Artificial Intelligence:
Search Methods for Problem Solving
A*: Pruning CLOSED and OPEN

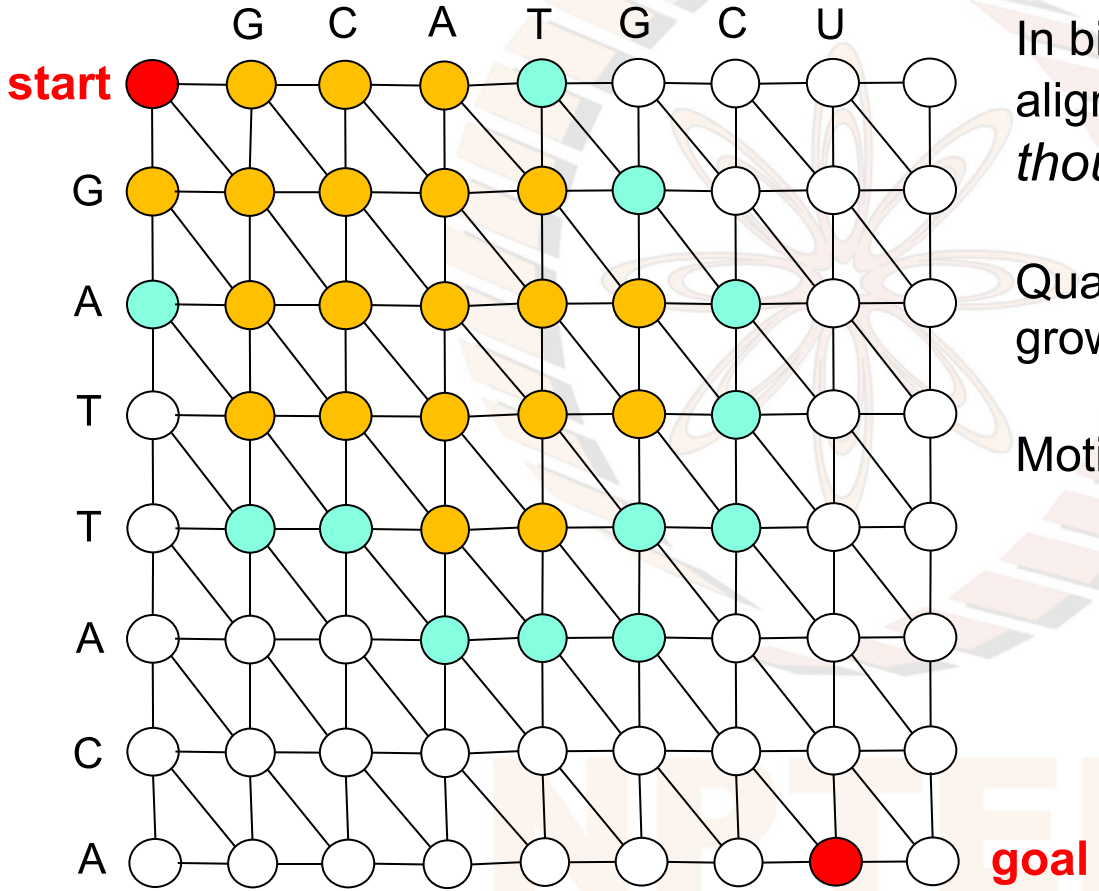
A First Course in Artificial Intelligence: Chapter 5

Deepak Khemani

Department of Computer Science & Engineering

IIT Madras

OPEN grows Linearly, CLOSED quadratically (recap)

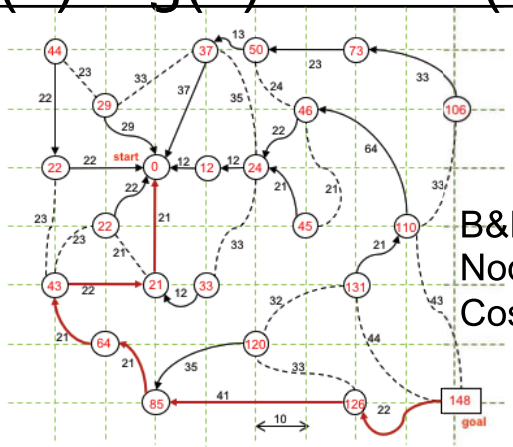


In biology the sequences to be aligned may have *hundreds of thousands* of characters.

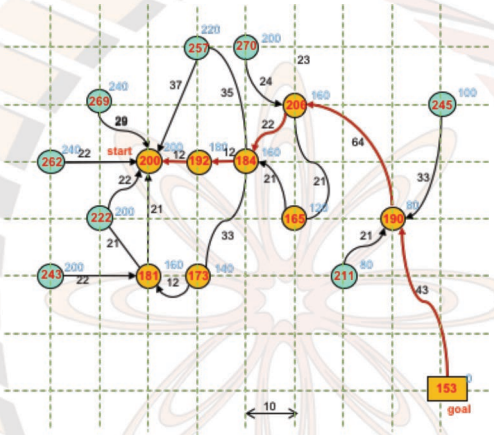
Quadratic is then a formidable growth rate.

Motivation to prune CLOSED

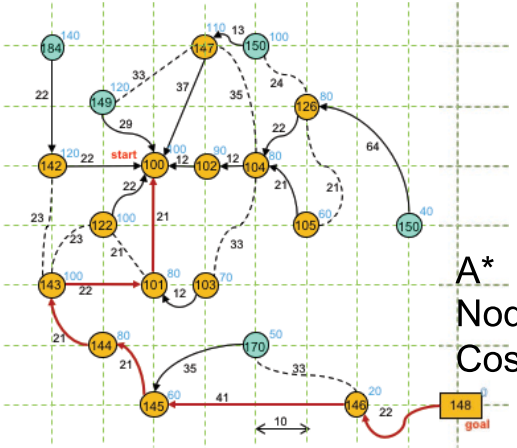
f(n) = g(n) + w × h(n): B&B (recap)



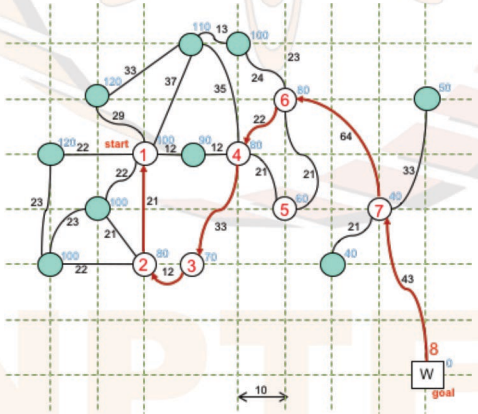
B&B
Nodes 23
Cost 148



wA*
Nodes 9
Cost 153



A*
Nodes 14
Cost 148



Best First
Nodes 8
Cost 195

As the weight of h(n) increases search requires less space,

but,

eventually becomes inadmissible

The Monotone Condition (recap)

The *monotone property* or the *consistency property* for a heuristic function says that for a node *n* that is a *successor* to a node *m* on a path to the goal being constructed by the algorithm *A** using the heuristic function h(x),

$$h(m) - h(n) \leq k(m,n)$$

The heuristic function

underestimates the cost of each edge

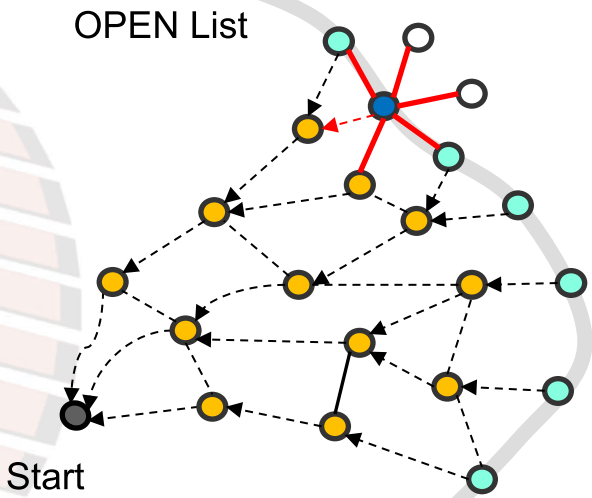
For *A** the interesting consequence is that every time it picks a node for expansion, it has found an optimal path to that node.

We can design algorithms that never look back!

