Minimum Çevreleyen Çember (MEC) ve B-Spline Problemleri

Ali Mohamed Osman Ali Elbashir
Bilgisayar Mühendisliği Bölümü
Kocaeli Üniversitesi
İzmit, Kocaeli
alielbashiir@gmail.com

***Özet*—Bu çalışmada kullanıcı tarafından tamsayı koordinatlı 2 boyutlu bir düzlemde sayısı bilinmeyen N nokta verildiğinde, o noktaların en yakınından geçen eğriyi ve tüm noktaları içeren minimum çevreleyen yarıçaplı dairenin merkezini ve yarıçapını hesaplanmasını ve ekrana çizdirilmesini C dili kullanarak gerçekleştirdim. MEC problemi için kullandığım algoritmayı nasıl seçtiğimden bahsettim. Yazdığım C programın zaman karmaşıklığı analizini yaparak Big O zaman karmaşıklığını buldum.**

***Anahtar kelimeler*—Dinamik bellek yönetimi, Struct, Zaman karmaşıklığı, Lineer zaman, MEC, B-spline eğrisi, Knot, Knot vektörü, Welzl algoritması, De Boor algoritması**

I. GIRIŞ

Bu çalışma için, 2 boyutlu düzlemde P nokta kümesi için, verilen n noktaları içeren ve matematiksel olarak en küçük yarıçapıya sahip çember "Minimum Enclosing Circle (MEC)"in merkezini ve yarıçapını hesaplamak istendi. Onunla birlikte, bilgisayar destekli tasarımda eğimli yüzeyleri temsil etmek için ve birçok tasarım programında kullanılan B-spline Eğrisini kullanarak o noktaların en yakınından geçen eğriyi çizdirmek istendi.

Minimum Enclosing Circle problemi birkaç farklı yöntem ile çözülebilir; Ancak bu çalışmada kullanılacak algoritmayı seçmek için kantitatif kriter olarak algoritmanın zaman karmaşıklığı, kalitatif ise uygulanma zorluk derecesi alındı. O iki kriteri göz önünde bulundurarak, O(n) (Lineer zamanda çalışan) Big O zaman karmaşıklığına sahip, orta uygulanma zorluğu olan Emo Welzl geliştirdiği algoritmayı [1] seçtim.

B-spline'i çizmek için Carl De Boor geliştirdiği algoritmayı [2] seçtim.

Yazdığım C programında, girdi olarak CSV dosyadan koordinat düzleminde sayısı bilinmeyen x ve y noktaları alıp, C dilinin dinamik bellek yönetimi ve Struct yapısından yararlanarak bir nokta diziye atıyor. O noktaları Welzl algoritmasına girdi olarak verilip, MEC'in merkez noktası ve yarıçapı elde edilir. B-spline için De Boor algoritmasına o noktaları verilip, eğride kalan noktaları hesaplanınca bellekte saklamadan çizilir.

II. YÖNTEM

A. *Girdi Ayrıştırma*

Noktaları temsil etmek için iki double tipinden x ve y değişkene sahip olan Nokta adlı bir struct oluşturulur. Girilecek noktalar tamsayı olacağına rağmen, MEC'i ve B-spline'i hesaplarken reel sayılar kullanması gerektiği için double ile tanımlamaya gerek buldum.

Girilen noktalar CSV formatında .csv uzantılı kullanıcı tarafından verilen dosya adresinden okunur. O noktalar, malloc() ve realloc() fonksiyonları kullanarak 2 dinamik diziye atılır (noktalar1 ve noktalar2). Her pointer kullanıldığında bir bellek kontrolü yapılır. Kontrol olumsuz ise, hata yazdırılıp, 0 olmayan return kodu ile sonlandırılır. Olumlu ise veri okumaya devam edilir.

