휴리스틱 탐색
(Heuristic Search)

한경수
성결대학교 컴퓨터공학과

1

Introduction

- 무정보 탐색은 문제 정의 외에 상태에 대한 추가 정보 없이 수행하는 탐색
  - 완전성, 최적성은 만족시킬 수 있지만
  - 시간/공간 복잡도가 너무 큼
- 어떤 한 상태가 다른 상태에 비해 문제 해결에 더 도움이 될 것인지를 알고 있다면?

2

학습 목표

- 이론지식 응용 역량
  - A* 탐색 알고리즘의 동작 원리를 설명할 수 있다.
  - 주어진 휴리스틱 함수의 최적성 여부와 그 이유를 설명할 수 있다.
- 공학기술 및 도구 활용 역량
  - 주어진 문제에 A* 탐색 알고리즘을 적용하여 해를 찾을 수 있다.

3

휴리스틱 탐색

heuristic search; informed search
문제 정의 이외에 문제에 대한 지식을 사용하는 탐색 전략

4

휴리스틱

- Heuristic
- 일반적 의미
  - 시간이나 정보가 불충분하여 합리적인 판단을 할 수 없거나, 굳이 체계적이고 합리적인 판단을 할 필요가 없는 상황에서 신속하게 어림짐작하는 것
- 탐색에서의 휴리스틱
  - 최적 해를 찾는다는 보장은 없지만, 충분히 좋은 해를 찾기 위해 사용되는 경험적 지식
  - 특정 상태에서 목표 상태까지의 비용(거리)

5

휴리스틱 탐색

- 무정보 탐색
  - 너비 혹은 깊이라는 탐색 기준이 있었음
- 휴리스틱 탐색은 어떤 기준으로 탐색?
  - 가장 적절한 노드를 하나 선택
  - 적절한?
  - 평가 함수(evaluation function) 이용

6

최고 우선 탐색

- best-first search
- 평가 함수를 사용하여 확장할 노드를 선택하는 일반적 탐색 알고리즘
- 평가 함수 $f(n)$
  - 비용 추정
  - 평가 함수 값이 가장 낮은 노드를 먼저 확장
    - 확장 가능한 노드들을 평가 함수 값에 따라 우선순위 큐(priority queue)에 저장
  - 평가 함수에 따라 탐색 전략이 결정됨

한경수 7

7

최고 우선 탐색

```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure
```

한경수 8

8

최고 우선 탐색: 휴리스틱 함수

- heuristic function
- 대부분의 최고 우선 탐색 알고리즘은 평가 함수의 구성요소로 휴리스틱 함수 $h(n)$을 포함
- **노드 n 상태에서 목표 상태에 이르는 가장 저렴한 경로의 추정 비용**
  - 노드 n 상태에만 의존함
  - 예: $h(Anyang)$ = Anyang에서 목표 상태(Yeosu)까지의 직선 거리
- 탐색 알고리즘에 문제에 대한 추가 지식을 전해주는 가장 흔한 방식
- n: 목표 노드 → $h(n) = 0$

한경수 9

9

탐욕적 최고 우선 탐색

한경수 10

10

탐욕적 최고 우선 탐색

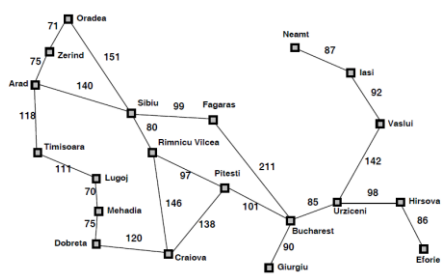
- greedy best-first search
- 목표에 가장 가까워 보이는 노드를 확장
  - $h(n)$ 값이 최소인 노드
  - 이 노드가 해결책에 빨리 도달할 가능성이 크기 때문
- $f(n) = h(n)$