The role of CLOSED in Search

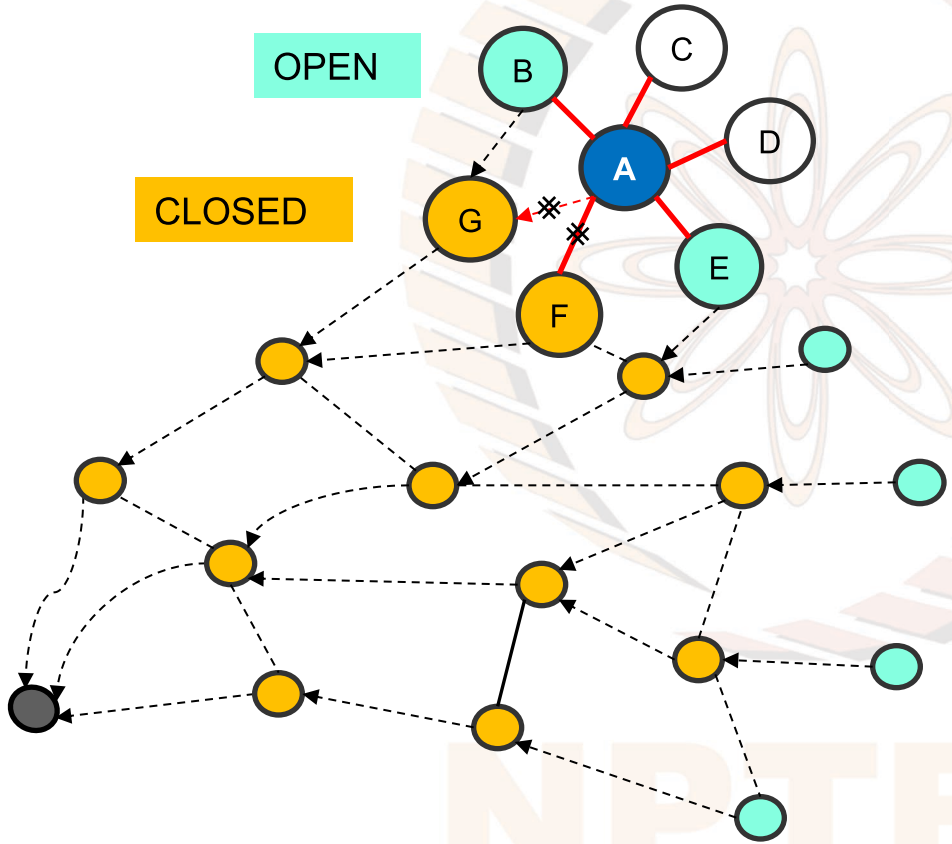
We needed to maintain the CLOSED list* of nodes for two reasons

- One, to avoid getting into infinite loops, which happen because a node on CLOSED may be a neighbour of the node being expanded.
  - We will looks at other ways to avoid the search from "*leaking back*"
- Two, to reconstruct the path once the goal node is picked up by the algorithm.
  - We will look at a new mechanism to do so



* In practice CLOSED should be implemented using a hash table for efficiency

Korf and Zhang, 2000: Frontier Search

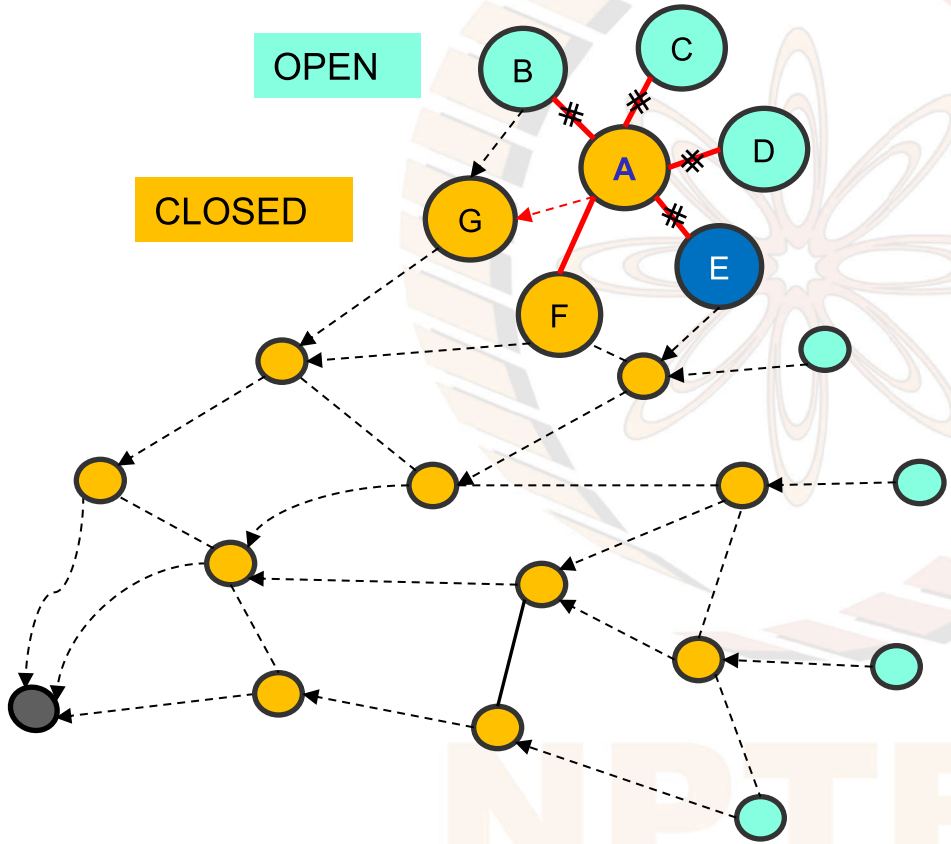


Let us say node A is about to be expanded.

Its *active* neighbours are
C,D (new nodes)
B,E (on OPEN)

Nodes G and F
(on CLOSED) are barred
(see next slide)

Korf and Zhang, 2000: Frontier Search

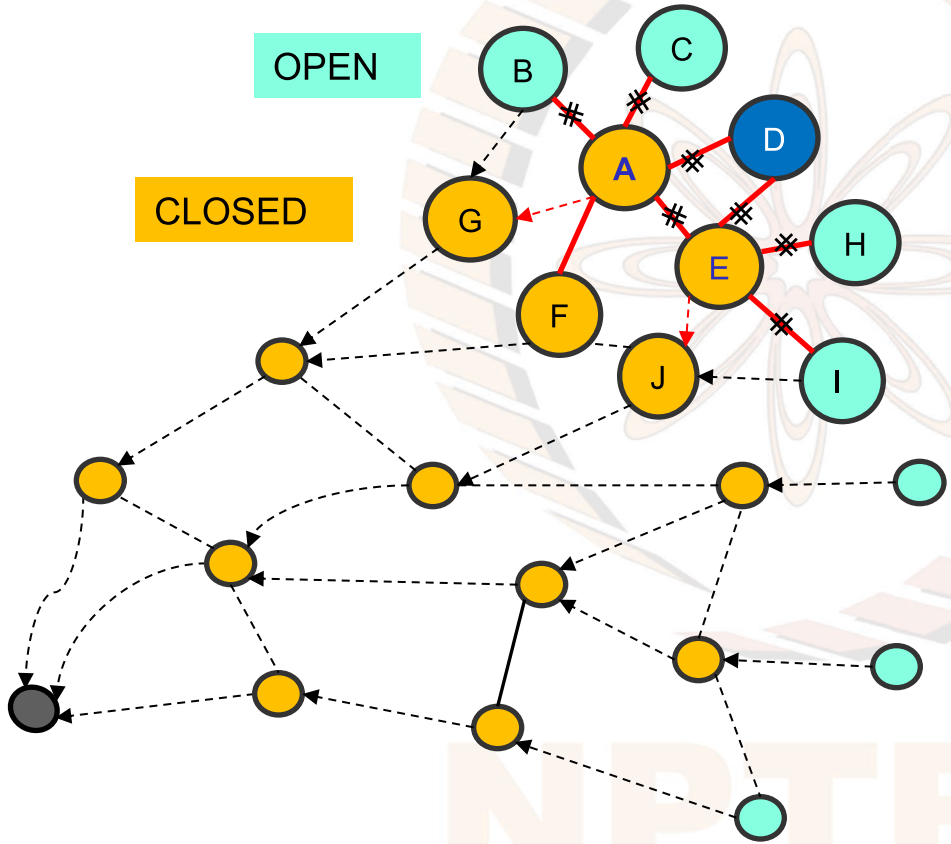


A is added to CLOSED
and its *active* neighbours
C,D,B, and E are barred from
generating A
when *they* are expanded

