Heuristic Search

UMaine COS 470/570 – Introduction to AI

Spring 2019

Uniformed search

Uninformed search: Time/space complexity

- Without some guidance: average case is likely to be exponential
- Can we do better by using *knowledge* to
  - prioritize nodes to expand?
  - prune some paths entirely?

Heuristic search

Heuristic search

- Use *heuristics* to search smarter
- Heuristic: “rule of thumb”, estimate, guess about
    - search space topology
    - problem domain property
    - problem-solving process itself
- *Defeasible*
- Should be easy to calculate

Heuristic search

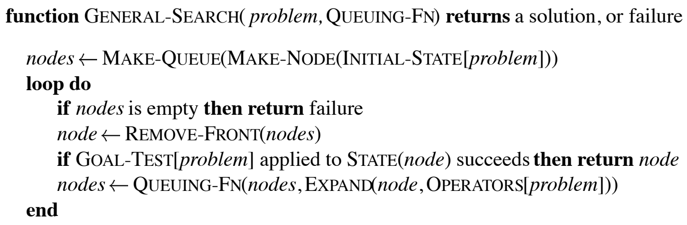
- *Heuristic function* maps state → worth
- Apply heuristic to child states
- Expand most desirable state first

Heuristic searches

- Differ in kind of information/heuristics available
    - Local information:
        - How good is *this* state?
        - How good are the *next* states
    - Global information:
        - How close is this state/next state(s) compared to goal?
        - How good is the path this/next states are on?
- Optimality or even completeness may not be guaranteed

Best-first search

- Idea: pick best node to expand next
- Recall R&N’s general algorithm for search:



```
function GENERAL-SEARCH( problem, QUEUING-FN) returns a solution, or failure
   nodes ← MAKE-QUEUE(MAKE-NODE(INITIAL-STATE[problem]))
   loop do
      if nodes is empty then return failure
      node ← REMOVE-FRONT(nodes)
      if GOAL-TEST[problem] applied to STATE(node) succeeds then return node
      nodes ← QUEUING-FN(nodes, EXPAND(node, OPERATORS[problem]))
   end
```

- Have `Queuing-Fn` pick *best* node picked first based on heuristic function

Hill-climbing

Hill-climbing

- Simple, purely local best-first search
- Analogy: real hill-climbing
  - When path branches, choose direction that increased altitude
  - May not be good: but best with available information
- Sometimes “up” is “down”: want lowest cost, e.g.
- *Gradient descent* (← neural network’s backprop)

Hill-climbing algorithms

- Let $h(s_i)$ = heuristic function, s = current state
- *Simple hill climbing:* if $h(s_i)) > h(s)$, choose s_i
- *Steepest-ascent hill-climbing*: choose *best* s_i that is better than s:
  Choose $s_m = \text{argmax}(h(s_i))$ if $h(s_m) > h(s)$

Which to choose?

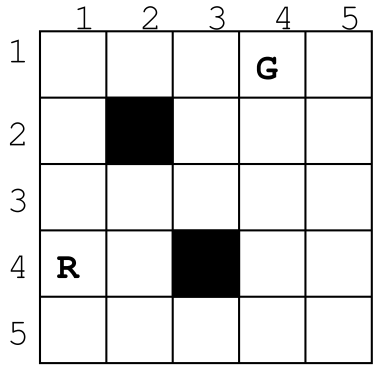
- Steepest-ascent: maybe quicker to goal
- Simple may be quicker to do: e.g., large # of children, expensive heuristic function

Save history or not?

- No history:
  - Reduce space complexity
  - But could repeat states if poor/uncertain heuristics → infinite loop
  - *Local minima* problem
- Save history:
  - If local minimum ≠ goal, can *backtrack*
  - Doesn't solve local minima problem in general…
  - …e.g., "go as far east as possible"

Hill-climbing: Simple Robot World

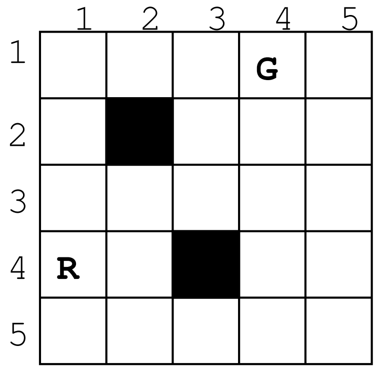
- World:



- Operators: R, L, U, D
- Heuristics?

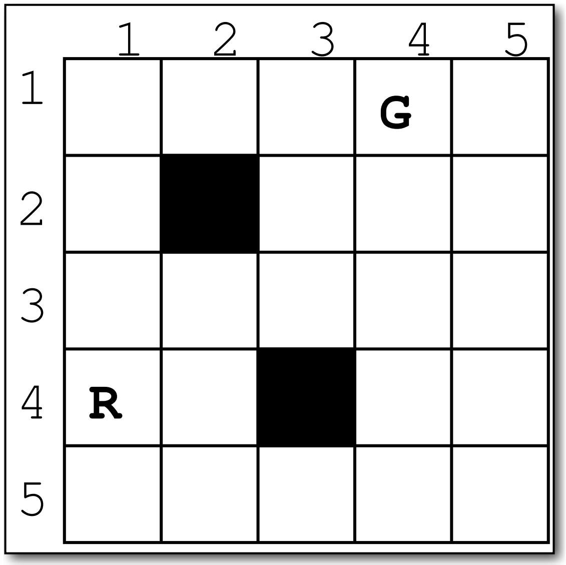
Hill-climbing: Simple Robot World

- World:

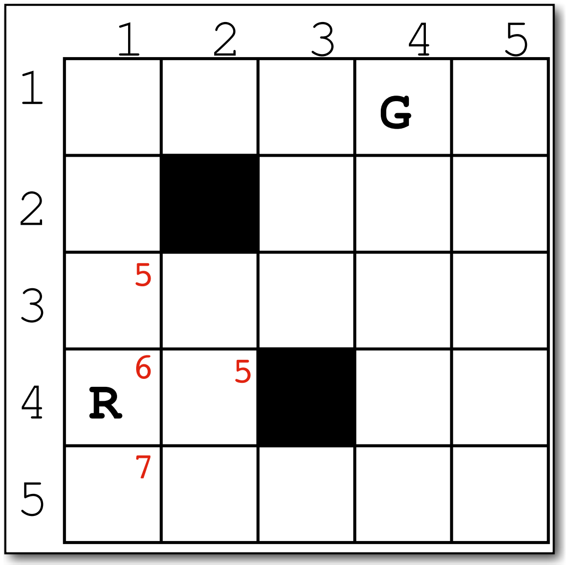


- Operators: R, L, U, D
- Heuristics?
  - Straight-line distance
  - Manhattan distance

Example

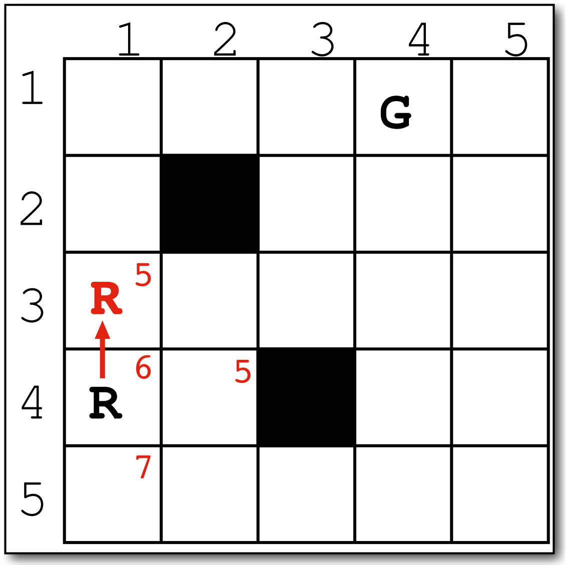


Example



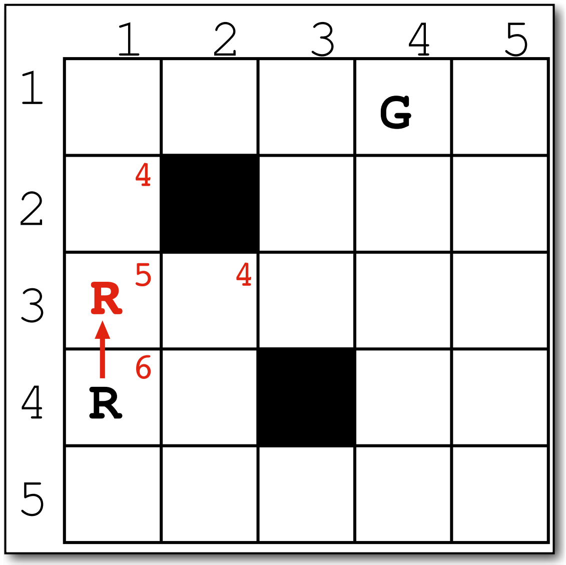
Uniformed search
Heuristic search
Hill-climbing
Greedy search
A*
Iterative Deepening A*
Memory-bounded A*
Simulated annealing
Beam search

Example



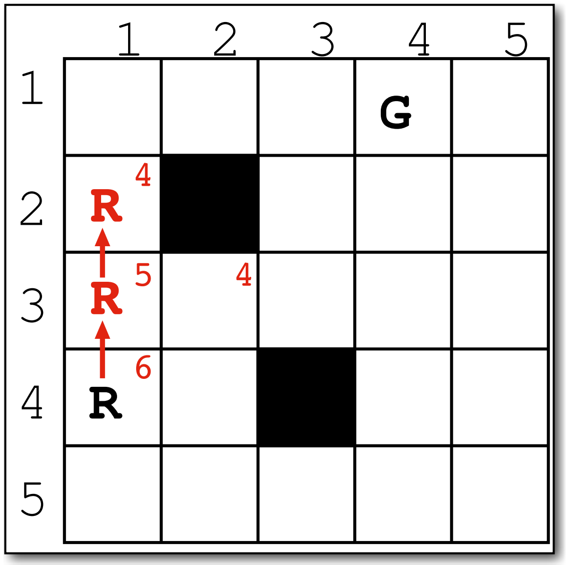
Uniformed search
Heuristic search
Hill-climbing
Greedy search
A*
Iterative Deepening A*
Memory-bounded A*
Simulated annealing
Beam search

Example



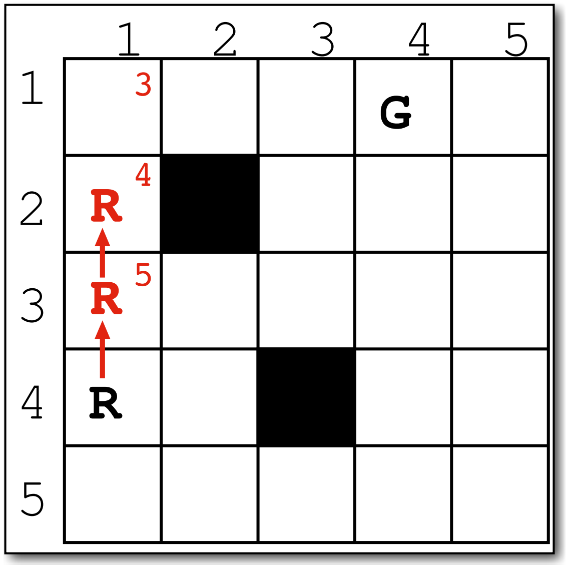
Uniformed search
Heuristic search
Hill-climbing
Greedy search
A*
Iterative Deepening A*
Memory-bounded A*
Simulated annealing
Beam search

