CONTOPT-JS: Sürekli Eniyileme Problemleri için Metasezgisel Algoritmalar tabanlı bir JavaScript Yazılım Kütüphanesi

CONTOPT-JS: Metaheuristic Algorithms based JavaScript Software Library for Continuous Optimization Problems

Osman Gokalp, Aybars Ugur, Sema Bodur
Bilgisayar Mühendisliği Bölümü, Ege Üniversitesi, İzmir, Türkiye
{osman.gokalp, aybars.ugur, sema.bodur}@ege.edu.tr

***Özetçe*—Bu çalışmada sürekli eniyileme problemlerinin çözümü için CONTOPT-JS isimli bir yazılım kütüphanesi tasarlanarak gerçekleştirimi yapılmıştır. JavaScript dili tabanlı bu kütüphane ile tamamen istemci taraflı çalışabilen web uygulamaları geliştirilebilmektedir. Kütüphanede Yapay Arı Kolonisi, Diferansiyel Gelişim, Parçacık Sürü Eniyilemesi ve Evrim Stratejileri metasezgiselleri yer almakla birlikte modüler bir yapıya sahip olduğundan dolayı yeni algoritma ve problemlerin de eklenebilmesine olanak tanımaktadır. CONTOPT-JS kütüphanesi ile, bazı standart eniyileme test fonksiyonları üzerinde ve ayrıca Sensör Yerleştirme problemi uygulama alanında deneysel çalışmalar yapılarak elde edilen sonuçlar sunulmuştur.**

Anahtar Kelimeler—sürekli eniyileme; metasezgisel algoritmalar; yazılım kütüphanesi, javascript, sensör yerleştirme problemi.

***Abstract*—In this study, a software library called CONTOPT-JS has been developed for solving continuous optimization problems. By using this JavaScript language based library, fully client-side web applications can be developed. In the library, Artificial Bee Colony, Differential Evolution, Particle Swarm Optimization and Evolution Strategies metaheuristics exist and new algorithms and new problems can be added because of its modular design. Using the CONTOPT-JS library, experimental works have been conducted on some standard optimization benchmark functions and Sensor Deployment application area and the obtained results have been presented.**

Keywords—continuous optimization; metaheuristic algorithms; software library, javascript, sensor deployment problem.

I. GİRİŞ

Gerçek hayattaki problemlerin birçoğu oluşacak olan maliyeti en aza indirgemek ya da elde edilecek karı en yüksek seviyeye getirmek üzerine kuruludur. Eniyileme problemleri olarak adlandırılan bu problemlerin zaman karmaşıkları genellikle NP-Tam veya NP-Zor seviyesinde olduğundan kesin çözümü bulmaya yönelik standart eniyileme yöntemleri yetersiz kalmaktadır. Bu noktada yaklaşım yöntemleri devreye girmekte ve verilen bir problem için en iyi sonucu bulmayı garanti etmese kısıtlı sürelerde en iyiye yakın, kabul edilebilir düzeyde çözümleri bulmayı sağlamaktadırlar.

Metasezgisel algoritmalar birer yaklaşım yöntemi olup zor eniyileme problemlerinin çözümünde başarı ile uygulanmaktadırlar. Bir metasezgisel algoritmanın en önemli özelliği kapalı kutu mantığıyla çalışması ve böylece problemden bağımsız olarak genel çözüm stratejileri sunmasıdır. Bu yönüyle aynı metasezgisel algoritma çok farklı problemlere uyarlanabilmektedir. Literatürde sıklıkla kullanılan metasezgisel algoritmalardan bazılarına örnek olarak Genetik Algoritmalar [1], Tabu Arama [2], Benzetimli Tavlama [3], Karınca Kolonisi Eniyilemesi [4], Parçacık Sürü Eniyilemesi [5] verilebilir. Metasezgisel algoritmalar ve sınıflandırılmaları hakkında daha ayrıntılı bilgi [6] içerisinde bulunmaktadır.

Eniyileme problemleri çözüm uzayının yapısı bakımından ayrık ve sürekli olmak üzere iki ana gruba ayrılmaktadır. Ayrık yapıya sahip problemlerde tam sayı değişkenleri bulunurken sürekli yapıya sahip problemlerde ise gerçel sayı değişkenleri yer almaktadır. Örneğin Gezgin Satıcı, Araç Rotalama ve Karesel Atama problemleri ayrık eniyileme sınıfına girmektedir ve belirlenen kısıtlar çerçevesinde en uygun kombinasyona sahip çözümün bulunması amaçlanmaktadır [7]. Sürekli eniyileme

Bu çalışma ASYU-2017 Konferansında sunulmuş ve sadece özet olarak Bildiri Özetleri Kitabı (ISBN: 978-605-84722-8-0)'nda yayımlanmıştır.

problemlerinde ise genel olarak n adet gerçel değer alan $f(x_1,x_2,\ldots,x_n)$ amaç fonksiyonunun en küçük veya en büyük değeri alması amaçlanmaktadır. Bu tip problemlerin gerçek dünyada birçok karşılığı bulunmaktadır [8], [9].

JavaScript (JS) web için tanımlanmış bir betik dili olup doğrudan HTML sayfaları içerisinde çalışabilmektedir. JS'in zengin yetenekleri sayesinde sunucu tarafında gerçekleştirilebilen birçok işlem istemci tarafta da yapılabilmektedir. Böylece sürekli sunucu bağlantısı gerektirmeyen ve sadece istemci tarafın kaynaklarından faydalanan internet uygulamaları geliştirilebilmektedir. Özellikle HTML sayfalarının arka planlarında iş parçacıkları çalıştırılabilme özelliğinin ("web worker") gelmesiyle birlikte yüksek işlem gücü gerektiren uygulamaların da önü açılmıştır.

