16.410/413
Principles of Autonomy and Decision Making

Lecture 24: Sequential Games

Emilio Frazzoli

Aeronautics and Astronautics
Massachusetts Institute of Technology

December 6, 2010

Outline

1. Game Theory
   - Overview
   - Games in normal form: Nash equilibria, pure and mixed strategies
   - Games in extensive form
2. Sequential Games

Game Theory

Games

- Multiple "players" independently choose actions, based on the available information, to pursue individual goals.
- Created by John Von Neumann in the late 1920s.

Applications

- Economics
- Political Science/Diplomacy/Military Strategy
- Biology
- Computer Science/Artificial Intelligence
- Computer games
- Resource allocation in networks (internet, cell phones,...)
- Robust control (disturbance rejection)
- Air traffic collision avoidance
- UAV Pursuit-evasion

Types of Games

Zero-sum games

All the gains/losses of a player are exactly balanced by the gains/losses of all other players (possibly modulo a constant).

- Zero-sum: a game of chess, tic-tac-toe, rock/paper/scissors, poker (with no house cut), risk, dividing a cake, presidential election, dogfights (?).
- Non-zero sum: contract negotiation, trade agreements, chicken and hawk/dove game, prisoners dilemma, MMORPGs, dogfights (?).

Cooperative vs. non-Cooperative Games

- A game is cooperative if groups of players may enforce binding agreements. (E.g., through a third party, such as a legal system.)
- A game is non-cooperative if no such binding agreements exist. Cooperation may occur, but is self-serving.

Types of Games, cont'd.

Symmetric games

The game is invariant to relabeling on the players.

Types of Games, cont'd.

Symmetric games

The game is invariant to relabeling on the players.

Sequential/simultaneous games

In a sequential game, the players act at well-defined turns, and have some information on what the other(s) did at previous turns. In a simultaneous game, all players act at the same time, or equivalently, have no information on the actions of the others in the same turn.

Types of Games, cont'd.

Symmetric games

The game is invariant to relabeling on the players.

Sequential/simultaneous games

In a sequential game, the players act at well-defined turns, and have some information on what the other(s) did at previous turns. In a simultaneous game, all players act at the same time, or equivalently, have no information on the actions of the others in the same turn.

Perfect information

In a sequential game, players have perfect knowledge of what others did in all previous turns.

Types of Games, cont'd.

Symmetric games

The game is invariant to relabeling on the players.

Sequential/simultaneous games

In a sequential game, the players act at well-defined turns, and have some information on what the other(s) did at previous turns. In a simultaneous game, all players act at the same time, or equivalently, have no information on the actions of the others in the same turn.

Perfect information

In a sequential game, players have perfect knowledge of what others did in all previous turns.

Are the games listed above symmetric/sequential/perfect information games?

Games in normal form

Normal Form

Suitable for simultaneous games, or for summarizing the effects of "strategies."

Games in normal form

Normal Form

Suitable for simultaneous games, or for summarizing the effects of "strategies."

Prisoner's dilemma

Two suspects ("players") are arrested and accused of a crime. Since the police do not have enough evidence, they can be convicted only if at least one of the suspects testifies against the other.

	Player B cooperates	**Player B defects**
Player A cooperates	(-1,-1)	(-10,0)
Player A defects	(0,-10)	(-5,-5)

Games in normal form

Normal Form

Suitable for simultaneous games, or for summarizing the effects of "strategies."

Prisoner's dilemma

Two suspects ("players") are arrested and accused of a crime. Since the police do not have enough evidence, they can be convicted only if at least one of the suspects testifies against the other.

	Player B cooperates	**Player B defects**
Player A cooperates	(-1,-1)	(-10,0)
Player A defects	(0,-10)	(-5,-5)

Nash equilibria

- A Nash equilibrium is a choice of strategies such that no player can gain by unilaterally changing his/her strategy.
- Nash equilibria are not necessarily efficient.

Pure and Mixed strategies

Rock-Paper-Scissors

Rock beats Scissors beats Paper beats Rock.

A/B	Rock	Scissors	Paper
Rock	(0,0)	(1,-1)	(-1,1)
Scissors	(-1,1)	(0,0)	(1,-1)
Paper	(1,-1)	(-1,1)	(0,0)

Pure and Mixed strategies

Rock-Paper-Scissors

Rock beats Scissors beats Paper beats Rock.

