Optimal Efficiency of A* Revisited

Research Lecture on work by
Mike Barley & Jorn Christensen

39 years ago, in 1968, Peter Hart, Nils Nilsson, & Bertram Raphael published

"A Formal Basis for the Heuristic Determination of Minimum Cost Paths"

It described a new search algorithm: A*

A* is the most widely used search algorithm today!

A* Algorithm

Mark *s* open and calculate *f(s)*.

While there are open nodes Do

Select open node *n* with smallest *f* value

resolve ties in favor of goal nodes,

If *goal(n)* then terminate with success

Mark *n* closed

For all successors, *j*, of *n* Do

calc *f(j)*

if *j* not in closed or *f(j)* is lower

mark *j* open.

A* Optimal Efficiency Result #1

Let A be any optimal algorithm dominated[1] by A* where $f(n) = f(m)$ implies $n = m$[2],

Then A* is at least as efficient[3] as A.

1. A^*_{h1} *dominates* A^*_{h2} if and only if for all non-goal nodes, n, $h1(n) >= h2(n)$.
2. The *no ties* clause.
3. X is as *efficient* as Y means every node expanded by X is also expanded by Y.

No Ties Clause

- OE Result #1 is not very useful, because the “no ties” limitations is too restrictive.

- We need to allow ties.

Handling Ties

The specification of A* does not state exactly how to handle ties. This means that there are many different refinements of A*, each differing in the way they order ties. ***A**** is the set of A* refinements that have different tie handling strategies.

A* Optimal Efficiency Result #2

Let A be any optimal algorithm that is "dominated" by every algorithm in ***A****

Then there exists an A* in ***A**** such that A* is at least as efficient as A.

Is this good enough?

- This only tells us that some A* is optimally efficient but not which one!!!!!
- Also it doesn’t tell us what algorithms any given A* is as efficient as.

A* Optimal Efficiency Result #3

If h_1 is *less informed*[1] *than* h_2
then A^*_{h2} is at least as efficient as A^*_{h1}

1. Heuristic h_1 *is less informed than* h_2 iff
for all non-goal nodes, n, $h_1(n) < h_2(n)$.

Where that leaves us

- *Non-Optimal Efficiency:* Don't know whether an A^*_h will be "as or more efficient" than any other optimal search algorithms that *h* dominates.
- *Heuristic Non-Equivalence:* Given $A^{*\prime}_h$ & A^*_h, they may not expand the same number of nodes.
- *Non-Monotonic Improvement:* Given an "improved" heuristic h_1, A^*_{h1} may be less efficient than A^*_h.
- *Inconsistent Performance:* If we run A^*_h twice in a row, we don't know whether we will get the same number of nodes both times.

What happened?

- If we don‘t allow ties then we get the result we want.

- If we allow ties then we have two very weak (and unsatisfactory) results.

A* Algorithm - Handling Ties

Mark *s* open and calculate *f(s)*.

While there are open nodes Do

Select open node *n* with smallest *f* value

resolve ties in favor of goal nodes,

If *goal(n)* then terminate with success

Mark *n* closed

For all successors, *j*, of *n* Do

calc *f(j)*

if *j* not in closed or *f(j)* is lower

mark *j* open.

Handling Ties

The specification of A* does state exactly that in case of ties in the open list, to always choose goal nodes over non-goal nodes with the same *f*-value.

So why are ties a problem & are all ties a problem?

Critical Ties:The Problem?

HNR defined *critical ties* as those nodes with the same *f*-value as optimal goal node.