Example

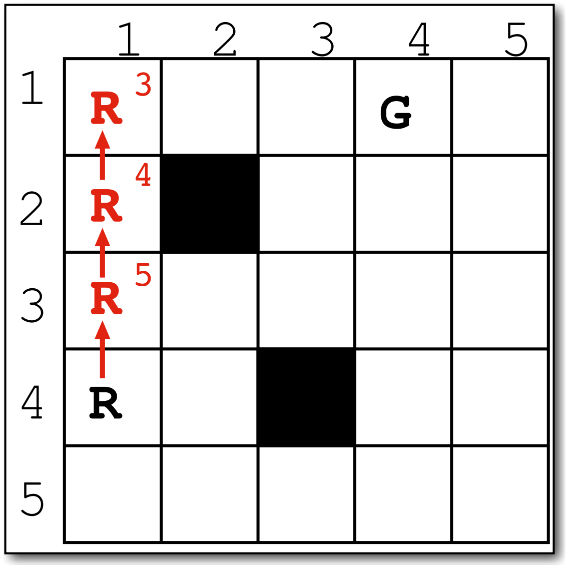


Uniformed search
Heuristic search
Hill-climbing
Greedy search
A*
Iterative Deepening A*
Memory-bounded A*
Simulated annealing
Beam search

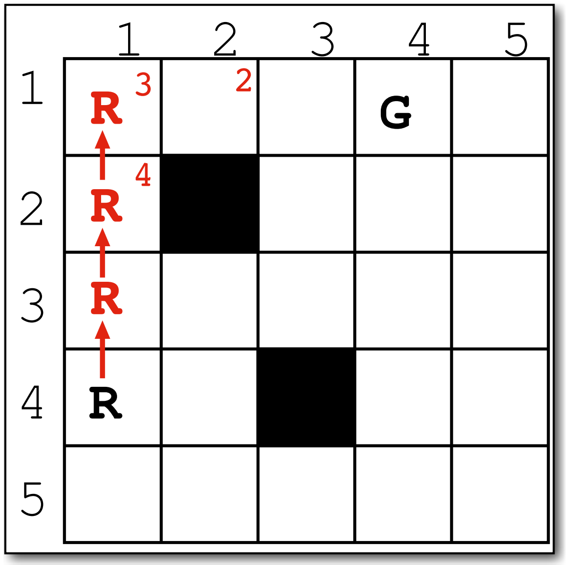
Example



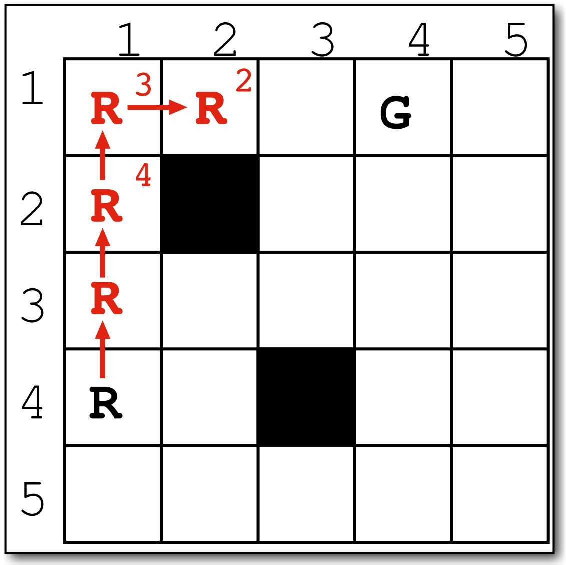
Example



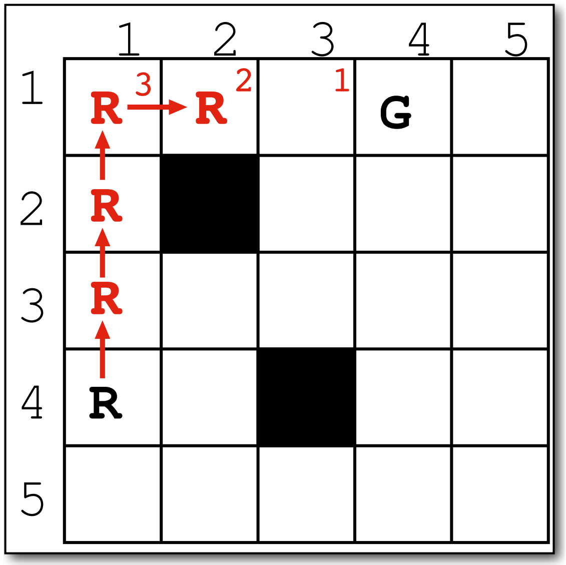
Example



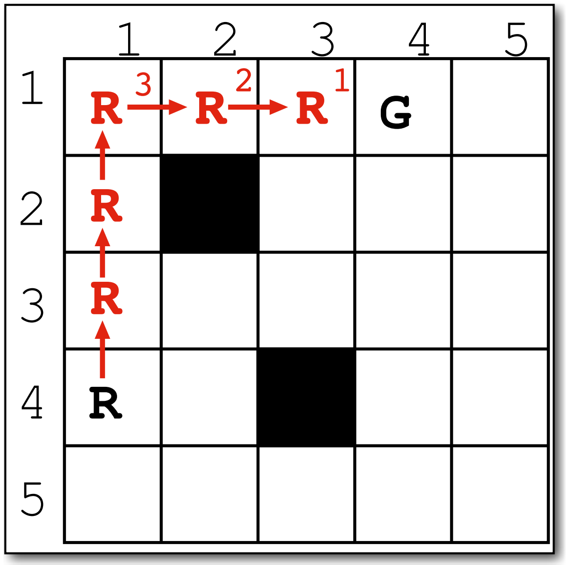
Example



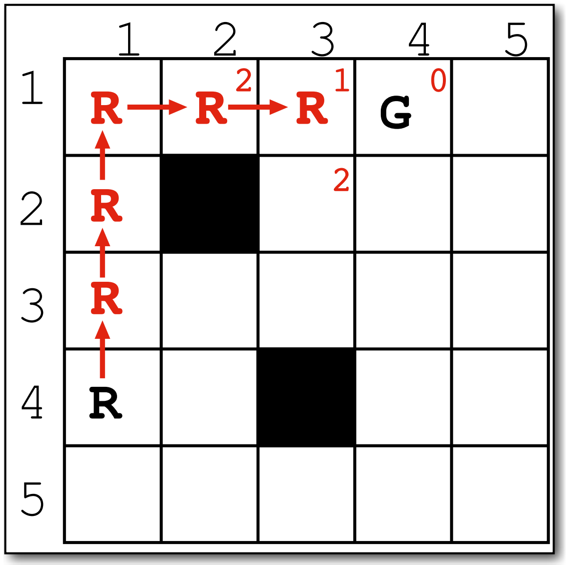
Example



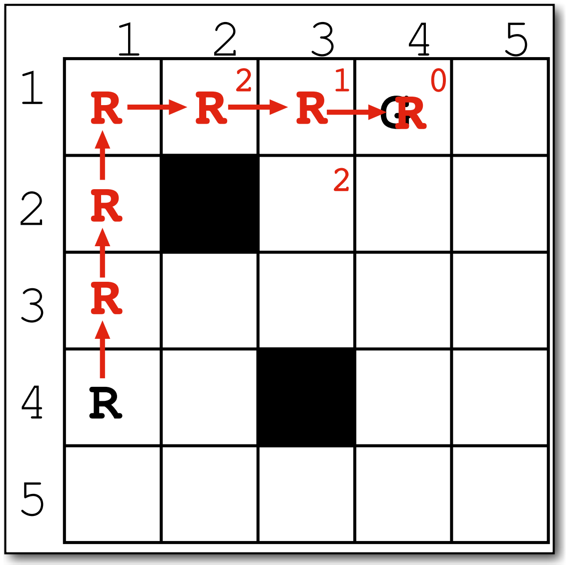
Example



Example



Example

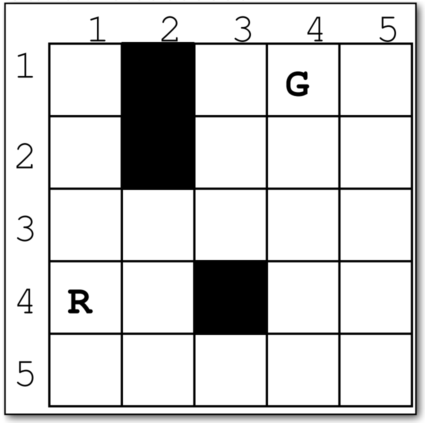


Problem: Local minima/maxima

- Occurs when at a and $\forall\, b \mid \text{child}(b, a) \wedge h(a) \geq h(b)$
- Can't find successor!

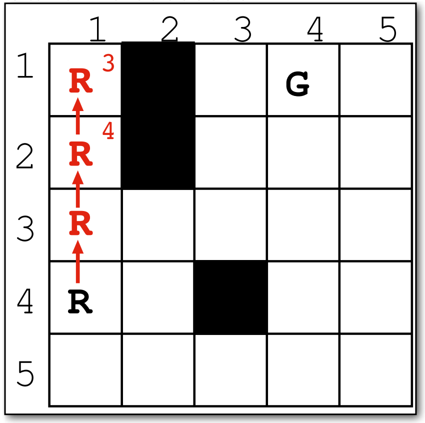
Problem: Local minima/maxima

- Occurs when at a and $\forall\, b \mid \text{child}(b, a) \wedge h(a) \geq h(b)$
- Can't find successor!



Problem: Local minima/maxima

- Occurs when at a and $\forall\, b \mid \text{child}(b, a) \wedge h(a) \geq h(b)$
- Can't find successor!



Problem: Local minima/maxima

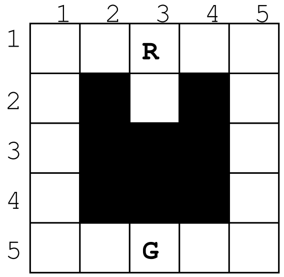
- Occurs when at a and $\forall\, b \mid \text{child}(b, a) \wedge h(a) \geq h(b)$
- Can't find successor!

Problem: Local minima/maxima

- Occurs when at a and $\forall\, b \,|\, \text{child}(b, a) \wedge h(a) \geq h(b)$
- Can't find successor!

