OPTİMİZASYON TEKNİKLERİ-6 Hafta

KARINCA KOLONİ ALGORİTMASI

1. Gerçek Karıncaların Davranışları

Gerçek karıncalar, yuvaları ile yiyecek kaynağı arasındaki en kısa yolu bulma kabiliyetine sahiptirler ve ayrıca çevredeki değişimlere de adapte olabilmektedirler. Örneğin, yuva ile yiyecek arasındaki en kısa yol belirli bir zamanda keşfedilir ve sonra çevre şartları nedeniyle bu en kısa yol artık en kısa yol olmaktan çıkarsa, karıncalar yeni en kısa yolu bulabilmektedirler. Diğer bir ilginç noktada karıncaların çok iyi görme kabiliyetlerinin olmamasıdır. Yani, en kısa yolu keşfetme uğraşında yönleri seçmek için etrafı tam olarak göremezler.

Karıncalar üzerine yapılan çalışmalar, en kısa yolu bulma kabiliyetlerinin birbirleri arasındaki kimyasal haberleşmenin bir sonucu olduğunu göstermiştir. Karıncalar birbirleriyle haberleşmede *feromon* olarak adlandırılan kimyasal bir madde kullanmaktadır. Karıncalar yürürken yolları üzerine bir miktar feromon maddesi bırakır ve her bir karınca yuva yada yiyecek bulmak için bir doğrultuyu seçer. Bir yönün seçilme ihtimali, bu yön üzerindeki feromon maddesi miktarına bağlıdır. Bütün yönlerin feromon miktarı birbirine eşit ise, o zaman bütün yönler karıncalar tarafından aynı seçilme olasılığına sahiptir. *Tüm karıncaların hızlarının ve yollara bıraktıkları feromon miktarının aynı olduğu kabul edildiğinde, daha kısa yollar birim zamanda daha çok feromon maddesi alacaktır.* Dolayısıyla, karıncaların büyük çoğunluğu hızla en kısa yolları seçecektir. Gerçek karınca kolonilerinin en kısa yolu bulmak için gösterdikleri davranış, doğal bir optimizasyon işlemini tanımlar.

2. Karınca Koloni Algoritması-KKA (Ant Colony Algorithm - ACA)

Dorigo ve arkadaşları tarafından önerilmiş en yeni sezgisel algoritmalardan biridir. Algoritma gerçek karınca kolonilerinin davranışları üzerine dayalıdır. Günümüze kadar KKA' nın yeni modelleri ortaya çıkmış ve bu modellerin özellikle ayrık optimizasyon problemlerinin çözümüne uygulanması konusunda çeşitli çalışmalar yapılmıştır.

Karınca koloni optimizasyon algoritması, yukarıda tanımlanan gerçek karınca kolonilerinin yapmış olduğu doğal optimizasyon işleminin yapay bir versiyonudur. Gerçek karınca kolonilerinin davranışını modelleyen temel bir algoritmanın adımları aşağıda verilmiştir.

Begin

Repeat

Bütün yapay karıncalar için yolların üretilmesi

Bütün yapay yolların uzunluğunun hesaplanması

Yapay yollar üzerinde bulunan feromon maddesi miktarının güncellenmesi

Şu ana kadar bulunan en kısa yapay yolun hafızada tutulması

Until (iterasyon=maksimum iterasyon yada yeterlilik kriteri)

End

Gezgin Satıcı Problemi

Gezgin satıcı problemi için algoritmayı ele alırsak, karıncalar çizelgedeki (graph) şehirleri gezerek paralel bir şekilde algoritmanın adımlarını uygularlar. Her karıncanın bir şehirden diğerine geçmesinde olasılığa dayalı bir kural uygulanır. Karınca k nın t. turunda i şehrinden j şehrine geçerken uygulanan Olasılık Fonksiyonu $P^k_{ij}(t)$ aşağıdakilere bağlıdır;

• **Tabu Listesi (Gezilen şehirlerin listesi)**: *j* ziyaret edilmemiş şehirler olmak üzere her sanal karınca gerçeğinden farklı olarak daha önce uğradığı yerlerin listesini tutar (Tabu Listesi). Tur sonunda liste tamamen dolarken yeni bir tur için liste yenilenir.