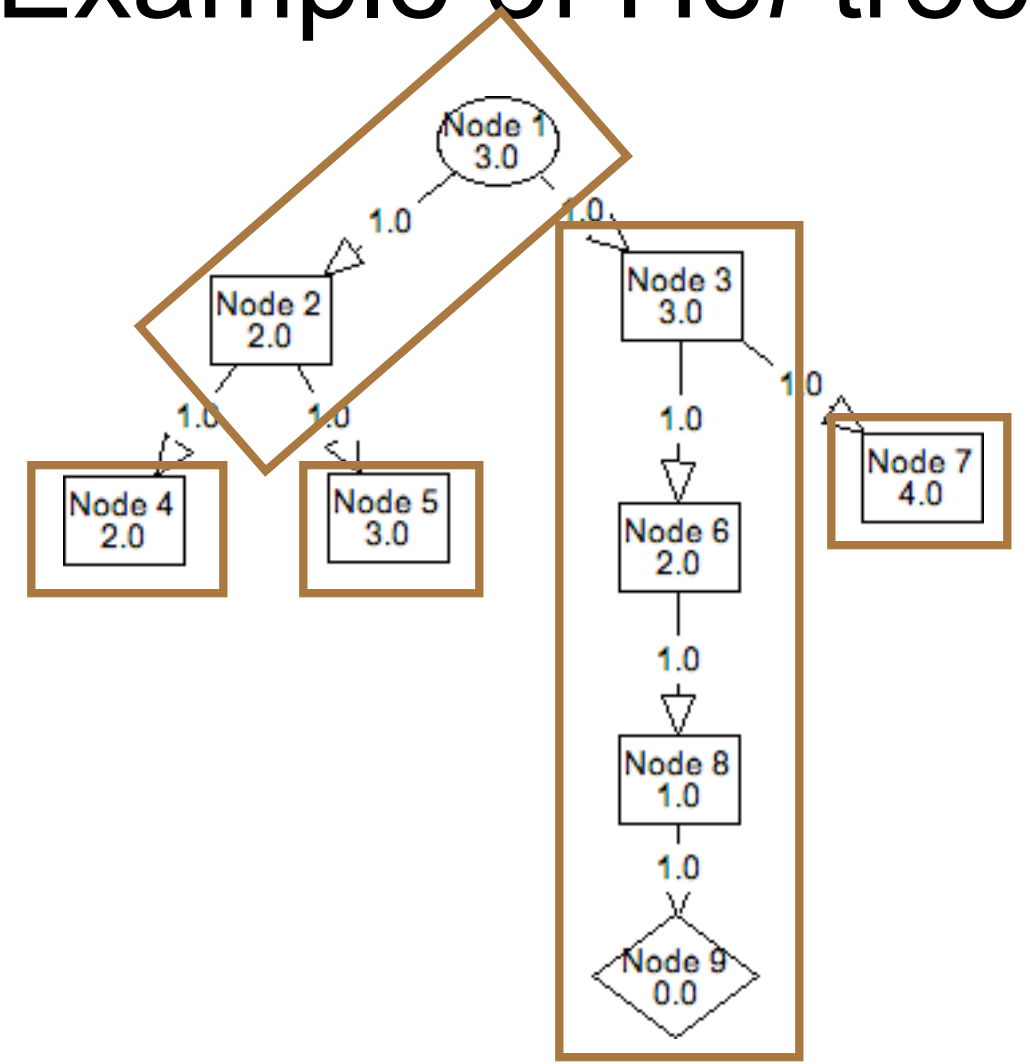
For them, the existence of critical ties was the reason why result #1 had to have the no ties clause.

Are all critical ties a problem?

Tracking down the culprit: Ho*f* trees

- A homogeneous *f*-value (Ho*f*) tree is a sub-tree of a search tree where all nodes have the same *f*-value, say **f**, where the parent (if there is one) of the root has a different *f*-value, and where all the children (if there are any) of the leaves have non-**f** *f*-values.

Example of H*of* trees



Why are Ho*f* trees problematic?

