Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi
Pamukkale University Journal of Engineering Sciences

ÇOK ETMENLİ SİSTEMLERDE NETLOGO İLE KARINCA KOLONİSİ OPTİMİZASYONU

ANT COLONY OPTIMIZATION IN MULTI-AGENT SYSTEMS WITH NETLOGO

Mustafa TUKER[1], Serkan BALLI[2], İzzet PEMBECİ[3]*

[1]Elektronik Bilgisayar Eğitimi Bölümü, Teknik Eğitim Fakültesi, Muğla Sıtkı Koçman Üniversitesi, 48187, Muğla.
tukermustafa@gmail.com
[2]Bilişim Sistemleri Mühendisliği Bölümü, Teknoloji Fakültesi, Muğla Sıtkı Koçman Üniversitesi, 48187, Muğla.
serkan@mu.edu.tr
[3]Bilgisayar Mühendisliği Bölümü, Mühendislik Fakültesi, Muğla Sıtkı Koçman Üniversitesi, 48187, Muğla.
pembeci@mu.edu.tr

Geliş Tarihi/Received: 14.06.2012, Kabul Tarihi/Accepted: 02.08.2012 doi: 10.5505/pajes.2013.32032
*Yazışılan yazar/Corresponding author

Özet

Çok etmenli sistemler (ÇES), karmaşık optimizasyon problemlerinin modellenmesi ve çözülmesi için etkin bir yol sunarlar. Bu çalışmada, Gezgin Satıcı Problemi (GSP)'ni çözmek için ÇES ve karınca kolonileri birlikte kullanılmıştır. Sistem benzetimi, etmen tabanlı bir programlama ortamı olan NetLogo ile gerçekleştirilmiştir. Problemin modellenmesi ve benzetimi için NetLogo'nun nasıl kullanılacağı kodlarla ayrıntılı olarak açıklanmıştır. Algoritma farklı düğüm sayıları için denenmiş ve elde edilen sonuçlar tartışılmıştır.

Anahtar kelimeler: Çok etmenli sistemler, Karınca kolonisi, NetLogo, Gezgin satıcı problemi.

Abstract

Multi-agent systems (MAS) offer an effective way to model and solve complex optimization problems. In this study, MAS and ant colonies have been used together to solve the Travelling Salesmen Problem (TSP). System simulation has been realized with NetLogo which is an agent-based programming environment. It has been explained in detail with code examples that how to use NetLogo for modeling and simulation of the problem. Algorithm has been tested for different numbers of nodes and obtained results have been discussed.

Keywords: Multi-Agent systems, Ant colony, NetLogo, Traveling salesman problem.

1 Giriş

Yapay zekada genel anlamda bir etmen; algılayıcıları ile içinde bulunduğu ortamı algılayan ve etkileyicileri ile bu ortam üzerinde eylemlerde bulunan bir sistemdir [1]. Otonom davranışlar sergileyen etmenler, kendi bilgi ve bireysel yeteneklerini kullanarak kullanıcıların çözemedikleri veya etkin bir biçimde çözemeyeceklerini düşündükleri problemleri çözmek amacıyla bir araya gelerek Çok Etmenli Sistemler adı verilen sistemleri oluşturmaktadırlar [2]-[3].

Bilgisayar ortamında çok etmenli sistemleri gerçekleştirebilmek için çeşitli benzetim ortamları geliştirilmiştir. Bunlardan biri olan NetLogo doğal ve sosyal bilimlerle ilgili kavramları, karmaşık sistemleri modellemek için geliştirilmiş etmen tabanlı (agent-oriented) bir programlama dili ve bütünleşik simülasyon ortamıdır [4]. 1999 yılında Uri Wilensky tarafından tasarlanmış ve geliştirilmiştir. Halen Northwestern Universitesi bünyesindeki (The Center for Connected Learning and Computer-Based Modeling) araştırma grubu tarafından geliştirilmeye devam etmektedir. NetLogo çok etmenli sistemlerin modellenmesi için oldukça kullanışlı, öğrenimi kolay ve pratik teknikler sağlar. Dünya çapında çok sayıda öğretmen, öğrenci ve araştırmacı tarafından kullanılmaktadır.

Çalışmada ele alınan GSP, aralarındaki uzaklıklar bilinen N adet noktanın (şehir, parça veya düğüm gibi) her birisinden yalnız bir kez geçen en kısa veya en az maliyetli yolun bulunmasını hedefleyen bir problemdir. Bu problemin çözümü için kullanılan sezgisel yöntemlerin en önemlileri genetik algoritmalar, benzetimli tavlama ve karınca kolonisi tabanlı algoritmalardır [5]. Tüm sezgisel algoritmaların kendilerine göre avantajları ve dezavantajları bulunmaktadır. Çalışmada kullanılan Netlogo programı Çok Etmenli Sistemlerin tasarlanması için geliştirilmiş bir yazılım ortamıdır. Her bir karınca bir etmen olarak ele alınırsa, karınca kolonisi, çok etmenli sistemlerin ifade edilmesi için uygun bir algoritmadır. Bu yüzden etmen kavramının daha iyi anlaşılması ve NetLogo ile etkin bir şekilde kullanılabilmesi açısından problemin çözümü için karınca kolonisi optimizasyonu algoritması üzerinde durulmuştur.

Karınca kolonisi optimizasyonu "Sürü zekası" (Swarm Intelligence) kavramı'na dayanan bir algoritmadır. Sürü zekası, özerk yapıdaki basit bireyler grubunun kolektif bir zeka geliştirmesidir [6]. Sürü zekası, sürüdeki bireylerin kendi başlarına yaptıkları eylemler sonucu ortaya çıkan etkiler aracılığıyla iletişim kurmaları, kendi aralarında hareketlerini düzenleme durumu ve kendiliğinden düzen (self organization) denen iki ana mekanizmadan oluşur. Bu iletişim, bireylerin yaptıkları işlerin ortamda meydana getirdiği değişiklikler sonucu sağlanır. Bu sayede, önceden yapılmış ve merkezi bir plan olmadan kendiliğinden düzen sayesinde sonuç üretebilmeleri ve merkezi yönetim birimi olmadan esnek ve sağlam bir şekilde yapılanmaları sağlanır. Sürülerin bu özellikleri ve gerçekleştirdikleri eylemler, kolektif olarak çözdükleri problemler (Yiyecek bulma, taşımada yardımlaşma, kolektif kümelenme gibi) klasik yapay zeka yöntemlerine yeni bir soluk getirmiştir. Klasik yapay zeka yöntemlerinde bulunan insan zekasını modellemeye odaklı, karmaşık, merkezî, planlı yaklaşımların aksine, sürü zekası basit yapılı, özerk, önceden planlama yapmayan dağınık ajanların karmaşık problemlerin çözümünde başarılı olduklarını göstermiştir [7].

