Game Playing

Good Domain for Exploring AI

Structured Task
Easy to measure Success or Failure

Originally, The following were thought to be true

Large amounts of Knowledge not required?

Straight Forward Search from starting position to winning position?

Counter Example:
Chess
Branching Factor: around 35
Number of moves: on the order of 50
This would mean the examination of 35^{100} positions

Generate And Test

Generate the Case and **Test** its Effectiveness

Extremes

Generate Everything
(For example, all possibilities from a node)
Then test for Effectiveness

Generate Individual Moves
and
Test each time

Efficient Generator
Generate only good moves
Replace *legal* move generator
with
plausible move generator
(i.e. Heuristics Built into the Generator also)

Efficient Tester
Best Moves will be explored first

Basic Minimax Procedure

Given a Current Move Situation:

See whether it is a win. If so, Call it the best

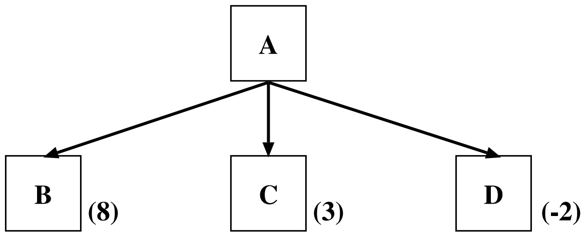
Consider all moves that the opponent can make

Look for the worst for us (best for him) and assume he will make this move

Pass this value to the node we are evaluating

Choose the node with the best rating

One Ply Evaluation



One-Ply Search

One-Ply

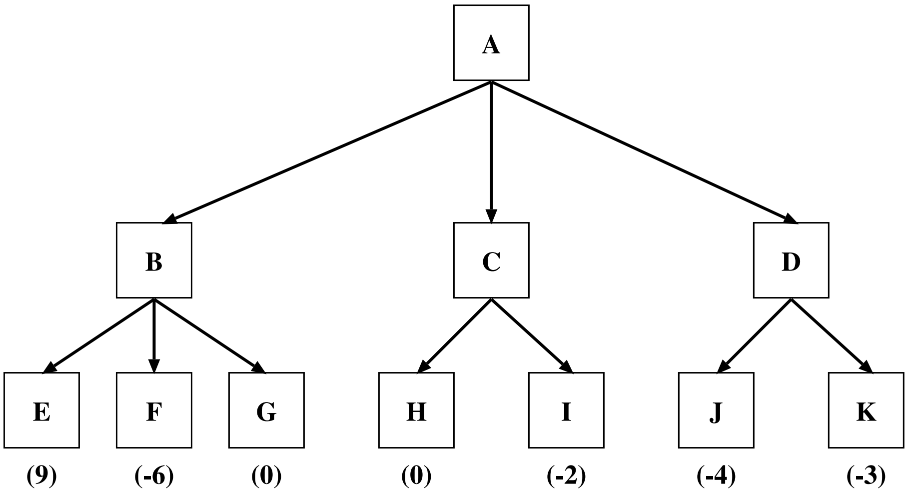
Start from the current position
and
Generate possible moves

One-Ply Search
From the current position, generate possible moves
Assuming a heuristic function
10 good for us
-10 good for them

Try to maximize the heuristic for the given set of moves
Based solely on the next move

Based on the one level, Choice is B

Two Ply Evaluation



Two-Ply Search

Two-Ply Search

But the heuristic will not be perfect
Generate the next set of plausible moves

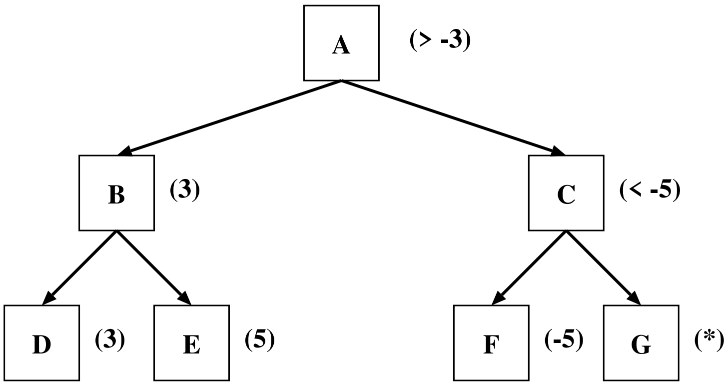
Looking two moves ahead
Our Move and then Their Move
The Heuristic function is then applied to these new moves