Escaping local minima

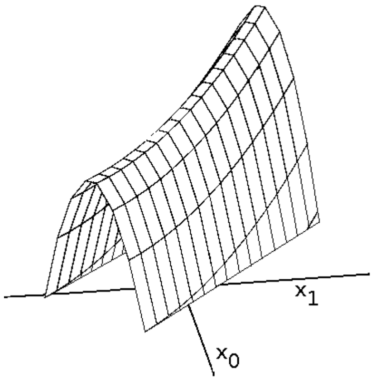
- Possible solution: *backtrack*
- Implementation: DFS, but order expansion by child cost
- But what if this is the initial state:



- Also, what if relative goal, e.g., "go East as far as you can"?

Problem: Ridges

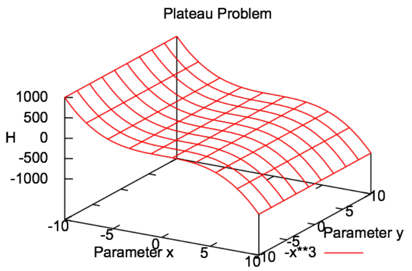
- Have ≥ 2 axes, continuous space
- Heuristic function looks something like:



- Progress if stepping in one dimension: slow, zig-zag
- Maybe can't make a single move to a better position
- Possible solution: try several moves in a row

Problem: Plateaus

- Reach area of search space where everything looks same (wrt $h(s)$)



- Potential solution: take n steps, do random jump

Hill-climbing advantages

- Good when we want to quickly find reasonable solution
  - Premise: local optimality ⇒ global optimality
  - If local heuristic always accurate ⇒ goal
  - May be the best we can do without *some* global information
- Can be used to search real world
- May sometimes get heuristic for free
  - If side-effect of checking for goal
  - E.g., if goal is to be close to x, then get distance during goal check

Hill-climbing disadvantages

- No guarantee of optimality!
- Local character of heuristics ⇒ plateau, ridge, minima problems
- Hard to get started in some problems if all choices look the same
  - Example: Robot in Boardman, wants to get to downtown Orono
  - Huge number of possible “next states”
  - All about the same in terms of distance from downtown

Related work

Started from the bottom, now we’re here…

–A.D. Graham

Always gonna be a uphill battle
Sometimes I’m gonna have to lose
Ain’t about how fast I get there,
Ain’t about what’s waiting on the other side
It’s the climb

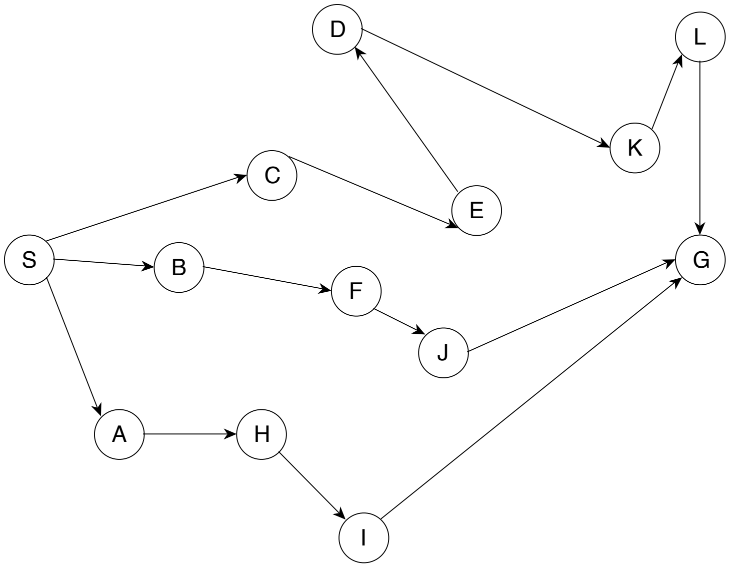
–M. Cyrus

Greedy search

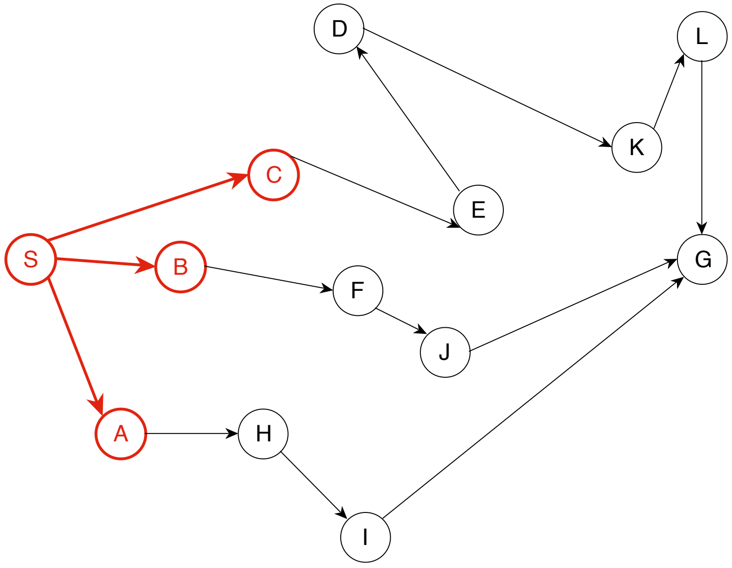
Greedy search

- Hill-climbing is one type of *greedy* search:
  - Pick better/best next node
  - HC is local, however
- Can also have non-local greedy search
- Choose best node from *frontier* – as in uniform-cost search
  - “Best” now incorporates heuristic
  - $h(s)$ estimates distance to goal