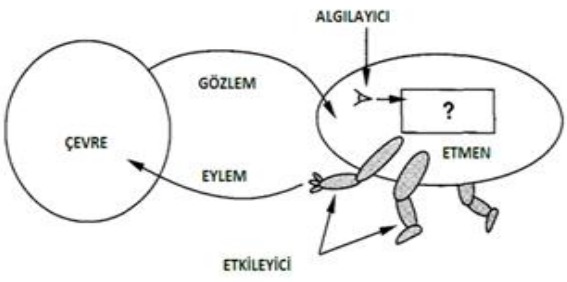
Netlogo ile gerçekleştirilen bu çalışmaya benzer olarak Johnson [13], sürü zekası kavramını NetLogo üzerinde hayata geçirmiştir. Bu çalışmada karınca sürüsünün yuva ile besin kaynağı arasında en kısa yolu bulmaları ele alınmıştır. Besin kaynakları ile yuva arasına çeşitli engeller yerleştirilmesi ve feromen buharlaşma oranlarının değiştirilmesi durumunda karınca kolonisinin hareketi araştırılmıştır. Washington ve Letendre [14], koloni davranışının daha net bir şekilde anlaşılması için koloni iki parçaya ayırmıştır. Aynı problemin çözümünü sağlamak için ilk koloni kullanıcı tarafından parametreleri değiştirilerek yönlendirilebiliyorken, ikinci koloni genetik algoritma ile yönlendirilmektedir. Roach [15] ise, gezgin satıcı problemini, çeşitli sayıdaki düğümler için çözümlenmesini Ant System algoritmasından yararlanarak gerçekleştirmiştir.

Bu çalışmada NetLogo benzetim ortamı üzerinde, kullanıcıların belirleyebileceği sayıda düğüm ve farklı düğüm bağlantılarından oluşan Gezgin Satıcı Probleminde optimum tur güzergahı ve mesafesini bulmalarını sağlayan etkileşimli ve grafik destekli bir yazılım gerçekleştirilmiştir. En iyileme yöntemi olarak karınca kolonisi optimizasyonu algoritması kullanılmıştır.

Çalışmanın ikinci bölümünde çok etmenli sistemlerin özellikleri, üçüncü bölümde çok etmenli sistemlerin benzetimi için çok kullanışlı bir yazılım ortamı olan NetLogo, dördüncü bölümde çok etmenli bir sistem olarak modelleyecek olduğumuz karınca kolonilerinin özellikleri, beşinci bölümde uygulama ve altıncı bölümde ise elde edilen sonuçlardan bahsedilmektedir.

2 Yazılım Etmenleri ve Çok Etmenli Sistemler

Yazılım etmenleri (“software agents”) ve bunların oluşturduğu Çok-etmenli Sistemler (“Multi-agent Systems”), karmaşık yapıdaki dağıtık sistemlerin modellenmesini ve oluşturulmasını sağlayan etkili birer teknoloji olarak ortaya çıkmışlardır. Şekil 1’de Etmenlerin algılayıcıları ve etkileyicileri yardımıyla çevreleriyle etkileşimde bulunması gösterilmiştir [1]. Yani bir etmen algılayıcıları ile ortamdaki gelişmeleri takip eder ve algıladığı veriler doğrultusunda etkileyicileri aracılığı ile eylemlerde bulunarak çevresinde değişikliklere sebep olabilir. ÇES bünyesinde; etmenlerin birbirleri ile olan etkileşimleri bencil veya işbirlikçi bir yapıda olabilir [8].



Şekil 1: Bir etmenin yapısı.

Başka bir deyişle etmenler ortak bir amacı paylaşabilir ya da kendi çıkarlarının takipçisi olabilirler [2]-[8]. ÇES ortamında her etmenin problem çözmek için tam olmayan bilgi ve yeteneğinin olması ve böylece daha sınırlı bir bakış açısının olması gerekir. Aynı zamanda sistem tek bir yerden kontrol edilmez bu yüzden verinin merkezde olmasına ihtiyaç duyulmaz. Bütün işlemler asenkron olarak gerçekleştirilir [9].

Etmen tabanlı yazılım geliştirmeye has karakteristikler ve zorluklar güncel yazılım mühendisliği metodolojilerinin sınırlarını zorladığından dolayı araştırmacılar aynı zamanda etmen tabanlı yazılım geliştirme için yeni yazılım metodolojileri ve araçları da önermektedirler [10]. Günümüzde ÇES’lerin geliştirilmesine katkıda bulunacak çeşitli iletişim dilleri, etkileşim protokolleri ve etmen mimarileri geliştirilmektedir. NetLogo, JADE, Jadex, FLUX, JACK, 3APL, Jason benzeri yazılım araçları etmen tabanlı dillere örnek olarak gösterilebilir. Çalışmada kullanılan NetLogo yazılımı bir sonraki bölümde ele alınacaktır.

3 Netlogo

NetLogo ÇES’e dayalı problemleri modellemek için geliştirilmiş bir programlama dili ve bütünleşik simülasyon ortamıdır. NetLogo dilinde komutlar konuşma diline çok yakın olduğu için kullanıcılar tarafından öğrenilmesi kolaydır. NetLogo özellikle zaman içerisinde gelişen karmaşık sistemleri modelleme için uygundur. Kullanıcılar birbirinden bağımsız çok sayıdaki etmene çeşitli komutlar verebilir. Böylelikle etmenler arasındaki mikro düzeyde davranışların ve onların etkileşiminden ortaya çıkan makro düzeydeki sonuçların kullanıcılar tarafından keşfedilmesini sağlayan bir benzetim ortamı ortaya çıkar. NetLogo benzetim ortamı Şekil 2’de verilmiştir. Şekil 2’de görüldüğü gibi program penceremizde etmenlerin üzerinde hareket ettikleri bir alan bulunmaktadır. Bu alan NetLogo’da ‘‘world’’ yani dünya olarak adlandırılmaktadır. Etmenlerin bu hareket alanı hücre (patch) adı verilen küçük parçalara ayrılmış bir ızgara (grid) veya matriks olarak düşünülebilir. Etmenlerin hareket alanının varsayılan boyutu 43*43 hücredir. Alanın boyutu kullanıcı tarafından değiştirilebilmektedir. Netlogo’da dünyayı oluşturan hücreler de etmen olarak varsayılmıştır. Dolayısıyla kullanıcı çeşitli özellikler (renk, etiket, enerji, yiyecek miktarı v.b) eklenebilir, hücrelere komutlar gönderebilir. Tüm etmenler bu alan üzerinde oluşturulup, hareket ettirilirler. NetLogo’da etmenler, Logo dilinden gelen bir gelenekle, “turtle” olarak adlandırılmaktadırlar. Etmenler çok basit komutlarla oluşturulabilir, özellikler eklenebilir ve istenen şekilde hareket ettirilebilirler.

Etmenler arasında fiziksel bir bağlantı kurmak veya belli bir amaçla ilişkilendirmek için bir çok parametreye sahip bağlantı (link) adı verilen bir diğer etmen sınıfı da mevcuttur. NetLogo’da Gözlemci (Observer) adı verilen yetkili kullanıcı kipi ile program adım adım çalıştırılabilir ve yönetilebilir. Ayrıca, Netlogo modeldeki parametrelerin dışarıdan değiştirilmesini, etmenlerin simülasyon boyunca özelliklerinin takip edilebilmesini, çeşitli istatistiki değerlerin grafiğinin çizilebilmesini sağlayan bir grafiksel kullanıcı arabirimine sahiptir. Bu tip arabirim araçlarının simülasyon ortamına eklenmesi hem çok kolaydır, hem de kullanıcı tarafından anlaşılmasında ve etkileşim sağlanmasında çok büyük rolleri vardır.

