Basics of AI and Machine Learning

State-Space Search: Tree Search and Graph Search

Jendrik Seipp

Linköping University

Slides modified from Basel AI group, with permission

State-Space Search: Overview

Chapter overview: state-space search

- Foundations
- Basic Algorithms
    - Tree Search and Graph Search
    - Breadth-first Search
    - Uniform Cost Search
    - Depth-first Search
- Heuristic Algorithms

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.

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.)

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

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** $\langle a, s' \rangle \in$ 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

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 ($\leadsto$ 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 $\langle a, s' \rangle \in$ 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

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

Criteria: Optimality

four criteria for evaluating search algorithms:

Optimality

Are the solutions returned by the algorithm always optimal?

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

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

Summary

Summary

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