문제 해결 에이전트

한경수
성결대학교 컴퓨터공학과

1

Introduction

- 이성적 에이전트 유형
  - 단순 반응 에이전트
  - 모델 기반 반응 에이전트
  - 목표 기반 에이전트
  - 효용 기반 에이전트
  - 학습 에이전트
- 특정 문제를 해결하는 에이전트는 어떻게?

2

학습 목표

- 이론지식 응용 역량
  - 다음 개념을 설명할 수 있다.
    - 문제 해결 에이전트
    - 문제 표현하기: 문제 정의 요소, 상태 공간
    - 해결책 탐색: 탐색 트리, 노드 확장
    - 탐색 알고리즘의 기반 구조
- 공학기술 및 도구 활용 역량
  - 문제의 조건이 주어졌을 때, 상태 공간 개념을 활용하여 문제를 정의할 수 있다

3

문제 해결 에이전트

성능 척도를 최대화하기 위해
- 목표를 설정하고
- 그 목표를 달성하는 행동 시퀀스를 찾아내는 에이전트

4

예: 루마니아 여행

현재 루마니아 Arad에서 휴가 여행 중.

작업 환경 정의

- **성능 척도**: 선탠, 루마니아어 수준 향상, 경치 구경, 클럽문화 체험, ...
- **환경**: 태양빛, 온도, 날씨, 소리, 주변행인, 여행 가이드북, 클럽, ...
- **작동기**: 태양열 저장기, 음성인식/합성기, 언어처리기, 보행 컨트롤러, 댄스 컨트롤러, ...
- **센서**: 조도계, 마이크, 스피커, 카메라, GPS, 가속도계

5

예: 루마니아 여행

현재 루마니아 Arad에서 휴가 여행 중.

내일 새벽 1시에 Bucharest에서 출발하여 인천으로 되돌아오는 비행편이 예약되어 있음. 이 일정 변경/취소는 불가능함. 도시 간 이동수단은 자동차.

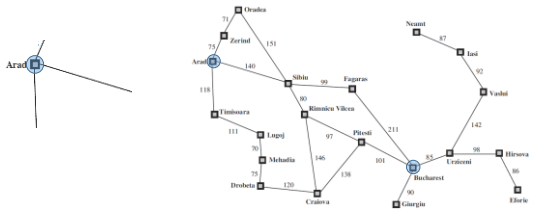
뭘 해야 할까?

6

예: 루마니아 여행

지도가 없는 경우
(알려지지 않은 환경)

지도가 있는 경우
(알려진 환경)



- 완전 관측 가능, 결정적, 알려진 환경의 해결책: 고정된 행동 시퀀스
- 부분 관측 가능 또는 비결정적 환경의 해결책: 지각 결과에 따라 다른 미래 행동을 추천하는 분기 전략(branching strategy)

7

문제 해결 절차

1. 목표(goal) 설정
   - 목표는 에이전트의 행위를 체계화하는데 도움이 됨
     - 내일 (새벽 1시 – 비행 수속 시간) 전에 Bucharest 공항에 도착해야 함
2. 문제(problem) 표현
   - 목표 달성에 필요한 상태들과 행동들의 표현 방법(추상 모델) 고안
     - 상태: 도시들
     - 행동: 인접 도시로의 도시 간 이동

8

문제 해결 절차

3. 해결책 **탐색(search)**
   - 실세계에서 행동을 취하기 전에, 모델에서 행동 시퀀스들을 시뮬레이션
   - 목표를 달성하는 행동 시퀀스(해결책) 하나를 찾아낼 때까지 탐색
   - 해결책을 찾거나, 해결책이 없다는 걸 알아내거나
4. 실행(execution)
   - 해결책의 각 행동을 하나씩 실행

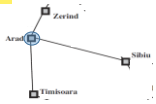
9

문제 표현하기

목표 달성에 필요한 상태들과 행동들의 표현 방법 고안
→ 추상 모델(abstract model)

10

탐색 문제 정의 요소



1. 상태
   - 환경의 가능한 상태들의 집합
   - 상태 공간(state space)이라 부름
2. 초기 상태(initial state)
   - 에이전트가 시작하는 상태; 예: $Arad$
3. 목표 상태(goal states)
   - 하나 또는 그 이상의 목표 상태; 예: $\{Bucharest\}$
   - 추상적인 특성으로 목표를 명시해야 하는 경우도 있음
     - 예: 청소기의 목표 상태는?
   - $IS-GOAL$ 함수를 통해 목표 상태 정의