Eniyileme problemlerinin çözümüne yönelik geliştirilen yazılımlar çok fazla işlem gücü gerektirdiğinden genellikle masaüstü uygulamaları olarak geliştirilmektedir. Bu da onları platform bağımlı hale getirmekte ve kullanımlarını sınırlamaktadır. HTML tabanlı uygulamalar ise platform bağımsız yapılarıyla herhangi bir web tarayıcı üzerinden erişilerek farklı işletim sistemleri veya farklı cihaz türlerinde (kişisel bilgisayar, tablet akıllı telefon vb.) kolaylıkla çalıştırılabilmektedir. Bu çalışmada sürekli eniyileme problemlerinin çözümüne yönelik istemci taraflı web uygulamalarında kullanılmak üzere metasezgisel algoritmalar tabanlı bir JS kod kütüphanesi geliştirilmiştir. Böylece web tabanlı sürekli eniyileme problemleri için geliştirilecek uygulamalar bu kütüphaneden faydalanabilecektir.

Bu çalışmanın ikinci bölümünde literatürde bulunan çeşitli metasezgisel algoritmalar tabanlı yazılım kütüphanelerinden örnekler verilmiştir. Üçüncü bölümde ise geliştirilen kütüphanenin tanıtımı yapılarak nasıl kullanılacağı ile ilgili bilgiler paylaşılmıştır. Geliştirilen kütüphane ile iki farklı örnek uygulama geliştirilmiş ve buna ait bilgiler dördüncü bölümde paylaşılmıştır. Beşinci ve son bölümde ise bu çalışma ile ilgili sonuç ve genel bir değerlendirme yer almaktadır.

II. BENZER ÇALIŞMALAR

Eniyileme problemlerini çözmeye yönelik çeşitli kod kütüphaneleri / yazılım çatıları bulunmaktadır. Bu kütüphaneler farklı platformlarda farklı algoritma destekleri sunabilmektedirler. Örneğin Java platformu için geliştirilmiş olan metasezgiseller veya evrimsel algoritmalar tabanlı bazı önemli kod kütüphaneleri şu şekildedir: Watchmaker [10] evrimsel ve genetik algoritmalar için Java platformunda geliştirilmiştir. Nesne tabanlı yapısıyla kolayca genişletilebilir ve özelleştirilebilirdir. Java dilinde geliştirilmiş olan bir diğer kütüphane de OPT4J [11]'dir. OPT4J Diferansiyel Gelişim, Parçacık Sürü Eniyilemesi ve Benzetimli Tavlama evrimsel algoritmalarına ek olarak bazı çok hedefli eniyileme algoritmaları da içermektedir. JAMES [12] Java uygulama çatısı ise daha çok ayrık eniyileme problemlerine yönelik olarak geliştirilmiş olup yerel arama metasezgisellerinden yararlanmaktadır.

Matlab ortamı için geliştirilmiş olan GEATbx [13] genetik ve evrimsel algoritmalar ile çeşitli türlerde (ayrık, sürekli) eniyileme problemleri, çözümlerini gerçekleştirir. Matlab platformu için geliştirilmiş bir diğer kütüphane olan MHTB'nin [14] eniyileme modülleri Java ve C/C++ dillerinde yazılmıştır. Kapsadığı algoritmaların bazıları, tek çözümlü, popülasyon tabanlı ve melez metasezgiseller olarak belirtilebilir.

Paradiseo [15] bir C++ nesne tabanlı uygulama çatısı olup metasezgisellerin yeniden kullanılabilirliği üzerine inşa edilmiştir. Tek çözümlü, çok hedefli ve popülasyon tabanlı metasezgiseller için ayrı paketleri olan Paradiseo oldukça kapsamlı bir içeriğe sahip olmasıyla öne çıkmaktadır. ANSI-C++ temelli bir evrimsel algoritmalar kütüphanesi olan EO [16] hem sürekli hem de ayrık eniyileme problemleri için özelleştirilebilir çok sayıda metasezgisel algoritma içermektedir.

C# dili temelli bir evrimsel algoritma çerçevesi olan HeuristicLab [17], Genetik Programlama, Makine Öğrenmesi algoritmaları, Veri Analizi, Yerel Arama, Benzetilmiş Tavlama, Tabu Arama, Parçacık Sürü Eniyilemesi, Evrim Stratejileri gibi birçok sezgisel ve metasezgisel algoritmaları içerisinde barındırmaktadır.

JavaScript dili için ise ayrık eniyileme problemlerinin çözümü için birkaç kod kütüphanesi bulunsa da [18], [19] bunlar yeterince kapsamlı değildirler. Yapılan araştırma sonucunda sürekli eniyileme problemlerine yönelik metasezgisel algoritma tabanlı kapsamlı bir kütüphaneye ise rastlanmamıştır.