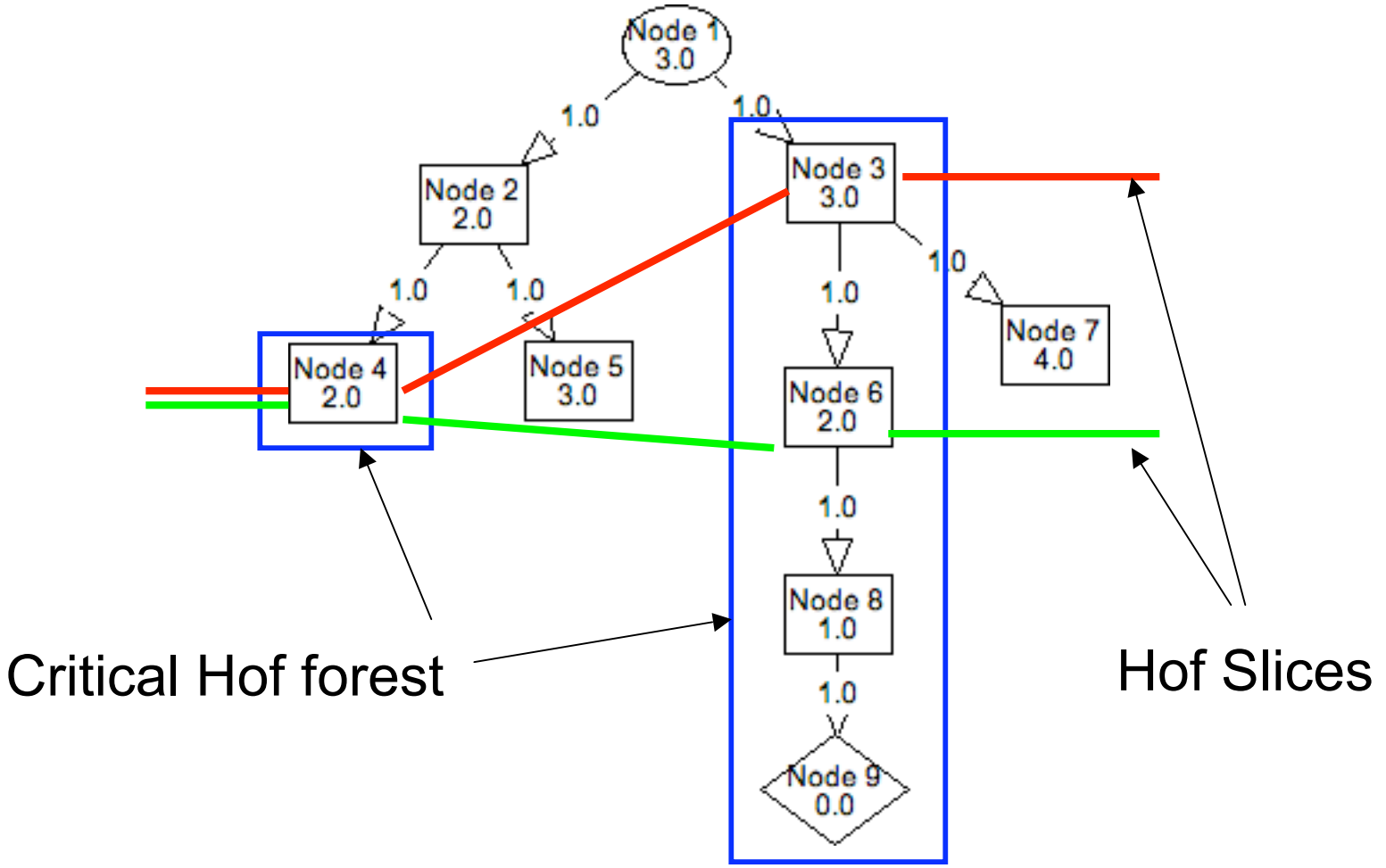
- To understand this we need to see what effect Ho*f* trees have on the open list.

- To understand we need to extend our vocabulary a little.

More Terminology

- A Ho*f* forest is the set of all the Ho*f* trees in a search tree with the same *f*-value.
- A Ho*f* slice is a “cut” through a Ho*f* forest.
- An *f*-open list is the open list sublist that contains all the nodes with the same *f*-value.
- A critical *f*-open list is a *f*-open list with same *f*-value as the optimal goal.

Example of a Ho*f* forest & slices



Tying Things Together

- An f-open list is a slice through a Hof forest.
- A critical f-open list is the set of critical ties that A* can see at a point in time.
- If the goal nodes are not in the current critical f-open list then A* doesn’t know where any of the optimal goals are.

Tying Things Together cont'd

- The different A*'s in ***A**** represent the different ways to access the nodes in the critical f-open list.
- Some A*'s will be luckier than others and pick the right node that leads to a goal node in the critical Hof forest.

An Aside

- While A* can't know which node in the current critical f-open list leads to a goal, it can increase its odds by choosing the node with the lowest *h* value.

Where We are Now

- We should understand now why allowing ties forces us to have such weak results.
- We now show how to get stronger results.
- First, we will look at what we want to get (but can’t currently).

What We Want (& Will Get)

1. *Consistent Performance:* If we run "A*$_h$" twice in a row, we want to get the same nodes both times.
2. *Monotonic Improvement:* Given an "improved" heuristic h_1, we want "A*$_{h1}$" to be at lest as efficient as "A*$_h$".
3. *Heuristic Equivalence*: Given "A*'$_h$", we want them both to expand the same nodes.
4. *Optimal Efficiency:* We want it to be at least as efficient as any other optimal search algorithms that *h* dominates.

How can we do this?

- We could “modify” A*.
- We could place more restrictions on the heuristics being used.
- We will look at both ways.

Modifying A*

- What type of modification should we be looking at?
- At least part of the problem seems to be that the handling of critical ties is underspecified.
- We need to refine our definition of A* to describe how ties are handled.

