Makale Raporu

Hazırlayan: Muhammed Kayra Bulut
Ders: İleri Algoritma Analizi ve Tasarımı
Ders Yürütücüsü: Dr. Hafiza İrem Türkmen Çilingir

Nisan 2024

1 Seyahat Satıcısı Problemi (TSP)

Seyahat Satıcısı Problemi (TSP) veya İngilizce adıyla "Traveling Salesman Problem" (TSP), operasyonel araştırma ve teorik bilgisayar bilimlerinde en çok incelenen optimizasyon problemlerinden biridir. Bu bölümde, problemi tanımlayıp, önemini ve uygulama alanlarını ele alacağız.

1.1 Seyahat Satıcısı Problemi Nedir?

Seyahat Satıcısı Problemi (TSP), optimizasyon ve bilgisayar bilimlerinde derinlemesine incelenmiş, hem teorik açıdan hem de pratik uygulamalar açısından büyük önem taşıyan klasik bir problemdir[3]. Bu problem, basit bir soruyla başlar: Bir satıcı, her biri farklı bir şehri temsil eden, ziyaret etmesi gereken bir dizi nokta verildiğinde, en düşük toplam maliyetle bu şehirlerin hepsini ziyaret edip başladığı noktaya nasıl döner? Bu, satıcının her şehri yalnızca bir kez ziyaret edeceği ve en sonunda başlangıç noktasına geri döneceği bir turun bulunması gerektiği anlamına gelir. Problemin çözümü, ideal olarak, satıcının kat etmesi gereken toplam mesafeyi veya maliyeti minimize eden bir rotayı belirler.

1.2 Problemin Kökeni ve Tarihi

Seyahat Satıcısı Problemi, 1930'larda matematikçiler tarafından formüle edilmiş olup, o zamandan bu yana bilgisayar bilimi, operasyonel araştırma ve ekonomi gibi çok çeşitli alanlarda yoğun bir ilgi görmüştür. Problemin popülaritesi, basit bir şekilde ifade edilebilmesine rağmen, genellikle karmaşık ve zorlu çözümler gerektirmesinden kaynaklanmaktadır. TSP, özellikle büyük şehir sayıları söz konusu olduğunda, bilgisayarların bile kolayca çözemediği bir problem olma özelliğini korumaktadır.

1.3 Önemi ve Uygulama Alanları

TSP, birçok gerçek dünya probleminin modellemesinde kullanılır. Bu problemler arasında lojistik planlama, araç rotalama, mikroçip üretimi ve hatta DNA

dizilimleme gibi alanlar bulunmaktadır. TSP'nin çözümü, maliyeti, zamanı ve kaynak kullanımını optimize etme potansiyeline sahiptir, bu da onu pek çok endüstri ve araştırma alanı için değerli kılar. Kullanıldığı bazı alanlar şunlardır;

- **Lojistik ve Dağıtım Planlaması** Lojistik sektöründe, malların depolardan son kullanıcıya etkin bir şekilde dağıtılması, TSP ile doğrudan ilişkilendirilebilecek bir problemdir. Şirketler, ürünleri müşterilere mümkün olan en kısa sürede ve en düşük maliyetle ulaştırmayı hedeflerken, TSP'nin prensipleri, rota optimizasyonu ve araç yükleme stratejilerinin temelini oluşturur. Bu, özellikle çok noktalı teslimat rotaları ve zaman kısıtlamaları gibi ek zorlukların olduğu durumlarda önemlidir.

- **Araç Rotalama Problemleri** Araç rotalama, özellikle toplu taşıma ve kurye hizmetleri gibi sektörlerde kritik bir öneme sahiptir. TSP, bir aracın birden fazla noktayı ziyaret etmesi gereken durumlarda, en etkili rotayı belirlemede kullanılır. Bu, hem yakıt tüketiminin azaltılmasına hem de toplam teslimat süresinin kısaltılmasına olanak tanır, böylece işletmelerin operasyonel verimliliği artırılır.

- **Mikroçip Üretimi** Mikroçip üretimi sürecinde, birçok ince detayın hassas bir şekilde yerleştirilmesi gerekmektedir. Burada, devre elemanlarının birbirine en kısa yollarla bağlanması, enerji tüketiminin ve üretim maliyetlerinin azaltılmasında önemli bir faktördür. TSP, bu elemanların optimal şekilde düzenlenmesini sağlayarak, mikroçiplerin daha etkili ve verimli bir şekilde tasarlanmasına yardımcı olur.

- **DNA Dizilimi ve Biyoenformatik** Biyoenformatik alanında, DNA dizilimleme ve protein katlanması gibi konular, büyük veri kümelerinin analiz edilmesini gerektirir. TSP ve benzeri optimizasyon problemleri, bu verilerin analiz edilmesi ve anlamlı biyolojik bilgilerin çıkarılması süreçlerinde kullanılır. Örneğin, bir DNA zincirindeki genler arasındaki en etkili bağlantı yollarını bulmak, hastalıkların anlaşılması ve tedavisi için kritik öneme sahip olabilir.

1.4 Matematiksel Formülasyon

Matematiksel olarak, TSP bir graf teorisi problemi olarak formüle edilir. Bir graf $G = (V, E)$, şehirleri temsil eden V düğümleri ve bu şehirler arasındaki yolları temsil eden E kenarları ile tanımlanır. Her bir kenarın bir maliyeti (uzunluğu, zamanı veya diğer bir maliyet ölçüsü) vardır. Problemin amacı, başlangıç düğümüne dönen bir Hamilton çevresi bulmak ve bu çevrenin toplam maliyetini minimize etmektir.

1.5 Problemin Zorlukları

TSP'nin çözümü, özellikle büyük ölçekli problemler için, birkaç önemli zorluğu beraberinde getirir:

- Kombinatorik Patlama: Şehirlerin sayısı arttıkça, olası tüm rotaların sayısı faktöriyel olarak artar. Örneğin, 10 şehir için 3,628,800 farklı rota vardır, ancak 20 şehir için bu sayı yaklaşık 2.43×10^{18}'e yükselir. Bu, tüm olası çözümlerin pratik bir sürede hesaplanamayacağı anlamına gelir.
- NP-Zorluk: TSP, NP-zor problemler sınıfına aittir. Bu, problem için bilinen hiçbir algoritmanın polinom zamanında en iyi çözümü bulamayacağı anlamına gelir (eğer $P \neq NP$ ise). Bu durum, özellikle büyük ölçekli problemler için, etkili bir çözüm bulmayı zorlaştırır.
- Yakınsama Problemleri: Sezgisel ve yaklaşık çözüm yöntemleri, genellikle en iyi çözüme "yakınsarlar" ancak bunun garantisi yoktur. Bu yöntemler, yerel optimumlara takılıp kalabilir ve global optimumu bulamayabilir.
- Çözümün Doğrulanması: Bir çözüm bulunduğunda, özellikle sezgisel yöntemlerle, bu çözümün gerçekten en iyi olup olmadığının doğrulanması zor olabilir. En iyi çözümün elde edildiğinden emin olmak için ek kontroller ve karşılaştırmalar gerekebilir.

1.6 Çözüm Yöntemleri

TSP'nin tam çözümü genellikle brute-force yaklaşımlarla, yani olası tüm çözümlerin denendiği yöntemlerle bulunabilir, ancak bu yaklaşımın hesaplama maliyeti şehirlerin sayısı arttıkça faktöriyel olarak artar. Bu nedenle, büyük problemler için çeşitli yaklaşık ve sezgisel çözüm yöntemleri geliştirilmiştir. Bunlar arasında genetik algoritmalar, karınca kolonisi optimizasyonu ve tabu arama gibi yöntemler bulunur.