III. YAZILIM PAKETİ TASARIMI

A. *Gerçekleştirimi Yapılan Metasezgisel Algoritmalar*

1) *Yapay Arı Kolonisi Algoritması*

Yapay Arı Kolonisi (YAK) algoritması [20] bir sürü zekası yöntemi olup arıların yiyecek arama davranışlarından esinlenmektedir. ABC algoritması (Şekil 1) sırasıyla işçi, gözlemci ve izci olmak üzere üç ana aşamadan oluşmaktadır. YAK algoritmasında çözümler birer nektar olarak temsil edilmekte ve nektar miktarı da çözümün kalitesini belirlemektedir. Buna göre amaç en yüksek değere sahip nektarı, bir başka deyişle evrensel en iyi çözümü bulmaktır.

```
İlkleme
While sonlanma koşulu sağlanmadı do
  İşçi arılar aşaması
  Olasılıkların hesaplanması
  Gözlemci arılar aşaması
  İzci arılar aşaması
  En iyi çözümü sakla
End while
```

Şekil 1. Yapay Arı Kolonisi Algoritması sözde kodu

Bu çalışma ASYU-2017 Konferansında sunulmuş ve sadece özet olarak
Bildiri Özetleri Kitabı (ISBN: 978-605-84722-8-0)'nda yayımlanmıştır.

YAK algoritmasının ilkleme aşamasında verilen problemin arama sınırları kapsamında rastgele çözümler oluşturulmaktadır. İşçi arılar aşamasında, her bir çözümün rastgele seçilen bir boyutu yine rastgele seçilen bir komşu çözümün aynı sıradaki boyutu ile çaprazlamaktadır. Eğer elde edilen yeni çözümün kalitesi öncekinden iyiyse, yeni çözüm olarak eskisinin yerini almaktadır. Bir sonraki aşamada eldeki çözümlere kaliteleri ile doğru orantılı olarak birer seçilme olasılığı atanmaktadır. Gözlemci arılar aşaması algoritmanın faydalanma evresi olup nektar sayısı kadar bir döngü boyunca kaliteli çözümlere ağırlık verilmek üzere yeni çözümler üretilmektedir. Burada bir önceki aşamada belirlenmiş olan olasılık değerleri dikkate alınmakta ve yüksek olasılığa sahip çözümlerin seçilme olasılıkları daha büyük olmaktadır. Son aşama olan izci arılar aşaması algoritmanın keşif evresini oluşturmaktadır. Buna göre belirli bir limit değeri sayısı kadar iyileştirilememiş olan çözümler atılarak yerine rastgele oluşturulmuş yeni çözümler dahil edilmektedir. Son aşamada ise o andaki en iyi çözüm bulunarak eğer eski çözümden daha iyiyse onun yerine konulmaktadır.

2) Parçacık Sürü Eniyilemesi Algoritması

Parçacık Sürü Eniyilemesi (PSE) algoritması[5] sürü zekası yöntemlerinin bir üyesi olup kuş ve balık sürülerinin davranışlarından esinlenmektedir. Her biri bir adet çözümü temsil eden parçacıkların hız ve konum vektörü olmak üzere temelde iki bileşeni bulunmaktadır. Konum vektörü, çözümün kendisini gösterirken hız vektörü, parçacığın o anki konumuna eklenerek bir sonraki durumunun belirlenmesinde görev almaktadır.

```
İlkleme
While sonlanma koşulu sağlanmadı do
  For her bir parçacık do
    For her bir boyut do
      Parçacığın hız vektörünü güncelle
      Parçacığın konumunu güncelle
    End for
    Parçacığın kendi en iyi konumunu güncelle
    Sürünün en iyi konumunu güncelle
  End for
End while
```

Şekil 2. Parçacık Sürü Eniyilemesi Algoritması sözde kodu

PSE'ne ait sözde kod Şekil 2'deki gibidir. İlkleme aşamasında sürü boyutu kadar parçacık rastgele hız ve konum vektörlerine sahip olacak şekilde yaratılır. Ardından sonlanma koşuluna kadar parçacıkların hız ve konum vektörleri güncellenerek yeni çözümler elde edilir. Hız vektörlerinin güncellenmesinde parçacığın o ana kadarki kendi en iyi konumu ile sürünün o ana kadarki en iyi konumu belirleyici olmaktadır.

3) Diferansiyel Gelişim Algoritması

Diferansiyel Gelişim (DG) [21] algoritması popülasyon tabanlı bir evrimsel algoritmadır. İterasyonlar boyunca popülasyonda yer alan çözümlerin farklarından yeni çözümler oluşturarak evrensel en iyiye yaklaşmaya çalışmaktadır.

```
İlkleme
While sonlanma koşulu sağlanmadı do
  For her bir çözüm do
    Rastgele çözümler seç
    Seçilen çözümlerin farkından yeni bir çözüm oluştur
    For her bir boyut do
      Çaprazlama olasılığına göre eski boyutu güncelle
    End for
    If yeni çözüm eskisinden iyiyse then
      Yeni çözümü eskisinin yerine koy
    End if
  End for
  En iyi çözümü güncelle
End while
```

Şekil 3. Diferansiyel Gelişim Algoritması sözde kodu

Şekil 3'te sözde kodu verilen DG algoritmasının ilkleme aşamasında popülasyon boyutu kadar rastgele çözüm oluşturulmaktadır. Ardından, sonlanma koşulu sağlanana kadar her bir birey için popülasyondan rastgele komşu çözümler seçilmekte ve bunların birbirlerine göre farklarından yeni bir aday çözüm elde edilmektedir. Bu şekilde elde edilen aday çözümlerin hangi boyutlarının yeni çözüme aktarılacağına (çaprazlama işlemi) daha önceden belirlenmiş bir olasılığa göre karar verilmektedir. Buna göre çaprazlama oranı ne kadar yüksekse eski çözüm o oranda değişmektedir. Her yeni çözümün oluşturulmasının ardından eski çözümün uygunluk değeri ile karşılaştırılmakta ve daha iyi bir çözüme ulaşılmışsa eskisinin yerine popülasyona dahil edilmektedir. Yukarıda açıklanan işlemlerin popülasyondaki tüm bireylere uygulanmasının ardından, eğer evrensel en iyi çözümden daha iyi bir çözüme ulaşılmışsa, bu çözüm yeni en iyi çözüm olarak kaydedilmektedir.