In this case, we are trying to
Minimize the goodness of the position
Because the opponent will choose
the move which is worst for us.

Given our move,
the worse move (with respect to us)
will be chosen as the following move

Choosing B as the first move,
would result in a bad move after the second move (-6)
(though it had the possibility of being good (9))

Two Ply Evaluation

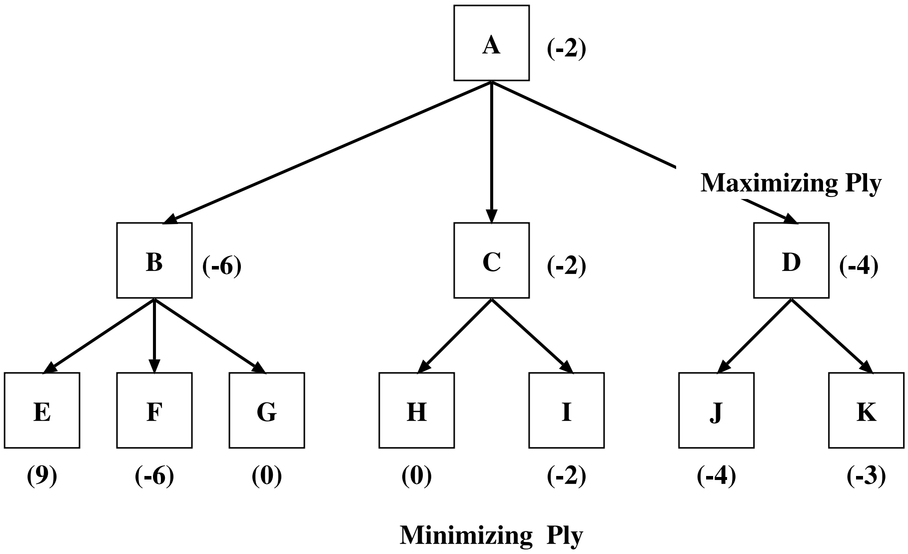


Backing Up Choice

One the second ply,
we are minimizing the score of possibilities

And then on the first ply,
we are maximizing the score of possibilities

Backing Up Values



Two Basic Functions

MOVEGEN(Pos)
The plausible move generator
Returns list of nodes that can be made from position, *Pos*

STATIC(Pos,Depth)
The Static Evaluation Function
goodness of Pos, for the player's point of view

High values are good moves for the one about to move

Termination of Search

Has one side Won?

How many ply have we already explored?

How promising is this path?

How much time is left?

MiniMax

MiniMax(current,depth)

current
The current position to be evaluated

depth
The current depth of the postion

Return

Value:
The heuristic value of the move

Path:
The path of moves

Deep-Enough

Has the maximum depth been reached?

Yes
Return with:
Value:
Static(current,depth)

Path:
No path

Generate Next Level

MoveGen(current)
Generates a list of plausible moves from the current position

Successor
The list of moves that were generated

If Successor is empty

Then
There are no more moves so return with:
Value:
The heuristic value of the move
Path:
The path of moves

Begin to Examine Successor Elements

Examine the Successors, keeping track of the best one:

Initialize

Best-Score

Minimum value of the *Static()* can return

For Each of the Successor Elements

Recursively Call
MiniMax(Successor-Element,Depth+1)

Set result into *Successor-Result*
New-Value
Negative of the *Successor-Result*
This reflects the position from the opposite perspective