Consistent Performance

- How is it possible to get different nodes when running the same algorithm twice?
- Make the selection of nodes from the open list determined solely by either the structure of the search tree or by its traversal.
- Then will get consistent performance.

Monotonic Improvements

- Why don’t we currently have monotonic improvements?

- To understand this we have to look at how current open list access mechanisms lead to this problem.

Common Open List Access Mechanisms

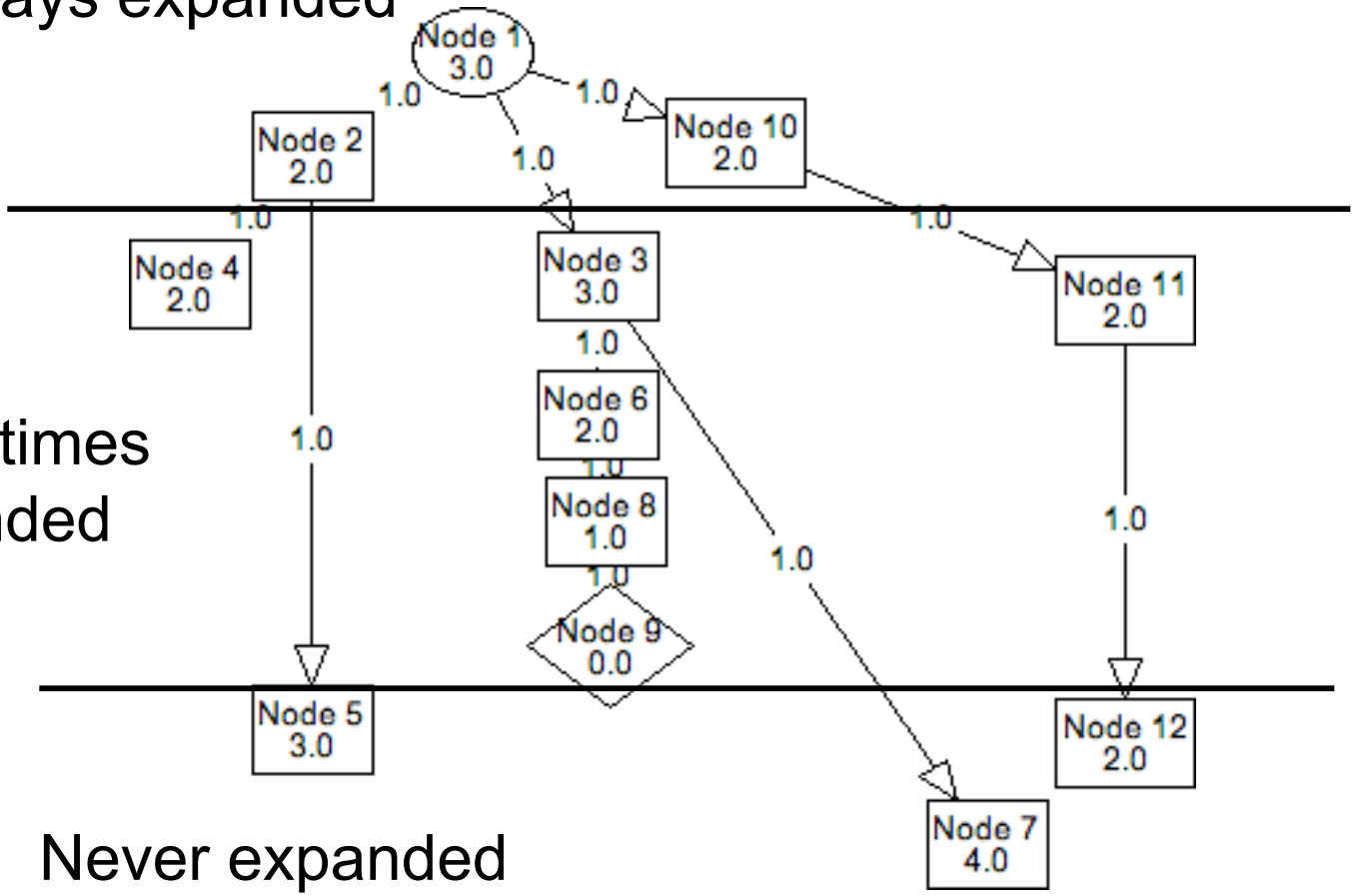
- Stack
- Queue
- Both of these leads to non-monotonic improvements.
- We will now look at why.