Bir eğitim aracı olarak NetLogo öğrencilerin simülasyonları çalıştırıp, çeşitli koşullar altındaki davranışlarını keşfetmelerini ve ayrıca öğretmen, öğrenci ve model geliştiricileri için kendi modellerini oluşturmalarını sağlar.

NetLogo aynı zamanda öğretmenler için yeterince basit bir kullanıma sahipken bilimsel anlamda araştırmacılar için de karmaşık sistemleri modellemek için güçlü bir araç olarak hizmet verecek bir esnekliğe ve kullanım yelpazesine sahiptir.

NetLogo kapsamlı bir dokümantasyon ve kılavuzlara sahiptir. Ayrıca önceden yazılmış olan, kullanılabilir ve değiştirilebilir bir model kütüphanesi de vardır. Bu modeller biyoloji, tıp,

fizik, kimya, matematik, bilgisayar bilimleri, ekonomi ve sosyal psikoloji gibi çeşitli doğal ve sosyal bilimlerin ilgi alanlarından seçilmiştir. Birçok model tabanlı eğitim müfredatları mevcuttur ve daha fazlası geliştirilme aşamasındadır.

NetLogo, çok kullanıcılı ÇES'e destek veren ve HubNet olarak adlandırılan bir eklentiyi de içermektedir. Bu sayede ağa bağlı bilgisayarlar ya da taşınabilir cihazların kullanımı sayesinde her kullanıcı bir etmeni kontrol edebilir, modelle kısmi olarak etkileşime geçebilir.

NetLogo ücretsiz olup, Logo'yla başlayıp StarLogo'yla devam eden çok etmenli modelleme dilleri serisinin en yeni üyesidir. NetLogo, Java tabanlı olduğu için tüm platformlarda (Mac, Windows, Linux, v.b) bağımsız bir uygulama olarak çalışabilir. Ayrıca modeller ve HubNet faaliyetleri bir web tarayıcısında Java uygulamaları olarak çalıştırılabilir. Netlogo modellerini komut satırından çalıştırma işlemi de desteklenmektedir

4 Karınca Kolonileri

İnsanlar doğadaki hayvan sürülerinin davranışlarını uzun zaman önce keşfetmişlerdir. Kuş sürülerinin farklı şekiller alarak uçmaları, karıncaların yuvalarına yiyecek taşımaları, arıların kovanları etrafında en zengin ve kaliteli nektar içeriğine sahip olan çiçek bölgelerini araştırmaları bu sürü davranışlarına verilebilecek örneklerdir. Yıllardan bu yana biyologlar ve bilgisayar uzmanları birlikte çalışmalar yaparak "Yapay Yaşam" alanı adı altında bu sürülerin davranışlarının matematiksel olarak modellenmesi ve sürüdeki bireyler arasındaki iletişimin nasıl gerçekleştiğinin belirlenmesi üzerine çalışmalar yapmaktadırlar.

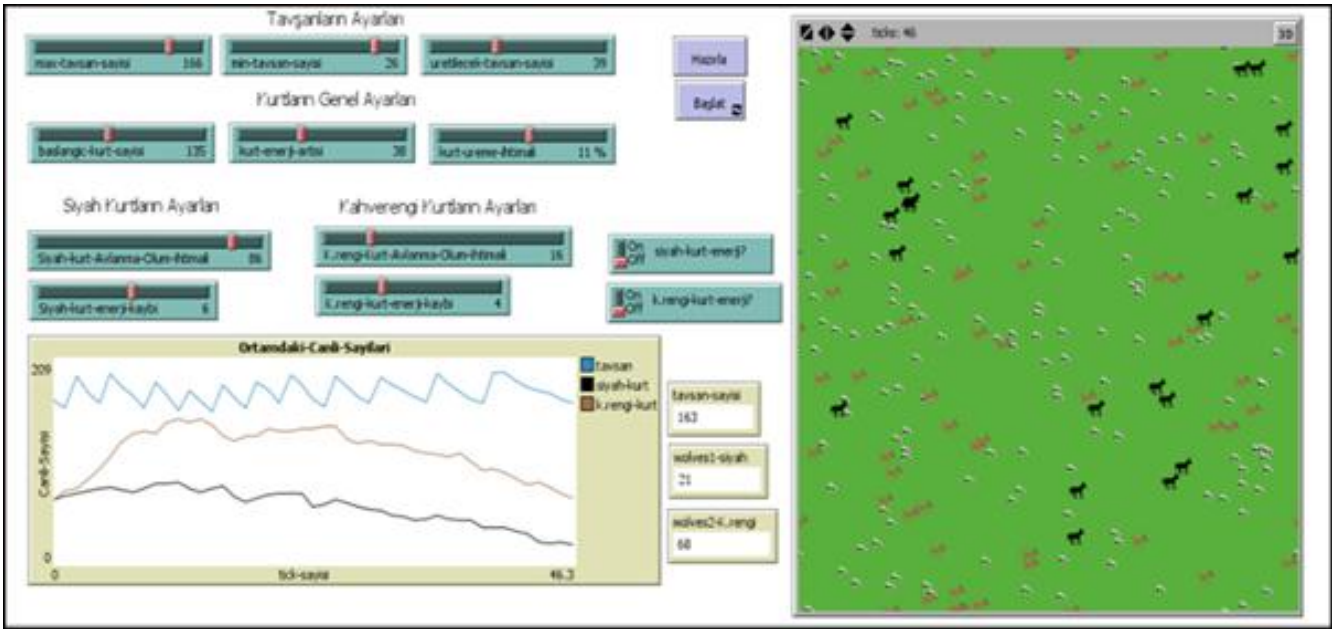
Biyolojik karıncaların doğadaki davranışları ve buna bağlı olarak geliştirilen yapay karıncalar alt bölümler halinde ele alınacaktır.

4.1 Biyolojik Karıncalar

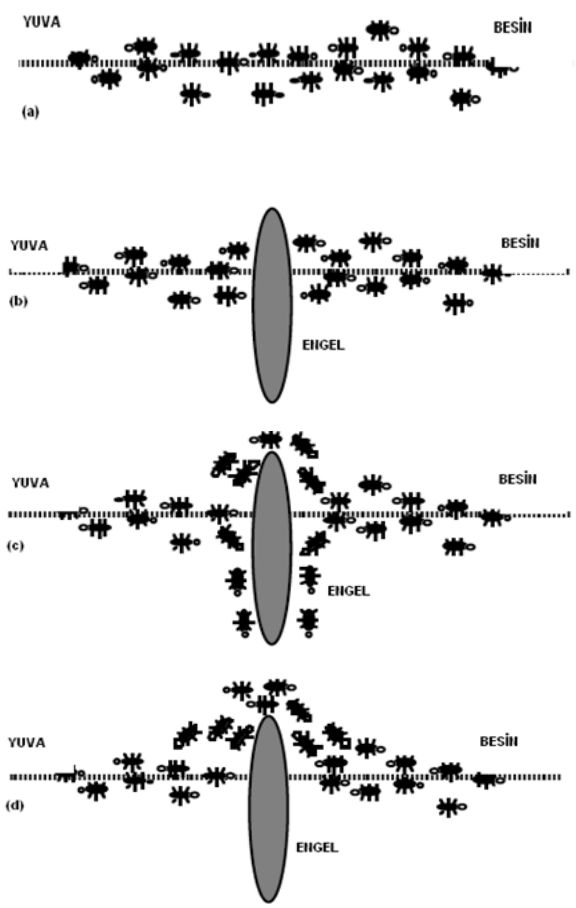
Doğada karıncalar, görsel ipuçları kullanmadan yiyecek kaynağından yuvaya en kısa yolu bulma yeteneğine sahiptir [11]. Ayrıca çevresel değişikliklere adapte olabilmektedirler. Örneğin yeni engeller nedeniyle uygulanabilir olmayan eski yol yerine yeni en kısa yolu bulma yeteneğine sahiptirler. Karıncaların besin ve yuva arasındaki hareketleri Şekil 3'te gösterilmiştir [12].