11

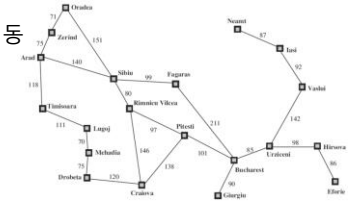
탐색 문제 정의 요소

4. 행동
   - 에이전트가 취할 수 있는 행동들
   - $ACTIONS(s)$: 상태 s에서 실행될 수 있는 행동들의 집합
     - 각 행동이 상태 s에서 적용가능(applicable)하다고 함
     - 예: $ACTIONS(Arad) = \{ToSibiu, ToTimisoara, ToZerind\}$
5. 이행 모델(transition model)
   - 각 행동의 결과
   - $RESULT(s,a)$: 상태 s에서 행동 a를 취했을 때의 결과 결과 상태
     - 예: $RESULT(Arad, ToZerind) = Zerind$
6. 행동 비용 함수(action cost function)
   - $ACTION-COST(s,a,s')$ 또는 $c(s,a,s')$
   - 상태 s에서 행동 a를 취하여 상태 s'에 도달하는데 필요한 비용 값
   - 성과 척도 반영; 예: 거리, 시간

12

상태 공간

- 상태 공간은 그래프를 형성함
  - 노드: 상태
  - 방향 간선: 행동



- 행동들의 한 시퀀스가 경로 하나를 형성함

13

비용과 해결책

- 해결책(solution; 해)
  - 초기 상태에서 목표 상태에 이르는 경로
- 최적 해결책(optimal solution; 최적해)
  - 경로 비용이 가장 적은 해결책
- 경로 비용
  - 경로를 구성하는 각 행동 비용의 총합

14

문제 표현

- 복잡한 실세계 문제를 잘 반영하는 상태 공간 선택

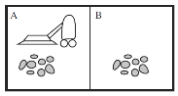
$Arad$
$ToSibiu$

추상화(abstraction) 과정이 필요함!

15

문제 표현 예: 청소 로봇

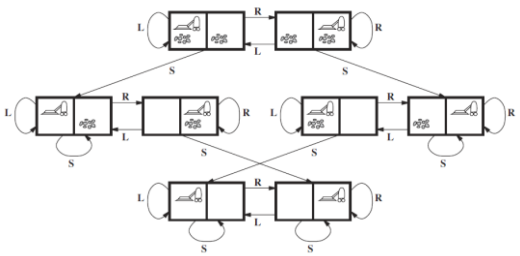
- 상태: 에이전트의 위치와 각 위치의 먼지 존재 여부
  - 상태 수: $2 \times 2 \times 2 = 8$
  - n개의 위치가 있다면: $n \cdot 2^n$



- 초기 상태: 상태 중 하나
- 목표 상태: 모든 위치가 깨끗한 상태들
- 행동: $\{Left, Right, Suck\}$
- 이행 모델:
  - $Left$, $Right$: 각 행동에 따른 위치 이동 결과
  - $Suck$: 현 위치의 먼지 제거
- 행동 비용: 각 행동 비용 = 1

16

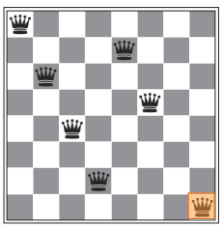
청소 로봇의 상태 공간 그래프



17

문제 표현 예: 8-Queens 문제

- 동일한 행, 열, 대각선에는 하나의 Queen만 존재하도록 8개의 Queen을 배치



18

문제 표현 예: 8-Queens 문제

- 상태: 보드 위 퀸들(0~8개)의 모든 배치
- 초기 상태: 보드에 퀸이 없는 상태
- 목표 상태: 행, 열, 대각선에 중복됨이 없이 8개의 퀸이 배치된 보드 상태
- 행동: 빈 공간에 퀸 하나를 추가
- 이행 모델: 해당 위치에 퀸이 추가된 보드 상태를 리턴
- 행동 비용: 모든 해결책의 경로 비용이 동일하여 행동 비용은 고려할 필요 없음
- 상태 공간 크기 = $64 \cdot 63 \cdots 57 \approx 1.8 \times 10^{14}$

19

문제 표현 예: 8-Queens 문제(개선)