Search Tree Expansion

Aways expanded

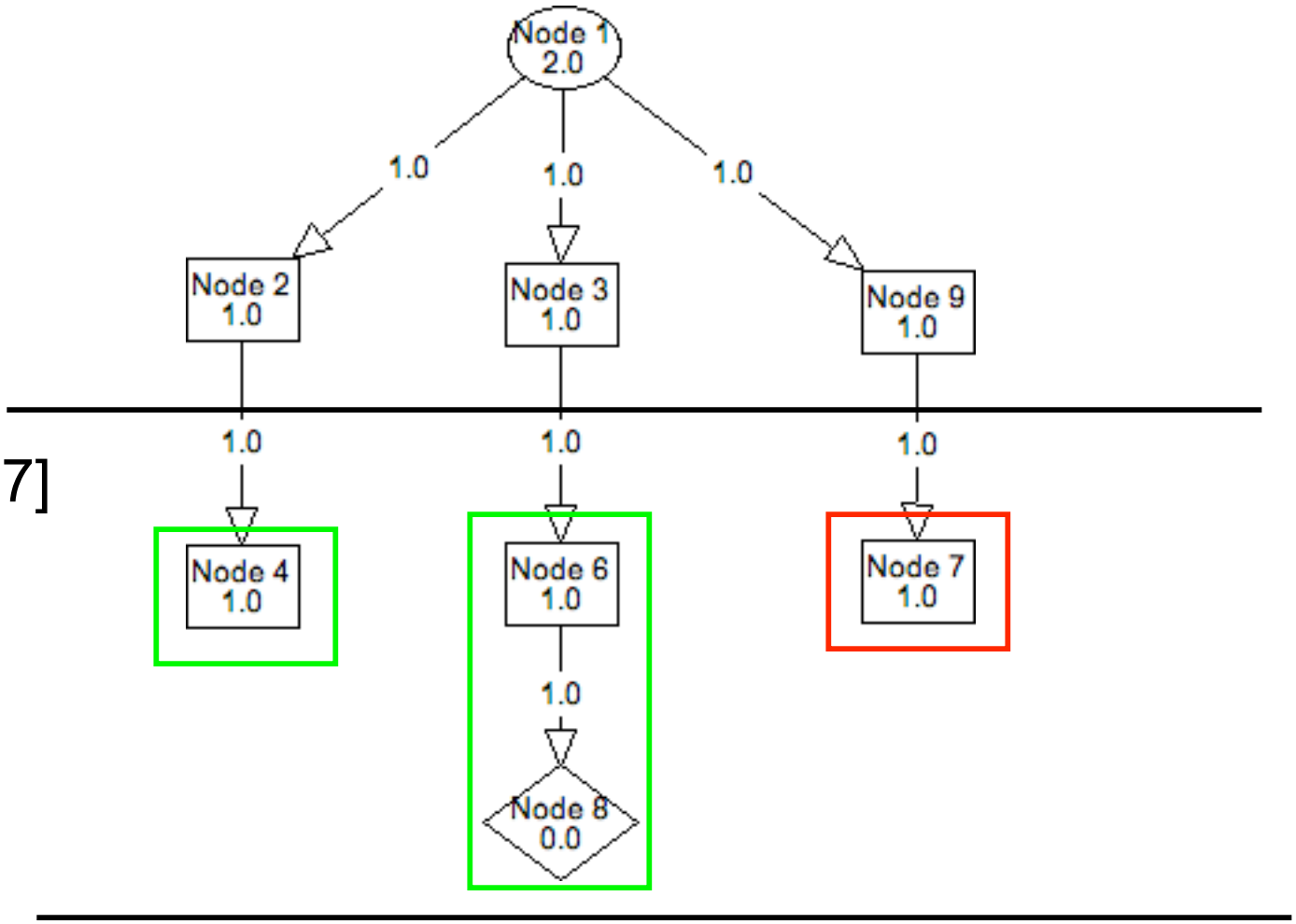
Sometimes expanded

Never expanded

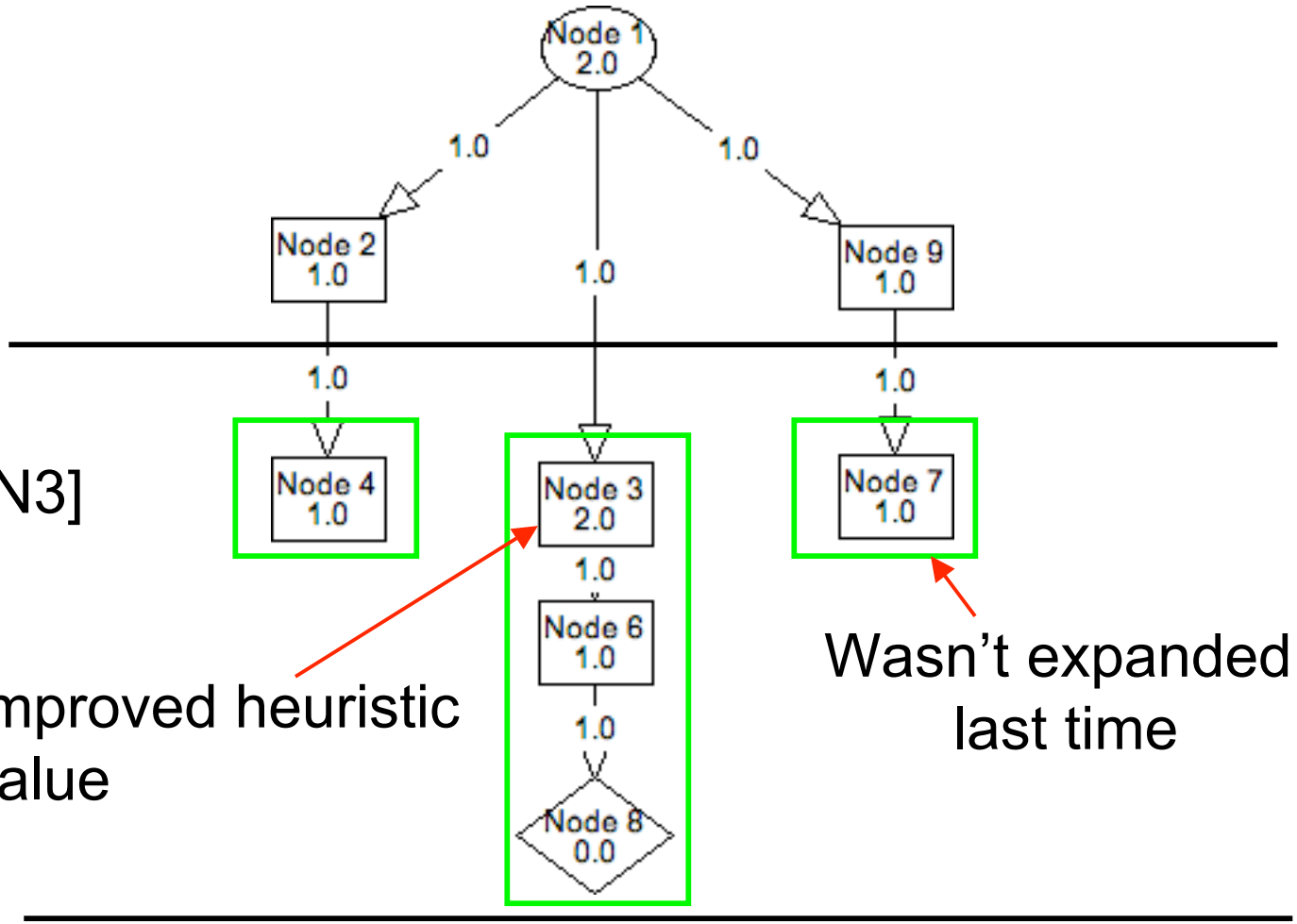


Stack Access Mechanism

Stack:
[N4,N6,N7]
[N6,N7]
[N8,N7]



Stack Access Mechanism



Stack:
[N4,N7,N3]
[N7,N3]
[N3]
[N6]
[N8]

Improved heuristic value

Wasn't expanded last time

Access Mechanism Problem

- Saw example of using a stack access mechanism, where an improved heuristic led to a loss of efficiency.
- We can do the same sort of thing for queue access mechanisms.
- Why do these access mechanisms have this type of problem?

Access Mechanism Problem

cont'd

- The problem is that the order elements are accessed is dependent upon the search tree traversal.
- Improving the accuracy of the heuristic can change order of search tree traversal changing the order that nodes are selected from the critical f-open list.
- This means that critical tie nodes that weren't expanded before the improvement, now get expanded after the improvement.

Traversal Order Independence

- We want to make the access mechanism independent of the traversal order but still dependent on the search tree structure.
- What feature(s) of the search tree structure could we use that are independent of traversal order?

Node Addressing Schemes

- If we can associate a unique address with each search node that is independent of traversal order then we could use that address to order the nodes in the critical f-open list.
- There are at least two such addressing schemes.

Node Addressing Schemes
cont'd

- If the children are always generated in a particular order, then the list of children numbers from the root to the node would be such an addressing scheme.
- Another such scheme is to label each node with the list of operators applied to each ancestor to get from the root to that node.