If
New-Value > *Best-Score*
Then
The best successor has been found thus far

Best-Score to *New-Value*
Best-Path to Successor Path

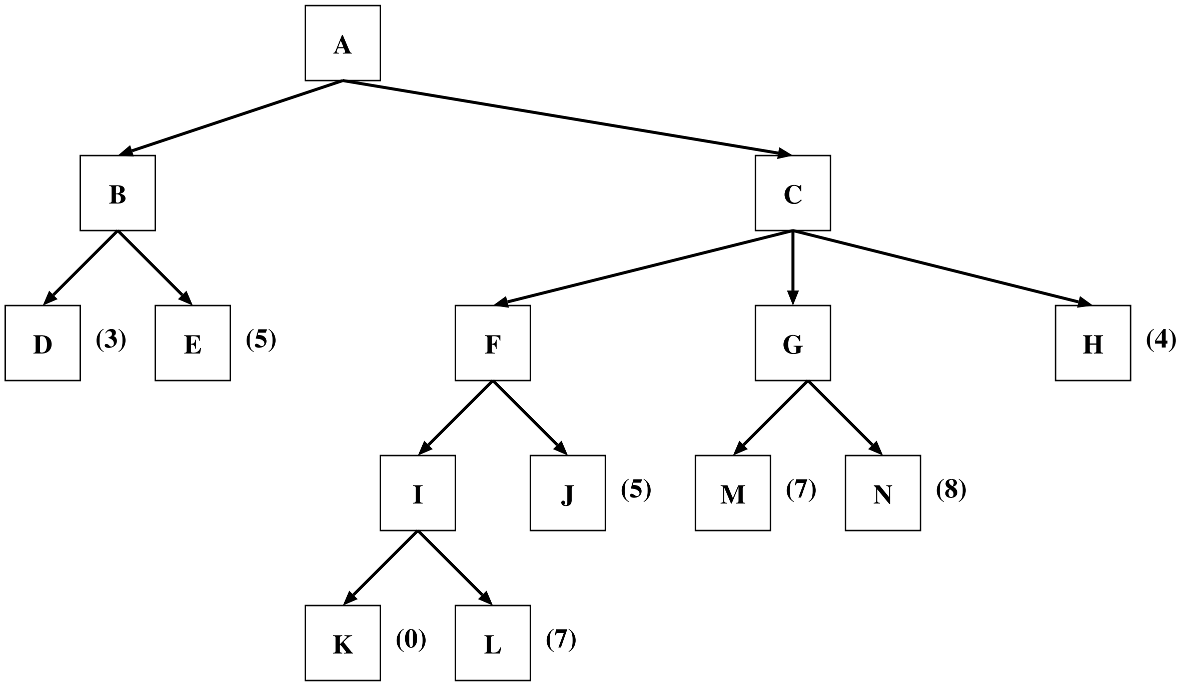
Alpha-Beta Pruning

An Improvement of the Heuristics
Through the introduction of
Bounds

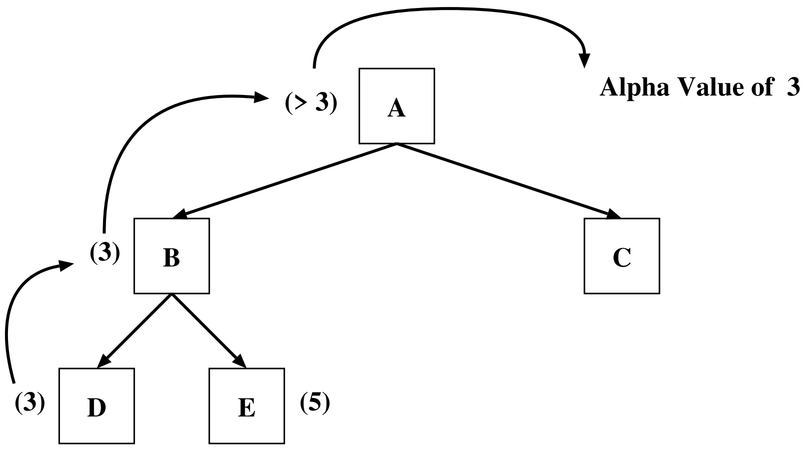
Alpha
Lower Bound
The lower bound that a *Maximizing* node may take
(i.e. the lower bound for our own good move)