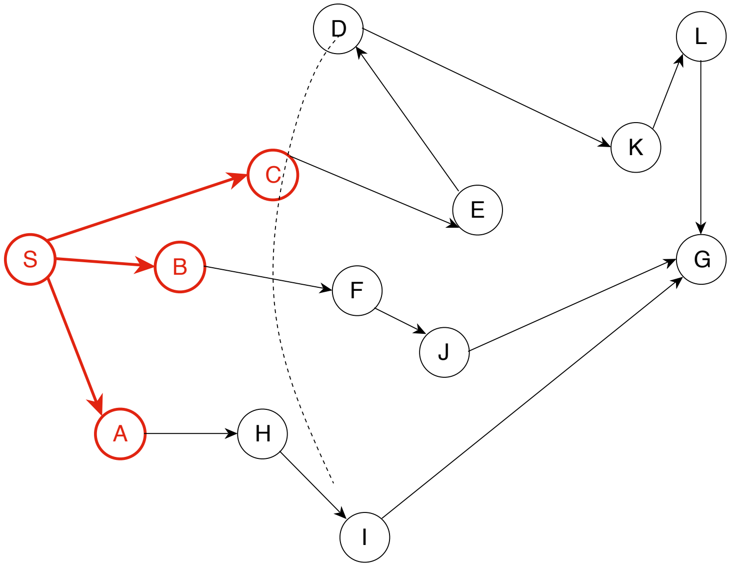
Example



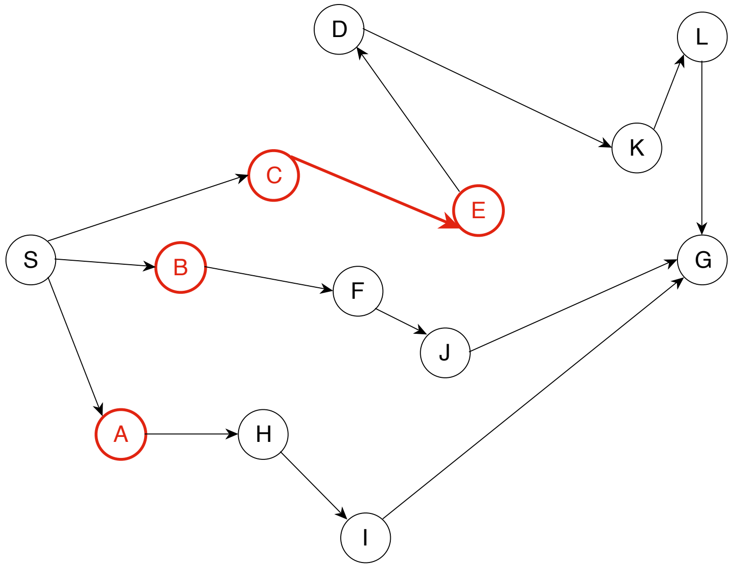
Example



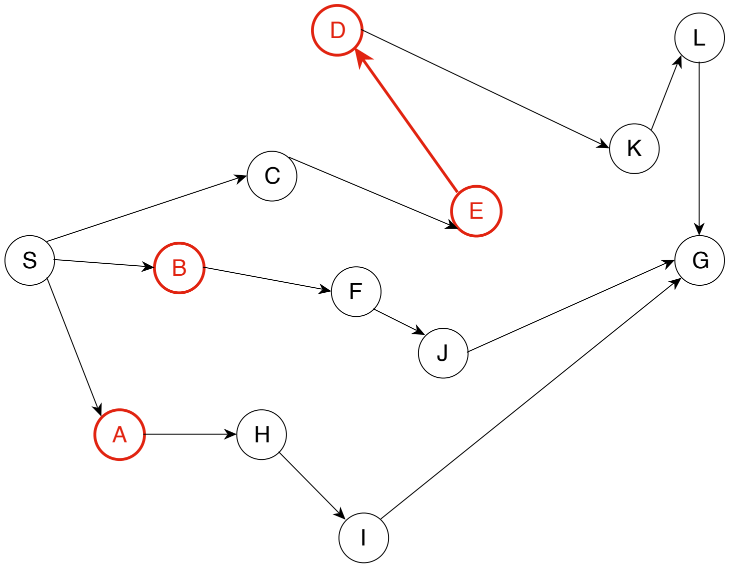
Example



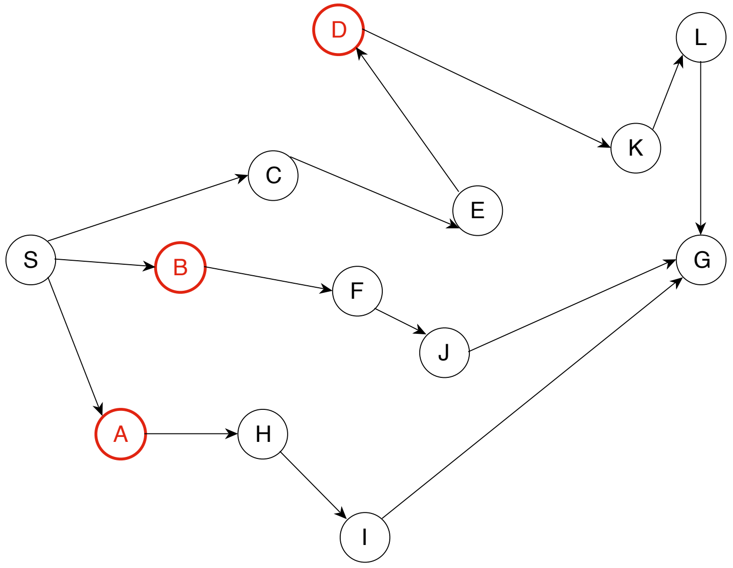
Example



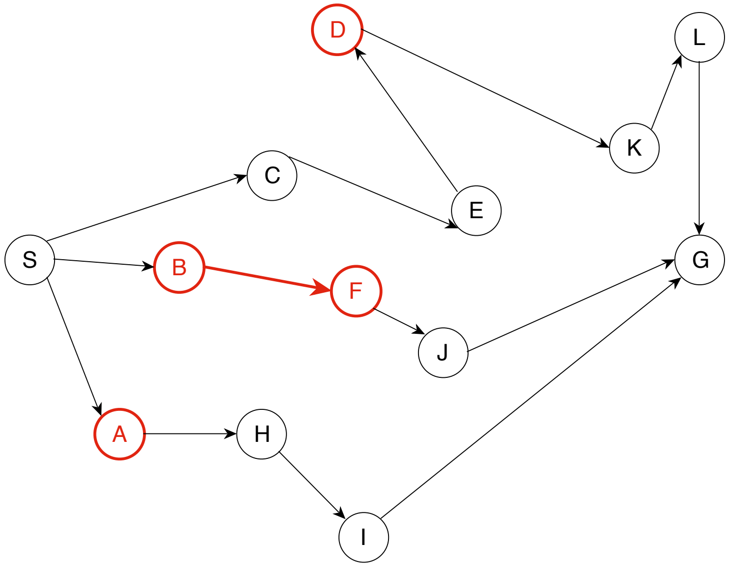
Example



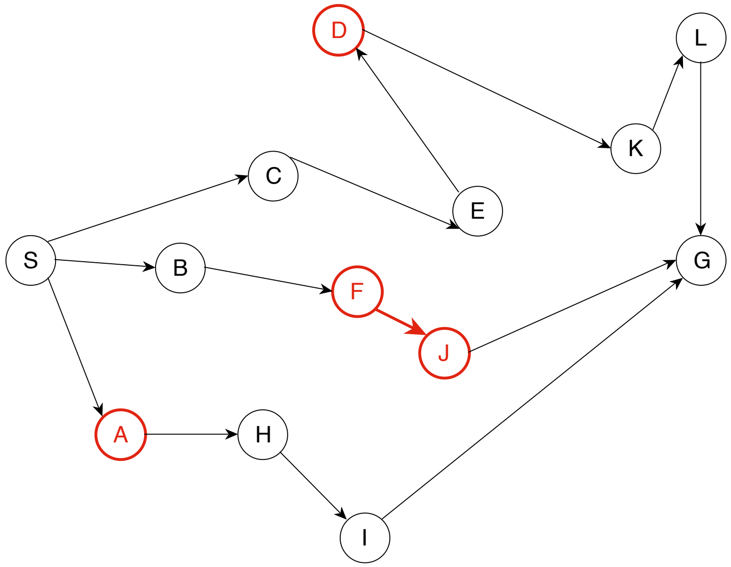
Example



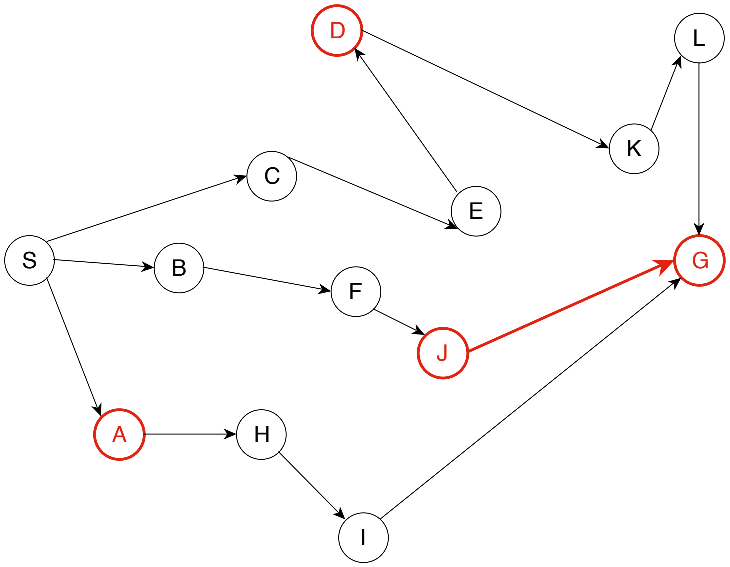
Example



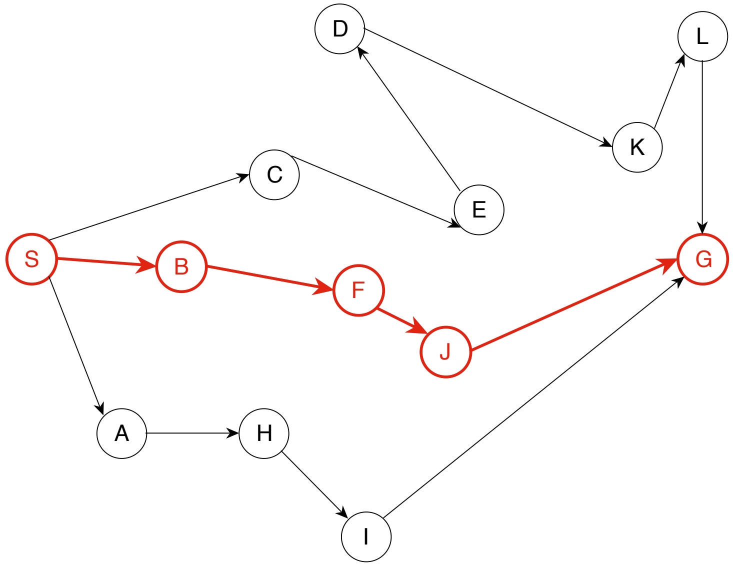
Example



Example



Example



Related work

I don't make merry myself at Christmas and I can't
afford to make idle people merry.

–E. Scrooge

And I'm greedy
'Cause I'm so greedy

–A. Grande

Can We Do Better?

- Uniform-cost (branch-and-bound): complete, optimal; no heuristics
- Greedy search: usually quick to zero in on goal; not guaranteed to be optimal
- Why not combine them?

A*

A*: A greedy heuristic search

- Greedy with respect to estimated *total path cost*
- Given: problem with start S and goal G
- Let $f(i) = g(i) + h(i)$ be best-cost path from $S \to G$ through i
  - $g(i)$ = cost of best path $S \to i$
  - $h(i)$ = cost of best path $i \to G$
- Can know $g(i)$
- Estimate $h(i)$ by $h'(i)$ (also: $h*(i)$): a heuristic function
- $f'(i) = g(i) + h'(i)$ = estimate of best cost path through i
- From the frontier: pick node with minimum f'

Algorithm overview

- Could use R&N's `Best-First-Search(problem,g+h\prime{})`
- May be easier to understand as standard algorithm
- Sketch:
  - Put start on a queue of open nodes
  - At each point:
    - Select the open (frontier) node with the best $f'(i)$
    - If none, fail; if goal, success.
    - Otherwise, update $f'(i)$ for the children, add them to queue
  - Hopefully f' is a better estimate of f as search progresses

Algorithm

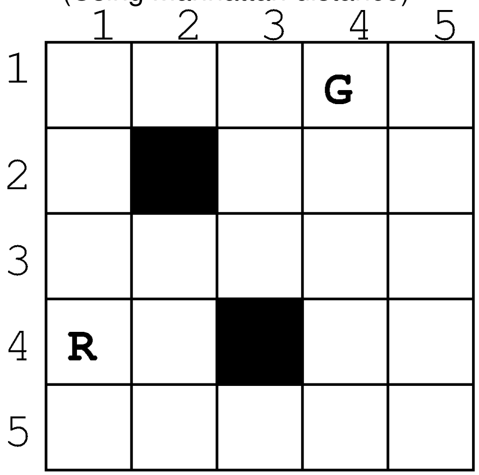
```
function A*(p)
    Input: a problem p
    Returns: path to solution or nil if none
    Let Open, Closed be empty lists
    Let Current = a search node
    Current.state = Start(p)
    Current.f = h(Start), Current.g = 0
    Add Current to Open
    while Open is non-empty do
        Current = node on Open with lowest f value
        Remove Current from Open, put on Closed
        if Current.state = Goal(p) then
            Compute path to Current, return path
        else
```

Algorithm (cont'd)

```
            for each successor state $i$ of Current.state do
                $g_i$ = Current.g + Cost(Current.state, $i$)
                $f_i = g_i$ + h($i$)
                if $i$ not on Open or Closed then
                    Create Child node, Child.state = $i$
                    Child.parent = Current
                    Child.g = $g_i$, Child.f = $f_i$
                    Add Child to Open list
                else
                    Child = Find($i$,Open) | Find($i$,Closed)
                    if $f_i$ < Child.f then
                        Child.g = $g(i)$, Child.f = $f_i$
                        Child.parent = Current
                        if Child ∈ Closed then
                            Remove, place on Open
    Return nil (failure)
```

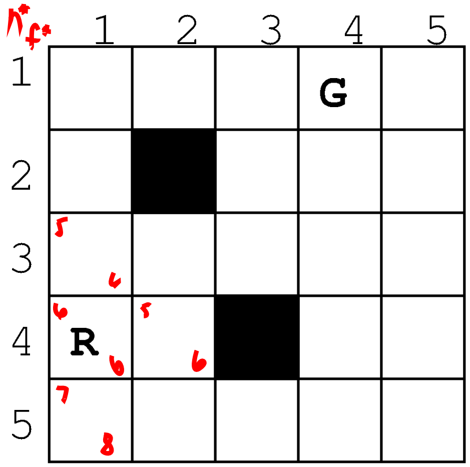
A* in the Robot World

(Using Manhattan distance)



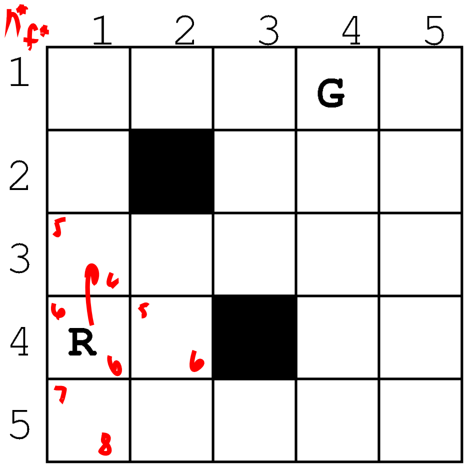
A* in the Robot World

(Using Manhattan distance)



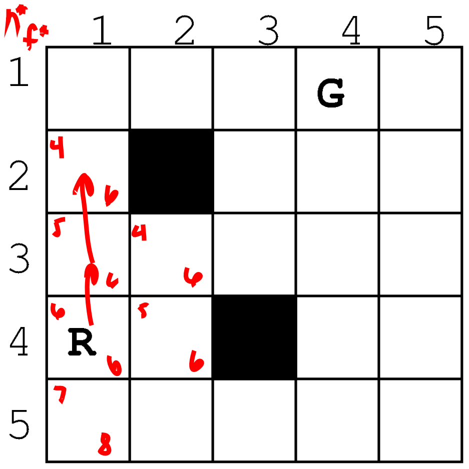
A* in the Robot World

(Using Manhattan distance)



