir adet yaprak düğüm oluşturulmasına sebep olur. Veri setimizde j tane harf serisi olduğu ve her serinin de n karakter içerdiği varsayımı doğrultusunda, ağaçta toplam jn tane yaprak düğüm olacağını söyleyebiliriz. Matematiksel ifade edecek olursak; ağaçtaki yaprak düğüm sayısı, T_β^d şöyle temsil edilir.

$$T_\beta^d = jn \quad (4)$$

Ağaçta oluşacak içsel düğüm sayısını belirlemek içinse dengeli sonek ağancının var olduğunu farz edelim. T sonek ağacının, S setindeki verileri dizinlediğini varsayalım. Ağaç tamamen dengeli olduğu için en alt derinlik seviyesi olan d derinliğinde toplam σ^d düğüm olur. Dolayısıyla dengeli sonek ağacında toplam düğüm sayısını (5) ile belirtebiliriz.

$$T^d = 1 + \sigma^1 + \sigma^2 + ... + \sigma^d \quad (5)$$

Ağacın dengeli olmasını göz önüne aldığımız için d derinliğindeki tüm düğümlerin yaprak, geriye kalan düğümlerin ise içsel düğüm olduğu sonucuna varmak mümkündür. Dolayısıyla ağaçta oluşan tüm içsel düğümlerin toplamı:

$$T_\alpha^d = 1 + \sigma^1 + \sigma^2 + ... + \sigma^{d-1} \quad (6)$$

kadar olur. Ayrıca, Formül (6)'dan yola çıkarak ağaçta bulunan içsel düğüm sayısı ile yaprak düğüm sayısının yaklaşık olarak eşit olacağını kestirebiliriz. Formal olarak ifade edersek:

$$T_\alpha^d = 2jn = \sigma^d \quad (7)$$

Dolayısıyla T ağacında yer alan düğüm sayısının yaklaşık $4jn$ ya da σ^{d+1} adet olacağını söyleyebiliriz. Başka bir deyişle ağaçta oluşan düğüm sayısı, veri setinde yer alan kelime sayısının yaklaşık iki katı kadardır.

Sonek ağacında yer alan düğüm sayısının çok fazla olması başka bir sorunu da beraberinde getirir. En kısıtlı durumda bile her düğümün en az 17 byte yer kapladığı hesaplanmıştır(Wong, v.d., 2007). Dolayısıyla ağaçta yer alan bütün düğümleri dizinlemek oldukça büyük miktarda yer kaplar. Bu gerçek ağaç için önemli bir engeldir.

4.3 Sonek Ağacının Gelişim Süreci

1973 yılında Weiner'in lineer zamanda sonek ağacı inşa algoritması bu alanda yapılan önemli kilometre taşlarından ilkidir (Weiner, 1973). Weiner'in dikkat çektiği temel bir nokta klasik sonek ağacı oluşturmanın uzun zaman almasıdır. Zira her sonek ekleme işlemi için ağaçta arama işlemi yapılması gerekir. Her sonekin ortalama uzunluğu $\frac{n}{2}$ olduğu düşünülürse, S^J kelimesine ait n adet soneki ağacı ekleme işlemi O(n^2) zaman alır. Biyolojik verilerde n değerinin milyar mertebesinde olduğu düşünülürse klasik sonek ağacını oluşturmak gerçekçi olamaz.

Weiner bu probleme sonek linklerini kullanarak çözüm getirmiştir. Onun yöntemine göre ağaca sonekleri ardışık olarak eklerken arama işlemine gerek kalmaz. Şekil 5'te gösterildiği gibi sonek linkleri kullanıldığı takdirde bir sonraki düğüm oluşum noktası, arama yapmadan temin edebilir.

Weiner ağacı, ilerleyen yıllarda (McCreight,1976) ve (Ukkonen, 1995) algoritmaları ile daha etkin bir hale getirilmiştir. Öte yanda Bieganski ise çok sayıda DNA serisini bir ağaca eklemenin yöntemini araştırmış ve genelleştirilmiş sonek kavramını ortaya atmıştır (Bieganski ve Carlis, 1994). Günümüzde, Ukkonen'in sonek oluşturma ağacı önceki iki algoritmanın tüm pozitif özelliklerini barındırır. Bunlara ek olarak çevrimiçi işlem yapma yeteneğine sahiptir.

Sonek ağacı RAM belleğe sığdığı müddetçe çok iyi performans sunar (Gusfield, 1997). Ancak ağaç belleğe sığmadığı takdirde karşımıza önemli bir sorun çıkar. Ağacın belleğe sığmayan kısmının diske yazılması istenmeyen sonuçlar doğurur. Çünkü ağaç gövdesinde yer alan düğümlerinin hafızaya yerleşimi rastgeledir ve bu durum henüz engellenememiştir. Dolayısıyla aynı yol üzerinde yer alan düğümler disk üzerinde farklı disk sayfalarında yer alır. Ne yazık ki üst düğümden alt düğüme ulaşırken her defasında diskten bir sayfanın belleğe yüklenmesi gerekebilr. Bu şekildeki senaryo, sonek ağacının disk üzerinde çalışması esnasında başarısızlığa sürüklenebilir.

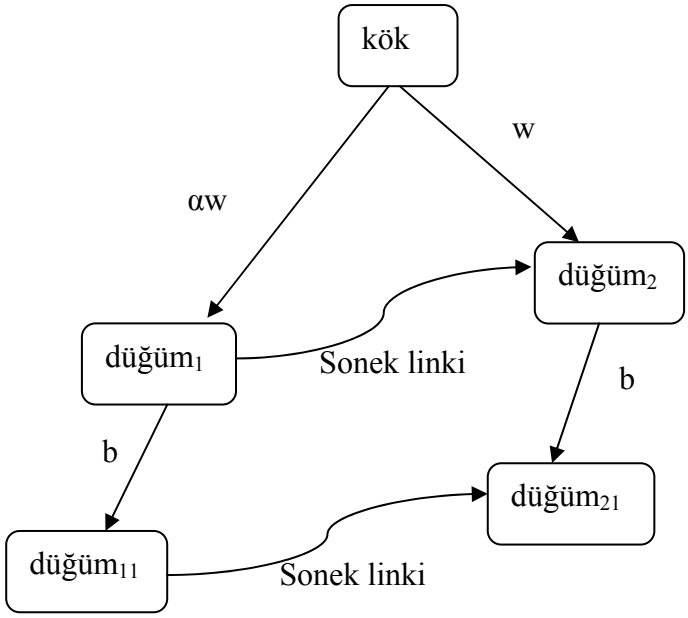
2000'li yılların başına kadar sonek ağaçlarının sadece belleğe sığan veriler için elverişli olduğu kanısı hâkimdi. Öte yandan daha fazla bellek ihtiyacı duyulan kelime işleme süreçleri için alternatif kelime işleme algoritmaları önerilmekteydi. Alternatif kelime dizinleme algoritmalarına Sonek Dizileri (Manber, 1993), Evirme dosyaları (Cormen, 1989) ve Kelime-B ağaçlarını (Ferragina, 1998) sayabiliriz. Ancak bu algoritmaların hiçbirisi çevrimiçi kelime işlemi yapmaya imkân tanımaz.

Bütün zorluklarına rağmen, son on yılda, sonek ağaçlarının disk üzerindeki uygulamalarının sayısı artmıştır(Kurzt, 1999). Ortaya çıkan bu ilginin en önemli sebebi ise mevcut algoritmaların hızlı ve dinamik olarak boyutu artan verileri dizinleme yeteneğinin yetersiz kalmasıdır.