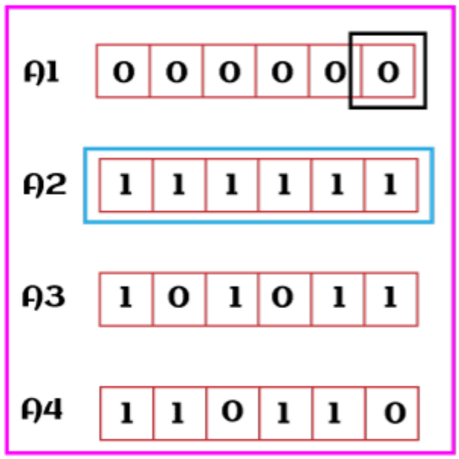
2 Genetik Algoritma

Genetik algoritma, evrimsel algoritmalar sınıfına ait bir global optimizasyon yöntemidir ve genellikle karmaşık problemlerin çözümünde kullanılır. Bu algoritma, doğal seleksiyon ve genetik mekanizmaların prensiplerinden esinlenerek tasarlanmıştır. Charles Darwin'in evrim teorisine dayanan bu algoritma, en uygun çözümleri "evrimleştirmek" için mutasyon, çaprazlama ve seçilim gibi işlemleri kullanır. Genetik algoritma, çözüm uzayındaki en iyi çözümleri bulma amacıyla, potansiyel çözümleri bir popülasyon olarak ele alır ve bu popülasyon üzerinde iteratif olarak evrimsel işlemler uygular.

2.1 Genetik Algoritmanın Temel Bileşenleri

Temel bileşenleri şu şekilde sıralanabilir;

- **Gen:** Resim 1'de siyah renkli alanla bir örneği gösterilmiştir. Genetik algoritmalar içerisinde, bir çözümün temel yapı taşıdır ve genellikle problemin bir özelliğini veya parametresini temsil eder. Biyolojideki genler gibi, bir genetik algoritma geni de bir karakteristik veya değer taşır. Örneğin,



Resim 1: Genetik Algoritma Bileşenleri

bir yolculuk planlama problemi için, bir gen belirli bir şehri temsil edebilir; bir tasarım optimizasyonu problemi için ise bir gen, tasarımın bir özelliğini (örneğin, bir parçanın boyutu) ifade edebilir. Genlerin bir araya gelmesiyle oluşturulan dizilim, bir çözümün veya bireyin "genotip"ini oluşturur. Genetik algoritmalarda genler, problemi çözmek için araştırılacak parametrelerin veya değişkenlerin kodlanması için kullanılır.

- **Kromozom:** Resim 1'de pembe mavi alanla bir önreği gösterilmiştir. Bir bireyin genetik yapısını oluşturan gen dizisidir. Genetik algoritmada, bir kromozom, problemin tam bir çözümünü temsil eder. Biyolojide olduğu gibi, genetik algoritmada da bir kromozom, bir dizi gen içerir ve bu genlerin her biri çözümün farklı bir yönünü kodlar. Kromozomlar, algoritmanın çalışması sırasında değerlendirilen, seçilen, çaprazlanan ve mutasyona uğrayan temel birimlerdir. Kromozomların topluluğu bir "popülasyon" oluşturur ve genetik algoritmanın her iterasyonu, bu popülasyon üzerinde gerçekleştirilen genetik işlemlerle (seçilim, çaprazlama, mutasyon) yeni nesillerin oluşturulmasını içerir.

- **Popülasyon:** Resim 1'de pembe renkli alanla bir önreği gösterilmiştir. Potansiyel çözümlerin bir koleksiyonudur. Her bir birey, problem alanındaki bir çözümü temsil eder.

- **Uygunluk Fonksiyonu:** Bir bireyin (çözümün) problemi ne kadar iyi çözdüğünü değerlendiren bir fonksiyondur. Bu fonksiyon, bireylerin evrimsel süreçte nasıl seçileceğini belirler.

- **Seçilim:** En uygun bireylerin bir sonraki nesile aktarılması için seçildiği süreçtir. En uygun bireylerin, genetik bilgilerini gelecek nesillere aktarma olasılığı daha yüksektir.

- **Çaprazlama (Genetik Rekombinasyon):** İki bireyin genetik bilgisinin birleştirilerek yeni bireyler (çocuklar) oluşturulması işlemidir. Bu süreç, çözüm uzayının geniş bir alanının araştırılmasını sağlar.
- **Mutasyon:** Bir bireyin genetik yapısında rastgele değişiklikler yapılması işlemidir. Bu, çözüm uzayının farklı bölgelerinin keşfedilmesine yardımcı olur ve yerel optimumlarda takılıp kalmanın önüne geçer.

2.2 Genetik Algoritmanın Uygulama Alanları

Genetik algoritma, çok çeşitli optimizasyon problemlerinde kullanılmaktadır. Bunların başında **Seyahat Satıcısı Problemi** gelir. Ayrıca bu alanlar arasında mühendislik tasarımı, yapay zeka, ekonomi, moleküler biyoloji, ve hatta sanat eserlerinin yaratılması gibi disiplinler bulunmaktadır. Özellikle, karmaşık ve çok boyutlu problemlerde, geleneksel optimizasyon yöntemlerinin başarısız olduğu veya çok zaman aldığı durumlarda genetik algoritmalardan faydalanılır.

2.3 Genetik Algoritmanın Avantajları ve Kısıtları

Avantajları:

- Genetik algoritma, çok geniş çözüm uzayları olan problemlerde etkilidir.
- Global optimuma yakın çözümler bulma potansiyeline sahiptir.
- Paralelleştirilebilir yapıdadır, bu sayede hesaplama süresi önemli ölçüde azaltılabilir.

Sınırlılıkları:

- Uygunluk fonksiyonunun ve parametrelerin (mutasyon oranı, popülasyon büyüklüğü vb.) doğru şekilde seçilmesi gerekir.
- Bazı problemlerde, genetik algoritma yerel optimumlarda takılıp kalabilir.
- Algoritmanın performansı, problem tipine göre değişiklik gösterebilir.

Genetik algoritma, problem çözme sürecinde doğanın evrimsel prensiplerinden ilham alarak, kompleks ve çok boyutlu problemlere etkili çözümler sunma potansiyeli taşır. Bu yöntemin uygulama alanlarının genişliği ve sağladığı avantajlar, onu günümüzde hala popüler ve araştırılmaya değer bir optimizasyon aracı yapmaktadır.

2.4 MTSP (Çoklu Seyahat Eden Satıcı Problemi)

MTSP (Çoklu Seyahat Eden Satıcı Problemi), klasik Seyahat Eden Satıcı Problemi'nin (TSP) genel bir versiyonudur. Burada, birden fazla satıcı (veya araç)

aynı anda çeşitli şehirleri ziyaret ederek, tüm şehirlerin en verimli şekilde ziyaret edilmesini sağlamak amacıyla en kısa toplam yolculuk mesafesini bulmaya çalışır[4]. MTSP, özellikle lojistik, araç rotalama ve hizmet alanı tasarımı gibi alanlarda uygulama bulur. Problem, her bir satıcının başlangıç noktasına dönmesi gereken kapalı bir tur şeklinde veya satıcıların farklı başlangıç ve bitiş noktaları olan açık turlar şeklinde modellenebilir. MTSP'nin çözümü, satıcıların sayısı ve ziyaret edilecek şehirlerin dağılımı göz önünde bulundurularak, genetik algoritmalar, sezgisel yöntemler ve tam sayılı programlama teknikleri gibi çeşitli yöntemlerle aranır.

2.5 OCMTSP (Açık-Kapalı Çoklu Seyahat Eden Satıcı Problemi)

OCMTSP (Açık-Kapalı Çoklu Seyahat Eden Satıcı Problemi), Çoklu Seyahat Eden Satıcı Problemi'nin (MTSP) daha spesifik bir versiyonudur. Bu varyasyonda, bazı satıcılar (veya araçlar) turu başladıkları noktada bitirmek zorunda değillerdir, yani açık turlar oluşturabilirler[4]. Diğerleri ise klasik TSP'deki gibi turu başladıkları noktada bitirirler; bu da kapalı turlara örnektir. OCMTSP, dinamik ve esnek rotalama çözümleri gerektiren durumlar için özellikle uygundur. Örneğin, bir lojistik şirketi farklı başlangıç ve bitiş noktalarına sahip araçlarla çeşitli teslimatları en maliyet-etkin şekilde nasıl planlayabilir? Bu tür problemler, özellikle birden fazla hedefe ulaşım ve zaman penceresi kısıtlamaları gibi ek zorluklar içerdiğinde, OCMTSP'nin çözüm yöntemleriyle ele alınır. Problemin çözümünde genetik algoritmalar, parçacık sürü optimizasyonu ve diğer evrimsel algoritmalar gibi yöntemler tercih edilebilir. Bu yöntemler, problemi farklı açılardan ele alarak, karmaşık rotalama senaryolarında etkili çözümler üretmeyi amaçlar.