A#: A* Refinement

- We will *refine* A* by specifying that the open list mechanism uses one of these addressing schemes.
- Here's what A# looks like:

A# Algorithm

Mark *s* open and calculate *f(s)*.

While there are open nodes Do

Select open node *n* with smallest *f* value

resolve ties *first* in favor of goal nodes

and secondly in favor of nodes with lower addresses,

If *goal(n)* then terminate with success

Mark *n* closed

For all successors, *j*, of *n* Do

calc *f(j)*

if *j* not in closed or *f(j)* is lower

mark *j* open.

Separating Orderings

- We can separate the initial access order the Hof tree roots appear in the f-open list, when the Hof forest is first selected, from the order the nodes in the Hof trees are processed.
- For example, after we do the initial ordering of the f-open list, we can process the Hof-trees either via a stack, a queue, etc.

$$A\#^{AddressScheme}_{MainScheme}$$

Put *s* into open and calculate *f(s)*.

While there are Hof forests Do

If there is a goal node in the f-open list

then exist with that goal node.

Select the Hof forest with the smallest f-value

Order the f-open list according to addressing scheme

While there are still nodes in the f-open list Do

Remove the first node, *n*, in the f-open list

Put *n* into closed

For all successors, *j*, of *n* Do

calc *f(j)*

if *j* not in closed or *f(j)* is lower

Put *j* into open according to maintenance scheme.

Where are we now?

- A# gives us *monotonic improvement*.
- However, we still lack *heuristic equivalence* and *optimal efficiency.*
- We can gain these by placing more restrictions on the heuristics used.

New Restriction on *h*

- As a matter of fact we are going to make A* *optimally efficient* and gain *heuristic equivalence* by making it less accurate!!!!!

The Problem with Hof Trees *Revisited*

- The problem with Hof trees is that they can hide goal nodes from A* so that when A* starts to explore a Hof forest, it doesn't know which Hof tree (if any) in that forest contains a goal node.
- However, if we could guarantee that all goal nodes were roots of their Hof-tree then A* could always pick them first.

When Aren't Goal Nodes Roots of Their Hof Trees?

- We will start off by looking at a specific situation where goal nodes cannot be roots of their Hof trees.
- Then we will generalise this.

Goal Discriminating Heuristics

- Most heuristics can distinguish between goal nodes and non-goal nodes.

- In other words, for all non-goal nodes *n*, $h(n) > 0$

“Standard” Problems & Heuristics

- Lets call problems which have search spaces where all edges have a cost of one, “*standard*” problems.
- Lets call heuristics which only give integer valued estimates, “*standard*” heuristics.

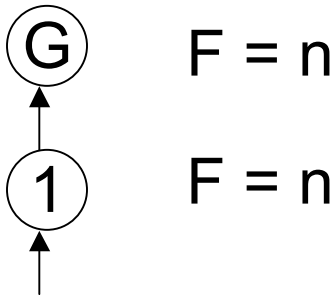
Standard Goal Discriminating Heuristics

- Lets call heuristics which are both standard and goal discriminating, "*standard goal discriminating*" heuristics.
- For any standard goal discriminating heuristic for a standard problem, the *h* value of a node that leads to a goal node must be 1!

Critical Hof Tree

- A Hof tree containing an optimal goal is a *critical* Hof tree.
- Given a standard goal discriminating heuristic for a standard problem, no optimal goal node will ever be the root of any critical Hof tree (except if that optimal goal node is the root of the problem's search tree).

The Problem



The optimal goal is not the root of its own Hof tree.

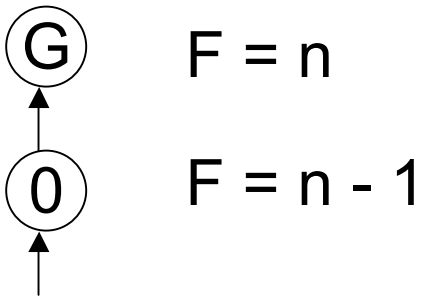
Possible Cures

- To make **G** the root of its own Hof tree, we could either try to make f(**G**) > n or make f(**G**'s predecessor) < n.
- The former is a *bad* idea, why?
- We will now look at the latter approach.

The Cure - Part I

- Given a standard problem, and a standard heuristic, we need the heuristic to be non-goal discriminating if we want the optimal goal to be the root of its own Hof tree.