Remember A* has already
found the *optimal path* to A

Let E be the next node visited

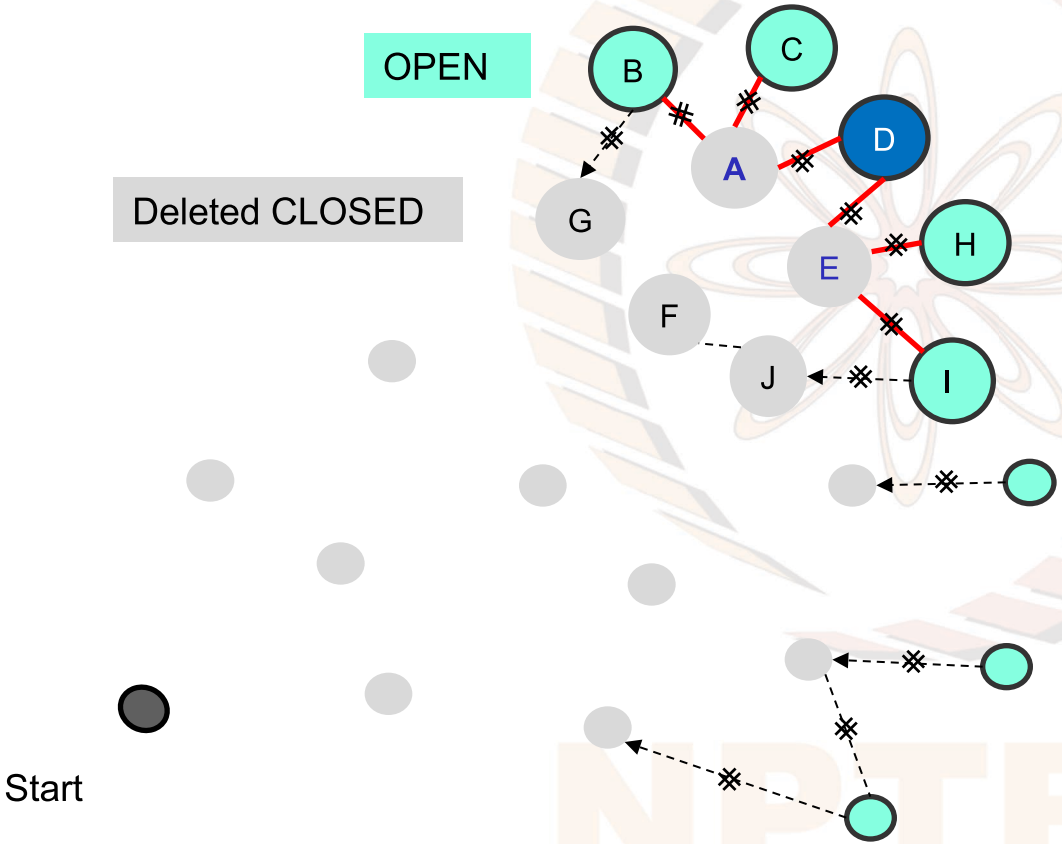
Korf and Zhang, 2000: Frontier Search



When E is added to CLOSED its *active* neighbours D,H, and I are barred from generating E when *they* are expanded

So D will not generate either of A and E

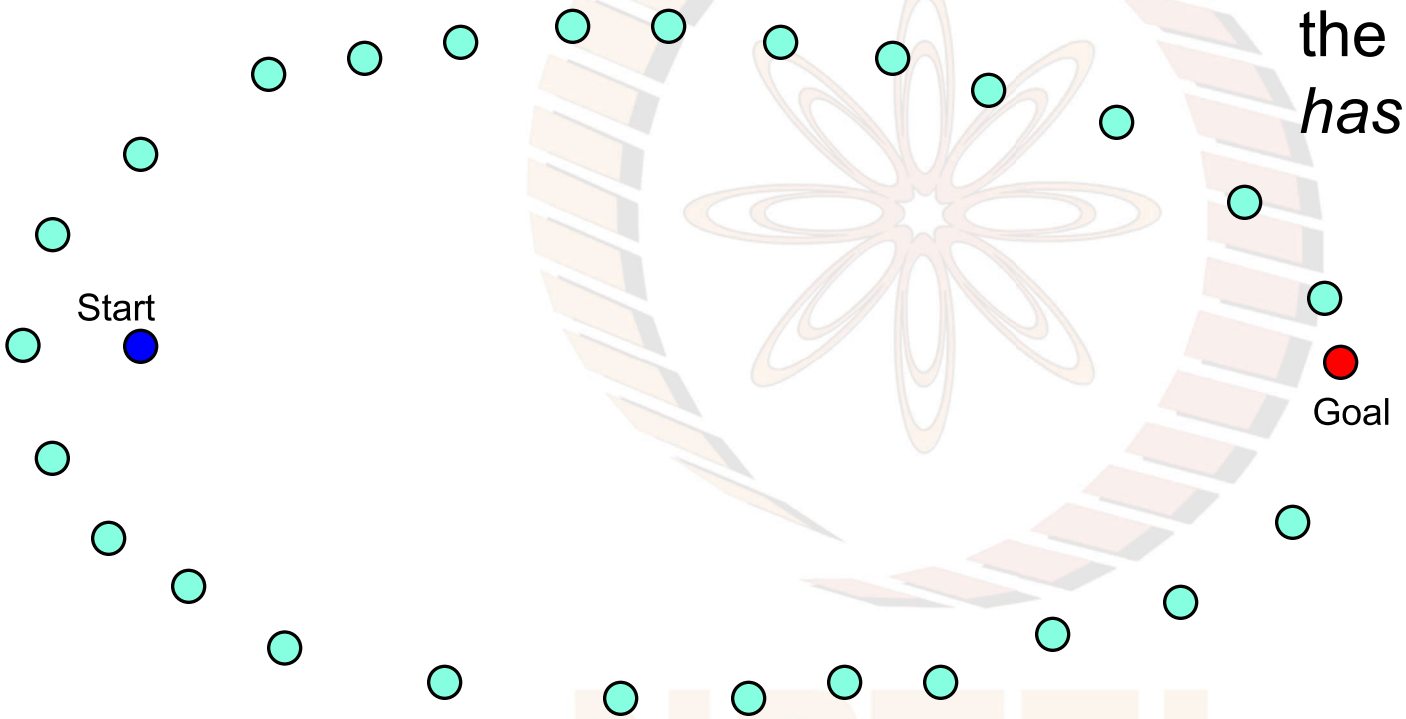
Korf and Zhang, 2000: Frontier Search



Frontier Search:
Keep only the OPEN nodes, along with list of barred (tabu) nodes with each node in OPEN.

Search only moves forward, without “leaking back”

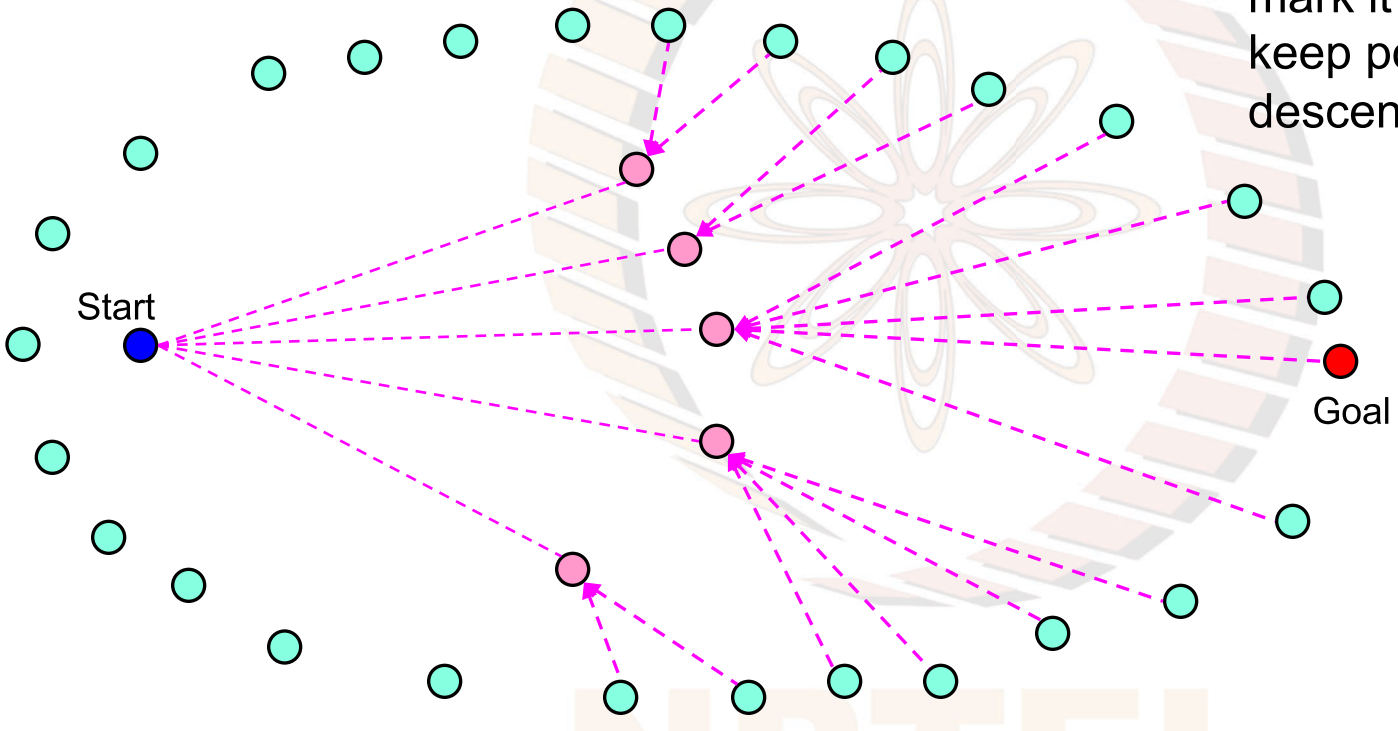
When Frontier Search picks the goal node...



...it has *no means of reconstructing* the optimal path it *has* found

Relay nodes: at roughly the half way mark

When a node on OPEN has g(n) ≈ h(n) mark it as relay and keep pointers from its descendants on OPEN



When Frontier Search picks the goal node...

...it has a pointer to its relay node Relay

Solve two recursive problems

1. From Start to Relay
2. From Relay to Goal
   - Divide and Conquer Frontier Search (DCFS)

If T(d) is the time complexity needed to find the goal at *d* steps then time complexity of DCFS is

$$T(DCFS) = T(d) + 2 \times T(d/2) + 4 \times T(d/4) \ldots$$

If T is exponential, then $T(DCFS) = T(d) \times d$

Smart Memory Graph Search (SMGS): Zhou and Hansen (2003)

Given that memory is getting cheaper and abundant, one *need not* make recursive calls when A* (without pruning) *could have* solved the problem!