2.6 Önerilen Genetik Algoritma'nın OCMTSP Kıyaslaması

OCMTSP Makalesinde[4], İyileştirilmiş Genetik Algoritma'nın OCMTSP (Açık Kapalı Çoklu Gezgin Satıcı Problemi) performansının analizi, çeşitli simetrik ve asimetrik TSP (Gezgin Satıcı Problemi) örnekleri üzerinden yapılan karşılaştırmalarla detaylandırılmış. Yapılan deneysel çalışmalar, İyileştirilmiş Genetik Algoritma'nın, özellikle CPU kullanımı açısından ve çözüm kalitesi bakımından mevcut algoritmaları geride bıraktığını gösteriyor. Bu algoritma, birden çok kromozom temelli bir yaklaşımla OCMTSP'yi çözmek için tasarlanmış ve Matlab 2023a'da gerçekleştirilmiş olan 50 bağımsız çalıştırma üzerinden elde edilen ortalama sonuçlara dayanıyor.

Hız açısından, İyileştirilmiş Genetik Algoritma'nın hesaplama zamanı son derece rekabetçi; TSPLIB'den elde edilen asimetrik TSP örneklerinin 17'sinde, bazı durumlarda en iyi bilinen çözümlere çok yakın sonuçlar elde edildiği görülüyor. Çözüm kalitesi bakımından da, önerilen algoritmanın diğer yöntemlerle karşılaştırıldığında oldukça iyi performans gösterdiği ve bunun istatistiksel metrikler kullanılarak teyit edildiği anlaşılıyor. Özellikle, algoritmanın her iki simetrik ve asimetrik

problemler için geçerli ve etkili olduğu vurgulanıyor. Bununla birlikte, araştırmanın sınırlılıkları varken, algoritma her zaman optimal çözümü garanti etmez.

2.6.1 Bazı Sayısal Sonuçlar

ftv33 Test Verisi

Tablo 1: OCMTSP için Çözüm Karşılaştırmaları (ftv33)

Metod	**LSA BKS**	**LKH-3**	**Önerilen GA**
En İyi Bilinen Çözüm	1,603	1,549	1,674

Tablo 1'de, önerilen GA'nın en iyi bilinen çözümden daha yüksek bir çözüm bulduğunu görüyoruz. Ancak bu, GA'nın her zaman optimal çözüm bulmadığını gösterse de, hesaplama süresi verilerini incelemeden tam bir karşılaştırma yapamayız.

ftv35 Test Verisi

Tablo 2: OCMTSP için Çözüm Karşılaştırmaları (ftv35)

Metod	**LSA BKS**	**LKH-3**	**Önerilen GA**
En İyi Bilinen Çözüm	1,829	2,112	1,796

Tablo 2, önerilen GA'nın LKH-3'e göre daha iyi bir çözüm bulduğunu ve en iyi bilinen çözüme oldukça yaklaştığını görebiliriz. Bu, önerilen GA'nın yüksek performansını gösteriyor.

Bu performans sonuçlarının yanında aynı zamanda hız verilerine bakacak olursak, önerilen GA'nın ortalama CPU çalışma süresi **6.12** saniye ile **6.92** saniye arasında değişiyor. Bu değer LSA için **6.8** iken LKH-3 için **6.85** saniyedir. Yani GA ortalamada daha hızlı bir çözüm yöntemidir.

Resim 2'ye baktığımızda ftv33 test verisinde yöntemlere göre alınan sonuçları görüyoruz. Burada harflerin anlamalrı şu şekilde;

- **Ş:**Şehir Sayısı
- **İS:**İç Satıcı Sayısı(Başladığı yere dönmek zorunda)
- **DS:**Dış Satıcı Sayısı(Geri dönmeye gerek yok)
- **HZ:**Hesaplama Zamanı (GA için)
- **SS:**Standart sapma (GA için)

Resim 2'deki grafikler, farklı çözüm metodolojilerinin veya algoritmalarının performanslarını TSP Lib (Gezgin Satıcı Problemi Kütüphanesi) veri setlerine göre karşılaştırıyor. Her bir grafikte üç sütun görülüyor: En İyi Bilinen Çözüm (LKH-3), Önerilen GA'nın (Genetik Algoritma) En Kötü ve En İyi çözümleri. Bu grafikler, çözüm stratejilerinin yol uzunluğu (Bulunan Yol Uzunluğu) açısından etkinliğini gösteriyor.

- İlk grafik (sol üstte), TSP Lib (ftv33) 34Ş-3İŞ-3DS-HZ6.92-SS0.0 veri seti için, önerilen GA'nın en kötü ve en iyi çözümleriyle LKH-3 algoritmasının bulduğu en iyi çözüm arasında çok az fark olduğunu gösteriyor. Bu, önerilen GA'nın bu özel veri seti için iyi performans gösterdiğini ve en iyi bilinen çözüme oldukça yakın sonuçlar elde ettiğini gösteriyor.
- İkinci grafik (sağ üstte), TSP Lib (ftv33) 34Ş-2İŞ-4DS-HZ6.87-SS1.77 veri seti için benzer bir performans gösteriyor ancak burada önerilen GA'nın en iyi çözümü LKH-3'ün çözümüne göre biraz daha uzun bir yol bulmuş gibi görünüyor.
- Üçüncü grafik (sol altta), TSP Lib (ftv33) 34Ş-3İŞ-3DS-HZ6.71-SS3.92'de, önerilen GA'nın en iyi çözümünün LKH-3'ün bulduğu çözüme göre ufak bir miktar daha kötü olduğunu gösteriyor.
- Dördüncü grafik (sağ altta), TSP Lib (ftv33) 34Ş-3İŞ-2DS-HZ6.8-SS5.33'de, önerilen GA'nın en iyi çözümü yine LKH-3'ün çözümüne yakınken, en kötü çözümün bir miktar daha kötü olduğunu gösteriyor.