Sonek ağacında diske erişimi azaltmak için üç temel değişiklik olasıdır. Birinci olarak düğümlerin bellekte kapladığı alanı azaltılabilir. Böylece daha fazla düğüm RAM belleğe sığar. İkincisi, ağaç içinde aynı patikada yer alan düğümlerin disk üzerinde aynı sayfaya yerleştirilmesidir. Böylece diskten getirilen bir tek disk sayfası ile belirli bir patikanın tüm düğümlerine erişilmiş olunur. Üçüncüsü ise alternatif bellek tamponlama yöntemleri önermektir. Sınırlı da olsa bellek alanı bazı disk sayfalarını hafızada saklayabilir. Dolayısıyla, ileride tekrar ihtiyaç duyulacak düğümleri hafızada tamponlamak diske erişimi azaltır. (Bedathur, 2004).

Disk tabanlı sonek uygulamalarının ilk örneklerinden birisi 1997 yılında Farach-Colton tarafından tasarlanmıştır (Farach, v.d., 1998). Lineer zamanda inşa edilen ağacın performansı teorik olarak ispatlanmış olmakla birlikte pratik uygulamada başarısız olmuştur. Öte yandan, (Hunt v.d., 2001) tasarladıkları PJama platformunda sonek linklerini devre dışı bırakarak farklı bir yöntem önermiştir. Bu yönteme göre kelimeler önek bilgilerine göre sıralandırıldıktan sonra ağaca belli bir düzenle eklenmiştir. Daha sonraki yıllarda, aynı temele dayalı yeni algoritmalar da önerilmiştir (Tian, v.d., 2005), (Wong, v.d., 2007). Ancak bu yöntemlerin hiçbirisi çevrimiçi sonek ağacı oluşturmayı hedef almamıştır. Yani dinamik olarak veri ekleme işlemi için uygun değillerdir.

2004 yılında, (Bedathur ve Haritsa, 2004) hızlı ve çevrimiçi özelliklerini sağlayan bir sonek ağacı tasarlamıştır.



Şekil 5. Sonek ağacında linklerin önemi. α, w, b en az bir harften oluşan kelimelerdir.Eğer ağaca αwb adlı sonek eklendiyse,bir sonraki adımda wb olan sonek eklenecektir. Bu öngörü ışığında bir sonraki sonek ekleme noktasına link yardımı belirlenebilir.

Bu araştırmacılar, Ukkonen algoritmasını temel alarak hem ağacın fiziksel yapısını irdelemiş hem de tamponlamanın ağacın performansına olan etkisini göstermişlerdir. Ancak araştırma diğerleri gibi sadece biyolojik verileri hedef almıştır. Dolayısıyla DNA'ya göre daha kısa boyutta olan müzik serileri için ağacın uygunluğu şüphelidir. Bu eksiklik (Ozcan ve Alpkoçak, 2008) tarafından incelenmiş ve alfabe seti büyüdüğü durumda mevcut yöntemlerin başarılı olamadığı görülmüştür.

Müziksel veriler ile biyolojik verileri arasında önemli farklar vardır. Birinci olarak, müziksel harf serilerinin boyutları biyoloji verilerine göre kısadır. Bunun yanı sıra, müziksel veri tabanında saklanan harf serilerinin sayısı binlerce, hatta milyonlarca olabilir. Oysa DNA serilerinde sayı çok azdır. Daha da önemlisi, müzik alfabesi daha fazla eleman içerir. Dolayısıyla sonek ağacının fiziksel yapısı müzik eserleri için farklıdır.

Düğümlerin sonek ağacında fiziksel yerleşim şekli, ağacın bellekte kaplayacağı alanı belirler. Dolayısıyla ağacın oluşma zamanına önemli oranda etki eder. Bu bölümde ağacın yerleşim biçimlerini açıklayacağız.

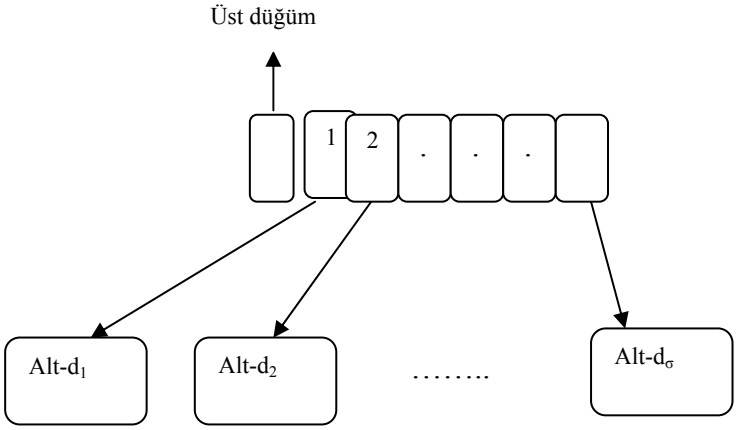
4.4.1 Düğümlerin Fiziksel Temsil Şekilleri

Sonek ağacında yer alan bir içsel düğümün alt düğüm sayısı öngörülemez. Bu sebeple alt düğümlerin adreslenmesi için farklı fiziksel düğüm temsil şekilleri denenebilir. En temel alt düğüm temsil şekilleri dizi tabanlı temsil ve liste tabanlı temsildir. Liste tabanlı düğüm temsili daha az bellek kullanımını sağlarken dizi tabanlı temsil alt düğüme daha hızlı erişim imkânını hedefler.

4.4.2 Dizi Tabanlı Düğüm Temsil

Dizi tabanlı düğüm temsili, alt düğüme doğrudan erişimi garanti eden bir yaklaşımdır. Şekil 6'da gösterildiği gibi, içsel düğümler Γ adet adres gösterici içerir. Düğüm içerisinde her bir gösterici σ^i ile başlayan alt patikasını dizinler.

Dizi tabanlı düğüm temsili için gerekli olan alanı Şekil 8-a'da gösterdik. Her bir içsel düğümün σ adet alt düğüm içermesi zorunluluğundan dolayı içsel düğüm toplam (3+σ) adres göstericisi içermek zorundadır. Kalan adres işaretçilerinden birisi üst düğüm adresini tutar. Son iki adres işaretçisi ise kenarlar tarafından temsil edilen kelimeleri dizinlemeyi sağlar.



Şekil 6. Dizi Tabanlı düğüm temsili: İçsel düğümün olası tüm düğümleri için üst düğümde yer ayrılmak zorundadır. Elbette rezerve edilen her düğümün kullanılması gerekmemektedir. Ancak; alfabe sayısının büyük olması, bu düğümün çok fazla yer kaplamasına sebep olabilmektedir.

Dizi tabanlı düğüm yapısı, alt düğümlere doğrudan erişim sağladığı için hızlı çalışabilir. Ayrıca yapısı itibarı ile basittir. Dolayısıyla Υ değeri küçük olan alfabelerde başarı göstereceği aşikârdır. Buna örnek olarak alfabesinde dört harf bulunan DNA'yı örnek gösterebiliriz.

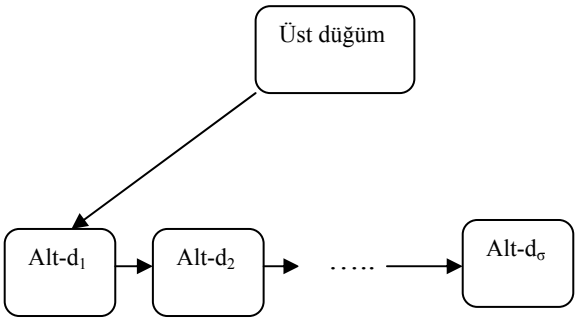
Öte yandan, alfabe boyutunun büyük olduğu durumlarda dizi tabanlı düğüm temsili önemli bir soruna sebep olur. Zira, düğüm boyutu gereksiz yere büyür. Örneğin alfabenin 129 olduğu MIDI müzik setinde içsel düğümlerin her biri (3+129)x4 = 528 byte yer kaplar. Bu durumda her disk sayfasına sadece sekiz tane düğüm sığabilir.