한경수 11

11

예: Arad → Bucharest 길 찾기

- 직선 거리 휴리스틱 h_{SLD}을 사용한다고 가정

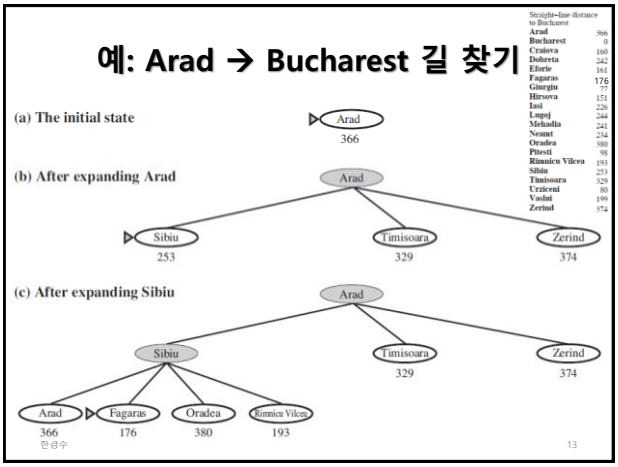


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

한경수 12

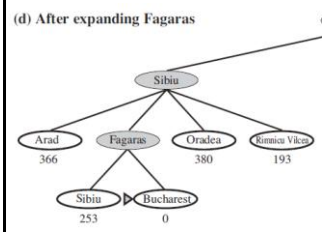
12

예: Arad → Bucharest 길 찾기



13

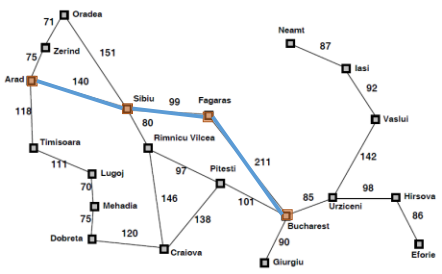
예: Arad → Bucharest 길 찾기



해결책: Bucharest ← Fagaras ← Sibiu ← Arad

14

탐욕적 최고 우선 탐색: 성능 평가



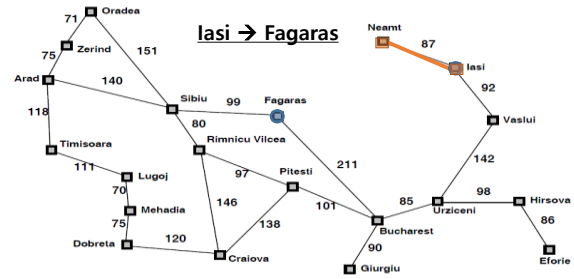
Straight-line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

- 최적이 아님

15

탐욕적 최고 우선 탐색: 성능 평가



- 확장 노드를 재탐사하지 않도록 제한할 경우, 유한 상태 공간에서 완전

16

탐욕적 최고 우선 탐색: 성능 평가

- 시간/공간 복잡도: $O(|V|)$
  - $|V|$: 상태 공간 그래프의 노드 개수
  - 적절한 휴리스틱 함수를 사용하면 복잡도를 상당히 줄일 수 있음

17

A* 탐색

18

A* 탐색

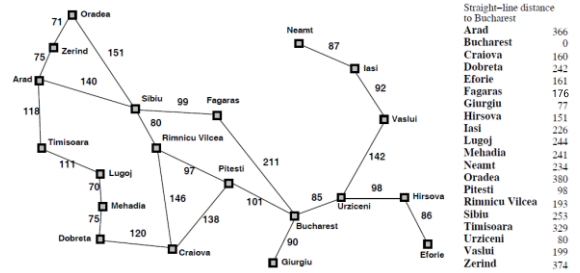
- 가장 널리 알려진 최고 우선 탐색 방법
- 평가 함수: $f(n) = g(n) + h(n)$
  - $f(n)$: 노드 n을 거치는 가장 좋은 경로(best path)의 비용 추정치 (전체 비용 추정치)
  - $g(n)$: 시작 상태로부터 노드 n까지 도달하는 데 소요된 경로 비용 (이미 투입한 비용)
  - $h(n)$: 노드 n에서 목표 상태까지 최단 경로의 비용 추정치 (남은 비용 추정치)

