Foundations of Artificial Intelligence

17. State-Space Search: IDA*

Thomas Keller and Florian Pommerening

University of Basel

March 27, 2023

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

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

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 $f(n) := g(n) + h(n.\text{state})$ as in A*)

$\rightsquigarrow$ IDA* (iterative-deepening A*)

- tree search, unlike the previous best-first search algorithms

IDA*: Algorithm

Reminder: Iterative Deepening Depth-first Search

reminder: iterative deepening depth-first search

Iterative Deepening DFS

for $\mathit{depth\_bound} \in \{0, 1, 2, \dots\}$:
    $\mathit{solution}$:= depth_bounded_search(init(), $\mathit{depth\_bound}$)
    if $\mathit{solution} \neq$ **none**:
        return $\mathit{solution}$

function depth_bounded_search(s, $\mathit{depth\_bound}$):

if is_goal(s):
    return $\langle\rangle$
if $\mathit{depth\_bound} > 0$:
    for each $\langle a, s' \rangle \in$ succ(s):
        $\mathit{solution}$:= depth_bounded_search(s', $\mathit{depth\_bound} - 1$)
        if $\mathit{solution} \neq$ **none**:
            $\mathit{solution}$.push_front(a)
            return $\mathit{solution}$
return none

First Attempt: IDA* Main Function

first attempt: iterative deepening A* (IDA*)

IDA* (First Attempt)

for *f_bound* $\in \{0, 1, 2, \dots\}$:
    solution := f_bounded_search(init(), 0, *f_bound*)
    if *solution* $\neq$ **none**:
        return *solution*

First Attempt: f-Bounded Search

function f_bounded_search(s, g, f_bound):

```
if g + h(s) > f_bound:
    return none
if is_goal(s):
    return ⟨⟩
for each ⟨a, s'⟩ ∈ succ(s):
    solution := f_bounded_search(s', g + cost(a), f_bound)
    if solution ≠ none:
        solution.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 Bound

- In IDDFS, we grow the bound from the smallest bound 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 bound 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 bound by 1 would be very arbitrary.

Setting the Next f Bound

idea: let the f-bounded search compute the next sensible f bound

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

$\rightsquigarrow$ f_bounded_search now returns two values:

- the next f bound that would include at least one new node in the search tree (∞ if no such bound 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*

$f\_bound = h(\text{init}())$
while $f\_bound \neq \infty$:
    $\langle f\_bound, solution \rangle :=$ f_bounded_search(init(), 0, f_bound)
    if $solution \neq$ **none**:
        return *solution*
return unsolvable

Final Algorithm: f-Bounded Search

function f_bounded_search(s, g, f_bound):

if $g + h(s) > f\_bound$:
  return $\langle g + h(s), \textbf{none} \rangle$
if is_goal(s):
  return $\langle \textbf{none}, \langle\rangle \rangle$
$new\_bound := \infty$
for each $\langle a, s' \rangle \in \text{succ}(s)$:
  $\langle child\_bound, solution \rangle :=$ f_bounded_search($s', g + \text{cost}(a), f\_bound$)
  if $solution \neq$ **none**:
    $solution$.push_front(a)
    return $\langle \textbf{none}, solution \rangle$
  $new\_bound := \min(new\_bound, child\_bound)$
return $\langle new\_bound, \textbf{none} \rangle$

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

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 considers a small number of new paths

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