• **Visibility (Görünürlük)**: Gezgin satıcı problemi için bu değer gidilecek şehrin ne kadar yakın olduğunu gösterir. Problem boyunca sabit kalır ve sezgisel seçimliliği etkiler. Gösterimi (η_{ij}) şeklindedir. Değeri gezgin satıcı porbleminde iki şehrin arasındaki uzaklığın tersidir ($1/d_{ij}$).

• **Feromon Miktarı**: Karıncaların turları sonunda attıkları turun uzunluğuna bağlı olarak bıraktıkları feromon miktarıdır. Bu değer problem devam ettikçe değişir ve öğrenilmiş yönleri seçmede etkin rolü oynar. Gösterimi zamanın fonksiyonu olarak ($\tau_{il}(t)$) şeklindedir.

Karınca koloni algortmasının temelini teşkil eden Olasılık Fonksiyonu aşağıdaki gibi formülize edilir.

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \epsilon J_k(i)}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} & eğer\ j\ \in\ J_k(i), \\ 0 & eğer\ j\ \notin\ J_k(i) \end{cases}$$

Buradaki α parametresi Feromonin, β parametresi ise Görünürlüğün ayar parametreleridir. Bu değerler Olasılık fonksiyonunun sonucunu değiştirerek karar vermeyi etkilerler. Eğer α = 0 olursa sadece görünürlüğe(visibility) göre (β ya göre) bir seçim yapılır. β = 0 olursa sadece feromon miktarına göre seçim yapılacağından optimal bir çözüme ulaşılamaz. Bu yüzden her ikisinin de gerektiği kadar fonksiyona ağırlık katması gerekir.

Bununla birlikte, feromon buharlaşması olmaksızın algoritma güzel sonuçlar veremez. Çünkü arama uzayındaki ilk keşifler tamamen rasgele olduğundan, bu aşamada bırakılan feromonler herhangi bir bilgi içermezler. Dolayısıyla diğer karıncaların bu işe yaramayan değerleri kolayca unutması ve iyi çözümler üzerinden yol alabilmesi için bir buharlaştırma mekanizması gereklidir.

Algoritmanın Adımları

Karınca Miktar algoritmasına göre adımlar gösterilmiştir.

İlk değerleri ata

- Zamanı sıfırla (t=0) (Zaman aynı zamanda tur sayacıdır)
- Her kenar için [Kenar(i,j)] başlangıç foremen miktarını [$\tau_{ij}(t)=c$] ata ve feromon artış miktarını [$\Delta\tau_{ij}=0$]ise sıfırla.
- m tane karıncayı n tane şehre (düğüme) rastgele yerleştir. m: toplam karınca sayısı olur, n: ise şehir sayısı olur. Böylece her karıncanın ilk şehri belirlenmiş olur.

Zamanı/Turu başlat. Karıncaları yola çıkar.

- k=1 karıncadan başlayarak toplam karınca sayısı m kadar dön (For k=1 to m)
  k. karıncanın tabu listesine ziyaret edilen s indeks numaralı şehrini ekle. ($tabu_k(s)$)

  Her karıncanın turu ile ilgili hesaplamaları yap.