A/B	Rock	Scissors	Paper
Rock	(0,0)	(1,-1)	(-1,1)
Scissors	(-1,1)	(0,0)	(1,-1)
Paper	(1,-1)	(-1,1)	(0,0)

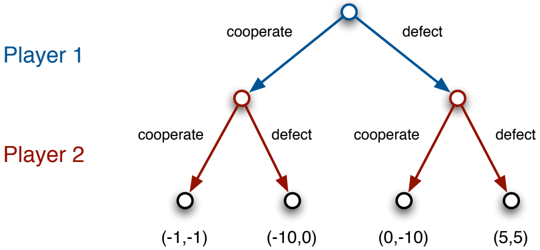
Repeated games and randomized strategies

- Nash proved that any finite game has at least a Nash equilibrium. However, such a Nash equilibrium is not necessarily **pure**, i.e., deterministically defined (each player adopts one strategy).
- In a **mixed** strategy, a player chooses his/her strategy randomly according to a given probability distribution.
- Rock-Paper-Scissors is a typical example of a game with a mixed Nash equilibrium.

Games in extensive form

Extensive Form

- Suitable for games played in sequential "turns."
- Consider the following version of the prisoner's dilemma: is it the same as the one we saw before?



Outline

Zero-Sum Two-Player Sequential Games

Key characteristics

- Two players;
- Zero-sum reward structure (the reward of a player is the cost for the other).
- Sequential moves (from a finite set);
- Perfect information;
- The game terminates in a finite number of steps, no matter how it is played.

Problem data

- An initial state (incl. whose turn it is);
- One or more terminal states;
- State/action pairs;
- The cost/reward associated with terminal states.

Objective

Compute, for each player, a strategy that associates to each state an action that maximizes the reward if the other player plays rationally.

Tic-Tac-Toe

Initial state

Empty board, **X** to go first.

Actions

Place **X** (or **O**) in an empty square.

Terminal states

- Three **X**s or **O**s on the same line is a win.
- No empty squares is a tie.

Reward (at terminal state)

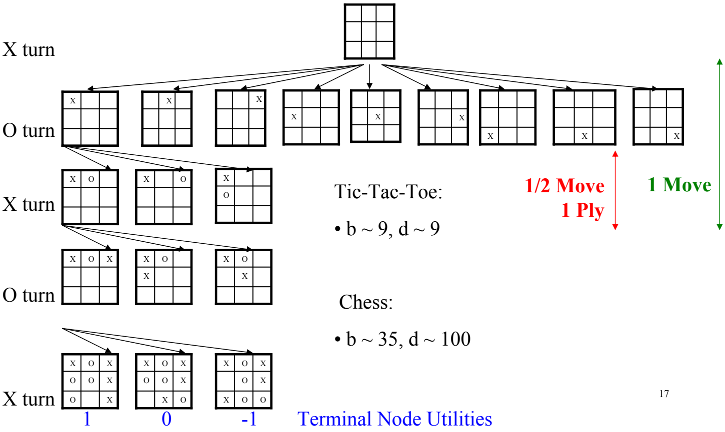
- 1 for a win, 0 for a tie, -1 for a loss.

X	X	O
	O	
X		

Notes

- Max depth: 9 "plies" (i.e., moves)
- Branching: at most $(10 - i)$ possible moves at the i-th ply.

Tic-Tac-Toe Game Tree



The complete tree has no more than 9! $= 362880$ nodes
(not accounting for symmetries and termination conditions).

Tree search for a single player

Tic-Tac-Toe

- Let us assume that we can construct the whole tree representing all possible play sequences.
- If you were playing "solitaire tic-tac-toe," you would choose one of the branches leading to a win (max reward), and place Xs, and Os consequently.
- Unfortunately, you do not get to choose the Os!
- Your adversary, if he/she had to choose, would seek to minimize your reward (i.e., maximize his/her own)!

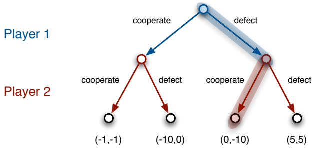
Tree search for a single player

Tic-Tac-Toe

- Let us assume that we can construct the whole tree representing all possible play sequences.
- If you were playing "solitaire tic-tac-toe," you would choose one of the branches leading to a win (max reward), and place Xs, and Os consequently.
- Unfortunately, you do not get to choose the Os!
- Your adversary, if he/she had to choose, would seek to minimize your reward (i.e., maximize his/her own)!