Şekil 3a'ya bakıldığında karıncalar yuvaları ile yiyecek kaynağı arasında düz bir çizgi halinde hareket ederler. Bu çizginin oluşturulması ve muhafaza edilmesi için karıncaların birincil yolunun feromen izi olduğu iyi bilinmektedir. Karıncalar yürürken belli miktarda feromeni bırakırlar ve her karınca feromence zengin bir yönü takip etmeyi olasılıklı bir şekilde tercih eder. Şekil 3b'de gerçek karıncaların bu temel davranışları aniden ortaya çıkan ve beklenmeyen bir engel tarafından ilk yol kesintiye uğradıktan sonra en kısa yolu bularak kırık hattı yeniden bağlamayı nasıl gerçekleştirdiklerini açıklar. Aslında engel ilk ortaya çıktığında engelin hemen önünde olan karıncalar feromen izini takip etmeye devam edemez ve bu yüzden sağa veya sola dönmek arasında bir seçim yapmak zorunda kalırlar. Bu durumda karıncaların yarısının sağa dönmeyi ve diğer yarısının sola dönmeyi seçtiği düşünülür. Şekil 3c'de çok benzer bir durum engelin diğer tarafında da bulunabilir. İlginçtir ki daha uzun yolu seçenlerle karşılaştırıldığında; şans eseri engelin etrafındaki daha kısa yolu seçen karıncaların kesintiye uğramış olan feromen izlerinin daha hızlı bir şekilde yeniden kurulduğu görülür. Böylece daha kısa olan yol her zaman biriminde daha büyük miktarda feromen alacaktır ve karıncaların büyük kısmı sırayla daha kısa olan yolu seçeceklerdir. Şekil 3d'de bu olumlu geribildirim süreci sayesinde tüm karıncalar hızla kısa yolu seçeceklerdir.

Bu olumlu geri bildirim sürecinin en ilginç yönü, haritanın şekli ve dağıtık karınca davranışlarının etkileşimi ile ortaya çıkan bir özellik gibi görünen engellerin etrafından dolaşarak en kısa yolu bulmaktır. Tüm karıncalar yaklaşık olarak aynı hızda hareket etmesine ve yaklaşık olarak aynı oranda feromen izi bırakmasına rağmen, şu bir gerçektir ki engellerin hatlarını belirlemek uzun taraf için kısa tarafa göre daha çok zaman alır. Bu da feromen izlerinin kısa tarafta daha hızlı oluşması demektir. Bu birikimi en kısa yolda hala daha çabuk yapan etken karıncaların daha yüksek feromen izi seviyesi olan yolları tercih etmeleridir [12].



Şekil 2: NetLogo benzetim ortamı.



Şekil 3: Karıncaların besin ve yuva arasındaki hareketleri.

4.2 Yapay Karıncalar

Yapay karıncalar, biyolojik karıncaların bilgisayar ortamında bir benzetimidir. ÇES ortamında yapay karınca, düğümler arasında hareket eden bir etmendir.

GSP için karınca gideceği düğümü, bulunduğu düğümden o düğüme giden yoldaki birikmiş feromen miktarı ve yol uzunluğunu hesaba katarak olasılıksal olarak seçer. Yapay karıncalar sonuçta daha yüksek olasılıkla uzunluğu az olan ve feromen miktarı fazla olan, yani daha önce başka karıncaların kullanılmış olduğu yolları seçerler. Başlangıçta yapay karıncalar rastgele seçilmiş düğümlere yerleştirilir. Her zaman aşamasında yeni düğümlere hareket ederler ve seçtikleri yoldaki feromen miktarını artırırlar. Bu yerel patika güncellenmesi olarak adlandırılır. Bütün karıncalar turlarını tamamladığında, en kısa turu tamamlayan karıncanın izlediği patikadaki yollardaki feromen miktarı ayrıyeten artırılır. Buna da global patika güncellemesi denilmektedir.

Yapay karınca kolonisi için doğal karınca davranışından transfer edilen üç fikir aşağıdaki gibidir:

- Yüksek feromene sahip yol tercihi,
- Daha kısa yollarda bulunan feromen miktarının yüksek artış oranı,
- Karıncalar arasında feromen izine bağlı iletişim,

Yapay karıncalara doğal karşılığı bulunmayan ancak GSP uygulamasına iyi uyacağı gözlemlenen birkaç özellik eklenmiştir. Yapay karıncalar şehir uzaklıklarını belirleyebilirler ve ziyaret edilmiş şehirlerin listesini tutan bir bellek ile donatılmışlardır. Bu bellek her turun başında sıfırlanır ve her zaman basamağında son ziyaret edilen şehir eklenerek güncellenir.

Şekil 3'teki prensipleri bilgisayar ortamında GSP çözmeye uygun bir sisteme dönüştürmek için farklı yollar vardır. Kullanılan algoritmada karıncalar sırayla çizge üzerindeki tüm şehirleri gezerek sonuca ulaşırlar. Her karıncanın bir şehirden başka bir şehre geçişinde olasılığa dayalı bir kural uygulanır. numaralı formülde bu kural ifade edilmiştir:

$$S=\begin{cases}\arg\max\left\{\left[\tau(r,u)\right].\left[\eta(r,u)\right]^{\beta}\right\} & \text{eğer } q\le q_0 \ \ ise\\ P_k(r,s) & \text{Diğer durumda}\end{cases} \quad (1)$$

Burada, öncelikle olasılığın belirleyici parametreleri olan q ve q_0 değerlerini açıklamak gerekir. $q_0$0 ile 1 arasında olan ve önceden belirlenen bir parametredir ($0\le q_0\le 1$).

Q ise her düğüm seçiminden önce program tarafından 0 ile 1 aralığında rastgele olarak üretilen bir değerdir (q, [0,1]). Bu iki değer karşılaştırılarak kullanılacak olan formül belirlenmektedir. Eğer $q \le q_0$ şartı sağlanırsa (1) numaralı formül çalışır, şart sağlanmaz ise (2) numaralı formüle göre seçilim yapılır.

Karınca Koloni Sistemimizde r şehrindeki k yapay karıncası (1) numaralı formüldeki olasılık eşitliğini uygulayarak belleğinde (Mk) yer almayan şehirler arasından s şehrine gitmeyi seçer.

Burada τ(r,u), r ve u şehirleri arasındaki yol üzerindeki feromen izinin miktarı iken, η(r,u) r ve u şehirleri arasındaki mesafenin tersi olarak seçilen sezgisel fonksiyondur.