한경수 19

19

예: Arad → Bucharest 길 찾기

- 직선 거리 휴리스틱 h_{SLD}을 사용한다고 가정

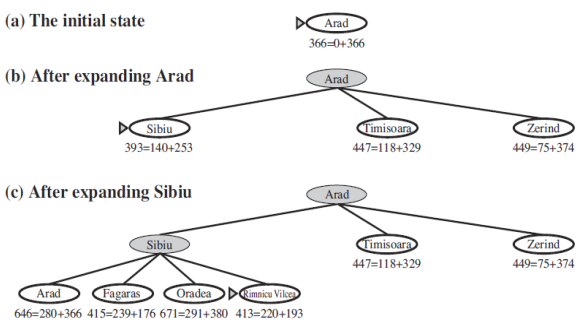


한경수 20

20

예: Arad → Bucharest 길 찾기

h_{SLD}

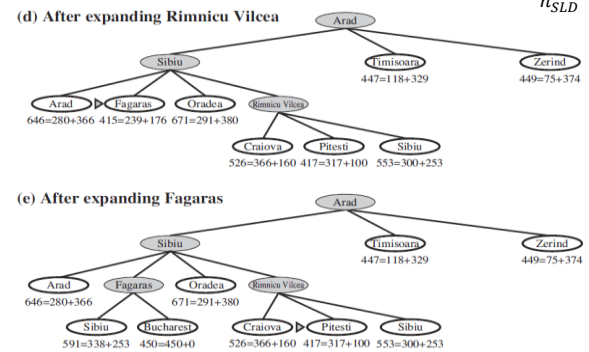


한경수 21

21

예: Arad → Bucharest 길 찾기

h_{SLD}

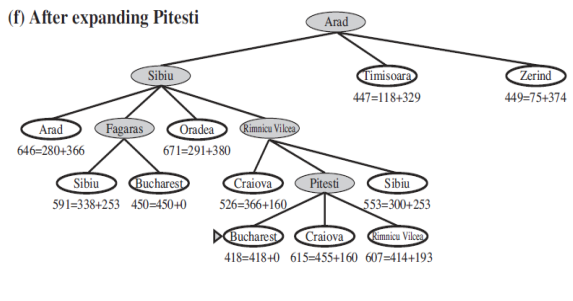


한경수 22

22

예: Arad → Bucharest 길 찾기

h_{SLD}



한경수 23

23

예: Arad → Bucharest 길 찾기

- 직선 거리 휴리스틱 h_{SLD}을 사용한다고 가정

한경수 24

24

A* 탐색: 성능 평가

- 모든 행동 비용이 $> \epsilon > 0$이고, 상태 공간에 해결책이 존재하거나 유한하면,
  완전함
- 휴리스틱 함수가 특정 조건을 만족하면,
  비용 최적임

한경수 25

25

최적 조건: 허용성(admissibility)

- $h(n)$이 허용 가능한 휴리스틱(admissible heuristic)이어야 함
- 허용 가능 휴리스틱
  - 목표에 도달하는 비용을 과대평가하지 않음
    - 실제 값보다 더 작은 값으로 추정
    - 낙관적(optimistic) 추정
  - 결국, $f(n)$이 n을 거치는 해결책의 실제 비용보다 과대평가하지 않게 됨

한경수 26

26

최적 조건: 허용성

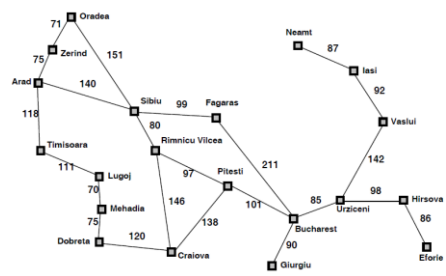
- $h(n)$이 허용 가능 → A*는 최적!
  - 허용 가능 휴리스틱 h를 사용하는 A* 알고리즘이 최적이 아닌 해결책을 리턴한다고 해보자.
    - 이 해결책 비용을 C라 하면, $C > C^*$
    - C^*: 최적 해결책 비용
  - 탐색이 종료했을 때, 최적 해결책 경로 상에 있으나 확장되지 않은 노드 n이 존재한다는 의미임
    - 노드 n이 확장되지 않았으므로, $f(n) > C^*$
    - $f(n) = g(n) + h(n) = g^*(n) + h(n) \leq g^*(n) + h^*(n)$
      - $g^*(n)$: 시작 상태부터 노드 n까지의 최적 경로 비용
      - $h^*(n)$: 노드 n에서 가장 가까운 목표까지의 최적 경로 비용
      - 허용성: $h(n) \leq h^*(n)$
    - $f(n) \leq C^*$ 모순!
      - $C^* = g^*(n) + h^*(n)$
  - 따라서 허용 가능 휴리스틱 h를 사용하는 A* 알고리즘은 최적 해결책을 리턴!

