KR-IST - Lecture 5a
Game playing with Minimax and Pruning

Chris Thornton

November 16, 2011

Introduction

An important application of AI search methods has been in the domain of 2-person games, such as draughts (checkers) and chess.

Until quite recently (late 1990s) it was widely believed by many that hard problems of intelligence would never be solved by computer.

Chess was often put forward as a good example.

Then, in May 1997, an IBM machine known as 'Deep Blue' defeated chess grandmaster Garry Kasparov.

No special techniques were used to achieve the victory. Deep Blue relied on tried and trusted methods.

The version of Deep Blue which beat Kasparov was able to evalute more than 200 million chess states per second.

Kasparov and Deep Blue

Deep Blue

Recordings

Recordings

- Kasparov points out that he is a human being: http://www.cnn.com/WORLD/9705/11/chess.update/kasparov.scar

Recordings

- Kasparov points out that he is a human being:
  http://www.cnn.com/WORLD/9705/11/chess.update/kasparov.scar
- Kasparov predicts Deep Blue will eventually be beaten:
  http://www.cnn.com/WORLD/9705/11/chess.update/kasparov.beat

Recordings

- Kasparov points out that he is a human being:
  http://www.cnn.com/WORLD/9705/11/chess.update/kasparov.scar
- Kasparov predicts Deep Blue will eventually be beaten:
  http://www.cnn.com/WORLD/9705/11/chess.update/kasparov.beat

Adapting search for game playing

Deep Blue used ordinary search methods. and the standard approach for adapting those methods to the problem of game-play.

Games like chess can readily be seen in terms of transitions between states. Transitions are moves; states are board configurations.

Normally, we would then solve the problem by searching for a path of transitions (i.e., moves) connecting the start state with a goal state.

Unfortunately, in this context, we 'lose' control over the choice of move every other turn.

Using search for evaluation

In a 2-person game, a solution path is unobtainable because we never know what the other player is going to do at any stage.

What we need to work out is the best move.

In the minimax method we use the search process not to find a solution path, but to derive the most accurate evaluation of the possible moves, i.e., an evaluation which takes into account the implications that any given move will have later in the game.

Minimax method

There are three elements to the minimax method.

Minimax method

There are three elements to the minimax method.

(1) Expand the search tree all the way down to a game conclusion (win, lose or draw). If this is too much search, choose a suitable cutoff.

Minimax method

There are three elements to the minimax method.

(1) Expand the search tree all the way down to a game conclusion (win, lose or draw). If this is too much search, choose a suitable cutoff.
(2) Obtain an evaluation of the relevant terminal state. (e.g., positive for a win, negative for a lose and neutral for a draw). This is known as the **static evaluation**.

Minimax method

There are three elements to the minimax method.

(1) Expand the search tree all the way down to a game conclusion (win, lose or draw). If this is too much search, choose a suitable cutoff.

(2) Obtain an evaluation of the relevant terminal state. (e.g., positive for a win, negative for a lose and neutral for a draw). This is known as the **static evaluation**.

(3) Then **back-up** the evaluations, level by level, working on the basis that when it is the opponent's turn, they will chose a transition which achieves the worst outcome from our point of view, and whenever it is our turn to move, we will choose the best.

Minimax method

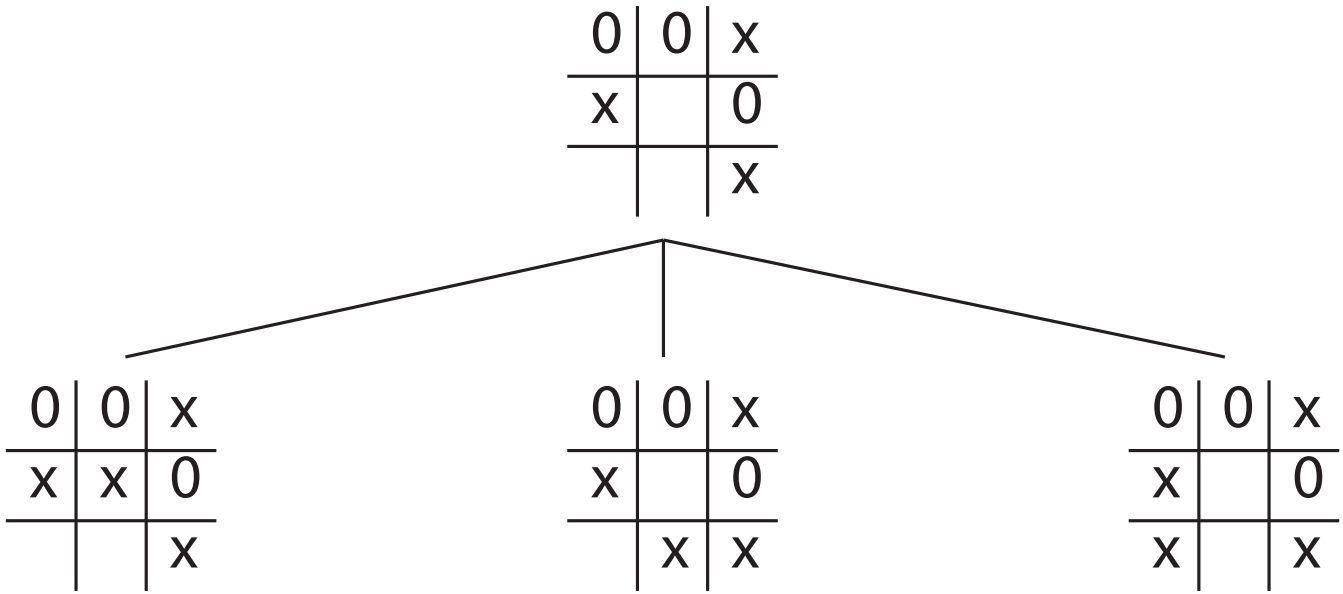
There are three elements to the minimax method.