4) Evrim Stratejileri Algoritması

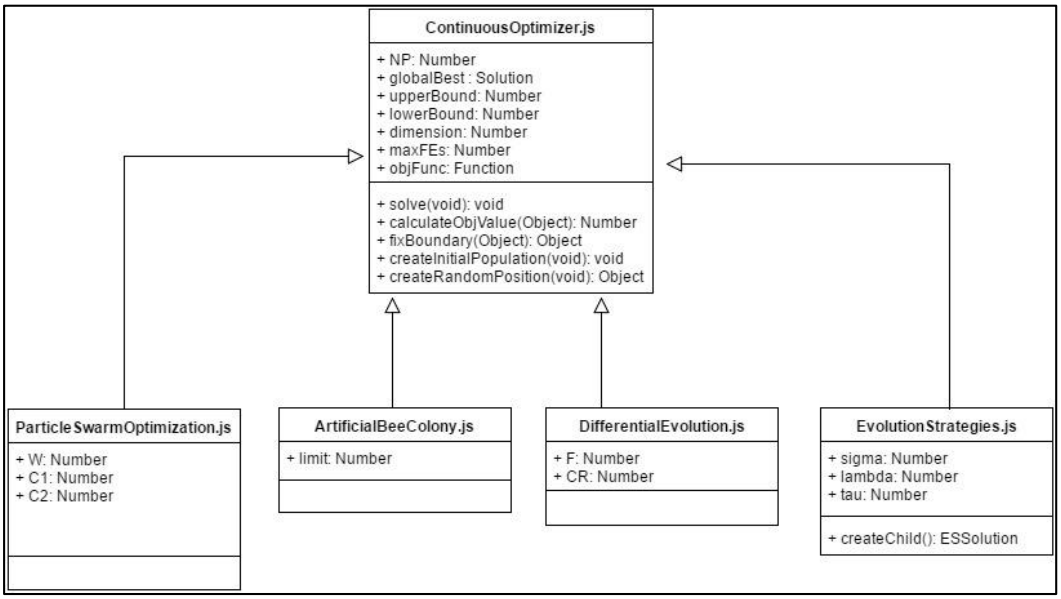
Evrim Stratejileri (ES) algoritması [22] evrimsel algoritmalar ailesinin bir üyesi olup kendinden uyarlanabilir bir özelliğe sahiptir. ES algoritmaları (μ,λ) veya (μ+λ) şeklinde tanımlanmaktadır. Burada μ ebeveyn sayısını, λ ise çocuk sayısını ifade etmektedir. Tanımda "+" bulunması ebeveyn düğümün yeni nesil popülasyon seçimine dahil edilmesi anlamına gelmekte olup "," bulunması seçimde sadece çocuk düğümlerin bulunacağı anlamına gelmektedir. Bu çalışmada (1+λ) stratejisi kullanılmış olup tek bir ebeveynden λ adet çocuk çözüm üretilip oluşan yeni popülasyondan (eski ebeveyn dahil) en iyisi yeni ebeveyn olarak seçilmektedir.

Bu çalışma ASYU-2017 Konferansında sunulmuş ve sadece özet olarak
Bildiri Özetleri Kitabı (ISBN: 978-605-84722-8-0)'nda yayımlanmıştır.

```
İlkleme
While sonlanma koşulu sağlanmadı do
  For her bir çocuk çözüm do
    Yeni mutasyon adım büyüklüğü belirle
    For her bir boyut do
      Adım büyüklüğüne göre rastgele mutasyon belirle
      Boyutun eski değerine mutasyona göre güncelle
    End for
  End for
  En iyi çözümü yeni ebeveyn olarak belirle
End while
```

Şekil 4. Evrim Stratejileri Algoritması sözde kodu

Şekil 4'te gösterilen (1+λ) ES algoritmasının ilkleme aşamasında bir adet rastgele çözüm rastgele mutasyon adım büyüklüğü ile birlikte oluşturulmaktadır. Yeni bir çocuk çözümün oluşturulmasındaki ilk aşama yeni bir mutasyon adım büyüklüğünün belirlenmesidir. Adım büyüklüğü çocuk düğümün mutasyon miktarını belirleyen ana parametre olup uyarlanabilir şekilde nesilden nesle aktarılmaktadır. Belirlenen adım büyüklüğü ile normal dağılıma göre üretilen rastgele mutasyon miktarı ebeveyn çözümün her bir boyutuna eklenmektedir. Bu işlemin sonucunda ortaya yeni bir çocuk düğüm çıkmaktadır. Tüm çocuk düğümler üretildikten sonra popülasyonun en iyisi



Şekil 5. CONTOPT-JS metasezgisel algoritmalarının kalıtım hiyerarşisine ait UML şeması

yeni ebeveyn olarak belirlenmektedir. Bundan sonraki nesiller yeni seçilen çözümün mutasyon adım büyüklüğünü kullanacağından algoritma kendinden uyarlanabilir hale gelmektedir.

B. Gerçekleştirim Ayrıntıları

Bu çalışmada geliştirilmiş olan CONTOPT-JS kütüphanesi tamamen JavaScript/HTML dilinde yazılmış olup herhangi bir üçüncü parti kütüphane kullanılmamıştır. CONTOPT-JS kütüphanesinin kapsamı bir dizi gerçel parametrenin eniyilenmesini gerektiren sürekli eniyileme problemlerini çözmektir.

