Heuristic Search

© Diarmuid O'Donoghue, 2005-2017.
Dept. Computer Science, Maynooth University, Co. Kildare, Ireland.

Search - Overview

Different Search strategies are used in many areas of AI

1. Blind Search
2. Heuristic Search
   - A*
3. Adversary Search (Game playing)
   - MiniMax
   - α–β Pruning
- Also Graph Search

Why **Search**?

- **Solve** problems
  - Oxo, Farmer-Fox-Goose, combinatoric problems
  - Chess, Checkers & competitive games
  - Shortest path to exit a maze
  - Find the best path for an NPC to attack a player character in a Computer Games…
- Create **predictive** models of
  - war, economics & other domains

Problem Solving

- Many AI problems can be solved by **Searching**
- Different search techniques are employed, each with their own characteristics
- **Generate and Test** **(G-n-T)**
  - **Problem Spaces** are **Generated** representing (all/many) possible solutions
  - **Test** different spaces to identify successful solutions

Sliding Tile Puzzles

- Such as 8 and 15 tile puzzles
- **Goal** state and **Start** State :

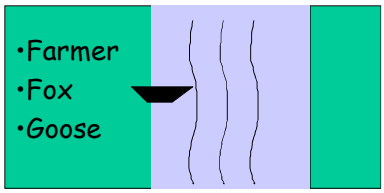
1	2	3
4	5	6
7	8	

8	5	3
1	2	7
6	8	

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

http://www.jaapsch.net/puzzles/fifteen.htm
http://www.tilepuzzles.com/default.asp?p=13

Farmer, Fox and Goose (1)



- Get all three across the river safely
- Constraints
  - The boat can only fit two "people" at a time
  - Fox and Goose can never be alone together on either bank, or in the boat.

Blind Search

- For small problem spaces we can generate every possible solution, using:
  - **DFS** - Depth First Search order
  - **BFS** - Breadth First Search order
- However, blind search strategies can only solve toy problems
  - Chess has 10^{140} different possible board states and all positions could never be generated
    - >> number of nanosecond since the big bang!

7

Backtracking

- **Backtracking** occurs when the part of the search space is found to be unproductive
- By a roll-back to a previous recursive position, we continue searching alternate branches
  - backtracking occurs very frequently during problem solving
  - Prolog executes programs as a search and backtrack process

8

Lookahead

- Determines the "depth" of the search space to be explored
- Its really just a simple integer parameter that's decremented every time the recursive function calls itself
  - Limited by a zero lower bound
  - Depth Limited Search

9

DFS - Depth First Search

- Recursive blind-search mechanism
- **Pre-Order** search examines the root before visiting either sub-tree
    - yielding a program employing **tail-recursion**, with a very low memory overhead
    - + 2 3
  - **Post-order** examines the root after subtrees
    - 2 3 +
  - **In-order** visits one subtree, then the root and then the other subtree
    - 1 + 2

10

BFS - Breadth First Search

- Recursive blind-search mechanism
- **Post-Order** search; stores the root node while it examines both sub-trees
    - yielding a program employing **head-recursion**, with a very high memory overhead
      - storing the entire search tree
  - Post-order examines the root after subtrees
  - In-order visits one subtree, then the root and then the other subtree
  - Sigma(n) = Sigma(n-1) + n
  - Sigma(n) = n + Sigma(n-1)

11

Iterative Deepening (ID)

- Breadth FS searches to a predetermined depth.
- ID tries all depths, 1, then 2 then 3 etc.

12

The 8/15 Tile Puzzles

- In the case of the 8-tile puzzle:
- **Goal State** might be:


1	2	3
4	5	6
7	8	

- but **Start State** might be

8	5	3
1	2	7
6	8	

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

http://www.jaapsch.net/puzzles/fifteen.htm

13

Search Spaces

- Consider the following search space(s)

2	8	3
1	6	4
7		5

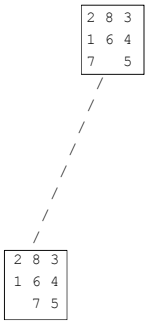
Start state

Generate each 3 alternate next state from the initial state.
-generate(Board) -> board

14

Search Spaces

- Consider the following search space(s)



Start state

Generate each 3 alternate next state from the initial state.
-generate(Board) -> board

15

Search Spaces

- Consider the following search space(s)