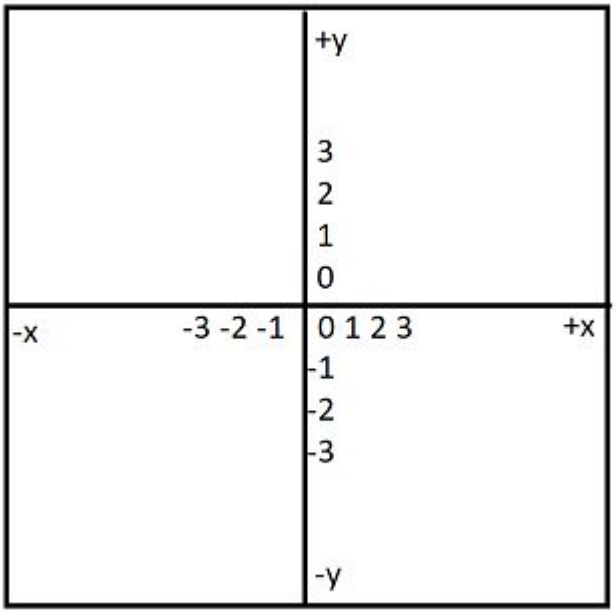
Her nokta okunduktan sonra, dosyayi_oku() fonksiyon noktaların sayısını döndürüyor ve Main()'de m değişkeninde tutulur.

B. *Girdi Biçimlendirme*

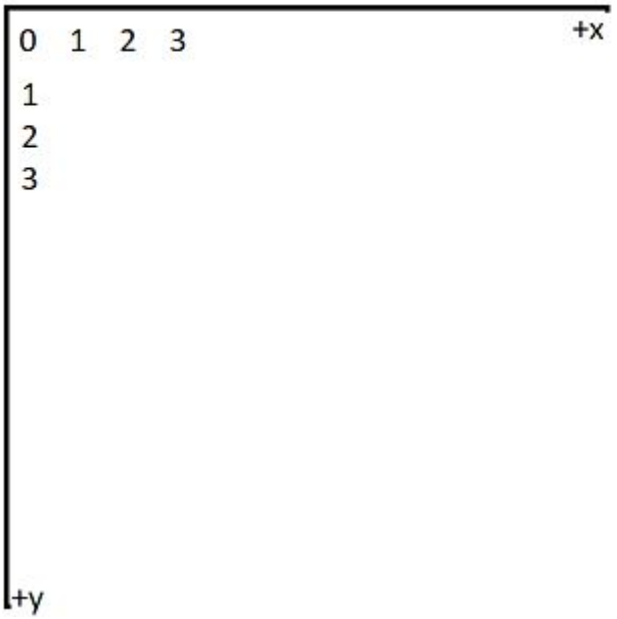
Girilen noktalar, X ve Y düzlemde herhangi bir pozitif veya negatif tamsayı kombinasyonu olabilir. Graphic kütüphanesi olarak Allegro 5 kullanıldığı için, o noktaları çizilmeden önce Allegro koordinat sistemine dönüştürülmesi gerek. Şekil 1 ve Şekil 2 koordinat sistemlerinin farkını gösteriyor.
Noktaları dönüştürmek için aşağıdaki formül kullanılır.

```
a.x = (genislik / 2.0) + a.x * sfx;
a.y = (yukseklik / 2.0) - a.y *
sfy;
```

Burada sfx ve sfy, pencerenin genişliği ve yüksekliği çarpı belirtilen maximum x ve y nokta değerleridir.



Şekil 1 Girilen noktaların olduğu koordinat sistemi



Şekil 2 Allegro koordinat sistemi

C. *Minimum Çevreleyen Çember Hesaplama*

Minimum Çevreleyen Çember için Welzl algoritmasını kullanarak MEC'i hesaplanır. Welzl rekürsif bir algoritmadır. O algoritmanın kabakodu Şekil 3'te gösterilir. C'de Python list'i veya Java ArrayList'i gibi büyütülebilen/küçültebilen array standart olarak olmadığı için, C'ye daha uygun bir algoritma yazıp onu kullandım. Kullandığım algorıtmanın kabakodu Şekil 4'te.