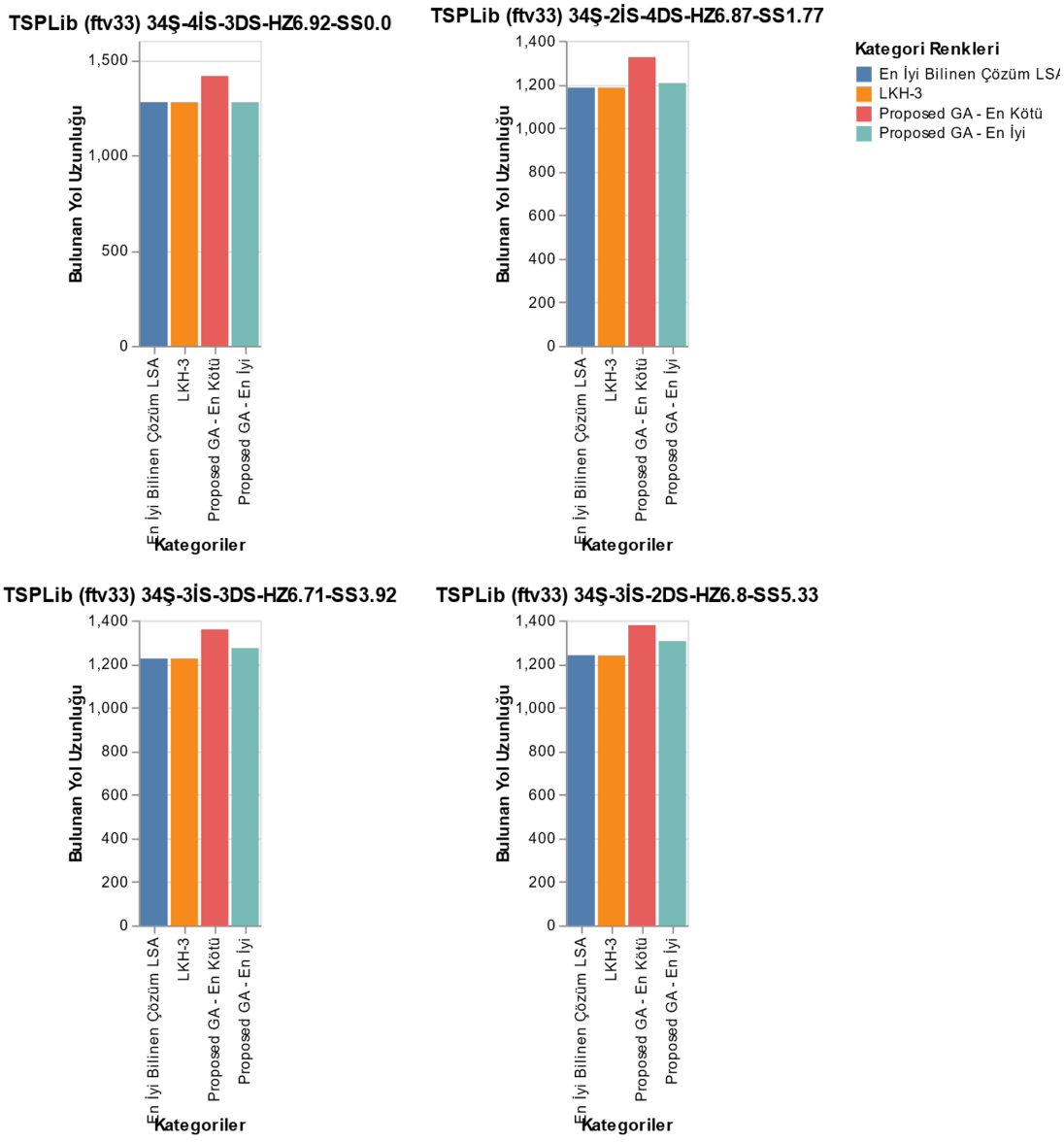
Genel olarak, önerilen Genetik Algoritmanın TSP problemlerinin çeşitli varyasyonları üzerinde tutarlı ve rekabetçi çözümler ürettiğini söyleyebiliriz. Ancak, GA'nın en kötü ve en iyi çözümleri arasındaki fark, oldukça düşüktür. Bu da algoritmanın tutarlı sonuçlar elde ettiğini gösterir. Aynı zamanda hız açısından en iyi algoritmanın GA olduğunu da gözardı etmemek gerekir.

3 Ayrık Yılan Optimizasyonu Algoritması (Discrete Snake Optimization Algorithm)[5]

Ayrık Yılan Optimizasyonu Algoritması (Discrete Snake Optimization Algorithm), doğadan ilham alan ve ayrık optimizasyon problemleri için tasarlanmış bir arama ve optimizasyon yöntemidir[5]. Bu algoritma, özellikle yılanların avlarını bulma ve çevrelerindeki alanı keşfetme stratejilerinden esinlenerek geliştirilmiştir. Algoritmanın temelinde, bir yılanın hareket etme biçimini taklit eden bir arama mekanizması yatar. Yılanın ilerleme, dönme ve kıvrılma hareketleri, problemin çözüm uzayında etkili bir şekilde arama yapılmasını sağlar.

Algoritma, potansiyel çözümleri yılanın vücudu olarak modelleyen ve bu çözümleri iteratif olarak geliştiren bir yaklaşım benimser. Her bir iterasyon sırasında, yılanın başı (yani çözüm uzayında bir nokta), en iyi sonuca doğru "ilerler", ve bu ilerleme esnasında, yılanın vücudunun geri kalan kısmı da uyum sağlayarak takip eder. Bu süreç, yılanın avını takip etmesine benzer şekilde, global optimuma doğru ilerlemeyi amaçlar.

Ayrık Yılan Optimizasyonu Algoritması, özellikle rotalama, zamanlama ve ağ tasarımı gibi ayrık optimizasyon problemlerinde kullanılır. Bu algoritmanın avantajlarından biri, karmaşık ve geniş çözüm uzaylarında, yerel optimumlara takılmadan, global optimuma yakın çözümlere ulaşabilme yeteneğidir. Ayrıca,



Resim 2: OCMTSP Yöntem Kıyaslaması [4]

algoritmanın esnek yapısı, farklı türdeki optimizasyon problemlerine kolaylıkla uyarlanabilmesini sağlar.

Yılanın doğal davranışlarından esinlenilerek geliştirilen bu algoritma, optimizasyon alanında yenilikçi bir yaklaşım sunar. Yılanın çevresini nasıl algıladığı, nasıl hareket ettiği ve avına nasıl yöneldiği gibi davranışlar, algoritmanın çözüm arama stratejisine temel teşkil eder. Bu, özellikle dinamik ve sürekli değişen problemlerin çözümünde, algoritmanın uyarlanabilirliğini ve etkinliğini artırır.

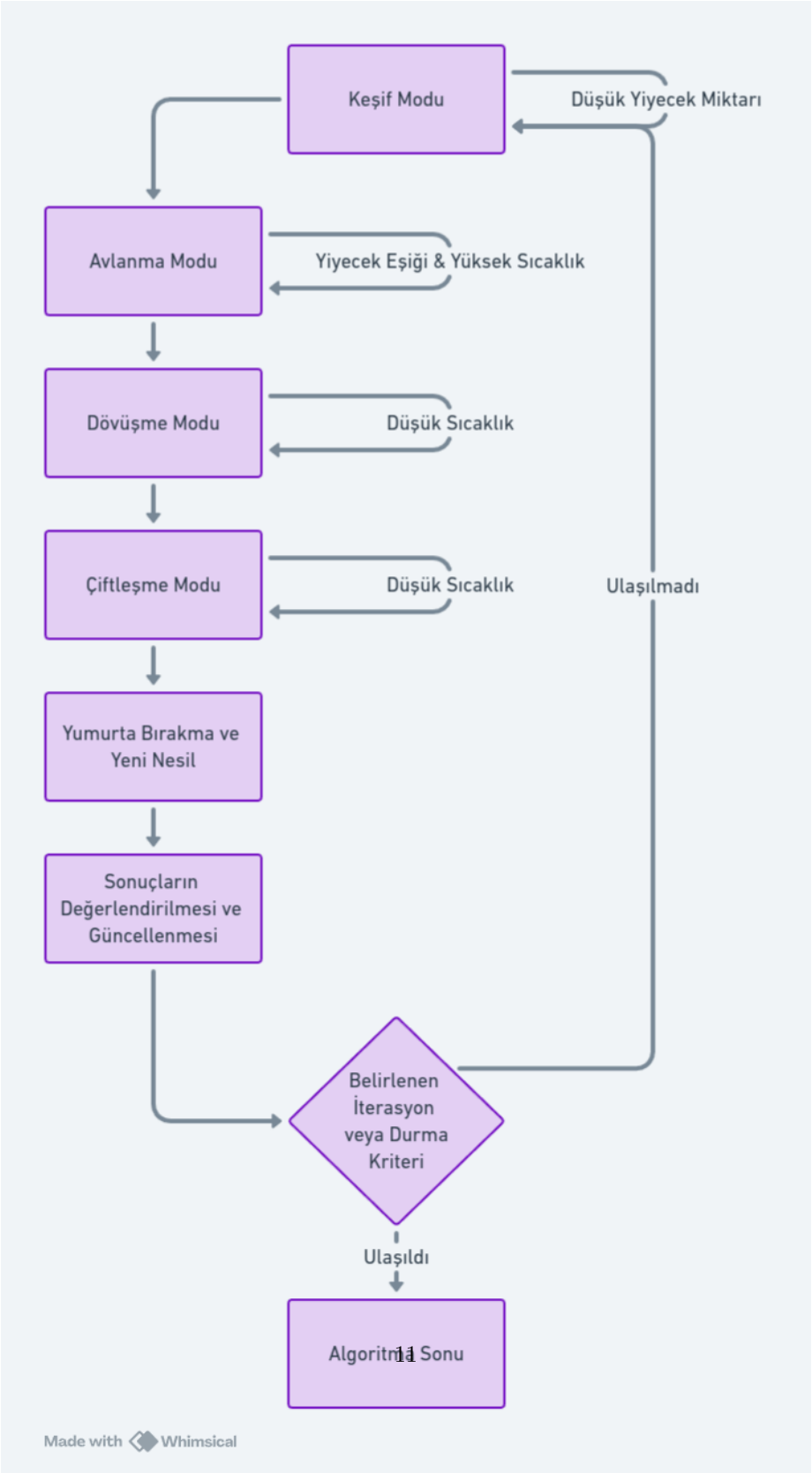
Resim **??**'da gösterilen akış diyagramı, belirli bir proses veya algoritmanın farklı modları arasında geçişi ve her modun ne zaman tetikleneceğini gösteriyor. Diyagramda yer alan adımları aşağıdaki gibi yorumlayabiliriz:

- **Keşif Modu:** Bu başlangıç noktası gibi görünüyor ve düşük yiyecek miktarı durumunda aktif oluyor.
- **Avlanma Modu:** Eğer yiyecek eşiği yüksekse ve sıcaklık yüksekse, avlanma moduna geçiliyor. Bu mod, yiyecek bulma ve enerji toplama eylemlerini temsil ediyor olabilir.
- **Dövüşme Modu:** Sıcaklık düşükse dövüşme moduna geçiliyor. Bu mod, bölgeyi savunma veya rekabet eden başka bir birey ile etkileşimde bulunmayı temsil edebilir.
- **Çiftleşme Modu:** Yine düşük sıcaklıkta çiftleşme moduna geçiş yapılıyor. Bu, bireylerin üremek için bir araya geldiği bir durumu ifade ediyor olabilir.
- **Yumurta Bırakma ve Yeni Nesil:** Çiftleşme sonrası doğal olarak yumurta bırakma ve yeni neslin ortaya çıkışını temsil ediyor.
- **Sonuçların Değerlendirilmesi ve Güncellenmesi:** Yumurta bırakma ve yeni nesil sonrasında alınan sonuçlar değerlendiriliyor ve proses güncelleniyor.
- **Belirlenen İterasyon veya Durma Kriteri:** Bu, algoritmanın belirlenen bir iterasyon sayısına ulaşıp ulaşmadığını veya belirlenen başka bir durma kriterine ulaşıp ulaşmadığını kontrol ediyor.
- **Algoritma Sonu:** Durma kriterine ulaşıldığında, algoritmanın sona erdiğini belirtiyor.

Her mod, Ayrık Yılan Optimizasyonu Algoritmasının farklı senaryolara nasıl tepki verdiğini ve çevresel şartlara göre hangi eylemleri gerçekleştireceğini temsil ediyor.

3.1 Ayrık Yılan Optimizasyonu Algoritmasının Özellikleri

Ayrık Yılan Optimizasyonu Algoritmasının (DSOA) benzersiz özellikleri, onu diğer optimizasyon yöntemlerinden ayıran ve özellikle ayrık problemlerin çözümünde öne çıkan bir yaklaşım haline getiren faktörlerdir. Bu algoritmanın temel özellikleri şunlardır:



Resim 3: Ayrık Yılan Optimizasyonu Algoritması Akışı [5]

- **Doğadan İlham Alma:** Yılanların avlanma ve hareket etme stratejilerinden esinlenen DSOA, doğal dünyanın karmaşık problemleri çözme konusundaki ustalığını taklit eder. Bu, algoritmanın, çözüm arama sürecinde doğal davranışları matematiksel işlemlere dönüştürme yeteneğine dayanır.
- **Esnek ve Uyarlanabilir Yapı:** Algoritma, çeşitli ayrık optimizasyon problemlerine kolayca uyarlanabilir bir yapı sunar. Farklı problemlerin özel gereksinimlerine göre algoritmanın parametreleri ayarlanabilir, bu da onu geniş bir uygulama alanına sahip kılar.
- **Yerel Optimumlardan Kaçınma:** Yılanın çevresel algılama kabiliyetini modelleyen algoritma, yerel optimumlara takılma riskini azaltır ve global optimuma ulaşma şansını artırır. Bu, özellikle geniş ve karmaşık çözüm uzaylarına sahip problemlerde önemlidir.
- **Yüksek Keşif Kapasitesi:** DSOA, çözüm uzayının geniş alanlarını keşfetme yeteneğine sahiptir. Yılanın vücudunun esnek hareketleri, algoritmanın çözüm uzayının farklı bölgelerini etkili bir şekilde tarayabilmesini sağlar.
- **Kolay Uygulanabilirlik:** Algoritma, basit ve anlaşılır bir yapıya sahiptir, bu da onun farklı problemlere ve farklı programlama dillerine kolayca uygulanabilmesini sağlar. Bu, algoritmanın popülerliğini ve erişilebilirliğini artırır.
- **Paralelleştirilebilir:** DSOA'nın işlemleri, paralel hesaplama teknikleriyle kolayca uygulanabilir. Bu, büyük ölçekli problemlerin daha hızlı çözülmesine olanak tanır ve algoritmanın verimliliğini önemli ölçüde artırır.

Bu özellikler, Ayrık Yılan Optimizasyonu Algoritmasını, özellikle zamanlama, ağ tasarımı ve rotalama gibi ayrık optimizasyon problemlerinde güçlü ve esnek bir çözüm aracı haline getirir. Algoritmanın doğadan ilham alması, problem çözme yaklaşımlarında yenilikçi ve orijinal çözümler sunar.

3.2 Ayrık Yılan Optimizasyonu Algoritması Performans Değerlendirmesi

Resim 4'e baktığımızda;

- İlk grafikte, 'eil51' veri seti için:
  - ACO ve PSO, nispeten yüksek maliyetli çözümler üretmişken,
  - DSOA, GA ve Opt algoritmaları daha düşük maliyetli çözümler üretmişlerdir.
  - Özellikle, DSOA'nın ürettiği çözümün maliyeti, diğer tüm algoritmaların ürettiği çözüm maliyetlerinden düşüktür, bu da DSOA'nın bu problem için üstün performansını göstermektedir.
- İkinci grafikte, 'berlin52' veri seti için:

  - DSOA, diğer tüm algoritmalara göre en düşük maliyetli çözümlerden birini bulmuştur, bu da onun bu veri seti için güzel bir algoritma olduğunu göstermektedir.
  - GA algoritmasıysa en kötü performansı göstermiştir.
  - ACO ve WOA'nın maliyetleri nispeten daha düşüktür.

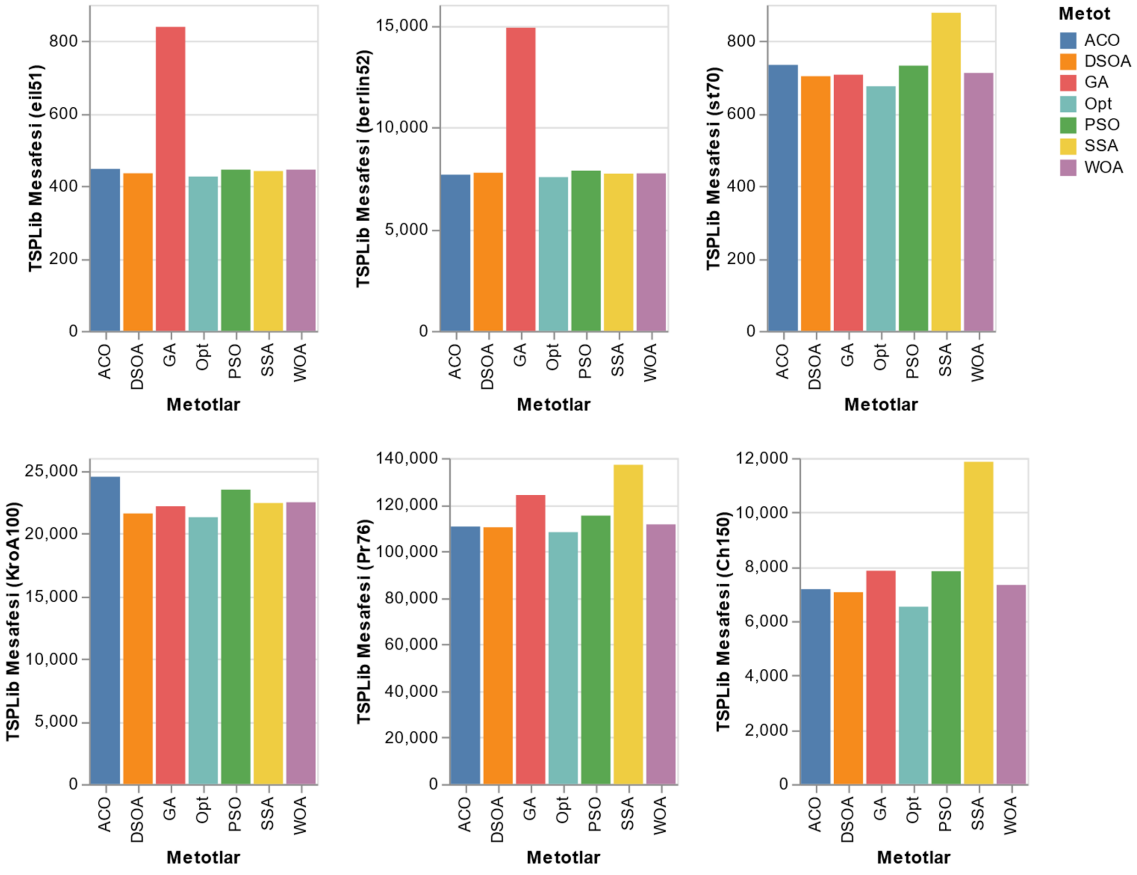
- Üçüncü grafikte, 'st70' veri seti için:
  - DSOA , en düşük maliyetli çözümü üretmiştir.
  - GA ve PSO orta seviyede performans gösterirken,
  - ACO ve SSA'nın çözümleri daha yüksek maliyetli çıkmıştır.