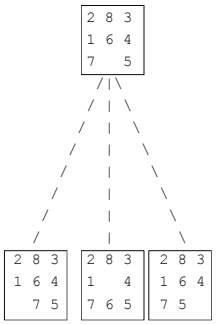
Start state

Generate each 3 alternate next state from the initial state.
-generate(Board) -> board

16

Search Spaces

- Consider the following search space(s)



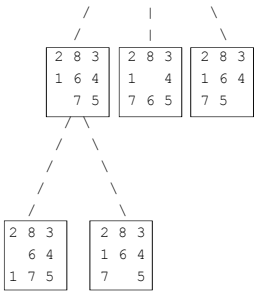
Start state

Generate each 3 alternate next state from the initial state.
-generate(Board) -> board

17

Search Spaces ctd..

- expanding one branch further, we get



2 subsequent states for one of those states

18

Complexity of Search

- **Branching Factor** determines the extent (size) of the search space - B
- Path length - L
- Total Search space - T
- $T=B(B^L - 1)/(B - 1)$
  - $T=B + B^2 + B^3 + + B^L$

19

Farmer, Fox, Goose & Beans (2)

- Once upon a time a **farmer** went to market and purchased a **fox**, a **goose**, and a **bag of beans**. On his way home, the farmer came to the bank of a river and hired a boat. But in crossing the river by boat, the farmer could carry only himself and a single one of his purchases - the fox, the goose, or the bag of the beans.
- If left alone, the fox would eat the goose, and the goose would eat the beans.
- The farmer's challenge was to carry himself and his purchases to the far bank of the river, leaving each purchase intact. How did he do it?
- **Solution:** Bring goose over, Return, Bring fox or beans over, Bring goose back, Bring beans or fox over, Return, Bring goose over
- Thus there are seven crossings, four forward and three back.

20

Heuristic Function - eg

- Heuristics are **Rules-of-Thumb** which help identify potentially fruitful portions of the problem space
  1. *"capturing pieces helps win at chess"*
  2. *"advancing up the board is a good idea"*
  3. *"loosing a piece damages your chances"*
  4. *"try to occupy the centre of the board"*
- **Static Evaluation Function** yields a numeric score for each solution space
  - *problem space* maps onto a *number*

21

Heuristic Search

- The heuristic function can be used to guide the search process
  - focus on the states with a high heuristic value
  - eliminate some search states with very low heuristic values
    - and their sub-states, their sub-sub-states *etc.*

22

Greedy Best First Search

- Heuristic search mechanism
- Like depth-first search, but guided by the heuristic value

1. Generate all *next* states (look ahead one "move")
   - We might be able to take a short-cut here
2. Select the (1) best of these states – adopting this as our current state
   - Using our heuristic function
3. Go to 1.

23

Best First Search

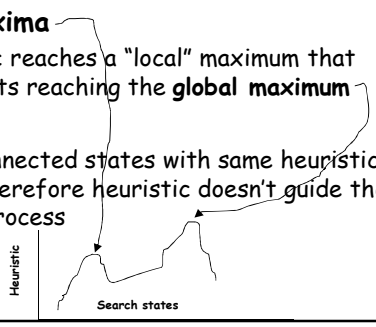
- Use the heuristic to identify (beforehand) the best squares on the board
- Use this ordering to "guide" Best First Search
  - 5, 1, 3, 7, 9, 2 ,4 ,6 8,

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

24

Local Maxima & Plateau

- 2 Main problems that plague Heuristic search
- **Local Maxima**
  - Heuristic reaches a "local" maximum that inhibits its reaching the **global maximum**
- **Plateau**
  - Many connected states with same heuristic value. Therefore heuristic doesn't guide the search process



25

A* Search

Best First Search Algorithm

See also http://en.wikipedia.org/wiki/A*_search_algorithm

26

"Repeated States" Anomaly

- Of course, there are many ways to reach many game states.
- In chess, different placement order produce the same state
  - but by different paths
- We don't want to duplicate path information
- Path A says its 5 move but path B says its 7 moves
  - only store the "shortest" path

27

DAG

- Trees – DAG – Graph
- DAG - Directed Acyclic Graph

- A* ensure the search stays as a tree
  - Never turns into a DAG

28

A* Fundamentals

- **Generalised Best-First Search**
- Extends one node per **iteration**
- **f'** approximation of current node's worth
- **g** cost of getting to current node (actual)
- **h'** additional cost of getting to solution

f' = g + h'

- **Closed:** (*internal*) set of nodes whose successors have all been examined *(internal nodes?)*
- **Open**: (*leaf*) set of nodes whose successors have not yet been examined *(leaf nodes?)*

29

A* = Best F. S.