Smart Memory Graph Search keeps track of available memory.

Only when it *senses memory is running out* it creates a relay layer.

In the process it *might create many relay layers*

.... or *even none* if the problem is small enough to be solved by A*.

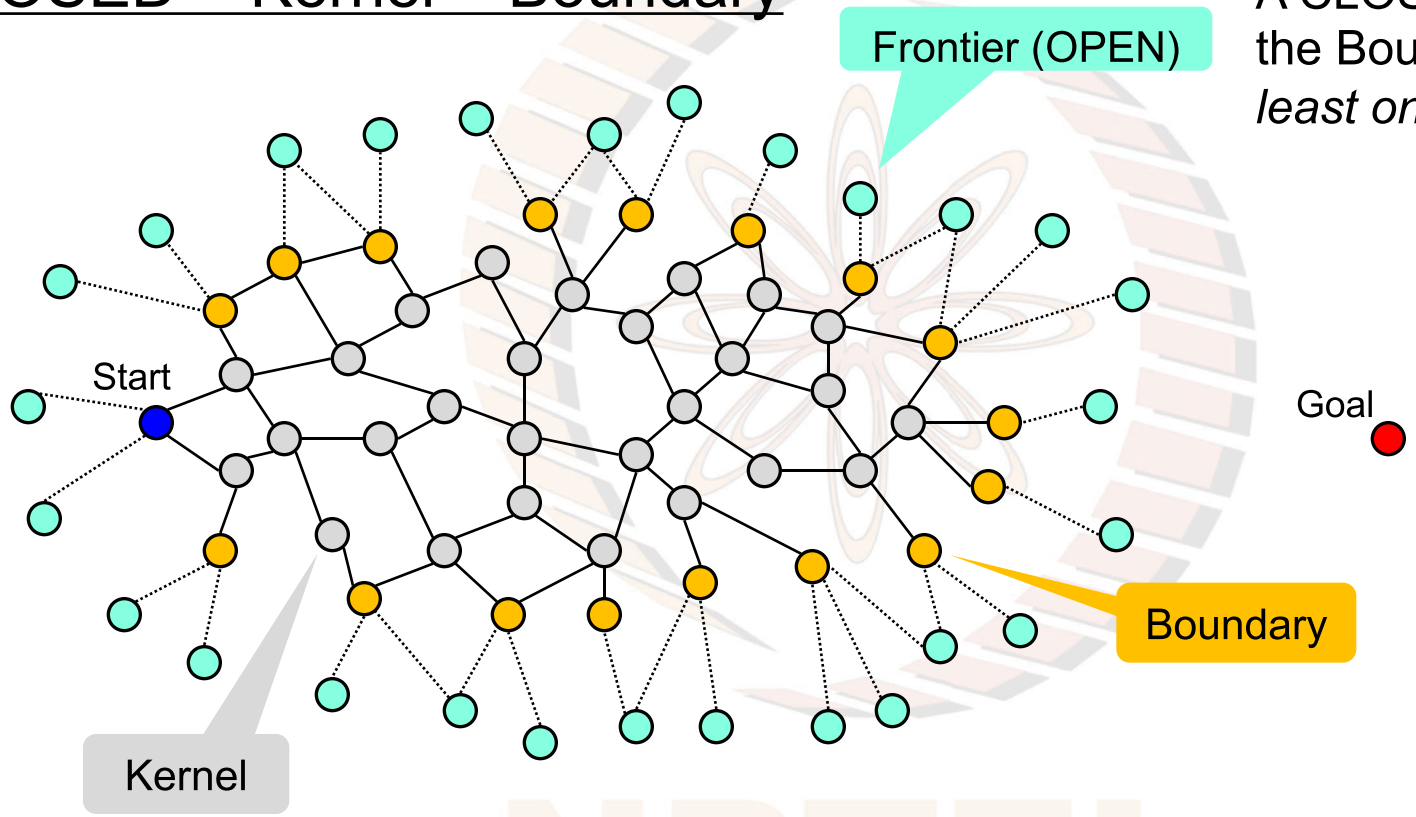
At all points it identifies a layer of *Boundary nodes*

that can be *potentially converted into Relay nodes.*

The boundary nodes also *stop the search from leaking back*!

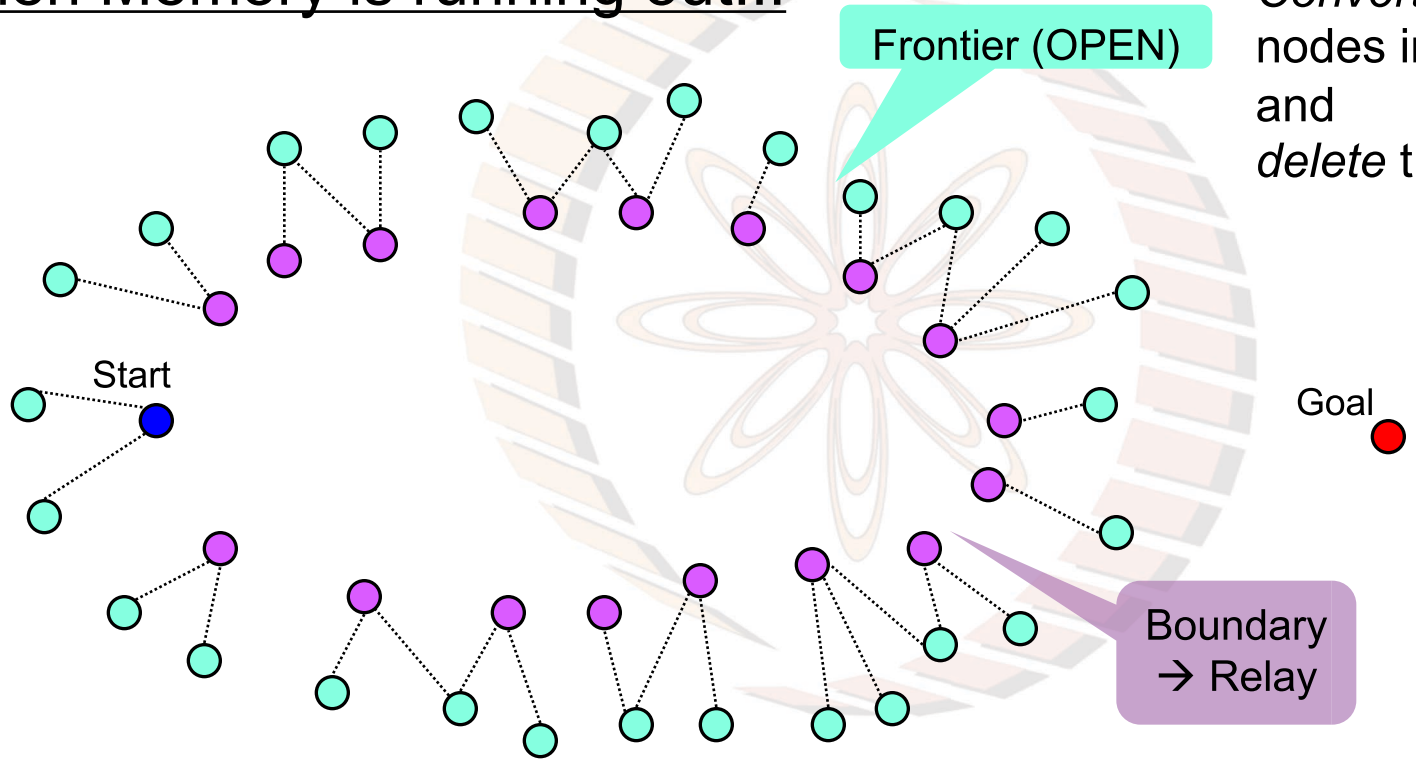
CLOSED = Kernel + Boundary

A CLOSED node is on the Boundary if it has at *least one child* on OPEN



A CLOSED node is on the Kernel if *all its neighbours* are in CLOSED

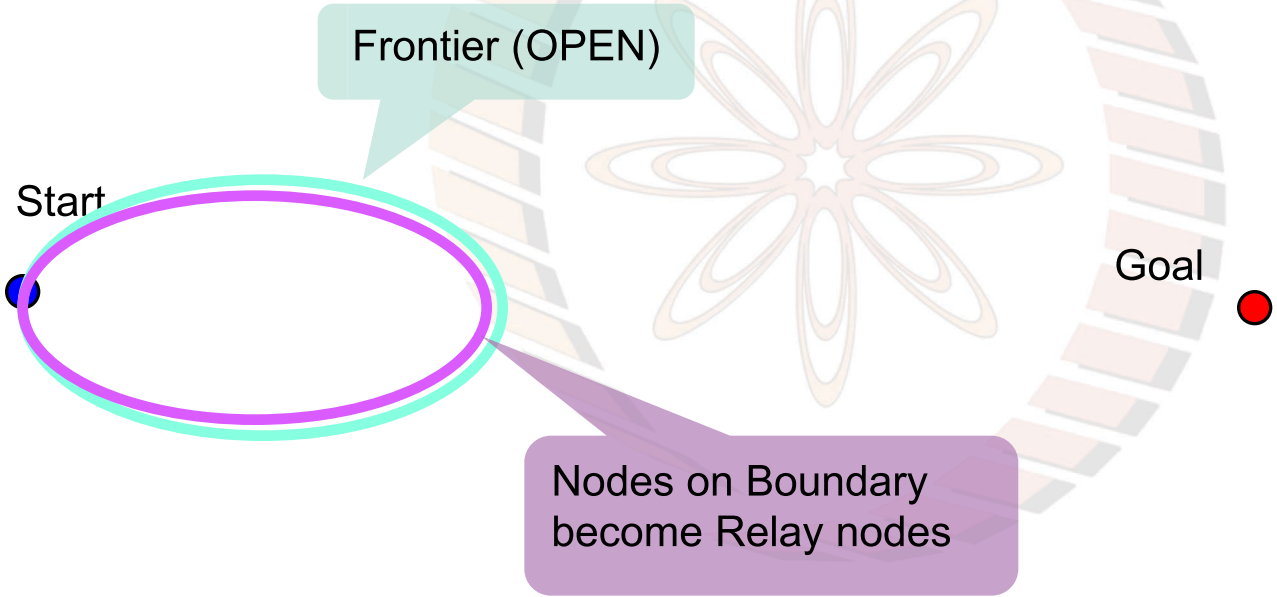
When Memory is running out...