- Dördüncü grafikte, 'kroA100' veri seti için:
  - DSOA yine en iyi performansı sergileyerek en düşük maliyetli çözümü bulmuştur.
  - GA ve SSA, orta düzeyde maliyetli çözümler üretmişler.
  - ACO ve PSO ise bu veri seti için en yüksek maliyetli çözümleri üretmişlerdir.

- Beşinci grafikte, 'pr76' veri seti için:
  - DSOA algoritması, en düşük maliyetli çözümle dikkat çekiyor.
  - GA'nın ve SSA'nın performansı bu veri setinde düşmüş, diğer algoritmalara kıyasla daha yüksek maliyetli çözümler üretmişlerdir.
  - SSA ve GA en yüksek maliyetli çözümlere sahip algoritmalar olarak belirmiştir.

- Altıncı grafikte, 'ch150' veri seti için:
  - DSOA bu veri setinde de en düşük maliyetli çözümü bulmuş, algoritmanın bu veri setlerinde tutarlı bir şekilde iyi performans gösterdiği görülmüştür.
  - WOA ve ACO algoritmaları, DSOA'dan sonra en iyi performansı göstermişlerdir.
  - SSA ve GA, en yüksek maliyetli çözümleri üretmişlerdir.

Genel sonuçlara bakacak olursak, Ayrık Yılan Optimizasyonu Algoritması(DSOA) iyi ve tutarlı bir performans göstermiştir.



Resim 4: Ayrık Yılan Optimizasyonu Algoritması Kıyaslaması [5]

4 PSO Algoritması[1]

Particle Swarm Optimization (PSO) algoritması, kuş sürülerinin veya balık sürülerinin doğal hareketlerinden esinlenerek geliştirilmiş bir meta-sezgisel arama ve optimizasyon yöntemidir. PSO, bir çözüm uzayı içerisinde hareket eden ve her biri bir çözümü temsil eden bir grup partikül (tanecik) üzerinden çalışır. Bu partiküller, kendi deneyimlerinin yanı sıra komşu partiküllerin deneyimlerinden de öğrenirler, böylece bireysel ve toplu bilgi birikimini kullanarak optimal çözüme doğru ilerlerler.

Algoritmanın temel mekanizması, her partikülün hem kendi kişisel en iyi konumuna (pbest) hem de tüm sürünün şu ana kadar bulduğu en iyi konuma (gbest) doğru bir hızla hareket etmesidir. Bu dinamik, partiküllerin hem kendi deneyimlerinden hem de sürünün kolektif bilgisinden yararlanmasını sağlayarak, çözüm uzayında etkili bir arama performansı sergilemelerine olanak tanır.

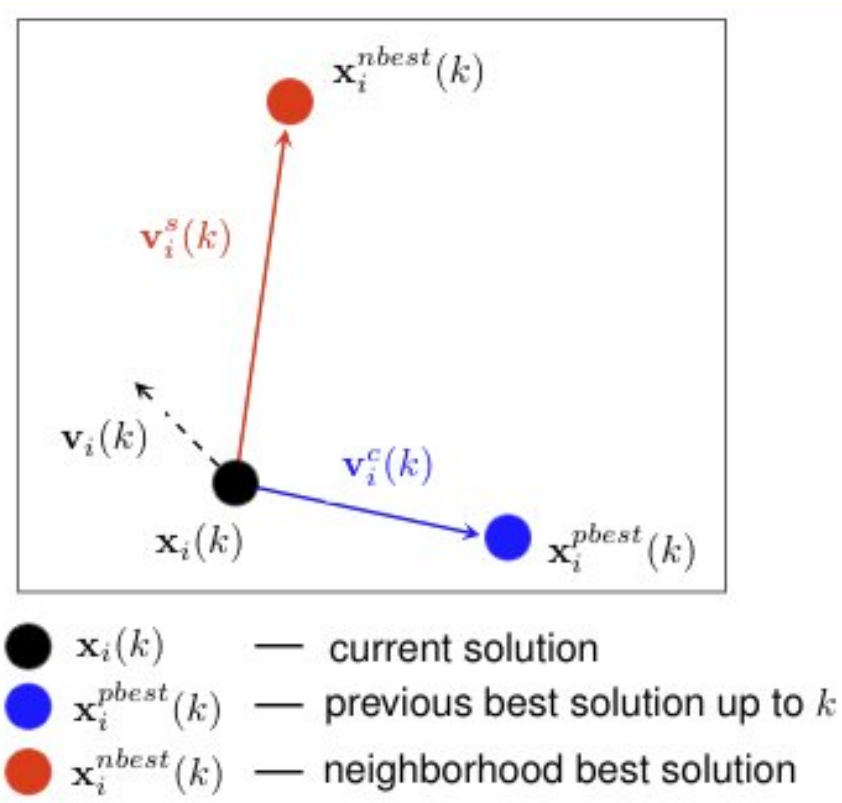
PSO algoritmasının uygulama alanları oldukça geniştir ve mühendislikten finansa, sağlık bilimlerinden yapay zekaya kadar birçok farklı problemin çözümünde kullanılmaktadır. Özellikle sürekli veya ayrık optimizasyon problemleri, fonksiyon optimizasyonu, ağ tasarımı ve kontrol sistemleri tasarımı gibi konular, PSO'nun başarılı bir şekilde uygulandığı alanlardandır.

Ancak, PSO algoritmasının da bazı sınırlamaları vardır. Bunlardan biri, özellikle karmaşık ve çok boyutlu çözüm uzaylarına sahip problemlerde, partiküllerin yerel optimuma erken yakınsaması ve global optimumu kaçırma riskidir. Bu durum, algoritmanın yakınsama hızını yavaşlatır ve bazen suboptimal çözümlere yerleşmesine neden olabilir.

Bu tür sınırlamalar, PSO algoritmasını daha etkin hale getirmek için çeşitli iyileştirme tekniklerinin geliştirilmesini teşvik etmiştir. Gelişmiş PSO versiyonları arasında, yerel arama stratejileri, adaptif hız ayarlama mekanizmaları ve çoklu hedef optimizasyonu destekleyen versiyonlar bulunmaktadır. Bu iyileştirmeler, PSO'nun çeşitli problemlere uygulanabilirliğini artırırken, algoritmanın etkinliğini ve güvenilirliğini de önemli ölçüde iyileştirmiştir.

