2장/3장. 에이전트와 문제 해결 방식

목차

- 에이전트 기본 유형
- 문제 정의 방법
- 문제 예시

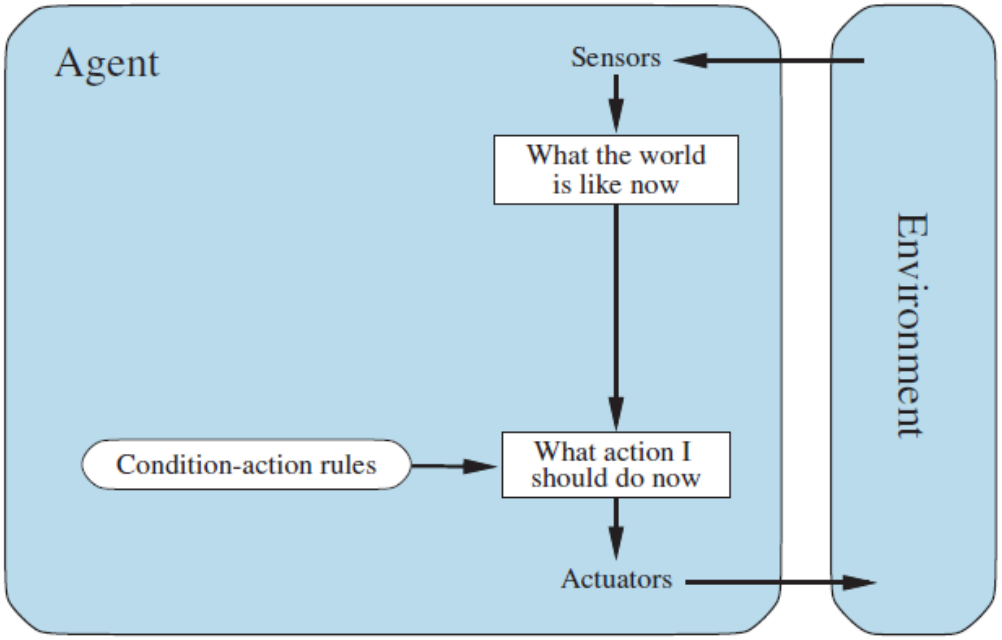
에이전트 구조

지능형 시스템의 기본 원칙에 따는 4가지 에이전트 유형

- ♦단순 반사 에이전트(Simple reflex agents)
- ♦모델 기반 반사 에이전트 (Model-based reflex agents)
- ♦목표 기반 에이전트(Goal-based agents)
- ♦효용 기반 에이전트(Utility-based agents)
- ♦학습형 에이전트(Learning-based agents)

다른 모든 유형의 에이전트는 학습형 에이전트로 구현 가능

단순 반사 에이전트



직사각형: 에이전트의 현재 내부 상태
타원: 처리 과정 중 사용되는 배경 정보

- 현재 지각에 근거해서 동작 선택
  - 지각 이력의 나머지 부분은 무시

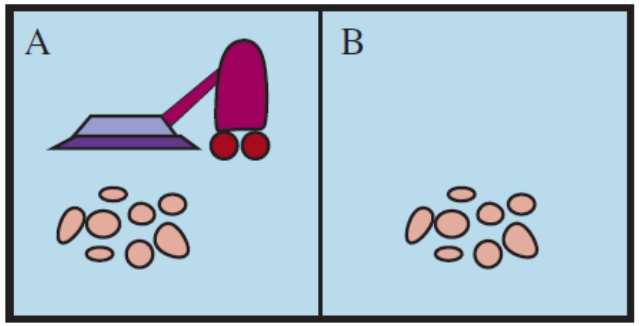
예. 로봇 청소기

- [조건-동작]으로 구성된 규칙의 형식으로 구현

```
if dirty then suck
```

```
if car-in-front-is-braking
    then initiate-braking
```

로봇 청소기 환경



if *status* == *Dirty* then return *Suck*
else if *location* == *A* then return *Right*
else if *location* == *B* then return *Left*

단순 반사 에이전트 알고리즘

function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *rules*, a set of condition–action rules

