MARKET ALIŞVERİŞİN SÜRE OPTİMİZASYONU YAPAN YAPAY ZEKA

Emre Belikırık
Söğütözü, Söğütözü Cd. No:43, 06510 Çankaya/Ankara

I. ÖZET

Araştırmanın kapsamı ve amacı:

Kullanıcıdan alınan alışveriş listesine göre belirli bir market planına bağlı kalaraktan, optimum sürede market alışverişini tamamlayıp kullanıcıya yol ve ürün sırasını dönen akıllı aracı yapmak. Aracının PEAS bilgileri aşağıda görüldüğü gibidir.

PEAS:

- Çevre: Market koridorları ve ürünlerin bulundukları reyonlar
- Çevre Özellikleri: tam gözlemlenebilir, değişmez, ayrık, single agent.
- Alıcılar: Klavye ile kullanıcıdan alışveriş listesinin alınması.
- Eyleyiciler: ekran (kullanıcıya uygun yol ve ürün listesinin dönülmesi.)
- Başarı ölçütü: Müşterinin ne kadar erken sürede alışverişini yaptığına karar veren bir metot

II. GİRİŞ

Aynı amaca sahip optimizasyon programlarında genelde makine öğrenmesi ve derin öğrenme metotlarının uygulandığı görülmüştür. Market tasarımına bağlı olaraktan (markette bulunan kat sayısı, tasarımda reyonların yerinden ziyade reyondaki ürünlerin dizilişi ve tasarımın daha büyük olması gibi) bizimkinden daha iyi sonuç verebilmektedirler. Ancak bizim uyguladığımız yöntem ile de kullanıcılara market alışverişi için büyük miktarda süre kazancı sağlayabiliyoruz. Marketimizde 112 tane ürün bulunmaktadır ve tüm ürünler için marka, kilogram ve boyut fark etmeksizin sadece tek bir ürün yerleştirilmiştir. Aynı zamanda birçok ürün için de sadece temsili bir ürün yerleştirilmiştir. Örneğin makyaj bazı, fondöten, kapatıcı, allık gibi bazı makyaj malzemeleri için ayrı ürün tanımlamak yerine sadece bir tane "Makyaj Malzemesi" isimli bir ürün tanımlanmıştır. Bunun yerine o ürünlerin de kullanılması süreyi daha doğru hesaplayıp daha doğru bir çıktı verebilirdi ancak projenin çok karmaşık olmasını istemediğim için bunu yapmaktan vazgeçtim. Bunun dengesini sağlamak için de bazı ürün gruplarına uygulamadım. Örneğin portakal, elma, kiraz yerine sadece "Meyve" şeklinde bir tanımlama yapmadım.

Markette bulunan ürünler:

Elma, portakal, muz, lahana, biber, salatalık, roka, patates, soğan, balık, balık teslim, süt, peynir, ekmek, zeytin, deterjan, çikolata, tavuk, ayçiçek yağı, zeytin yağı, toz şeker, yumurta, irmik, tuz, çubuk kraker, cipsi, kurabiye, lokum, labne, ketçap, su, kola, pil, şampuan, duş jeli, deodorant, diş fırçası, diş macunu, parfüm, meyve suyu, pasta, maden suyu, yüz temizleme jeli, makyaj malzemesi, nemlendirici, güneş kremi, tavuk, sade un, mısır unu, makarna, salça, ton balığı, turşu, pirinç, bulgur, fasulye, mercimek, nohut, mısır, türk kahvesi, filtre kahve, soğuk kahve, sıcak çikolata, salep, süt tozu, televizyon, şalgam suyu, limonata, kefir, enerji içeceği, çay, bitki çayı, puzzle, oyuncak, oyun hamuru, kitap, dergi,gazete,defter, kalem, boya, yapıştırıcı, ofis aksesuarı, televizyon, kulaklık, kağıt, telefon, lamba, bilgisayar, monitor, bilgisayar aksesuarı, simit, poğaça, kek, yaş pasta, tatlı, kakao, maya, vanilin, kabartma tozu, krem şanti, nişasta, kahvaltılık,petshop, atıştırmalık, kuruyemiş, alkol.