  - Karıncanın başlangıç şehrini tabu listesine ekle
  - Her k karınca için s=1 nolu şehirden başlayarak s=n-1 olan sondan bir önceki şehre kadar dön. Yani tabu listesi dolana kadar dön (For s=1 to n-1).
    - $P^{k}_{ij}(t)$ olasılığına göre hareket edilecek j şehrini seç.

      $$P_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \epsilon J_k(i)}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} & eğer\ j\ \in\ J_k(i), \\ 0 & eğer\ j\ \notin\ J_k(i) \end{cases}$$

    - k. karıncayı j şehrine hareket ettir.
    - j şehrini $tabu_k(s)$ listesine ekle.
  - k. karınca için L_k tur uzunluğunu hesapla
  - En iyi tur değeri ise bu değeri yenile.
  - Buharlaşmayıda içine katarak feromon miktarını hesapla ve gidilen kenara bırak.
    - k. karıncanın her kenara bırakacağı feromon artış miktarını aşağıdaki formülle hesapla

      $$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} & eğer\ (i,j)\ \in\ tabu_k, \\ 0 & diğer\ durumlarda \end{cases}$$

    - Her kenarın birikimli Feromon artışını hesapla
      $\Delta\tau_{ij} = \Delta\tau_{ij} + \Delta\tau_{ij}^{k};$
- Her kenar için buharlaşmayıda katarak zaman sonundaki feromon miktarlarını hesapla.
  $\tau_{ij}(t+1)=\rho.\tau_{ij}(t) + \Delta\tau_{ij}(t+1)$
- Zamanı bir artır (t=t+1)
- Feromon artışlarını sıfırla($\Delta\tau_{ij}=0$)

Aşağıyı düzenle

4. Tur sonunda tur uzunluğunu ölç ve en iyi tur değerini yenile

For k=1 to m
k. karıncanın indeks numarasını $tabu_k(n)$ den $tabu_k(1)$ getir.
L_k tur uzunluğunu her k. karınca için hesapla
En iyi tur değerini bul ve yenile
Her edge(i,j) yoluna buharlaşmayıda içine katarak feromon izini bırak.
For k=1 to m

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & eğer\ (i,j)\ \in\ tabu_k, \\ 0 & diğer\ durumlarda \end{cases}$$

$\Delta\tau_{ij} = \Delta\tau_{ij} + \Delta\tau_{ij}^{k}$;

değerlerini hesapla

Her edge(i,j) için $\tau_{ij}(t+1)=\rho.\tau_{ij}(t) + \Delta\tau_{ij}(t+1)$ eşitliğini hesapla

t=t+n ve NC=NC +1 değerlerini artır

Her edge(i,j) için $\Delta\tau_{ij}=0$ ata.

5. Sonlandırma

Eğer ($NC<NC_{max}$) ise {NC_{max}: maksimum tur sayısı}

Tüm karıncaların Tabu listesini boşalt.

2. basamağa git.

değilse

En kısa turu göster

2.1. Karınca Yoğunluk (Ant-Density) ve Karınca Miktar (Ant Quantity) Algoritmaları

Karınca yoğunluk modelinde karıncalar yollara birim uzunluk başına Q_1 miktarınca feromon bırakmaktadır. Karınca miktar modelinde ise yol uzunluğunu d olarak kabul edersek Q_2 / d_{ij} miktar feremon bırakmaktadır. Her iki algoritma da feromon miktarını, karınca bir noktadan diger bir noktaya geçtiğinde güncellemektedir.

Karınca yoğunluk modeli için $Q= Q_1$

Karınca miktar modeli için $Q= Q_2 / d_{ij}$

Bu tanımlamalardan hattaki feremon artışının karınca yoğunluk modelinde mesafeden bağımsız olduğu görülmektedir. Buna karşın karınca miktar modelinde ise artış mesafe ile ters orantılıdır. Yani karınca miktar modeli daha kısa hatları daha tercih edilir kılmaktadır.

Gezgin Satıcı Problemi için Karınca Yoğunluk ve Karınca Miktar Algoritmalarının temel adımları:

1. Tüm şehirlere belli miktarlarda bj(t) karınca yerleştir. Buna göre karıncaların tabu listesini yenile. Her hattın koku miktarını sıfırla. Sayacı sıfırla.