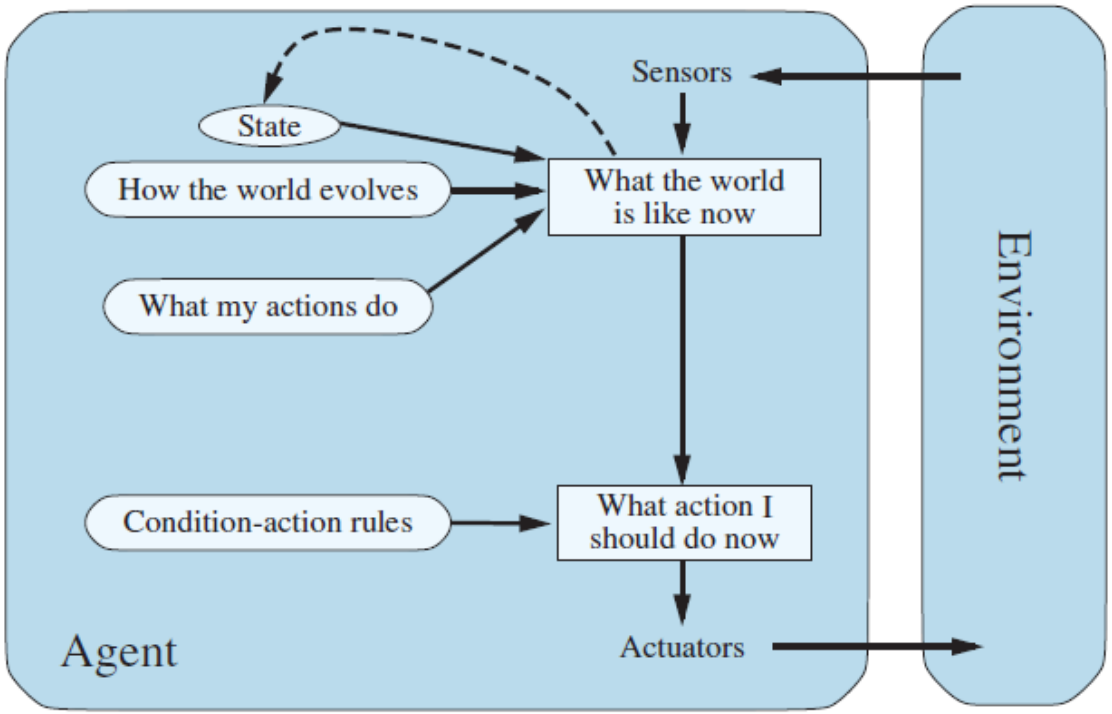
  state ← INTERPRET-INPUT(*percept*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

(*) persistent: 지속 변수 (알고리즘 과정 중 상존)

♣ 지능이 제한적임

정확한 결정을 현재 지각에만 기초해서 내릴 수 있는 경우에만,
즉 환경이 완전 관찰 가능일 때에만 작동

모델 기반 반사 에이전트



- 부분적으로 관측 가능한 환경
- 일부 정보는 내부 상태 형식으로 유지
- 지식을 통해 상태 갱신

⇩

환경을 모델링

- 환경이 어떻게 바뀌었나?
- 액션이 환경에 어떤 영향?

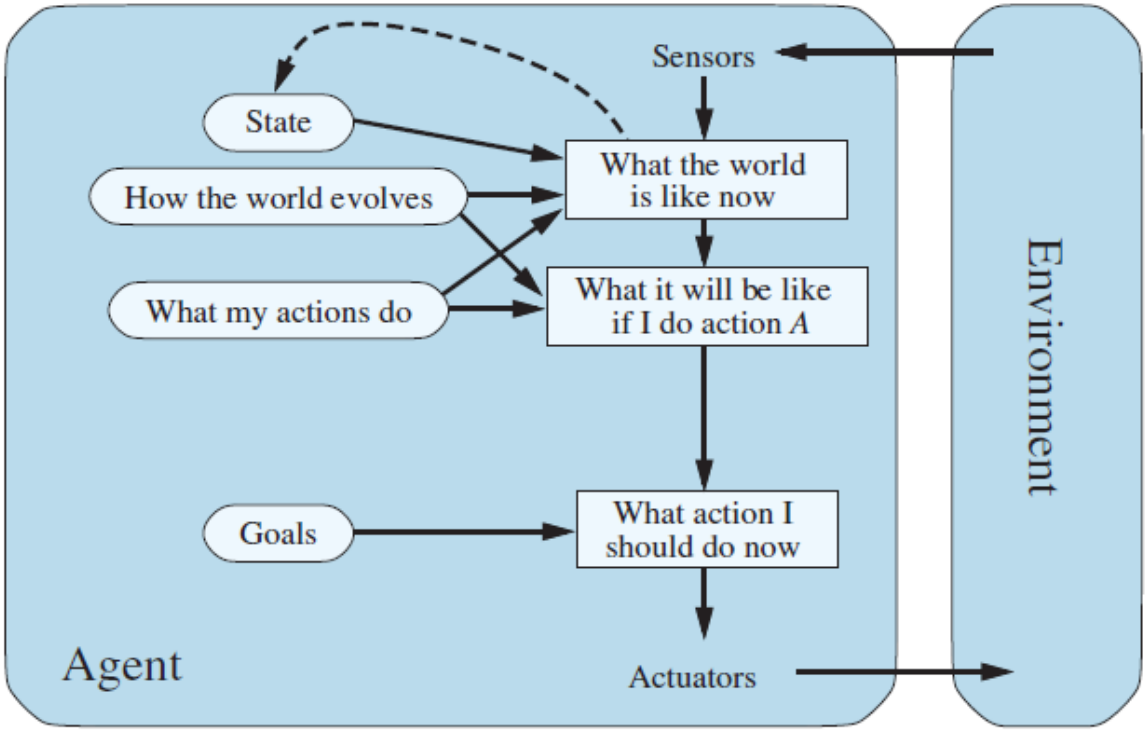
모델 기반 반사 에이전트 알고리즘

function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** an action
  persistent: *state*, the agent's current conception of the world state
    transition_model, a description of how the next state depends on the current state and action
    sensor_model, a description of how the current world state is reflected in the agent's percepts
    rules, a set of condition–action rules
    action, the most recent action, initially none

  state ← UPDATE-STATE(*state*, *action*, *percept*, *transition_model*, *sensor_model*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

