ANT SYSTEM ALGORİTMASININ JAVA İLE GÖRSELLEŞTİRİLMESİ

Aybars Uğur
Ege Üniversitesi
Bilgisayar Müh. Bölümü
aybars.ugur@ege.edu.tr

Doğan Aydın
Ege Üniversitesi
Bilgisayar Müh. Bölümü
dogan.aydin@ege.edu.tr

ÖZET

Bu çalışmada sürü zekası, karınca kolonisi optimizasyonu ve Ant System algoritması hakkında bilgiler verilmiş, gezgin satıcı probleminin (Traveling Salesman Problem) (TSP) karınca sistemi algoritması ile çözümünü içeren Java uygulaması geliştirilmiştir. İnternet üzerinden erişilen bu program ile kullanıcılar sisteme değişik sayıda şehirler ekleyebilmektedir. Bu sayede sistemin büyüklüğünü ve verilerini değiştirebilmeleri sağlanmıştır. Ayrıca kullanıcılar sistemde kritik görevler alan parametrelerin değerlerini etkileşimli olarak değiştirmekle bu parametrelerin etkilerini kolaylıkla görebilmektedirler. Bununla birlikte çözümün adımları grafiksel olarak da gösterilmektedir. Bunlar sayesinde, bu uygulama ile farklı problem boyutları için görsel olarak problemin çözümleri bulunmakta ve algoritmanın çalışma prensipleri etkileşimli olarak kullanıcılara sunulmaktadır.

ABSTRACT

In this study, information is given about Swarm Intelligence, Ant Colony Optimization and Ant System Algorithm and a Java application that solves The Travelling Salesman Problem with this algorithm is generated. In this application that is accessed on the internet, Users may add various number of cities to the system. Thus, it supplies that the sistem size and data can be changed. Furthermore users may easily see the effects of values of parameters, that have critical mission in it, via changing their values interactively. In addition solution steps are shown graphically. Thanks to the all, for different size of problem solutions are found visually and working priciples of the algorithm is presented to the users interactively by this application.

Anahtar kelimeler : Sürü Zekası, Karınca Kolonisi Optimizasyonu, Ant System Algoritması, Gezgin Satıcı Problemi (TSP)

1. GİRİŞ

Uzun zaman önce, insanlar doğada bulunan hayvan ve böcek sürülerinin davranışlarını keşfettiler. Kuş sürülerinin havada süzülmesi ve farklı şekil alması, karıncaların yiyecek araştırması, balık sürülerinin beraberce yüzmesi ve kaçışması bu sürü davranışlarından sadece birkaçıdır. Son yıllarda ise biyologlar ve bilgisayar uzmanları "Yapay Yaşam" alanı kapsamı altında bu sürülerin davranışlarının nasıl modellenebileceği ve aralarındaki iletişimin mantığını üzerinde çalışmalar yapmaktadırlar. Üstelik "Sürü zekası" (Swarm Intelligence) adı verilen bu yaklaşımların optimizasyon problemleri, robotbilim ve askeri uygulamalarda başarılar göstermeleri bu konu üzerindeki çalışmaları arttırmıştır.

Sürü zekası, özerk yapıdaki basit bireyler grubunun kolektif bir zeka geliştirmesidir.[1] Bu ise "Stigmergy", yani "ortam"daki etmenlerin ortama müdahale ederek iletişim kurmaları ve birbirlerinin hareketlerini düzenlemeleri, ve "Self-Organization" (Kendinden organizasyon) denen iki mekanizma üzerine kuruludur. Stigmergy vasıtasıyla iletişim, bireyler yaptıkları işlerle ortamda değişikliğe sebep olarak, sağlanırken, kendinden organizasyon yardımıyla önceden yapılmış herhangi bir plan olmadan sonuç üretebilmelerini, esnek ve sağlam, merkezi bir yönetim birimi olmadan yapılanmalarını sağlar.