III. YÖNTEM

Projede tercih ettiğimiz yöntem olarak genetik algoritma modelimize uygulanmıştır ve bir tane yardımcı arama algoritması olarak da derinlik öncelikli arama(BFS) kullanılmıştır. Derinlik öncelikli aramanın programdaki amacı koridorlara arası en kısa yolu bulmaktır. Bizim tasarımımızda her koridor arası geçiş eşit süre gerektirmektedir. Farklı bir durum olsaydı BFS yerine başka bir arama algoritması kullanıp(A* olabilir) bunu yine aynı şekilde çözebilirdik. Birkaç tane de düzenleme için yapılmış olan fonksiyonlar bulunmaktadır (kullanıcıdan alınan girdinin üzerinde uygulanan). Aşağıda kullandığım başlıca fonksiyonların Python kodlarını görebilirsiniz.

```
def BFS_SP(graph, start, goal):
    explored = []
    queue = [[start]]

    if start == goal:
        return [start]

    while queue:
        path = queue.pop(0)
        node = path[-1]

        if node not in explored:
            neighbours = graph[node]

            for neighbour in graph[node]:
                new_path = list(path)
                new_path.append(neighbour)
                queue.append(new_path)

                if neighbour == goal:
                    temp = []
                    for t in new_path:
                        temp.append(str(t))
                    return temp
            explored.append(node)
    return
```

Figure1: Markette iki reyon arasındaki en kısa yolu dönen *BFS* algoritması.

A. Genetik algoritmanın kendi sistemimize uygulanışı

a) Kullanıcıdan girdi alınması ve popülasyon oluşturulması: Kullanıcıdan önce alışveriş listesi girdisini alıyoruz. Girdi alındıktan sonra eğer içerisinde sipariş verilip hazırlanması gereken bir ürün bulunuyorsa listemize o ürünü teslim almayı içeren bir madde daha ekliyoruz. Örneğin alışveriş listemizde "*Et*" bulunuyorsa "*Et teslim"* şeklinde bir madde daha ekliyoruz. Sonrasında alışveriş listesini yardımcı fonksiyonumuz olan *targets* ile gidilmesi gereken reyonları belirliyoruz.

```
def targets(list):
    x = []
    for i in list:
        for j in adjac_lis.keys():
            if i in j.getElements():
                x.append(j)
    for i in range(len(x)):
        if x[i] != 0:
            for j in range(i+1,len(x)):
                if(x[i]==x[j]):
                    x[j] = 0

    return remove_values_from_list(x,0)
```

Figure2: Gidilmesi gereken reyonları belirleyen *targets* fonksiyonumuz.

Gidilmesi gereken reyonlar belirlendikten sonra sırada popülasyon yaratma var. Popülasyon yaratma fonksiyonumuz verdiğimiz parametre kadar genom oluşturur. Genomlar ise gidilmesi gereken reyonların hepsinin bulunduğu rasgele bir sıra içerir.

```
def generate_population(size: int,alist) -> Population:
    return [generate_genome(alist) for _ in range(size)]
```

Figure3: Popülasyon yaratma fonksiyonu.

```
def generate_genome(alist) -> Genome:
    t = targets(alist)
    return sample(t, k=len(t))
```

Figure4: Genom oluşturma fonksiyonu.

b) *Popülasyonun uygunluğunun hesaplanması:* Oluşturduğumuz popülasyondaki en iyi iki genomu belirlemek ve genomların başarısını ölçmek için ise uygunluk fonksiyonumuzu kullanıyoruz. Uygunluk fonksiyonu verdiğimiz genomdaki alışveriş sırasını uygulayıp ortaya çıkan süre ağırlığını hesaplayıp döner. Uygunluk fonksiyonu biraz uzun olduğu için buraya ekleyemiyorum ancak temel olarak bahsedeceğim. Ağırlığı hesaplarken marketin girişinden genomdaki ilk reyona doğru gider ve gidilen yolun masrafını ağırlığa ekler. Genomdaki sıralı ikililer arasındaki yolu belirlerken BFS kullanıyoruz. Reyona ulaştıktan sonra alınması gereken tüm ürünleri alır ve önceden belirlenmiş olan ürünleri almanın süre masrafını ağırlığa ekler. Sipariş verilip sonra alınması gereken ürünlerde ise eğer ürünü ürünün hazırlanış süresi bittikten sonra almaya gelirsek sadece yol masrafı eklenir ancak hazırlanış süresi bitmeden önce gelirsek yol masrafına ek olarak aradaki süre farkını da ağırlığa ekler. Uygunluk fonksiyonunun bir yan işlevi ise o genomun

oluşturduğu alışveriş süresi boyunca geçtiğimiz tüm koridorların sırasını dönmektir.

c) Popülasyondaki en iyi iki genomun belirlenmesi ve yeni jenerasyonun oluşturulması: Tüm popülasyonun uygunluğunu hesapladıktan sonra en kısa sürede alışverişi tamamlayan iki genomu direkt yeni jenerasyona ekliyoruz ve popülasyondaki uygunluğu yüksek olan genomların çaprazlanması ile orijinal popülasyonun büyüklüğüne ulaşıncaya kadar yeni jenerasyona çaprazlama ile yeni genomlar ekliyoruz. Çaprazlama fonksiyonumuz vermiş olduğumuz iki genomdan iki yeni genom oluşturur.