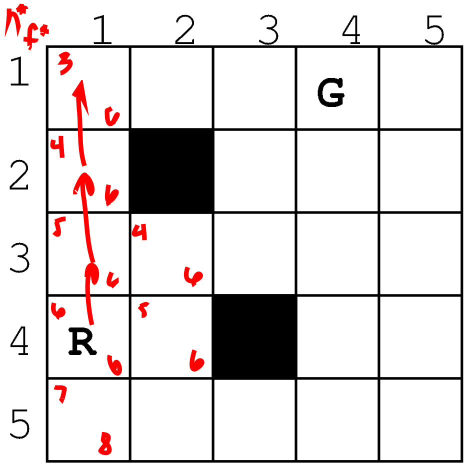
A* in the Robot World

(Using Manhattan distance)



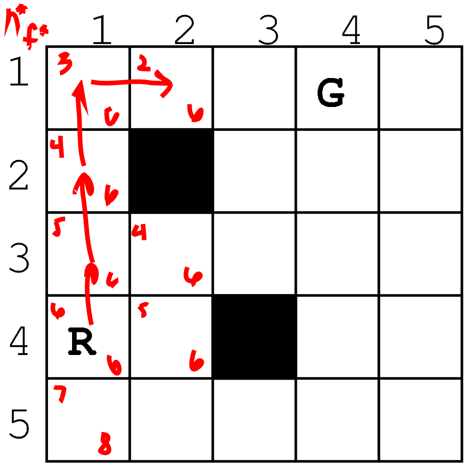
A* in the Robot World

(Using Manhattan distance)



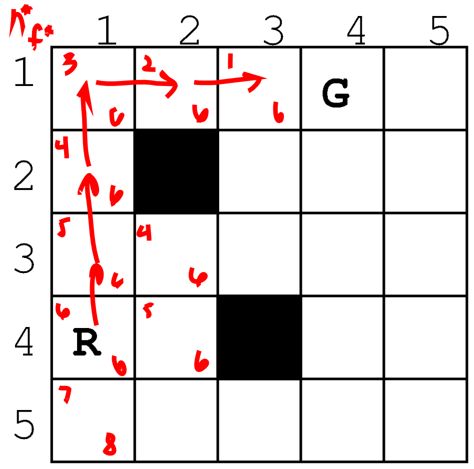
A* in the Robot World

(Using Manhattan distance)



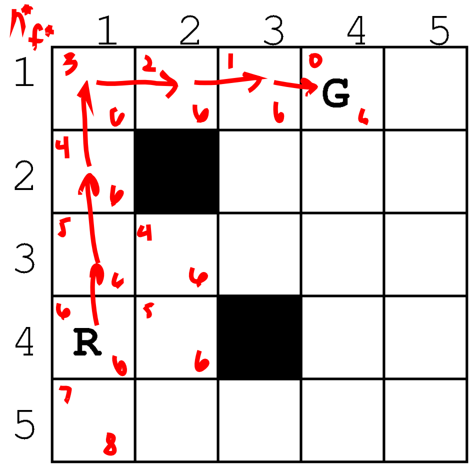
A* in the Robot World

(Using Manhattan distance)



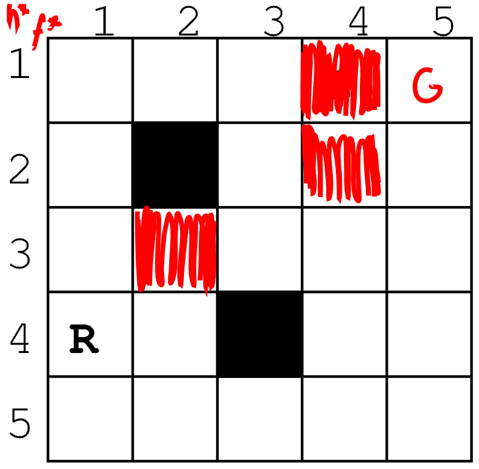
A* in the Robot World

(Using Manhattan distance)



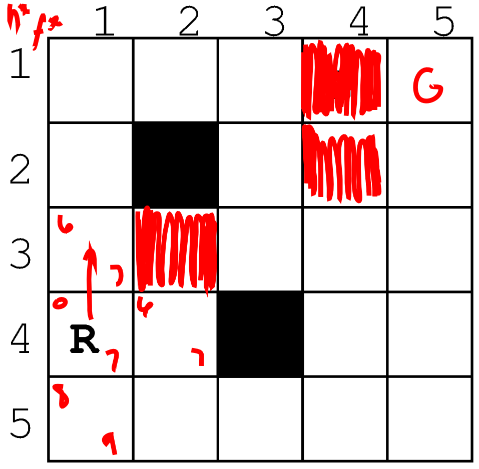
A∗ in the Robot World

(Using Manhattan distance)



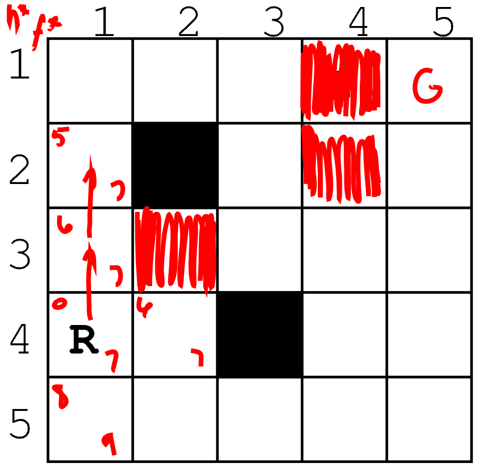
A∗ in the Robot World

(Using Manhattan distance)



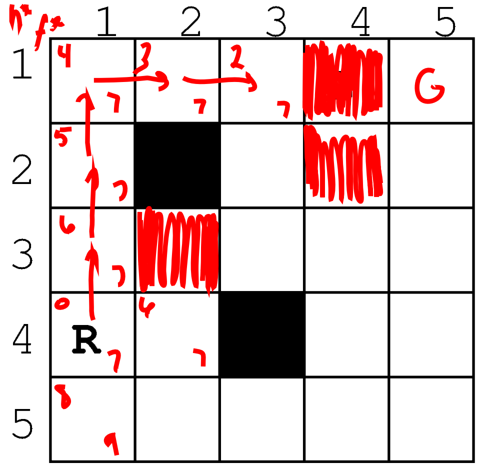
A∗ in the Robot World

(Using Manhattan distance)



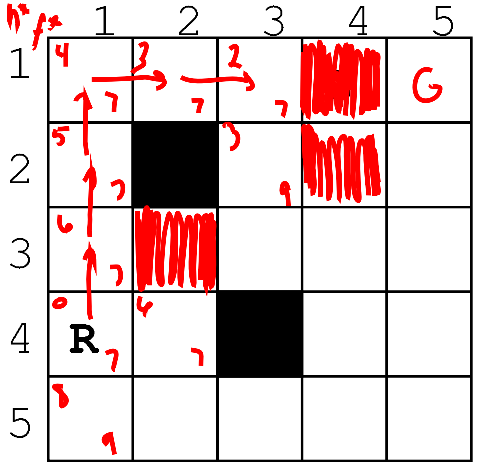
A∗ in the Robot World

(Using Manhattan distance)



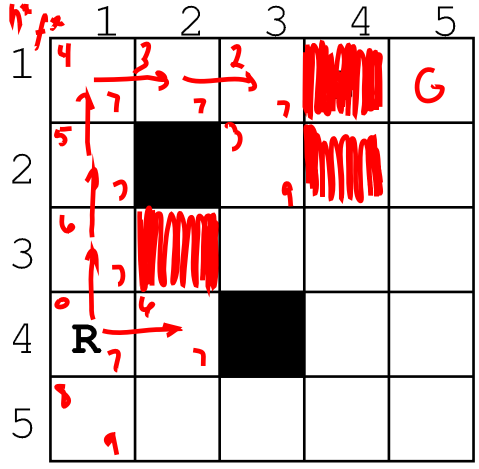
A∗ in the Robot World

(Using Manhattan distance)



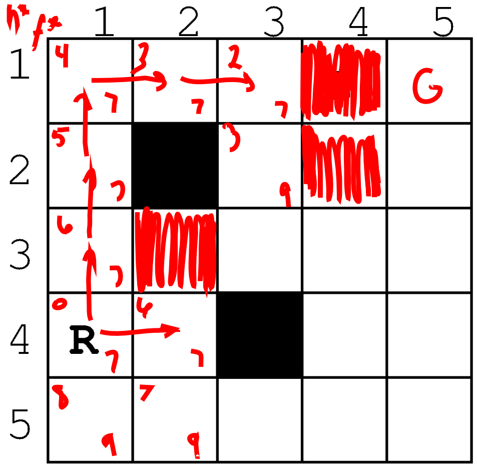
A∗ in the Robot World

(Using Manhattan distance)



$A*$ in the Robot World

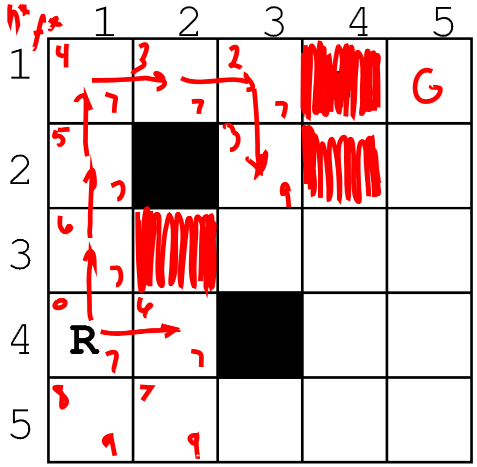
(Using Manhattan distance)



Artificial Intelligence

$A*$ in the Robot World

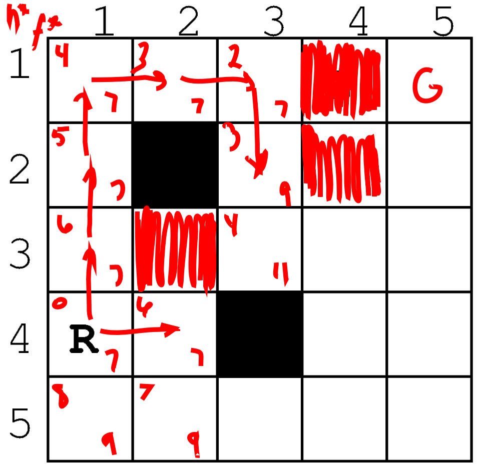
(Using Manhattan distance)



Artificial Intelligence

$A*$ in the Robot World

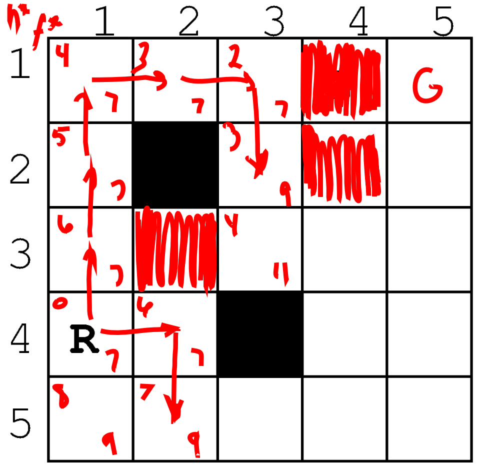
(Using Manhattan distance)