Sürülerin bu özelliklerini ve yaptıkları işleri (Yem bulma, taşımada yardımlaşma, kolektif kümelenme gibi) klasik yapay zeka kavramına yeni bir soluk getirmiştir. Klasik yapay zeka kavramında bulunan insan zekası modelleme odaklı, karmaşık, merkezî, planlı yaklaşımların aksine, sürü zekası basit yapılı, özerk, önceden planlama yapmayan dağınık ajanların komplex problemlerin çözümünde başarılı olduklarını göstermiştir. Bunların en başarılıları ise Karınca Kolonisi Algoritmaları ile Particle Swarm Optimization (Parçaçık Sürü Optimizasyonu) algoritmalarıdır. Bu algoritmaların benzerlerine olan üstünlükleri ve gelişime açık olmaları sürü zekasının yapay zekanın gittikçe önem kazanan ve gelişen bir konusu olmasını sağlamıştır. Bu yazımızda da sürü zekası uygulamalarının en başarılılarından biri olan Karınca Kolonisi Algoritmalarına değindikten sonra bu algoritmalardan ilki olan Ant System algoritmasından ve bu algoritma üzerinde yaptığımız bir simülasyon uygulamasından bahsedeceğiz.

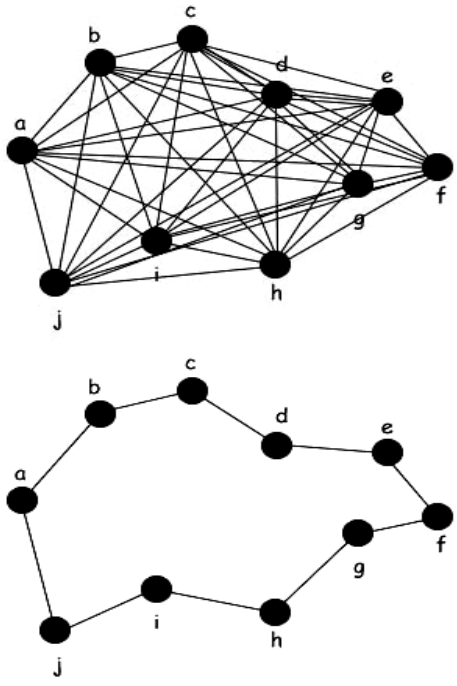
2. KARINCA KOLONİSİ OPTİMİZASYONU

Karınca Kolonisi Optimizasyonu (ACO) bir çok kombinasyonel optimizasyon problemlerinde iyi sonuçlar veren bir meta-heuristic tekniktir [2][3]. Bu tekniğin geliştirilmesinde gerçek karınca kolonilerinin gıda arama tekniklerinden faydalanılmıştır. Bir çok karınca kolonisinde karıncalar yiyeceklerini ararken, öncelikle yuvalarının etrafında rasgele dolaşarak keşfe başlarlar. Yiyecek kaynaklarını bulduklarında, yiyeceğin kalitesini ve miktarını değerlendirdikten sonra bir kısmını yuvaya taşırlar. Bu dönüş sırasında diğer karıncaların da aynı kaynağı bulabilmeleri için yiyeceğin kalitesine ve miktarına bağlı olarak kimyasal feromen (pheromone) maddesini geçtikleri yolun üzerine bırakırlar. Bırakılan bu izler diğer karıncalara rehberlik ederek belli olasılıkla o yolu takip etmelerini ve kaynağı bulmalarına yardım eder. Bu şekilde feromen vasıtasıyla yapılan dolaylı iletişim (stigmergy), karıncaların gıda ile yuva arasında en kısa yolu bulmalarına olanak tanır. İşte karıncaların bu davranışları ACO algoritmalarının geliştirilmesinde ilham kaynağı olmuştur.

