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
Deutsch
فارسی
日本語
Русский
Српски / srpski
ไทย
Українська
Edit links

Bidirectional search

From Wikipedia, the free encyclopedia

Bidirectional search is a graph search algorithm that finds a shortest path from an initial vertex to a goal vertex in a directed graph. It runs two simultaneous searches: one forward from the initial state, and one backward from the goal, stopping when the two meet in the middle. The reason for this approach is that in many cases it is faster: for instance, in a simplified model of search problem complexity in which both searches expand a tree with branching factor *b*, and the distance from start to goal is *d*, each of the two searches has complexity $O(b^{d/2})$ (in Big O notation), and the sum of these two search times is much less than the $O(b^d)$ complexity that would result from a single search from the beginning to the goal.

As in A* search, bi-directional search can be guided by a heuristic estimate of the remaining distance to the goal (in the forward tree) or from the start (in the backward tree).

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

Ira Pohl (1971) was the first one to design and implement a bi-directional heuristic search algorithm. Andrew Goldberg and others explained the correct termination conditions for the bidirectional version of Dijkstra's Algorithm.[1]

Contents [hide]

Description [edit]

A Bidirectional Heuristic Search is a state space search from some state s to another state t, searching from s to t and from t to s simultaneously (or quasi-simultaneously if done on a sequential machine). It returns a valid list of operators that if applied to s will give us t.

While it may seem as though the operators have to be invertible for the reverse search, it is only necessary to be able to find, given any node n, the set of parent nodes of n such that there exists some valid operator from each of the parent nodes to n. This has often been likened to a one-way street in the route-finding domain: it is not necessary to be able to travel down both directions, but it is necessary when standing at the end of the street to determine the beginning of the street as a possible route.

Similarly, for those edges that have inverse arcs (i.e. arcs going in both directions) it is not necessary that each direction be of equal cost. The reverse search will always use the inverse cost (i.e. the cost of the arc in the forward direction). More formally, if n is a node with parent p, then $k_1(p,n) = k_2(n,p)$, defined as being the cost from p to n.(Auer Kaindl 2004)

Terminology and notation [edit]

b
  the branching factor of a search tree

$k(n,m)$
  the cost associated with moving from node n to node m

$g(n)$
  the cost from the root to the node n

$h(n)$
: the heuristic estimate of the distance between the node n and the goal

s
: the start state

t
: the goal state (sometimes g, not to be confused with the function)

d
: the current search direction. By convention, d is equal to 1 for the forward direction and 2 for the backward direction (Kwa 1989)

d'
: the opposite search direction (i.e. $d' = 3 - d$)

$TREE_d$
: the search tree in direction d. If $d = 1$, the root is s, if $d = 2$, the root is t

$OPEN_d$
: the leaves of $TREE_d$ (sometimes referred to as $FRINGE_d$). It is from this set that a node is chosen for expansion. In bidirectional search, these are sometimes called the search 'frontiers' or 'wavefronts', referring to how they appear when a search is represented graphically. In this metaphor, a 'collision' occurs when, during the expansion phase, a node from one wavefront is found to have successors in the opposing wavefront.

$CLOSED_d$
: the non-leaf nodes of $TREE_d$. This set contains the nodes already visited by the search

Approaches for Bidirectional Heuristic Search [edit]

Bidirectional algorithms can be broadly split into three categories: Front-to-Front, Front-to-Back (or Front-to-End), and Perimeter Search (Kaindl Kainz 1997). These differ by the function used to calculate the heuristic.

Front-to-Back [edit]

Front-to-Back algorithms calculate the h value of a node n by using the heuristic estimate between n and the root of the opposite search tree, s or t.

Front-to-Back is the most actively researched of the three categories. The current best algorithm (at least in the Fifteen puzzle domain) is the BiMAX-BS*F algorithm, created by Auer and Kaindl (Auer, Kaindl 2004).

Front-to-Front [edit]

Front-to-Front algorithms calculate the h value of a node n by using the heuristic estimate between n and some subset of $OPEN_d'$. The canonical example is that of the BHFFA (Bidirectional Heuristic Front-to-Front Algorithm) (de Champeaux 1977/1983), where the h function is defined as the minimum of all heuristic estimates between the current node and the nodes on the opposing front. Or, formally:

$$h_d(n) = \min_i \{H(n, o_i) | o_i \in OPEN_{d'}\}$$

where $H(n, o)$ returns an admissible (i.e. not overestimating) heuristic estimate of the distance between nodes n and o.

Front-to-Front suffers from being excessively computationally demanding. Every time a node n is put into the open list, its $f = g + h$ value must be calculated. This involves calculating a heuristic estimate from n to every node in the opposing $OPEN$ set, as described above. The $OPEN$ sets increase in size exponentially for all domains with $b > 1$.

References [edit]

1. ^ Efficient Point-to-Point Shortest Path Algorithms

- de Champeaux, Dennis; Sint, Lenie (1977), "An improved bidirectional heuristic search algorithm", *Journal of the ACM* **24** (2): 177–191, doi:10.1145/322003.322004.
- de Champeaux, Dennis (1983), "Bidirectional heuristic search again", *Journal of the ACM* **30** (1): 22–32, doi:10.1145/322358.322360.
- Pohl, Ira (1971), "Bi-directional Search", in Meltzer, Bernard; Michie, Donald, *Machine Intelligence* **6**, Edinburgh University Press, pp. 127–140.
- Russell, Stuart J.; Norvig, Peter (2002), "3.4 Uninformed search strategies", *Artificial Intelligence: A Modern*

Approach (2nd ed.), Prentice Hall.

Categories: Graph algorithms | Search algorithms