- 환경의 현재 상태를 정확히 묘사하는 것은 거의 불가능
- 내부적으로 유지되는 "상태" 정보가 환경을 정확히 묘사할 필요는 없음

목표 기반 에이전트



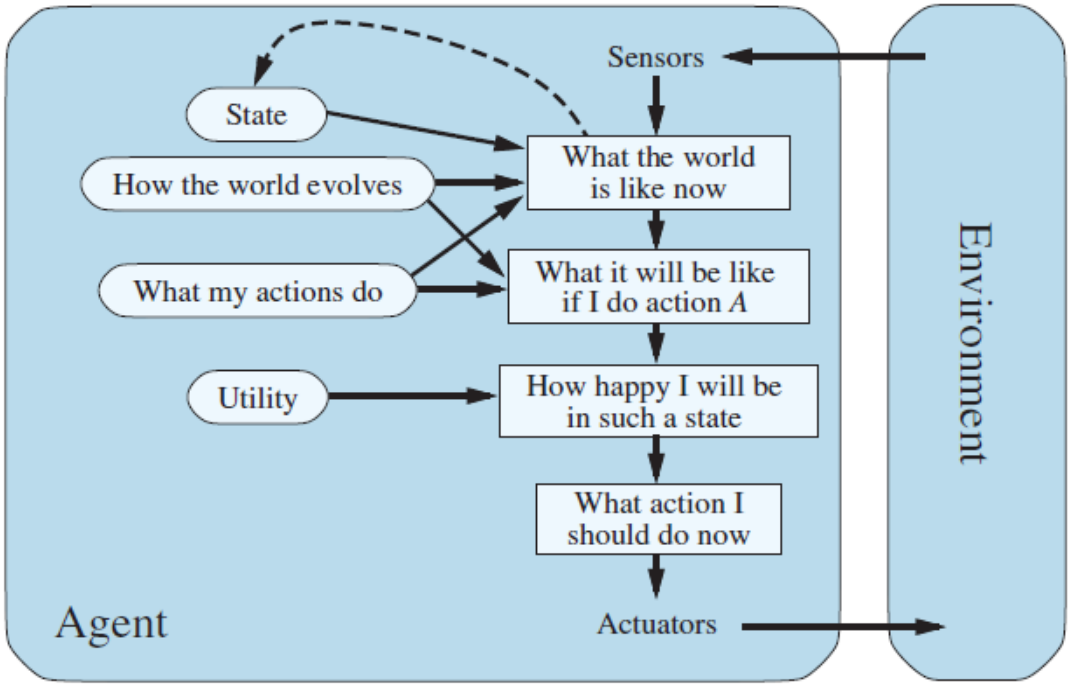
- 목표에 대한 정보 필요.
원하는 최종 상황을 설명

- *검색*과 *계획*

목표 달성까지 긴 액션 시퀀스가 필요한 경우

- 액션에 따른 미래 상황 고려 여부
(이전의 [조건-반응]형 에이전트와 다름)

효용 기반 에이전트



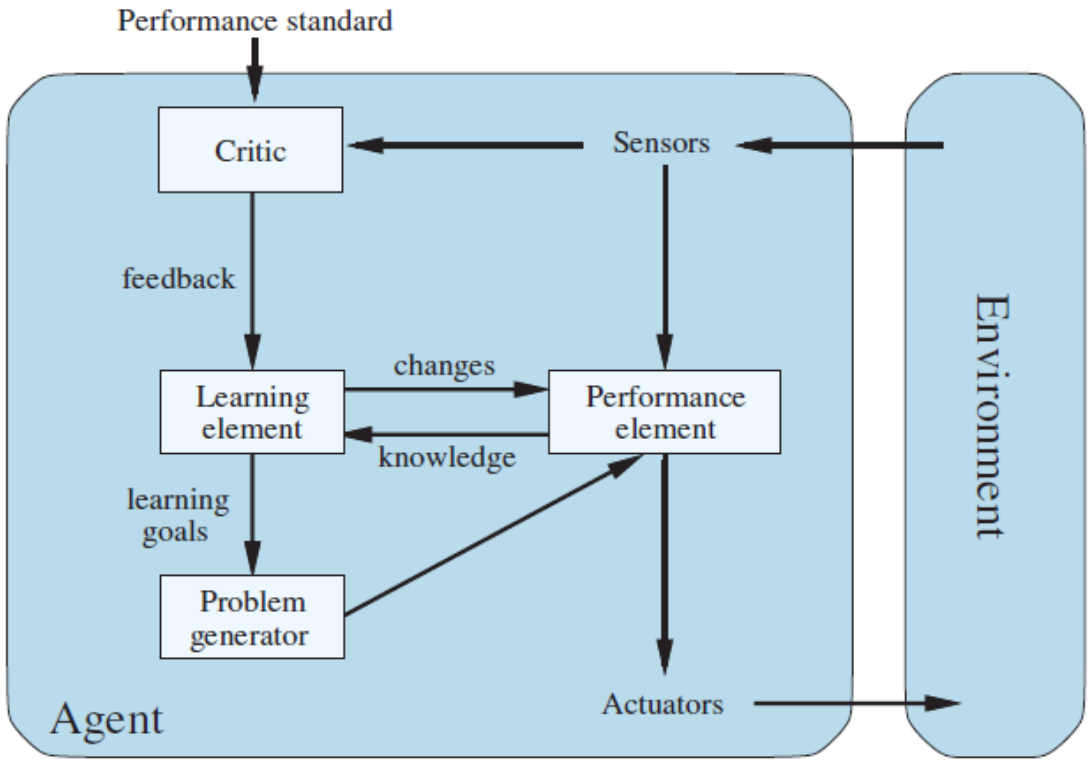
- 경우에 따라 목표 달성 방법이 달라짐
- 효용 함수를 사용하여 상태 시퀀스를 실수(효용 가치)로 매핑