4.4.3 Liste Tabanlı Düğüm Temsili

Liste Tabanlı düğüm temsili, düğümlerin bellek kullanımını en aza indirmek üzere tasarlanmıştır. Elbette bunun bir bedeli vardır. Çünkü iç düğümlerin alt düğümlere doğrudan erişim imkânı feda edilmiştir. Buna göre içsel düğümler sadece bir alt düğüm adresini saklayabilir. Geriye kalan alt düğümlere erişmek için liste üzerinde hareket gerekir. Liste tabanlı düğüm yapısında erişim yöntemi Şekil 7'de gösterilmiştir.

Şekil 7'de gösterildiği gibi, üst düğümden alt düğümlere doğrudan erişim mümkün olmayabilir. Örneğin listenin en sonundaki düğüme erişmek için bütün alt düğümlere erişmek gerekir. Eğer alt düğümler disk üzerinde farklı bir sayfalarda yer alıyorsa, alt düğüme erişmek için Υ adet sayfa okunması gerekebilir. Basit bir alt düğüme erişim operasyonunun Υ adet sayfa erişim ihtiyacı, ağaç oluşumu için ciddi bir engeldir. Liste tabanlı düğümün yapısı Şekil 8-b'den takip edilebilir.

Bu düğüm yapısında üst düğüm adresini saklamak için ayrı bir gösterici tutulur. Böylece her düğüm için bir adet gösterici alanı tasarruf edilir. Öte yandan üst düğüm adresi listenin en sonunda yer alan kardeş düğüm tarafından saklanır. Zira, liste sonunda yer alan düğüm başka bir kardeşine referans göstermeyeceği için atıl durumdadır.



Şekil 7. Liste tabanlı Düğüm Temsili ile alt düğüme erişim stratejisi. Kardeş düğümler birbirlerine bir liste yapısı ile bağlı oldukları için üst düğümün sadece bir alt düğüm adresini bilmesi yeterlidir.

4.4.4 Hızlı ve Az Bellek Kullanan Fiziksel Düğüm Yaklaşımı

Bu bölümde hem belleği tasarruflu kullanan, hem de alt düğüme erişim zamanını makul bir süreye indiren fiziksel düğüm önerimizi tanıtacağız. Daha önceden belirttiğimiz gibi dizi tabanlı fiziksel yaklaşımlar, alfabesinde

128 eleman içeren müzik verileri için uygun olamaz. Dolayısıyla bu çalışmada liste tabanlı bir fiziksel düğüm yapısı çerçevesinde bir çözüm önermekteyiz.

Sonek ağacından yüksek performans alabilmek için üç temel noktaya dikkat etmek gerekir. Bunlar (1) üst düğüme hızlı erişim, (2) alt düğüme hızlı erişim ve (3) bellek optimizasyonudur. Üst düğüme hızlı erişimi sağlamak gayet basittir. Her düğüme üst düğüm adresini tutacak işaretçi alanı eklenerek bu sorun çözülebilir.

Belirtmek lazımdır ki bellekten fedakârlık yapmak sadece alfabe boyutu büyük olduğu zaman anlam kazanır. Öte yandan, alt düğüme hızlı erişim ise liste üzerinde yer alan kardeş düğümlerin erişim frekansını tahmin ederek temin edilebilir. Bellek optimizasyonu içinse dizi tabanlı fiziksel yaklaşımlardan uzak durmak gerekir.

Sonek ağacında kullanmak üzere tasarladığımız fiziksel yaklaşımı Şekil 8-c'de göstermekteyiz. Üst Düğüm Adresi Ekli Liste (ÜDAEL) yapısı, Liste tabanlı düğümden farklı olarak üst düğüm adresini saklamaktadır.

Alt düğüme erişim zamanını düşürmek için en çok aranan düğümleri en kolay ulaşılabilecek yerlere koymak gerekir. Şekil 7'deki örneğe geri dönecek olursak üst düğümden "Alt-d_1" adlı düğüme ulaşmak için gereken süre "Alt-d_σ" adlı düğüme erişmek için gereken süreden daha kısadır. Dolayısıyla en sık erişilme ihtiyacı duyulan kardeş düğümleri listenin en başına koymak performansı arttırır. Öte yandan seyrek erişilen düğümleri listenin en sonuna atmak çok fazla performans kaybına sebep olmaz.

Sonek ağacında sık erişilen düğümleri tahmin etmek mümkündür. Bunun için alfabenin en sık kullanılan karakterlerini belirlemek yeterlidir. Şekil 7'da yer alan kardeş düğümlerin her biri farklı bir alfabe elemanı ile devam eden patikayı ifade eder. Dolayısıyla veri setinde sık kullanılan harf ile ilintili düğümü listenin en başına yerleştirmek performansı arttırır.

MIDI müzik setinde yer alan her notanın frekansı aynı değildir. Örneğin çok kalın ya da çok tiz perdeye sahip notaların frekansı azdır. Dolayısıyla bu tip notalar ağaçta bir düğüm oluşumuna katkıda bulunduğunda kardeş listesinin en sonuna atılmalıdır. Örneğin, Tablo 1'de gösterildiği gibi perde değeri 70 ile 80 değerleri arasındaki notaların görülme sıklığı çok fazladır ve ilgili düğümlerinin listelerin baş tarafında yer alması ağacın oluşturulma süresini kısaltır.

5. TESTLER

5.1 Testlerin Kapsamı ve Amacı

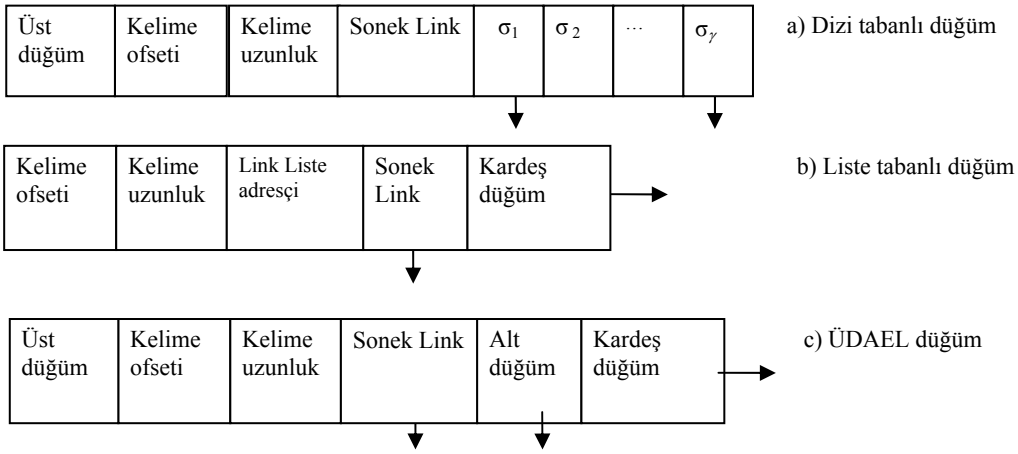
Bu bölümde, çevrim içi çalışan Sonek ağacının oluşturulmasına ilişkin testler irdelenmiştir. Çevrimiçi ağaç tasarımı sayesinde, ağaca sonek eklemeden önce veri analizine gerek bulunmamaktadır. Dolayısıyla oluşturulan ağaca sonradan eklemeler yapılabilmektedir.

5.2 Veri Seti

Sonek ağacı deneylerini yapabilmek için MIDI müzik serilerinden oluşan iki veri tabanı kullandık. Kullandığımız birinci veri tabanı monofonik müzik eserlerinden oluşmaktadır ve yaklaşık 4000 farklı müzik parçasından oluşmaktadır. Bu veri tabanında yer alan müzik eserleri toplam 250.000 notadan oluşmaktadır. Öte yandan, polifonik müzik eserlerini test edebilmek için ise farklı bir veri seri kullanmamız gerekmiştir. Bu amaçla 190 ülkenin milli marşlarını barındıran polifonik bir veri tabanı kullandık. Bunun yanı sıra veri tabanımıza, tanınmış filmlerin soundtrack'lerini içerenmüzik eserleri ile genişlettik. Sonuçta 13.7 MB büyüklüğünder ve 617 adet polifonik MIDI eserinden meydana gelen bir veri tabanı temin ettik.