- **f'** approximation of nodes worth
- **g** cost of getting to current node (actual)
- **h'** additional cost of getting to solution
  - BFS => h'=0
- **f' = g + 0**

30

A* Intro

1. Pick **Bestnode** with lowest f'
2. Compute Successor (Bestnode)
   - g(Bestnode) + new-cost
3. Set Successor to point back to Bestnode, so we can recover moves later
4. If Successor ε Open, then link it (*Successor(Bestnode)*) to the cheaper of Bestnode-path or Oldnode-path, by comparing g values

Note: Each node points to the best parent node

31

A*

5. If Successor ε Closed, call it Old

If newpath to old is less than old-path to Old then set g and f' values of Old appropriately

BUT must propagate f' to Successors

DFS Successors(Old) altering g until find a node with equivalent or better g value

But: If in propagating g down, the path may become better than nodes current parent,

then Stop else reset parent and continue propagating

32

A*

6. If Successor not in Open or Closed, put in Open and add to Bestnodes successors

f'(Successor) = g(Successor) + h'(Successor)

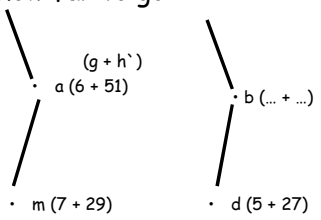
33

A* Example

g = actual cost of getting to state

measured in number of moves?

h' = estimate of how far to go



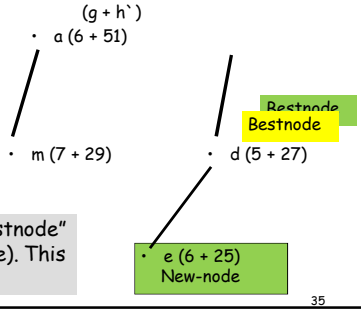
•4 states depicted a, b, m, d g and h` values are illustrated

•a and b are in closed m and d are leaf nodes – in open

34

Situation 1 A*

The Simplest case is when we develop a totally new successor of d(Bestnode) e.

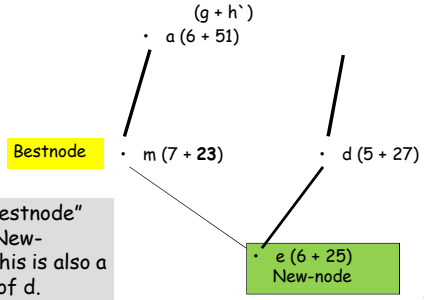


Select and Develop "Bestnode" d to create New-Node(e). This is a successor of d.

35

Situation 2 A*

M is Bestnode, but its successor e already exists and has a better path ...

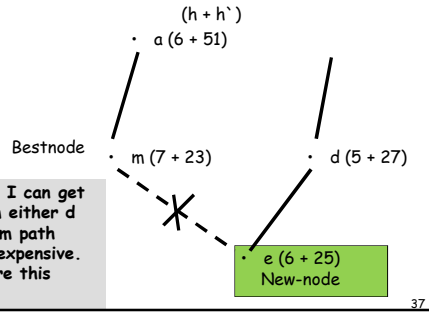


Develop "Bestnode" to create New-Node(e). This is also a successor of d.

36

Situation 2 A*

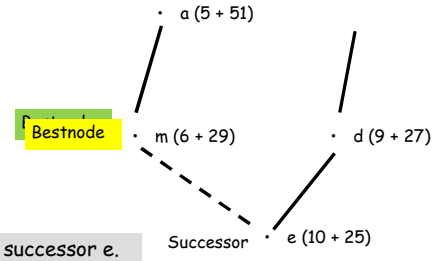
.. Then we don't maintain the newer but less-efficient path.



In this example I can get to state e from either d or m. The new m path would be more expensive. So, I *don't* store this possibility!

