

Proje Raporu

Proje Adı: Akıllı Arama Simülatörü

Grup Üyeleri:

- [Hatice Sena BAYAR] ([22060348])
- [İlayda ÇETİN] ([22060392])

1. Giriş

Bu proje, uninformed ve bilgili informed arama stratejilerinin performansını bir labirent ortamı üzerinde simüle etmek amacıyla gerçekleştirilmiştir. Proje kapsamında **Genişlik Öncelikli Arama (BFS)**, **Derinlik Öncelikli Arama (DFS)** ve **A* Arama Algoritması** labirent çözümü için uygulanmıştır.

Temel amaç, farklı labirent yapıları ve başlangıç/hedef koşulları altında algoritmaların etkinliklerini; **genişletilen düğüm sayısı**, **bulunan yolun maliyeti (uzunluğu)** ve **algoritmanın çalışma süresi** gibi metrikler üzerinden karşılaştırmaktır.

2. Yöntem (Kullanılan Algoritmalar ve Uygulama)

2.1 Labirent Yapısı (Maze Sınıfı)

Simülatör, **maze.py** dosyasında tanımlanan Maze sınıfı üzerinden çalışır.

- Labirentler, .txt dosyalarından okunur ve S (Başlangıç), G (Hedef), # (Duvar) ve . (Yol) karakterlerinden oluşur.
- Hareket, labirent üzerinde yukarı, aşağı, sola ve sağa olmak üzere dört yönde (komşu düğümler) gerçekleştirilir.

2.2 Arama Algoritmaları (algorithms.py)

Üç temel arama algoritması **algorithms.py** dosyasında uygulanmıştır:

- Genişlik Öncelikli Arama (BFS):** Bir kuyruk (queue) kullanarak tüm komşuları katman katman ziyaret eder ve bu sayede **en kısa (optimal) yolu** bulmayı garanti eder.
- Derinlik Öncelikli Arama (DFS):** Bir yığın (stack) kullanarak, olabildiğince derinlere inmeye çalışır. Çözümü hızlı bulabilir ancak **optimal (en kısa) yolu bulmayı garanti etmez**.

3. **A* Arama Algoritması (A* Search):** Maliyeti ($cost_so_far$, $g(n)$) ve sezgiseli (heuristic, $h(n)$) birleřtirerek öncelikli kuyruk (heapq) kullanan bilgili bir arama yöntemidir.

Sezgisel Fonksiyon: Bu projede **Manhattan Mesafesi** kullanılmıřtır: $h(n) = |x_{\{hedef\}} - x_n| + |y_{\{hedef\}} - y_n|$. Bu sezgisel, labirent ortamında kabul edilebilir (admissible) bir sezgiseldir ve A*'ın optimal yolu bulmasını saęlar.

2.3 Görselleřtirme

Tüm arama süreçleri, **visualizer.py** modülü kullanılarak matplotlib kütüphanesi aracılığıyla **adım adım** görselleřtirilmiřtir. Bu sayede, her bir algoritmanın labirentte izledięi arama yolu ve genişlettięi düğümler anlık olarak takip edilmiřtir.

3. Test Senaryoları

Algoritmaları farklı kořullar altında karřılařtırmak amacıyla iki farklı labirent ve bařlangıç/hedef çifti belirlenmiřtir:

Senaryo	Labirent Dosyası	Bařlangıç Koordinatı	Hedef Koordinatı	Senaryo Amacı
Senaryo 1	maze1.txt	(0, 0)	(6, 4)	Algoritmaların standart bir labirent üzerindeki genel performansını karřılařtırma.
Senaryo 2	maze2.txt	(1, 1)	(9, 9)	Daha karmařık bir labirent yapısında (uzun dar koridorlar) algoritmaların verimlilik ve keřfetme stratejisi farklılıklarını test etme.

4. Sonuçlar, Karşılaştırmalar ve Değerlendirme

Tüm algoritmaların iki farklı senaryoda çalıştırılması sonucunda elde edilen metrikler aşağıdaki tabloda özetlenmiştir.