5.3 Veri Setinin Sonek Ağacında Dizinlenecek Şekilde Düzenlenmesi

Testleri yapabilmek için, MIDI müzik formatındaki verilerin nota değerlerini karaktere dönüştürerek veri tabanımızı oluşturduk. Gayet tabiidir ki, MIDI alfabesi en çok 128 tamsayı değerini aldığı için, notalar karakter verileri olarak temsil edilebilir. Nota perdelerini karakterler ile temsil etmek için ise tamsayıların ASCII kodlarını kullandık. Örneğin MIDI dosyasındaki nota perdesi { 67, 68, 65} şeklinde ise bunun ASCII karşılığı olan dokuman dosya {A, B, @} şeklinde olur. Daha sonra harf serisine dönüştürülen MIDI dosyalarını sonek ağacına sırasıyla ekledik. Veri tabanında farklı müzik serilerinin birbirine karışmaması için her kelimenin sonuna özel karakter kullandık. Dolayısıyla alfabe sayısını 128'den 129'a çıkardık. Ağaç oluşturma testlerinin performansını karşılaştırmak için ise her deneyde ihtiyaç duyulan toplam sayfa erişim sayısını gözlemledik. Bilindiği üzere diske erişim gerektiren testlerde bellekte harcanan süre ihmal edilebilir (Salzberg, 1988).



Şekil 8. Fiziksel Düğüm Temsil Yöntemleri

İçinde 129 karakter yer alan bir alfabe, sonek ağaçları için oldukça ilginç bir konudur. Zira sonek ağaçları hakkında yapılan çalışmaların büyük bir çoğunluğu alfabesinde 4 ile 20 karakter barınan biyoloji verileri içermektedir. Oysa alfabedeki eleman sayısının artması sonek ağacının yapısında önemli bir ayrışmaya sebep olmaktadır.

Yapılan deneylerde sayfa erişim sayısı ölçülürken işletim sisteminin tamponlamaya etkisi önemli bir sorun teşkil edebilir. Önlem alınmazsa yapılan deneyler gerçekçi sonuçlar üretemez. Bu sorunu gidermek ve tamponlama deneylerini güvenilir kılmak için sayfa erişim sayısını emüle ettik.

Ağaç oluşturma mantığına göre her düğümün diskte yer alacağı adres ve sayfa numarası belli olacaktır. Dolayısıyla, düğümün yer aldığı sayfa okunduğunda, ilgili sayfa numarası tampona eklenerek bir emulasyon yapılabilecektir. Ağaç içinde ikinci bir düğüm okunduğunda ise ikinci bir sayfa no değeri hesaplanacaktır. Emülasyon programı, ikinci sayfanın tamponda yer alıp almadığını kontrol edecektir. Eğer sayfa yok ise Sayfa Erişim sayısını bir arttıracaktır. Ağacın oluşturulması sonucu meydana gelen Sayfa Erişim Sayısı deneylerin güvenilirliğini garanti altına alınacaktır.

Emülasyona ait tampon alanı aslında bir tamsayı dizisinden ibarettir. Bu dizide sayfa numaraları saklanır. Dizinin boyutu, tampona kaç sayfanın sığdığına bağlıdır.

Deneylerimizi temel olarak iki kısma ayırdık. Birinci kısımda tek sesli müzik eserlerini, ikinci kısımda ise çok sesli müzik eserlerini inceledik.

5.4 Tek Sesli Müziğin Dizinlenmesi

Tek sesli müzik eserlerinin dizinlenmesi için üç ayrı düğüm yapısını emülasyon programı ile kurguladık. Sonek ağacını her bir fiziksel düğüm yapısı için ayrı ayrı oluşturduk. Sonuçta her bir fiziksel düğüm yapısının disk kullanım ve sayfa erişim sayılarını karşılaştırdık.

Ağaç oluşturma esnasında alt düğüme, üst düğüme ve kardeş düğüme erişimler gerekmektedir. Sayfa erişim ihtiyacını doğuran da düğüm erişimleridir. Tablo 2'de ağaç oluşumu esnasında kaçar adet düğüm erişimi yapıldığı gösterilmektedir. Bu deney sonu göstermiştir ki fiziksel düğüm yapısı, toplam düğüm erişim sayısında farklılıklara sebep olmaktadır. Tablo 2'de gösterildiği üzere Dizi Tabanlı Düğüm yapısı toplamda daha az Düğüm Erişimi yapmaktadır. Bunun temel sebebi Dizi Tabanlı Düğüm yapısının Kardeş düğüme erişim gereksimi duymamasıdır.

5.4.1 Tampon Kullanmadan Ağaç Oluşturma

Ağaç oluşturma deneylerini yaparken ilk önce Tampon kullanmadık. Bu noktadan şu yorumu çıkarabiliriz: Dizi Tabanlı Fiziksel Düğüm, daha az düğüm erişimi yaptığına göre en az sayfa erişim sayısına ulaştıracaktır. Bu basit iddianın doğruluğu Şekil 10'da da görülebilir.

Ağacın bellekte kapladığı alan açısından baktığımızda ise dizi tabanlı düğüm temsili yöntemi oldukça kötü performans ortaya çıkarmaktadır. Şekil 9'da görülebileceği gibi dizi tabanlı düğüm kullanmak, belleğin açıkça sömürülmesine sebep olmaktadır.

Tablo 2. Ağaç oluşumu esnasında gereksinim duyulan toplam Düğüm Erişim Sayıları.

	Liste Tabanlı Düğüm	**Dizi Tabanlı Düğüm**	**ÜDAEL**
Alt Düğüm	719724	719724	719724
Üst Düğüm	383232	126453	256776
Kardeş Düğüm	3633864	0	2407703
	==========	==========	=======
Toplam	4736820	846177	3384203

Bir düğümün dizi tabanlı temsil ile bu kadar fazla alan kaplamasının tek sebebi MIDI alfabesinin 128 adet harf içermesidir. Bu veri ışığında şu yargılara varabiliriz:

1) - Dizi tabanlı düğüm temsili sadece alfabe içerisinde az sayıda harf varsa verimli olabilir.

2) - Dizi tabanlı düğüm temsili Tamponlamada verimsiz kalır.

Tampon kullanılmadan ağaç oluşturulduğunda Dizi Tabanlı Düğüm yapısının erişim sayısını en aza indirdiğini Şekil 10'da gösterilen deney de yinelemektedir. Tampon kullanmadan ağaç oluşturmada en kötü performansı ise Liste Tabanlı Düğüm Temsili sunmaktadır. Bizim önerimiz olan ÜDAEL yöntemi ise iki yöntemin ortasında performans göstermektedir. Dizi tabanlı düğüm temsilinin üstün performansının sebebi alt düğüme doğrudan ulaşabilmesidir. Oysa Liste tabanlı düğüm yönetiminde hem alt düğüme hem de üst düğüme erişmek için kardeş düğümlere erişmek gerekmektedir. Bu sebeple daha fazla sayıda disk sayfasının diskten okunmasına sebep olmaktadır. Bizim önerdiğimiz ÜDAEL sadece üst düğüme doğrudan erişim olanağı tanıdığı için iki yöntemin ortasında bir performans sergilemektedir.

ÜDAEL modelinin en iyi performansı verememesi başarısızlık olarak görülmemelidir. Çünkü dizi temsiliyle tasarlanan düğümlerin kapladığı devasa alan çok önemli bir sorundur ve dizi temsili düğüm tamponlama açısından verimsiz kalacaktır. Bu gerçeği daha net bir şekilde ifade edebilmek için aynı algoritmaların tamponlama kullanıldığındaki performansını sunmak gerekmektedir:

5.4.2 Tampon Kullanarak Ağacın Tekrar Oluşturulması

Tamponlama işlemi diskte yer alan belirli sayıdaki sayfanın bellekte saklanmasıdır. Tamponlamadan amaç yakın gelecekte diskten tekrar istenecek sayfaları bellekte tutmak ve böylece diskten erişilen toplam sayfa sayısını azaltmaktır (Tanenbaum, 2006). Bilindiği üzere diskten sayfa okumak ağaç oluşum performansını belirleyen asıl ve tek gerekçedir. (Salzberg, 1988).

