Informed State Space Search

A4B33ZUI, LS 2017

Branislav Bošanský, Karel Horák, Ondřej Vaněk
{name.surname}@agents.fel.cvut.cz
Artificial Intelligence Center, Czech Technical University

Informed Search Problems

- **Problem**
- **Initial state** – s_0
- **Successor function** – $x \in S \ \rightarrow succ(x) \in 2^S$
- **Goal test** –$x \in S \ \rightarrow goal(x) = T \mid F$
- **Arc cost** –$c(x, succ(x))$

Heuristic $\boldsymbol{s \in S: h(s) \rightarrow R}$

- g(s): cost to reach the state s
- h(s): **estimated** cost to get from state s to goal state

Best-first Search Algorithms

- Evaluation function f(n) for each state/node

$$f(n) = g(n) + h(n)$$

COST HEURISTIC

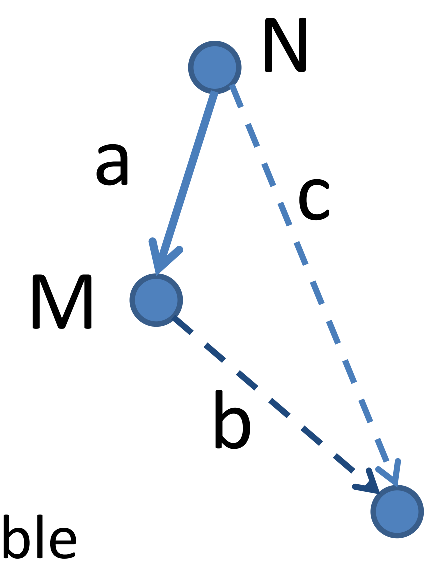
- → Selecting **best** node first – “best-first search”

Uniform cost search: h(N) = 0
Greedy search: g(N) = 0, h(N) arbitrary
A search: g(N), h(N) arbitrary
A* search: g(N), h(N) admissible

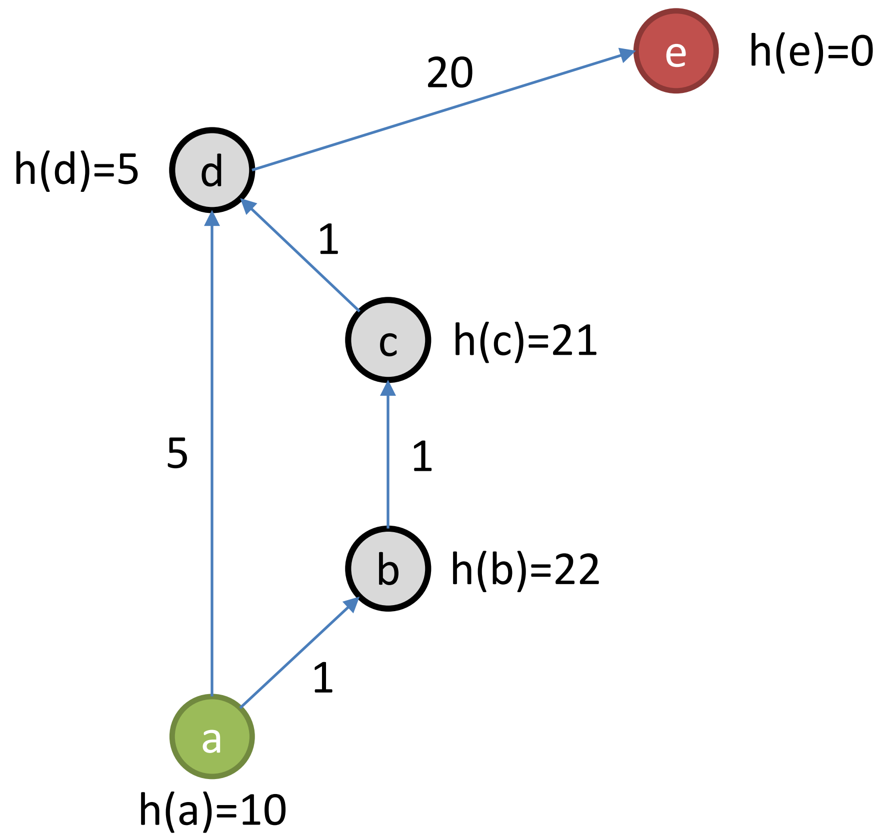
H(N) – Heuristic function

- We know the cost to the node g(n) – nothing to tune here
- We don't know the exact cost from n to goal h(n) – if we knew, no need to search – **estimate it!**

- H(N) – **admissible** and **consistent** heuristic
- Admissible = *optimistic* – it never overestimates the cost to the goal
  - $0 \leq h(N) \leq h^*(N)$
- Consistent = Triangle inequality is valid
  - $a + b \geq c$
  - $g(N, M) + h(M) \geq h(N)$
  - ➔ once a node is expanded, the cost by which it was reached is the lowest possible



Consistent Heuristic



Modifications of A*

- A* finds optimal path for admissible heuristic *h*.
- A* is optimally efficient for any heuristic *h*.