İlk ACO algoritması 1991 yılında M. Dorigo tarafından doktora tezi olarak gerçekleştirilmiş ve yayınlanmıştır. Bu algoritmaya ise "Ant System" yani karınca sistemi (AS) adını vermiştir. Dorigo Bu algoritmayı değişik boyutlardaki bir çok TSP probleminde denemiş, Küçük ölçekli TSP problemlerinde (75 şehirden az TSP' lerde) başarılı olurken daha büyük ölçeklilerde başarılı olamamıştır. Ant System algoritması aslında Ant-Cycle, Ant-Density ve Ant-Quantity olmak üzere üç farklı algoritma kümesinden oluşmaktadır. Bunların arasındaki fark ise feromenin depolanmasında yatmaktadır. Ant-Cycle algoritmasında feromen sadece her karınca turunun sonunda o karıncanın turu üzerindeki her kenar, uzunluklarıyla ters orantılı olarak feromenle depolanmaktadır. Bununla birlikte her sanal karınca gerçeğinin aksine geçtiği yolları hatırlamak için bir hafızaya sahiptir. Diğer iki algoritmada ise feromen yolu, karınca bir şehirden (noktadan) diğer bir noktaya geçerken güncellenmektedir. Yapılan testler sonunda Ant-Cycle algoritmasının daha iyi sonuçlar verdiği ortaya çıkmış ve genel bir yapı kazanmış ve Ant System algoritması denildiğinde bu algoritma kastedilir hale gelmiştir. Algoritma üzerindeki ilk geliştirim, Elitist strategy olarak yine Dorigo tarafından 1996' da gerçekleştirilmiştir. Bu yaklaşımda arama süreci esnasında bulunan en iyi yola, deamonlar tarafından daha fazla feromen doldurulması yöntemi kullanılarak daha iyi sonuçlar elde edilmiştir. Yine 1996 da Dorigo ve Gamberdalla tarafından Ant Colony System (ACS) algoritması geliştirilmiştir. ACS, "en iyi sonucun komşularında" sonucu arama üzerine yoğunlaşmaktadır. Bunu ise iki mekanizma ile gerçekleştirir. Birincisi, sadece en iyi karıncalara feromen bırakması izin verilmesi, diğeri ise bir şehirden diğer bir şehre gidilirken denge unsuru olarak bir pseudo-random orantı kuralı kullanılmasıdır. 1997' de ise Hoos ve Stutzle feromen izini maximum ve minimum aralıklarda sınırlayan MIN-MAX Ant System adında bir algoritma önermişler, 1999' da Bullnheimer, Hartl ve Stauss ASRank adı altında Elitist Strategy nin gelişmiş bir çeşidini sunmuşlardır. Bu algoritmaya göre her tur sonunda karıncalar tur uzunluklarına göre sıralanmakta, sadece sıralamadaki belli sayıda en iyi karıncaların ve o ana kadarki en iyi karıncanın belli bir miktara göre feromen bırakmasına izin verilmektedir. Bu temel algoritmalardan başka daha bir çok algoritma ACO alanında geliştirilmiş ve geliştirilmektedir. İşte AS den bu güne kadar bu kadar gelişim gösteren ACO algoritmaları bugün NP-Hard (Zor) problemlerin çözümünde genetik algoritmalar, tavlama benzetimi, tabu araştırmaları gibi diğer meta-sezgisel yaklaşımlı rakiplerine rağmen en iyi algoritmalar haline geldiler.

3. ANT SYSTEM ALGORİTMASI

Karıncaların feromen bırakma ve takip etme mantığı üzerine kurulu olan ilk algoritmadır. Bu algoritmadan kastedilen yukarıda da bahsedildiği gibi Ant-Cycle algoritmasıdır. Bu algoritma ilk olarak Travelling Salesman Problem(TSP) problemi üzerinde denenmiştir. TSP problemi bir kişinin, verilen şehirler üzerinden sadece bir kez geçmek şartıyla, tüm şehirleri dolaşarak en kısa turu bulması problemidir. Bu problem NP-Hard bir problemdir. Şekil 1'de de gösterildiği gibi bir çok tur ihtimali bulunmaktadır ve normal algoritmik yöntemlerle zaman karmaşıklığı O(n!) dir.