Bellekte tamponlanan sayfa sayısı sınırlıdır. Dolayısıyla tamponda sayfa değiştirme stratejileri uygulamak gerekmektedir. Deneylerde kullandığımız sayfa değiştirme stratejileri şunlardır:

LRU: Yeni sayfa okunduğunda en uzak geçmişte kullanılan sayfayı tampondan çıkarır (Tanenbaum, 2006).

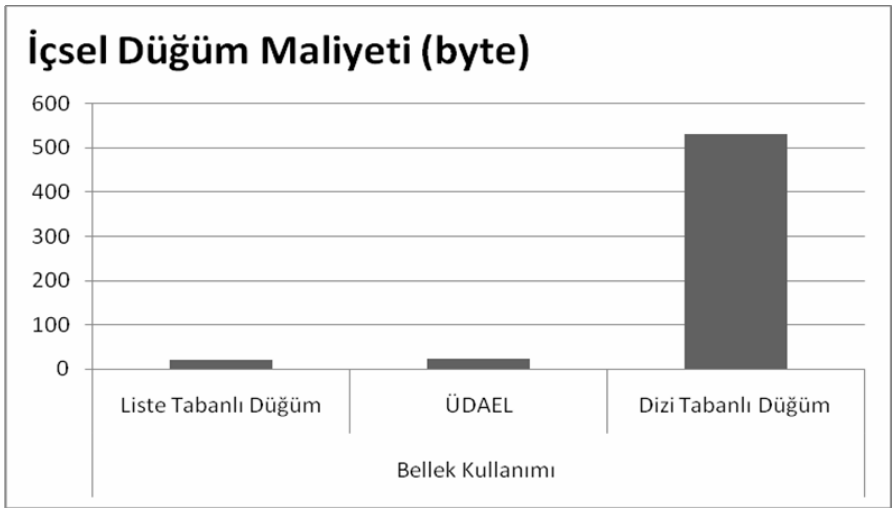
FIFO: İlk giren ilk çıkar prensibine uygun olarak ilgili sayfayı tampondan atar.

TOP_Q: Kökten uzakta yer alan düğümleri içeren sayfayı atar. Yeni gelen sayfayı hemen atmamak için geçici bir FIFO tampon da içerir (Bedathur, 2004).

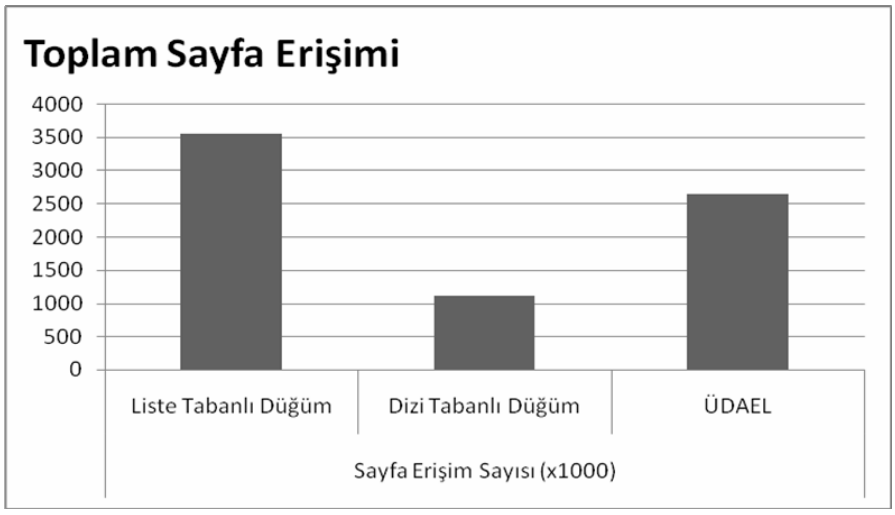
Fiziksel yerleşim algoritmalarını tamponlama kullanarak tekrar test ettiğimizde önerdiğimiz ÜDAEL yaklaşımının en iyi sonucu verdiğini gözlemledik. Testleri yapmak için belleğin 1024 sayfayı tamponlamasına imkân verdik. Deney sonucunda kullanılan tamponlama yöntemi ne olursa olsun; bizim yaklaşımımız MIDI müzik setinin dizinlenme süresi açısından en iyi sonucu verdi. Bu sonuçlar Şekil 11, 12 ve 13'te gösterilmiştir.

Dizi Tabanlı Düğüm yaklaşımının tamponlama söz konusu olduğunda geri kalmasının gayet açık bir sebebi vardır. Bir adres göstericinin dört byte yer kapladığını ve bir sayfanın 4096 byte olduğunu hesaba katarsak, bir disk sayfasına sadece 7 adet düğüm sığacağı gerçeği ile karşılaşırız. Dolayısıyla aynı sayfadan başka bir düğüme kısa süre içerisinde ulaşma ihtimali son derece düşüktür.

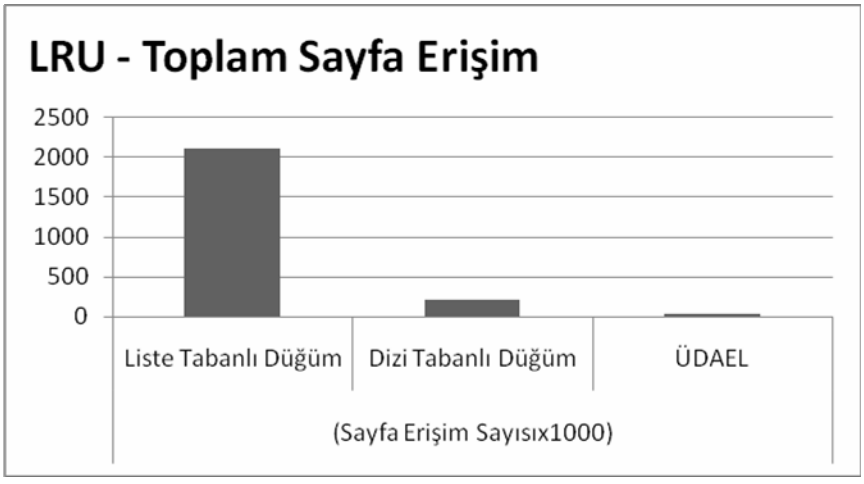
Tamponlama kullanılması durumunda Liste Tabanlı Düğüm yapısının da ciddi oranda performans artışı yapmaktadır.