Beta
The Upper bound that a *Minimizing* node may take
(i.e. the upper bound that the opponents move)

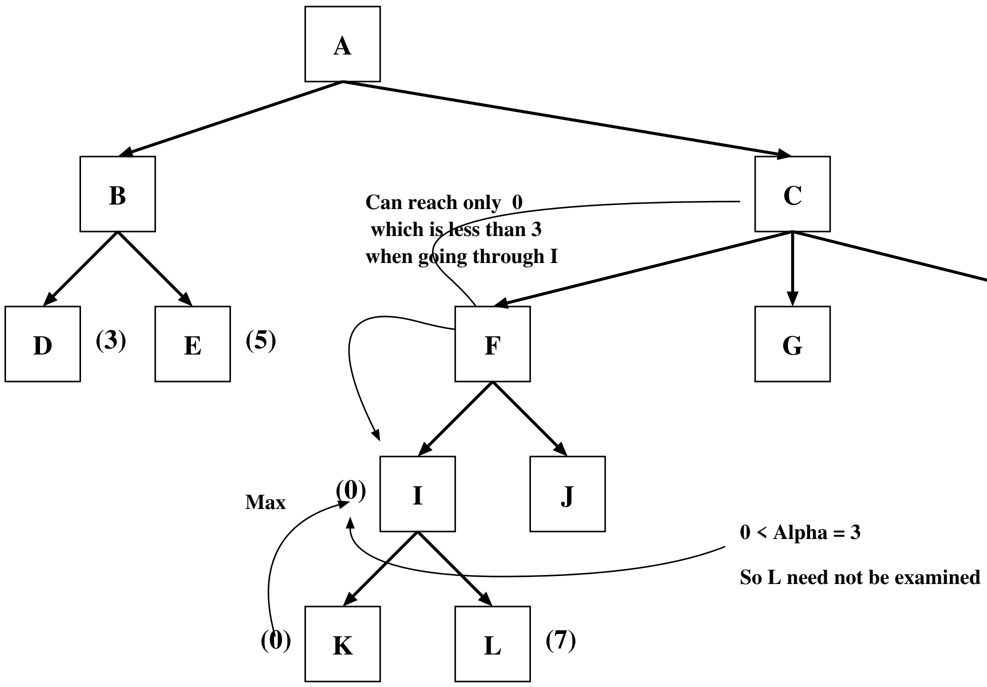
Alpha-Beta Search



Alpha Value is now three



Use of Alpha Value



Node I

The entire subtree under B is searched

From this, we expect a score of at least 3

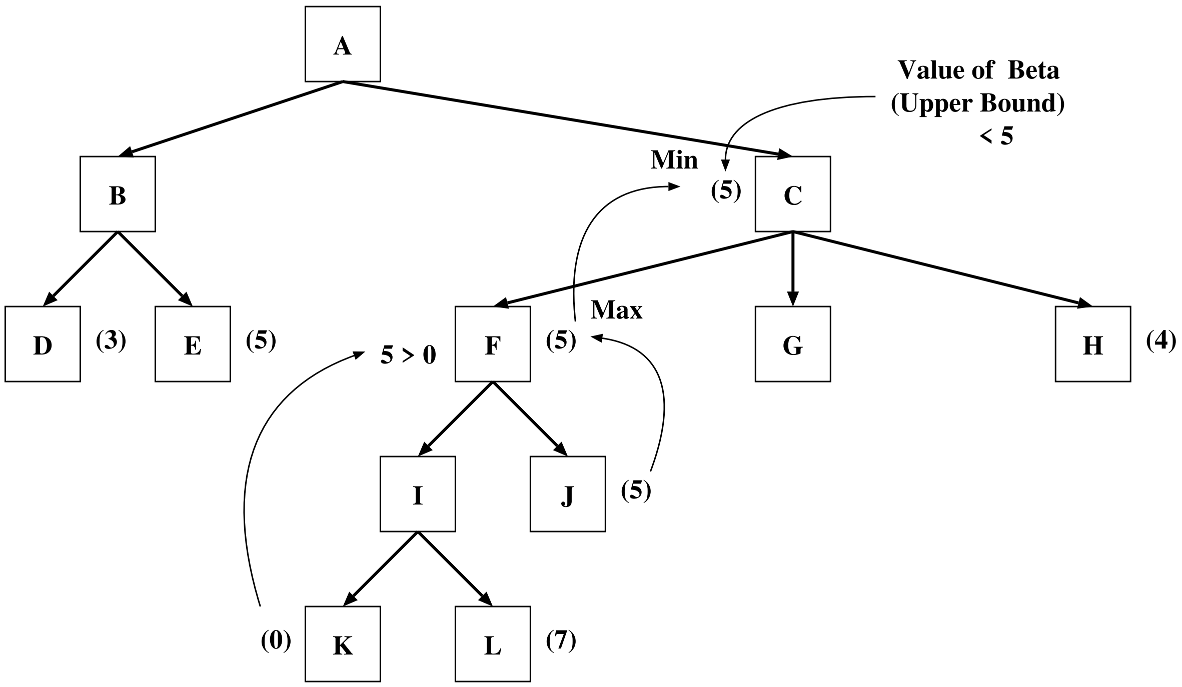
When passed to subtree F
We can skip the exploration of L

Why?

After K is examined,
I is guarenteed a *maximum* of 0
F is guarenteed a *minimum* of 0
But this is less than alpha's value of 3
so
No more branches of I need be considered

The maximizing player already knows not to choose to move to C and then to I, because if that is done, a score no better than 0 can be reached and this is less than 3 (of node B)