Şekil 1. Örnek bir TSP ve Çözümü

ilk olarak TSP nin seçilmesinde ise;

- Shortest path problemi olduğu için kolay adapte edilebilir olması
- NP-hard bir problem olması
- Çokça kullanıldığı için başka algoritma testleriyle kolayca karşılaştırılabilir olması
- Herkesin kolayca anlayabileceği bir problem olması

rol oynamıştır.[4]

Ant System algoritmasında, karıncalar sonucu sırayla çizgedeki (graph) şehirleri gezerek paralel bir şekilde çözerler. 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;

- *j* şehrine gidilip gidilmediği ve ziyaret edilmemiş şehirler: 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.
- Problem boyunca sabit kalan ve sezgisel seçimliliği etkileyen sezgisel değer η_{ij}: Bu değer TSP için görünürlük (visibility) olup iki şehrin arasındaki uzaklığın tersidir ($1/d_{ij}$).
- İki şehir arasındaki feromen miktarı: Karıncaların turları sonunda attıkları turun uzunluğuna bağlı olarak bıraktıkları feromen miktarıdır. Bu değer problem devam ettikçe değişir ve öğrenilmiş yönleri seçmede etkin rolü oynar.

Bu fonksiyon aşağıdaki gibi formülize edilir.

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

Buradaki α ve β değerleri ayar parametreleridir ve feromen izi ile sezgisel fonksiyonun katılım oranlarını etkilerler. Eğer α = 0 olursa sadece görünürlüğe(visibility) yani sezgisel fonksiyona göre bir seçim yapılır ve algoritma stokastik bir greedy search algoritmasına döner. β = 0 olursa sadece feromen izine göre seçim yapılacağından optimal bir çözüme ulaşılamaz. Bu yüzden her ikisinin de gerektiği kadar fonksiyona ağırlık katması gerekir.

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

Diğer önemli bir parametre ise sistemde var olan karınca sayısıdır. Kullanılan karınca sayısı çok fazla olursa sub-optimal sonuçlara neden olurken, bu sayının az olması feromen buharlaşması nedeniyle beklenen kollektif çalışma özelliği kaldıracaktır. O yüzden Dorigo karınca sayısının şehirlere eşit olmasını (m = n) ve karıncaların başlangıçta rasgele şehirlere dağıtılmalarını uygun bulmuştur.

Bu verilen bilgilere göre AS algoritması Şekil 2'deki gibidir.

1. İlkleme:
t = 0 {zaman sayacı}
NC:=0 {NC tur sayacı}
Her edge (i,j) için τij(t)=*c ve* Δτij= 0 olarak ilkle {τij feromen yoğunluğu}
m tane karıncayı n tane şehre(düğüme) rasgele yerleştir {m toplam karınca sayısı}

2.Tur başlangıcı
s:=1 {s tabu listesi(ziyaret edilen şehirlerin listesi) indexi}
For k:=1 to m do
k. karıncanın **tabu**$_k$(s) listesine başlangıç şehrini ekle

3.Her karıncanın turu
Tabu listesi dolana kadar tekrarla {bu basamak n-1 defa tekrarlanacak}
s:=s+1
For k:=1 to m do
p^k_{ij}(t) olasılığına göre j şehri hareket etmek için seç
{k. Karınca t zamanında i=**tabu**$_k$(s-1) şehrinde}
k. karıncayı j şehrine hareket ettir
j şehrini **tabu**$_k$(s) ya ekle

4. Tur sonunda tur uzunluğunu ölçme ve en iyi tur değerini yenileme
For k:=1 to m do
k. karıncanın indeksini **tabu**$_k$(n) dan **tabu**$_k$(1) e getir
L_k (tur uzunlupunu) her k. Karına için hesapla
En iyi tur değerini bul ve yenile
Her edge(i,j) yolu için {--- Buharlaştırmayla birlikte feromen izi bırakma ---}
For k:=1 to m do

$$\Delta\tau^{k}_{i,j} = \begin{cases} \dfrac{Q}{L_k} & \text{if (i, j) } \in \text{ tour described by } \mathbf{tabu_k} \\ 0 & \text{otherwise} \end{cases}$$

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

Her edge (i,j) için τij(t+n)=ρ.τij(t)+Δτij eşitliğine göre τij(t+n) i bul
t:=t+n , NC:=NC+1
Her edge (i,j) için Δij:=0

5. Sonlandırma
Eğer (NC < NCMAX) ise{NC_{MAX} maximum tur sayısı}
Tüm karıncaların Tabu listelerini boşalt
2. basamağa git
değilse
En kısa turu bastır

Bitiş

Şekil 2. Ant System Algoritması

Son olarak bu konuda yapılan araştırmalar ve testler sonucunda şunu söyleyebiliriz ki;