Sequential prisoner's dilemma (note: not zero-sum)

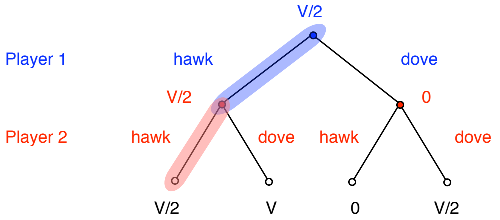
- In the sequential prisoner dilemma game, the best plan for the first player is to defect and make the other player cooperate.
- The other player may not agree...
- in fact, it will be better for him/her to defect!



Two-player search: Min-Max

- Each player tries to **maximize his/her own reward**, **assuming that the other player uses an optimal strategy** (for his/her own reward)
- In a zero-sum game, this is equivalent to saying that Player 1 is trying to maximize his/her reward, and Player 2 is trying to minimize Player 1's reward
  i.e., Player 1 MAXimizes, Player 2 MINimizes.
- In practice:
  - build the whole tree, find terminal states and evaluate the corresponding rewards;
  - Moving backwards from the leaves, associate to parent nodes the MIN or MAX value of all their children (depending on whose turn it is).

Hawk/Dove game with no cost for confrontation



Practical Considerations

The MinMax (or MiniMax) algorithm finds optimal strategies. However, it requires building/searching the complete game tree.

- Tic-Tac-Toe: about 10^5 nodes.
- Chess: about $35^{100} = 2.5 \times 10^{154}$ nodes!

In order to limit the complexity of the search, build a partial tree, i.e., a tree whose leaves are not necessarily terminal states. Two problems:

- How do we choose when to stop expanding the tree?
  - Fixed cut-off, e.g., depth d.
  - Iterative deepening.
- What value do we associate to the leaves?
  - Use "evaluation functions," ideally designed to give a good estimate of the terminal reward given an intermediate state (same function as heuristic functions).
  - E.g., in chess, you can give a numeric value to each piece in play.

Alpha-Beta Pruning

- Still, the complexity of a good search might be excessive.
- Performance gains can be attained by using **branch and bound** techniques, removing from the search subtrees that are guaranteed to be no better than others already discovered (w.r.t. the actual reward, or the evaluation function, depending on the tree construction).
- In practice:
  - Associate to each node an interval in which the reward can lie. Initialize with $(-\infty, +\infty)$.
  - Do a depth first search, tightening the bounds for the reward, i.e., $[\alpha, \beta]$.
  - If a node provably cannot offer any improvements, prune (i.e., do not search further) the corresponding subtree.

Characteristics of Alpha-Beta Pruning

What are the α and β values of a vertex *s*?

- α: this represent the largest known lower bound on the value of the game if it started at the vertex *s* (the value of *s*).
- β: this represent the smallest known upper bound on the value of *s*.

Initial values of (α, β) for a vertex *s*

- If *s* is the root of the tree: $(\alpha, \beta) = (-\infty, \infty)$.
- If *s* is a terminal state, i.e., a leaf of the tree: $\alpha = \beta =$ value of *s*.

Properties of (α, β) for a vertex *s*

- α never decreases.
- β never increases.

Pseudocode for alpha-beta

minimax(node, player, depth)

```
return alphabeta(node, player, depth, −∞, ∞)
```

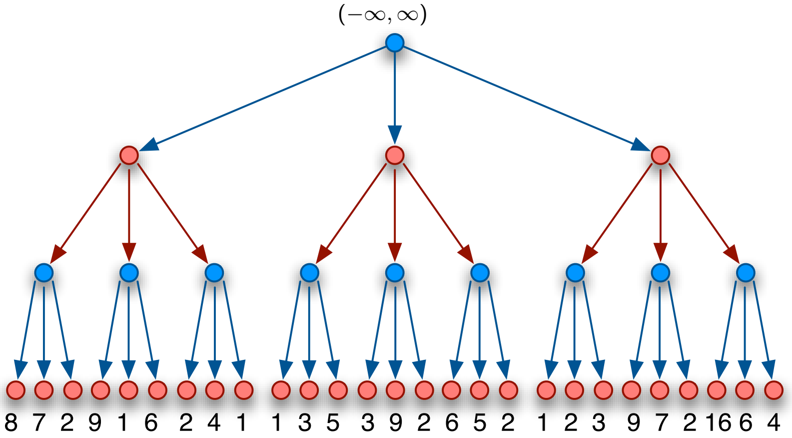
alphabeta(node, player, depth, α, β)

```
if node is a terminal node, or depth = 0 then
    return the (heuristic) value of the node
foreach child of node do
    if player == MAX then
        aux = alphabeta(child, MIN, depth-1, α, β));
        if  aux > α then α = aux ;                          // Adjust the bound
        if α > β then break;                             // No reason to continue...
        return α ;                    // This is the best result for MAX from here
    else
        aux = alphabeta(child, MAX, depth-1, α, β));
        if  aux < β then β = aux ;                          // Adjust the bound
        if α > β then break;                             // No reason to continue...
        return β ;                    // This is the best result for MIN from here
```

