

Foundations of Artificial Intelligence

B11. State-Space Search: Best-first Graph Search

Malte Helmert

University of Basel

March 18, 2026

Foundations of Artificial Intelligence

March 18, 2026 — B11. State-Space Search: Best-first Graph Search

B11.1 Introduction

B11.2 Best-first Search

B11.3 Algorithm Details

B11.4 Reopening

B11.5 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

B11.1 Introduction

Heuristic Search Algorithms

Heuristic Search Algorithms

Heuristic search algorithms use heuristic functions to (partially or fully) determine the order of node expansion.

German: heuristische Suchalgorithmen

- ▶ this chapter: short introduction
- ▶ next chapters: more thorough analysis

B11.2 Best-first Search

Best-first Search

Best-first search is a class of search algorithms that expand the “most promising” node in each iteration.

- ▶ decision which node is most promising uses heuristics...
- ▶ ...but not necessarily exclusively.

Best-first Search

Best-first search is a class of search algorithms that expand the “most promising” node in each iteration.

- ▶ decision which node is most promising uses heuristics...
- ▶ ...but not necessarily exclusively.

Best-first Search

A best-first search is a heuristic search algorithm that evaluates search nodes with an evaluation function f and always expands a node n with minimal $f(n)$ value.

German: Bestensuche, Bewertungsfunktion

- ▶ implementation essentially like uniform cost search
- ▶ different choices of $f \rightsquigarrow$ different search algorithms

The Most Important Best-first Search Algorithms

the most important best-first search algorithms:

- ▶ $f(n) = h(n.state)$: greedy best-first search
 ~> only the heuristic counts
- ▶ $f(n) = g(n) + h(n.state)$: A*
 ~> combination of path cost and heuristic
- ▶ $f(n) = g(n) + w \cdot h(n.state)$: weighted A*
 $w \in \mathbb{R}_0^+$ is a parameter
 ~> interpolates between greedy best-first search and A*

German: gierige Bestensuche, A*, Weighted A*

~> properties: next chapters

What do we obtain with $f(n) := g(n)$?

Best-first Search: Graph Search or Tree Search?

Best-first search can be graph search or tree search.

- ▶ now: graph search (i.e., with duplicate elimination), which is the more common case
- ▶ Chapter B13: a tree search variant

B11.3 Algorithm Details

Reminder: Uniform Cost Search

reminder from Chapter B7:

Uniform Cost Search

```

open := new MinHeap ordered by g
open.insert(make_root_node())
closed := new HashSet
while not open.is_empty():
    n := open.pop_min()
    if n.state ∉ closed:
        closed.insert(n.state)
        if is_goal(n.state):
            return extract_path(n)
        for each ⟨a, s'⟩ ∈ succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable
  
```

Best-first Search without Reopening (1st Attempt)

Best-first Search without Reopening (1st Attempt)

```

open := new MinHeap ordered by f
open.insert(make_root_node())
closed := new HashSet
while not open.is_empty():
    n := open.pop_min()
    if n.state ∉ closed:
        closed.insert(n.state)
        if is_goal(n.state):
            return extract_path(n)
        for each ⟨a, s'⟩ ∈ succ(n.state):
            n' := make_node(n, a, s')
            open.insert(n')
return unsolvable

```

Best-first Search w/o Reopening (1st Attempt): Discussion

Discussion:

This is already an acceptable implementation of best-first search.

two useful improvements:

- ▶ **discard states** considered **unsolvable** by the heuristic
 \rightsquigarrow saves memory in *open*
- ▶ if multiple search nodes have identical *f* values,
use *h* to break ties (preferring low *h*)
 - ▶ not always a good idea, but often
 - ▶ obviously unnecessary if $f = h$ (greedy best-first search)

Best-first Search without Reopening (Final Version)

Best-first Search without Reopening

```

open := new MinHeap ordered by ⟨f, h⟩
if h(init()) < ∞:
    open.insert(make_root_node())
closed := new HashSet
while not open.is_empty():
    n := open.pop_min()
    if n.state ∉ closed:
        closed.insert(n.state)
        if is_goal(n.state):
            return extract_path(n)
        for each ⟨a, s'⟩ ∈ succ(n.state):
            if h(s') < ∞:
                n' := make_node(n, a, s')
                open.insert(n')
return unsolvable

```

Best-first Search: Properties

properties:

- ▶ **complete** if *h* is safe (Why?)
- ▶ **optimality** depends on *f* \rightsquigarrow next chapters

B11.4 Reopening

Reopening

- ▶ **reminder:** uniform cost search expands nodes in order of increasing g values
- ↪ guarantees that **cheapest path** to state of a node has been found when the node is expanded
- ▶ with arbitrary evaluation functions f in best-first search this does **not** hold in general
- ↪ in order to find solutions of low cost, we may want to **expand duplicate nodes** when cheaper paths to their states are found (**reopening**)

German: Reopening

Best-first Search with Reopening

Best-first Search with Reopening

```

open := new MinHeap ordered by  $\langle f, h \rangle$ 
if  $h(\text{init}()) < \infty$ :
    open.insert(make_root_node())
distances := new HashMap
while not open.is_empty():
    n := open.pop_min()
    if  $\text{distances.lookup}(n.\text{state}) = \text{none}$  or  $g(n) < \text{distances}[n.\text{state}]$ :
         $\text{distances}[n.\text{state}] := g(n)$ 
        if is_goal(n.state):
            return extract_path(n)
        for each  $\langle a, s' \rangle \in \text{succ}(n.\text{state})$ :
            if  $h(s') < \infty$ :
                 $n' := \text{make\_node}(n, a, s')$ 
                open.insert(n')
return unsolvable

```

↪ *distances* controls reopening and replaces *closed*

B11.5 Summary

Summary

- ▶ **best-first search**: expand node with minimal value of **evaluation function f**
 - ▶ $f = h$: **greedy best-first search**
 - ▶ $f = g + h$: **A^***
 - ▶ $f = g + w \cdot h$ with parameter $w \in \mathbb{R}_0^+$: **weighted A^***
- ▶ **here**: best-first search as a graph search
- ▶ **reopening**: expand duplicates with lower path costs to find cheaper solutions