- Algoritma $O(t.n^2.m)$ (eğer n = m ise $O(t.n^3)$) zamanda çözüm yapar.($t = NC_{MAX}$).
- Problemin doğru çözülmesinde parametre ayarlarının güzel yapılması etkilidir. Yanlış verilen kararlarla çalışan algoritma iyi sonuçlar vermez. Parametre değerlerinin seçim kuralları ise yapılan bir çok deney sonucunda anlaşılmıştır.
- Bu algoritma yalnızca küçük ölçekli TSP'lerde (75 şehirden az) optimal sonuçlar üretirken daha büyük ölçeklilerde optimal sonuçlardan uzaklaşılır. Bu ise bu algoritma üzerinde düşünülüp yeni stratejilerin bulunmasına (Örneğin Elitist Strategy) olanak tanımıştır.

4. EĞİTİMDE GÖRSELLEŞTİRME VE ALGORİTMA SİMÜLASYONU

Bir çok öğrenci, bazı algoritmaların anlaşılması ve analiz edilmesinin zor olduğunu söylerken eğitmenler de yine bazı algoritmaların anlaşılmasının zorluğundan bahsederler. Bunun en büyük sebeplerinden birisi de algoritmaların soyut verilerden oluşması ve karmaşık bir yapıya sahip olmalarıdır. Algoritma simülasyonları ve görselleştirilmesi, algoritmaları grafiksel olarak göstererek, öğrencilerin ve eğitmenlerin bu problemine çözüm bulmak için geliştirilen yazılımlardır.[5]

Kara tahtaya yapılan ve kitaplardaki grafiksel gösterimler ve anlatımlar da öğrenmeyi sağlayabilir ama dinamik ve sembolik resimlerle ifade edilmiş bir görselleştirme, algoritmanın mantığının anlaşılmasında daha etkili olacaktır. Fakat bu bir gerçek olmasına rağmen bu tip görselleştirme uygulamaları çok yaygınlık kazanmamıştır. Bunun sebepleri ise eğitmenlerin;

- Bu uygulamaları öğrenmek için yeterli zamanlarının olmaması
- Bunların diğer sınıf aktivitelerinin zamanından çalacağı düşünmeleri
- Bu tip uygulamaların çok zaman alıcı ve güç olmasından dolayı çekinmeleri
- Bu uygulamaları eğitimsel olarak faydalı bulmamaları olarak söylenebilir.[5]

Bu dört sebep bir algoritma simülasyonu yaparken bir köşeye bırakılamaz. Çünkü yapılan test ve deneyler ne kadar da bu gösterme ve simülasyonların öğrenmede etkili olduğunu gösterse de [5][6][7] var oluş amaçlarına (öğrenmeye yardımcılık) ulaşamayan simülasyonların gerçekleştirilmesi, hem zaman kaybı hem de hüsranla sonuçlanacaktır. Bu yüzden simülasyonu tasarlarken onu etkin yapacak etkenler iyice düşünülüp ona göre karar verilmesi gereklidir.

Algoritma simülasyon ve görselleştirimlerinin bir başka kullanım sebebi ise konuya ilgiyi arttırmak ve konuyu daha zevkli bir hale getirmektir. Özellikle yukarıda bahsedilen Karınca Kolonisi Optimizasyonu algoritmaları gibi yeni gelişmekte ve kullanımı gittikçe yaygınlaşmakta olan bu tip algoritmalara dikkat çekebilmek ve bunlara ilgiyi arttırmak için simülasyonlar gerçekleştirmek algoritmanın öğreniminden çok daha farklı kazanımları sağlayabilir.