↑ 내부적인 성과 측정 기준 적용

- 효용 기대값 최대화 목표

- 효용 함수(목표의 구체화):
  - 상충되는 목표 중에 효용가치 최대의 목표 선택
  - 성공 가능도와 목표 중요도를 기반으로 선택

학습형 에이전트



- 최신(SOTA) 인공지능 시스템 구현 시 선호되는 방식

  - 초기에 환경에 대한 지식이 없을 때에도 작업 수행

  - 환경 변화에 적응 - 강건성(견고성, robustness)

- 네 가지 구성 요소를 수정하여 사용 가능한 피드백과 더 가깝게 일치시킴는 것입니다.

⇩

전반적인 성능 향상

학습형 에이전트 *cont'd*

- 네 가지 구성 요소를 수정하여 사용 가능한 피드백과 더 가깝게 일치시킴

전반적인 성능 향상

- 비평가(*Critic*): 설정된 성과 기준에 따라 에이전트의 성과에 대한 피드백을 제공
- 학습 요소(*Learning element*): 성능 요소로부터 개선 사항을 도입
- 성과 요소(*Performance element*): 지각에 기반하여 행동을 선택
- 문제 생성기(*Problem generator*): 신규 정보를 경험하는 방향으로 행동 제안

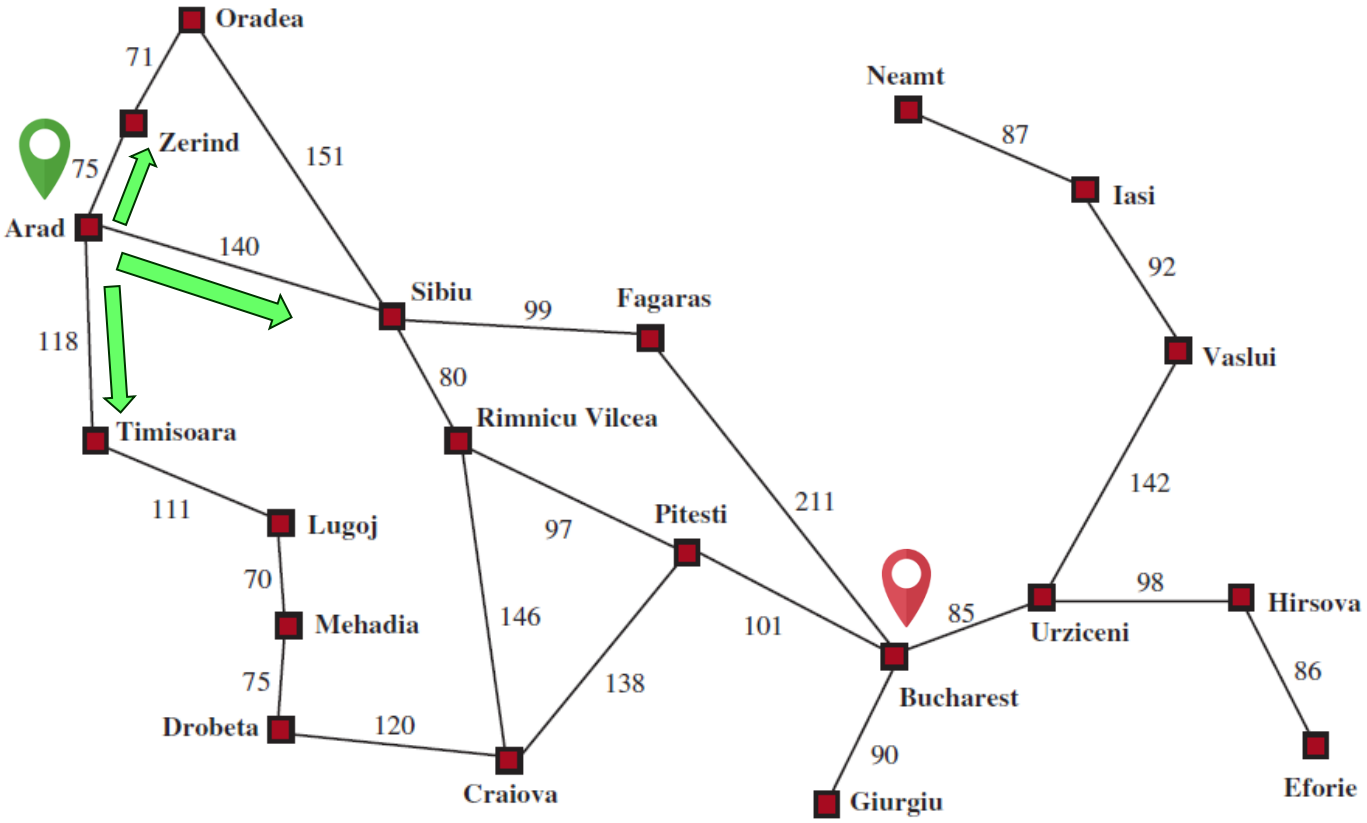
문제 해결 에이전트

♣ **반사적 에이전트**는 [상태-행동] 매핑 집합이 너무 커서 저장할 수 없을 경우 실현 불가

```
if current percepts then action
```

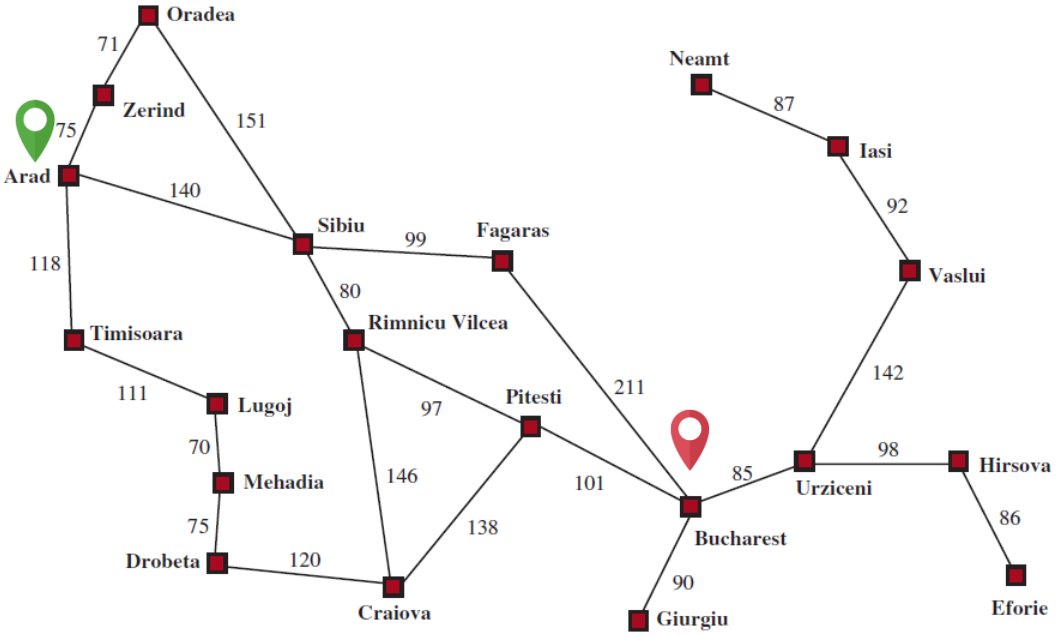
- **문제 해결 에이전트**는 **목표 중심 에이전트** 유형에 속함:
  - ♦ 환경의 상태는 그 내부 구조가 밖으로 보이지 않는 상태. 즉, 원자적 표현
- 범용 탐색(*search*) 알고리즘(BFS, DFS, 등)은 모든 일반적인 문제제 적용 가능

루마니아 관광 코스



"아라드"에서 출발하여 "부카레스트"까지 가는 방법은?