Setting of Beta



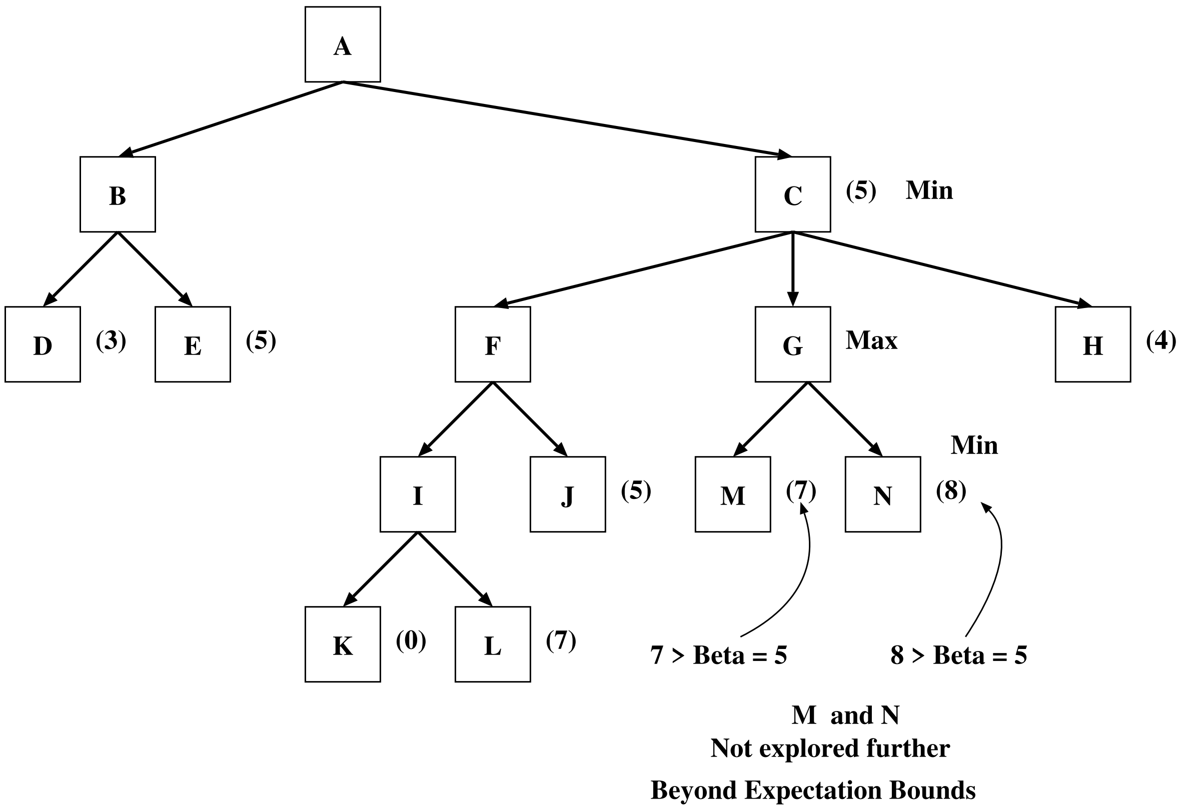
Node J

After cutting off exploration of I,
J is examined
yielding 5

This gives F a value of 5
The maximum of 0 (of node I) and 5 (of node J)

This becomes the beta of node C
This means C is guarenteed to be 5 or less

Beta Inhibits Further Expansion



MiniMax

MiniMax
(current,depth,Use-Threshhold, Pass-Threshold)

current
The current position to be evaluated

depth
The current depth of the postion

Return

Value:
The heuristic value of the move

Path:
The path of moves

Deep-Enough

Has the maximum depth been reached?

Yes
Return with:
Value:
Static(current,depth)

Path:
No path

(Same as MiniMax)

Generate Next Level

MoveGen(current)
Generates a list of plausible moves from the current position

Successor
The list of moves that were generated

If Successor is empty

Then
There are no more moves so return with:
Value:
The heuristic value of the move
Path:
The path of moves
(Same as MiniMax)

For Each of the Successor Elements

Recursively Call
MiniMaxAlphaBeta(Successor-Element,Depth+1,Use-Threshold,Pass-Threshold)

Set result into *Successor-Result*
New-Value
Negative of the *Successor-Result*