(1) Expand the search tree all the way down to a game conclusion (win, lose or draw). If this is too much search, choose a suitable cutoff.
(2) Obtain an evaluation of the relevant terminal state. (e.g., positive for a win, negative for a lose and neutral for a draw). This is known as the **static evaluation**.
(3) Then **back-up** the evaluations, level by level, working on the basis that when it is the opponent's turn, they will chose a transition which achieves the worst outcome from our point of view, and whenever it is our turn to move, we will choose the best.

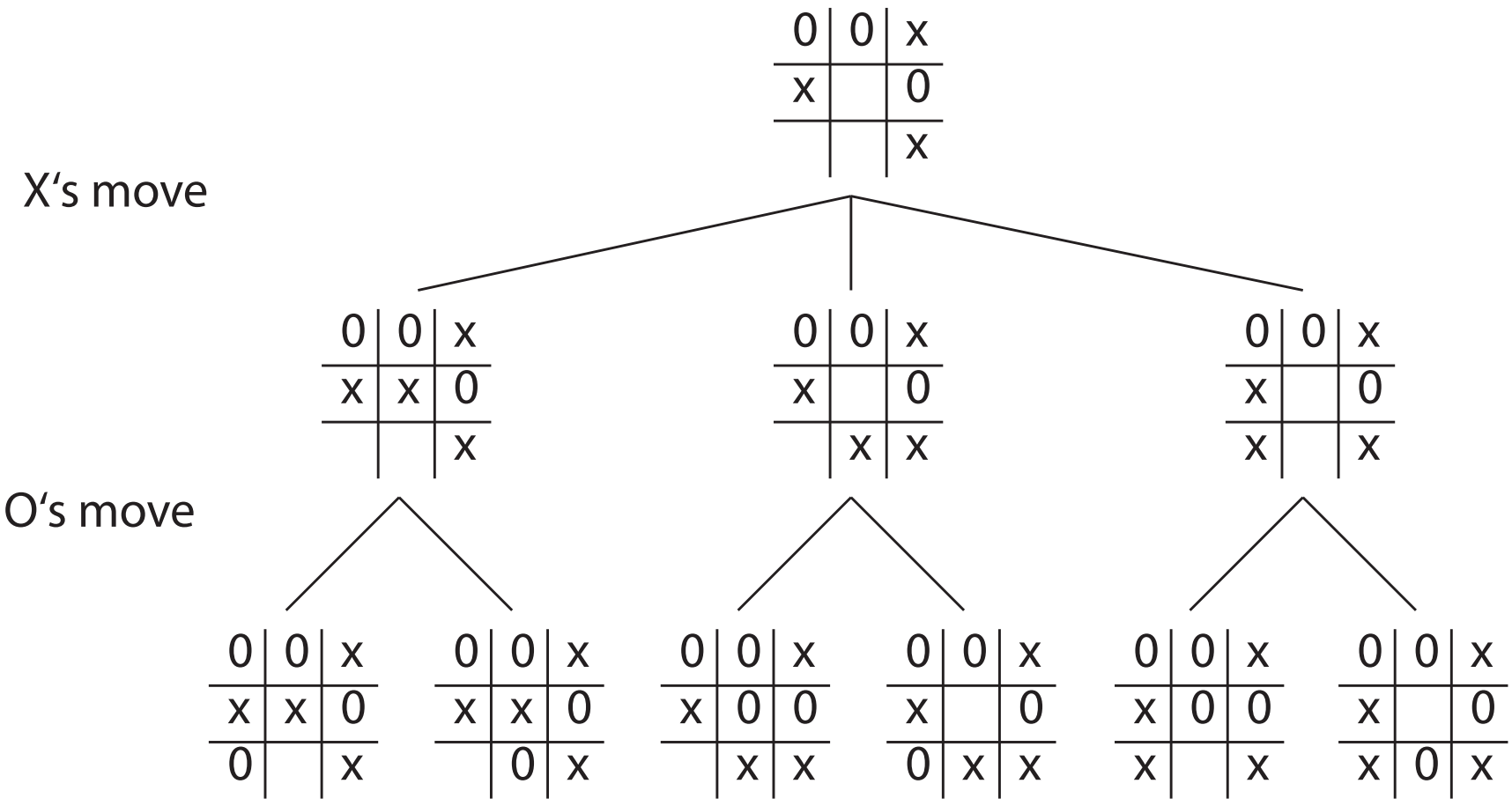
To do this we need to identify the *minimum* evaluation in any level of the tree corresponding to the opponent's move, and the *maximum* otherwise.