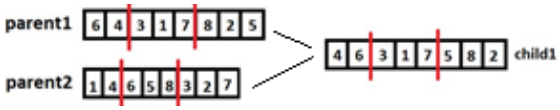


Figure5: *Çaprazlama* işleminin örneği.

Yukarıda sadece bir yeni genom için çaprazlama örneği verilmiştir. Bu işlemin tersi de ikinci genom için uygulanır ve iki yeni genom oluşturulur.

Çaprazlama sonucu oluşan genomlara 0.5 olasılıkla mutasyon işlemi uyguluyoruz. Bu işlem ise rastgele iki ürünün yer değiştirilmesi şeklinde gerçekleşir.

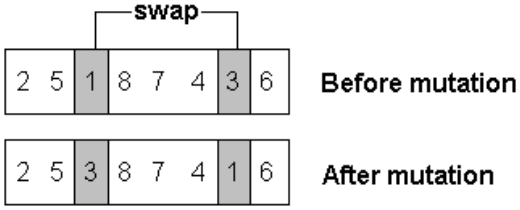


Figure6: *Mutasyon* işlemi.

d) Sonuç dönülmesi ve evrim fonksiyonu: Bu zamana kadar bahsettiğimiz tüm olayların bir arada gerçekleşmesini sağlayan evrim fonksiyonumuz bulunmaktadır. Bu fonksiyon parametre olarak jenerasyon sayısı alır ve tüm bu olayları jenerasyon sayısı kadar tekrarlar. Tüm jenerasyonlar sonuçlanınca elde ettiğimiz en uygun sonucu ve yolu fonksiyonumuz döner.

Figure7: *Market tasarımı*.

IV. SONUÇLAR VE YORUMLAR

Verilen girdilere göre çıkan sonuçları örnek olarak ekliyorum. Ürünleri virgül ile ayırarak terminale yazıyoruz.

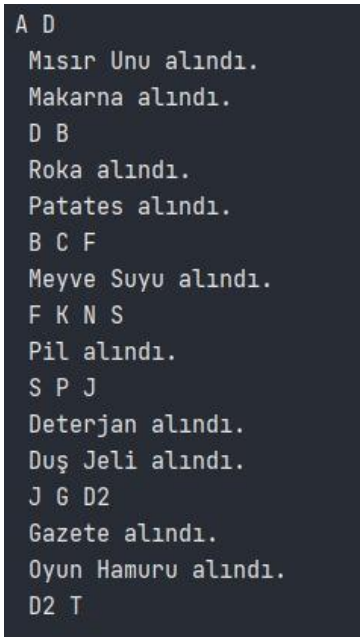
Girdi: Ekmek, Muz, Portakal, Şalgam Suyu, Televizyon, Balık, Kalem, Dergi, Tavuk, Diş Fırçası.

Çıktı:

```
A B C F K N
 Tavuk siparişi verildi.
 N K
 Balık siparişi verildi.
 K F
 Şalgam Suyu alındı.
 F H G D2
 Kalem alındı.
 Dergi alındı.
 D2 D A
 Ekmek alındı.
 A B
 Muz alındı.
 Portakal alındı.
 B E J
 Diş Fırçası alındı.
 J H K
 Balık alındı.
 K N
 Tavuk alındı.
 N S
 Televizyon alındı.
 S R T
```

Girdi: Mısır Unu, Makarna, Roka, Patates, Meyve Suyu, Pil, Deterjan, Duş Jeli, Gazete, Oyun Hamuru.

Çıktı:



Alışveriş listesi aynı kalmasına rağmen birçok girdide ürünlerin alış sırası değişebiliyor ancak süre değişmiyor. Bunun sebebi de her bir ürünü almanın masrafının ve gidilen yolun masrafının gerçekçi değerlerin hesaplanamayıp aynı süreyi dönen birden fazla sıralamanın oluşuyor olması. Daha önceden bahsettiğim gibi eğer marketimiz daha büyük olsaydı, ürünlerimizin alış süreleri gerçekçi olsaydı ve zamanı etkileyen diğer birçok faktör de göz önüne alınabilseydi aynı ürünler için programımız bir tek sıraya düşmese bile farklı oluşan sıraların sayısı azaltılabilirdi.

V. KAYNAKÇA

[1] https://medium.com/basecs/breaking-down-breadth-first-search-cebe696709d9

[2] https://www.javatpoint.com/ai-informed-search-algorithms#:~:text=Greedy%20best%2Dfirst%20search%20algorithm,the%20advantages%20of%20both%20algorithms.

[3]https://www.veribilimiokulu.com/genetik-algoritma/

[4] Berkeley CS 188 Lectures https://www.youtube.com/channel/UCHBzJsIcRIVuzzHVYabikTQ/videos

[5] https://pygad.readthedocs.io/en/latest/