- 상태: 아래 요건을 만족하는 n개 퀸들의 모든 가능한 배치
  - $0 \leq n \leq 8$
  - 각 열마다 하나씩; n번째 퀸은 좌측으로부터 n번째 열에 배치
  - 행과 대각선에 중복되는 퀸이 없도록 배치
- 행동: 빈 열 중 가장 왼쪽 열 공간 하나에 퀸 하나를 추가
  - 행과 대각선에 중복되는 퀸이 없도록 위치 선정
- 상태 공간 크기 = 2,057

20

해결책 탐색

해결책 = 행동 시퀀스

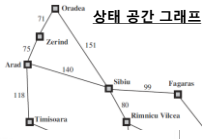
문제 풀이 과정 = 가능한 행동 시퀀스들 중에서 적절한 행동 시퀀스를 찾아내는 과정

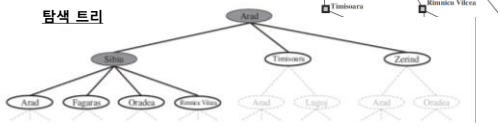
즉, 문제 풀이 = 상태 공간 탐색

21

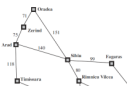
탐색 트리

- 초기 상태에서 목표 상태에 이르는 경로(해결책)를 찾아 내기 위해, 초기 상태에서 시작하는 다양한 경로들로 구성되는 트리
  - 노드: 상태 공간의 상태
  - 간선: 행동
  - 루트: 초기 상태





22



상태 공간과 탐색 트리

상태 공간

- 세계에 대한 상태 집합과
- 한 상태에서 다른 상태로 이행시키는 행동들을 표현함

탐색 트리

- 상태 공간 상태들 간에, 목표 상태를 향하는, 경로들을 표현함
- 한 상태에 이르는 복수의 경로가 포함될 수 있음
  - 한 상태에 대한 복수의 노드 존재 가능
- 트리의 각 노드에서 루트에 이르는 역경로는 유일함

23

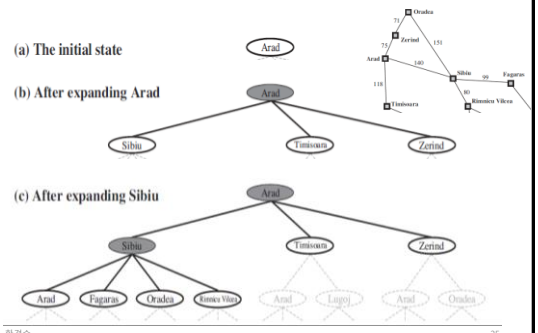
노드 확장

1. 해당 상태에서 취할 수 있는 행동 확인: $ACTIONS(s)$
2. 각 행동의 결과 상태 확인: $RESULT\ (s, a)$
3. 각 결과 상태에 대한 새로운 노드 생성
   - 자식 노드 또는 후속자(successor) 노드라고 부름

- 탐색 트리의 노드 확장을 통해 탐색이 진행됨

24

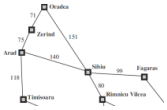
노드 확장 예: Arad → Bucharest 길 찾기



한경수 25

25

경계와 도달 상태



- 경계(frontier)
  - 개방 리스트(open list)
  - 생성된 노드들 중 아직 확장되지 않은 노드들을 탐색 트리의 경계라고 부름
    - 미확장 단말 노드들

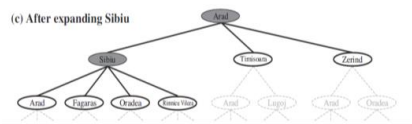
- 도달 상태
  - 어떤 상태에 대한 노드가 탐색 트리에 생성되면 그 상태는 도달되었다고 함(reached)

한경수 26

26

탐색

- 결국 다음 번에 경계에서 어떤 노드를 선택하여 확장할 것인지에 따라 탐색 방법이 달라짐



한경수 27

27

우회 경로

- 탐색 트리에 상태들이 반복되고 있음
  - 순환 경로(loopy path; cycle) 때문
  - 탐색 트리가 무한(infinite)!