Convert the boundary nodes into relay nodes, and
delete the kernel nodes

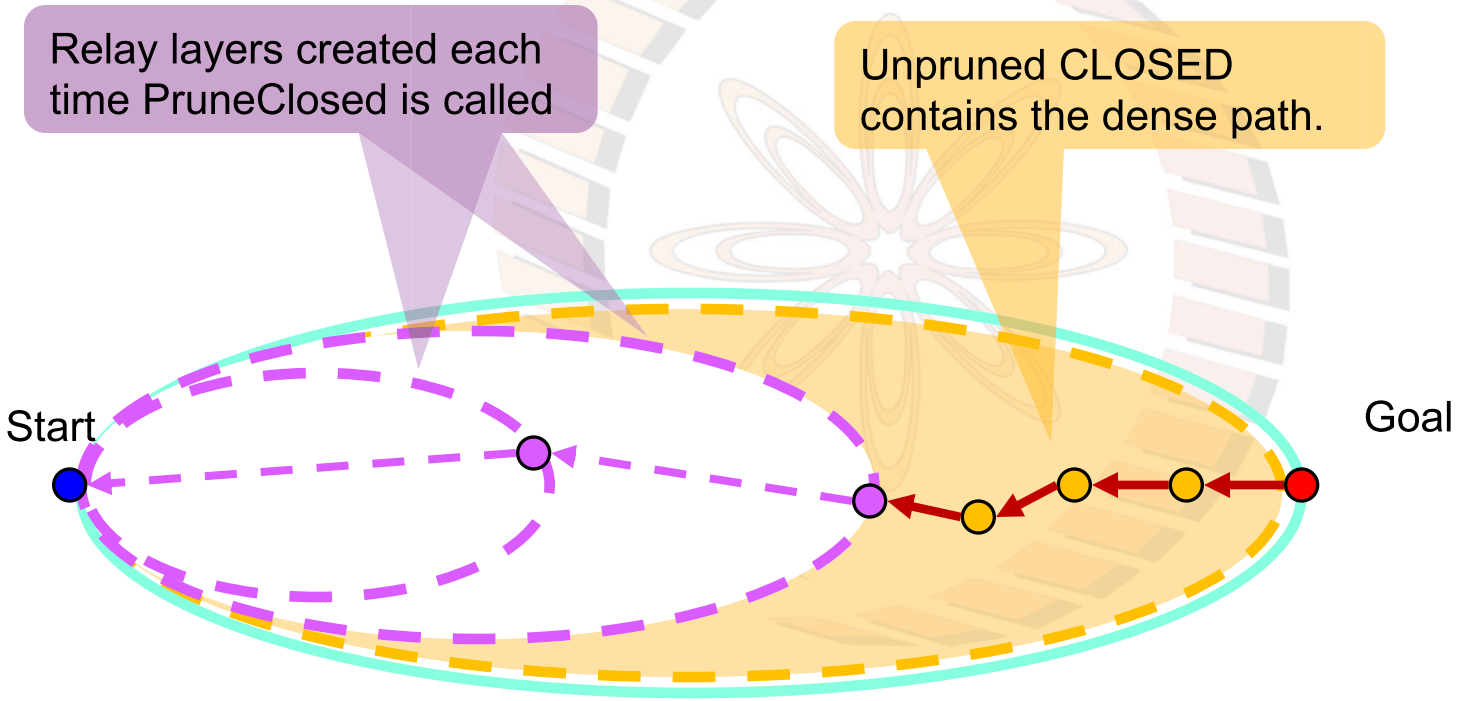
Boundary nodes in the CLOSED list
are enough to prevent search from “leaking” back.

A relay layer when needed



SMGS converts boundary nodes into Relay when prune module is called.

SMGS breaks up the problem into many subproblems if needed



On termination SMGS has a Sparse Solution Path to the goal.
It makes an appropriate number of recursive calls

Next

Pruning OPEN

Beam Search (with beam width w)

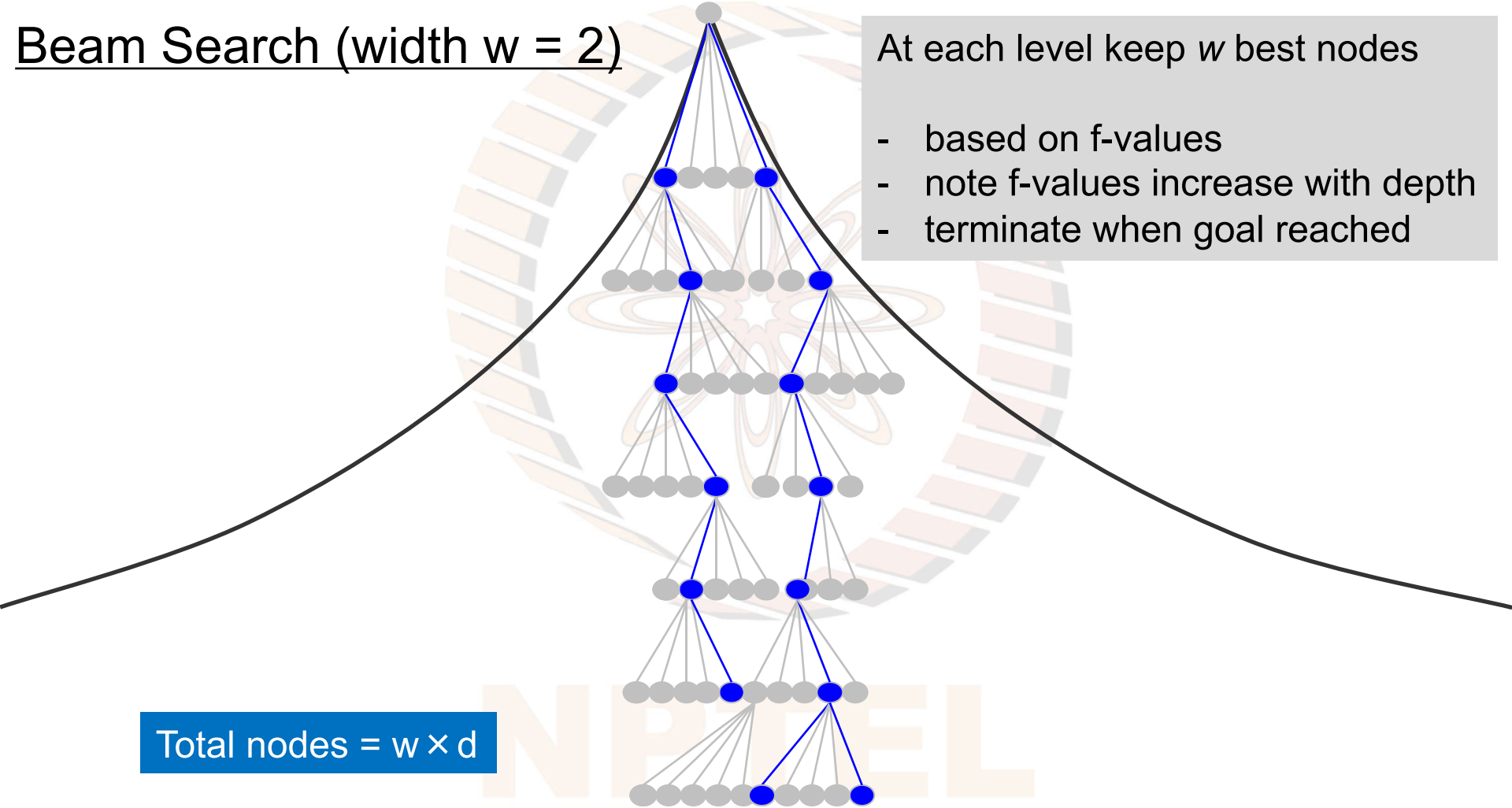
We looked at Beam Search as a variation of Hill Climbing

- Optimizing the *h-value*
- Hill Climbing chose one best successor (only if better)
- Beam Search keeps *w* best successors (only if better)
- Termination criterion – no better successors → may be local optima

Consider Beam Search in the context of finding the goal node

- Optimizing the *g-value of goal node*
- Nodes chosen based on *f-values*
- *f-values* are known to increase with depth
- Beam Search keeps *w* best successors (*even if not better*)
- Termination criterion – goal found
- Will *always* find a path – may not be optimal
- **Total space required = width × depth**