This reflects the position from the opposite perspective

If

New-Value $>$ *Pass-Threshold*

Then

The best successor has been found (up until now)

Pass-Threshold to *New-Value*

Best-Path to Successor Path

Use-Threshold

If
Pass-Threshold < Use-Threshold
Then Stop Examining This Branch

If
Pass-Threshold >= Use-Threshold
Then Exit Immediately with: Value: Pass-Thresh
Path: Best-Path

SSS: Best First Search for Optimal Playing Strategy

SSS^*
Searches similar to AO and AO^*
Shown superior to alpha-beta

Search of an AND/OR graph

The opponent is hostile and perfectly-informed

The strategy is specifying one's response to the opponent's move

He is at liberty to choose any of the available moves

Tree Cluster

The value of any strategy is the minimum value of all the terminal nodes

The value of the frontier node is the upper bound of the scores possible from the subtree it represents

The possible moves of the opponent are joined together with an AND
He would choose the worse move from this collection
Thus the minimum value
The node with the lowest upper bound

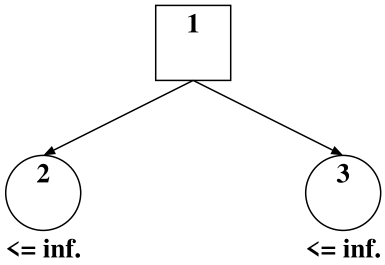
The AND node
To find a solution
We have to take into account all the opponents moves