2. Tabu listeleri dolana kadar aşağıdaki işlemleri tekrarla.

Her şehirdeki tüm karıncalar için Olasılık Denklemi aracılığıyla hesaplanan Pij(t) değerine bağlı olarak hareket etmek amacıyla j şehrini seç.

Karınca k'yı j. şehre hareket ettir ve J. Şehri k. Karıncanın tabu listesine dahil et.

Koku miktarını yenile

$\Delta\tau_{ij}(t,t+1)= \Delta\tau_{ij}(t,t+1)+Q$

Her kenar(i,j) için, feromon maddesi miktarını hesapla.

3. Şu ana kadar bulunan en kısa turu hafızaya al.

Durdurma kriteri sağlanıyorsa Adım 4'e git. Yoksa tüm tabu listesini boşalt.

Tüm şehirlere belli miktarda karınca yerleştir ve 2. adıma git.

4. En kısa turu yaz ve dur.

2.2. Karınca Çevrim Algoritması (Ant-Cycle)

Karınca çevrim algoritmasında her hareket sonrası değil tam bir tur bittiğinde feremon miktarı yenilenmektedir. Bu nedenle bu algoritmanın daha iyi performans göstereceği beklenmektedir. Sebebi ise üretilen çözümün problem için tam bir çözüm olmasıdır.

Gezgin Satıcı Problemi için Karınca Çevrim Algoritmasının temel adımları:

1. Tüm şehirlere belli miktarlarda bj(t) karınca yerleştir. Buna göre karıncaların tabu listesini yenile. Her hattın koku miktarını sıfırla.Sayacı sıfırla.

2. Tabu listeleri dolana kadar aşağıdaki işlemleri tekrarla.

Her şehirdeki tüm karıncalar için Olasılık Denklemi) aracılığıyla hesaplanan Pij(t) değerine bağlı olarak hareket etmek amacıyla j şehrini seç.

Karınca k'yı j. şehre hareket ettir ve J. Şehri k. karıncanın tabu listesine dâhil et.

3. Tüm karıncalar için tur uzunluğunu hesapla.

Her kenar için (t,t+1) zaman aralığında depolanan koku miktarını hesapla

4. Her kenar(i,j) için, ηij(t+n) feremon maddesi miktarını hesapla.

$$\tau_{ij}(t+n)=\rho^{*}\tau_{ij}(t)+\Delta\tau_{ij}(t,t+n) \qquad \Delta\tau_{ij}(t,t+n)=\sum_{k=1}^{m}\Delta\tau_{ij}(t,t+n)$$

5. Şu ana kadar bulunan en kısa turu hafızaya al.

Durdurma kriteri sağlanıyorsa Adım 7'e git. Yoksa tüm tabu listesini boşalt.

Tüm şehirlere belli miktarda karınca yerleştir ve 2. adıma git.

6. En kısa turu yaz ve dur.

Yukarıda bahsedilen 3 algoritmaya tek isim olarak Karınca Sistem (Ant-System) algoritması da denilmektedir. Bunlar Ant-Cycle, Ant-Density, Ant-Quantity. Bazen Ant-Cycle algoritması Ant-System olarakda anılmaktadır.

2.3. Karınca Max-Min Algoritması

KKO üzerine yapılan araştırmalarda en iyi çözüme sahip olan karıncayı çözüm araştırması esnasında daha fazla kullanmanın performansı artırdığı gözlemlenmişti. Karınca Max-Min Algoritması üç mad ile Karınca Sistem Algoritmasından ayrılmaktadır.

1. Bir çevrimde en iyi sonuçların kullanılabilmesi için her çevrimde tek bir karıncanın feromon izini güncellemesi gereklidir. Söz konusu karınca ya çevrimdeki en iyi sonucu bulan(*çevrim-eniyi*) yada algoritmanın başından beri en iyi sonucu bulan karınca olur(*global-eniyi*).
2. Çözüm uzayında aramanın durağanlaşmasından yani tıkanmasından kaçınmak için mümkün feromon izlerinin güncellenmesi[min,max] aralığında kısıtlanır. Böylece güncellenen feromon bu aralığın dışında değer alamaz.
3. Feromon izlerini başlangıçta max'a eşitleyerek daha geniş çözüm araştırmaları sağlanmış olur.