Algoritma (Labirent)	Başlangıç	Hedef	Yol Uzunluğu	Genişletilen Düğüm	Süre (ms)
BFS (maze1.txt)	(0, 0)	(6, 4)	35	57	8369.35
DFS (maze1.txt)	(0, 0)	(6, 4)	43	53	7667.84
A* (maze1.txt)	(0, 0)	(6, 4)	35	51	7263.68
BFS (maze2.txt)	(1, 1)	(9, 9)	17	43	6816.86
DFS (maze2.txt)	(1, 1)	(9, 9)	31	41	7492.49
A* (maze2.txt)	(1, 1)	(9, 9)	17	21	3887.27

4.1 Çözüm Maliyeti (Yol Uzunluğu) Karşılaştırması

- **Optimal Çözüm:** BFS ve A* algoritmaları, her iki senaryoda da en kısa yolu bulmuşlardır (S1: 35 adım; S2: 17 adım). Bu, her iki algoritmanın da **optimal (en iyi) çözümü bulma** özelliğini doğrular.
- **DFS Dezavantajı:** DFS algoritması her iki labirentte de daha uzun yollar bulmuştur (S1: 43 adım; S2: 31 adım). Bu durum, DFS'in kör bir şekilde derinlere inme eğilimi nedeniyle suboptimal çözümler üretebileceğini göstermektedir.

4.2 Verimlilik (Genişletilen D ğ m Sayısı ve S re) Karşılařtırması

Kriter	En Verimli Algoritma	Açıklama
Geniřletilen D�ğ�m (Verimlilik)	A*	A*, her iki labirentte de en az d�ğ�m� genişleten algoritmadır (S1: 51, S2: 21). Bu, sezgisel (Manhattan Mesafesi) fonksiyonunun arama alanını etkin bir řekilde daralttığını kanıtlar.
S�re (ms)	A*	A*, en az d�ğ�m genişletme sayesinde, her iki senaryoda da en hızlı sonuca ulařan algoritmadır (S2'de 3887.27 ms ile açık ara farkla �ndedir).
DFS Yanılması	DFS	DFS, BFS'ten daha az d�ğ�m genişletebilir (S2'de 41 vs 43) ve daha kısa s�rede �z�m� bulabilir (S1'de 7667 ms vs 8369 ms), ancak bu avantajı daha uzun bir yol maliyeti pahasına elde etmiřtir.

4.3 Değerlendirme

Algoritma	Avantaj	Dezavantaj
BFS	Optimal (en kısa) yolu bulmayı garantiler.	Arama alanını genişlettiğı i�in A*'a g�re daha fazla d�ğ�m genişletir ve daha yavařtır.
DFS	Hafıza kullanımı genellikle d�ř�kt�r. Sığ bir �z�m varsa hızlı bulabilir.	Optimal yolu bulmayı garanti etmez ve gereksiz yere labirentin derinliklerine inerek zaman kaybedebilir.
A*	Hem optimal �z�m� garantiler hem de sezgisel sayesinde en az d�ğ�m� genişleterek en verimli �alıřır.	Sezgisel fonksiyonun kalitesine baėlıdır; k�t� bir sezgisel performansı d�ř�rebilir (Bu projede iyi �alıřmıřtır).

Sonuç: Sim lat r sonu larına g re, verilen labirent ortamı i in **A* Arama Algoritması**, hem  z m kalitesi (optimal yol) hem de verimlilik (en az genişletilen d ğ m ve en kısa s re) a ısından en  st n performansı sergilemiřtir. BFS, optimal yolu bulsa da, A* kadar verimli deėildir. DFS ise hızlı olabilir ancak bulunan yolun kalitesi d ř kt r.

5. Kaynakça

Proje Geliştirme Ortamı ve Kütüphaneler

Python: Proje geliştirme ortamı ve programlama dili (Python 3.x).

Matplotlib: Görselleştirme için kullanılmıştır.

NumPy: Matris ve sayısal işlemler için kullanılmıştır.

tabulate: Sonuç tablosunun terminalde gösterilmesi için kullanılmıştır.