Resim 5, Parçacık Sürü Optimizasyonu (PSO) algoritmasının iterasyon k sırasında bir partikülün konumunu göstermektedir. Partikülün şu anki konumu $x_i(k)$, önceki en iyi konumu $x_i^{pbest}(k)$ ve komşuluk içindeki en iyi konumu $x_i^{nbest}(k)$ ile temsil edilir. Her partikülün konumu, kendi deneyimleri ve komşu partiküllerin deneyimleri dikkate alınarak güncellenir.

Partikülün şu anki hızı $v_i(k)$ ile gösterilir ve bu hız, şu anki konumdan önceki en iyi konuma ve komşuluk içindeki en iyi konuma olan çekimlerle birleştirilerek yeni bir hız $v_i^{'}(k)$ oluşturulur. Bu yeni hız, partikülün bir sonraki konumu $x_i(k+1)$'e doğru hareket etmesine yardımcı olur. Şu anki konumdan önceki en iyi konuma olan çekim $v_i^c(k)$ ile ve komşuluk içindeki en iyi konuma olan çekim ise $v_i^g(k)$ ile gösterilmiştir. Bu iki çekim vektörü, partikülün hem kendi deneyimlerinden hem de komşulardan öğrenmesini ve genel en iyi çözüme doğru ilerlemesini sağlar.



Resim 5: PSO Algoritması Yeni Değer Belirleme [1]

4.1 PSO Geliştirilmiş - PSO Geleneksel

Geliştirilmiş PSO algoritması, klasik PSO yaklaşımını, yüksek boyutlu optimizasyon problemleri için daha verimli bir çözüm sağlayacak şekilde yenilikçi Sobol ve Halton kuazi-rastgele sayı örneklemeleri ile genişletir. Monte Carlo rastgele sayı örneklemelerini kullanan geleneksel PSO'ya kıyasla, geliştirilmiş PSO; Cigar, Elipsoid ve Paraboloid fonksiyonları gibi karmaşık ve sürekli değişken problemlerinde daha hızlı yakınsama ve global optimuma daha hızlı ulaşma avantajı sunar [1].

Bu çalışmada, karşılaştırmalı sonuçlar hem sürekli hem de karışık değişken problemleri üzerinde gerçekleştirilmiş ve her iki kuazi-rastgele sayı dizisinin kullanımının standart PSO'ya kıyasla etkinliği artırdığı gösterilmiştir. Özellikle, Sobol ve Halton dizilerinin kullanımıyla, boyut sayısı arttıkça elde edilen iyileşme oranının da arttığı tespit edilmiştir. Bu, geliştirilmiş PSO'nun, karışık değişken fonksiyonlarında Sobol'un ve sürekli değişkenlerde Halton'un tercih edilmesi gerektiğini göstermektedir. Sobol, TSP gibi tamamen ayrık problemlerde daha iyi performans göstermiş, ancak Halton sürekli değişken problemlerinde daha iyi bir yakınsama sergilemiştir.

Kuazi-rastgele sayı dizileri, partikül hareketinde kümeleşme veya yanlılığı azaltarak algoritmanın arama alanında daha geniş bir keşif yapmasını ve yerel optimadan kaçınarak global optimumu daha hızlı keşfetmesini sağlar. Bu iyileştirmeler, çeşitli optimizasyon alanlarındaki araştırmacılara potansiyel faydalar sunar ve algoritma-agnostik bir teknik olarak, Genetik Algoritma, Karınca Koloni Optimizasyonu (ACO) algoritması gibi diğer kurulu optimizasyon algoritmalarıyla birlikte kullanılabilir. Sonuçlar, bu yeni yaklaşımın, PSO'nun standart formunu kuazi-rastgele sayılarla değiştirmenin, hem sürekli hem de karışık değişkenli fonksiyonlarda elde edilen optimum fonksiyon değerine ulaşmak için gereken yinelemelerin sayısını tutarlı bir şekilde iyileştirdiğini göstermiştir.

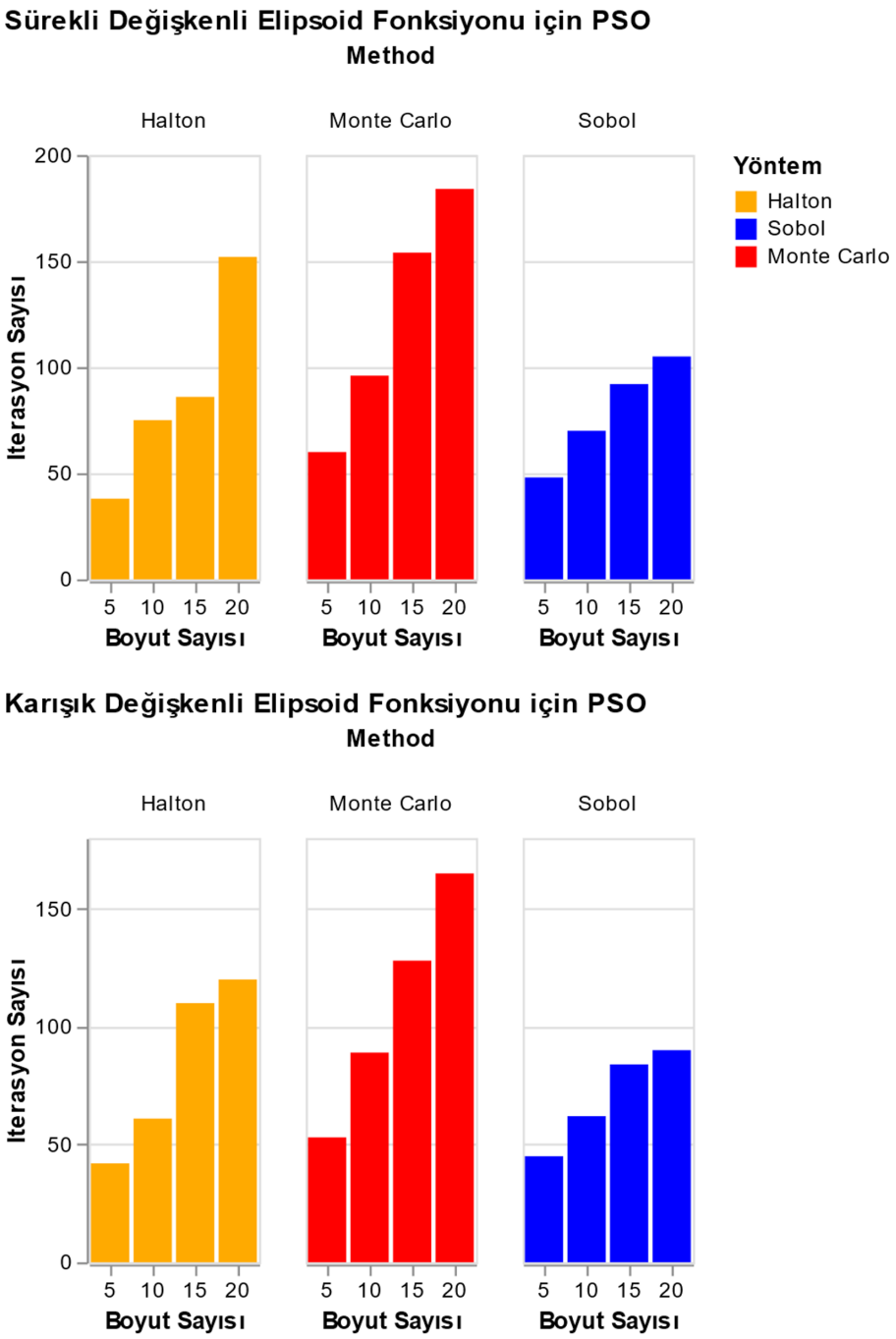
4.2 Performans Kıyaslaması

Resim 6'da elipsoid fonksiyonları için PSO algoritmasının performans karşılaştırması iki farklı senaryo altında gerçekleştirilmiştir: sürekli değişkenli ve karışık değişkenli durumlar. İlk grafikte, sürekli değişkenli Elipsoid fonksiyonu için Halton, Monte Carlo ve Sobol yöntemlerinin performansları farklı boyut sayıları (5, 10, 15 ve 20) için değerlendirilmiştir. Sobol yöntemi, özellikle yüksek boyut sayılarında daha az iterasyon ile daha hızlı bir yakınsama göstermiş, bu da algoritmanın verimliliğini artırmasını sağlamıştır. Monte Carlo, bütün boyutlarda en yüksek iterasyon sayısına sahipken, Halton orta seviyede bir performans sergilemiştir.