Klasik Welzl algoritmasında, her döndüğünde P kümesinin karıştırılmasını gerektiriyor, ama Welzl kendisi de bu karıştırma işlemin bir kere yapılması yeterli olduğunu söyledi [1], ve programda öyle yazıldı.

```
input: Finite sets P and R of points in the plane |R|≤ 3.
output: Minimal disk enclosing P with R on the boundary.

if P is empty or |R| = 3 then
    return trivial(R)
choose p in P (randomly and uniformly)
D := welzl(P - { p }, R)
if p is in D then
    return D

return welzl(P - { p }, R ∪ { p })
```

Şekil 3 Welzl Algoritması

```
Welzl(P, m, S, n)
    girdi: P Nokta dizisi, m nokta sayısı, S Nokta dizisi, n nokta sayısı
    çıktı: P noktaları içeren, S noktaları sınırında olan en küçük çember
    // NOT: Çalıştırmadan önce P'yi karıştır
    eğer P boş ise veya S'de 1den fazla nokta varsa
        return cember(S)
    mec = welzl(P, m-1, S, n)
    eğer p mec'in içinde değilse
        S[n] = P[m-1]
        n++
    return welzl(P, m-1, S, n)
```

Şekil 4 Değiştirilmiş Welzl Algoritması

D. *B-spline Hesaplama*

B-spline'i hesaplamak için şekil 5'te gösterilen De Boor fonksiyon kullanıldı.

$$\mathbf{S}(x) = \sum_{i=k-p}^{k} \mathbf{c}_i B_{i,p}(x).$$

Şekil 5 De Boor algoritması

x: zaman aralığı
k: x'in içinde bulunduğu knot aralığın indisi,
p: spline derecesi. Daha noktaları yakından geçmesi, ve zaman karmaşıklığı küçük olduğu için 2 olarak tanımladım (Quadratic).
c: kontrol noktaları
B: B-spline baz fonksiyonu

E. *Sonuç Gösterme*

MEC için onu bir kere hesaplayıp, merkez noktası ve yarıçapını Cember adlı struct yapısıyla bellekte sakladıktan sonra ekrana çizdirilir.
B-spline'da eğride kalan tüm noktaları hesaplamak gerektiği için, fazla bilgisayar kaynak kullanmamak için her nokta hesaplandığında bellekte tutulmadan aniden ekrana çizdirilir.

III. KABA KOD

1. verilen CSV dosyayı oku
2. okunan noktaları, noktalar1 ve noktalar2 nokta dizilerine at
3. nokta sayısını m değişkene ver

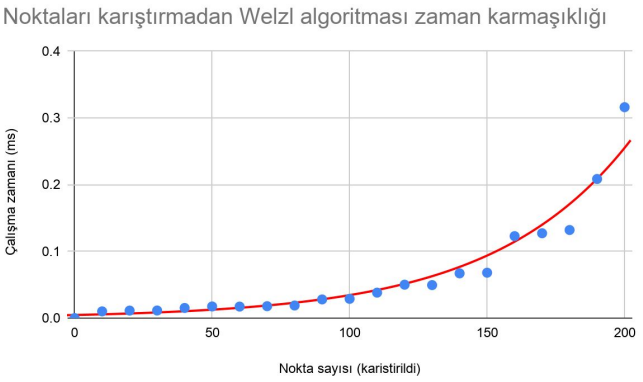
4. noktalar1'i karıştır
5. 3 elemanlı Nokta bosDizi'yi oluştur
6. Welzl fonksiyonunu Çağır ve mec değişkenine ata Welzl(noktalar1, m, boşDizi, 0)
6.1 mec cemberi tanımla
6.2 if m == 0 and n == 2:
mec = iki_noktali_cember(S[0], S[1])
elif n == 3:
mec = uc_noktali_cember(S[0], S[1], S[2])
elif m == 0:
mec = 0 yarıçaplı, S[0] merkezli bir çember
else
mec = Welzl(P, m - 1, S, n)
if P[m - 1], mec çemberin içinde değilse
S[n++] = P[m - 1];
mec = Welzl(P, m - 1, S, n);
6.3 return mec
7. noktalar1 ve noktalar2'deki tüm noktaları dönüştür
8. meci dönüştür
9. koordinat eksenlerini çiz
10.noktaları çiz
11.noktalar arasındaki çizgileri çiz
12.mec'i çiz
13.B-splinei_ciz fonksiyonunu çağır b_splinei_ciz(noktalar2, m, 3)
13.1 knot sayısını belirle ve lent değişkenine ata (lent = m + p)
13.2 t knot vektörü tanımla
13.3 t knot vektörünü doldur
13.4 p1'i girilen ilk nokta olarak tanımla
13.5 Eğride her nokta için, 0dan en büyük knot'a kadar j'yi tanımla
k = j'nin içinde bulunduğu knot aralığın indisi
p2 = deBoor(k, j, t, P, p -1)
ekranda p1'den p2'ye bir çizgi çiz
p1 = p2

