Best-First Search
Minimizing Space or Time

RTA*

Save time, non-optimal solution

Overview

◊ Do not find a complete solution, until near the goal

Overview – 2

- ◊ Do not find a complete solution, until near the goal
- ◊ Instead look ahead a fixed depth D

Overview – 3

- ◊ Do not find a complete solution, until near the goal
- ◊ Instead look ahead a fixed depth D
  - » **Generating all nodes**

Overview – 4

- Do not find a complete solution, until near the goal
- Instead look ahead a fixed depth D
  - **Generating all nodes**
- Evaluate the cost function for the tip nodes

Overview – 5

- Do not find a complete solution, until near the goal
- Instead, from node N look ahead a fixed depth D
  - **Generating all nodes**
- Evaluate the cost function for the tip nodes
- Backup the cost to the immediate successors of N

Overview – 6

- Do not find a complete solution, until near the goal
- Instead, from node N look ahead a fixed depth D
  - **Generating all nodes**
- Evaluate the cost function for the tip nodes
- Backup the cost to the immediate successors of N
- Select the best successor node S

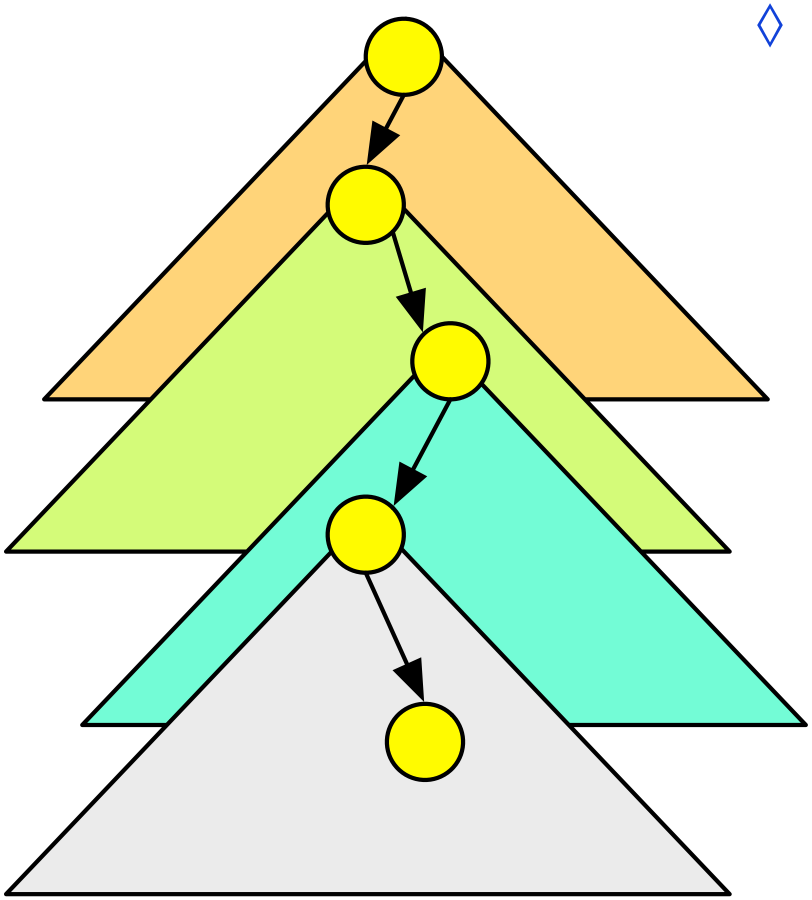
Overview – 7

- ◊ Do not find a complete solution, until near the goal
- ◊ Instead, from node N look ahead a fixed depth D
  - » **Generating all nodes**
- ◊ Evaluate the cost function for the tip nodes
- ◊ Backup the cost to the immediate successors of N
- ◊ Select the best successor node S
  - » **If S is not a goal state, then repeat**

Overview – 8

- Do not find a complete solution, until near the goal
- Instead, from node N look ahead a fixed depth D
  - **Generating all nodes**
- Evaluate the cost function for the tip nodes
- Backup the cost to the immediate successors of N
- Select the best successor node S
  - **If S is not a goal state, then repeat**
  - **If S is a goal state, then done**

Overview picture



- ◊ Two alternating stages
  - » **Planning**
    - > **Generating a tree**
    - > **Selecting most promising new state**
  - » **Executing**
    - > **Doing the action to move to the new state**

Algorithm

```
s := start_state
goal_found := false
while not goal_found do
  Plan:
      evaluate successors of s by look_ahead to depth d
      best_s := successor with minimum backed-up value
      second_best_f := f value of the second-best successor
      store s among “visited nodes”
      store f(s) := second_best_f     -- avoid looping if at s again
  Execute:
      s := best_s        -- do actions to achieve this
      if s is a goal then goal_found := true
end
```