Worked example

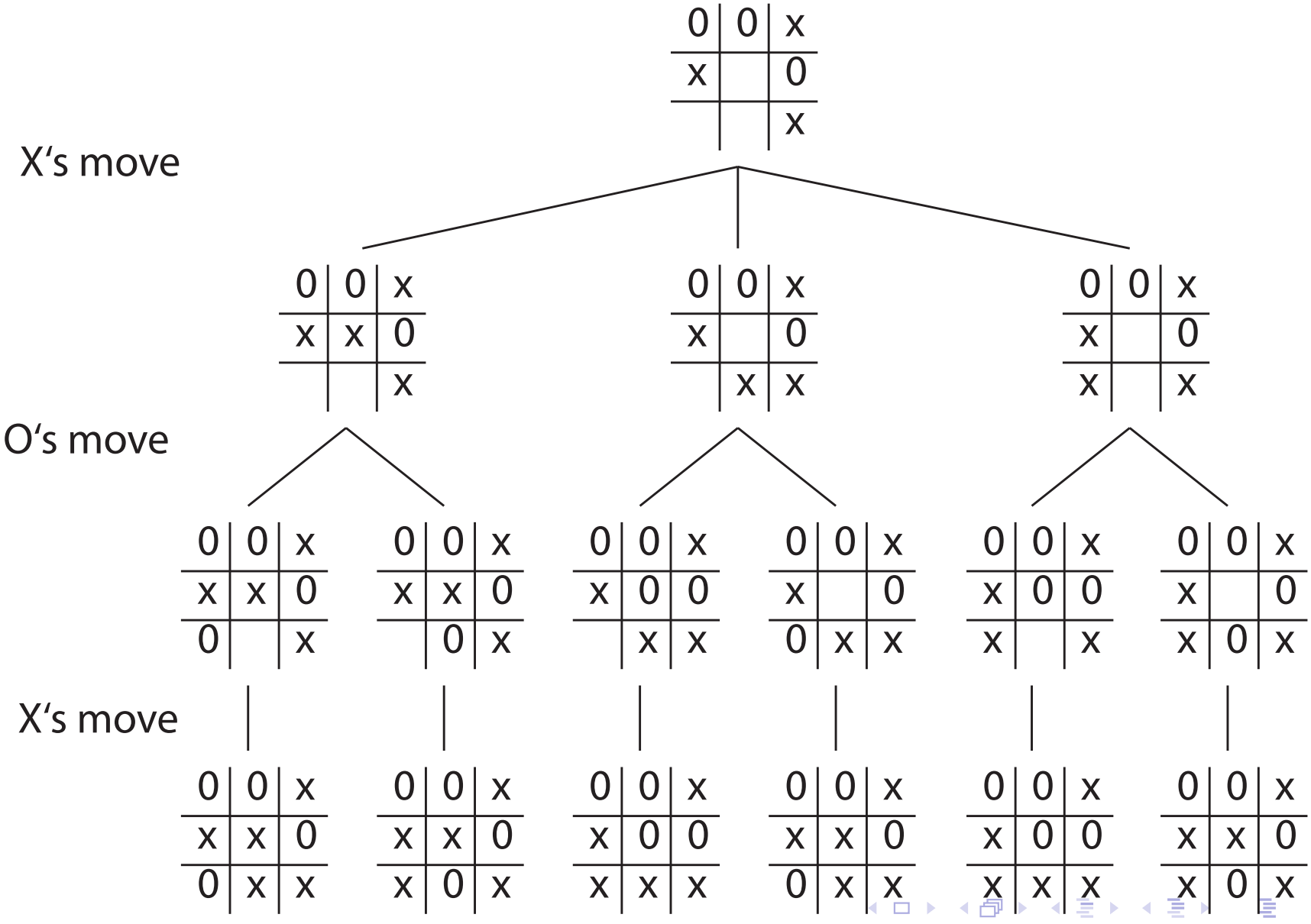
X's move



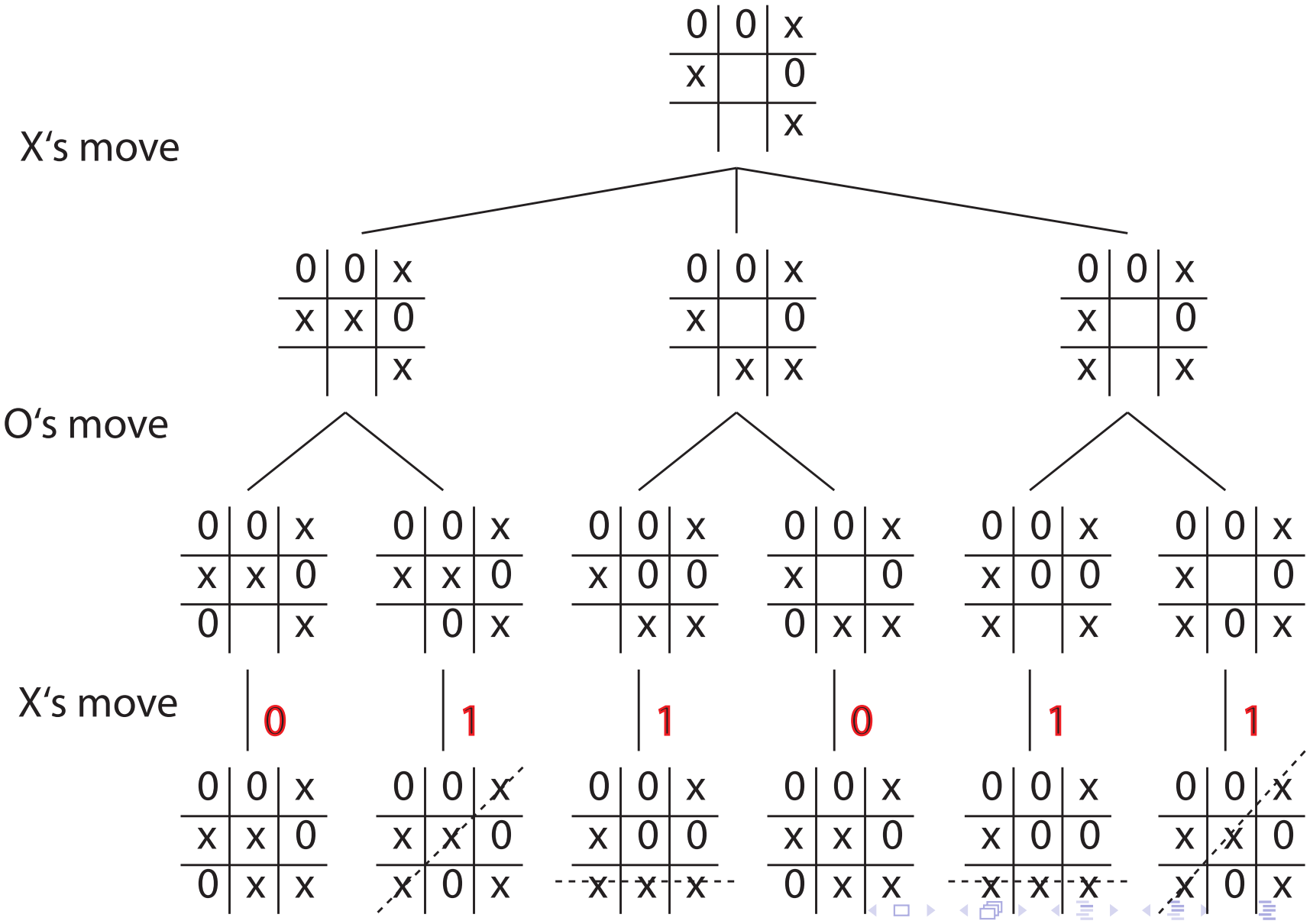
Cont.