Bu amaçla *"ContinuousOptimizer.js"* adlı bir üst sınıf tanımlanarak bir kalıtım hiyerarşisi oluşturulmuştur (Şekil 5). Bu sınıfta yer alan ve tüm alt sınıflarda ortak bir şekilde yer alan değişken ve fonksiyonlar aşağıda listelenmektedir:

- *"NP"*: Popülasyon büyüklüğü.
- *"globalBest"*: Evrensel en iyi çözüm.
- *"upperBound"*: Üst arama sınırı.
- *"lowerBound"*: Alt arama sınırı.
- *"dimension"*: Problem boyutu (eniyilenecek parametre sayısı).
- *"maxFEs"*: Amaç fonksiyonu çalıştırma sayısı sınırı (sonlandırma koşulu).
- *"objFunc"*: Amaç fonksiyonu.
- *"solve()"*: Eniyileme işlemini başlatan fonksiyon.
- *"calculateObjValue()"*: Amaç fonksiyonunu çalıştırıp bulduğu sonucu döndüren fonksiyon.
- *"fixBoundary()"*: Verilen bir çözüm vektörünü problemin tanımlamış olduğu arama sınırlarına geri çeken fonksiyon.
- *"createInitialPopulation()"*: Başlangıç popülasyonunu oluşturan fonksiyon.

Bu çalışma ASYU-2017 Konferansında sunulmuş ve sadece özet olarak
Bildiri Özetleri Kitabı (ISBN: 978-605-84722-8-0)'nda yayımlanmıştır.

- *"createRandomPosition()"*: Rastgele yeni bir çözüm vektörü üreten fonksiyon.

"ContinuousOptimizer.js" sınıfından türeyen sınıflar şu şekildedir:

- "ArtificialBeeColony.js": Yapay Arı Kolonisi algoritması.
- "DifferentialEvolution.js": Diferansiyel Gelişim algoritması.
- "ParticleSwarmOptimization.js": Parçacık Sürü Eniyilemesi algoritması.
- "EvolutionStrategies.js": Evrim Stratejileri algoritması ((1+λ) stratejisi).

JS dilinde sınıf tanımlama özelliği ECMAScript (ES) standardının [23] 6. sürümü ile birlikte gelmiştir. Ancak bu sürüm henüz web tarayıcıların çoğu tarafından desteklenmediğinden bu çalışmada ES 5 sürümü tercih edilmiştir. ES 5 sürümünde doğrudan sınıf tanımı ve kalıtım hiyerarşisi olmadığından çeşitli yöntemlerle bu yapı taklit edilmektedir. JS ES 5 sürümü için bir üst ve alt sınıf arasındaki kalıtım hiyerarşisi Şekil 6'daki gibi kurulabilmektedir.

```
function ÜstSınıf(param){
  this.değişken1=param
};
function AltSınıf(param1, param2) {
  ÜstSınıf.call(this, param1);
  this.değişken2=param2;
};
AltSınıf.prototype = Object.create(ÜstSınıf.prototype);
AltSınıf.prototype.constructor = ÜstSınıf;
```

Şekil 6. JavaScript'te kalıtım kod örneği

C. Kütüphanenin Kullanılması

CONTOPT-JS kütüphanesine ait kaynak kodlara http://yzgrafik.ege.edu.tr/~ugur/contopt_js adresinden erişilebilmektedir. Daha önceden tanımlanmış sürekli eniyileme test problemleri *"BenchmarkFunctions.js"* dosyası içerisinde yer almaktadır. Bu dosyaya isteğe göre kullanıcı tarafından yeni fonksiyonlar eklenebilmektedir.

Kütüphanede yer alan eniyileme algoritmaları "web worker" desteği ile çalışacak şekilde tasarlanmıştır. Bu sayede eniyileme görevi arka planda çalışan bir iş parçacığına verilmektedir. Seçilen eniyileme algoritmasını çalıştırabilmek için onun öncelikle bir "Worker" nesnesi oluşturulmalı ve belirlenen parametreler "postMessage()" fonksiyonu yoluyla kendisine iletilmelidir (Şekil 7). Burada hangi problem çözülmek isteniyorsa *"objFunc"* değişkenine ismi atanmalıdır (örneğin objFunc="sphere").

```
var w = new Worker("ArtificialBeeColony.js");
w.postMessage([NP, limit, maxFEs, objFunc, upperBound, lowerBound, dim]);
```

Şekil 7. "Worker" nesnesi oluşturulması ve çalıştırılması kod örneği

"postMessage()" çağrımının ardından iş parçacığı çalışacak ve verilen problemi çözmeye başlayacaktır. İş parçacığı tarafından üretilen sonuçlar *"onmessage()"* fonksiyonu ile alınabilmektedir (Şekil 8). Burada *"event"* parametresi ile gelen değerlerden ilki o anki amaç fonksiyonu hesaplama sayısını, ikincisi o ana kadar bulunmuş en iyi amaç fonksiyonu değerini ve üçüncüsü de en iyi çözümü oluşturan pozisyon vektörünü içermektedir.

```
w.onmessage = function (event) {
  fonksiyonHesaplamaSayısı = event.data[0];
  enİyiDeğer = event.data[1];
  enİyiÇözüm = event.data[2];
};
```

Şekil 8. "Worker" nesnesinden gelen sonuç bilgilerinin alınması

IV. ÖRNEK UYGULAMALAR

Bu bölümde, geliştirilen yazılım kütüphanesi kullanılarak gerçekleştirilen iki adet örnek uygulamadan bahsedilmektedir. Her iki örneğe de http://yzgrafik.ege.edu.tr/~ugur/contopt_js adresinden erişilebilmektedir.

