

# Foundations of Artificial Intelligence

## 11. State-Space Search: Uniform Cost Search

Malte Helmert and Gabriele Röger

University of Basel

March 15, 2017

# Foundations of Artificial Intelligence

March 15, 2017 — 11. State-Space Search: Uniform Cost Search

## 11.1 Introduction

## 11.2 Algorithm

## 11.3 Properties

## 11.4 Summary

## State-Space Search: Overview

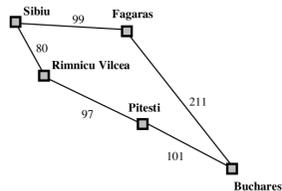
### Chapter overview: state-space search

- ▶ 5.–7. Foundations
- ▶ 8.–12. Basic Algorithms
  - ▶ 8. Data Structures for Search Algorithms
  - ▶ 9. Tree Search and Graph Search
  - ▶ 10. Breadth-first Search
  - ▶ 11. Uniform Cost Search
  - ▶ 12. Depth-first Search and Iterative Deepening
- ▶ 13.–19. Heuristic Algorithms

## 11.1 Introduction

## Uniform Cost Search

- ▶ breadth-first search optimal if all action costs equal
- ▶ otherwise no optimality guarantee  $\rightsquigarrow$  example:



remedy: **uniform cost search**

- ▶ always expand a node with **minimal path cost** ( $n.path\_cost$  a.k.a.  $g(n)$ )
- ▶ implementation: **priority queue** (min-heap) for open list

## 11.2 Algorithm

## Reminder: Generic Graph Search Algorithm

reminder from Chapter 9:

### Generic Graph Search

```

open := new OpenList
open.insert(make_root_node())
closed := new ClosedList
while not open.is_empty():
    n := open.pop()
    if closed.lookup(n.state) = none:
        closed.insert(n)
        if is_goal(n.state):
            return extract_path(n)
        for each <a, s'> in succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable
  
```

## Uniform Cost Search

### Uniform Cost Search

```

open := new MinHeap ordered by g
open.insert(make_root_node())
closed := new HashSet
while not open.is_empty():
    n := open.pop_min()
    if n.state not in closed:
        closed.insert(n)
        if is_goal(n.state):
            return extract_path(n)
        for each <a, s'> in succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable
  
```

## Uniform Cost Search: Discussion

Adapting generic graph search to uniform cost search:

- ▶ here, early goal tests/early updates of the closed list **not** a good idea. (Why not?)
- ▶ as in BFS-Graph, a **set** is sufficient for the closed list
- ▶ a tree search variant is possible, but rare: has the same disadvantages as BFS-Tree and in general **not even semi-complete** (Why not?)

Remarks:

- ▶ identical to **Dijkstra's algorithm** for shortest paths
- ▶ for both: variants with/without delayed duplicate elimination

## Uniform Cost Search: Improvements

possible improvements:

- ▶ if action costs are small integers, **bucket heaps** often more efficient
- ▶ additional early duplicate tests for generated nodes can reduce memory requirements
  - ▶ can be beneficial or detrimental for runtime
  - ▶ must be careful to keep shorter path to duplicate state

## 11.3 Properties

## Completeness and Optimality

properties of uniform cost search:

- ▶ uniform cost search is **complete** (Why?)
- ▶ uniform cost search is **optimal** (Why?)

## Time and Space Complexity

properties of uniform cost search:

- ▶ **Time complexity** depends on distribution of action costs (no simple and accurate bounds).
  - ▶ Let  $\varepsilon := \min_{a \in A} \text{cost}(a)$  and consider the case  $\varepsilon > 0$ .
  - ▶ Let  $c^*$  be the optimal solution cost.
  - ▶ Let  $b$  be the branching factor and consider the case  $b \geq 2$ .
  - ▶ Then the time complexity is at most  $O(b^{\lfloor c^*/\varepsilon \rfloor + 1})$ . (Why?)
  - ▶ often a very weak upper bound
- ▶ **space complexity** = time complexity

## 11.4 Summary

## Summary

**uniform cost search:** expand nodes in order of **ascending path costs**

- ▶ usually as a graph search
- ▶ then corresponds to Dijkstra's algorithm
- ▶ **complete** and **optimal**