4단계 해법



1) 목표 수립

2) 문제 정의

상태: 도시

액션: 특정 도시에서 인접 도시로 이동

3) 검색

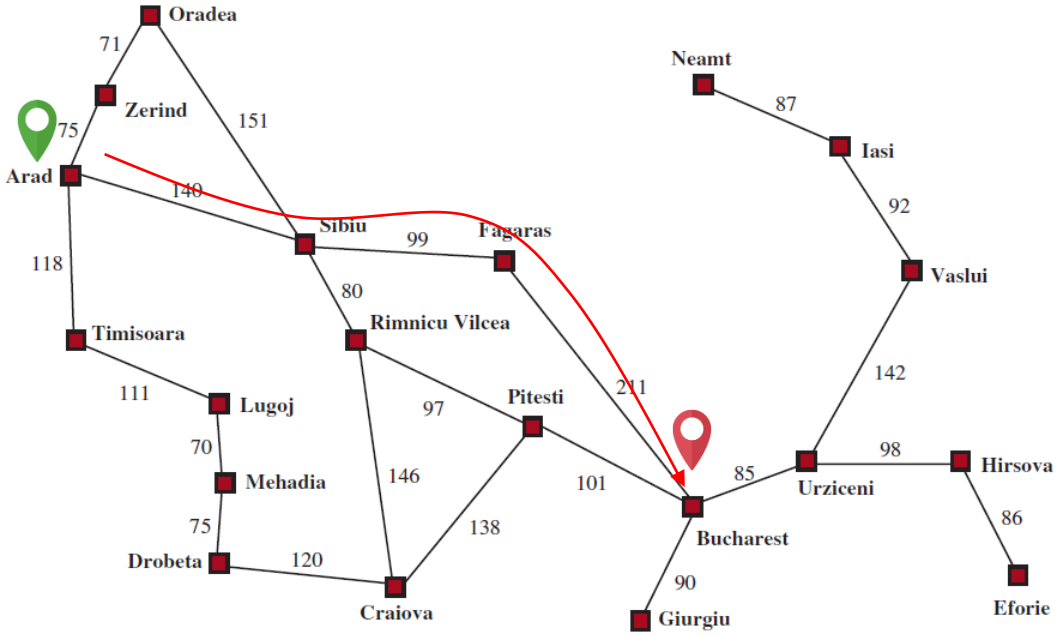
해법 발견

↑

목표에 도달하기 위한 액션 시퀀스

4) 실행

검색 문제



- 상태 공간 (그래프)
- 초기 상태(e.g., "아라드")
- 목표 상태(e.g., "부카레스트")

  `IS-GOAL("파가라스")`
- 액션

ACTIONS(s): 상태 s에서 취할 수 있는 한정된 액션 집합.

ACTIONS$(Arad) = \{ToSibiu, ToTimisoara, ToZerind\}$

- 전이 모델

RESULT$(Arad, ToZerind) = Zerind$

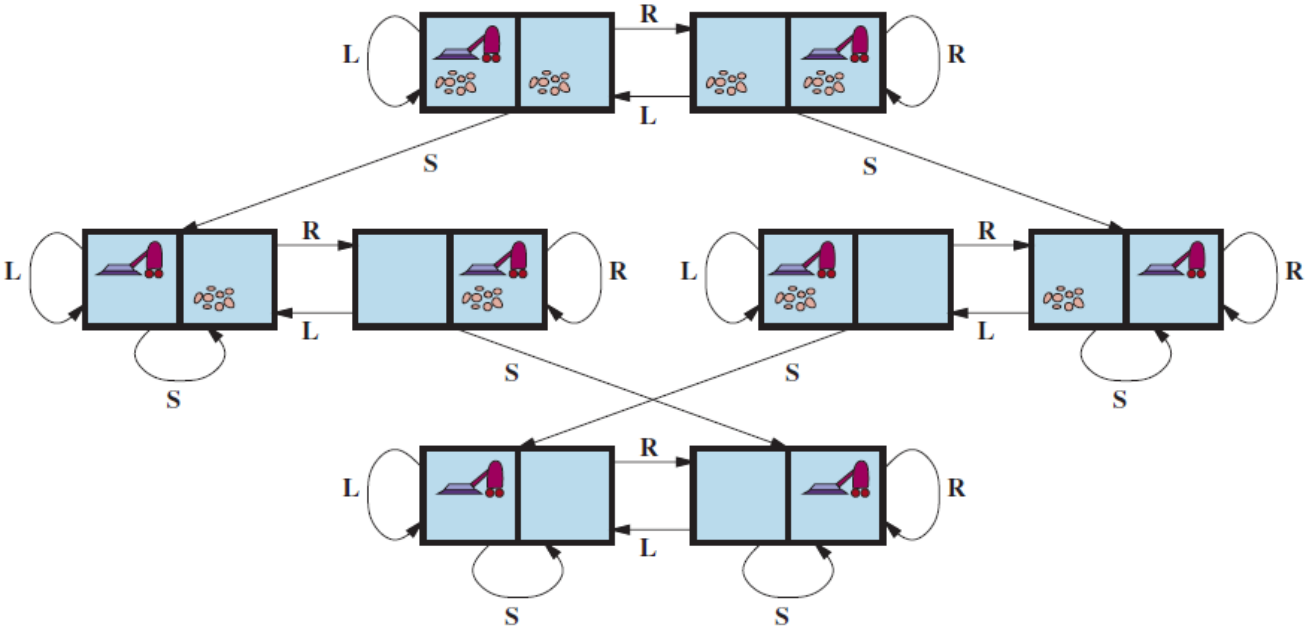
- 액션 비용 함수

$c(s, a, s')$: 상태 s에서 다음 상태 s'으로 가기 위해 액션 a를 실행할 때의 비용

- 해법: 초기상태 ⇝ 목표상태 (e.g., Arad – Sibiu – Fagaras – Bucharest)

