

## Frontier search (manage choices)

```
frontierSearch([Node|Rest]) :- goal(Node);  
    (findall(Next, arc(Node,Next), Children),  
     add2frontier(Children, Rest, NewFrontier),  
     frontierSearch(NewFrontier)).
```

## Frontier search (manage choices)

```
frontierSearch([Node|Rest]) :- goal(Node);  
    (findall(Next, arc(Node,Next), Children),  
     add2frontier(Children, Rest, NewFrontier),  
     frontierSearch(NewFrontier)).
```

**Depth first:** append(Children, Rest, NewFrontier)

**Breadth-first:** append(Rest, Children, NewFrontier)

## Frontier search (manage choices)

```
frontierSearch([Node|Rest]) :- goal(Node);  
    (findall(Next, arc(Node,Next), Children),  
     add2frontier(Children, Rest, NewFrontier),  
     frontierSearch(NewFrontier)).
```

**Depth first:**    append(Children, Rest, NewFrontier)

**Breadth-first:** append(Rest, Children, NewFrontier)

For    add2frontier(Children, Rest, NewFrontier),    require

          NewFrontier merges Children and Rest

where a list L is defined to *merge* lists L1 *and* L2 if

- (a) every member of L is a member of L1 or L2
- (b) every member of L1 or of L2 is a member of L.

## Exercise (Prolog)

Suppose a positive integer *Seed* links nodes  $1, 2, \dots$  in two ways

`arc(N,M,Seed) :- M is N*Seed.`

`arc(N,M,Seed) :- M is N*Seed +1.`

e.g. `Seed=3` gives arcs  $(1,3)$ ,  $(1,4)$ ,  $(3,9)$ ,  $(3, 10)$  ...

## Exercise (Prolog)

Suppose a positive integer Seed links nodes  $1, 2, \dots$  in two ways

`arc(N,M,Seed) :- M is N*Seed.`

`arc(N,M,Seed) :- M is N*Seed +1.`

e.g. Seed=3 gives arcs (1,3), (1,4), (3,9), (3, 10) ...

Goal nodes are multiples of a positive integer Target

`goal(N,Target) :- 0 is N mod Target.`

e.g. Target=13 gives goals 13, 26, 39 ...

## Exercise (Prolog)

Suppose a positive integer Seed links nodes 1,2,... in two ways

`arc(N,M,Seed) :- M is N*Seed.`

`arc(N,M,Seed) :- M is N*Seed +1.`

e.g. Seed=3 gives arcs (1,3), (1,4), (3,9), (3, 10) ...

Goal nodes are multiples of a positive integer Target

`goal(N,Target) :- 0 is N mod Target.`

e.g. Target=13 gives goals 13, 26, 39 ...

Modify frontier search to define predicates

`breadth1st(+Start, ?Found, +Seed, +Target)`

`depth1st(+Start, ?Found, +Seed, +Target)`

that search breadth-first and depth-first respectively for a Target-goal node Found linked to Start by Seed-arcs.

## Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

## Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`



## Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

(A2) `Node1` is deemed no further from a goal node than `Node2`

## Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

(A1) `Node1` costs no more than `Node2`

(A2) `Node1` is deemed no further from a goal node than `Node2`

(A3) some mix of (A1) and (A2)

## Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

- (A1) `Node1` costs no more than `Node2`  
     $\rightsquigarrow$  minimum cost search (= breadth-first if every arc costs 1)
- (A2) `Node1` is deemed no further from a goal node than `Node2`
- (A3) some mix of (A1) and (A2)

## Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

- (A1) `Node1` costs no more than `Node2`  
     $\rightsquigarrow$  minimum cost search (= breadth-first if every arc costs 1)
- (A2) `Node1` is deemed no further from a goal node than `Node2`  
     $\rightsquigarrow$  best-first search (= depth-first for heuristic  $\propto$  depth<sup>-1</sup>)
- (A3) some mix of (A1) and (A2)

## Refining frontier search

For `add2frontier(Children, Rest, NewFrontier)`, require

`NewFrontier` merges `Children` and `Rest`

and for `NewFrontier = [Head|Tail]`, ensure

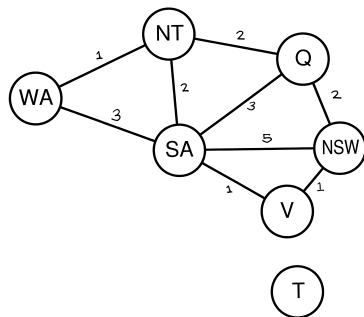
`Head` is “no worse than” any in `Tail`.

What can it mean for `Node1` to be *no worse than* `Node2` ?