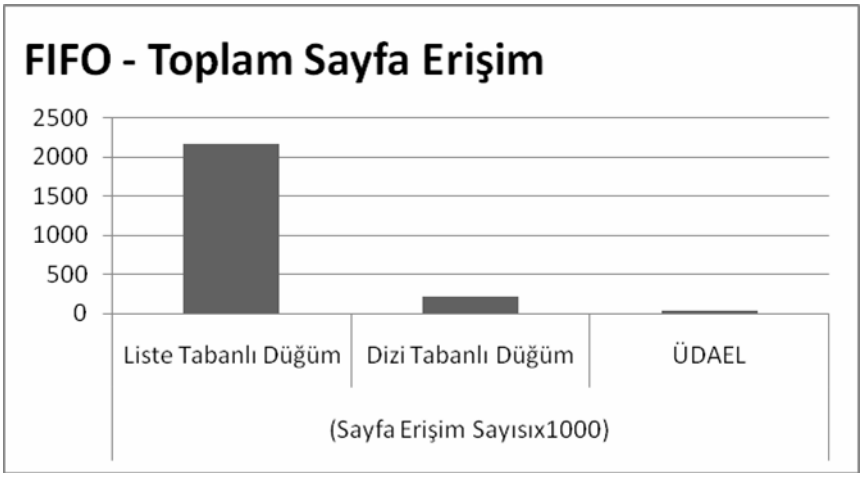
Şekil 9. Fiziksel düğüm yapıları



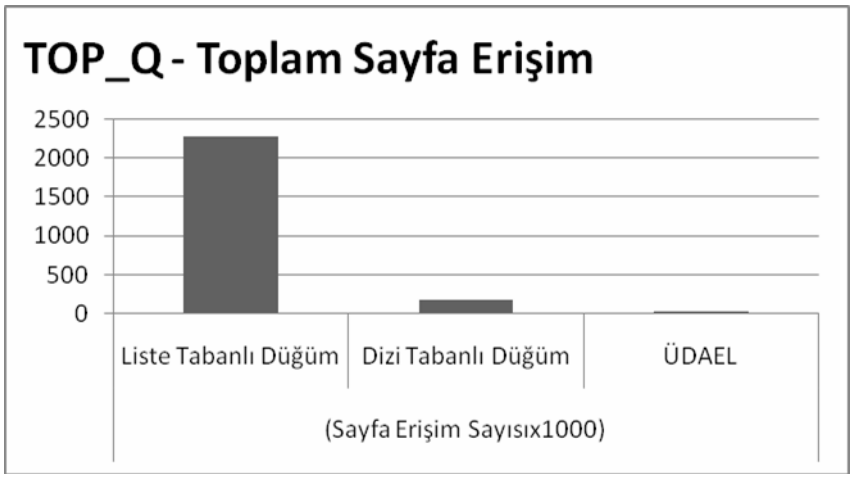
Şekil 10. Düğüm Yapılarının sayfa erişimi



Şekil 11. LRU Tamponlama kullanılması durumunda fiziksel yerleşim yaklaşımlarının karşılaştırılması



Şekil 12. FIFO Tamponlama kullanılması durumunda fiziksel yerleşim yaklaşımlarının karşılaştırılması



Şekil 13. TOP_Q tamponlama kullanılması durumunda fiziksel yerleşim yaklaşımlarının karşılaştırılması

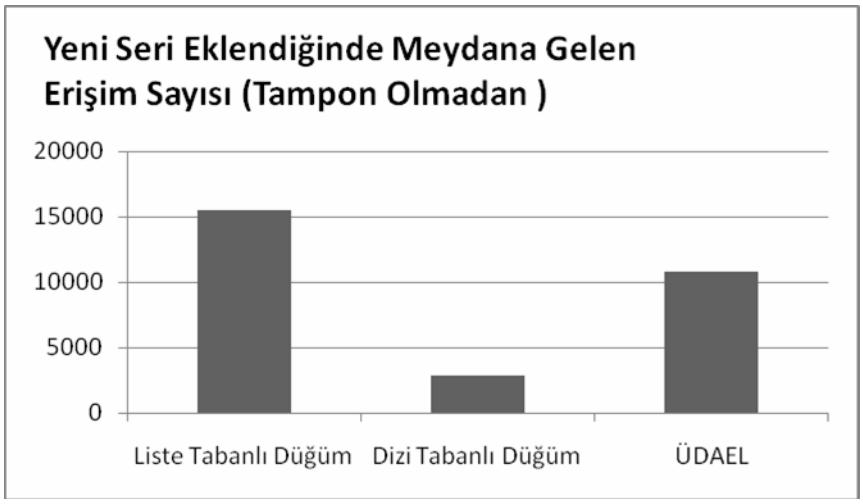
Ancak bu artış oranı Dizi Tabanlı Düğüm kadar olamayacaktır. Zira Liste Tabanlı Düğüm yapısı üst düğüme erişmek için çok fazla efor sarfetmektedir. Ağaç yapısı oluşurken seyrek kullanılan düğümler listenin en sonuna atıldığı ve üst düğüm adresi listenin sonunda yer aldığı hesaba katılırsa bu durumun sürpriz olmadığı anlaşılabilir.

Araştırmalarımız göstermiştir ki sonek ağacında alfabenin büyüklüğü performans üzerinde kritik bir öneme sahiptir. Alfabe sayısının çok olduğu koşulda içsel düğümlerin muhtemel alt düğüm sayısı da aynı oranda artmaktadır. Dolayısıyla Sonek Ağacının geniş alfabeli veri setlerinde uygulanması daha fazla maliyet doğurmaktadır.

5.5 Sonek Ağacına Yeni Serilerin Eklenmesi

Çevrimiçi sonek ağacını farklı kılan özellik, ağaca sonradan veri eklenebilmesidir. Deneyin bu safhasında oluşturulan sonek ağacına yeni serilerin eklenmesinin performansa etkisini analiz ettik.

Sonek ağacı oluşturulduktan sonra ağaca 10 adet yeni MIDI serisi daha ekledik. Eklenen son MIDI serilerinin performansa etkisini test ettik. Performans analizi için yine sayfa erişim sayısını emulasyon yöntemi kullanarak saydık. Deney sonuçları Şekil 14 ve Tablo 3'te gösterilmiştir. Deney sonuçları göstermektedir ki ÜDAEL tamponlama kullanıldığında en iyi sonucu vermektedir.



Şekil 14. Ağaca ek serilerin eklenmesi durumunda oluşan sayfa erişimleri (Tampon kullanmadan)

Tablo 3. Ağaca ek serilerin eklenmesi durumunda oluşan sayfa erişimleri (Tampon kullanarak)

SayfaErişim Sayısı (1024 Sayfa)	**Liste Tabanlı Düğüm**	**Dizi Tabanlı Düğüm**	**ÜDAEL**
LRU Tampon	6283	86	23
FIFO Tampon	6490	319	35
TOP_Q Tampon	7971	471	52

5.6 Çok Sesli Müzik Eserlerinin Dizinlenmesi

Çok sesli müzik eserlerinin klasik sonek ağacı yardımıyla dizinlenmesi mümkün değildir. Çünkü klasik sonek ağacı çok boyutlu serilerini dizinleyecek kabiliyete sahip değildir. Dolayısıyla, çok sesli veri setlerini dizinleyebilmek için bu verilerin öncelikle tek sesli müziğe çevrilmesi gerekir.

Veri setini tek sesli müziğe çevirmek için müzik literatüründe yer alan Skyline algoritmasından faydalandık (Uitdenbogert). Bu yönteme göre birden fazla nota eşzamanlı olarak çaldığı takdirde düşük perdeli olan notalar veri setinden atılarak tek sesli müzik elde edilir.

Tek sesli müziğe çevirme esnasında melodiyi belirleyen notaların yanlışlıkla veri setinden atılma ihtimali vardır. Böyle bir problemi engellemek için her bir çalgı aletinin notaları ayrı bir dosyada saklanır. Dolayısıyla çok sesli bir MIDI müzik dosyası çalgı alet sayısına bağlı olarak 16 farklı seride temsil edilebilir. Elbette ki her bir çalgı aleti tarafından çalınan notalar da çok sesli olabilir. Ancak tek çalgının bulunduğu bir nota serisinde Skyline Algoritmasını kullanmak melodi kaybına sebep olmaz.