β, iki şehir arasındaki yakınlığın ve yol üzerindeki feromenin olasılıksal seçimde göreceli önemlerini ayarlayan parametredir. β değeri artırılırsa iki şehir arası mesafe önem kazanırken, β değerinin azaltılması durumunda ise iki şehir arasındaki yol üzerinde bulunan feromen miktarı önem kazanmaktadır.

(2) numaralı formülde ifade edilen Pk(r,s), k karıncasının r şehrinden s şehrine hareket tercihinin olasılığını ifade eden formüldür.

$$P_k(r,s)=\begin{cases}\dfrac{\left[\tau(r,s)\right].\left[\eta(r,s)\right]^{\beta}}{\sum_{u\notin M_k}\left[\tau(r,u)\right].\left[\eta(r,u)\right]^{\beta}} & \text{eğer } s\notin M_k \text{ ise}\\ 0 & \text{Diğer durumda}\end{cases} \quad (2)$$

S, daha kısa olan ve daha yüksek düzeyde feromen izine sahip yola eğilim gösteren olasılık dağılımına göre seçilmiş rastgele bir değişkendir. Algoritmanın çalışması sırasında feromen izi yerel ve global güncellemelerle değişir. (3) numaralı formülde ifade edilen global güncelleme daha kısa turlara ait yolların öne çıkmasını sağlar. Global feromen izi güncelleme formülü ise şöyledir:

$$\varphi(r,s)\leftarrow(1-\alpha).\varphi(r.s)+\alpha.\Delta\varphi(r,s) \quad (3)$$

$$\Delta\varphi(r,s)=(\text{en kısa tur})^{-1} \quad (4)$$

Yapay karıncalar bir kez turlarını tamamlayınca, patikası en kısa olan en iyi karınca tarafından ziyaret edilen yollarda biriken feromen miktarı (4) numaralı formülde gösterilen Δ φ (r,s) miktarı kadar artırılır. Bu değer turun uzunluğu ile ters orantılıdır, yani tur kısaldıkça yolda biriken feromen miktarı artar. Bu feromen biriktirme tarzı gerçek karıncalardaki yol

uzunluğu ile süre arasındaki karşılıklı etkileşime bağlı olarak gerçekleşen feromen izi yığılım özelliğini model almayı amaçlamaktadır.

$$\tau(r,s) \leftarrow (1-\alpha).\tau(r,s)+\alpha.\tau_0 \quad (5)$$

(5) numaralı formülde ifade edilen yerel güncelleme ise çok güçlü bir yolun bütün karıncalar tarafından seçilmesini engellemeyi amaçlamaktadır. Bir karınca tarafından bir yolun seçilmesi durumunda o yol üzerindeki feromen miktarının güncellenmesi (5) numaralı formül kullanılarak gerçekleştirilir.

Burada kullanılan τ_0 değeri ise (6) numaralı formül kullanılarak hesaplanmaktadır.

$$\tau_0=(n.Lnn)^{-1} \quad (6)$$

(6) numaralı formülde kullanılan n değeri çizge üzerinde bulunan toplam şehir sayısını, Lnn değeri ise sistem tarafından en başta en yakın komşu sezgisel algoritmasına göre hesaplanan tur uzunluğunu ifade eder. Lokal iz güncellemesi gerçek karıncalarda feromen buharlaşması olarak ifade edilir. Feromen buharlaşması olmaksızın Karınca Koloni Sistemi güzel sonuçlar veremez. Çünkü şehirler arasındaki yollara en başta bırakılan feromenler tamamen rastgele gerçekleşen bir keşif sonucu oluşturulmuşlardır ve herhangi bir bilgi değerine sahip değildirler. Bu yüzden diğer karıncaların bu değerlerden etkilenmemeleri için yerel güncelleme sırasında feromen buharlaşması gerçekleştirilmelidir [7].

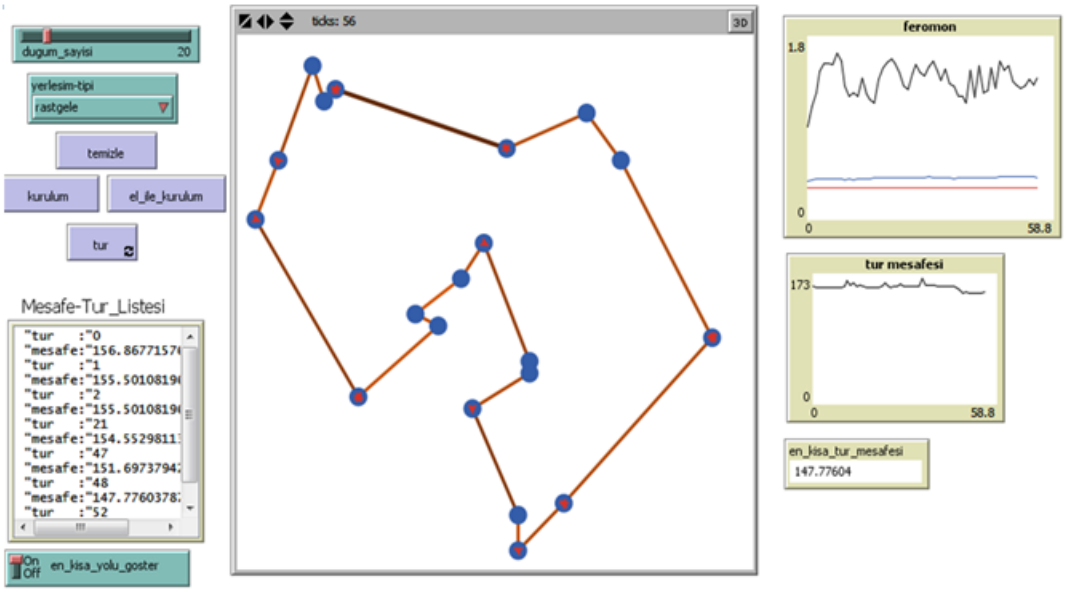
Yukarıdaki (1) ve (2) numaralı formüller göstermektedir ki bir karınca ya q0 olasılığı ile feromen izi miktarı karınca kolonisi tarafından yığınlanmış yolları seçecektir. Ya da (1-q0) olasılığı ile gidilecek şehri rastgele olarak feromen izi, deneme yanılma işlevi ve bellekten oluşan olasılık dağılımı ile seçerek rastgele bir keşif uygulayacaktır. Burada feromen izi kısa turlara ait yollarda büyüme eğilimindedir.

Sonuç olarak karınca kolonilerinin bu çalışmada kullanılan versiyonu olan ACS (Ant Colony System) algoritması şu şekilde bir keşif stratejisi kullanmaktadır. Öncelikle formül 1'in olasılıksal parçası S vardır. Burada yeni patikaların keşfi kısa ve yüksek feromen izine sahip yollara eğilimlidir. İkinci olarak yerel iz güncelleme, gidilen her patika kendine has lokal güncelleme formülü tarafından indirgenmiş feromen değerine sahip olduğu için keşif yapımını cesaretlendirici eğilimdedir [12].

