Best-first search

From Wikipedia, the free encyclopedia

Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.

Judea Pearl described best-first search as estimating the promise of node *n* by a "heuristic evaluation function $f(n)$ which, in general, may depend on the description of *n*, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain."[1][2]

Some authors have used "best-first search" to refer specifically to a search with a heuristic that attempts to predict how close the end of a path is to a solution, so that paths which are judged to be closer to a solution are extended first. This specific type of search is called **greedy best-first search**[2] or **pure heuristic search**.[3]

Graph and tree search algorithms

α–β · A* · B* · Backtracking · Beam · Bellman–Ford · **Best-first** · Bidirectional · Borůvka · Branch & bound · BFS · British Museum · D* · DFS · Depth-limited · Dijkstra · Edmonds · Floyd–Warshall · Fringe search · Hill climbing · IDA* · Iterative deepening · Johnson · Jump point · Kruskal · Lexicographic BFS · Prim · SMA*

Listings

Graph algorithms · *Search algorithms* · *List of graph algorithms*

Related topics

Dynamic programming · Graph traversal · Tree traversal · Search games

v · t · e

Efficient selection of the current best candidate for extension is typically implemented using a priority queue.

The A* search algorithm is an example of best-first search, as is B*. Best-first algorithms are often used for path finding in combinatorial search. (Note that neither A* nor B* is a greedy best-first search as they incorporate the distance from start in addition to estimated distances to the goal.)

Contents [hide]

Algorithm [edit]

An algorithm implementing best-first search follows.[4]

```
OPEN = [initial state]
while OPEN is not empty or until a goal is found
do
 1. Remove the best node from OPEN, call it n.
 2. If n is the goal state, backtrace path to n (through recorded parents) and
return path.
 3. Create n's successors.
 4. Evaluate each successor, add it to OPEN, and record its parent.
done
```

Note that this version of the algorithm is not *complete*, i.e. it does not always find a possible path between two nodes, even if there is one. For example, it gets stuck in a loop if it arrives at a dead end, that is a node with the only successor being its parent. It would then go back to its parent, add the dead-end successor to the `OPEN` list again, and so on.

The following version extends the algorithm to use an additional `CLOSED` list, containing all nodes that have been evaluated and will not be looked at again. As this will avoid any node being evaluated twice, it is not subject to infinite loops.

```
OPEN = [initial state]
CLOSED = []
```

Wikipedia The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Deutsch
فارسی
Français
Italiano
日本語
Português
Русский
ไทย
Українська
Tiếng Việt
Edit links

```
while OPEN is not empty
do
 1. Remove the best node from OPEN, call it n, add it to CLOSED.
 2. If n is the goal state, backtrace path to n (through recorded parents) and
return path.
 3. Create n's successors.
 4. For each successor do:
       a. If it is not in CLOSED and it is not in OPEN: evaluate it, add it to OPEN,
and record its parent.
       b. Otherwise, if this new path is better than previous one, change its
recorded parent.
          i.  If it is not in OPEN add it to OPEN.
          ii. Otherwise, adjust its priority in OPEN using this new evaluation.
done
```

Also note that the given pseudo code of both versions just terminates when no path is found. An actual implementation would of course require special handling of this case.

Greedy BFS [edit]

Using a greedy algorithm, expand the first successor of the parent. After a successor is generated:[5]

1. If the successor's heuristic is better than its parent, the successor is set at the front of the queue (with the parent reinserted directly behind it), and the loop restarts.
2. Else, the successor is inserted into the queue (in a location determined by its heuristic value). The procedure will evaluate the remaining successors (if any) of the parent.

See also [edit]

- Beam search
- A* search algorithm
- Dijkstra's algorithm

References [edit]

1. ^ Pearl, J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984. p. 48.
2. ^ [a] [b] Russell, Stuart J.; Norvig, Peter (2003), *Artificial Intelligence: A Modern Approach* (2nd ed.), Upper Saddle River, New Jersey: Prentice Hall, ISBN 0-13-790395-2. pp. 94 and 95 (note 3).
3. ^ Korf, Richard E. (1999). "Artificial intelligence search algorithms". In Atallah, Mikhail J. *Handbook of Algorithms and Theory of Computation*. CRC Press. ISBN 0849326494.
4. ^ http://www.macs.hw.ac.uk/~alison/ai3notes/subsubsection2_6_2_3_2.html [*dead link*] Best First Search
5. ^ http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume28/coles07a-html/node11.html#modifiedbestfs Greedy Best-First Search when EHC Fails, Carnegie Mellon

External links [edit]

- Wikibooks: Artificial Intelligence: Best-First Search

Categories: Search algorithms