Cost evaluation

- The cost associated with a node is the same as for A*

Cost evaluation – 2

◊ The cost associated with a node is the same as for A*

» $f(N) = g(N) + h(N)$

g (N) evaluation

◊ g (N) is evaluated with respect to the current state

g (N) evaluation – 2

- g (N) is evaluated with respect to the current state
  - **g (N_k) is the actual cost from the root, N, of the current tree**
  - **Not the original starting node S**

h (N) evaluation

◊ A node N encountered during the look ahead is assigned its heuristic h-value as

h (N) evaluation – 2

- A node N encountered during the look ahead is assigned its heuristic h-value as
  - **If goal(N) then h(N) = 0**
    - **Do not search beyond N**

h (N) evaluation – 4

- ◊ A node N encountered during the lookahead is assigned its heuristic h-value as
  - » **If goal(N) then h(N) = 0**
    - > **Do not search beyond N**
  - » **If visited(N) then h(N) = stored h(N)**
    - > **Do not search beyond N**

h (N) evaluation – 5

◊ A node N encountered during the lookahead is assigned its heuristic h-value as

- » If goal(N) then h(N) = 0
  - > Do not search beyond N
- » If visited(N) then h(N) = stored h(n)
  - > Do not search beyond N
- » If N is at the depth-search limit then
  h(N) = evaluation of the heuristic function h(N)

h (N) evaluation – 6

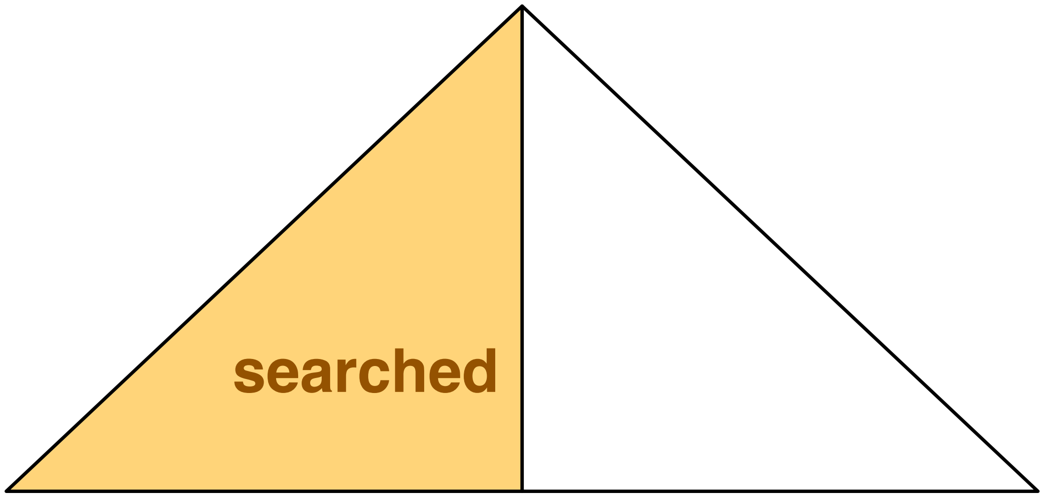
- A node N encountered during the lookahead is assigned its heuristic h-value as
  - **If goal(N) then h(N) = 0**
    - **Do not search beyond N**
  - **If visited(N) then h(N) = stored f(n)**
    - **Do not search beyond N**
  - **If N is at the depth-search limit then h(N) = evaluation of the heuristic function h(N)**
  - **If N is not at the depth-search limit then generate N's successors and backup f-value from them**

Alpha pruning

- If f(N) is monotonic, then can use alpha pruning

Alpha pruning – 2

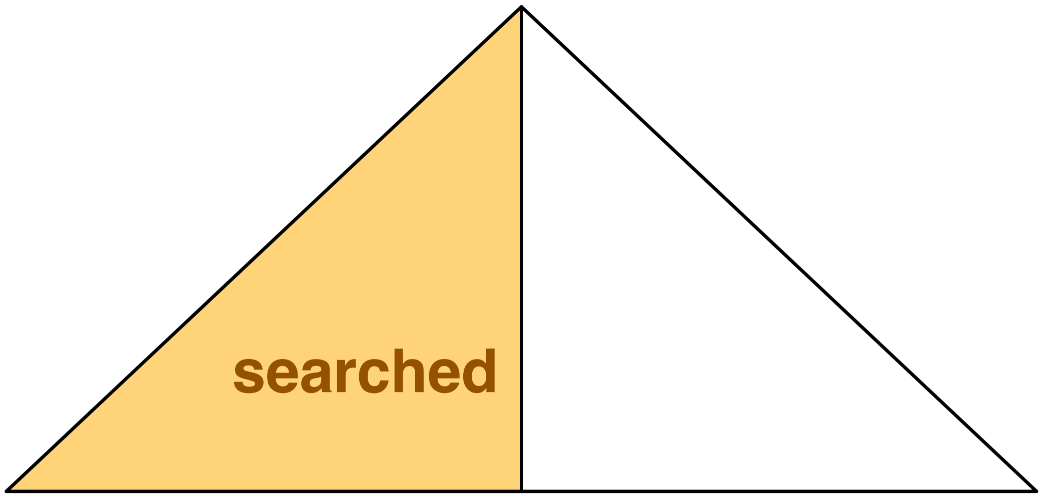
- ◊ If f(N) is monotonic, then can use alpha pruning
  - » **RTA* sorts searched horizon nodes low to high**



low f ….. high f

Alpha pruning – 3

◊ If f(N) is monotonic, then can use alpha pruning
  - » **RTA* sorts horizon nodes in sequence**
  - » **Minimum f value is alpha**