IV. ZAMAN KARMAŞIKLIĞI HESAPLAMA

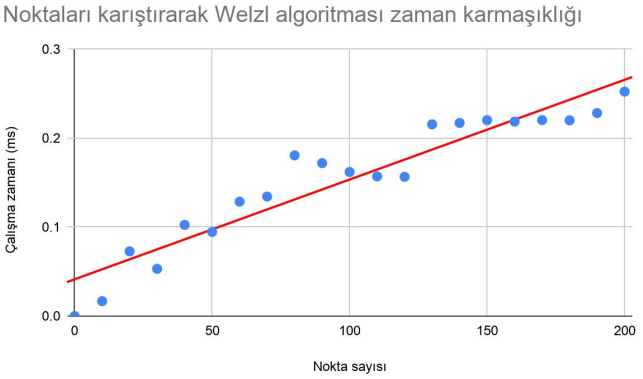
Programın zaman karmaşıklığını hesaplamak için, ilk önce içinde olan fonksiyonlarının zaman karmaşıklığını hesaplamak lazım.

A. *WELZL ZAMAN KARMAŞIKLIĞI*

Welzl algoritması genelde polynom zamanda çalışması gerek, ama noktaları işlemeden önce karıştırılırsa, ortalamada Lineer zamanda çalışır. Bunu kanıtlamak için, sıralanmış noktaları girdi olarak alarak, aynı noktaları karıştırmadan, ve karıştırarak aldım. Şekil 6 ve 7 bu test sonucunu veriyor.



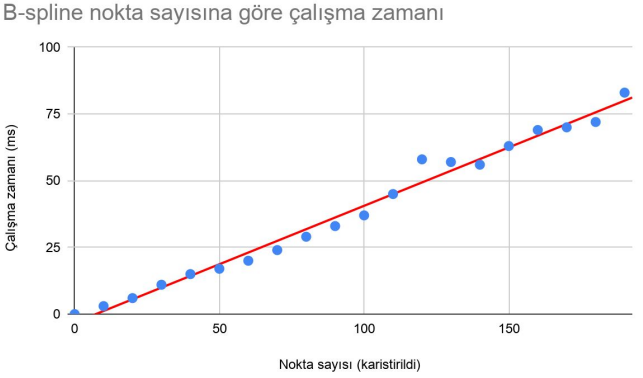
Şekil 6 noktaları karıştırmadan Welzl algoritması



Şekil 7 noktaları karıştırarak Welzl algoritması

Bu iki sonucundan kullanıldığı Welzl'in O(n) zaman karmaşıklığına sahip olduğunu anlaşılır.