로봇 청소기 사례

상태 공간 그래프



에이전트가 각 방에 있을 경우의 수

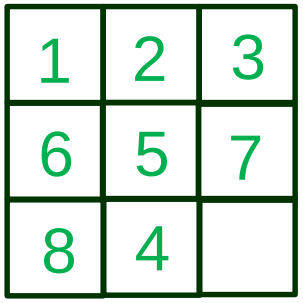
$2 \cdot 2 \cdot 2 = 8$ 가지 상태

왼쪽 방 상태에 대한 경우의 수

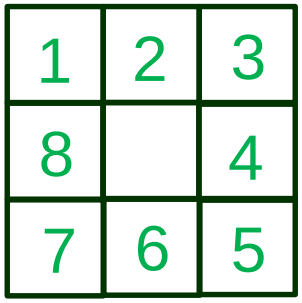
오른쪽 방 상태에 대한 경우의 수

- 액션: *Suck*, *Left*, *Right*
- 목표: 모든 방을 청소
- 비용: 각 액션 당 '1'

8 퍼즐 사례



초기 상태



목표 상태

♦ 액션: 타일 움직이기 (인접한 빈 공간으로 타일 이동)

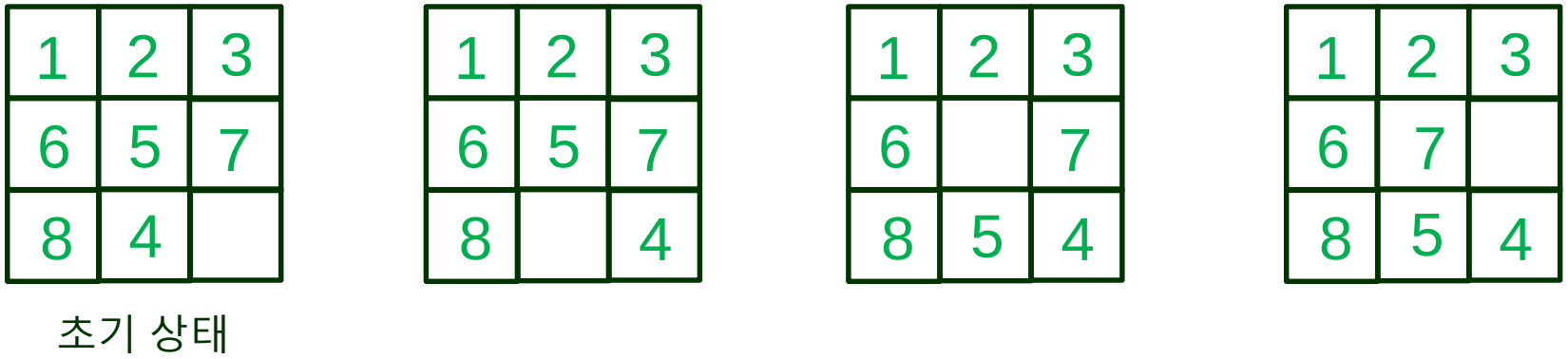
왼쪽, 오른쪽, 위로, 아래로

빈 공간의 오른쪽에 인접한 타일이 취할 수 있는 액션

초기 상태에서는 단 2 가지 액션만 가능.
즉, 오른쪽으로 이동, 아래로 이동

각 액션당 소요 비용은 1

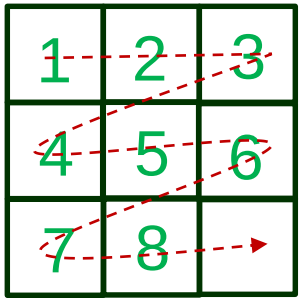
8 퍼즐의 해법


1 2 3
6 5 7
8 4
1 2 3
6 5 7
8 4
1 2 3
6 7
8 5 4
1 2 3
6 7
8 5 4
초기 상태
1 2 3
6 7 4
8 5
1 2 3
6 7 4
8 5
1 2 3
6 4
8 7 5
목표 상태
1 2 3
6 4
8 7 5
1 2 3
8 6 4
7 5
1 2 3
8 6 4
7 5
1 2 3
8 4
7 6 5

8 퍼즐, 타일의 "역전(inversion)"

모든 타일이 올바른 순서로 나타나 있는 상태: 0 역전

0 inversion



작은 숫자 타일이 항상 큰 숫자 타일 보다 앞에 위치(즉, 작은 숫자 타일이 큰 숫자 타일보다 반드시 위에 있거나, 왼쪽에 위치)

이 순서를 어길 경우를 역전(*inversion*)이라고 정의

위로부터 한 줄씩 체크. 각 줄에서는 왼쪽에서 오른쪽으로 체크. 빈 칸은 무시

♦ 작은 숫자 타일보다 큰 숫자 타일이 먼저 나오는 경우를 "역전" 횟수로 누적 계산

역전 계산

목표 상태에는 몇 개의 역전이 있는가?

1	2	3
8		4
7	6	5

순서 스캔: 1, 2, 3, 8, 4, 7, 6, 5.

6 다음에 5
7 다음에 5
7 다음에 6
8 다음에 4
8 다음에 5
8 다음에 6
8 다음에 7

7 개의 역전 존재!