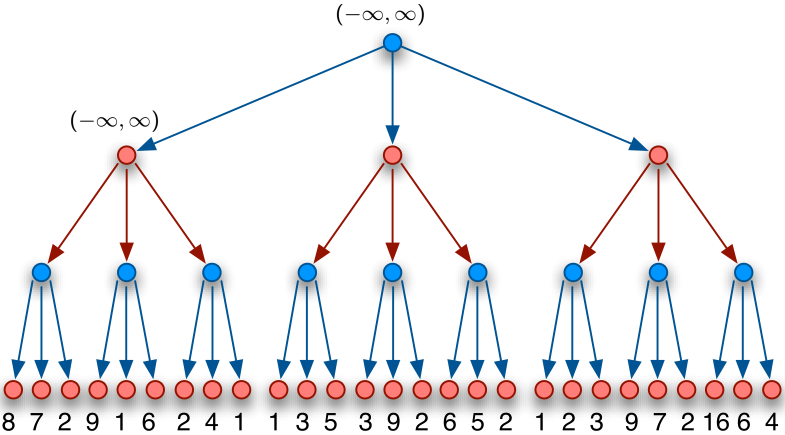
Alpha-Beta in practice

- Visit the vertices of the tree in Depth-First Search order.
- At the first visit of a MAX node, set its β value to the β value of its parent.
- At the first visit of a MIN node, set its α value to the α value of its parent.
- Every time a MAX node is revisited, update its α value to the maximum known value of its children.
- Every time a MIN node is revisited, update its β value to the minimum known value of its children.
- If at any point it happens that $\alpha \geq \beta$, it means that that particular vertex cannot be part of an optimal solution, since there is at least another solution that is certainly no worse than any solution containing the vertex $\Rightarrow$ there is no point in further investigating the subtree rooted at that vertex $\Rightarrow$ **PRUNE THE SUBTREE**.
- When leaving a vertex *s* for the last time (i.e., when moving back towards the root), set its value to α if *s* is a MAX node, or to β if *s* is a MIN node.