- But…

- The number of expanded nodes might still be huge.

$$O\left(b^{d}\right)$$

(*b* – branching factor, *d* – depth of optimal solution)

Modifications of A*

- **Question:** What if we do have limited memory and the goal is deep?

Modifications of A*

- **Question:** What if we do have limited memory and the goal is deep?

- IDA* (Iterative Deepening A*)
  - Space complexity linear in depth
  - Lot of work done multiple time (despite same time complexity)
  - Inefficient use of available memory

Modifications of A*

- **Question:** What if we do have limited memory and the goal is deep?

- IDA* (Iterative Deepening A*)
  - Space complexity linear in depth
  - Lot of work done multiple time (despite same time complexity)
  - Inefficient use of available memory

- SMA* (Simplified Memory-Bounded A*)
  - Works as A* until free memory is available
  - Then it drops node that is the least likely to lead to solution

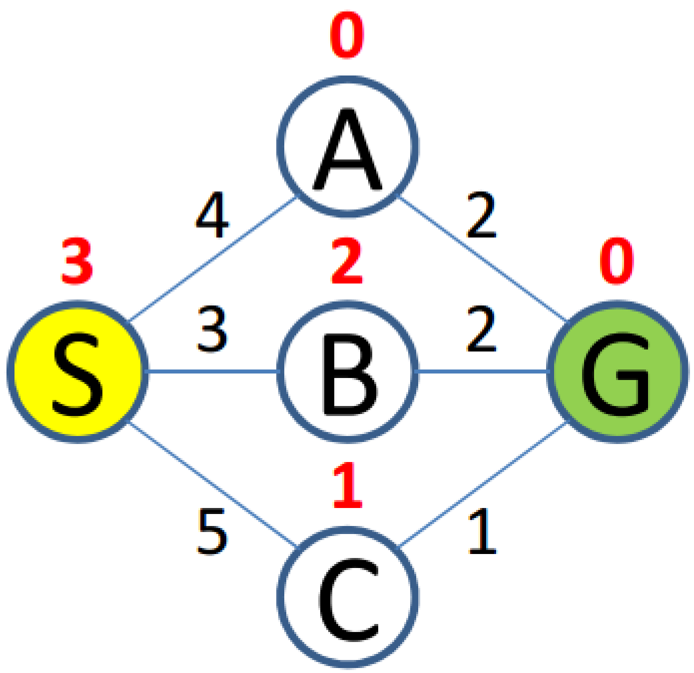
Simplified Memory-Bounded A*

- Ideas:
  - Generate one node at a time by expanding node with the least *f*-value (the “most promising node”)
  - If the expansion exhausted memory, forget node with greatest *f*-value (the “least promising node”)

  - Values are backed up:
    - Refined version of *f* when all children of a node are expanded
    - Values of “forgotten” alternative

D		X			S
		X		X	
		X			
		X			
		X	X		

Simplified Memory-Bounded A*



Example from: https://dtai.cs.kuleuven.be/education/ai/Exercises/Session2/Solutions/solution.pdf

Modifications of A*

- **Question:** What if the time for planning is bounded? e.g.
  - graph is huge like in your assignment
  - you need to plan really quickly (e.g. in real-time games)

Modifications of A*

- **Question:** What if the time for planning is bounded? e.g.
  - graph is huge like in your assignment
  - you need to plan really quickly (e.g. in real-time games)

- We can sacrifice optimality for performance.
- We can sacrifice **admissibility** of the heuristic.

Modifications of A*

- **Question:** What if the time for planning is bounded? e.g.
  - graph is huge like in your assignment
  - you need to plan really quickly (e.g. in real-time games)

- We can sacrifice optimality for performance.
- We can sacrifice **admissibility** of the heuristic.