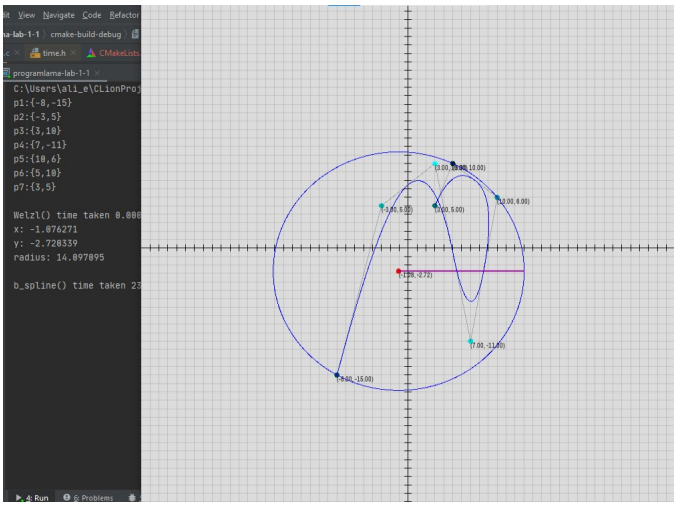
B. *B-SPLINE ZAMAN KARMAŞIKLIĞI*



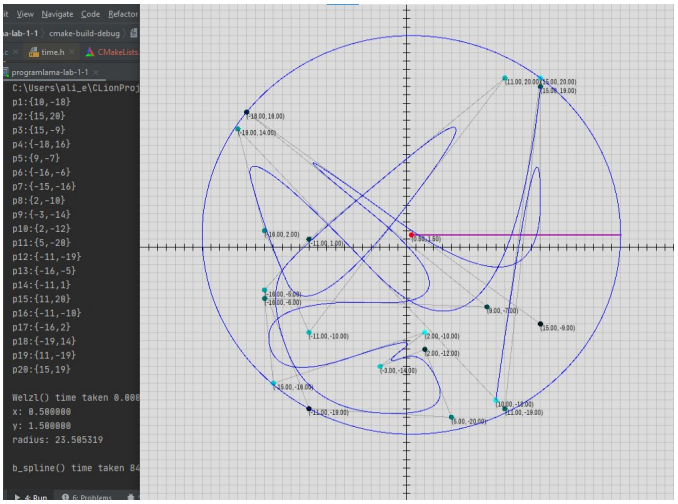
Şekil 8 B-spline nokta sayısına göre çalışma zamanı

Bu sonuçtan B-spline'in O(n) zaman karmaşıklığına sahip olduğunu anlaşılır. programda çalışan 2 algoritma O(n) olduğu için, programın da Big O zaman karmaşıklığı O(n) dir.

V. DENEYSEL SONUÇLAR



Şekil 9



Şekil 10

VI. Sonuç

Sonuç olarak, bu çalışmada başarılı bir şekilde dosyadan sayısını bilinmeyen noktaları okumak, o noktaları dinamik bir şekilde bellekte saklamak, o noktaları okumak, tüm noktaları içeren en küçük çemberini hesaplamak, ve noktaları girdiği sırasıyla tüm noktaların en yakınından geçen bir eğriyi çizmek gerçekleştirildi. Kullanılan algoritmaların zaman karmaşıklığını hesaplayıp, yazılan programın zaman karmaşıklığı O(n) olmasını kanıtlandı.

Kaynakça

[1] Welzl, Emo (1991), "Smallest enclosing disks (balls and ellipsoids)", in Maurer, H. (ed.), *New Results and New Trends in Computer Science*, Lecture Notes in Computer Science, **555**, Springer-Verlag, pp. 359–370, https://people.inf.ethz.ch/emo/PublFiles/SmallEnclDisk_LNCS555_91.pdf.

[2] C. de Boor (1971), "Subroutine package for calculating with B-splines", Techn.Rep. LA-4728-MS, Los Alamos Sci.Lab, Los Alamos NM; p. 109, 121