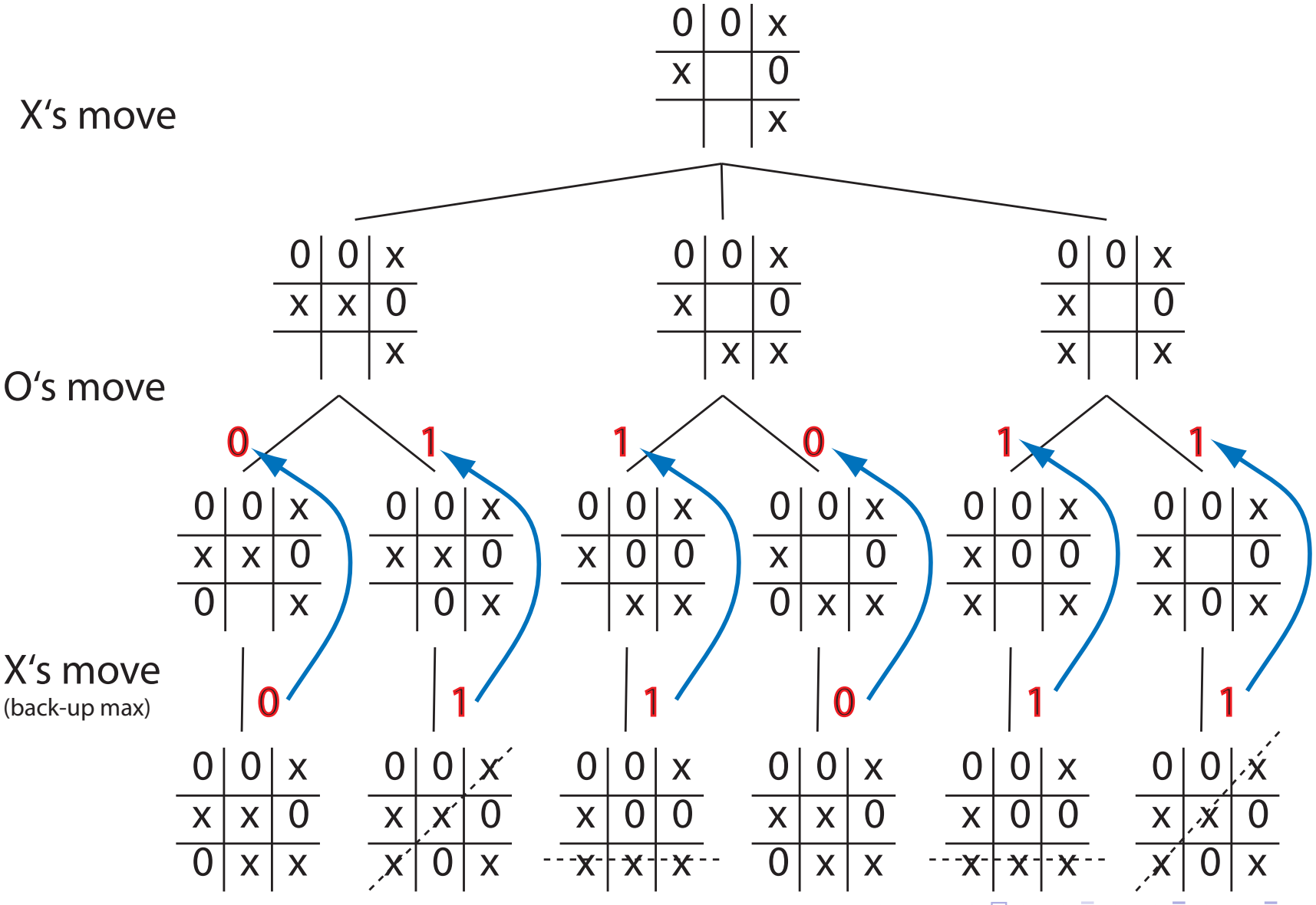
Cont.