Situation 3 A*

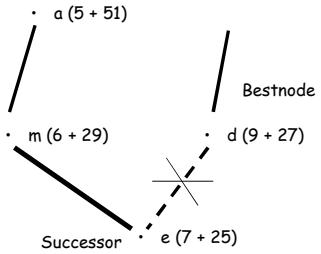
M is Bestnode. Its successor e already exists, but we now have a better path to e



D generated its successor e. But e is also a successor of m. We only keep one path to e

Situation 3 A*

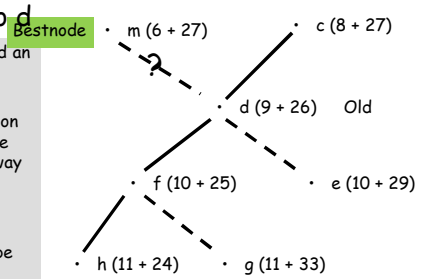
... so we must update that successor. So we must replace the d-b branch with m-e.



We keep the new "short cut"!

Situation 4 A*

M is Bestnode, its successor d already exists and had "children" and we found a shortcut to d



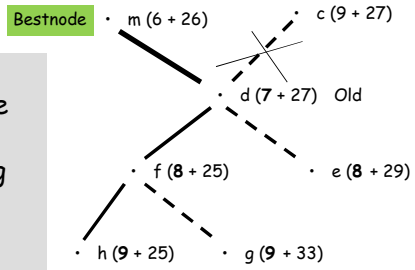
We have generated an existing node – d.

d has many follow-on states, but have we found a cheap**ER** way of getting to d?

If we're luck, the existing path will be cheapest…

Situation 4 A*

Re-parent d (from c to m) and propagate the changes to the successors of d (f,e,h,g...)



Updating a Closed node might mean propagating **new** values down the tree.

- Other heuristically founded search techniques

Tabu Search

- As stated, Hill-Climbing (best F. S.) often works dreadfully
  - Getting stuck in local maxima at the very beginning
- Tabu Search overcomes this problem for almost all domains (sub-optimal)
- Maintain a list of "recent" nodes & Never revisit a recent node
  - even if you have to reduce the heuristic value to avoid this.

43

BLAST Algorithm

- Blast is a **heuristic** based search used by DNA researchers
- They often look for a "target" of 1000 nucleotides in a GenBank database of billions of nucleotides
  - …AGTTAC… target query
  - | | | |
  - …ACTTAG… GenBank database
- 50 times faster than optimal, but less accurate

1. Look for a partial "**seed**" match
2. **Extend** seed match in both directions

44

Blast Heuristic

- Basic Local Alignment Search Tool
- Only **high-scoring** matches are passed to 3rd & final stage
  - heuristic requires = 11 nucleic seeds

3. Blast then looks for **gaps** in alignment between squences
   - …AGTTAC… target
   - ||| \\
   - …AGT…TAG… GenBank database

Blast paper is highly cited. See Document Ranking notes nearer end of course.