5 Uygulama

Bu bölümde NetLogo simülasyon ortamında GSP'yi çözmeye çalışan ve yapay karıncalardan oluşan benzer bir sürecin nasıl işlediği gösterilecektir. Geliştirilen programda bir önceki bölümde anlatılmış olan ACS algoritması kullanılmıştır. Bu algoritmada karınca sayısı düğüm sayısına eşit tutulmuştur. Dorigo'nun algoritmasına göre karınca sayısının düğüm sayısına eşit olması ve karıncaların başlangıçta rastgele düğümlere dağıtılmaları uygundur. Ayrıca geliştirilen programda kullanılan diğer parametreler için de şu değerler kullanılmıştır: β=2, α=0.1, q0=0.9. Program çok sayıda prosedürden oluşmaktadır. Prosedürler ilgili yerlerde çağırılarak etmenlerin eylemlerini istenen şekilde gerçekleştirmeleri ve belirlenen parametreler doğrultusunda sonuç üretmeleri sağlanır.

Şekil 4'te görüldüğü gibi oluşturulan ara yüzde düğüm sayısı kullanıcı tarafından belirlenebilmektedir. Düğüm sayısı için kullanılan aralık 5-100 arasında tutulmuştur. Eğer istenirse bu aralık değiştirilebilir ve daha yüksek seviyelere taşınabilir. Düğümlerin program tarafından otomatik olarak ya da kullanıcı tarafından elle rastgele yerleştirilmesi sağlanmıştır. Kurulum düğmesi düğüm yerleştirmelerinin otomatik olarak gerçekleştirilmesi durumu için kullanılmaktadır. El ile kurulum düğmesi ise kullanıcıların istedikleri sayıda düğümü istedikleri şekilde çalışma alanına yerleştirmelerini sağlar. Düğümlerin yerleştirilmesiyle birlikte tüm düğümler arasında bağlantı kurulur. n adet düğüm için (n-1)! adet bağlantı otomatik olarak oluşturulmakta ve görüntülenmektedir. Bu işlem ara yüzdeki tüm düğümlerde gerçekleşmektedir.



Şekil 4: Programın ana menü görünümü.

1. Dairesel:

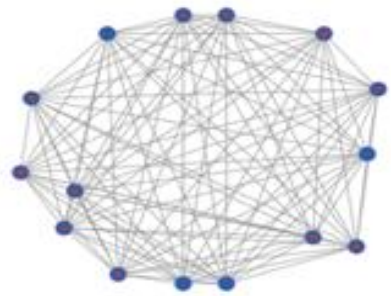
Tüm düğümler arası bağlantı kurulmasını sağlayan kod aşağıdaki gibidir:

```
Ask cities[create-links-with other cities]
```

Şekil 4'te görüldüğü gibi çalışma alanının sağ tarafına iki adet grafik eklenmiştir. Feromen isimli grafik programın çalışması sırasında tur sayısına bağlı olarak düğümler arasındaki bağlantılar üzerindeki feromen miktarlarının minimum, ortalama ve maksimum değerlerinin görüntülenmelerini sağlar. Tur mesafesi isimli grafik ise o turda bulunan en kısa patikayı göstermektedir. Şekil 4'te görüldüğü gibi düğüm sayısının altında bulunan yerleşim tipi adlı seçme menüsü düğümlerin program tarafından otomatik olarak yerleştirilmesinin istenmesi durumunda kullanıcıya seçenekler sunmaktadır. Bu seçenekler iki şekildedir:

Kullanıcı tarafından ara yüzde belirtilen sayıda düğümün çalışma alanı üzerinde yaklaşık dairesel bir şekilde dağıtılmasını sağlar.

Şekil 5'te görüldüğü gibi tam bir dairesel yapı olmaması için düğümlerin dağıtılmasında merkezden uzaklıklarının farklı olması ve aynı hücre üzerinde birden fazla sayıda düğüm olmaması sağlanmıştır.



Şekil 5: Düğümlerin dairesel yerleşimi.

Dairesel seçeneğinin uygulanmasını sağlayan kod bloğu aşağıdaki gibidir:

```
if yerlesim-tipi = "dairesel" [layout]
to layout
let center one-of cities
layout-radial cities links center
ask center [die]
ask cities[
facepatch 0 0
fd 2
]
repeat 3 [ask cities [move-to one-of neighbors with [not
any? cities-here] ]]
let x_max max [xcor] of cities
let x_min min [xcor] of cities
let y_max max [ycor] of cities
let y_min min [ycor] of cities
ask one-of patches with [pxcor<x_max - 2 and pxcor>x_min
+ 2 and
pycor<y_max - 2 and pycor>y_min + 2 and
abs pxcor> 2 and abs pycor> 2
]
[
sprout-cities 1 [
set color blue
set size 1.5
create-links-with other cities
]
```

2. Rastgele:

Kullanıcı tarafından ara yüzde belirtilen sayıda düğümün çalışma alanı üzerinde rastgele bir şekilde dağıtılmasını sağlar.

Şekil 6'da görüldüğü gibi düğümlerin rastgele yerleştirilmesi programda farklı yol alternatiflerinin daha fazla olacağı ortamlar oluşturulabilmesini sağlanmıştır. Bu seçeneğin uygulanmasını sağlayan kod bloğu aşağıdaki gibidir:

```
ask patches [set pcolor white]
set-default-shape cities "circle"
;;create a random network
ask n-of number-of-cities patches with
[not any? cities-here] [ sprout-cities 1 ]
ask cities[
set color blue
set size 1.5
create-links-with other cities]
```

Temizle düğmesi ile ara yüzdeki tüm çalışma alanı temizlenir ve tüm hücrelerin beyaz renk almaları sağlanarak arka planın beyaz görünmesi sağlanır. Kullanıcı uygun olan model şekli seçmek istediğinde istenmeyen modeli silmek için bu düğme kullanabilir ve yeni model parametrelerini belirleyip kurulum düğmesine basarak yeni bir model oluşturabilir. Temizle düğmesi kullanılmasıyla aşağıdaki prosedür çalıştırılmaktadır:

```
to temizle
ca
ask patches [set pcolor white]
end
```

Düğümler kullanıcı tarafından yerleştirilmek istendiğinde el_ile_kurulum düğmesi kullanılır. Belirlenen sayıdaki düğüm ara yüze kullanıcı tarafından fare ile yerleştirilir. Belirlenen sayıya göre son düğüm yerleştirildiğinde tüm düğümler arası bağlantılar daha önce de belirtildiği gibi otomatik olarak oluşturulur ve program çalıştırılmaya hazır hale gelir.

Düğümlerin konumlandırılma işleminin bitmesinin ardından tur düğmesi kullanılarak karınca kolonisinin harekete geçmesi sağlanır.

Şekil 7'de görüldüğü gibi Mesafe-Tur Listesi aracılığıyla programda her yeni kısa yol bulunduğunda bu yolun mesafesi ve kaçıncı turda bulunduğu ekranda görüntülenmektedir. Bunun için çıkış ekranı nesnesinden yararlanılmıştır. Bu nesne üzerine ilgili verileri yazdırmak için aşağıdaki kod bloğu kullanılmıştır:

```
ifelse best-tour-length<min-tour-length
[set min-tour-length best-tour-length
output-write "tur :" output-print ticks
output-write "mesafe:" output-print best-tour-length
set a0 1 ]
[set a0 a0 + 1 ]
update-links
update-plots best-tour-length
```

Algoritmada yapılan hesaplamalar için yuvarlama işlemi kullanılmamıştır. Bu yüzden program çok büyük bir hassasiyet

ile çalışmaktadır. Örneğin bulunan en iyi yolun 0.000001 oranında daha kısa olanı bulunduğunda bile bulunan yol yeni en iyi tur olarak kaydedilir ve program çalışmaya devam eder.