한경수 27

27

최적 조건: 허용성

- 직선 거리 h_{SLD}는 허용 가능 휴리스틱인가?



Straight-line distance to Bucharest

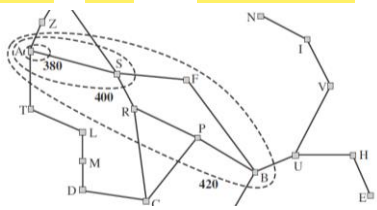
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

한경수 28

28

A* 탐색의 특징

- $f(n) < C^*$인 모든 노드를 확장함
  - C^*: 최적 해의 경로 비용
- $f(n) = C^*$인 목표 등고선(goal contour)에 있는 일부 노드를 확장한 후 목표 노드 선택



한경수 29

29

A* 탐색의 특징

- $f(n) > C^*$인 노드는 전혀 확장하지 않음
  → 부분트리를 가지치기(pruning)
  : 더 이상 살펴볼 필요 없이 고려 대상에서 제거
- 최적으로 효율적임(optimally efficient)
  - 동일한 휴리스틱 정보를 이용한다면, 루트에서 시작해서 탐색 경로를 확장하는 알고리즘 중에서, A*보다 더 적은 노드를 확장하는 최적 알고리즘은 없음

한경수 30

30

A* 탐색의 특징

- 그렇다면, A*가 모든 탐색 문제에 대한 답인가?
- 확장되는 노드의 개수가 해 길이에 따라 지수적으로 증가
  - 휴리스틱 함수가 정확할수록 증가폭이 둔화됨
- 최적해를 찾는 것은 비현실적임
  - 차선책을 신속히 찾는 A*의 변형 사용
  - 엄밀하게 허용 가능하지 않더라도 보다 정확한 휴리스틱을 설계
- 좋은 휴리스틱을 사용하면 무정보 탐색에 비해 막대한 절약을 얻을 수 있음

한경수 31

31

가중치(weighted) A* 탐색

- 휴리스틱 값에 가중치를 부여하여 확장되는 노드 수를 감소시키는 A* 탐색 방법
  - 평가 함수: $f(n) = g(n) + W \times h(n),\ W > 1$
- 최적해가 아닌 해를 찾을 수 있음
  - $C^* \sim W \times C^*$

A* 탐색 가중치 A* 탐색(W=2)

한경수 32

32

가중치 A* 탐색

- 최고 우선 탐색의 평가 함수들
  - A* 탐색: $g(n) + h(n)$ $(W = 1)$
  - 균일 비용 탐색: $g(n)$ $(W = 0)$
  - 탐욕적 최고 우선 탐색: $h(n)$ $(W = \infty)$
  - 가중치 A* 탐색: $g(n) + W \times h(n)$ $(1 < W < \infty)$
    - 다소 탐욕적인(somewhat-greedy) 탐색

한경수 33

33

A* 탐색의 특징

- A*의 가장 큰 단점은 메모리 문제
- 생성된 모든 노드를 메모리에 유지
- 대규모 문제에 적용하기 어려움
- 메모리 문제 완화를 위한 다양한 A* 탐색의 변형이 존재함

한경수 34

34

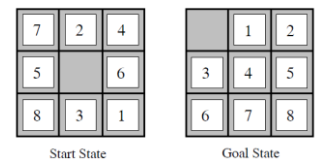
휴리스틱 함수

휴리스틱의 정확성이 성능에 얼마나 영향을 미치는가?

한경수 35

35

예: 8-퍼즐



Start State Goal State

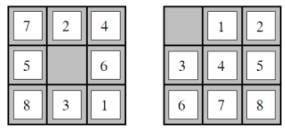
- 도달 가능한 상태 개수: $\frac{9!}{2} = 181{,}400$개
- 15-퍼즐이면, $\frac{16!}{2} \approx$ 약 10^{13}개
- 좋은 휴리스틱 함수를 찾아야 함!