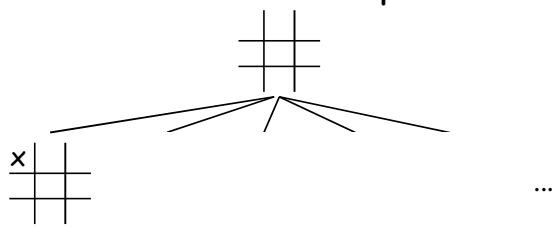
45

Adversary Search
Game Playing Search

Search in the presence of an opponent

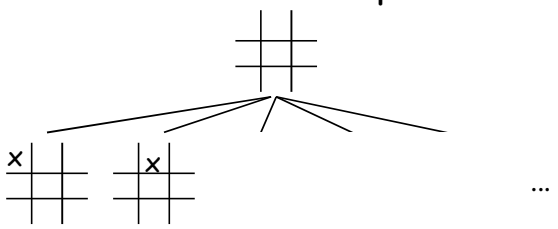
46

OXO Search Space



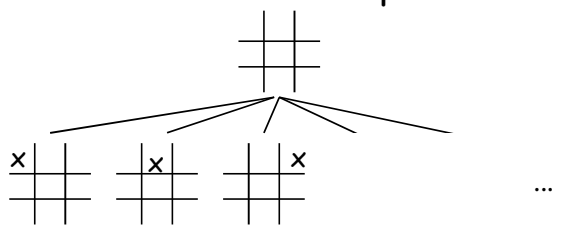
47

OXO Search Space



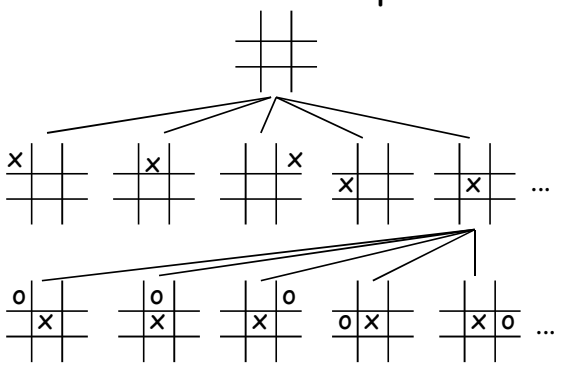
48

OXO Search Space



49

OXO Search Space



50

Adversary Search

- Many **games** involve competition between opponents
- When generating the problem space we must assume the opponent will counteract our moves
  - and not willingly try to loose
- Our search procedure must know whose move it is in order to know how to apply the heuristic rule

51

Game Theory

- **Zero-sum** states that one player's gain is the opponent's loss
  - these are the simplest games, covers the games we will look at *eg chess, tic-tac-toe*
- **Zero-sum payoff rule**
- As opposed to Mutual loss and Mutual gain (cooperative) games
  - Consensus decision making (eg contract law)

52

Game Theory 2

- **Equilibrium** is when both player are unlikely to change their strategies
- **Equilibrium** can be achieved when no change in strategy can improve the players position
  - regardless of what the other player does
- Game theory is also used in the study of (competivive) Economic (Nash, 1/3 Nobel in Economics '94), Political and Warfare systems
  - See "Nash Equilibrium".

53

Game Theory 2

- **Minimax Theorem**
  - Fundamental theorem of Game Theory
- **Every Finite, Zero-Sum, two-person Game has optimal Mixed Strategies.**
  - Mixed Strategy: A collection of moves together with a corresponding set of weights which are followed probabilistically in the playing of a game.

54

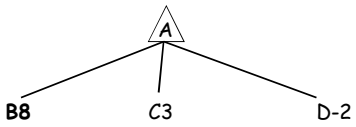
MiniMax

- A depth-first depth-limited search which examines all states
- **2 phase** algorithm
  - descend to the search ply limit and apply the heuristic function
  - propagate these values up the tree
- My move: I **maximises** the heuristic value
- Opponents move: Opponents **minimises** the heuristic value function(h)
  - Alternate between max. and min. levels

http://en.wikipedia.org/wiki/File:Plminmax.gif

55

Minimax Eg

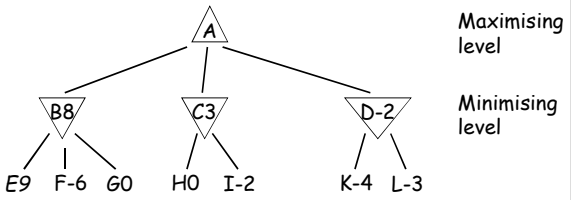


•Generate and evaluate future game positions

• We *would* select B as it gives us the best (maximum) heuristic score, but…

56

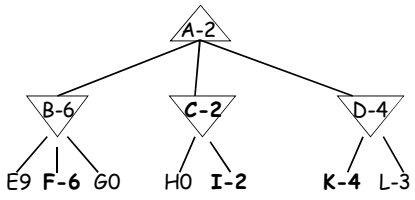
Minimax Eg



- Two-ply moves generated
- At the limit of search space, Leaf values get "backed up" to previous levels
- Which **move** should we select?

57

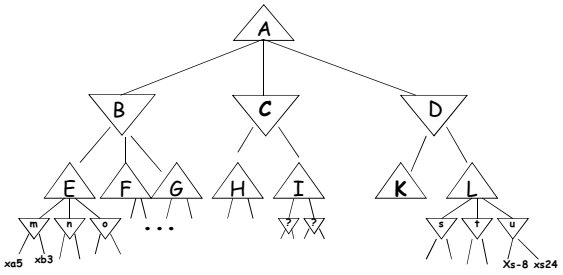
Minimax Eg



- Selecting B lets the opponent "clobber" us - will end up in F(-6)
- So, select C as even a clever opponent can't hurt us more than I(-2)

58

4-ply search



- What a 4-ply search might look like

59

Minimax Problems

- Exhaustive
- Explores unnecessary search spaces
  - slow, inefficient etc.
- A more clever algorithm could do better
  - see α–β pruning which can double the explored space

60

α–β Pruning

- Remove **(prune)** *irrelevant* states from MiniMax search tree
  - depth first expansion of Minimax tree
- Two threshold values (global variables) required, one for each player
- α **alpha** - represents the **lower** bound of the **maximising** level
  - *alpha at least*
- β **beta** - represents the **upper** bound of the **minimising** level
  - *beta below*

61

α–β Pruning

- α–β pruning allows MiniMax to search much "deeper" within the same amount of time
  - removes many irrelevant states
- So, we can now look 6 moves ahead instead of just 4
  - Or maybe even 8 moves ahead!!