A. Test Fonksiyonlarının Çalıştırılması

Kütüphanenin kullanımına yönelik ilk örnek uygulama test fonksiyonlarının eniyilenmesi için yapılmıştır. Bu kapsamda 30 boyutlu *"Sphere"* fonksiyonu üzerinde bir deney gerçekleştirilmiştir. Bu deneyde her bir algoritma varsayılan parametre değerleri (Tablo II) ile 20'şer kez çalıştırılmış ve elde edilen amaç fonksiyonu değerlerinin sırasıyla ortalaması, standart sapması, en iyisi ve en kötüsü bulunarak Tablo I'de sunulmuştur. *"Sphere"* fonksiyonunun en iyi amaç fonksiyonu değeri 0'dır ve tüm algoritmaların bu değere oldukça yaklaştıkları görülmektedir. Metasezgisel algoritmaların performansları, kullanılan parametrelere oldukça bağımlıdır. Bu nedenle, herhangi bir parametre ayarlama yöntemi kullanılması durumunda bu sonuçlar daha da iyileştirilebilecektir.

	YAK	**PSE**	**DG**	**ES**
Ortalama	1,60E-33	4,99E-3	4,47E-09	9,74E-89
Standart Sapma	4,29E-33	2,08E-2	1,51E-8	4,35E-88
En iyi	2,36E-36	5,38E-31	2,06E-11	8,15E-122
En kötü	1,69E-32	9.34E-2	6,79E-8	1,95E-87

Tablo I. 30 boyutlu "Sphere" fonksiyonu üzerinde YAK, PSE, DG ve ES algoritmalarının performansı.

YAK	NP:20, limit:100
PSE	NP:25, W:0,39, C1:2,55, C2:1,33
DG	NP:20, CR:0,9, F:0,8
ES	sigma:1, lambda:10, tau: 0,18

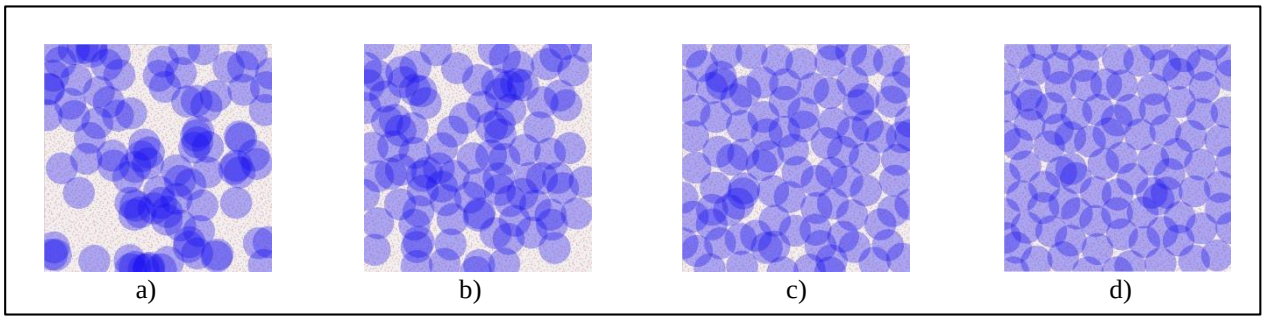
Tablo II. YAK, PSE, DG ve ES algoritmaları için varsayılan parametre değerleri.

Bu çalışma ASYU-2017 Konferansında sunulmuş ve sadece özet olarak
Bildiri Özetleri Kitabı (ISBN: 978-605-84722-8-0)'nda yayımlanmıştır.

B. *Kablosuz Sensör Yerleştirme Probleminin Çözülmesi*

Kütüphanemiz ile ikinci örnek uygulama Sensör Yerleştirme problemi için geliştirilmiştir. Bu problemde amaç, 2 boyutlu Öklid uzayında verilen bir alan içerisine rastgele dağıtılmış olan özdeş dinamik sensörlerin, bu alanı olabildiğince fazla kapsayacak şekilde yeniden konumlandırılmasıdır. Bir sensör probleminin tanımlanabilmesi için öncelikle sensörlerin sayısı, kapsama yarıçapı ve kapsanacak alanının sınırları belirlenmelidir.

Sensör Yerleştirme probleminin çözülebilmesi için kütüphanede daha önceden tanımlanmış eniyileme algoritmalarından kalıtım yoluyla yenileri türetilmiştir. Türetilen bu sınıflar öncekinden farklı olarak sensör yarıçapı, kapsanacak alan ölçüleri, sensör sayısı gibi bilgileri de içermektedir. Ayrıca *"calculateObjValue()"*, *"createInitialPopulation()"*, *"fixBoundary()"* vb. fonksiyonlar yeniden tanımlanarak özelleştirilmiştir.