Artificial Intelligence

$A*$ in the Robot World

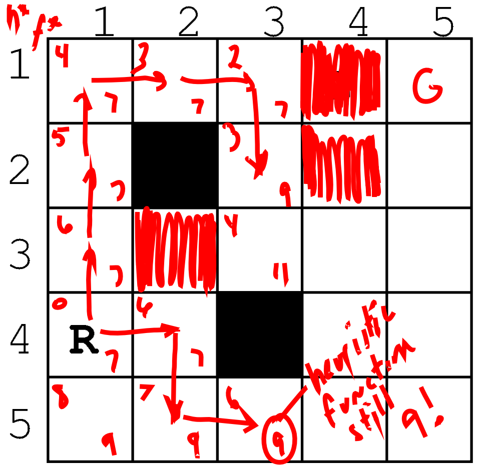
(Using Manhattan distance)



Artificial Intelligence

$A*$ in the Robot World

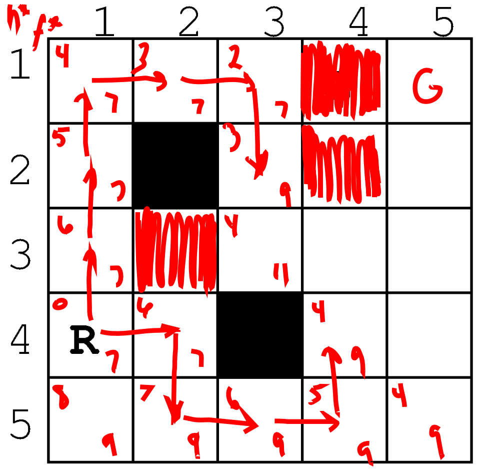
(Using Manhattan distance)



Artificial Intelligence

$A*$ in the Robot World

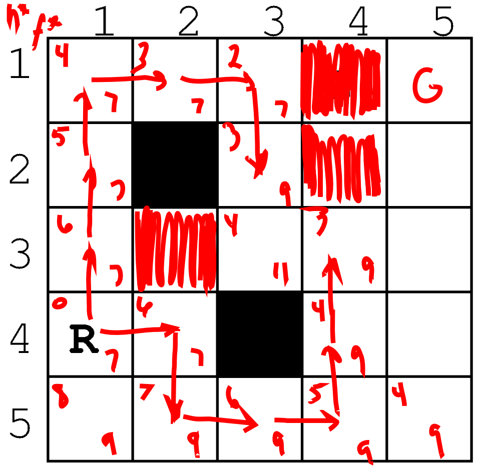
(Using Manhattan distance)



Artificial Intelligence

$A*$ in the Robot World

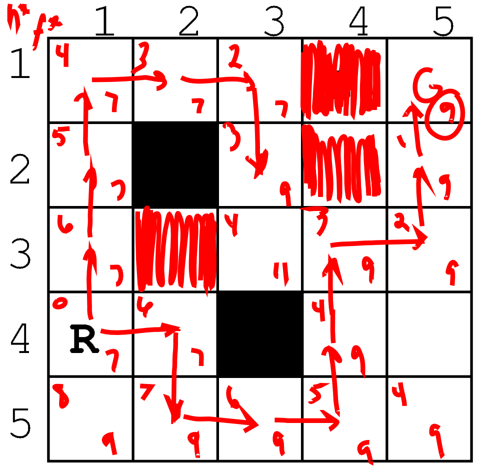
(Using Manhattan distance)



Artificial Intelligence

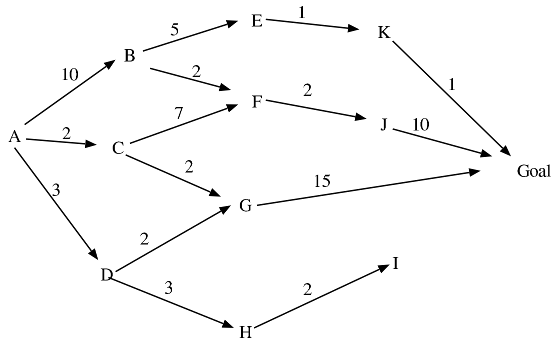
$A*$ in the Robot World

(Using Manhattan distance)



Artificial Intelligence

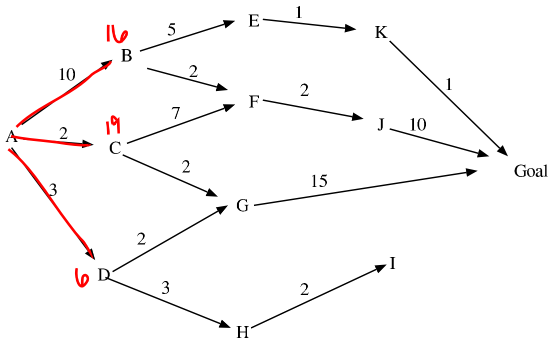
Another *A*∗ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

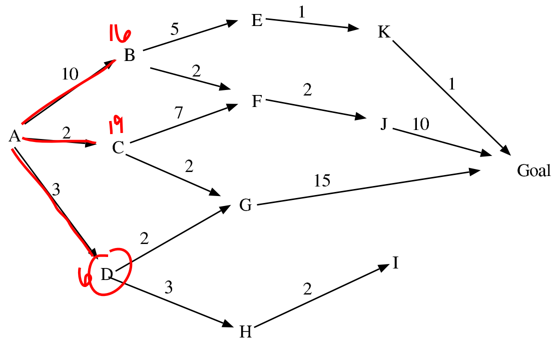
Another *A*∗ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

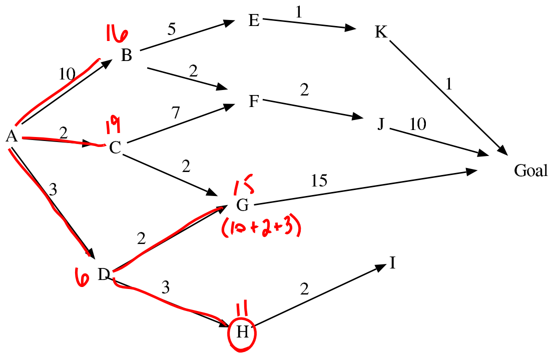
Another *A*∗ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

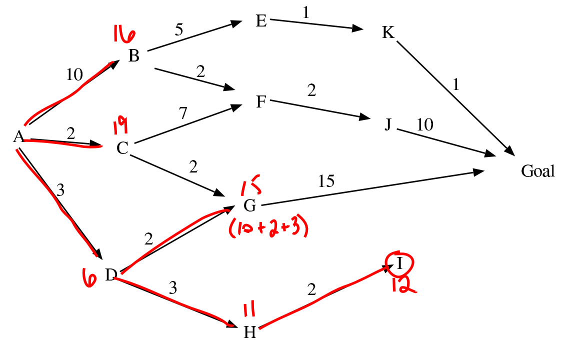
Another *A*∗ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

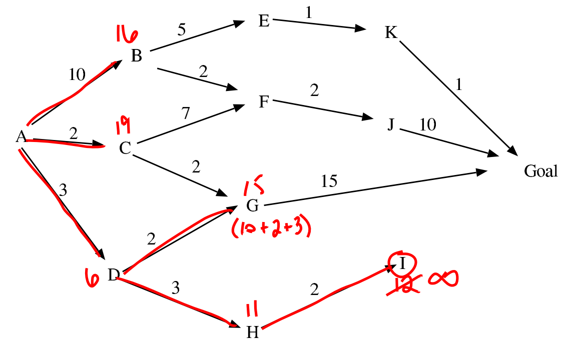
Another $A*$ Example



Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

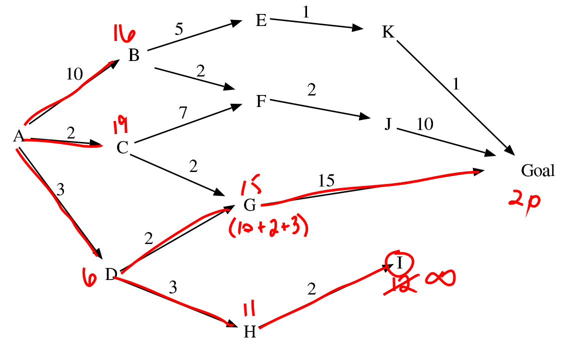
Another $A*$ Example



Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

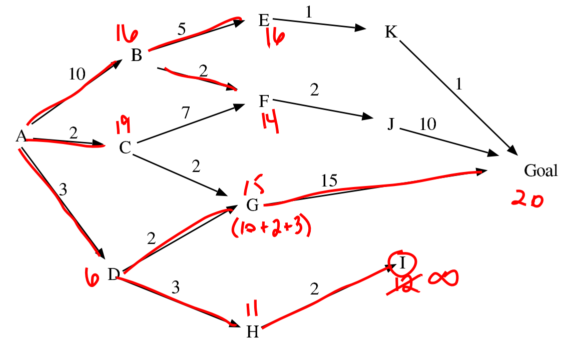
Another $A*$ Example



Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

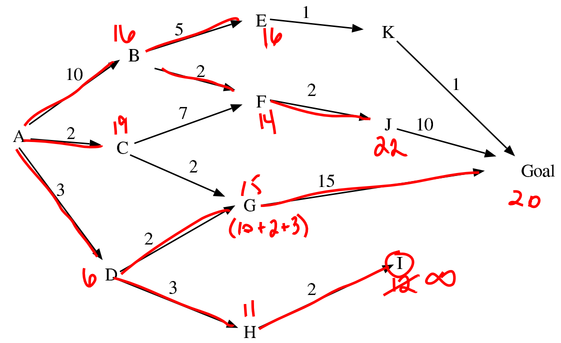
Another $A*$ Example



Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

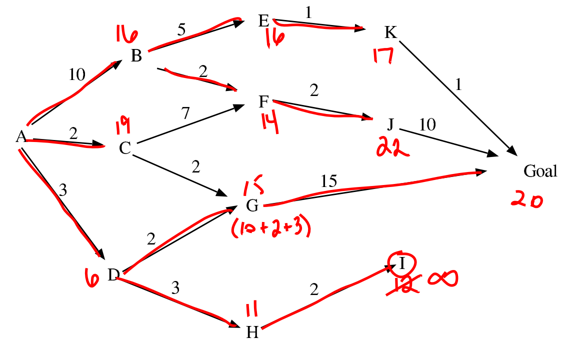
Another $A*$ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

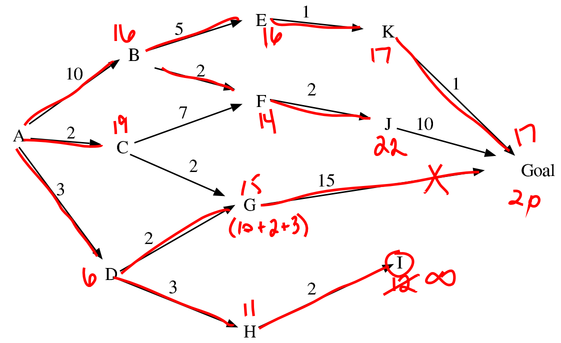
Another $A*$ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

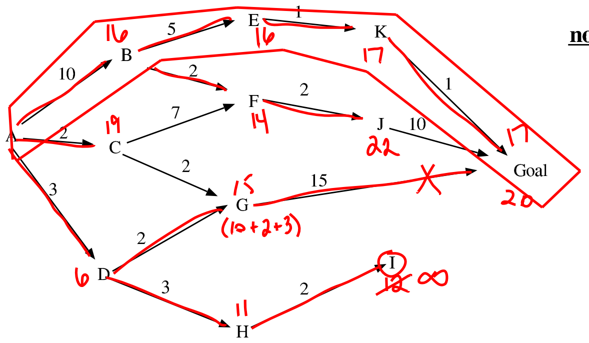
Another $A*$ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

Another $A*$ Example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

Properties of A*

- Complete?