62

α–β Pruning

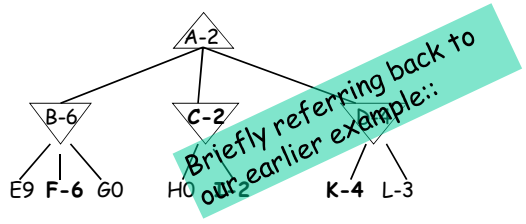
- Terminate a search
  1 below a MAX node with α >= β of *any* of its MIN ancestors
  2 below a MIN node with β <= α of *any* of its MAX ancestors
- Depth-first search means we only ever consider nodes along one path at a time
- Pruning for the Maximising player, means cutoff at the minimising level
  - and vice versa

63

- Check out the Wikipedia entry
  - http://en.wikipedia.org/wiki/Alpha-beta_pruning
  - View the animated graphic half way down the page
  - It will take about 10 minutes to view the full playable-gif.

64

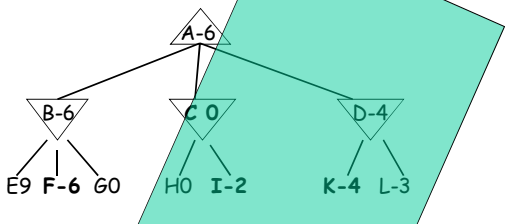
Minimax Eg



- Selecting B lets the opponent "clobber" us - will end up in F(-6)
- So, select C as even a clever opponent can't hurt us more than I(-2)

65

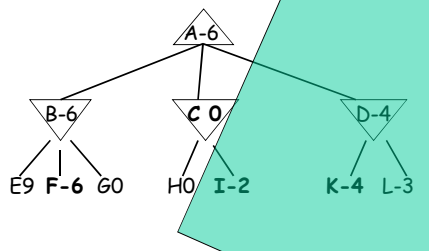
Minimax Eg



- Selecting B lets the opponent "clobber" us - will end up in F(-6)
- So, select C as even a clever opponent can't hurt us more than I(-2)

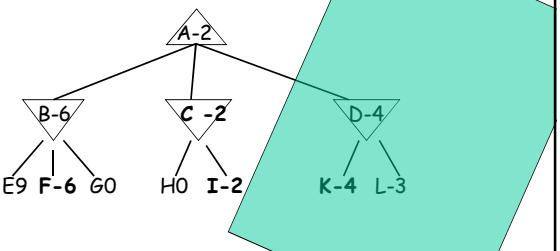
66

Minimax Eg



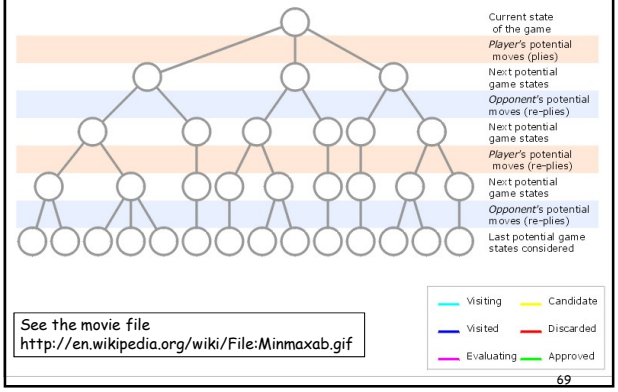
- Selecting B lets the opponent "clobber" us - will end up in F(-6)
- So, select C as even a clever opponent can't hurt us more than I(-2)

Minimax Eg



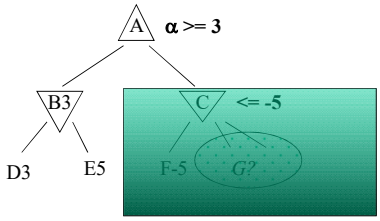
- Selecting B lets the opponent "clobber" us - will end up in F(-6)
- So, select C as even a clever opponent can't hurt us more than I(-2)