Feromon güncellemesi aşağıdaki gibi olmaktadır.

$$\tau_{ii}=\begin{cases}\tau_{max} & \text{eğer } (1-\rho)\,\tau_{ij}^{t-1}+\Delta\tau_{ij}^{eniyi}>\tau_{max}\\ \tau_{min} & \text{eğer } (1-\rho)\,\tau_{ij}^{t-1}+\Delta\tau_{ij}^{eniyi}<\tau_{min}\\ (1-\rho)\,\tau_{ij}^{t-1}+\Delta\tau_{ij}^{eniyi} & \text{aksi halde}\end{cases}$$

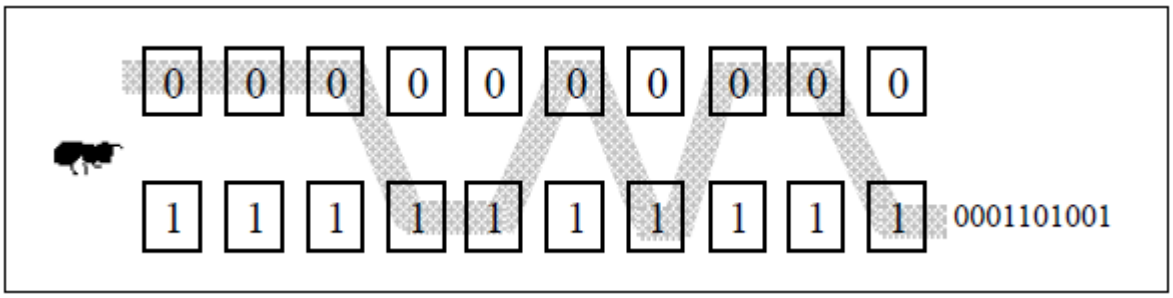
$$\Delta\tau_{ij}^{eniyi}=\begin{cases}Q/L_{eniyi} & \text{eğer en iyi çözümü bulan karınca,}\\ & \text{(i,j) yolunu kullanıyorsa}\\ 0 & \text{değilse}\end{cases}$$

2.4. Mertebe Temelli Karınca Algoritması

Bu algoritma da Karınca Sistemi algoritmasındaki formülasyon kullanmaktadır. fakat farkı feromon madde güncelleştirmesinde olmaktadır. Bu algoritmada yalnızca en iyi çözümü bulan karınca ve o tur içerisinde iyi çözümler bulan belli sayıda karıncanın feromon eklemesine izin verilmektedir.

2.5. TACO (Touring Ant Colony Optimisation) Algoritması:

Bu algoritma Hiroyasu ve arkadaşları tarafından özellikle sürekli optimizasyon problemleri için önerilmiştir. Bu algoritmada çözümler ikili sayılarla temsil edilmiş tasarım parametrelerinin bir vektörüdür. Dolayısıyla bir çözüm, ikili sayıların alt guruplarından oluşan bir vektördür. Bu nedenle, her bir yapay karınca dizideki ikili sayının değerini araştırır. Başka bir deyişle ikili sayının değerinin 1 yada 0 olup olmadığına karar vermeye çalışır. TACO algoritması kavramı aşağıda şekilde gösterilmiştir.



Şekil. Bir karınca tarafından bulunan yapay bir yol (çözüm).