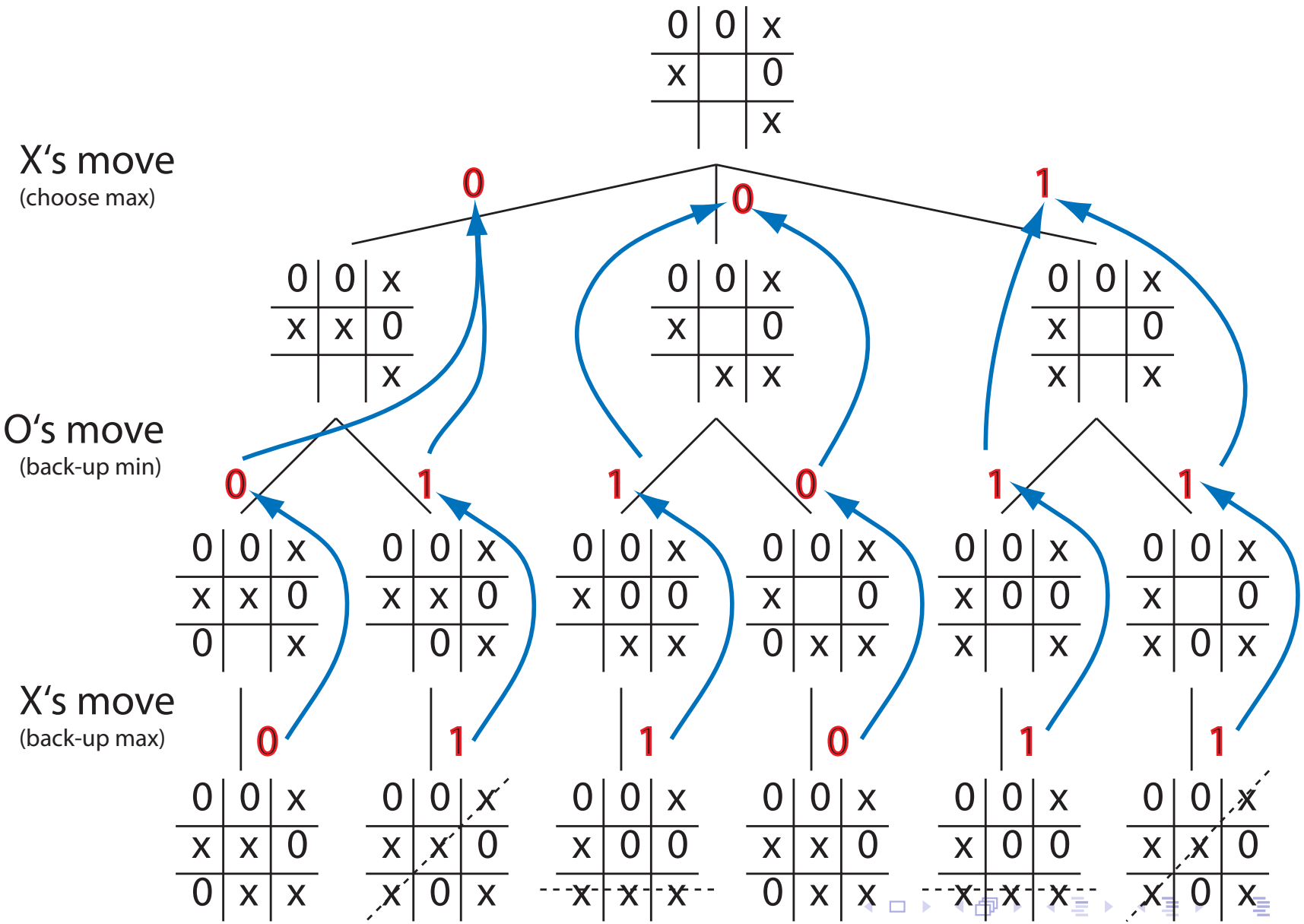
Cont.



Cont.



Evaluation obtained



Negmax simplification

Implementing minimax can be a pain because of the need to alternate between minimisation and maximisation in the backing-up of evaluations.