Minimax with alpha-beta pruning on a two-person game tree of 4 plies



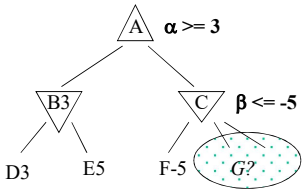
See the movie file
http://en.wikipedia.org/wiki/File:Minmaxab.gif

Simple α–β Pruning



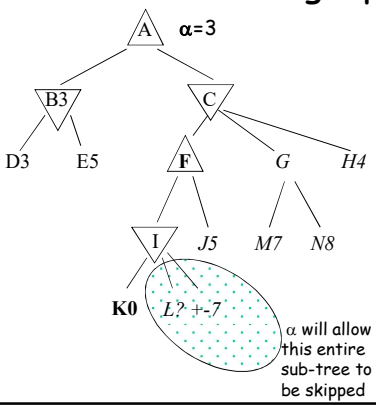
- Regardless of what G is worth, we are guaranteed a score of -5 or less at C
- This is worse than α=3, so we don't continue with the rest of the C sub-tree

Simple α–β Pruning

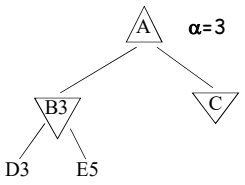


- Regardless of what G is worth, we are guaranteed a score of -5 or less at C
- This is worse than α=3, so we don't continue with the rest of the C sub-tree

α based Pruning - preview



α–β Eg



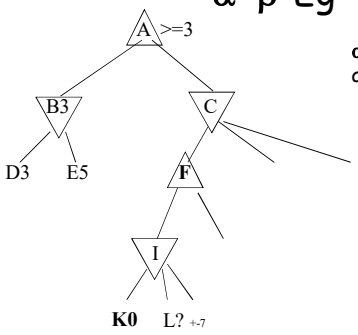
Maxmising level

Minimising level

Maxmising level

Minimising level

α–β Eg

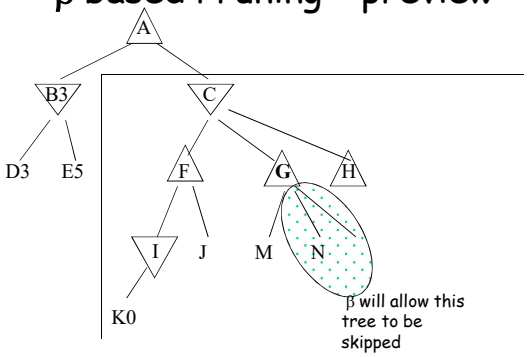


After examining K0, so *I* <= 0 (β=0), thus F >= 0 (which would α=0) - so abandon the rest of the C sub-tree

Using alpha

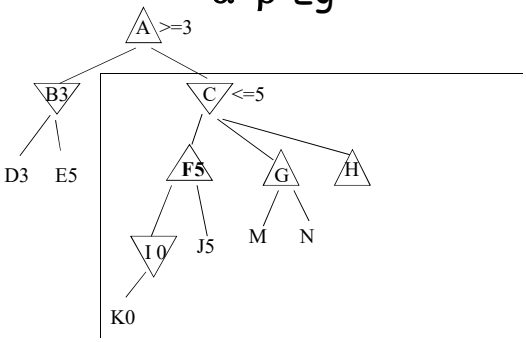
- D3 gets backed up so A>3 and α gets 3.
- Then K<-0, so I is guaranteed a max of 0, (β=0) which means F is guaranteed a minimum of 0
- This is less than the existing α=3, so we don't need to explore the rest of I!
  - so skip nodes: L and its siblings

β based Pruning - preview



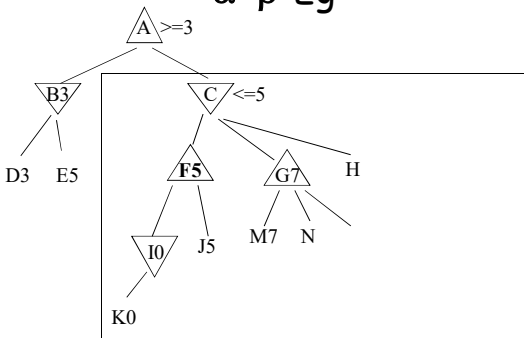
Moving on to the opponents move...