Şekil 6: Düğümlerin rastgele yerleştirilmesi.

Şekil 7: Mesafe-Tur listesi.

Şekil 8: En iyi tur mesafesinin görüntülenmesi.

Şekil 8'de görüldüğü gibi en iyi turun mesafesi ara yüzün sağ tarafında grafiklerin altında en kısa tur mesafesi isimli çıkış monitöründe görüntülenir.

Programda iki düğüm arası bağlantıyı link adı verilen etmenler sağlamaktadır. Bağlantıların feromon ve uzunluk değerleri vardır. Bu özellikler programın en başında aşağıdaki kodlar kullanılarak eklenmiştir:

```
links-own [ feromon inv-dist  ]
```

Uzunluk değeri bağlantı kurulduğu an hesaplanır ve düğümün yeri değişmediği için program boyunca sabit bir değerdir. Feromen ise her turda güncellendiği için değişkenlik göstermektedir. Feromen miktarının değişmesini görselleştirmek için bağlantıların kalınlıkları doğru orantılı bir şekilde değiştirilmektedir. Bu değişim için bağlantıların thickness isimli özelliğinden yararlanılmıştır.

Bağlantılarda feromen miktarı arttıkça bağlantının kalınlığı artar, feromen azaldıkça kalınlık azalır; Bu işlemin gerçekleşmesi için aşağıdaki kod bloğu kullanılmıştır:

```
to update-links
let feromons [feromon] of links
let max-feromon max feromons
let mean-feromon mean feromons
let min-feromon min feromons
let k-color 6 / (max-feromon - min-feromon)
let best-tour-edges [tour-linksvisited] of min-one-of ants [tour-
length tour-links visited]
ask links[
set color green + 3
set thickness 0
set label ""
setshape "default"
sethidden? true
]
ask links with [feromon > mean-feromon] [
set thickness 0.4 * feromon / max-feromon
set color green + 3 - (k-color * (feromon - min-feromon))
set hidden?  false
                                    ]
```

Programın çalışması sırasında alternatif tüm yollar bağlantılar aracılığı ile görüntülenmektedir. Yollar üzerlerindeki feromen miktarı ile doğru orantılı olarak kalınlaşıp incelerek en iyi yolun görüntüsü vurgulanmaktadır. Feromen miktarı ortalama feromen miktarından düşük olan yollar gösterilmeyerek görüntü sadeleştirilmektedir. Ara yüzdeki en kısa yolu göster düğmesi etkinleştirildiğinde programın tüm çalışması boyunca yalnızca en iyi yolu oluşturan bağlantılar görüntülenir ve diğer bağlantılar görüntülenmez. Bu işlemin gerçekleşmesi için aşağıdaki kod bloğu kullanılmıştır:

```
if en_kisa_yolu_goster [ask links [set hidden? true]]
ask best-tour-edges[
    set color color - 30 ; 30 = green - red
    set hidden? false
]
end
```

Düğüm sayısı ve yerleşim şekli belirlendikten sonra kurulum düğmesine bastığımız ilk anda algoritma tarafından bir yol hesaplaması yapılır. Bu hesaplamanın yapılması için fazladan bir karınca oluşturulup rastgele bir düğüme yerleştirilir. Bu karınca en yakın komşu sezgisel algoritması kullanılarak sadece kendine en yakın olan düğüme hareket ederek tüm düğümleri ziyaret eder ve turunu tamamlar. Turun tamamlanmasıyla birlikte fazladan oluşturulan bu karınca yok edilir. Bu işlemin gerçekleşmesi için aşağıdaki kod bloğu kullanılmıştır:

```
to tour-nearest
create-ants 1 [  ; create an extra ant to complete this tour
init-ant-values
]
let cnt 1
let some-ant one-of ants
while [cnt<dugum_sayisi] [
ask some-ant [
let next-city choose-next-city-heuristics
move-to next-city
set current-city next-city
]
set cnt cnt + 1
]
  let edges tour-links [visited] of  some-ant
  set min-tour-length tour-length edges
set f0 1 / (dugum_sayisi * min-tour-length)
ask some-ant  [die]
end
```

Bu aşamada bulunan patikanın uzunluğu tüm bağlantılara eklenecek ilk feromen değerini (τ_0) belirlemektedir. Bir önceki

bölümde yapay karıncaların kullandığı algoritmanın açıklamasında da belirtildiği gibi bağlantılar üzerindeki feromen miktarı tur uzunluğuna bağlı olarak belirlenmektedir. Bunun için aşağıdaki kod bloğu kullanılmaktadır:

```
tour-nearest ; to find  f0
ask links[
set  f0 1 / (dugum_sayisi * min-tour-length)
set  feromon  f0
set inv-dist 1 / (link-length * link-length)
set label-color black ]
end
```

Buraya kadar anlatılanlar kurulum düğmesine basılması sonucu elde edilen ilk değerlerin oluşturulması ve karınca kolonisinin tura başlamak için hazırlanmasıdır. Kullanıcı bu kısımda hiçbir değer görmemektedir.

Tur düğmesine basılması ile birlikte düğümler üzerine rastgele bir şekilde yerleştirilmiş olan karınca kolonisi harekete geçer. Bu hareket sırasında karıncalar tüm düğümleri gezerler ve bunu yaparken patika belleklerinden yararlanırlar. Bu işlemin yapılması için aşağıdaki kod bloğu kullanılmıştır:

```
to prepare-for-tour
set current-city one-of cities with [count ants-here <= 1]
move-to current-city
set visited (list current-city)
set candidates remove current-city [self] of cities
face one-of candidates
end
```

Karıncaların düğümler arasında gezerken bir sonraki düğümü belirlemek için kullanacakları olasılıksal formül yapay karıncalar bölümünde anlatılmıştı. Bu olasılık hesabının yapılması ve bir sonraki düğümün seçilebilmesi için aşağıdaki kod bloğu kullanılmıştır:

```
to-report choose-next-city
let city1 current-city
ifelse random-float 1 <= q0
[report max-one-of (turtle-set candidates)
                [ [feromon * inv-dist] of  link-with city1] ]
[let values [[feromon * inv-dist] of link-with city1] of (turtle-
set candidates)
let i 0
let cum-prob item 0 values
let p random-float sum values
while [p > cum-prob] [
set i i + 1
set cum-prob cum-prob + item i values
]
repor t item i candidates] end
```

Karınca kolonisinin hareketi sırasında sistemin doğru bir şekilde işlemesi için yapılması gereken en önemli işlemlerden biri olan yerel ve global feromen güncellemeleri yapılır. Yerel feromen güncellemesi aşağıdaki kod bloğunda gösterildiği üzere karıncalar bir düğümden bir başka düğümü hareket ettiklerinde yapılmaktadır:

```
to step
if  empty? [candidates] of one-of ants
     [user-message "Tour completed" stop]
ask ants[
let next-city choose-next-city
move-to next-city
    ; local trail update
ask [link-with next-city] of current-city[
set feromon  (1 - alpha) * feromon +  alpha * f0 ]
set candidates remove next-city candidates
set visited fput next-city visited
set current-city next-city
]
set tour-count  tour-count + 1
end
```

Global feromen güncellemesi için ise aşağıdaki şekilde gerçekleştirilmektedir:

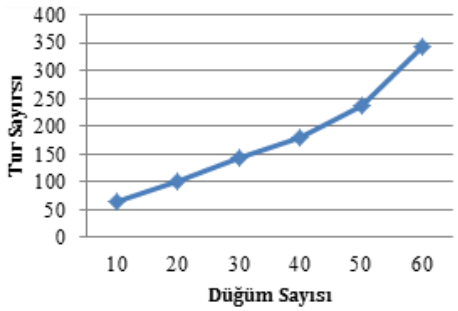
```
; global trailupdate based on best tour
ask best-tour-edges [set feromon (1-alpha) * feromon+ alpha /
best-tour-length]
```

Program her düğüm sayısı için on kez çalıştırılmış ve çıkan sonuçların ortalaması alınmıştır. Düğüm sayısının artması ile birlikte tur sayısı ve sürenin değişimine ait değerler ve bu değerlerin standart sapmaları Tablo 1'de gösterilmiştir. Buna ait grafik de Şekil 9 ve Şekil 10'da verilmiştir.

