BIDIRECTIONAL SEARCH

BIDIRECTIONAL SEARCH

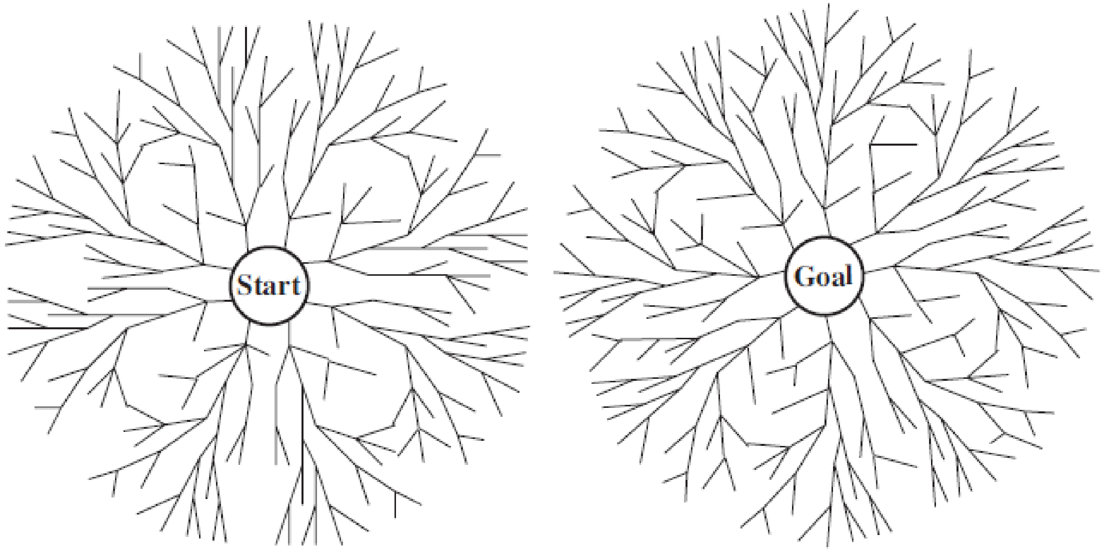

Figure 3.20 A schematic view of a bidirectional search that is about to succeed when a branch from the start node meets a branch from the goal node.

BIDIRECTIONAL BEST-FIRST SEARCH

```
function BIBF-SEARCH(problem_F, f_F, problem_B, f_B) returns a solution node, or failure
  node_F ← NODE(problem_F.INITIAL)                          // Node for a start state
  node_B ← NODE(problem_B.INITIAL)                          // Node for a goal state
  frontier_F ← a priority queue ordered by f_F, with node_F as an element
  frontier_B ← a priority queue ordered by f_B, with node_B as an element
  reached_F ← a lookup table, with one key node_F.STATE and value node_F
  reached_B ← a lookup table, with one key node_B.STATE and value node_B
  solution ← failure
  while not TERMINATED(solution, frontier_F, frontier_B) do
    if f_F(TOP(frontier_F)) < f_B(TOP(frontier_B)) then
      solution ← PROCEED(F, problem_F frontier_F, reached_F, reached_B, solution)
    else solution ← PROCEED(B, problem_B, frontier_B, reached_B, reached_F, solution)
  return solution

function PROCEED(dir, problem, frontier, reached, reached_2, solution) returns a solution
        // Expand node on frontier; check against the other frontier in reached_2.
        // The variable "dir" is the direction: either F for forward or B for backward.
  node ← POP(frontier)
  for each child in EXPAND(problem, node) do
    s ← child.STATE
    if s not in reached or PATH-COST(child) < PATH-COST(reached[s]) then
      reached[s] ← child
      add child to frontier
      if s is in reached_2 then
        solution_2 ← JOIN-NODES(dir, child, reached_2[s]))
        if PATH-COST(solution_2) < PATH-COST(solution) then
          solution ← solution_2
  return solution
```

#	Frontier (F)	Frontier (B)	Reached (F)	Reached (B)
0	Arad	Bucharest	Arad, Nil, Nil, 0	Bucharest, Nil, Nil, 0
1	Arad	Urzieeni, Giurgiu, Pitesti, Fagaras	Arad, Nil, Nil, 0	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211
2	Zerind, Timisoara, Sibiu	Urzieeni, Giurgiu, Pitesti, Fagaras	Arad, Nil, Nil, 0 Zerind, Arad, Go[Z], 75 Sibiu, Arad, Go[S], 140 Timisoara, Arad, Go[T], 118	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211
3	Timisoara, Sibiu, Oradia	Urzieeni, Giurgiu, Pitesti, Fagaras	Arad, Nil, Nil, 0 Zerind, Arad, Go[Z], 75 Sibiu, Arad, Go[S], 140 Timisoara, Arad, Go[T], 118 Oradia, Zerind, Go[O], 146	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211