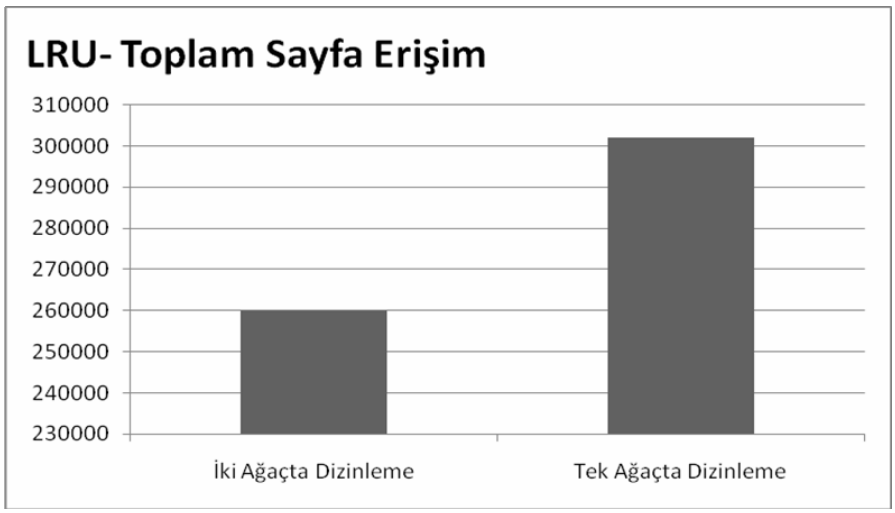
Müziksel nota serilerinin çalgı aletine göre sınıflandırılması bize bazı temel avantajlar sağlar. Örneğin sınıflandırılmış nota serileri sayesinde, MIDI müzik veri tabanı birden çok sonek ağacı kullanılabilir. Dolayısıyla, nota özellikleri açısında birbirine benzemeyen iki farklı nota serisinin farklı sonek ağaçlarında dizinlenmesi ağacın performansına olumlu katkıda bulunur.

Müzik eserlerinde vurmalı çalgı tarafından çalınacak notalar dokuz numaralı MIDI kanalında saklanır(Ghias, v.d., 1995). Öte yanda vurmalı çalgı kanalına ait notalar melodik kanallara ait notalardan faklı bir özellik gösterir; daha küçük perdeler içerirler. Dolayısıyla farklı notaları kullanan nota serilerini farklı bir sonek ağacında saklamak iyi sonuç verebilir.

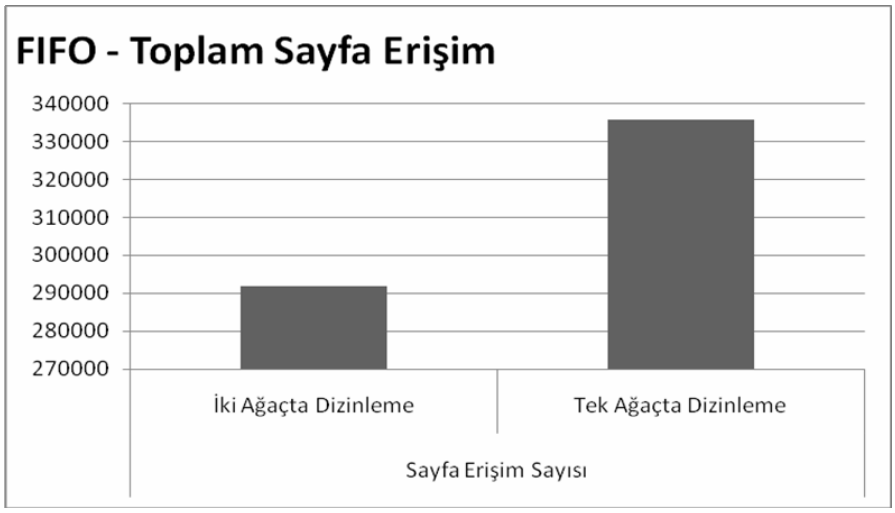
Önerimizi ispatlamak için birinci adımda müzik eserlerinin tamamını bir sonek ağacında dizinledik ve toplam sayfa erişim sayfasını gözlemledik. Önerimizin ikinci adımında müzik verilerini iki sınıfa ayırdık. Birinci sınıfta dokuzuncu kanalda saklanan vurmalı çalgı notalarını, ikinci sınıfta ise geriye kalan tüm nota serilerini sakladık. Son olarak her bir sınıfa ait nota serilerini ayrı bir sonek ağacı ile dizinledik. Dolayısıyla veri setini iki ağaç kullanarak dizinlemiş olduk. Ağacı dizinlemek içinse ÜDAEL fiziksel yerleşim tekniği kullandık.

Şekil 15, 16 ve 17'da gösterilen deney sonuçları, veriyi sınıflandırarak birden çok sonek ağacı kullanmanın daha az sayıda sayfa erişimine sebep olacağını gösterir. Şekillerde de görüldüğü üzere, iki ağacın oluşumu esnasında meydana gelen toplam sayfa erişim sayısı, veri setinin tamamını tek ağaçla dizinlemesi durumundakinden daha azdır. Bu yüzden de iki ağaç kullanmak daha başarılı sonuç verir.

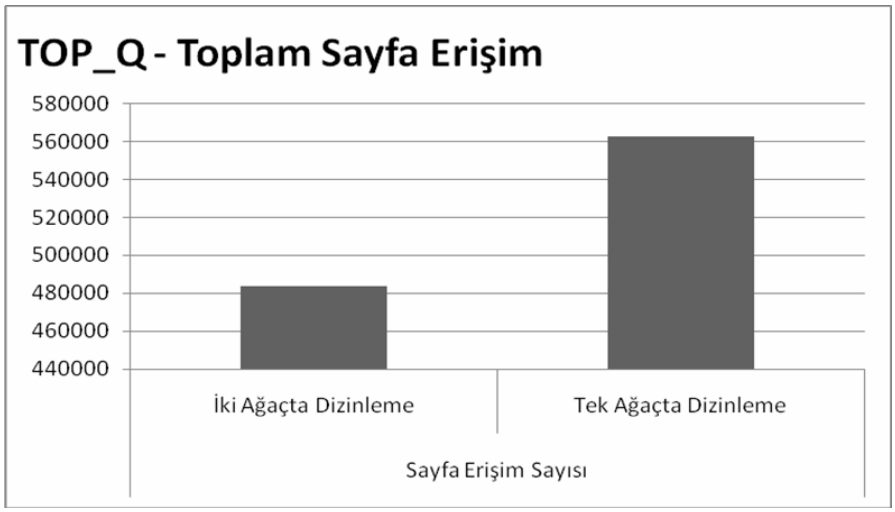
Veri setini sınıflandırarak iki ağaçta dizinlemenin performans arttırması bize göre tesadüf değildir. Zira sınıflandırılırmış veri setlerinde kullanılan alfabe sayısı azalmıştır. Dolayısıyla içsel düğümlerin daha az sayıda alt düğümleri oluşur ve alt düğüme erişim zamanında ciddi bir azalma sağlanır. Öte yandan veri setinin sınıflandırılmadan tek ağaç ile temsil edilmesi durumunda düğümlerin ortalama alt düğüm sayısı çok fazladır. Dolayısıyla ortalama alt düğüme erişim zamanı artar.



Şekil 15. LRU tampon kullanılması durumunda veri sınıflandırma ve çok sayıda ağaç kullanmanın performansa katkısı.



Şekil 16. FIFO tampon kullanılması durumunda veri sınıflandırma ve çok sayıda ağaç kullanmanın performansa katkısı.



Şekil 17. TOP_Q tampon kullanılması durumunda veri sınıflandırma ve çok sayıda ağaç kullanmanın performansa katkısı.

6. SONUÇ

Bu çalışmada çevrimiçi çalışan sonek ağacının efektif oluşturulması için yapılan önermeler anlatılmıştır. Özellikle alfabesindeki eleman sayısı büyük olan veri setleri için sonek ağacı oluşturmak farklı problemleri gündeme getirmektedir. Çalışmada MIDI müzik seti verileri sonek ağacında dizinlenmiştir. Bu amaçla ağacın fiziksel yapısında daha optimize bir yöntem önerilmiştir. Önerilen yöntem alfabe sayısı yüksek olmak ve tampon kullanmak koşulu dâhilinde en üstün performans vermektedir.

