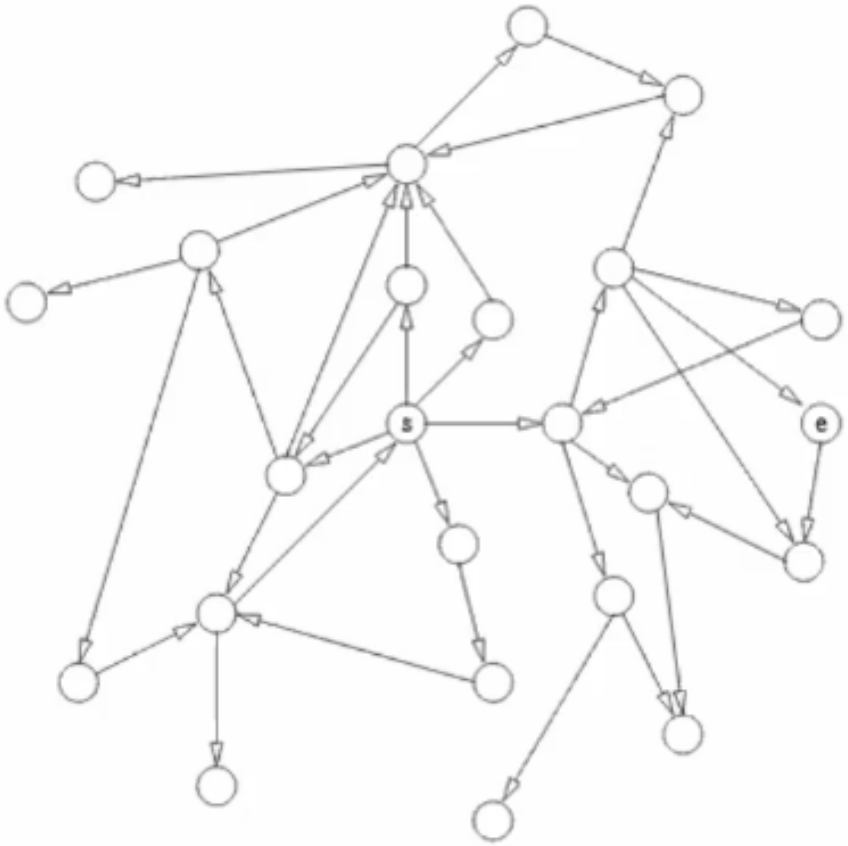
backtracking with heuristics

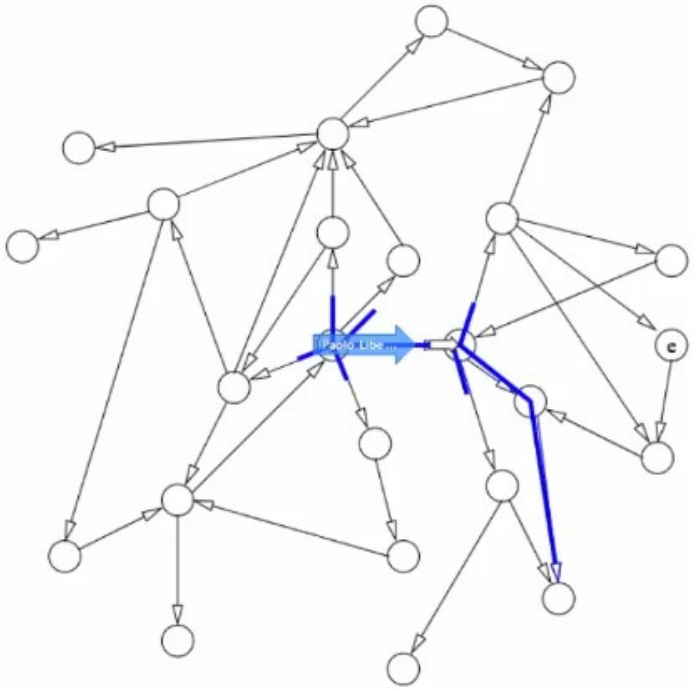


no heuristics

```
backtracking(s) {
        if (s==e)
                return reachable
        for each arrow s→s'
                if backtracking(s')
                        return reachable
        return fail
}
```

heuristics

in the cycle `for each arrow s→s'`
consider first the `s'` that is furthest on the right,
then the furthest among the remaining ones, etc.