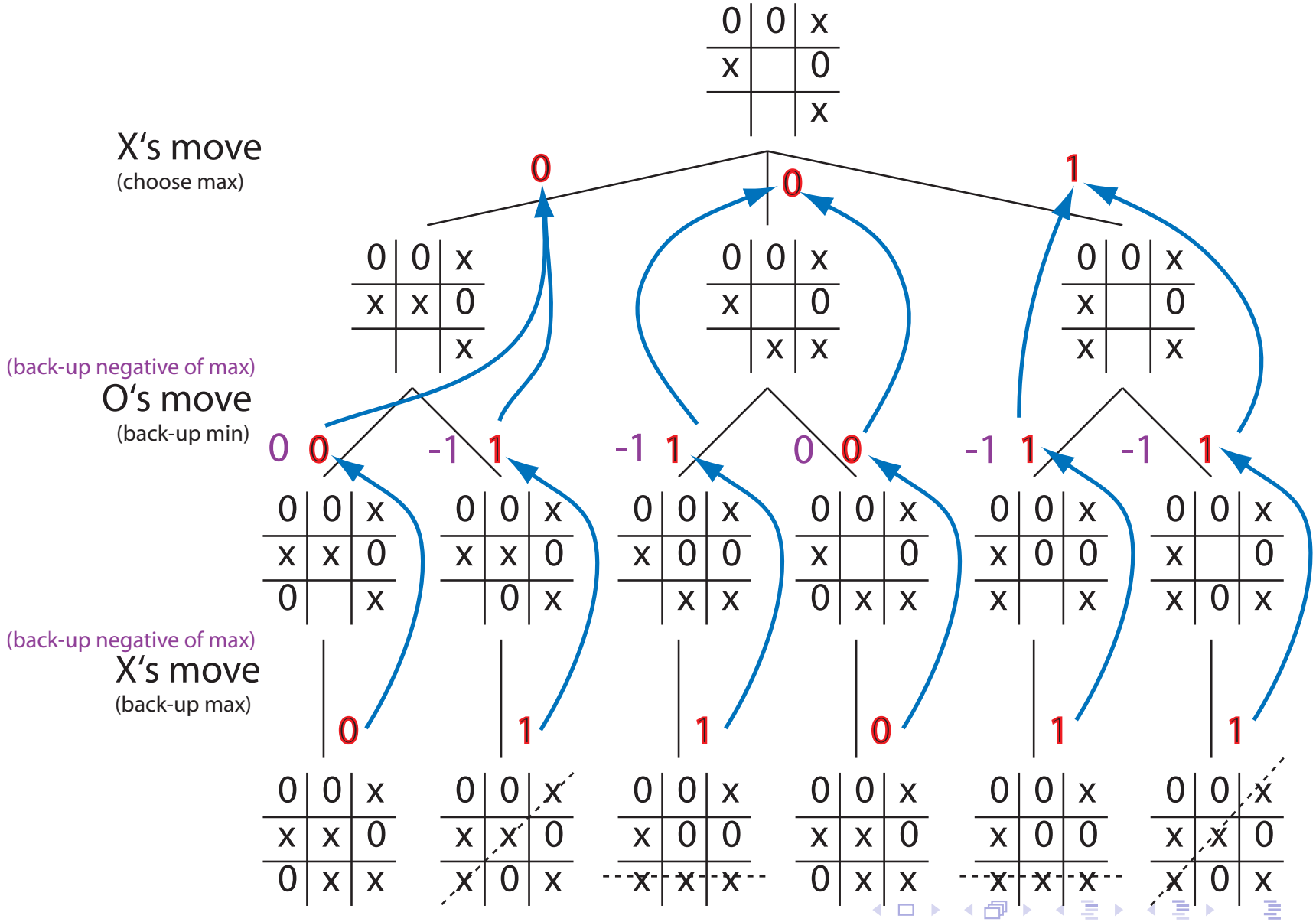
The **negmax** idea gets around this problem.

Board states are still evaluated from the 'current' player's point of view (i.e., whichever player has control at the given depth). but the value which is backed-up is always the *negative* of the *maximum*.

As in minimax, the effect is to ensure that the value backed-up is the value of the *worst* outcome that the opponent can achieve from our point of view.

But the code to implement the method can be written using a simple recursive procedure.

Negmax illustration



Alpha-beta pruning

When using minimax (or negmax), situations can arise when search of a particular branch can be safely terminated.

Alpha-beta pruning

When using minimax (or negmax), situations can arise when search of a particular branch can be safely terminated.

- Applying an **alpha-cutoff** means we stop search of a particular branch because we see that we already have a better opportunity elsewhere.

Alpha-beta pruning

When using minimax (or negmax), situations can arise when search of a particular branch can be safely terminated.

- Applying an **alpha-cutoff** means we stop search of a particular branch because we see that we already have a better opportunity elsewhere.
- Applying a **beta-cutoff** means we stop search of a particular branch because we see that the opponent already has a better opportunity elsewhere.

Alpha-beta pruning

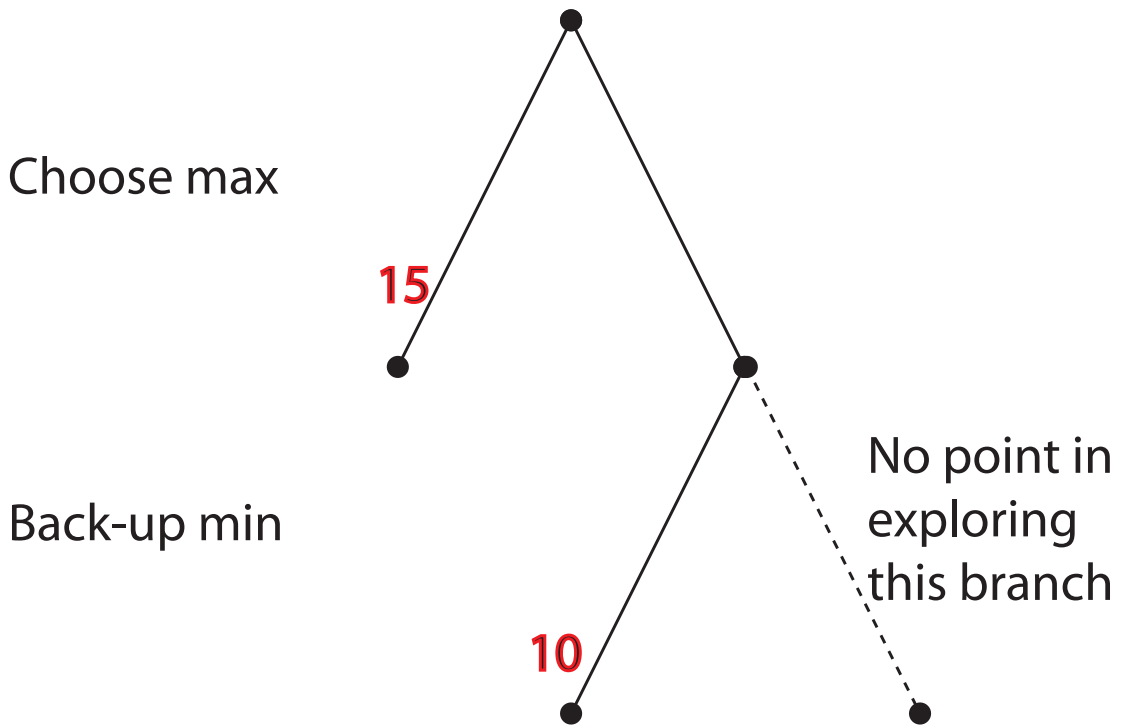
When using minimax (or negmax), situations can arise when search of a particular branch can be safely terminated.