The merit of a partially developed tree is determined solely by the properties of the frontier nodes it contains and not the cost of the paths

The merit of the frontier nodes are the
upper bounds of the sub-tree it represents

A solution tree represents the optimal strategy
It's value coincides with the minimax value of the root

Expand Node Root



From the Root (1)
Generate Two Successors (1,2)

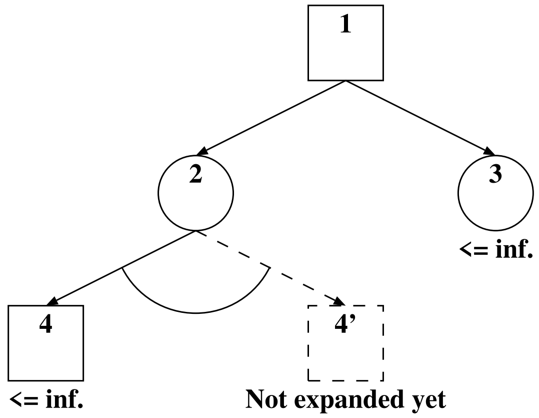
Each Successor represents a cluster of solution trees

We have no information about the merit of these
So bound above by infinity

We now have two ties
By default choose the leftmost

Expand node 2

Expand Node 2



Expand Node 2

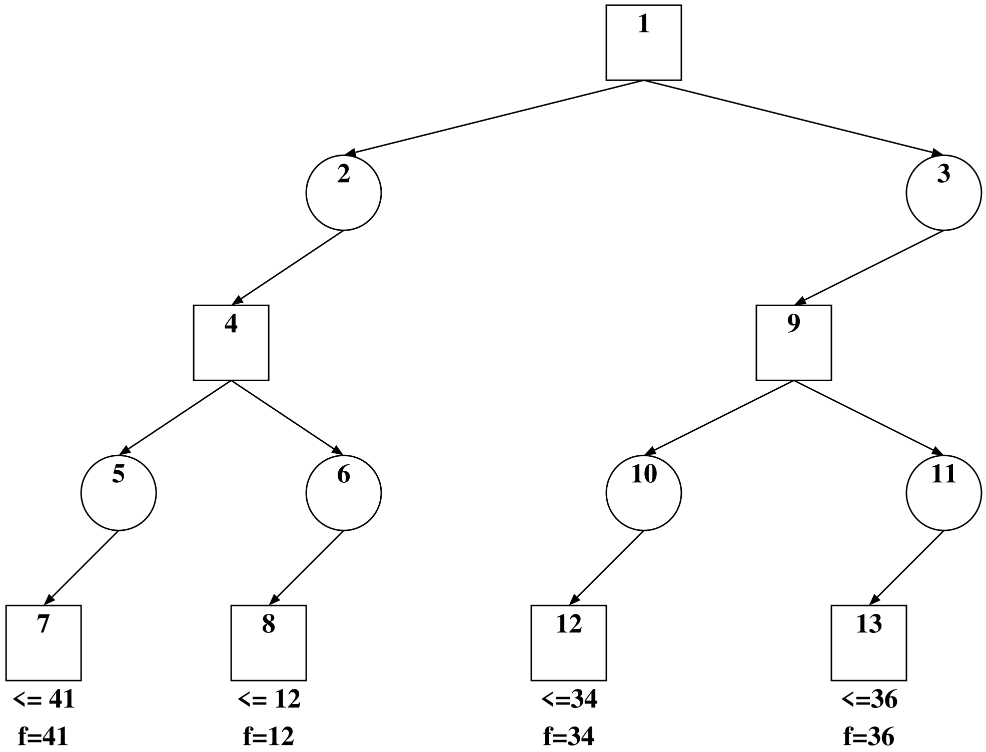
Node 4 is generated

Node 4' is not generated
We want to examine the tips of the set of clusters of trees first