#	Frontier (F)	Frontier (B)	Reached (F)	Reached (B)
4	Timisoara, Sibiu, Oradia	Giurgiu, Pitesti, Hirsova, Fagaras, Vaslui	Arad, Nil, Nil, 0 Zerind, Arad, Go[Z], 75 Sibiu, Arad, Go[S], 140 Timisoara, Arad, Go[T], 118 Oradia, Zerind, Go[O], 146	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211 Hirsova, Urzieeni, Go[H], 183 Vaslui, Urzieeni, Go[V], 227
5	Timisoara, Sibiu, Oradia	Pitesti, Hirsova, Fagaras, Vaslui	Arad, Nil, Nil, 0 Zerind, Arad, Go[Z], 75 Sibiu, Arad, Go[S], 140 Timisoara, Arad, Go[T], 118 Oradia, Zerind, Go[O], 146	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211 Hirsova, Urzieeni, Go[H], 183 Vaslui, Urzieeni, Go[V], 227
6	Timisoara, Sibiu, Oradia	Hirsova, Rimnicu Vilcea, Fagaras, Vaslui, Craiova	Arad, Nil, Nil, 0 Zerind, Arad, Go[Z], 75 Sibiu, Arad, Go[S], 140 Timisoara, Arad, Go[T], 118 Oradia, Zerind, Go[O], 146	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211 Hirsova, Urzieeni, Go[H], 183 Vaslui, Urzieeni, Go[V], 227 Rimnicu Vilcea, Pitesti, Go[RV], 198 Craiova, Pitesti, Go[C], 239

#	Frontier (F)	Frontier (B)	Reached (F)	Reached (B)
7	Sibiu, Oradia, Lugoj	Hirsova, Rimnicu Vilcea, Fagaras, Vaslui, Craiova	Arad, Nil, Nil, 0 Zerind, Arad, Go[Z], 75 Sibiu, Arad, Go[S], 140 Timisoara, Arad, Go[T], 118 Oradia, Zerind, Go[O], 146 Lugoj, Timisoara, Go[L], 229	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211 Hirsova, Urzieeni, Go[H], 183 Vaslui, Urzieeni, Go[V], 227 Rimnicu Vilcea, Pitesti, Go[RV], 198 Craiova, Pitesti, Go[C], 239
8	Oradia, Lugoj, Rimnicu Vilcea	Hirsova, Rimnicu Vilcea, Fagaras, Vaslui, Craiova	Arad, Nil, Nil, 0 Zerind, Arad, Go[Z], 75 Sibiu, Arad, Go[S], 140 Timisoara, Arad, Go[T], 118 Oradia, Zerind, Go[O], 146 Lugoj, Timisoara, Go[L], 229 Rimnicu Vilcea, Sibiu, Go[RV], 220	Bucharest, Nil, Nil, 0 Urzieeni, Bucharest, Go[U], 85 Giurgiu, Bucharest, Go[G], 90 Pitesti, Bucharest, Go[P], 101 Fagaras, Bucharest, Go[F], 211 Hirsova, Urzieeni, Go[H], 183 Vaslui, Urzieeni, Go[V], 227 Rimnicu Vilcea, Pitesti, Go[RV], 198 Craiova, Pitesti, Go[C], 239

BIDIRECTIONAL SEARCH (FINAL SOLUTION)

- Join Nodes (Forward, Rimnicu Vilcea, [Rimnicu Vilcea, Pitesti, Go[RV], 198]

- Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest

TIME COMPLEXITY

- Time complexity = $O(b^{d/2} + b^{d/2})$
- Space complexity = $O(b^{d/2})$

COMPARING UNINFORMED SEARCH ALGORITHMS

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes^{a}	Yes^{a,b}	No	No	Yes^{a}	Yes^{a,d}
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes^{c}	Yes	No	No	Yes^{c}	Yes^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^{a} complete if b is finite; ^{b} complete if step costs $\geq \epsilon$ for positive ϵ; ^{c} optimal if step costs are all identical; ^{d} if both directions use breadth-first search.