

Foundations of Artificial Intelligence

17. State-Space Search: IDA*

Malte Helmert

Universität Basel

April 1, 2016

Foundations of Artificial Intelligence

April 1, 2016 — 17. State-Space Search: IDA*

17.1 IDA*: Idea

17.2 IDA*: Algorithm

17.3 IDA*: Properties

17.4 Summary

State-Space Search: Overview

Chapter overview: state-space search

- ▶ 5.–7. Foundations
- ▶ 8.–12. Basic Algorithms
- ▶ 13.–19. Heuristic Algorithms
 - ▶ 13. Heuristics
 - ▶ 14. Analysis of Heuristics
 - ▶ 15. Best-first Graph Search
 - ▶ 16. Greedy Best-first Search, A*, Weighted A*
 - ▶ 17. IDA*
 - ▶ 18. Properties of A*, Part I
 - ▶ 19. Properties of A*, Part II

17.1 IDA*: Idea

IDA*

The main drawback of the presented best-first graph search algorithms is their space complexity.

Idea: use the concepts of iterative-deepening DFS

- ▶ bounded depth-first search with increasing bounds
- ▶ instead of **depth** we bound **f**
(in this chapter $f(n) := g(n) + h(n.state)$ as in A^*)
- ↔ **IDA*** (**iterative-deepening A^***)
- ▶ **tree search**, unlike the previous best-first search algorithms

17.2 IDA*: Algorithm

Remarks on the Algorithm (1)

- ▶ We describe an IDA* implementation with **explicit search nodes**.
- ▶ More efficient implementations leave search nodes **implicit**, as in the depth-first search algorithm in Chapter 12.

Remarks on the Algorithm (2)

- ▶ Our recursive function calls yield two values:
 - ▶ **f_limit** , the next useful f -bound for the subtree considered by the call (or **none** if a solution was found)
 - ▶ **$solution$** , the found solution (or **none**)
- ▶ More efficient implementations store these values (in instance variables of a class or in a closure) to save time for passing these values.

IDA*: Pseudo-Code (Main Procedure)

IDA*: Main Procedure

```

n0 := make_root_node()
f_limit := 0
while f_limit ≠ ∞:
    ⟨f_limit, solution⟩ := recursive_search(n0, f_limit)
    if solution ≠ none:
        return solution
return unsolvable

```

IDA*: Pseudo-Code (Depth-first Search)

```

function recursive_search(n, f_limit):
    if f(n) > f_limit:
        return ⟨f(n), none⟩
    if is_goal(n.state):
        return ⟨none, extract_path(n)⟩
    next_limit := ∞
    for each ⟨a, s'⟩ ∈ succ(n.state):
        if h(s') < ∞:
            n' := make_node(n, a, s')
            ⟨rec_limit, solution⟩ := recursive_search(n', f_limit)
            if solution ≠ none:
                return ⟨none, solution⟩
            next_limit := min(next_limit, rec_limit)
    return ⟨next_limit, none⟩

```

17.3 IDA*: Properties

IDA*: Properties

Inherits important properties of A* and depth-first search:

- ▶ **semi-complete** if h safe and $cost(a) > 0$ for all actions a
- ▶ **optimal** if h admissible
- ▶ **space complexity** $O(\ell b)$, where
 - ▶ ℓ : length of longest generated path
(for unit cost problems: bounded by optimal solution cost)
 - ▶ b : branching factor

⇒ proofs?

IDA*: Discussion

- ▶ compared to A* potentially considerable overhead because no **duplicates** are detected
 - ↪ exponentially slower in many state spaces
 - ↪ often combined with partial duplicate elimination (cycle detection, transposition tables)
- ▶ overhead due to **iterative increases** of f bound **often negligible**, but **not always**
 - ▶ especially problematic if action costs vary a lot: then it can easily happen that each new f bound only reaches a small number of new search nodes

17.4 Summary

Summary

- ▶ **IDA*** is a tree search variant of A* based on iterative deepening depth-first search
- ▶ main advantage: **low space complexity**
- ▶ disadvantage: **repeated work** can be significant
- ▶ most useful when there are **few duplicates**