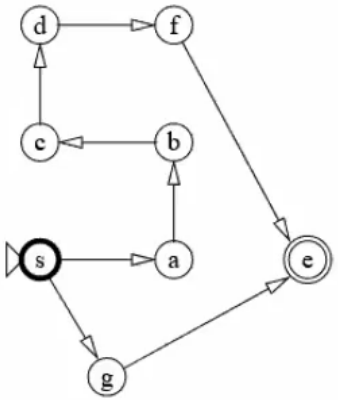


heuristics (domain dependent): go to the right

go as much as possible on the right
remember the other arrows

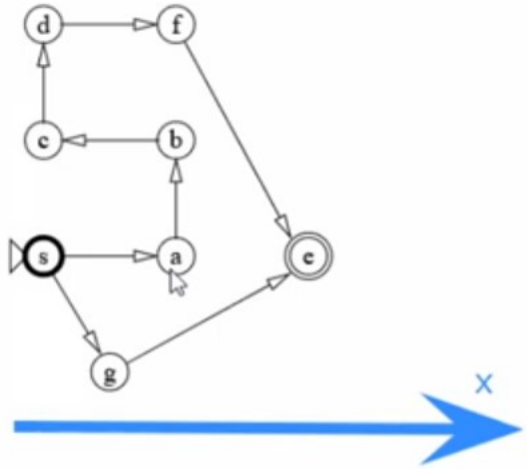
when stuck: go back and follow another arrow

optimality



start from s
always move as much right as possible

best initial move

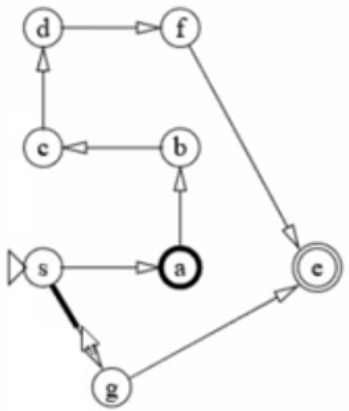


choose between s→a and s→g

a is closer to the goal than g
go to a

remember road not taken a→g

forced move



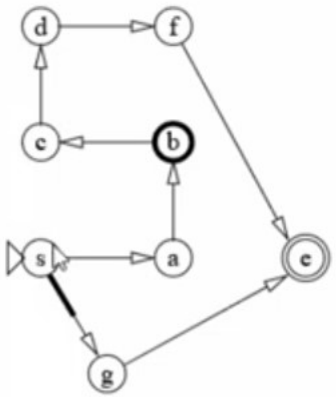
h(s) = e.x - s.x

x

moved to a
stored s→g as an unexplored alternative

from a, only possible move is to b

other forced moves



h(s) = e.x - s.x

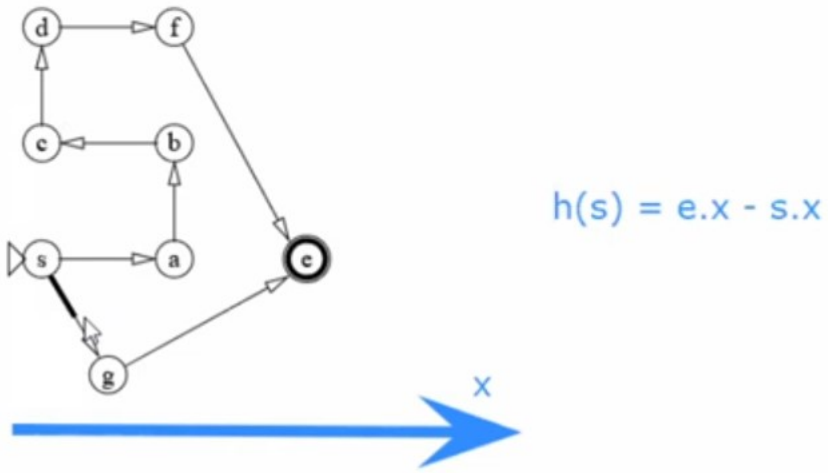
x

from b, only possible move is to c
and then: c→d→f→e

goal reached, but not optimally
shortest path was s→g→e

first path to the goal may not be optimal solution?