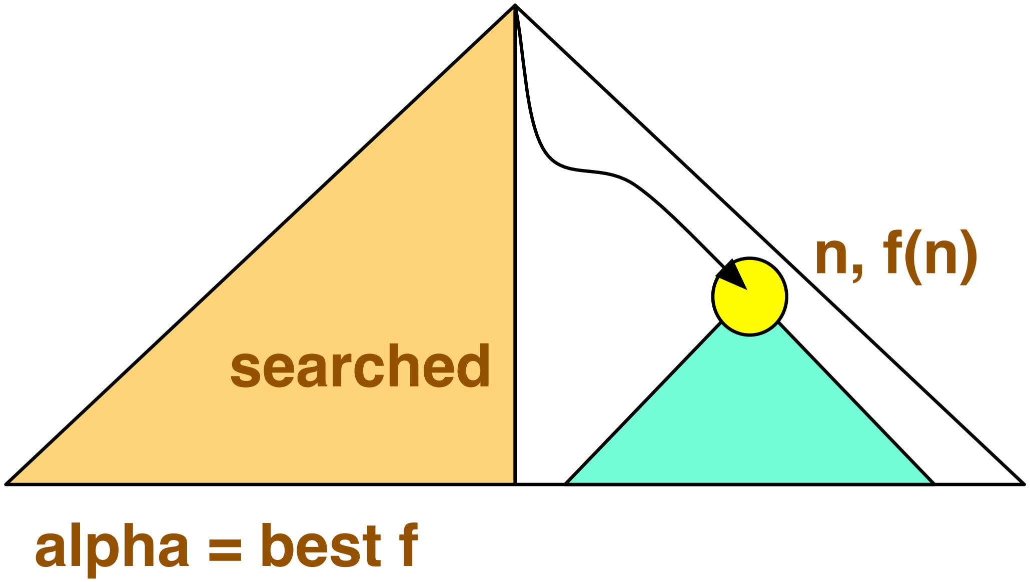


alpha = best f

Alpha pruning – 4

◊ If f(N) is monotonic, then can use alpha pruning

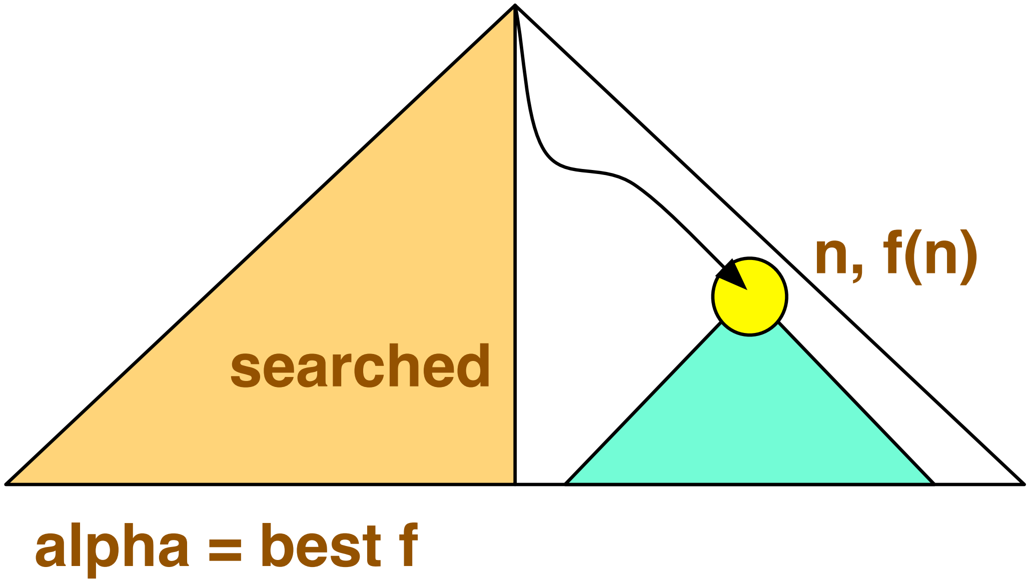
- » **RTA* sorts horizon nodes in sequence**
- » **Minimum f value is alpha**
- » **If f(n) ≥ alpha, that subtree of n can be pruned**



© Gunnar Gotshalks

Alpha pruning – 5

- If f(N) is monotonic, then can use alpha pruning
  - RTA* sorts horizon nodes in sequence
  - Minimum f value is alpha
  - If f(n) ≥ alpha, that subtree of n can be pruned
    - Cannot do better alpha


n, f(n)
searched
alpha = best f