Bir ikili sayının değeri için karar verme aşamasında, karıncalar sadece feromon maddesi bilgisini kullanır. Bir karınca dizideki tüm ikili sayıların değeri için karar verdiğinde problem için bir çözüm üretmiş demektir. Bu çözüm, problem için değerlendirilir ve kalite fonksiyonu olarak adlandırılan bir fonksiyon aracılığıyla çözüme ilişkin bir kalite değeri elde edilir. Bu değere bağlı olarak bir yapay feromon maddesi miktarı ikili sayılar arasında oluşan yapay bütün alt yollara yapıştırılır.

3. Özet

Karınca koloni sistemi 5 anahtar kelime temelinde incelenebilir. Bunlar;

- Yakınlık (Visibility): Problem boyunca sabit kalan ve sezgisel seçimliliği etkileyen sezgisel değer η_{ij}: Bu değer Gezgin Satıcı için görünürlük (visibility) olup iki şehrin arasındaki uzaklığın tersidir ($1/d_{ij}$).
- Feromon: Karıncaların gittikleri yol üzerine bıraktıkları kimyasala verilen isimdir. Algoritmalarda koku olarak da adlandırılır.
- Tabu Listesi: Her sanal karınca gerçeğinden farklı olarak daha önce uğradığı yerlerin listesini tutar ve bu yerlere tur tamamlanıncaya kadar tekrar uğranmaz.
- Karar verme: Hangi yolun seçileceği aşağıdaki olasılık formülüne bağlı olarak belirlenir. En yüksek olasılığa sahip yol tercih edilir.

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in J_k(i)}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} & \text{if } j \in J_k(i), \\ 0 & \text{if } j \notin J_k(i), \end{cases}$$

- Buharlaşma: Yoldaki feromonin belli bir oranda azaltılmasını ifade eder. İlk keşifler tamamen rasgele olduğundan, bu aşamada bırakılan feromonler herhangi bir bilgi içermezler. Dolayısıyla diğer karıncaların bu işe yaramayan değerleri kolayca unutması ve iyi çözümler üzerinden yol alabilmesi için buharlaştırma mekanizması gereklidir.

Karınca koloni algoritmaları feromonin ne zaman, ne kadar ve nasıl bırakılacağı üzerine çeşitlenir. Algoritmaları bu anlamda bir tablo içinde özetlersek tablomuz aşağıdaki gibi olacaktır.

Feromon	**Karınca Yoğunluk Algoritması**	**Karınca Miktar Algoritması**	**Karınca Çevrim Algoritması**	**Max-Min Karınca Sistemi**	**Mertebe Temelli Karınca Sistemi**
Güncelleme zamanı	Her adımdan sonra	Her adımdan sonra	Tam bir tur bittikten sonra	Tam bir tur bittikten sonra	Tam bir tur bittikten sonra
Guncelleyen Karınca	Her biri	Her biri	Her biri	En iyi karınca çevrim-eniyi veya global-eniyi	En iyi n karınca
Güncelleme	$\tau_{ij}(t+1)=\rho^{*}\tau_{ij}(t)+\sum_{k=1}^{m}\Delta\tau_{ij}(t,t+1)$			$\tau_{ij}(t+1)=(1-\rho)^{*}\tau_{ij}(t)+\Delta\tau_{ij}(t,t+1)$	

Artışı(fark)	$\Delta\tau_{ij} = Q_1$	$\Delta\tau_{ij} = Q_2 / d_{ij}$	$\Delta\tau_{ij} = Q_3 / L_k$	$\Delta\tau_{ij} = Q_4 / L_{eniyi}$	$\Delta\tau_{ij} = Q_5 / L_{nkarınca}$