- 우회 경로(redundant path)
  - Arad-Sibiu vs. Arad-Zerind-Oradea-Sibiu
  - 한 상태에서 다른 상태로 가는 길이 2가지 이상 있을 때 존재
  - 순환 경로도 우회 경로의 특별 케이스에 해당
  - 최적 경로를 찾아야 하므로 우회 경로(예: Arad-Zerind-Oradea-Sibiu)는 고려할 필요 없음

한경수 28

28

우회 경로 해결 방법 1: 그래프 탐색

- 이전에 도달(reach)된 모든 상태를 기억하고, 각 상태에 이르는 최고 경로만 유지
  - 모든 우회 경로 탐지 가능
  - 노드 확장 시 각 자식 노드에 대해 다음 경우에만 경계(frontier)에 추가
    - 이전에 도달된 적이 없는 경우 또는
    - 이전의 어떤 경로보다도 더 낮은 비용의 경로로 도달되고 있는 경우
- 많은 우회 경로가 존재하는 상태 공간에 적합한 방법
- 도달 상태 테이블이 메모리에 로딩 가능할 때 선호됨

한경수 29

29

우회 경로 해결 방법 2: 트리 탐색

- 동일 상태에 도달하는 둘 이상의 경로가 불가능하거나 드문 문제인 경우, 트리 탐색(tree-like search) 수행
  - 도달 상태를 추적하지 않고 우회 경로를 체크하지 않음 → 메모리 공간 절약
  - 방법 1처럼 우회 경로를 체크하는 탐색 알고리즘을 그래프 탐색(graph search)이라고 함

한경수 30

30

우회 경로 해결 방법 3: 순환 경로 체크

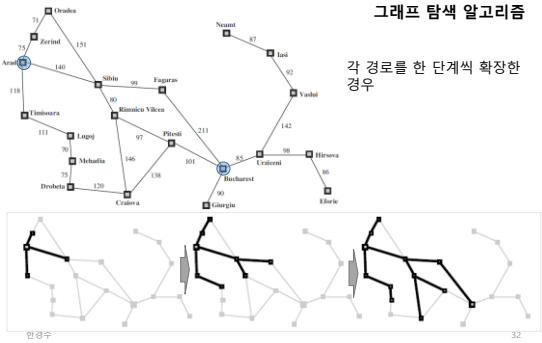
- 순환 경로(cycle)만 체크
- 각 노드에 연쇄적인 부모 포인터가 존재하므로, 추가적인 메모리 필요 없이 순환 경로 체크 가능
  - 부모 포인터를 연쇄적으로 따라 올라가면서, 그 상태가 경로에서 이전에 등장했는지 확인
    - 부모 포인터 체인을 끝까지 따라 올라가서, 모든 순환 경로 삭제 또는
    - 부모 포인터 체인을 몇 개까지만(예: 부모, 조부모, 증조부모) 따라가서 짧은 순환 경로만 삭제

31

탐색 트리 예: Arad → Bucharest 길 찾기

그래프 탐색 알고리즘

각 경로를 한 단계씩 확장한 경우



32

탐색 알고리즘 구조 및 성능

탐색 알고리즘 구현을 위해 필요한 구조, 알고리즘 성능 측정

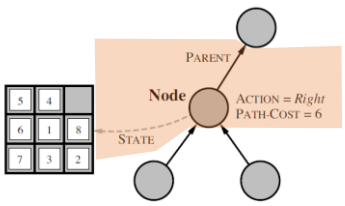
33

탐색 알고리즘의 데이터 구조

- 탐색 과정 중 생성되는 탐색 트리를 저장할 자료 구조가 필요함
- 탐색 트리의 각 노드에 저장되는 요소
  - 상태: 노드에 대응되는 상태 공간의 상태
  - 부모: 이 노드를 생성한 부모 노드
  - 행동: 이 노드를 생성하기 위해 부모의 상태에 적용된 행동
  - 경로 비용: 초기 상태에서 이 노드까지의 경로 비용($g(n)$)

34

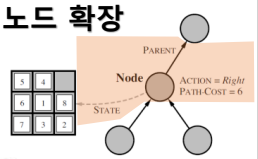
데이터 구조: 노드



- 해결책
  - 목표 노드가 발견되면, 부모 포인터를 따라 루트까지 거슬러 올라가면서 행동 시퀀스 생성

35

데이터 구조: 노드 확장



function EXPAND(*problem*, *node*) **yields** nodes
  $s \leftarrow node$.STATE
  for each *action* **in** *problem*.ACTIONS(s) **do**
    $s' \leftarrow problem$.RESULT(s, *action*)
    $cost \leftarrow node$.PATH-COST + *problem*.ACTION-COST(s, *action*, s')
    yield NODE(STATE=s', PARENT=*node*, ACTION=*action*, PATH-COST=*cost*)

