

Foundations of Artificial Intelligence

B13. State-Space Search: IDA*

Malte Helmert

University of Basel

March 23, 2026

Foundations of Artificial Intelligence

March 23, 2026 — B13. State-Space Search: IDA*

B13.1 IDA*: Idea

B13.2 IDA*: Algorithm

B13.3 IDA*: Properties

B13.4 Summary

State-Space Search: Overview

Chapter overview: state-space search

- ▶ B1–B3. Foundations
- ▶ B4–B8. Basic Algorithms
- ▶ B9–B15. Heuristic Algorithms
 - ▶ B9. Heuristics
 - ▶ B10. Analysis of Heuristics
 - ▶ B11. Best-first Graph Search
 - ▶ B12. Greedy Best-first Search, A^* , Weighted A^*
 - ▶ B13. IDA^*
 - ▶ B14. Properties of A^* , Part I
 - ▶ B15. Properties of A^* , Part II

B13.1 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

- ▶ depth-limited search with increasing limits
- ▶ instead of **depth** we limit **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

B13.2 IDA*: Algorithm

Reminder: Iterative Deepening Depth-first Search

reminder from Chapter B8: iterative deepening depth-first search

Iterative Deepening DFS

```
for  $depth\_limit \in \{0, 1, 2, \dots\}$ :  
     $solution := depth\_limited\_search(\text{init}(), depth\_limit)$   
    if  $solution \neq \text{none}$ :  
        return  $solution$ 
```

function $depth_limited_search(s, depth_limit)$:

```
if  $is\_goal(s)$ :  
    return  $\langle \rangle$   
if  $depth\_limit > 0$ :  
    for each  $\langle a, s' \rangle \in succ(s)$ :  
         $solution := depth\_limited\_search(s', depth\_limit - 1)$   
        if  $solution \neq \text{none}$ :  
             $solution.push\_front(a)$   
            return  $solution$   
return none
```

First Attempt: IDA* Main Function

first attempt: iterative deepening A* (IDA*)

IDA* (First Attempt)

```
for  $f\_limit \in \{0, 1, 2, \dots\}$ :  
     $solution := f\_limited\_search(init(), 0, f\_limit)$   
    if  $solution \neq \mathbf{none}$ :  
        return  $solution$ 
```

First Attempt: f -Limited Search

```
function f_limited_search( $s, g, f\_limit$ ):  
  if  $g + h(s) > f\_limit$ :  
    return none  
  if is_goal( $s$ ):  
    return  $\langle \rangle$   
  for each  $\langle a, s' \rangle \in \text{succ}(s)$ :  
     $solution := \text{f\_limited\_search}(s', g + \text{cost}(a), f\_limit)$   
    if  $solution \neq \text{none}$ :  
       $solution.\text{push\_front}(a)$   
    return solution  
return none
```

IDA* First Attempt: Discussion

- ▶ The pseudo-code can be rewritten to be even more similar to our IDDFS pseudo-code. However, this would make our next modification more complicated.
- ▶ The algorithm follows the same principles as IDDFS, but takes path costs and heuristic information into account.
- ▶ For unit-cost state spaces and the trivial heuristic $h : s \mapsto 0$ for all states s , it behaves **identically** to IDDFS.
- ▶ For general state spaces, there is a problem with this first attempt, however.

Growing the f Limit

- ▶ In IDDFS, we grow the limit from the smallest limit that gives a non-empty search tree (0) by 1 at a time.
- ▶ This usually leads to exponential growth of the tree between rounds, so that re-exploration work can be amortized.
- ▶ In our first attempt at IDA*, there is no guarantee that increasing the f limit by 1 will lead to a larger search tree than in the previous round.
- ▶ This problem becomes worse if we also allow non-integer (fractional) costs, where increasing the limit by 1 would be very arbitrary.

Setting the Next f Limit

idea: let the f -limited search compute the next sensible f limit

- ▶ Start with $h(\text{init}())$, the smallest f limit that results in a non-empty search tree.
- ▶ In every round, increase the f limit to the **smallest** value that ensures that in the next round at least one additional path will be considered by the search.

↪ `f_limited_search` now returns two values:

- ▶ the next f limit that would include at least one new node in the search tree (∞ if no such limit exists; **none** if a solution was found), and
- ▶ the solution that was found (or **none**).

Final Algorithm: IDA* Main Function

final algorithm: iterative deepening A* (IDA*)

IDA*

$f_limit = h(\text{init}())$

while $f_limit \neq \infty$:

$\langle f_limit, solution \rangle := \text{f_limited_search}(\text{init}(), 0, f_limit)$

if $solution \neq \text{none}$:

return $solution$

return unsolvable

Final Algorithm: f -Limited Search

```
function f_limited_search( $s, g, f\_limit$ ):  
  if  $g + h(s) > f\_limit$ :  
    return  $\langle g + h(s), \text{none} \rangle$   
  if is_goal( $s$ ):  
    return  $\langle \text{none}, \langle \rangle \rangle$   
   $new\_limit := \infty$   
  for each  $\langle a, s' \rangle \in succ(s)$ :  
     $\langle child\_limit, solution \rangle := f\_limited\_search(s', g + cost(a), f\_limit)$   
    if  $solution \neq \text{none}$ :  
       $solution.push\_front(a)$   
      return  $\langle \text{none}, solution \rangle$   
     $new\_limit := \min(new\_limit, child\_limit)$   
return  $\langle new\_limit, \text{none} \rangle$ 
```

Final Algorithm: f -Limited Search

```
function  $f\_limited\_search(s, g, f\_limit)$ :  
if  $g + h(s) > f\_limit$ :  
    return  $\langle g + h(s), \text{none} \rangle$   
if  $is\_goal(s)$ :  
    return  $\langle \text{none}, \langle \rangle \rangle$   
 $new\_limit := \infty$   
for each  $\langle a, s' \rangle \in succ(s)$ :  
     $\langle child\_limit, solution \rangle := f\_limited\_search(s', g + cost(a), f\_limit)$   
    if  $solution \neq \text{none}$ :  
         $solution.push\_front(a)$   
        return  $\langle \text{none}, solution \rangle$   
     $new\_limit := \min(new\_limit, child\_limit)$   
return  $\langle new\_limit, \text{none} \rangle$ 
```

B13.3 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

We state these without proof.

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 limit **often negligible**, but **not always**
 - ▶ especially problematic if action costs vary a lot: then it can easily happen that each new f limit only considers a small number of new paths

B13.4 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