Tablo 1. Literatürde karınca algoritmaları ile ilgili yayınlar

Problem	Yazar	Yıl	Algoritma
Gezgin Satıcı Problemi	Dorigo, Maniezzo & Colorni	1992	AS
	Gambardella & Dorigo [8]	1995	Ant-Q
	Dorigo & Gambardella	1996	ACS & 3 opt
	Stützle & Hoos [21]	1999	MMAS
	Bullnheimer, Hartl &Strauss[25]	1996	AS_{rank}
Karesel Atama	Dorigo, Maniezzo & Colorni	1994	AS-QAP
	Gambardella, Taillard &Dorigo[22]	1997	HAS-QAP
	Stützle & Hoos [15]	1998	MMAS-QAP
	Maniezzo & Colorni [23]	1998	AS-QAP
	Maniezzo [24]	1998	ANTS-QAP
Araç Rotalama	Bullnheimer, Hartl & Strauss [13]	1996	AS-VRP
	Gambardella, Taillard &Agazzi[26]	1999	HAS-VRP
Bağlantı Temelli Network Rotalama	White, Pagurek & Oppacher [16]	1998	ASGA
	Di Caro & Dorigo [7]	1998	AntNet-FS
	Bonabeau, Henaux, Guerin Snyers, Kuntz & Theraulaz [29]	1998	ABC-smart ants
Bağlantısız Network Rotalama	Di Caro & Dorigo [30]	1997	Ant Net
	Subramanian, Druschel & Chen[31]	1997	ACS
	Heusse, Guerin, Snyers & Kauntz	1998	CAF
	Van der Put & Rothkrantz [32]	1998	ABC-backward
Grafik Boyama	Costa & Hertz [34]	1997	ANTCOL
Sequential Ordering	Gambardella & Dorigo [35]	1997	HAS-SOP
Kesikli Optimizasyon	Dorigo & Maniezzo	1994	Tüm
	Dorigo & Gambardella	1996	Ant-Q
	Dorigo & Stützle	2001	Simple AC
	Middendorf, Reichle & Schmeck	2000	Multi-Colony
Atölye Tipi Çizelgeleme	Colorni, Dorigo, Maniezzo & Trubian [2]	1994	AS-JSP
Tek Makine Çizelgeleme	Den Besten, Stützle, Dorigo	1999	ACO

Proje

Genetik algoritma, Karınca Koloni ve Isıl İşlem Algoritması gibi 3 tane algoritmanın performanslarının karşılaştırıldığı bir proje yapılabilir. Her algoritmanın üç tane alt özellikleri değiştirilerek Algoritmanın da kendi içerisinde farklı versiyonlarınında karşılaştırılması sağlanabilir.

NOTLAR

Arılar, karıncalar ve hatta bakteriler hayatta kalma stratejilerini çok karmaşık grup davranış biçimleri ile gerçekleştirirler. Günümüzdeki bilim adamları bu davranış biçimlerini ayrıntılı inceleyip değişik uygulamalarında örnek almaktadır.

Karıncalar, kolonilerinin menfaatleri için beraber çalışan sosyal böceklerin en iyi örneklerindendir. Koloni halinde yaşayan karıncalar yiyecek bulmak için ilk olarak öncü karıncaları tek başına gönderirler. Bu öncüler çevreyi araştırarak uygun yiyecek kaynağını bulmaya çalışır. Öncüler yiyecek bulursa, kolonin olduğu yere geri dönerken arkalarında özel bir koku izi bırakarak ilerler. Bu iz sayesinde diğer karıncalar da bu yiyecek kaynağını bulabilirler.

Aslında karıncalar yukarıda bahsedilenden çok daha karmaşık bir yöntemi uygularlar. Yiyecek kaynağını başarıyla bulan öncü karınca geri dönerken en kısa yoldan dönmüş olmayabilir, tabii bu da kolonidekilerin karmaşık yollarla kaynağa gitmesine neden olacaktır. Aynı yiyecek kaynağını keşfeden başka bir öncü karınca belki buraya daha kestirme bir yol bulmuş olabilir. Peki kolonidekiler hangi öncünün en kestirme yolu bulduğunu nasıl bilecektir? Bu nedenle bazen koloniler gereksiz derecede uzun bir yoldan gitmek zorunda kalabilirler. Ama kestirme yollardaki izler daha düzenli olarak yenilenir ve bu sayede de karıncalar daha belirgin izi olan yani daha kısa yolu tercih ederek gereksiz derecede uzun yollardan ilerlemek zorunda kalmazlar.