keep searching

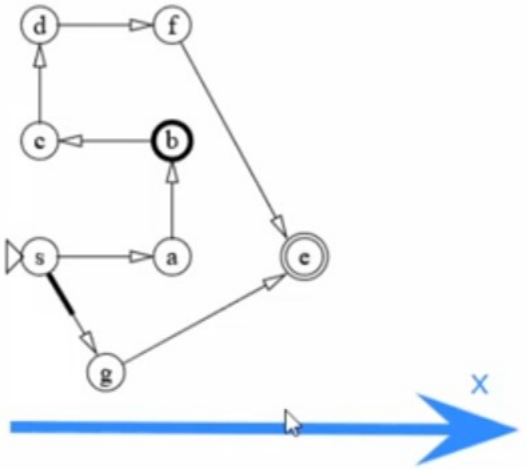


h(s) = e.x - s.x

after the goal is reached,
go back and consider all alternative roads

in this case, only s→g
this is optimal

variant: do not go left



h(s) = e.x - s.x

after s→a→b the only possible move is to b→c

but c is worse than b
further to the goal
the arrow b→c points leftwards

suspend following this path and reconsider previous alternatives
in this case, s→g

do not move left: algorithm

at each step, choose the best successor
(according to the heuristics)

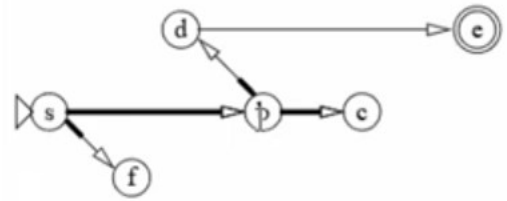
remember the other successors as alternatives

if the successor is worse than the current node
(according to the heuristics):

- do not go there
- choose the best in the set of previous alternatives
  (this may actually go left! next slide)

still not guaranteed to find optimal solutionso

when to move left



the "do not move left" algorithm may actually move left

it first tries `s→b→c`
alternatives left back: `s→f` and `b→d`

no exit from `c`
use the alternatives

`d` is on the right of `f`
follow `b→d`
even if it is a left-pointing arrow
and the right pointing `s→f` exists

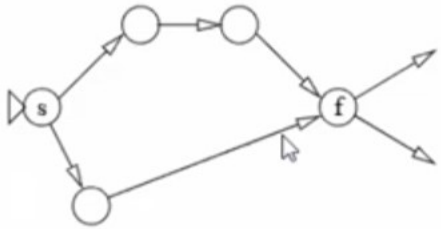
greedy best-first search

algorithms that immediately go to the successor that is the closest to the goal according to the heuristics

- do not do backtracking, or
- do backtracking, or
- also go back when the heuristic distance from goal of successors is greater than that of current node
  (in the example, "do not go left")
- reopen or do not reopen

greedy = avido
hasty = frettoloso

reopen

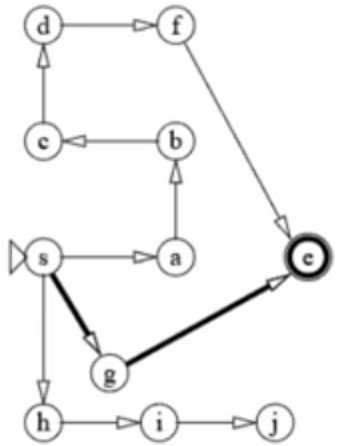


the same node `f` is reached from different paths

when reaching it the second time:

do not reopen
    consider the node `f` already visited
    do not go there
reopen
    remember length of path reaching each node
    check if the new path to `f` is shorter than the old

improvement: branch and bound



after the solution a→g→e is found
remember that this solution has two steps only

keep searching

after a→h→i, no need to continue to j
solution would have three steps or more

also: a solution has not been found but one of cost $\leq c$ is known to exist