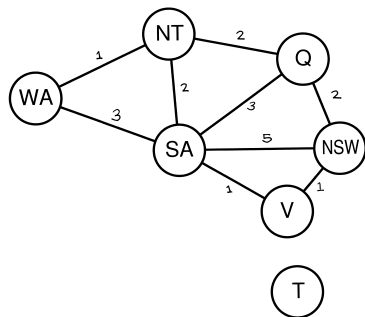
- (A1) `Node1` costs no more than `Node2`  
     $\rightsquigarrow$  minimum cost search (= breadth-first if every arc costs 1)
- (A2) `Node1` is deemed no further from a goal node than `Node2`  
     $\rightsquigarrow$  best-first search (= depth-first for heuristic  $\propto$  depth<sup>-1</sup>)
- (A3) some mix of (A1) and (A2)  
     $\rightsquigarrow$  A-star (next week)

## Arc costs (space, time, money, ...)

```
arc(wa,nt,1).    arc(nt,q,2).  
arc(q,nsw,2).   arc(wa,sa,3).  
arc(nt,sa,2).   arc(sa,q,3).  
arc(sa,nsw,5).  arc(sa,v,1).  
arc(v,nsw,1).
```



## Arc costs (space, time, money, ...)

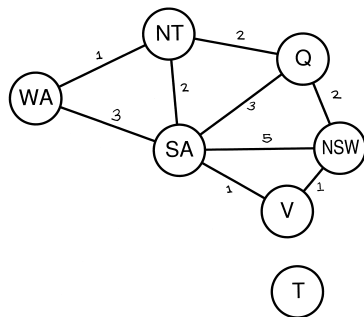


`arc(wa,nt,1).`      `arc(nt,q,2).`  
`arc(q,nsw,2).`    `arc(wa,sa,3).`  
`arc(nt,sa,2).`    `arc(sa,q,3).`  
`arc(sa,nsw,5).`   `arc(sa,v,1).`  
`arc(v,nsw,1).`

$$\text{cost}(\text{wa}, \text{nt}, \text{q}, \text{nsw}) = 1 + 2 + 2 = 5$$

$$\text{cost}(x_1, x_2, \dots, x_{k+1}) := \sum_{i=1}^k \text{cost}(x_i, x_{i+1})$$

## Arc costs (space, time, money, ...)



`arc(wa,nt,1).`      `arc(nt,q,2).`  
`arc(q,nsw,2).`    `arc(wa,sa,3).`  
`arc(nt,sa,2).`    `arc(sa,q,3).`  
`arc(sa,nsw,5).`   `arc(sa,v,1).`  
`arc(v,nsw,1).`

$$\text{cost}(\text{wa}, \text{nt}, \text{q}, \text{nsw}) = 1 + 2 + 2 = 5$$

$$\text{cost}(x_1, x_2, \dots, x_{k+1}) := \sum_{i=1}^k \text{cost}(x_i, x_{i+1})$$

$$\text{cost}(\text{wa}, \text{sa}, \text{nsw}) = 3 + 5 = 8$$



# Heuristics

$h(\text{Node})$  = estimate the minimum cost of  
a path from Node to a goal node

# Heuristics

$h(\text{Node})$  = estimate the minimum cost of  
a path from Node to a goal node

## EXAMPLES

- ▶ Fsm accept where node =  $[Q, \text{String}]$  and every arc costs 1

$$h([Q, \text{String}]) = \text{length}(\text{String})$$

# Heuristics

$h(\text{Node})$  = estimate the minimum cost of  
a path from Node to a goal node

## EXAMPLES

- ▶ Fsm accept where node =  $[Q, \text{String}]$  and every arc costs 1

$$h([Q, \text{String}]) = \text{length}(\text{String})$$

- ▶ Prolog search where node = list of propositions to prove, and every arc costs 1

$$h(\text{List}) = \text{length}(\text{List})$$

# Heuristics

$h(\text{Node})$  = estimate the minimum cost of  
a path from Node to a goal node

## EXAMPLES

- ▶ Fsm accept where node =  $[Q, \text{String}]$  and every arc costs 1

$$h([Q, \text{String}]) = \text{length}(\text{String})$$

- ▶ Prolog search where node = list of propositions to prove, and every arc costs 1

$$h(\text{List}) = \text{length}(\text{List})$$

- ▶ Node = point on a Euclidean plane, cost = distance between nodes, goal is a point G

$$h(\text{Node}) = \text{straight-line distance to G}$$

# Heuristics

$h(\text{Node})$  = estimate the minimum cost of  
a path from Node to a goal node

## EXAMPLES

- ▶ Fsm accept where node =  $[Q, \text{String}]$  and every arc costs 1

$$h([Q, \text{String}]) = \text{length}(\text{String})$$

- ▶ Prolog search where node = list of propositions to prove, and every arc costs 1

$$h(\text{List}) = \text{length}(\text{List})$$

- ▶ Node = point on a Euclidean plane, cost = distance between nodes, goal is a point G

$$h(\text{Node}) = \text{straight-line distance to G}$$

- ▶ estimate assuming lots of arcs (simplifying the problem)

## Best-first search

Form `NewFrontier` = `[Head|Tail]` such that

$h(\text{Head}) \leq h(\text{Node})$  for every Node in Tail

## Best-first search

Form  $\text{NewFrontier} = [\text{Head}|\text{Tail}]$  such that

$$h(\text{Head}) \leq h(\text{Node}) \text{ for every Node in Tail}$$