Beam Search (width w = 2)



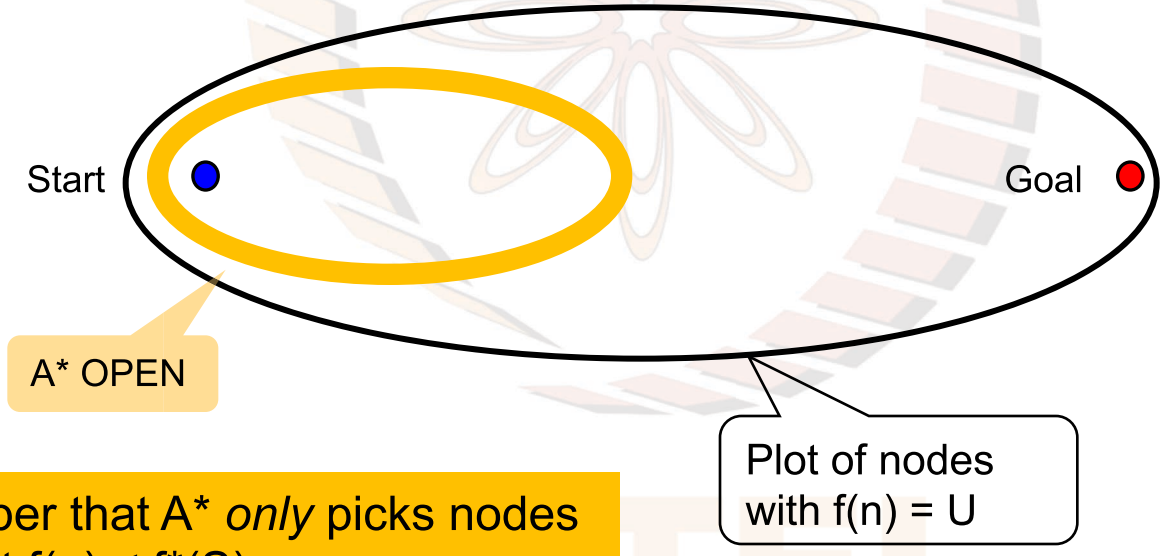
At each level keep *w* best nodes

- based on f-values
- note f-values increase with depth
- terminate when goal reached

Total nodes = w × d

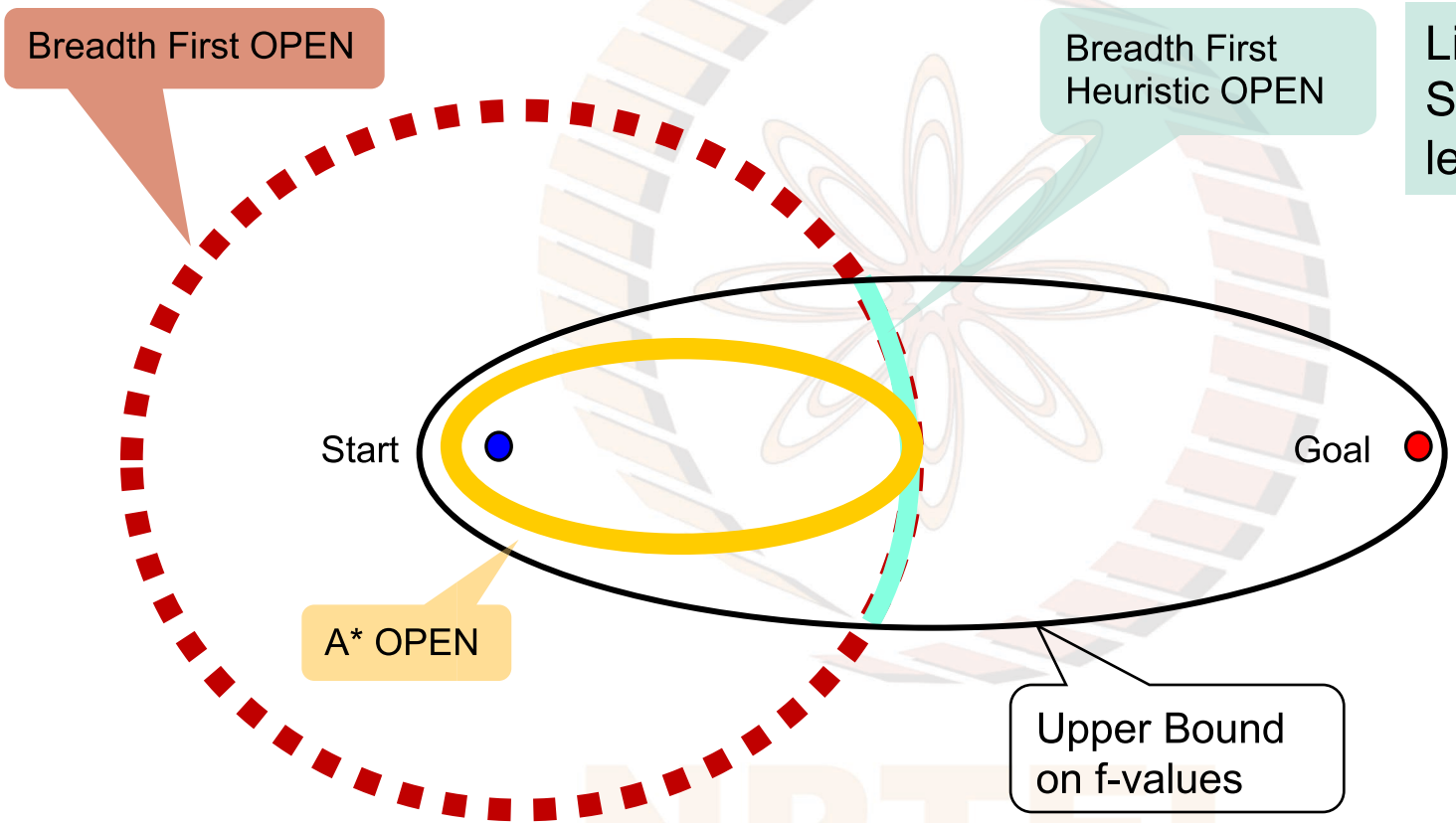
An upper bound on cost of solution

- Find any path from the start to the goal node (could use Beam Search)
- Let the cost of the path be U
- U is an upper bound on the cost of the optimal path
- *Need not* look at nodes with f(n) > U

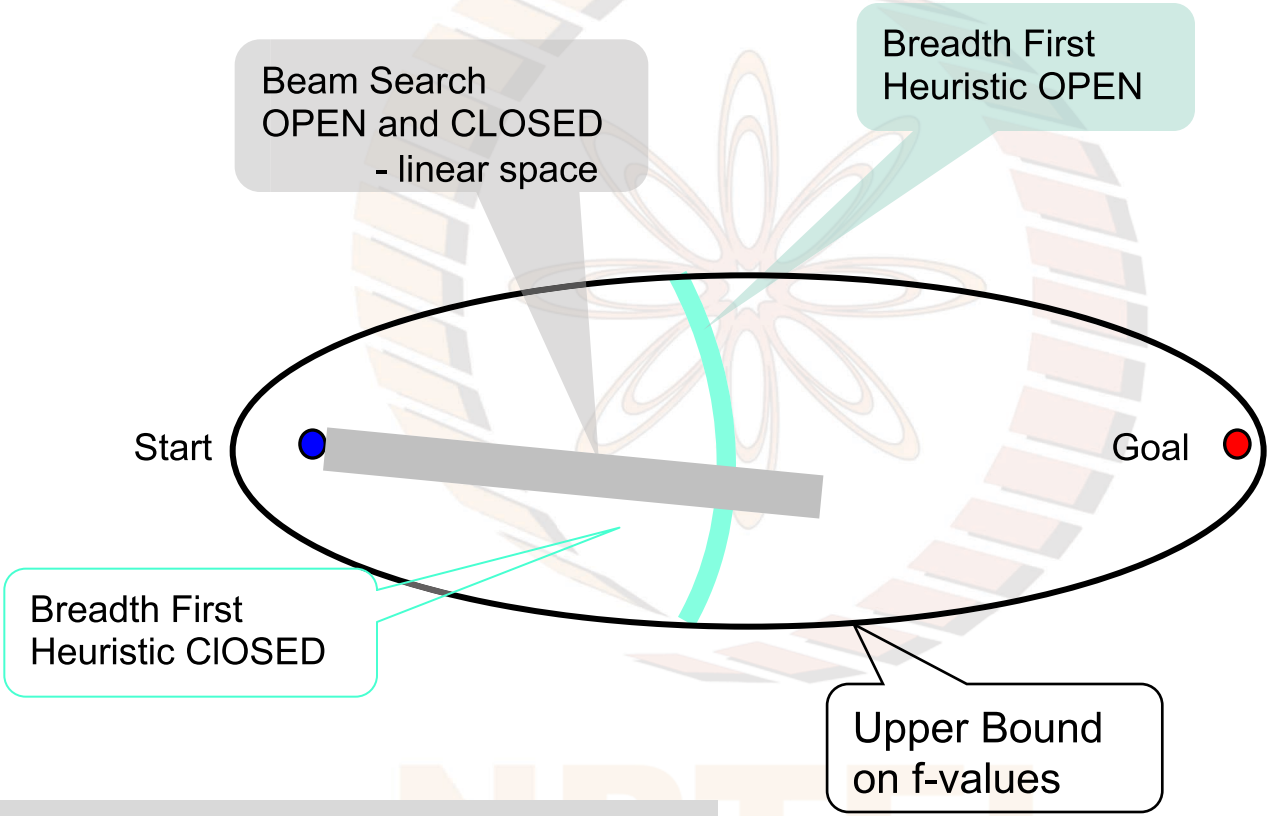


Remember that A* *only* picks nodes such that f(n) ≤ f*(S)