8퍼즐의 해결 가능성 판정

- **정리**: 초기 상태와 목표 상태가 정의된 8퍼즐은 두 상태에 대한 역전의 개수가 서로 짝수 개 차이가 날 경우에만 해결 가능

목표 상태 ⇨

1	2	3
8		4
7	6	5

(7 개의 역전)

(*) 초기 상태의 역전 개수가 홀수 개인 경우에만 해결 가능

초기 상태 1:

4	1	2
3	5	
8	6	7

5 역전:

4 다음에 1, 2, 3

8 다음에 6, 7

⇨ 해결 가능

초기 상태 2:

4	1	2
5	3	
8	6	7

6 역전:

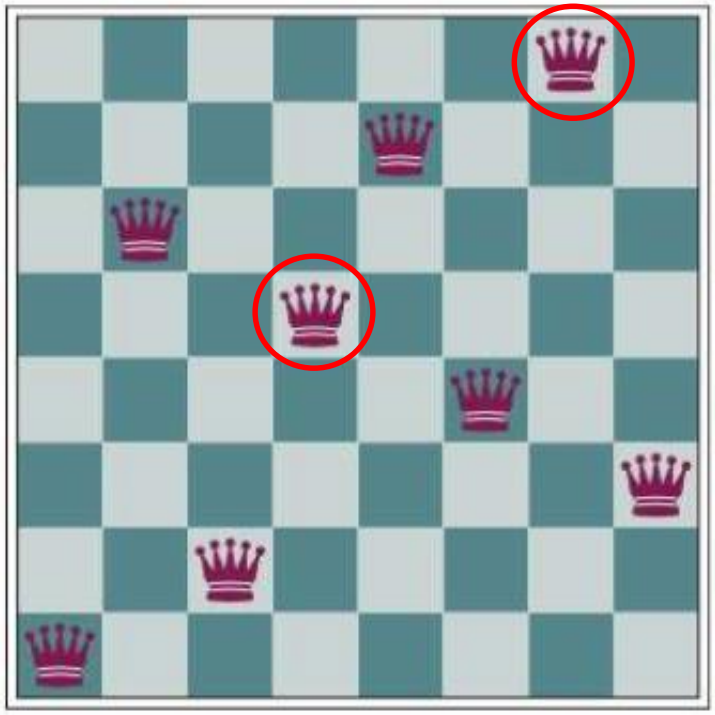
4 다음에 1, 2, 3

5 다음에 3

8 다음에 6, 7

⇨ 해결 불가능

8 퀸 문제



거의 해결한 상태의 예

- 목표 상태: 체스판에 퀸 8개를 배치하되 서로 위협하지 않는 상태로 배치

  ↓

  같은 행, 열, 대각선에 오면 안됨

- 초기 상태: 빈 보드(퀸 없음)

n-퀸 문제

$n \times n$ 체스 판에 n 개의 퀸을 상호 위협하지 않도록 배치

크누스의 추정 (1964)

도널드 크누스(스탠포드)
"알고리즘 해석의 아버지"
ACM 튜링상 수상(1974)
미국 국가 과학 메달 수상(1979)

정수 4에 일련의 제곱근, 최대 정수 함수, 팩토리얼 등의 연산을 가해 4보다 큰 임의 정수를 만들 수 있다.

(*) 최대 정수 함수
--> [2.5]=2

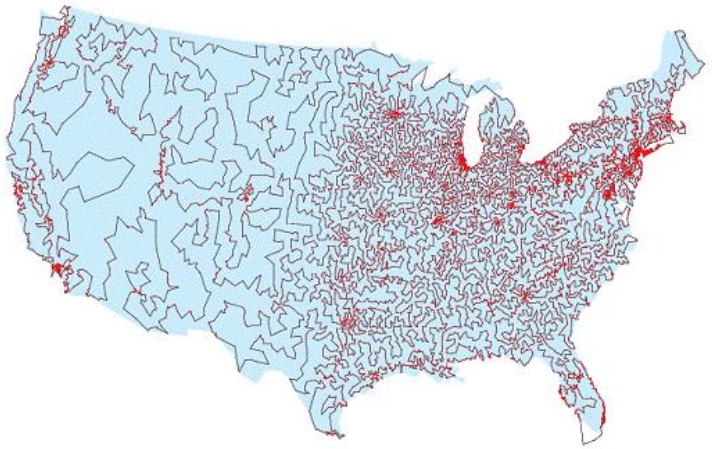
$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}} \right\rfloor = 5$$

- 상태 공간: 양의 실수
- 초기 상태: 4
- 목표 상태: 미리 정해둔 4보다 큰 정수
- 액션: 제곱근, 최대 정수 함수, 팩토리얼 연산
- 액션 비용: 1

현실적인 문제

- 길 찾기 (예: 아이오와주 에임스에서 캘리포니아주 마운틴뷰까지)
- 떠돌이 외판원 문제
- VLSI 레이아웃 설계

  작은 칩 상에 수백만 개의 구성 요소와 연결선을 배치하는 작업
- 로봇 내비게이션
- 조립 작업 순서 자동화(예: 단백질 서열 설계)



500 이상의 인구가 있는 미국 도시 13,509개를 정확히 1번씩만 방문하는 라운드트립 여행