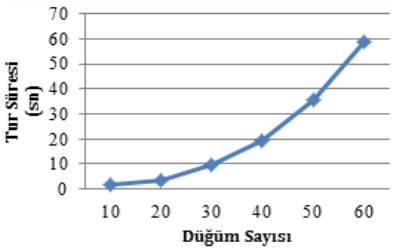
Tablo 1: Düğüm sayılarının değişmesiyle sonuca ulaşılması için gerekli olan tur sayılarının ve sürenin değişimi.

Düğüm Sayısı	Tur Sayısı	Süre(sn)	Standart Sapma
10	63,1	2,0094	0,507
20	101,3	3,6396	0,884
30	143,4	9,9543	3,837
40	180	19,330	5,875
50	236,3	35,530	11,095
60	341,7	58,819	17,703

Şekil 9 ve Şekil 10'daki grafiklerde de görüldüğü gibi düğüm sayısının artması, sonuca ulaşmak için gerekli olan süreyi ve tur sayısını arttırmaktadır. Yani buradan anlaşıldığı üzere karınca kolonisi algoritmasının problem kümesi büyütüldüğü zaman, alternatif yolların sayısı artmakta ve buna bağlı olarak çözüm süresi uzamaktadır.



Şekil 9: Tur sayısı-düğüm sayısı grafiği.



Şekil 10: Tur süresi-düğüm sayısı grafiği.

6 Sonuç ve Tartışma

Günümüzde önemli bir uygulama alanına sahip olan ÇES'lerin kullanımı ile geleneksel programlama dillerinden farklı bir yapı kullanılarak modellemeler yapılabilmektedir. Bu çalışmada ÇES yapılarının özellikleri, karınca kolonisi algoritmasının nasıl çalıştığı ve NetLogo benzetim ortamının kullanımı üzerinde durulmuştur.

Uygulama için kullanılan gezgin satıcı problemi karmaşık bir problem olup düğüm sayısının artırılması ile birlikte problemin çözümü zorlaşmakta ve uzun zaman almaktadır. Bunun yanında problemin doğru çözülmesi için parametre değerlerinin doğru belirlenmesi çok önemli bir etkendir. Çalışmada parametre olarak Dorigo, [12] çalışmasında belirtmiş olduğu değerler kullanılmıştır. Dorigo, [12]'deki sonuçlarla, bu çalışmada elde edilen sonuçlar karşılaştırıldığında, benzer olarak problem kümesinin artmasıyla birlikte çözüm süresinin de uzadığı görülmektedir. Dorigo kendi çalışmasında farklı sezgisel yöntemlerle üretilen çözüm sonuçlarını ACS ile üretilen çözümlerle karşılaştırarak ACS'nin hangi çözüm kümeleri için üstün olduğunu göstermiştir. Bu sonuçlar ışığında ACS algoritmasının diğer sezgisel yöntemlere göre bir çok durum için daha avantajlı sonuçlar elde ettiği görülmüştür. Bu çalışmada bu tarz bir karşılaştırma yapılmamış, sadece farklı senaryolar için üretilen çözümler incelenmiştir.

Uygulama için kullanılan NetLogo benzetim ortamı gerek kullanım kolaylığı, gerekse de ücretsiz bir yazılım olup herkes tarafından edinilebilmesi sebeplerinden dolayı tercih edilmiştir. NetLogo dilinin daha iyi anlaşılması amacıyla mümkün olduğu kadar kullanılan kodlara da yer verilmiştir. Kodlardan da anlaşılacağı üzere NetLogo konuşma diline çok yakın komutlara sahiptir. Bu özelliği kullanıcılar tarafından kolay anlaşılıp, hızlı bir şekilde öğrenilmesini sağlamaktadır. Gelecekte daha fazla kişi tarafından kullanılıp, zengin model kütüphanesi sayesinde eğitim aracı olarak da kullanılabilir.

7 Kaynaklar

[1] Russell, S.J. and Norvig, P., Artificial Intelligence: A Modern Approach, New Jersey, 32, 1995.

[2] Kardaş, G. Anlamsal web ortamında çalışan çok etmenli sistemlerin model güdümlü geliştirilmesi. Ege Üniversitesi Fen Bilimleri Enstitüsü Uluslararası Bilgisayar A. D. Doktora Tezi, İzmir. 2008.

[3] Weiss, G., Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, MIT Press, USA, 619 p., 1999.

[4] Wilensky, U. Netlogo web Sayfası. 1999. Available: http://ccl.northwestern.edu/netlogo/

[5] Cevre, U., Barış, Ö., Uğur, A., Gezgin satıcı probleminin genetik algoritmalarla eniyilemesi ve etkileşimli olarak Internet üzerinde görselleştirilmesi, XI I. "Türkiye 'de İnternet" Konferansı, Ankara, 8-10 Kasım 2007.

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

[7] Uğur, A., Aydın, D., Ant system algoritmasının jawa ile görselleştirilmesi, Akademik Bilişim 2006, Bildiri No: 53, Pamukkale Üniversitesi, Denizli, 9-11 Şubat 2006.

[8] Sycara, K., Multiagent Systems, AI Magazine, 19 (4): 79-92, 1998.

[9] Yurdakul, İ. Çoklu ortam verilerini aranması ve aktarılması için bir çok-etmenli sistem geliştirme. Ege Üniv. Bilgisayar Mühendisliği A. D. Yüksek lisans tezi, İzmir. 2007.

[10] Bergenti, F., Gleizes, M. and Zambonelli, F., Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook, Kluwer Academic Publishers, USA, 506, 2004.

[11] Hölldobler, B. and Wilson, E.O., The ants (Springer-Verlag, Berlin), 1990.

[12] Dorigo, M., Gambardella, L.M., Ant colonies for the traveling salesman problem, Biosystems, 43 (2), 73-81, 1996.

[13] Johnson, S., Ant Optimization in NetLogo. 2012. Available: http://www.tjhsst.edu/~sjohnson/NetLogo%20Work/Ants/Project%20Proposal%203.doc

[14] Washington, D. And Letendre, K. UNMdensity-fidelity. 2012. Available:http://ccl.northwestern.edu/netlogo/models/community/UNMdensity-fidelity

[15] Roach, C., NetLogo Ant System Model. 2007. Available: http://ccl.northwestern.edu/netlogo/models/community/AntSystem