- Applying an **alpha-cutoff** means we stop search of a particular branch because we see that we already have a better opportunity elsewhere.
- Applying a **beta-cutoff** means we stop search of a particular branch because we see that the opponent already has a better opportunity elsewhere.

Applying both forms is **alpha-beta pruning**.

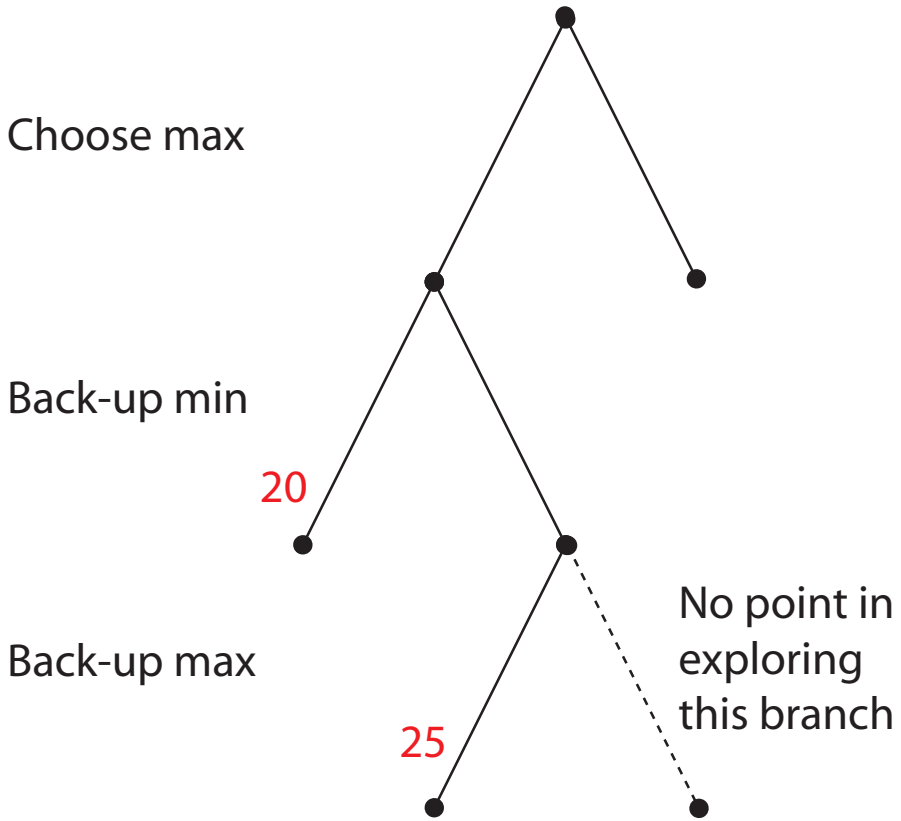
Alpha-cutoff

If, from some state S, the opponent can achieve a state with a *lower* value for us than one achievable in another branch. we will certainly *not* move the game to S. We do not need to expand S.



Beta-cutoff

If, from some state S, we would be able to achieve a state which has a *higher* value for us than one the opponent can hold us to in another branch, we can assume the opponent will not choose S.



Summary

Summary

- Adapting search for game playing

Summary

- Adapting search for game playing
- Minimax method

Summary

- Adapting search for game playing
- Minimax method
- Negmax simplification

Summary

- Adapting search for game playing
- Minimax method
- Negmax simplification
- Alpha-cutoff

Summary

- Adapting search for game playing
- Minimax method
- Negmax simplification
- Alpha-cutoff
- Beta-cutoff

Summary

- Adapting search for game playing
- Minimax method
- Negmax simplification
- Alpha-cutoff
- Beta-cutoff

Questions

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?
- What is special about a *static* evaluation?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?
- What is special about a *static* evaluation?
- How are static evaluations of board states in a 2-person game derived?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?
- What is special about a *static* evaluation?
- How are static evaluations of board states in a 2-person game derived?
- In a game like chess, where the full search tree is very large, how might reasonable static evaluations of board states be obtained?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?
- What is special about a *static* evaluation?
- How are static evaluations of board states in a 2-person game derived?
- In a game like chess, where the full search tree is very large, how might reasonable static evaluations of board states be obtained?
- What is the advantage of the negmax method?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?
- What is special about a *static* evaluation?
- How are static evaluations of board states in a 2-person game derived?
- In a game like chess, where the full search tree is very large, how might reasonable static evaluations of board states be obtained?
- What is the advantage of the negmax method?
- What is the minimum depth of search for the application of alpha-cutoffs?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?
- What is special about a *static* evaluation?
- How are static evaluations of board states in a 2-person game derived?
- In a game like chess, where the full search tree is very large, how might reasonable static evaluations of board states be obtained?
- What is the advantage of the negmax method?
- What is the minimum depth of search for the application of alpha-cutoffs?
- What is the minimum depth of search for the application of beta-cutoffs?