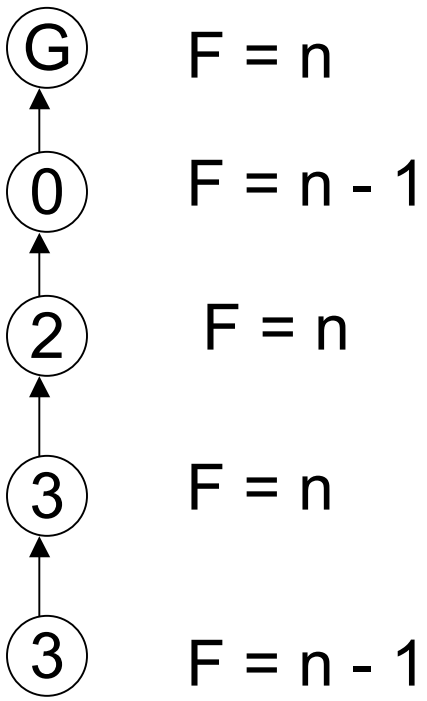
The Cure - Part I cont'd



The optimal goal is now the root of its own Hof tree. However, is this enough?

An example of the problem with our first cure

We still don't see the goal node until we start exploring the n-Hof tree.



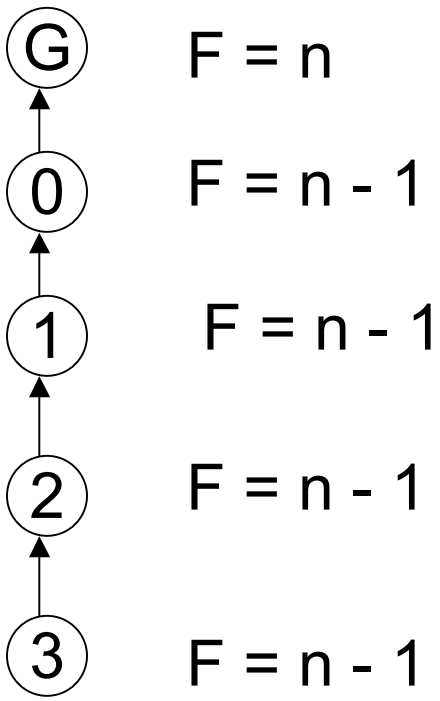
The Generalised Cure - Part II

- We need to keep the heuristic monotonic so that we don't have these disconnected parts of the Hof forest, which hide the optimal goal node.
- We do this be continuing our process of lowering our estimate to the ancestors, *n*, of optimal goal node as long as $h(n) = h^*(n)$ and we haven't yet encountered an ancestor where this wasn't true.

The Generalised Cure - Part II

cont'd

Now, nothing hides the optimal goal node when A* starts exploring the n-Hof forest and it can be chosen first.



A Generalised Solution: Monotonic Non-Goal Discriminating Heuristics

- Start at the goal node and work backwards building a table of nodes and their lowered values. Stop along a path whenever you encounter a node where $h(n) =/= h^*(n)$.
- After the table is built, when *h* is called for a node, first check the table if it's there then return its recorded value else calculate the value as usual.

Where we are now

- It seems obvious (but not yet proven) that A* is optimally efficient when it uses admissible monotonic non-goal-discriminating (AMNGD) heuristics.
- It also seems obvious that we can automatically convert admissible monotonic goal-discriminating (AMGD) heuristics into AMNGD heuristics.

Where we are now cont'd

- However, that does not mean that when we convert an AMGD heuristic into its corresponding AMNGD heuristic that A* will never expand more nodes using the latter than when using the former.

Where we are now cont'd

- There are a number of loose ends to tie up:
  - How big are these tables?
  - How much does it cost to use these tables?
  - How much does it cost to build these tables?
  - How do the search spaces compare for AMGD and their AMNGD counterparts?
- We have some tentative answers.

How big are these tables?

- Probably not very big
- Hof trees tend not be very deep.

How much does it cost to use these tables?

- Probably a hash function will be used to access the table, so the additional overhead will probably not be very much.

How much does it cost to build these tables?

- The cost is probably linear with respect to the size of the table (exponential with respect to the length of goal ancestral critical tie (GACT) chains).

How do the search spaces compare for AMGD and their AMNGD counterparts?

- The worst case is that the latter will expand

|longest GACT chain| - |shortest GACT chain|
more nodes than the former.

- Which is likely to be very small.
- On average, the latter will probably do much better than the former.

An Aside

- We can actually make a better version of A* if we know that we are only using AMNGD heuristics, which will be much faster on average than the standard A*.
- However, we then probably lose our claim of optimal efficiency.