Bu tür bir plan benzeri zaman alan karmaşık bilgisayar problemlerinin çözülmesine örnek teşkil edebilir. Dünyanın en saygın birkaç bilim dergisinden birisi olan Nature, geçen sayısında ABD'deki New Mexico eyaletinin Santa Fe Enstitüsü'nde yapılan bu konuyla ilgili bir araştırmayı yayınladı. Bazı problemleri çözmenin tek yolu cevapla ilgili bütün ihtimallerin tek tek test edilmesi ile gerçekleşir. Buna verilebilecek klasik bir örnek, şehir şehir dolaşan bir işadamının izlemesi gereken en kısa rotanın bulunmasıdır. Bu tür problemler yukarıda örneği verilen arkalarında iz bırakarak ilerleyen öncü karıncaların uyguladığı yöntemle çözülebilir. Santa Fe Enstitü'sündeki Bonabeau ve arkadaşları "sanal karıncalar" oluşturarak benzeri problemlerin bilgisayarlarla daha kolay çözülebileceğini gösterdiler. Buna göre sanal karıncalar arkalarında buldukları rotanın uzunluğunu da simgeleyen bir nevi koku izi bırakacak ve diğer sanal karıncalar da kestirme rotaları bu sayede bularak tercih edeceklerdir. Koku izinin kokusunu veren maddenin belirli bir hızda buharlaşması da simüle edilerek tercih edilmeyen uzun rotalardaki koku izleri de yavaş yavaş yok olacak ve bu da sanal karıncaların kestirme yol dışındaki uzun rotalara sapmasını önleyecektir.

Araştırmacılar bu yeni problem çözme metodunu "Karınca Koloni Optimizasyonu Algoritması" olarak adlandırmaktadırlar. Bu algoritme bilgisayar problemlerinin çözülmesinde kullanılan Sürü Zekası Yaklaşımına yeni bir örnektir. Şu sıralarda İsviçre'deki petrol tankerlerinin rotalarının oluşturulmasında Karınca Koloni Optimizasyonu Algoritmasının potansiyel rolü incelenmektedir.

Mühendisler sanal karıncalarını networklerine bırakarak bazı problemlerini çözebilecekleri gerçeğini şimdiden çok sevmişler gibi görünüyor. Haberleşme ağlarında kullanılan yönlendirici sinyallerin en kısa rotadan gönderilmesi, trafik sıkışıklığının önlenmesi gibi problemlerin de bu yöntemle kolayca çözülebileceği düşünülmektedir. Karınca Kolonisi Yönlendirmesinin son derece esnek olması ve networke yeni kanalların eklenmesi veya çıkarılması gibi değişikliklerin kolayca adapte edilebilmesi de önemli avantajları arasında sayılmaktadır. Bugünlerde İngiliz Telekom firması bu yeni algoritmayı telekomunikasyon sistemlerine adapte etmeye çalışmaktadır.

Bu algoritmanın en heyecan verici uygulamalarından birisi kollektif hareket eden minik robotların yapılmasında olacaktır. Minik bir robot kolonisi karıncalardan öğrendiğimiz bu algoritma sayesinde daha basit programlama prensipleri kullanarak karmaşık işlemleri gerçekleştirebilecekler.

Kaynaklar: 1.Bonabeau, E., Dorigo, M. & Theraulaz, G. Inspiration for optimization from social insect behaviour. Nature 406, 39-42 (2000).

Kaynaklar

Videolar

1. https://www.youtube.com/watch?v=eVKAIufSrHs
2. http://www.youtube.com/watch?v=SJM3er3L6P4

3. http://www.youtube.com/watch?v=8WjUVEamGKA
4. https://www.youtube.com/watch?v=Y-yW1nvfnkI

BU NOTLAR DAHA DÜZENLENECEK...