Questions

- Could we use the A* algorithm to improve the effectiveness of minimax?
- Why is there no point searching for a solution path in a game-playing problem?
- What is minimised and what is maximised in minimax search?
- What is special about a *static* evaluation?
- How are static evaluations of board states in a 2-person game derived?
- In a game like chess, where the full search tree is very large, how might reasonable static evaluations of board states be obtained?
- What is the advantage of the negmax method?
- What is the minimum depth of search for the application of alpha-cutoffs?
- What is the minimum depth of search for the application of beta-cutoffs?

Exercises

Exercises

- In the game of noughts and crosses (tic-tac-toe), the two players take it in turns to capture an empty cell of a 3x3 grid. The game is won once a line of three cells has been captured. Devise a suitable representation scheme for states in this game.

Exercises

- In the game of noughts and crosses (tic-tac-toe), the two players take it in turns to capture an empty cell of a 3x3 grid. The game is won once a line of three cells has been captured. Devise a suitable representation scheme for states in this game.
- Estimate the branching factor of the space.

Exercises

- In the game of noughts and crosses (tic-tac-toe), the two players take it in turns to capture an empty cell of a 3x3 grid. The game is won once a line of three cells has been captured. Devise a suitable representation scheme for states in this game.
- Estimate the branching factor of the space.
- Calculate the maximum depth of a search tree in this game.

Exercises

- In the game of noughts and crosses (tic-tac-toe), the two players take it in turns to capture an empty cell of a 3x3 grid. The game is won once a line of three cells has been captured. Devise a suitable representation scheme for states in this game.
- Estimate the branching factor of the space.
- Calculate the maximum depth of a search tree in this game.
- Calculate the total size of the state space in this game.

Exercises

- In the game of noughts and crosses (tic-tac-toe), the two players take it in turns to capture an empty cell of a 3x3 grid. The game is won once a line of three cells has been captured. Devise a suitable representation scheme for states in this game.
- Estimate the branching factor of the space.
- Calculate the maximum depth of a search tree in this game.
- Calculate the total size of the state space in this game.

Exercises cont.

Exercises cont.

- On the basis that the underscore represents an unfilled cell, draw out the full tree of states that can be reached from the state

Exercises cont.

- On the basis that the underscore represents an unfilled cell, draw out the full tree of states that can be reached from the state

```
X X O
X O O
_ _ _
```

Exercises cont.

- On the basis that the underscore represents an unfilled cell, draw out the full tree of states that can be reached from the state

```
X X O
X O O
_ _ _
```

- Annotate the tree to differentiate levels where X has control (i.e., where it is X's turn) from the levels at which O has control.

Exercises cont.

- On the basis that the underscore represents an unfilled cell, draw out the full tree of states that can be reached from the state

```
X X O
X O O
_ _ _
```

- Annotate the tree to differentiate levels where X has control (i.e., where it is X's turn) from the levels at which O has control.
- Annotate nodes of the tree which represent won/lost states, giving them a value of one 1 if the state is won by X, and zero if it is won by O.

Exercises cont.

- On the basis that the underscore represents an unfilled cell, draw out the full tree of states that can be reached from the state

```
X X O
X O O
_ _ _
```

- Annotate the tree to differentiate levels where X has control (i.e., where it is X's turn) from the levels at which O has control.
- Annotate nodes of the tree which represent won/lost states, giving them a value of one 1 if the state is won by X, and zero if it is won by O.

Exercises cont.

Exercises cont.

- Show how the evaluations of immediate successor states can be produced by backing up evaluations of terminal states using first MINIMAX and then NEGMAX.

Exercises cont.

- Show how the evaluations of immediate successor states can be produced by backing up evaluations of terminal states using first MINIMAX and then NEGMAX.
- How well would the NEGMAX evaluation work if you used -1 as the static evaluation for a lost state?

Exercises cont.

- Show how the evaluations of immediate successor states can be produced by backing up evaluations of terminal states using first MINIMAX and then NEGMAX.
- How well would the NEGMAX evaluation work if you used -1 as the static evaluation for a lost state?
- Devise a representation scheme for states in this game which minimises the difficulty of generating successors and evaluating states.

Exercises cont.

- Show how the evaluations of immediate successor states can be produced by backing up evaluations of terminal states using first MINIMAX and then NEGMAX.
- How well would the NEGMAX evaluation work if you used -1 as the static evaluation for a lost state?
- Devise a representation scheme for states in this game which minimises the difficulty of generating successors and evaluating states.
- In this game, the first player to move can always force a draw provided a certain procedure is followed. Devise a way of using the evaluation mechanism (i.e., backing up of terminal evaluations) to identify what this procedure is.

Exercises cont.

- Show how the evaluations of immediate successor states can be produced by backing up evaluations of terminal states using first MINIMAX and then NEGMAX.
- How well would the NEGMAX evaluation work if you used -1 as the static evaluation for a lost state?
- Devise a representation scheme for states in this game which minimises the difficulty of generating successors and evaluating states.
- In this game, the first player to move can always force a draw provided a certain procedure is followed. Devise a way of using the evaluation mechanism (i.e., backing up of terminal evaluations) to identify what this procedure is.

Resources

Resources

- Another chess applet: http://chess.captain.at/

Resources

- Another chess applet: http://chess.captain.at/
- CNN website on Deep Blue v. Kasparov: http://www.cnn.com/WORLD/9705/11/chess.update/

Resources

- Another chess applet: http://chess.captain.at/
- CNN website on Deep Blue v. Kasparov: http://www.cnn.com/WORLD/9705/11/chess.update/