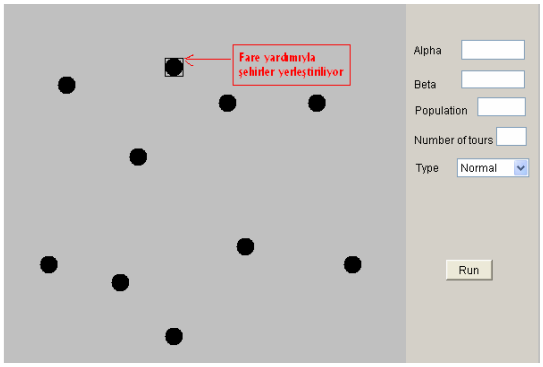
5. UYGULAMA

Uygulamamızda ACO algoritmalarının önemini ve çıkış noktasını göz önüne alarak ACO algoritmalarının ilki olan Ant Sistem algoritmasının uygulaması ve görselleştirmesi gerçekleştirilmiştir. İnternet üzerinden de işletilebilen bu Ant System Algoritması simülasyonu yoluyla;

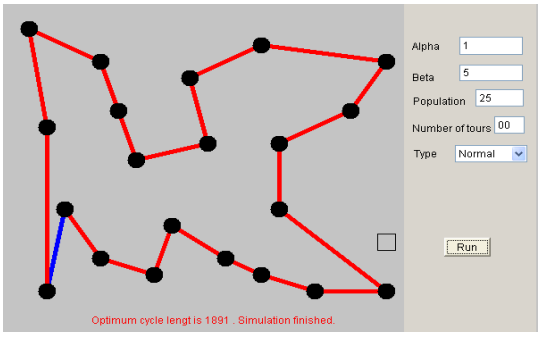
- Ant Colony Optimization algoritmalarını öğrenciye daha iyi tanıtmak, öğretmek ve bu konuya olan ilgilerini arttırmak
- Ant System algoritması etkileyen parametrelerin daha iyi anlaşılmasını sağlamak

amaçlanmıştır. Daha önce de bahsettiğimiz gibi eğer etkin bir simülasyon yapmak istiyorsak kararlarımızı amaca uygun vermek zorundayız. Bu yüzden bu simülasyonu geliştirirken amacımıza uygun olacak şekilde bazı kriterleri gerçekleştirdik. Bunlar;

- Ant System algoritmasının doğasına uygun olacak şekilde simülasyonu gerçekleştirmek
- Diğer ACO algoritmaları yerine Ant System algoritmasını ve problem olarak TSP yi seçmek. Bu ACO algoritmalarının çıkış noktasını anlatabilmek için önemlidir.
- Şekil 3 ve şekil 4'de görüldüğü gibi kullanıcıların şehirleri istedikleri gibi şehir eklemelerine izin vererek farklı koşullarda çalışan algoritmanın çalışmasını göz ile analiz edebilmeleri sağlamak

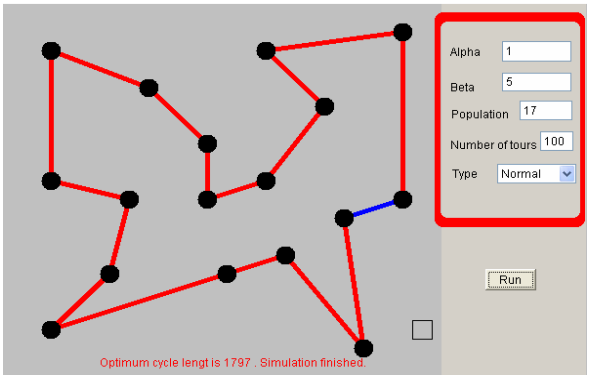


Şekil 3. Simülasyonda Şehir Ekleme



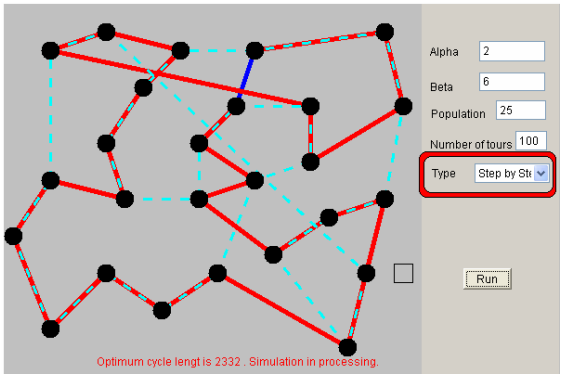
Şekil 4. Simülasyonda Çözümü Bulma

- Şekil 5'te de gösterildiği gibi algoritmada kritik rol oynayan parametreleri değiştirme olanağı sağlanarak parametre etkilerinin daha kolay anlaşılmasını sağlamak. Çünkü görsel olmayan çözümlerde sadece sonuçlar incelenebildiğinden bu etkiler fazla anlaşılamayabilmektedir.