Şekil 9. YAK algoritması ile örnek bir sensör eniyilemesi problemi çözümü. **a)**başlangıç durumu, kapsama oranı: %78,98 **b)**1000 fonksiyon hesaplaması sonrası, kapsama oranı: %85,52, **c)** 5000 fonksiyon hesaplaması sonrası, kapsama oranı: %94,94, **d)** 15000 fonksiyon hesaplaması sonrası, kapsama oranı: %98,32

Sensör yerleştirme probleminin sürekli eniyileme problemi biçimine dönüştürülmesi için ise Tablo III'te yer alan yaklaşım kullanılmıştır. Buna göre her bir sensörün 2 boyutlu Öklid düzlemindeki x ve y koordinatları yan yana konularak birleştirilmekte ve eniyilenecek olan çözüm vektörü oluşturulmaktadır. Böylece vektörün boyutu (2 x sensör sayısı) şeklinde olmaktadır. Metasezgisel algoritmalar kapalı kutu mantığında çalıştığından kendisine verilen herhangi bir gerçel sayı vektörünü belirtilen amaç fonksiyonuna göre eniyilemektedir. Bu sayede sensör yerleştirme problemi çözülmüş olmaktadır.

Sensör 1	**Sensör 2**	**…**	**Sensör n**
x1,y1	x2,y2	…	xn,yn

Tablo III. Bir sensör problemi çözümünün temsili.

Örnek uygulama olarak yapılan sensör yerleştirme probleminde kapsanacak alan 500x500 birim^2 ölçülerinde belirlenmiştir. 100 adet sensör 35 birim yarıçapında kapsama alanına sahip olacak şekilde eklenmiştir. Kapsama oranının hesaplanabilmesi için 5000 adet nokta alana olabildiğince eşit yoğunlukta dağıtılmıştır. Dağıtılan noktalardan yüzde kaçının sensörlerin kapsama alanına girdiği hesaplanarak kapsama oranı olarak bulunmuştur. YAK algoritması kullanılarak böyle bir problemin çözümüne ait sonuçlar sırasıyla Şekil 9'da yer almaktadır.

Sensör yerleştirme uygulamasına yönelik bir de deneysel çalışma gerçekleştirilmiştir. Kapsanacak alanın genişliği, sensör kapsama yarıçapı ve kapsanacak nokta sayısı yukarıdaki örnek uygulamayla aynı olacak şekilde belirlenmiştir. Bundan farklı olarak, sensör sayıları 90, 100, 110 ve algoritmaların harcayabileceği maksimum fonksiyon hesaplama sayıları ise 5.000 ve 50.000 şeklinde denenmiştir.

YAK ve PSE algoritmaları için sensör yerleştirme problemine uygun olacak parametreler çeşitli kombinasyonlar denenerek belirlenmiştir. Buna göre YAK için kullanılan parametreler sırasıyla *NP*: 20 ve *limit*: 100 şeklinde olup PSE algoritması için *NP*: 25, *W*: 0,39, *C1*: 2,55 ve *C2*: 1,33 şeklindedir.

Her bir algoritma ilgili problem konfigürasyonu için deneyler 20'şer kez tekrarlanmış ve elde edilen kapsama oranlarının ortalamaları Tablo IV'te sunulmuştur.

	YAK		**PSE**	
	mf=5.000	**mf=50.000**	**mf=5.000**	**mf=50.000**
n=90	% 92,44	% 99,10	% 91,95	% 96,05
n=100	% 94,85	% 99,67	% 94,33	% 98,33
n=110	% 96,44	% 99,98	% 95,99	% 99,26

Tablo IV. YAK ve PSE algoritmalarının farklı sensor sayısı (n) ve maksimum fonksiyon hesaplama sayıları (mf) ile çalıştırılması sonucu elde edilen ortalama kapsama oranları yüzdesi

Tablo IV'te yer alan deneysel sonuçlara göre kütüphanemizde kodladığımız algoritmaların farklı hesaplama bütçelerinde dahi iyi sonuçlar verdiği gözlemlenmiş ve kanıtlanmıştır. YAK algoritmasının performansının Sensör Yerleştirme probleminde daha yüksek olduğu da görülmektedir.

V. SONUÇ VE DEĞERLENDİRME

Bu çalışmada JavaScript dili ile geliştirilmiş, açık kaynak kodlu ve sürekli eniyileme problemlerinin çözümüne yönelik, metasezgisel algoritmalar tabanlı bir

Bu çalışma ASYU-2017 Konferansında sunulmuş ve sadece özet olarak
Bildiri Özetleri Kitabı (ISBN: 978-605-84722-8-0)'nda yayımlanmıştır.

kütüphane olan CONTOPT-JS tanıtılmıştır. Literatürde başka dil ve platformlarda birçok eniyileme kütüphanesi olsa da, sürekli eniyileme problemlerine yönelik JS tabanlı kapsamlı bir kütüphane bulunmamaktadır. Başka bir hazır yazılım kullanılmadan tamamen JS/HTML ile yazılan bu kütüphane ile tamamen istemci taraflı web uygulamaları geliştirilebilmektedir. Bu sayede farklı işletim sistemi ve platformlarda çalışabilecek eniyileme uygulamaları geliştirilebilecektir.

Eniyileme problemlerinin çözümü genellikle yüksek işlem gücü gerektirmektedir. Böyle bir durum web uygulamalarının çalışmasını olumsuz etkilediğinden bu çalışmada gerçekleştirimi yapılan algoritmaların "web worker" yeteneği ile uyumlu olarak çalışabilmesi sağlanmıştır. HTML5 ile gelen "web worker" özelliği arka planda bir iş parçacığı açarak kendisine verilen hesaplamaları yapabilmekte ve sayfanın performansının düşmesini engellemektedir.

CONTOPT-JS'nin bazı standart eniyileme test fonksiyonları ve Sensör Yerleştirme problemi üzerinde olmak üzere iki farklı örnek uygulaması geliştirilmiştir. Elde edilen sonuçlar eniyileme algoritmalarının gerçekleştirimlerinin başarısını ortaya koymaktadır.