36

데이터 구조: 경계(frontier)

- 경계는 다음 확장할 노드의 선택이 용이하도록 저장되어야 함
- 탐색 전략에 따라 큐(FIFO 큐), 스택(LIFO 큐), 우선순위 큐 이용
- 연산
  - $IS-EMPTY(frontier)$: 경계가 비어 있는지 검사
  - $POP(frontier)$: 경계의 최상위(top) 노드를 삭제하여 리턴
  - $TOP(frontier)$: 경계의 최상위(top) 노드 리턴(미삭제)
  - $ADD(node, frontier)$: 경계의 적절한 위치에 노드 삽입

37

데이터 구조: 도달 상태(reached state)

- 반복 상태 검사를 효율적으로 수행할 수 있도록 룩업 테이블(해시 테이블) 이용
  - 키: 상태
  - 값: 그 상태에 대한 노드

38

탐색 알고리즘 성능 평가 척도

- 완전성(completeness):
  - 해결책이 존재한다면, 해결책 발견을 보장하는가?
  - 존재하지 않는다면, '해결책 없음'을 정확히 알릴 수 있는가?
- 비용 최적성(cost optimality): 최적(경로 비용이 가장 적은) 해결책을 찾는가?
- 시간 복잡도: 해결책 찾는데 소요되는 시간
  - 고려하는 상태/행동의 수; 생성되는 노드의 수
- 공간 복잡도: 탐색 수행을 위해 필요한 메모리
  - 저장되는 노드의 최대 개수

39

시간/공간 복잡도 측정

- 시간/공간 복잡도를 문제의 난도(difficulty) 측면에서 생각할 수 있지 않을까?
  - 상태 공간 그래프의 크기: 노드의 개수 + 간선의 개수
  - 상태 공간 그래프를 명시적으로 표현하지 못하는 경우도 있음; 무한(infinite) 그래프
- 시간/공간 복잡도는 다음 3가지 값으로 표시
  - 깊이(d): 최적 해결책에서 행동의 수
    - 가장 얕은 목표 노드의 깊이
  - 경로들에서 행동의 최대값(m)
    - 탐색 트리의 최대 깊이
  - 분기 지수(b; branching factor): 노드 하나에서 고려해야 하는 후속자(successor)의 개수

40

정리

- 문제 해결 에이전트
- 문제 표현하기
  - 문제 정의 요소
  - 상태 공간
- 해결책 탐색
  - 탐색 트리
  - 노드 확장
  - 우회 경로
- 탐색 알고리즘 기반 구조
  - 노드, 경계, 도달 상태
- 탐색 알고리즘 성능 평가 척도
  - 완전성, 최적성, 시간 복잡도, 공간 복잡도

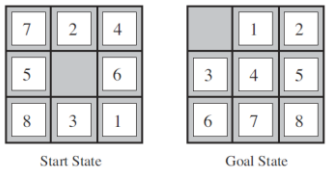
41

정리 문제 풀이

42

정리 문제 1: 8-퍼즐

- 상태 공간 개념을 이용하여 다음과 같은 8-퍼즐 문제를 정의하시오.



43

정리 문제 2: 미로 탈출

- 미로 속에 로봇이 하나 있다. 다음과 같은 조건이 주어졌을 때, 각 물음에 답하시오.
  - 로봇이나 미로의 높이는 고려치 않음. 미로 안에 다른 에이전트는 없음
  - 미로의 크기는 가로, 세로 길이가 동일한 정사각형 형태임
  - 로봇의 목표는 이 미로에서 탈출하는 것임
  - 로봇은 동/서/남/북 네 방향으로 한번에 방향 전환을 할 수 있음
  - 로봇은 미로 중앙에서 북쪽을 바라본 상태에서 시작함
  - 로봇은 정면 방향으로 원하는 거리만큼 직선 형태의 전진 이동이 가능함
  - 로봇은 벽을 만나면 벽에 부딪히기 직전에 자동으로 멈춤
  - 미로의 길은 동/서/남/북 네 방향으로만 존재함
  - 미로에는 길이 서로 만나는 교차로가 존재할 수 있음
  - 상태 공간 개념을 이용하여 문제를 정의하시오.
  - 상태 공간의 크기는 얼마나 되는가?

44