Breadth First Heuristic Search (BFHS) – Zhou and Hansen, 2004



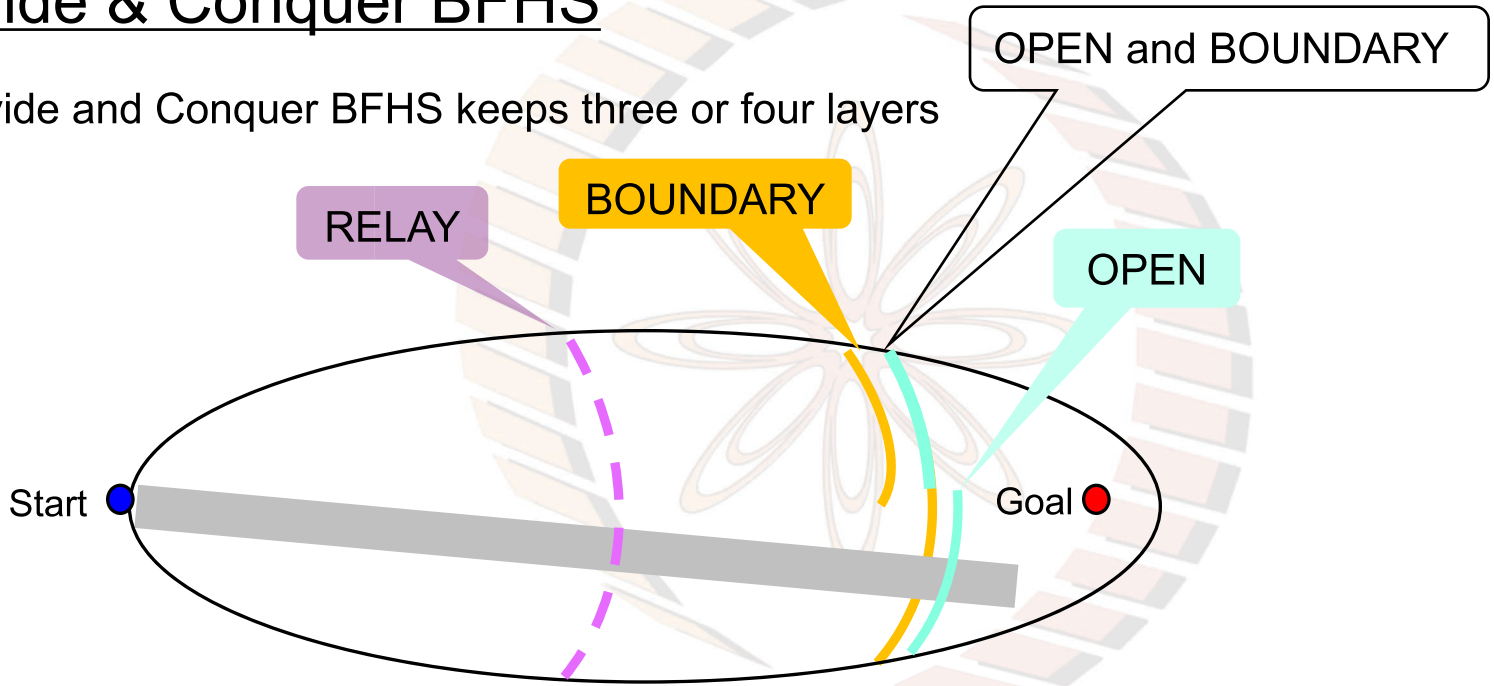
Beam Search



Warning: Beam Search is inadmissible

Divide & Conquer BFHS

Divide and Conquer BFHS keeps three or four layers



Divide and Conquer Beam Search keeps three layers of constant width

Beam Stack Search: Zhou and Hansen, 2005

Beam Search is inadmissible, may terminate with a non optimal path

Beam Stack Search is like Beam Search + Backtracking

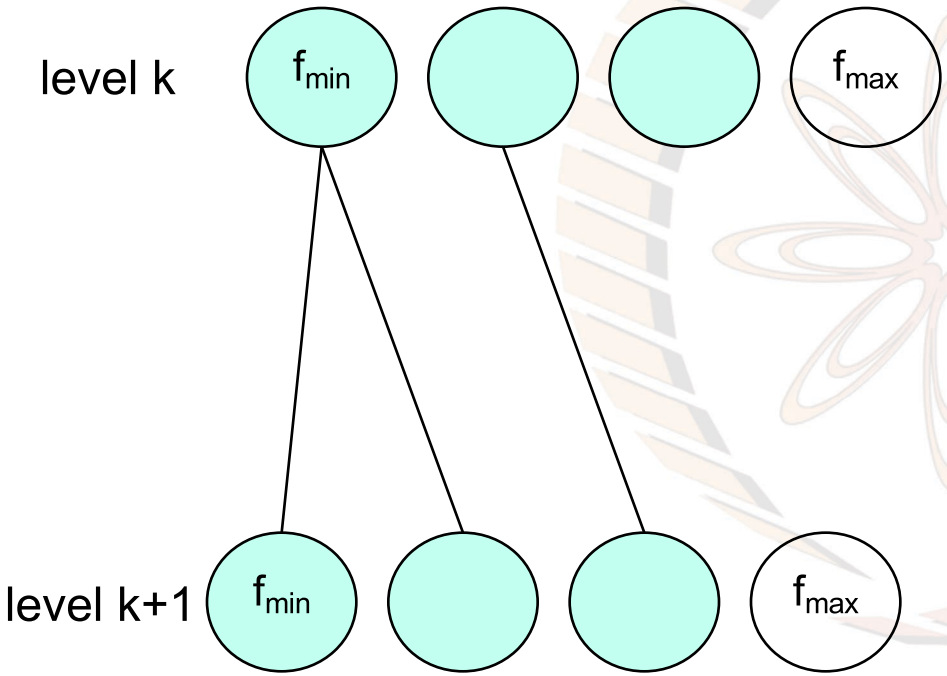
Backtracking is ***explicit*** guided by a Beam Stack

- stack contains pairs of values $[f_{min}, f_{max})$ at each level
- the pair of values identify the current nodes in the beam
- used to guide the regeneration of nodes *while* backtracking

Beam Stack Search orders nodes on f-values.

It slides the f_{min},f_{max} window on backtracking,
but does not go beyond the upper bound U

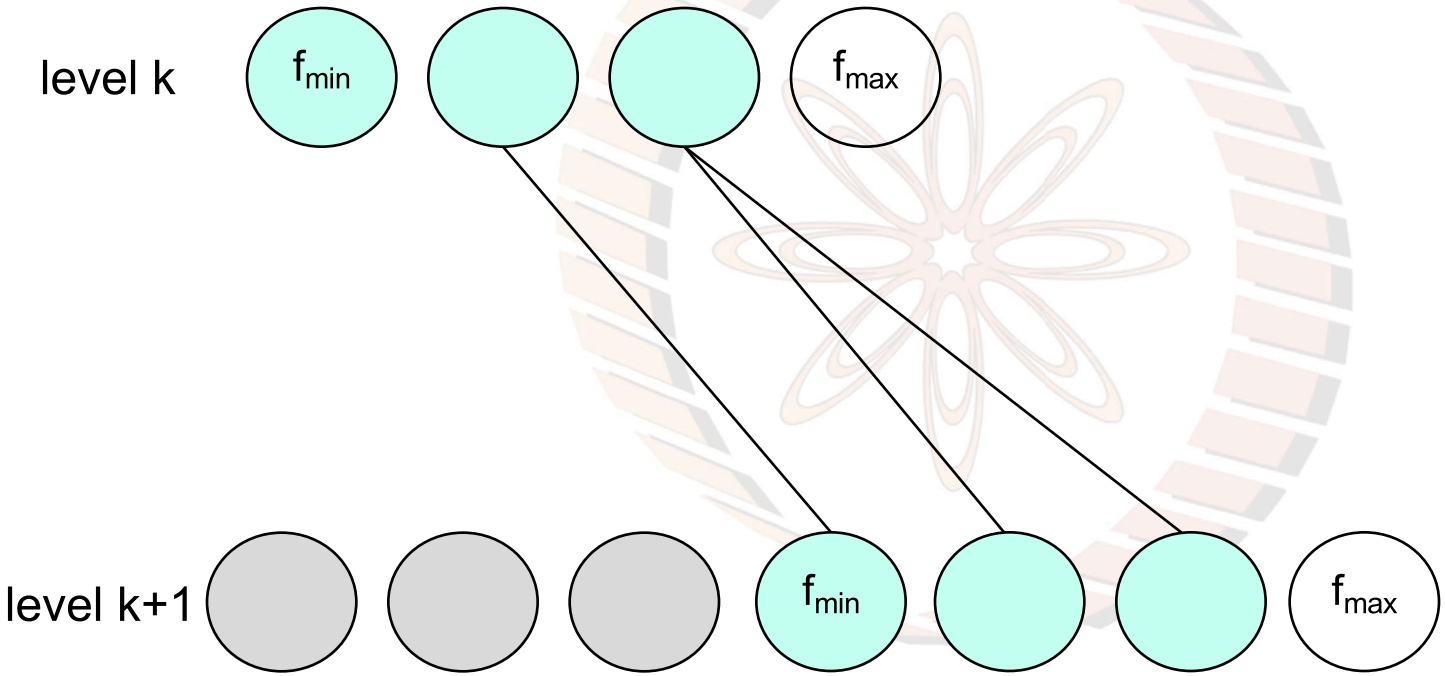
Beam width = 3



The Beam Stack of pairs of values $[f_{min}, f_{max})$ where,
f_{min} is *lowest value in the beam*,
and
f_{max} is the lowest value ***not*** in the beam