Şekil 5. Simülasyonda Parametreler ve Sonuca Etkisi

- Şekil 6'da gösterildiği gibi adım adım işletilmesine olanak tanınarak sanal karıncaların hangi aşamalardan geçerek sonuca ulaştıklarını göstermeyi sağlamak.



Şekil 6. Simülasyonun Adım Adım Çalışması (Kırmızı yollar o anki en iyi turu, açık mavi kesikli çizgili yollar ise o anki turunu tamamlayan karıncanın turunu göstermekte)

- İnternet üzerinden de kullanımı sağlayabilmek. Bu yüzden Java dili ve görselleştirmede Java2D API kullanılmıştır. Birincil amacımız problem çözmek değil de görselleştirmek olduğundan yazılım geliştirme ortamının performans özellikleri dikkate alınmamıştır. Bu yüzden C/C++ yerine Java programlama dili tercih edilmiştir. Bu dilin doğrudan İnternet tabanlı olması ve Java2D gibi hazır API lere sahip olması gibi avantajlarından da yararlanılmıştır.

6. SONUÇLAR

Sürü zekası uygulamaları yapay zeka ve yapay yaşam kapsamı altındaki multi-agent sistem yaklaşımları olup optimizasyon problemleri, robotbilim, telekomünikasyon, veri madenciliği gibi bir çok konuda başarı sonuçlar veren uygulamalardır. Bunlardan biri olan Karınca Kolonisi Optimizasyonu Algoritmaları da zamanla geliştikçe yapay zekanın önemli bir alanı olan TSP gibi NP-Hard problemlerin çözümünde genetik algoritmalar gibi rakiplerine karşı üstünlüğünü göstererek önemini daha da arttırmaktadır. Fakat bu konunun bilgisayar bilimlerinden başka biyoloji ile ilintili olması ve tamamen formülizasyona dayalı ve dağınık bir yapıda olan algoritmalardan oluşması konuya adaptasyonu ve öğrenmeyi güçleştirmektedir. Algoritma simülasyonları ise öğrenme sürecini kısaltan uygulamalardır. Geliştirdiğimiz uygulamada bu kriterler göz önüne alınarak bir simülasyon gerçekleştirilmiştir. Bu yöntemlerin çıkış noktası olan Ant System algoritmasını görselleştirmek yoluyla kullanıcıların konuya hızlı uyum sağlaması hedeflenmiştir. Kritik rol oynayan parametrelerin ve karınca davranışlarının etkileri grafiksel olarak görselleştirilerek daha iyi verim alınmıştır. Karmaşık görünmesi nedeniyle beklenenden az kişinin çalışma ve araştırmalar yapmakta olduğu bu alan grafikler yardımı ile eğlenceli hale getirilmiştir.

KAYNAKLAR

[1] Bonabeau, E., Theraulaz, G., 'Swarm Smarts', Scientific American Inc., March 2000, pp. 72-79.

[2] I Osman and J Kelly. Meta-Heuristics: Theory & Applications. Kluwer Academic Publishers, Boston, 1996.

[3] Marco Dorigo, Thomas Sttzle, Ant Colony Optimization. Bradford Books. July 1st 2004.

[4] Bonabeau, E., Dorigo, M., Theraulaz, G., 'Swarm Intelligence: From Natural to Artificial System', Oxford University Pres, Oxford, 1999.

[5] Hundhausen, C.D., Dougles, S.A., Stasko J. T., 'A Meta Study of Algorithm Visualization Effectiveness', Journal of Visual Languages and Computing, no.13, 2002, pp. 259-290.

[6] Byrne, M.D., Catrambone, R., Stasko J. T., 'Do Animation Aid Learning?', Technical Report GIT-GUV-96-18, August 1996.

[7] Kehoe, C., Stasko J. T., Taylor, A., 'Rethinking the Evaluation of Algorithm Animation as Learning Aids: An Observational Study ', Technical Report GIT-GUV-99-10, March 1999.