Properties of A*

- Complete? Yes

Properties of A*

- Complete? Yes
- Optimality: Two types: optimal solution and optimal search

Properties of A*

- Complete? Yes
- Optimality: Two types: optimal solution and optimal search
- *Admissible* search:

  > *A search algorithm is admissible if, for any graph, it always terminates in an optimal path from [the start] to goal when ever a path from [the start] to a goal node exists.” (Nilsson)*

Properties of A*

- Complete? Yes
- Optimality: Two types: optimal solution and optimal search
- *Admissible* search:

  > *A search algorithm is admissible if, for any graph, it always terminates in an optimal path from [the start] to goal when ever a path from [the start] to a goal node exists." (Nilsson)*

- Is A* admissible?

Admissibility of A*

- Suppose A* selects a goal node G from Open
- $\Rightarrow \forall\, i \in$ Open, $f'(G) \leq f'(i)$
- Suppose $\forall\, i \in$ Open, $h'(i) \leq h(i)$
  - $\Rightarrow$ h' is an *underestimating* heuristic
  - $\Rightarrow$ f' also underestimates f for all nodes
- Nodes really represent *paths* to goal through a state
- $f'(G) = f(G)$ since we are at goal
- Cost of path to G $\leq$ all other estimated costs. . .
- . . . and estimated costs $\leq$ actual costs. . .
- $\therefore$ G is optimal path
- $\therefore$ A* is admissible with underestimating heuristics

Overestimating heuristics

- We consider $h' \leq h$ to be underestimating heuristic
- What if sometimes $h' > h$?
- Suppose G, representing a path to goal, is selected from Open
  - $f(g) \leq f'(i)$, $\forall\, i \in$ Open
  - But some $f'(i) > f(i)$
  - $\therefore$ possible: $f(i) < f(G) \Rightarrow$ G not optimal path
- Also:
  - Extra work may be done during search
  - Select node j, but possible $f(i) < f(j)$

Fun with heuristics

- What happens if $\forall\, i, h'(i) = 0$?

Fun with heuristics

- What happens if $\forall\, i, h'(i) = 0$? $\Rightarrow$ uniform-cost search

Fun with heuristics

- What happens if $\forall\, i, h'(i) = 0$? $\Rightarrow$ uniform-cost search
- What if we ignore g, i.e., $f'(i) = h'(i)$?

Fun with heuristics

- What happens if $\forall\, i, h'(i) = 0$? $\Rightarrow$ uniform-cost search
- What if we ignore g, i.e., $f'(i) = h'(i)$?$\Rightarrow$ greedy/best-first search
- What if $f'(i) = \text{depth of } i$

Fun with heuristics

- What happens if $\forall\, i, h'(i) = 0$? $\Rightarrow$ uniform-cost search
- What if we ignore g, i.e., $f'(i) = h'(i)$?$\Rightarrow$ greedy/best-first search
- What if $f'(i) = \text{depth of } i \Rightarrow$ BFS

Heuristics for A*

- Best heuristic function: highest value without overestimating cost
- Limitation of admissibility: not always easy to find underestimating heuristic function
- Graceful decay of admissibility
  - Let C_o be the cost of the optimal solution
  - Suppose h' rarely overestimates h by more than δ
  - $\Rightarrow$ A* will rarely find a solution whose cost is $> C_o + \delta$

How to find heuristics?

- *Relax* problem by taking away some condition in problem statement
- Exact solution to relaxed problem often good heuristic
- E.g., in Robot World:
  - Problem: Move from S to G using Manhattan moves and avoiding obstacles
  - Relaxed 1: Move from S to G using Manhattan moves.
  - Relaxed 2: Move from S to G

Heuristic Accuracy

- *Effective branching factor:*
  - Suppose expand N nodes for depth d solution
  - In a balanced tree, what would have been branching factor?
  - No closed-form solution, but can estimate $b*$
    - $b* \approx 2^{\log N/d}$ – or
    - $b* \approx N^{1/d}$
  - E.g.:
    - Expand $N = 1024$ nodes, depth $d = 10$: $b* \approx 2$
    - Expand $N = 1000$ nodes, depth $d = 5$: $b* \approx 4$; $4^5 = 1024$
- Heuristic h_1 is better h_2 if $b_1* < b_2*$ for all nodes
- Ideally: $b*$ close to 1

Dominance

- Heuristic h_2 *dominates* h_1 if for any node n, $h_2(n) > h_1(n)$
- A* using h_2 will never expand more nodes than using h_1
- What if no heuristic dominates any other?

Properties of A*

- A* is *optimally efficient*: for any heuristic function, no other optimal algorithm is guaranteed to expand fewer nodes
- If A*$_1$ uses h'_1 and A*$_2$ uses h'_2 and $h'_1(n) > h'_2(n)$ for all n, then A*$_2$ expands *at least* every node that A*$_1$ does
- Time complexity: still $\mathcal{O}(b^d)$ in worst case
- Space complexity: Poor – keeps all expanded nodes in memory!

More examples

A*, others in JavaScript
A* vs Dijkstra
A* example video

Related work

Every day A is born.*

–J. –Z

*When you wish upon A**
...
Anything your heart desires will come to you.

–J. Cricket

Iterative Deepening A*

Iterative Deepening A*

- Space complexity of A* is terrible – maybe do something like IDFS?
- Instead of depth, think *cost*
- Use DFS multiple times, each time within some cost "contour" limit (min. of any node exceeding prev. limit)

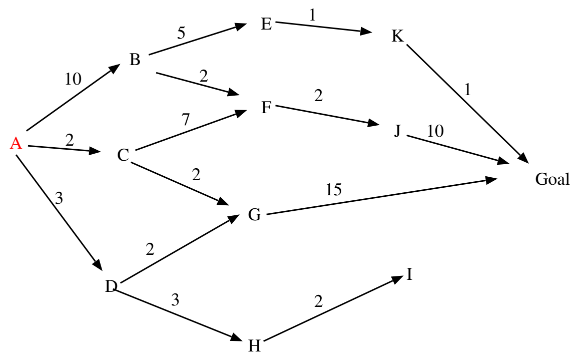
Algorithm

```
function IDA*(problem) returns a solution sequence
    inputs: problem, a problem
    static: f-limit, the current f- COST limit
            root, a node

    root ← MAKE-NODE(INITIAL-STATE[problem])
    f-limit ← f- COST(root)
    loop do
        solution, f-limit ← DFS-CONTOUR(root, f-limit)
        if solution is non-null then return solution
        if f-limit = ∞ then return failure; end
```

```
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f- COST limit
    inputs: node, a node
            f-limit, the current f- COST limit
    static: next-f, the f- COST limit for the next contour, initially ∞

    if f- COST[node] > f-limit then return null, f- COST[node]
    if GOAL-TEST[problem](STATE[node]) then return node, f-limit
    for each node s in SUCCESSORS(node) do
        solution, new-f ← DFS-CONTOUR(s, f-limit)
        if solution is non-null then return solution, f-limit
        next-f ← MIN(next-f, new-f); end
    return null, next-f
```