Kütüphanede şu anda Yapay Arı Kolonisi, Diferansiyel Gelişim, Parçacık Sürü Eniyilemesi ve Evrim Stratejileri metasezgiselleri bulunmaktadır ve ilerleyen çalışmalarda algoritmaların sayısının artırılması hedeflenmektedir.

YAZAR KATKILARI

Birinci yazar çalışmanın sorumlu yazarı olup yazılım kütüphanesi tasarımı ve eniyileme algoritmalarının kodlanmasında görev almıştır. *İkinci yazar* proje ekibinin oluşturulması ve yönetimi, algoritmaların belirlenmesi ve çalışma kalitesinin iyileştirilmesi yönündeki faaliyetlerde bulunmuştur. *Üçüncü yazar* ise Sensör Yerleştirme problemi örnek uygulamasında ve ilgili deneysel çalışmanın yapılmasında görev almıştır.

TEŞEKKÜR

Bu çalışma Ege Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü tarafından desteklenmektedir (Proje numarası: 16-MÜH-072). Yazar Osman Gökalp TÜBİTAK BİDEB 2211-A Genel Yurt İçi Doktora Burs Programı kapsamında desteklenmektedir.

KAYNAKÇA

[1] J. H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, c. viii. Oxford, England: U Michigan Press, 1975.

[2] F. Glover, "Tabu Search—Part I", ORSA J. Comput., c. 1, sayı 3, ss. 190–206, Ağustos 1989.

[3] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, ve others, "Optimization by simulated annealing", science, c. 220, sayı 4598, ss. 671–680, 1983.

[4] M. Dorigo, "Optimization, Learning and Natural Algorithms", Ph.D. Thesis, Politecnico di Milano, Italy, 1992.

[5] J. Kennedy ve R. Eberhart, "Particle swarm optimization", sunulan IEEE International Conference on Neural Networks, 1995, ss. 1942–1948.

[6] I. Boussaïd, J. Lepagnot, ve P. Siarry, "A survey on optimization metaheuristics", Inf. Sci., c. 237, ss. 82–117, Tem. 2013.

[7] A. Uğur, "Path planning on a cuboid using genetic algorithms", Inf. Sci., c. 178, sayı 16, ss. 3275–3287, Ağustos 2008.

[8] Eberhart ve Y. Shi, "Particle swarm optimization: developments, applications and resources", içinde Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), 2001, c. 1, ss. 81–86 c. 1.

[9] S. Das ve P. N. Suganthan, "Problem definitions and evaluation criteria for CEC 2011 competition on testing evolutionary algorithms on real world optimization problems", Jadavpur Univ. Nanyang Technol. Univ. Kolkata, 2010.

[10] D. W. Dyer, "The Watchmaker Framework for Evolutionary Computation". [Çevrimiçi]. Available at: http://watchmaker.uncommons.org/index.php. [Erişim: 28-Haz-2017].

[11] M. Lukasiewycz, M. Gla\s s, F. Reimann, ve J. Teich, "Opt4J: a modular framework for meta-heuristic optimization", içinde Proceedings of the 13th annual conference on Genetic and evolutionary computation, 2011, ss. 1723–1730.

[12] H. De Beukelaer, G. F. Davenport, G. De Meyer, ve V. Fack, "JAMES: An object-oriented Java framework for discrete optimization using local search metaheuristics", Softw. Pract. Exp., c. 47, sayı 6, ss. 921–938, 2017.

[13] H. Pohlheim, "GEATbx - Genetic and Evolutionary Algorithms Toolbox in Matlab". [Çevrimiçi]. Available at: http://www.geatbx.com/. [Erişim: 28-Haz-2017].

[14] "MetaHeuristics ToolBox for MATLAB". [Çevrimiçi]. Available at: http://neo.lcc.uma.es/software/mhtb/. [Erişim: 28-Haz-2017].

[15] S. Cahon, N. Melab, ve E.-G. Talbi, "Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics", J. Heuristics, c. 10, sayı 3, ss. 357–380, 2004.

[16] M. Keijzer, J. Merelo, G. Romero, ve M. Schoenauer, "Evolving objects: A general purpose evolutionary computation library", içinde Artificial evolution, 2002, ss. 829–888.

[17] S. Wagner ve M. Affenzeller, "Heuristiclab: A generic and extensible optimization environment", Adapt. Nat. Comput. Algorithms, ss. 538–541, 2005.

[18] "js-metaheuristics: Metaheuristic algorithms for JavaScript". [Çevrimiçi]. Available at: https://github.com/aureooms/js-metaheuristics. [Erişim: 28-Haz-2017].

[19] "genetic-js: Advanced genetic and evolutionary algorithm library written in Javascript". [Çevrimiçi]. Available at: https://github.com/subprotocol/genetic-js. [Erişim: 28-Haz-2017].

[20] D. Karaboga, "An idea based on honey bee swarm for numerical optimization", Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[21] R. Storn ve K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces", J. Glob. Optim., c. 11, sayı 4, ss. 341–359, 1997.

[22] I. Rechenberg, Evolutionsstrategie;: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart-Bad Cannstatt: Frommann-Holzboog, 1973.

[23] "ECMAScript 6: New Features: Overview and Comparison". [Çevrimiçi]. Available at: http://es6-features.org/#Constants. [Erişim: 28-Haz-2017].

Bu çalışma ASYU-2017 Konferansında sunulmuş ve sadece özet olarak Bildiri Özetleri Kitabı (ISBN: 978-605-84722-8-0)'nda yayımlanmıştır.