

Foundations of Artificial Intelligence

9. State-Space Search: Tree Search and Graph Search

Malte Helmert and Thomas Keller

University of Basel

March 9, 2020

Foundations of Artificial Intelligence

March 9, 2020 — 9. State-Space Search: Tree Search and Graph Search

9.1 Introduction

9.2 Tree Search

9.3 Graph Search

9.4 Evaluating Search Algorithms

9.5 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

9.1 Introduction

Search Algorithms

General Search Algorithm

- ▶ Starting with **initial state**,
- ▶ repeatedly **expand** a state by **generating** its **successors**.
- ▶ Stop when a **goal state** is expanded
- ▶ or **all reachable states** have been considered.

In this chapter, we study two essential classes of search algorithms:

- ▶ **tree search** and
- ▶ **graph search**

(Each class consists of a large number of concrete algorithms.)

German: expandieren, erzeugen, Baumsuche, Graphensuche

9.2 Tree Search

Tree Search

Tree Search

- ▶ possible paths to be explored organized in a tree (**search tree**)
- ▶ **search nodes** correspond **1:1** to **paths** from initial state
- ▶ **duplicates** (also: **transpositions**) possible, i.e., multiple nodes with identical state
- ▶ search tree can have unbounded depth

German: Suchbaum, Duplikate, Transpositionen

Generic Tree Search Algorithm

Generic Tree Search Algorithm

```

open := new OpenList
open.insert(make_root_node())
while not open.is_empty():
    n := open.pop()
    if is_goal(n.state):
        return extract_path(n)
    for each <a, s'> ∈ succ(n.state):
        n' := make_node(n, a, s')
        open.insert(n')
return unsolvable

```

Generic Tree Search Algorithm: Discussion

discussion:

- ▶ **generic template** for tree search algorithms
- ↪ for concrete algorithm, we must (at least) decide how to implement the open list
- ▶ concrete algorithms often **conceptually** follow template, (= generate the same search tree), but deviate from details for efficiency reasons

9.3 Graph Search

Reminder: Tree Search

reminder:

Tree Search

- ▶ possible paths to be explored organized in a tree (**search tree**)
- ▶ **search nodes** correspond **1:1** to **paths** from initial state
- ▶ **duplicates** (also: **transpositions**) possible, i.e., multiple nodes with identical state
- ▶ search tree can have unbounded depth

Graph Search

Graph Search

differences to tree search:

- ▶ recognize **duplicates**: when a state is reached on multiple paths, only keep one search node
- ▶ **search nodes** correspond **1:1** to **reachable states**
- ▶ search tree bounded, as number of states is finite

remarks:

- ▶ some graph search algorithms do not immediately eliminate all duplicates (↪ later)
- ▶ one possible reason: find optimal solutions when a path to state s found later is cheaper than one found earlier

Generic Graph Search Algorithm

Generic Graph Search Algorithm

```

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') ∈ succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable

```

Generic Graph Search Algorithm: Discussion

discussion:

- ▶ same comments as for generic tree search apply
- ▶ in “pure” algorithm, closed list does not actually need to store the search nodes
 - ▶ sufficient to implement *closed* as set of states
 - ▶ advanced algorithms often need access to the nodes, hence we show this more general version here
- ▶ some variants perform goal and duplicate tests elsewhere (earlier) ↔ following chapters

9.4 Evaluating Search Algorithms

Criteria: Completeness

four criteria for evaluating search algorithms:

Completeness

Is the algorithm guaranteed to find a solution if one exists?

Does it terminate if no solution exists?

first property: semi-complete

both properties: complete

German: Vollständigkeit, semi-vollständig, vollständig

Criteria: Optimality

four criteria for evaluating search algorithms:

Optimality

Are the solutions returned by the algorithm always optimal?

German: Optimalität

Criteria: Time Complexity

four criteria for evaluating search algorithms:

Time Complexity

How much **time** does the algorithm need until termination?

- ▶ usually **worst case** analysis
- ▶ usually measured in **generated nodes**

often a function of the following quantities:

- ▶ **b** : (**branching factor**) of state space
(max. number of successors of a state)
- ▶ **d** : **search depth**
(length of longest path in generated search tree)

German: Zeitaufwand, Verzweigungsgrad, Suchtiefe

Criteria: Space Complexity

four criteria for evaluating search algorithms:

Space Complexity

How much **memory** does the algorithm use?

- ▶ usually **worst case** analysis
- ▶ usually measured in (concurrently) **stored nodes**

often a function of the following quantities:

- ▶ **b** : (**branching factor**) of state space
(max. number of successors of a state)
- ▶ **d** : **search depth**
(length of longest path in generated search tree)

German: Speicheraufwand

Analyzing the Generic Search Algorithms

Generic Tree Search Algorithm

- ▶ Is it complete? Is it semi-complete?
- ▶ Is it optimal?
- ▶ What is its worst-case time complexity?
- ▶ What is its worst-case space complexity?

Generic Graph Search Algorithm

- ▶ Is it complete? Is it semi-complete?
- ▶ Is it optimal?
- ▶ What is its worst-case time complexity?
- ▶ What is its worst-case space complexity?

9.5 Summary

Summary (1)

tree search:

- ▶ search nodes correspond 1:1 to paths from initial state

graph search:

- ▶ search nodes correspond 1:1 to reachable states

↔ duplicate elimination

generic methods with many possible variants

Summary (2)

evaluating search algorithms:

- ▶ completeness and semi-completeness
- ▶ optimality
- ▶ time complexity and space complexity