한경수 36

36

예: 8-퍼즐 휴리스틱 함수

- h_1= 타일 위치가 목표 상태와 불일치하는 타일 수; 허용 가능 (예: $h_1 = 8$)
- h_2= 각 타일이 목표 위치와 떨어져 있는 거리의 총합; 허용 가능
  - 도시 블록 거리(city block distance); 맨해튼 거리(Manhattan distance): 수평/수직 거리의 합
  - 예: $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$



Start State Goal State

실제 해 비용: 26

37

탐색 비용 비교

- 예: 8-퍼즐 (사례 100개에 대한 평균)

	Search Cost (nodes generated)		
d	BFS	A*(h_1)	A*(h_2)
6	128	24	19
8	368	48	31
10	1033	116	48
12	2672	279	84
14	6783	678	174
16	17270	1683	364
18	41558	4102	751
20	91493	9905	1318
22	175921	22955	2548
24	290082	53039	5733
26	395355	110372	10080
28	463234	202565	22055

38

휴리스틱의 유용성

- h_2가 h_1보다 항상 더 좋은가?
- 모든 노드 n에 대해, $h_2(n) \geq h_1(n)$
  → h_2가 h_1보다 우세함(dominate)
- 우세는 곧바로 효율성을 의미함
  - h_2를 사용한 A*는 h_1을 사용한 A*보다 더 적은 노드를 확장하게 됨

39

좋은 휴리스틱의 조건

- 높은 값을 갖는 휴리스틱
  단,
  - 과대평가하지 않아야 함
  - 휴리스틱 계산 시간이 너무 크지 않아야 함

효율적이면서 정확성 높은 휴리스틱!

40

휴리스틱 생성

- 어떤 문제에 대해 여러 개의 허용 가능 휴리스틱 $h_1, \ldots, h_m$이 생성됐을 때, 어떤 것을 선택해야 하는가?
  - 서로 우세 관계가 없다고 가정
- 최고의 합성 휴리스틱:
  $$h(n) = \max\{h_1(n), \ldots, h_m(n)\}$$
  - 허용 가능함
  - $h_1, \ldots, h_m$보다 우세함

41

완화 문제로부터 휴리스틱 생성

- 완화 문제(relaxed problem)
  - 어떤 문제의 행동에 대한 제약을 더 완화시킨 문제
  - 예1: "8-퍼즐에서 타일은 어느 곳으로든 바로 이동 가능하다."; h_1=경로 길이
  - 예2: "8-퍼즐에서 타일은 가로/세로 방향으로(다른 타일이 있더라도) 한 칸 이동 가능하다."; h_2=경로 길이
- 완화 문제의 상태 공간 그래프는 원 상태 공간의 슈퍼그래프(supergraph)
  - 그래프의 간선이 더 추가됨

42

완화 문제로부터 휴리스틱 생성

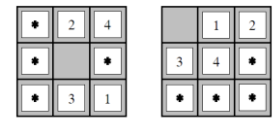
- 원 문제의 최적해는 완화 문제에서도 해이긴 하지만 최적은 아닐 수 있음
  - 추가된 간선으로 인해 완화 문제에는 더 좋은 해가 있을 수 있음
- 완화 문제에 대한 최적해 비용은 원 문제에 대한 허용 가능한 휴리스틱임

한경수 43

43

부분 문제로부터 휴리스틱 생성

- 부분 문제의 해결책 비용으로부터 허용 가능 휴리스틱을 유도할 수 있음
- 부분 문제의 최적해 비용은 전체 문제 비용의 하한값; 즉, 허용 가능 휴리스틱
- 예: 8-퍼즐 부분 문제
  - 1, 2, 3, 4 타일을 목표 위치로 이동



한경수 44

44

정리

- A* 탐색
  - 최적 조건
- 휴리스틱 함수
  - 휴리스틱의 유용성
  - 휴리스틱 생성 방법

한경수 45

45

정리 문제 풀이

한경수 46

46

정리 문제 1: A* 탐색

- 다음과 같이 주어진 8-퍼즐 문제에 A* 알고리즘을 적용하여 해를 찾고자 한다. 탐색 트리를 그리시오.

2	8	3
1	6	4
7		5

초기 상태

1	2	3
8		4
7	6	5

목표 상태

한경수 47

47