α–β Eg



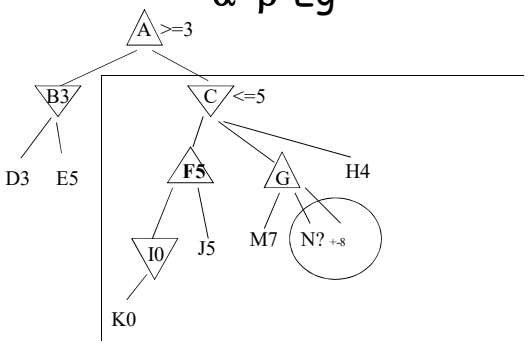
after examining J5, this is passed to F becoming α(*C*)<=5. Now...

α–β Eg



Expanding G to M we get 7, which is passed up to G. But 7>β so this player will not choose G.

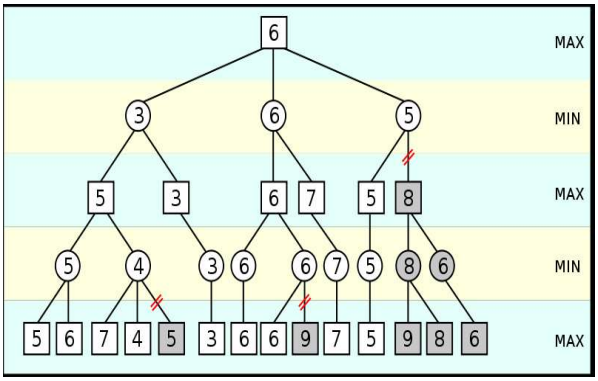
α–β Eg



so the rest of the G sub-tree can be "pruned"!

79

α-β Pruning Example



80

Using beta

- β prunes branches of a Maximising node
- after finding that Fβ=5
  - C is guaranteed 5 or less
- Now expand G, giving M7 which is passed back to G
- but 7 > 5(beta), my opponent trying to minimise the Heuristic value
  - so player will not choose G as this would lead to a score of at least 7
  - so skip the rest of G

81

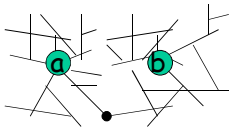
Quiescent Search

- A further optimisation strategy utilized by the Deep Blue Chess program
- *Don't cut off the search in the middle of an exchange of pieces*
- If there is a large variation in the heuristic value across "sister" states, then extend the search depth for that sub-stree

82

Bi-Directional Graph Search

- **Graph Searching** algorithm
  - labeled, interleaved iterative deepening



http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon

83

6 degrees of Kevin Bacon

- If you use Kevin Bacon (the actor) as an end point, you can link him in six degrees or less to almost any other performer.
- Here's how we connect Kevin Bacon to the Alien in the Alien1 movie:
  - Kevin Bacon was in Footloose with Dianne Weist,
  - who was in The Birdcage with Gene Hackman,
  - who was in The Firm with Holly Hunter,
  - who was in Copycat with Sigourney Weaver,
  - who was in Alien with the Alien.

84

Big Bushy Networks

- N neighbours for each node
- X look-ahead
- X^N -> 10^8 = 100,000,000
- 2 * $X^{(n/2)}$ -> 2 * 10^4 = 20,000

- **Small World Phenomenon**
- Choosing any 2 random people on earth, and looking at friends and friends of friends, how many "links" separate those people?

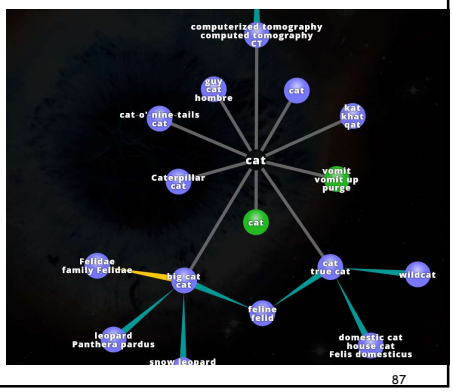
85

Graph: Shortest Path

- Searching in graphs is central to many disciplines
- Epidemiology simulations and understanding
- Estimates of similarity between words
  - ***visuwords.com/***
- What is average number of links separating 2 pages on the internet?

86

Compute similarity between words in a Thesaurus style graph



87

Conclusion

- Search can solve problems
- Heuristic search can solve (some) very complex problems

1. A* Algorithm eliminates duplicate paths to search states
2. MiniMax used for adversary-based search
3. α-β pruning optimises adversary-based search

- Graph searching is broadly applicable

88