- Lower number of expanded nodes.
- Faster search

Modifications of A*

- **ϵ-admissible heuristic** ($\epsilon > 1$)

$$f(n) = g(n) + \epsilon \, . \, h(n)$$

- Using it leads to bounded approximation of the optimal path (no worse than ϵ times worse)

D		**X**			S
		X		**X**	
		X			
		X			
		X	**X**		

Bidirectional Search

- Unidirectional search - $O(b^d)$ nodes need to be expanded

- What if we search simultaneously from start in the forward sense and from the goal backwards?
  - If we search to the depth *d/2* in both direction and happen to meet in the middle, the complexity becomes $O(b^{d/2})$

- **Question:** What do we need for doing bidirectional search?

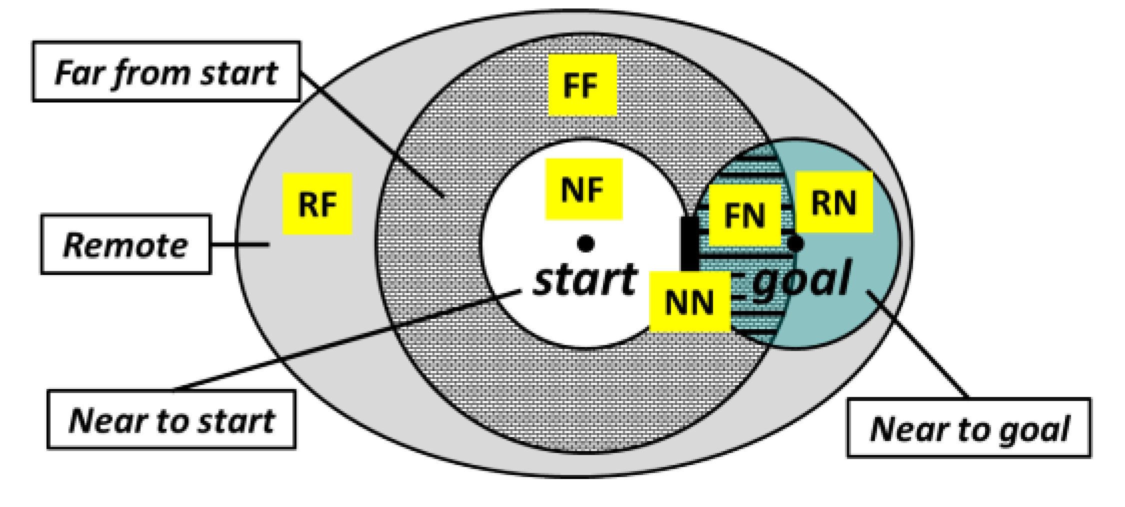
Bidirectional Search

- Heuristics in bidirectional search:

- Front-to-Front
  - Heuristics estimate distance needed to “meet” (i.e. distance to any node in the open list for opposite direction)
  - Computationally expensive
  - Proof that the searches meet in the middle missing

- Front-to-End
  - Standard heuristics
  - Distance to the goal (when searching forward) or the start (when searching backwards)

MM Search



- Is there a Front-to-End algorithm that guarantees that searches meet in the middle?
  - "Bidirectional Search That Is Guaranteed to Meet in the Middle" by Holte et al. AAAI 2016
  - Best-paper award

MM Search

- The modification is rather simple
  - $p = max\ (f(n), 2g(n))$

D		X			S
		X		X	
		X			
		X			
		X	X		

Algorithm 1: Pseudocode for MM

$g_F(start) := g_B(goal) := 0; Open_F := \{start\};$
$Open_B := \{goal\}; U := \infty$
while $(Open_F \neq \emptyset)$ *and* $(Open_B \neq \emptyset)$ **do**
  $C := min(prmin_F, prmin_B)$
  if $U \leq \max(C, fmin_F, fmin_B, gmin_F + gmin_B + \epsilon)$ **then**
    return U
  if $C = prmin_F$ **then**
    // Expand in the forward direction
    choose $n \in Open_F$ for which $pr_F(n) = prmin_F$ and $g_F(n)$ is minimum
    move n from $Open_F$ to $Closed_F$
    for ***each*** *child* c *of* n **do**
      if $c \in Open_F \cup Closed_F$ *and* $g_F(c) \leq g_F(n) + cost(n, c)$ **then**
        continue
      if $c \in Open_F \cup Closed_F$ **then**
        remove c from $Open_F \cup Closed_F$
      $g_F(c) := g_F(n) + cost(n, c)$
      add c to $Open_F$
      if $c \in Open_B$ **then**
        $U := min(U, g_F(c) + g_B(c))$
  else
    // Expand in the backward direction, analogously
return ∞