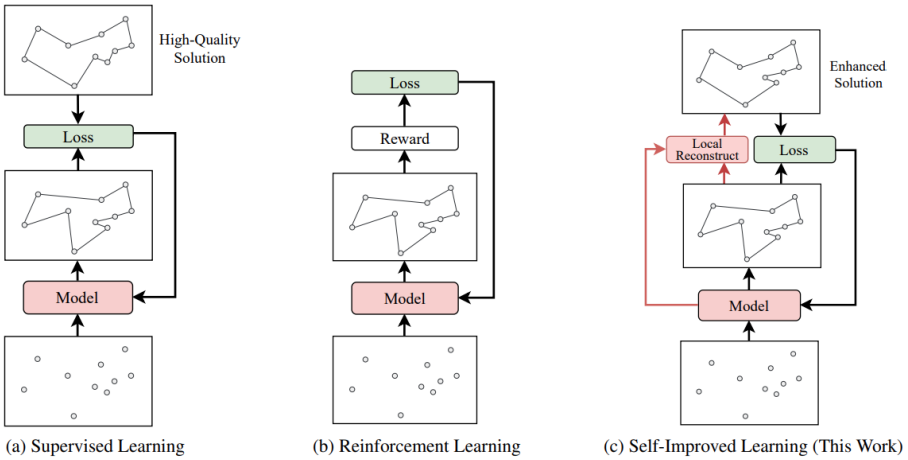
İkinci grafikte, karışık değişkenli Elipsoid fonksiyonu için aynı yöntemler karşılaştırılmıştır. Bu senaryoda da, Sobol yöntemi düşük ve yüksek boyut sayılarında en iyi performansı sergileyen yöntem olarak öne çıkmıştır. Halton, özellikle düşük boyut sayılarında, Monte Carlo'ya göre daha iyi bir performans göstermiş, ancak yüksek boyutlarda Sobol'un gerisinde kalmıştır. Monte Carlo'nun performansı yine diğer iki yönteme göre daha düşük olmuştur.

Bu karşılaştırma, PSO algoritmasının çeşitli rastgele sayı üretim yöntemlerini

kullanarak nasıl farklı optimizasyon senaryolarında değişken performanslar sergileyebileceğini göstermektedir. Sobol ve Halton gibi kuazi-rastgele sayı üreticilerinin, PSO'nun hızını ve çözüm kalitesini artırabileceğini kanıtlamaktadır.



Resim 6: PSO Algoritması Performans Kıyaslama [1]



Resim 7: SIL Algoritması Anlatım [2]

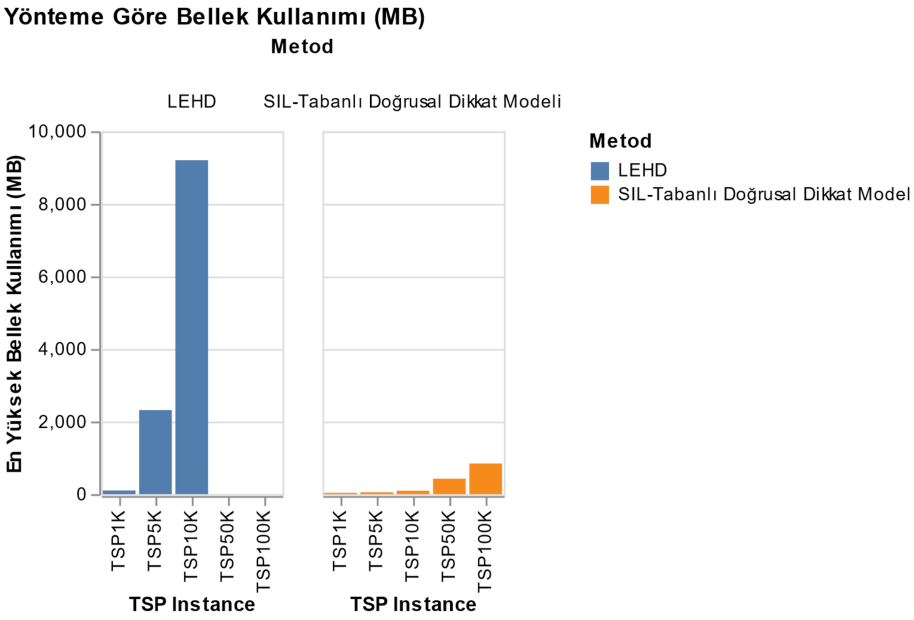
5 Self-Improved Learning[2]

5.1 Algoritma Anlatımı

Resim 7 görselinde sunulan SIL Algoritması, öğrenme süreçlerini temsil eden üç farklı yaklaşımın karşılaştırmalı bir anlatımını sunmaktadır. Denetimli öğrenme yaklaşımı (a), yüksek kaliteli bir çözüm elde etmek için modeli bir kayıp fonksiyonu üzerinden eğitmeyi içermektedir. Güçlendirme öğrenmesi (b), modelin ödül mekanizması vasıtasıyla iyileştirilmesi prensibini kullanırken, bu çalışmada[2] önerilen SIL (Self-Improved Learning) yöntemi (c), modelin sadece genel kayıp fonksiyonu üzerinden değil, aynı zamanda yerel olarak yeniden yapılandırma (Local Reconstruct) yoluyla kendi çözümünü iyileştirmesini içerir. Yerel yeniden yapılandırma işlemi, modelin sadece genel bir hedefe odaklanmak yerine, yerel düzeydeki veri örüntülerini ve dinamiklerini inceleyerek daha doğru ve özelleştirilmiş güncellemeler yapmasını sağlar. Bu, özellikle karmaşık problemlerde modelin daha hızlı ve etkin bir şekilde iyileşmesine olanak tanır. Her yaklaşım, problemleri çözme konusundaki farklı metodolojileri ve bu metodolojilerin algoritmanın performansına nasıl etki ettiğini göstermektedir.

5.2 Performans Değerlendirme

Resim 8 görselinde, farklı TSP (Travelling Salesman Problem) örnekleri üzerinde uygulanan SIL algoritması ve LEHD (Lineer Eniyileme Hedefli Düzlemsel) metodunun bellek kullanımını karşılaştıran bir performans değerlendirmesi yer almaktadır. SIL-Tabanlı Doğrusal Dikkat Modeli, LEHD'ye göre daha az bellek kullanımı göstermiş ve bu da algoritmanın verimliliğinin göstergesidir. Özellikle, büyük ölçekli TSP örneklerinde (TSP100K), SIL algoritmasının önemli ölçüde



Resim 8: SIL Algoritması Performans Değerlendirme [2]

daha düşük bellek kullanımı sağladığı görülmektedir, bu da büyük veri setleri üzerinde algoritmanın uygulanabilirliğini artırmaktadır.

Referanslar

[1] Shivakumar Kannan and Urmila Diwekar. An enhanced particle swarm optimization (pso) employing quasi-random numbers. 2024.

[2] Fu Luo, Xi Lin, Zhenkun Wang, Tong Xialiang, Mingxuan Yuan, and Qingfu Zhang. Self-improved learning for scalable neural combinatorial optimization. *arXiv preprint arXiv:2403.19561*, 2024.

[3] Büşra Özoğlu, Emre Çakmak, and KOÇ Tuğçe. Clarke & wright's savings algorithm and genetic algorithms based hybrid approach for flying sidekick traveling salesman problem. *Avrupa Bilim ve Teknoloji Dergisi*, pages 185–192, 2019.

[4] M Veeresh, T Kumar, and M Thangaraj. Solving the single depot open close multiple travelling salesman problem through a multi-chromosome based genetic algorithm. *Decision Science Letters*, 13(2):401–414, 2024.

[5] Yanming Zhang and Xianlong Li. Solving tsp problem based on discrete snake optimization algorithm. In *International Conference on Computer Graphics, Artificial Intelligence, and Data Processing (ICCAID 2023)*, volume 13105, pages 1177–1182. SPIE, 2024.