

Foundations of Artificial Intelligence

17. State-Space Search: IDA*

Malte Helmert and Gabriele Röger

University of Basel

March 27, 2017

Foundations of Artificial Intelligence

March 27, 2017 — 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

```

 $n_0 := \text{make\_root\_node}()$ 
 $f\_limit := 0$ 
while  $f\_limit \neq \infty$ :
     $\langle f\_limit, solution \rangle := \text{recursive\_search}(n_0, f\_limit)$ 
    if  $solution \neq \text{none}$ :
        return  $solution$ 
return unsolvable
  
```

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

function recursive_search(n, f_limit):

```

if  $f(n) > f\_limit$ :
    return  $\langle f(n), \text{none} \rangle$ 
if  $\text{is\_goal}(n.\text{state})$ :
    return  $\langle \text{none}, \text{extract\_path}(n) \rangle$ 
 $next\_limit := \infty$ 
for each  $\langle a, s' \rangle \in \text{succ}(n.\text{state})$ :
    if  $h(s') < \infty$ :
         $n' := \text{make\_node}(n, a, s')$ 
         $\langle rec\_limit, solution \rangle := \text{recursive\_search}(n', f\_limit)$ 
        if  $solution \neq \text{none}$ :
            return  $\langle \text{none}, solution \rangle$ 
         $next\_limit := \min(next\_limit, rec\_limit)$ 
return  $\langle next\_limit, \text{none} \rangle$ 
  
```

17.3 IDA*: Properties

IDA*: Properties

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

- ▶ **semi-complete** if h safe and $\text{cost}(a) > 0$ 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**