Biyoloji, tıp, jeofizik, ekonomi, müzik bilim dalları her saniye yeni bir veri setini üretmekte ve bunlar bir veri tabanına kaydedilmektedir. Aynı şekilde hacmi artan veri tabanlarında benzerlik bulma ihtiyacı da aynı oranda artmaktadır. Dolayısıyla, değişen şartlara göre veri tabanlarında hızlı sorgulama yapacak algoritmalara ihtiyaç duyulmaktadır. Bu çalışmada müzik veri tabanlarında bilgi erişim yöntemlerinden bahsettik ve MIDI dosyalarına ait verileri sonek ağaçları ile dizinledik. Dizinleme için Sonek Ağacını tercih etmemizin sebebi, harf serilerinden oluşan veri setlerinde en hızlı sorgulamayı temin etmesinden kaynaklanmaktadır. Öte yandan Sonek Ağaçlarının oluşturulması çok fazla zaman almaktadır. Bu çalışmada MIDI müzik verilerinin Sonek Ağacına dizinlenmesi yöntemlerini araştırdık.

Bizim tahminlerimize göre, teknolojik gelişmeler Sonek Ağaçlarının kullanım oranını arttıracaktır. Bunun iki temel sebebi vardır. Birincisi, bilgisayarların RAM bellek kapasitelerinin artacağı beklentisidir. Dolayısıyla daha büyük bellek hacmine sahip bir bilgisayarda, sonek ağacı oluşturulması işlemi diske erişim ihtiyacını azaltacak ya da ortadan kaldıracaktır. Çünkü bellekte tamponlanabilecek disk sayfa sayısı da belleğin büyüklüğüne paralel olara artacaktır. Teknolojik olarak Sonek Ağaçlarını etkin kılacak diğer gelişme ise SSD yani Katı Durum Disk'lerinin bulunmasıdır. Zira günümüz diskleri, mekanik sınırlamalar dolayısıyla hızlı okumaya ve yazmaya imkân tanımamaktadır. Oysa SSD'lerde mekanik parça bulunmadığından diskten okuma / yazma işlemleri çok daha hızlı bir şekilde yapılabilecektir. Dolayısıyla sonek ağaçlarının en büyük engeli olan disk erişim problemi önemini kaybedecektir. Bu bilgiler ışında, bize göre sonek ağacı, ileride oldukça yaygın kullanılan bir veri yapısı olacaktır.

KAYNAKLAR

Bainbridge, D., Nevill-Manning, C.G., Witten, I.H., Smith, L.A. and McNab, R.J. (1999). Towards a digital library of popular music. *Proceedings of the fourth ACM conference on Digital libraries*, ss. 161 – 169.

Bedathur, S. ve Haritsa, J. (2004). Engineering a fast online persistent suffix tree construction, *Proceedings of the 20th International Conference on Data Engineering*, ss. 720–731.

Bieganski, J.R.P. ve Carlis, J.V. (1994). Generalizedsuffix trees for biological se-

quence data: Applicationand implantation, *Proceedings of 27th HICSS*, IEEE, Computer Society, ss. 35–44.

Boyer, R.S. ve Moore, J.S. (1977). A fast string searching algorithm. *ACM Comm.* 20, 762-72.

Cormen, T.H., Leiserson, C.E. ve Rivest, R.L. (1989). *Introduction to Algorithms* The MIT Press, Boston.

Dowling, W. (1982). Melodic information processing and its development, *The Psychology of Music*, Ch. 13, 413- 429. Academic Press, Inc.

Farach, M., Ferragina, P. ve Muthukrishnan, S. (1998). Overcoming the Memory Bottleneck in Suffix Tree Construction, *Proceedings of 39th Symp on Foundations of Computer Science, IEEE Computer Society Press*, ss. 174-185.

Feridunoglu, L. (2004). *Müziğe giden yol*. İnkılap yayınevi, İstanbul.

Fingerhut, M. (2004). Music Information Retrieval, or how to search for music and do away with incipits, *IAML - IASA Congress*.

Ghias, A., Logan, J., Chamberlin, D. ve Smith, B. (1995). Query by humming Musical information retrieval in an audio databases, *Proceedings of ACM Multimedia.*

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences Computer Science and Computational Biology*. Cambridge Univ. Press, Cambridge.

Hunt, E., Atkinson, M.P. ve Irving, R.W. (2001). *A Database Index to Large Biological Sequences*, *Proc. of 27th Int'l Conf. Very Large Data Bases*, ACM Press, ss. 139-148.

Knuth, D.E., Morris, J.H. ve Pratt, V.B. (1977). Fast Pattern Matching in strings. *SIAM Journal Comput*ing (6), 323-350.

Kurtz, S. (1999). Reducing the Space Requirement of Suffix Trees, *Software, Practice and Experience* 29(13), 1149–1171.

Lemström, K. (2000). String Matching Techniques for Music Retrieval, *PhD thesis*, University of Helsinki.

Manber, U. ve Myers, G. (1993). Suffix arrays: a new method for on-line string searches, *SIAM Journal on Computing* 22 (5).

McCreight, E.M. (1976). A space-economical suffix tree construction algorithm. *Journal of the ACM* 23(2), 262–272.

Özcan, G., Işıkhan, C. ve Alpkocak, A. (2005). Melody Extraction on MIDI Music Files, *Proceedings of ISM*, IEEE Computer Society Press, California.

Özcan, G. ve Alpkoçak, A. (2008). Online Suffix Tree Construction for Streaming Sequences*, Communications on Computer & Information Sciences*, Vol. 6, Part 1 0f 2, Ed: H.S. et al. ss.69-81, Springer Verlag, Berlin-Heidelberg.

Phoophakdee, B. ve Zaki, M. (2007). Genome-scale Disk based Suffix Tree Indexing, *Proceedings of ACM SIGMOD*, ACM Press.

Salzberg, B. (1988*). File Structures: An Analytic Approach*. Prentice-Hall, New Jersey.

Tanenbaum, A. (2006). *Operating Systems, Operating Systems Design and Implementation*, 3rd Edition, Prentice Hall, New Jersey.

Tian, Y., Tata, S., Hankins, R.A. & Patel, R.M. (2005). Practical methods for constructing suffix trees, *The VLDB Journal*.

Temperley, D. (2001). *The Cognition of Basic Musical Structures*, Boston, The MIT Press.

Uitdenbogerd, A. & Zobel, J. (1999). Melodic Matching techniques for large music databases, *Proceedings of ACM International Multimedia Conference*, ACM Press.

Ukkonen, E. (1995). On-line construction of suffix-trees*, Algorithmica*.

Weiner, P. (1973). Linear Pattern Matching Algorithm, *Proceedings of 14th IEEE Symposium on Switching and Automata Theory*.

Wong, S., Sung, W. ve Wong, L. (2007). CPS-tree: A compact Partitioned Suffix Tree for Disk based Indexing on Large Ge-

nome Sequences, *Proceedings of IEEE ICDE*, Istanbul.

General MIDI - Wikipedia the free encyclopedia,http://en.wikipedia.org/wiki/General_MIDI.

Yottabyte – Wikipedia the free encyclopedia, http://en.wikipedia.org/wiki/Yottabyte.

Gıyasettin ÖZCAN, 1974 yılında Elazığ'da doğdu. Lisans, yüksek lisans ve doktora derecelerini sırasıyla İstanbul Üniversitesi, Massachusetts Üniversitesi ve Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği Bölümlerinde tamamlamladı. Halen Dumlupınar Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümünde Öğretim Üyesi olarak çalışmaktadır. İlgi duyguğu çalışma alanları Bilgi Erişim Sistemleri, Metin İşleme, Bioinformatik ve Yapay Zeka'dır.

Adil ALPKOÇAK, 1965 yılında Aydın'da doğdu. Lisans derecesini Hacettepe Üniversitesinde, Yüksek Lisans ve Doktora derecelerini ise Ege Üniversitesi Bilgisayar Mühendisliği Bölümünden aldı. Halen Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği Bölümünde Öğretim Üyesi olarak görev yapmaktadır. Başlıca araştırma ve ilgi alanı resim, müzik ve video için içerik Tabanlı Çoklu ortam erişim sistemleridir.