Eventually, we want to compare the merits of the tree clusters
This can only be done when one tip node in each cluster is expanded

At this point, expand node until each cluster has a tip node

Expand To Node Tips



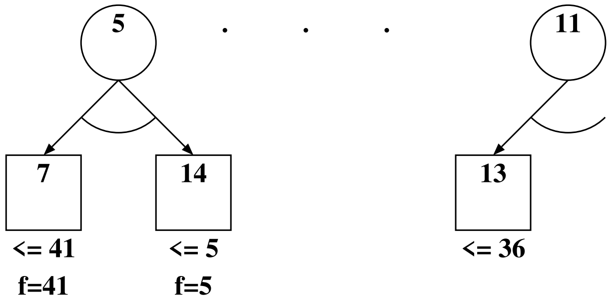
Expand to Node Tips

All clusters expanded until at least one tip

The most meritous node is that which has
the possibility of
the highest bound

This is node 5 with the upper bound of 41
According to our statistic

Expand Node 5



Expand Node 5

Expanding node 5 consisting of further development of the cluster

The expanded node turns out to have an upper bound of 5

This renders the cluster with node 13
having an upper bound of 36
As the best candidate for expansion

Expand Node 11

Node 11 is expanded
and generates node 15

At this point Node 11 still is the best candidate for expansion

But it is complete

Thus
Node 9 can be considered “solved”
with the best upper bound of
≤ 35

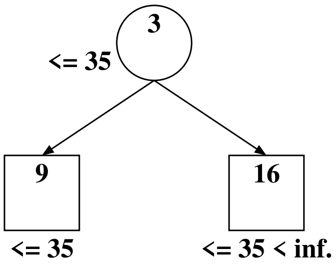
Node 9 is “solved” because no further expansion can yield a better result

Why?

Can only expand node 10
if a better it yields a better value than 34
the value of node 10 is still ≤ 34 because of AND node 12

The upper bound of node 3 is now 35
(tracing back values)

To Expand Node 16



To Expand Node 16

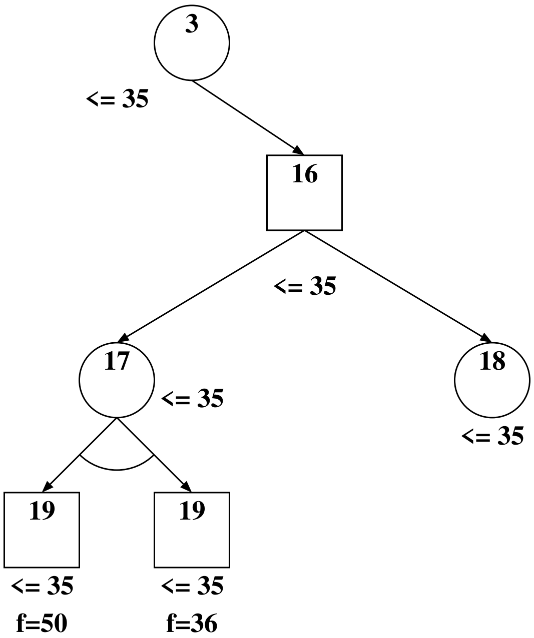
Node 3 has the best upper bound of ≤ 35

Expand to Node 16

Normally
Since it is not a frontier node
It has a value of Infinity
But infinity is less than 35, so it takes on the value of node 3

Expand to frontier node

Expand Node 17



Expand Node 17

Node 19 had a value of 50

But this is greater than the upper bound of 17 so node 19 has ≤ 35

Afterwards, both nodes 17 and 18 have upper bound ≤ 35

Node 17 is expanded first because under ties, the leftmost is expanded first

Expanded node 19 has a value of 36
But this is greater that the upper bound
so the upper bound of node 17 stays the same

But now node 9 is solved because the expansion of node 18 cannot yield any better upper bound than 35

Node 16 is now solved with a bound of ≤ 35

Final Solution