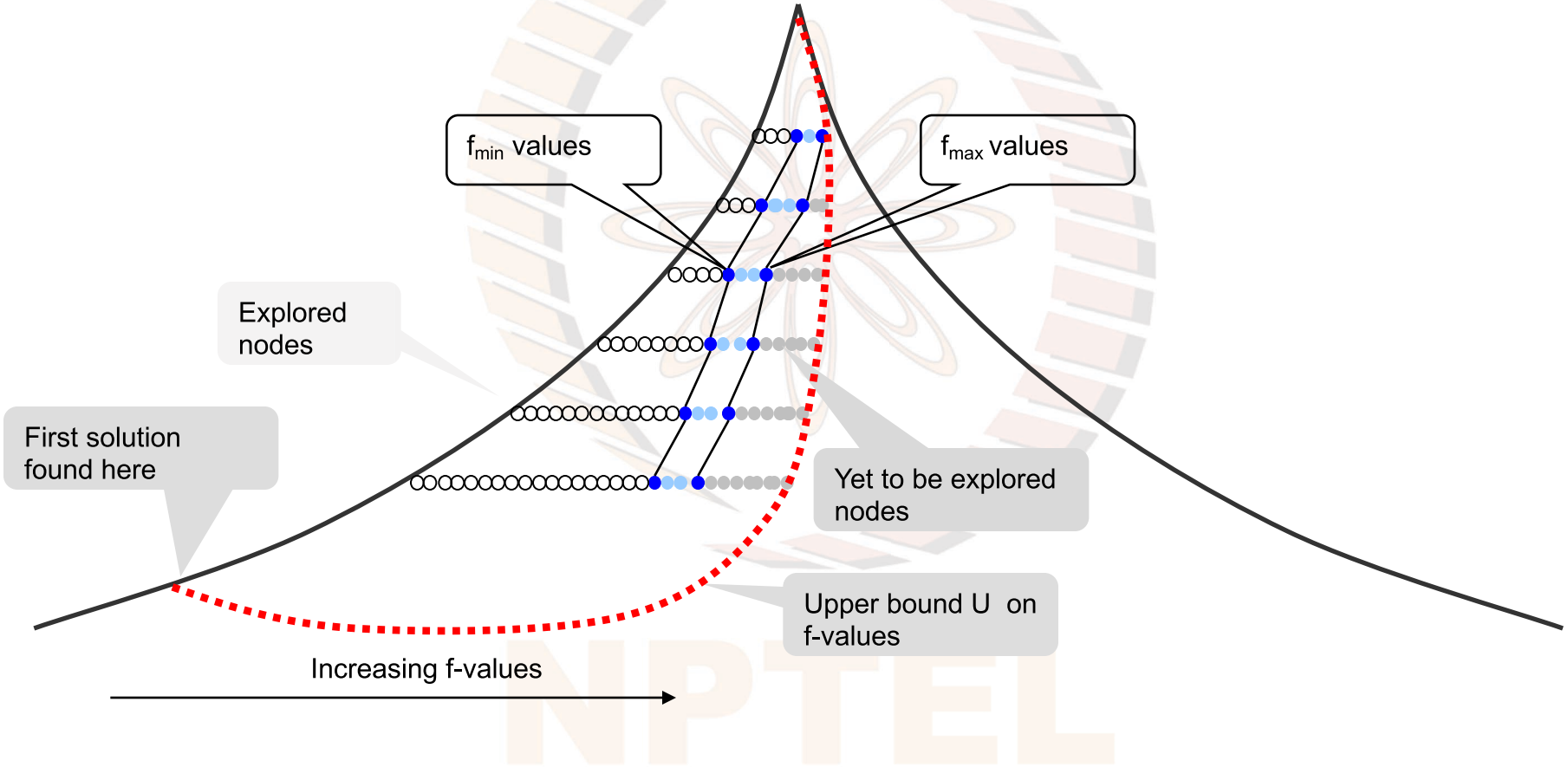
If you want to backtrack from level k+1.....

Include children starting with f_{max}

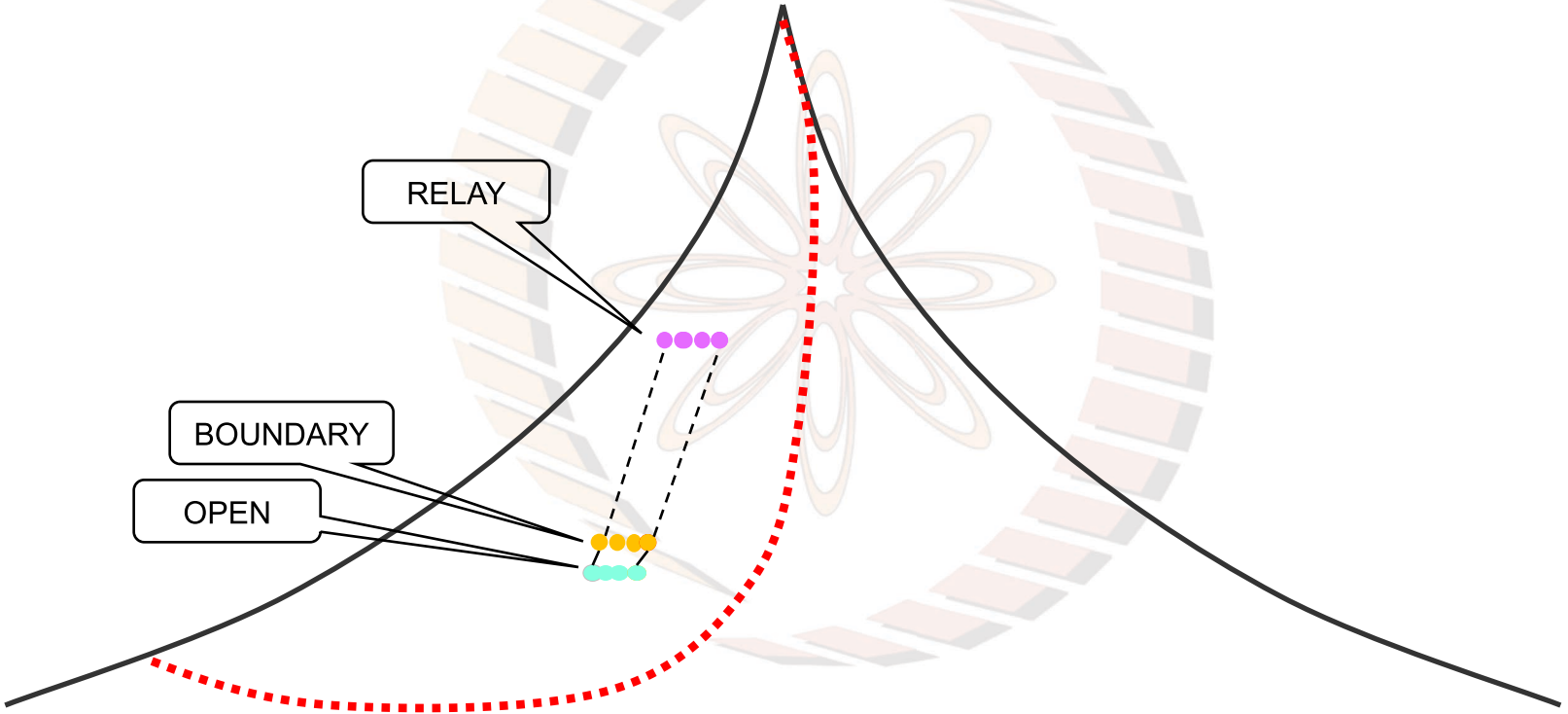


Backtrack to level k, generate children again, and keep 3 nodes starting with the old f_{max} which becomes the new f_{min}

Beam Stack Search: Zhou and Hansen, 2005



Divide & Conquer Beam Stack Search: Zhou & Hansen, 2005



DCBSS stores three layers of width w.

Divide & Conquer Beam Stack Search backtracking

DCBSS has deleted the parents of nodes in the beam

How does it backtrack then?

Answer:

- Regenerate nodes from the start state
- Guided at each level by the values in the beam stack

Space complexity of DCBSS is O(1)

- three layers of constant width
- relay, boundary and open
- *if you ignore* the memory needed by the beam stack

The End

A* and its variations