IDA* example

f-limit = 16

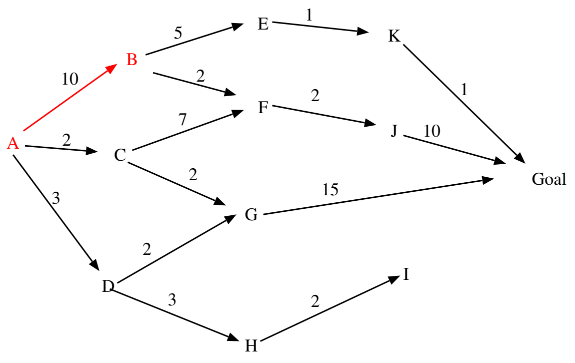


Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

IDA* example

f-limit = 16

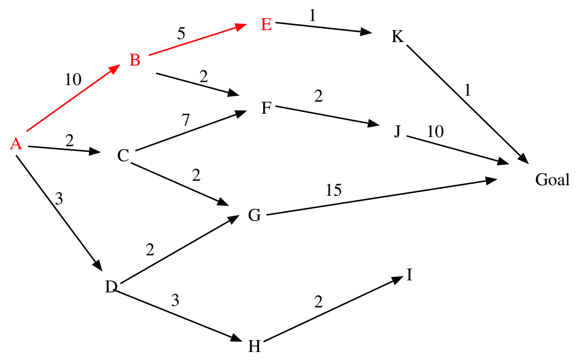


Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

IDA* example

f-limit = 16

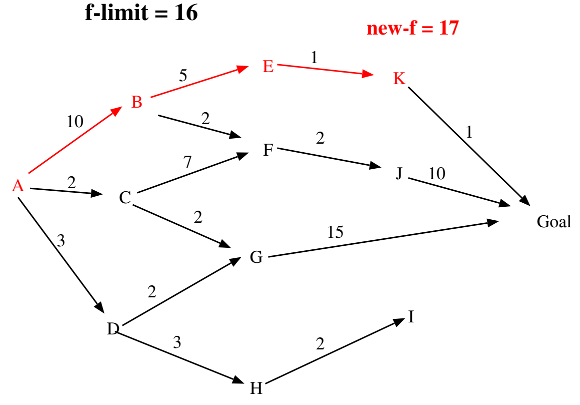


node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

IDA* example

f-limit = 16

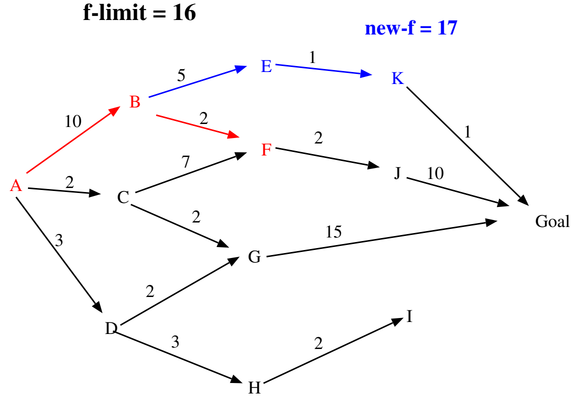


node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

IDA* example

f-limit = 16

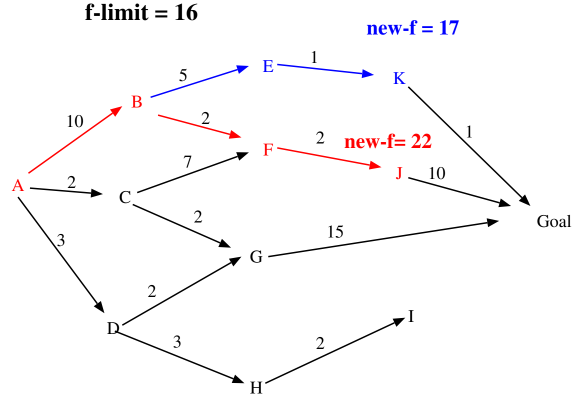


node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

IDA* example

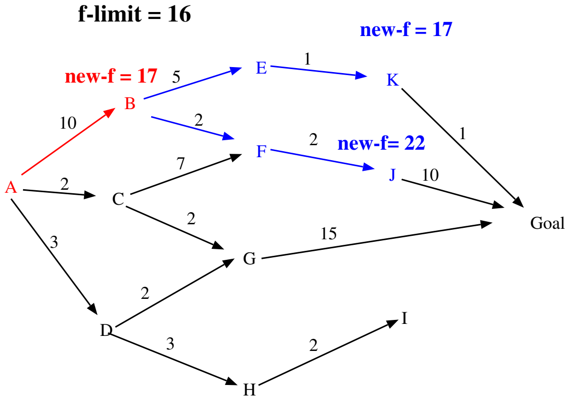
f-limit = 16



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

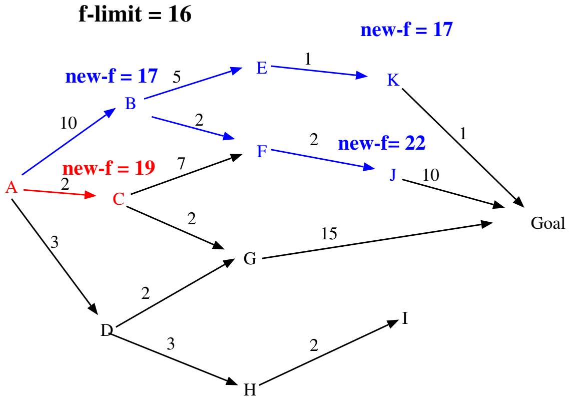
IDA* example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

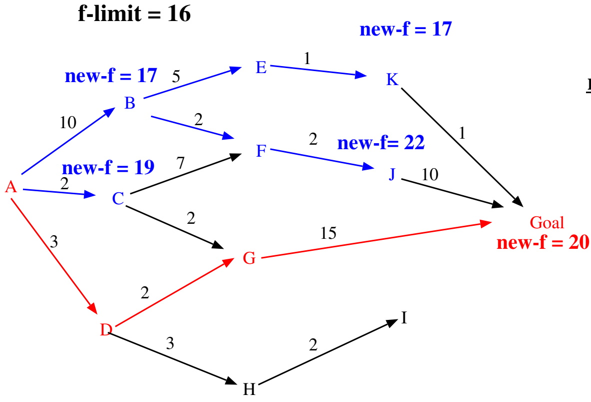
IDA* example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

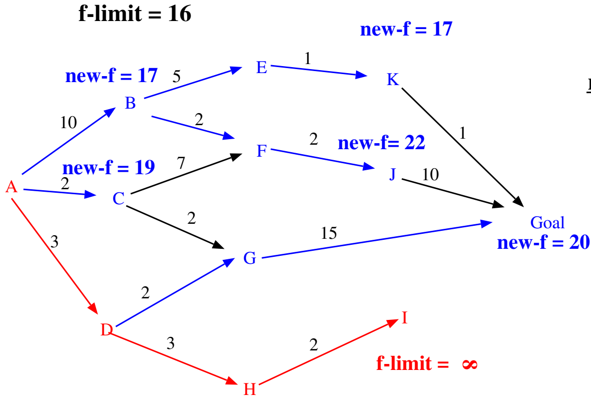
IDA* example



node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

IDA* example

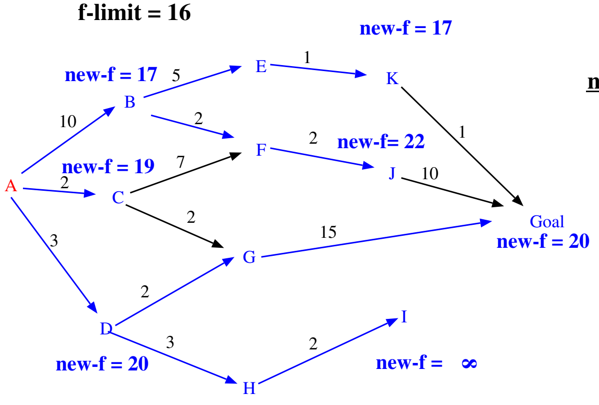


node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Search Space

IDA* example

f-limit = 16

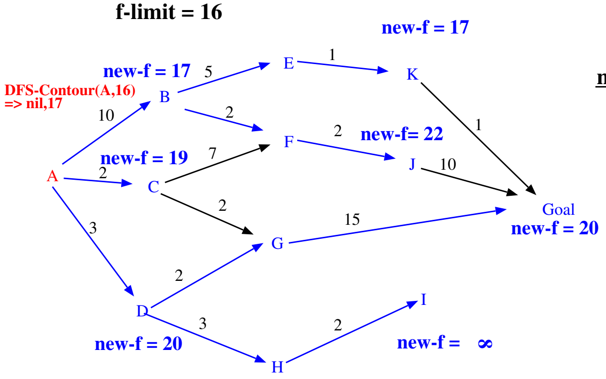


Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

IDA* example

f-limit = 16

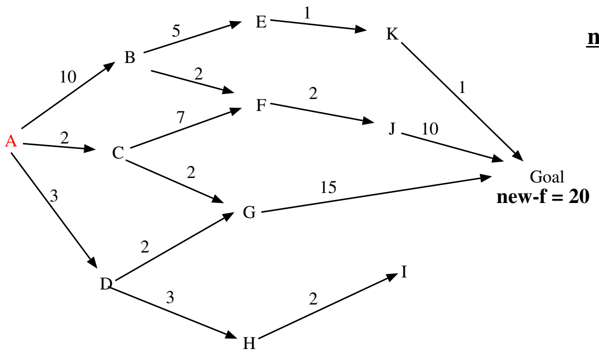


Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

IDA* example

f-limit = 17

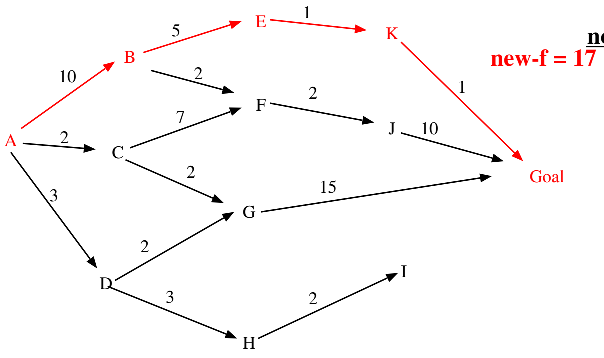


Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

IDA* example

f-limit = 17



Search Space

node	h*(node)
A	16
B	6
C	17
D	3
E	1
F	2
G	10
H	5
I	4
J	8
K	1

Properties

- Complete, optimal – with same restrictions as A*
- Space complexity: worst case $\mathcal{O}(bf'/\delta)$, where:
  - b = branching factor, f' = cost of optimal solution
  - δ = smallest operator cost
- Can estimate usually as $\mathcal{O}(bd)$

Time complexity

- Time depends on properties of h'
  - If h' has large *grain size*, then search quite a bit of the tree each DFS call
  - Small grain size: DFS may be called many times – worst case, once per expanded node
    - if A* expands *a* nodes, IDA* in this case expands $1 + 2 + ... + a = a^2$ nodes
    - worst case: $\mathcal{O}((b^d)^2) = \mathcal{O}(b^{2d})$
    - Example
  - Can ameliorate this by forcing granularity to be coarse
    - Increase f' contour by ϵ each time
    - Solution could be as much as ϵ sub-optimal
    - *ϵ-admissibility*
    - Example

Related work

> *They meant to set up a standard maxim for free society, which should be familiar to all, and revered by all; constantly looked to, constantly labored for, and even though never perfectly attained, constantly approximated, and thereby constantly spreading and deepening its influence and augmenting the happiness and value of life to all people of all colors everywhere.*

–A. Lincoln

Memory-bounded A*

Simple memory-bounded A*

- Can we do better with respect to space?
- Simple memory-bounded A*
  - Uses whatever memory you give it
  - If enough memory to store a solution ⇒ complete
  - If enough to store optimal solution ⇒ optimal
  - If not, will return best solution that will fit in memory
- Idea:
  - Proceed like A*, but when bump against memory limit, drop the highest-cost node from queue
  - Record in a node the cost of its best descendant node
  - Don’t re-expand unless all other paths in memory are worse
- Complex search! – see R&N for details

Related work

Enough is as good as a feast.

–Joshua Sylvester, *Works* (1611).

Simulated annealing

Simulated annealing

- Hill-climbing: *iterative improvement* algorithm
- Some drawbacks: e.g., local minima/maxima
- Addressed drawbacks with (e.g.) random jumps–can we do better?
- *Simulated annealing:* allows some “downhill” moves to escape local maxima
- Analogous to annealing

Annealing

- Goal: Metal at lowest energy level
- ⇒ Most stable crystal structure
- Problem: ∃ local minima, “trap” metal as it cools
- Solution: *annealing*
  - Make use of randomness – thermal noise – in physical system
  - Devise *schedule* of temperature reduction
  - Hold/slow at some temperatures for a while ⇒ escape local minima

Simulated annealing

- At start, probability of random moves high
- As progress, ↓probability
- Define:
  - “Temperature” T: P(uphill move) $\propto$ T
  - *Schedule* for lowering temperature over time/as moves made

General approach

- At node: Try a random move
- If better state, take it
- If not, then with $P = f(\mathrm{T})$, take move
- Reduce temperature according to schedule

Example

Simulated annealing example

Related work

Love's a different sort of thing, hot enough to make you flow
into something, interflow, cool and anneal and be a weld
stronger than what you started with.

Theodore Sturgeon, *More than Human* (1953)

Beam search

Beam search

- Problem with breadth-first searches: branching factor!
- If can reduce *b*, speed up the search
- Approach: search only *i* best open nodes at level – *i* = *beam width*
- Pros: faster, cheaper (wrt. space)
- Cons: maybe not optimal, maybe not complete!

Stochastic beam search

- Like beam search, but random element
- Choose *i* nodes at random: prob of selection is function of worth

Related work

Dim as the borrowed beams of moon and stars
To lonely, weary, wandering travellers. . .

John Dryden, *Religio Laici* (1682)