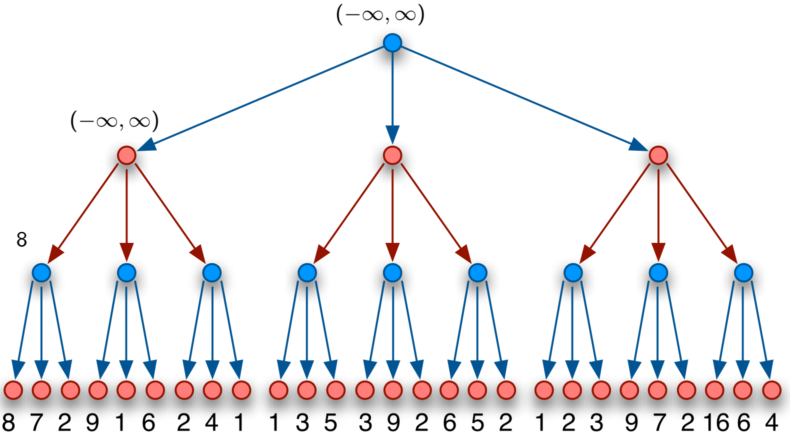
Alpha-Beta Pruning Example



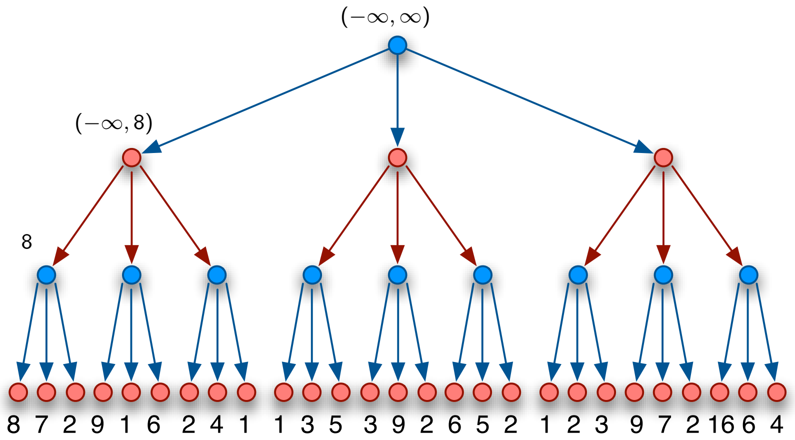
Alpha-Beta Pruning Example



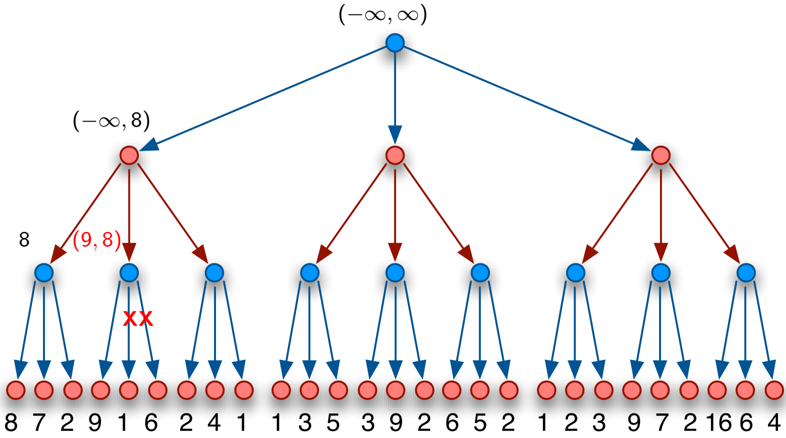
Alpha-Beta Pruning Example



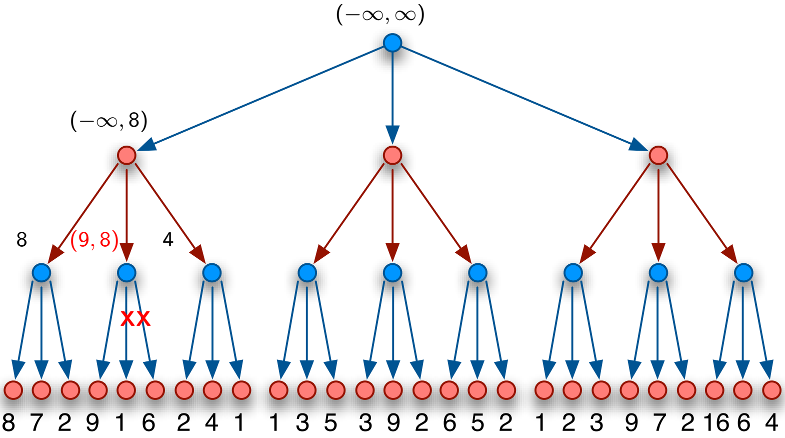
Alpha-Beta Pruning Example



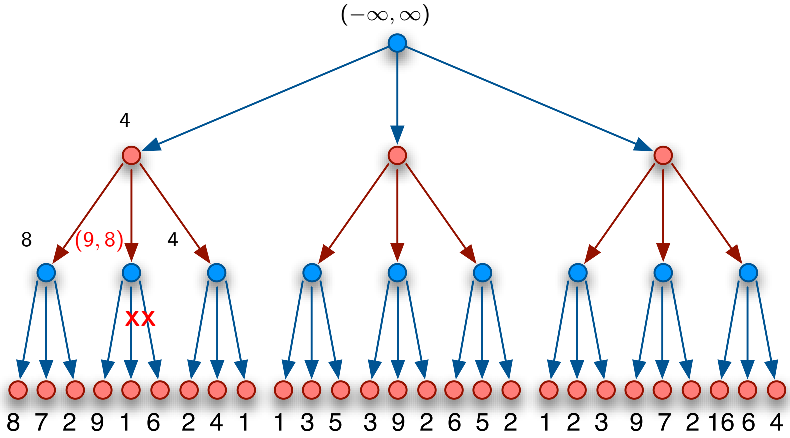
Alpha-Beta Pruning Example



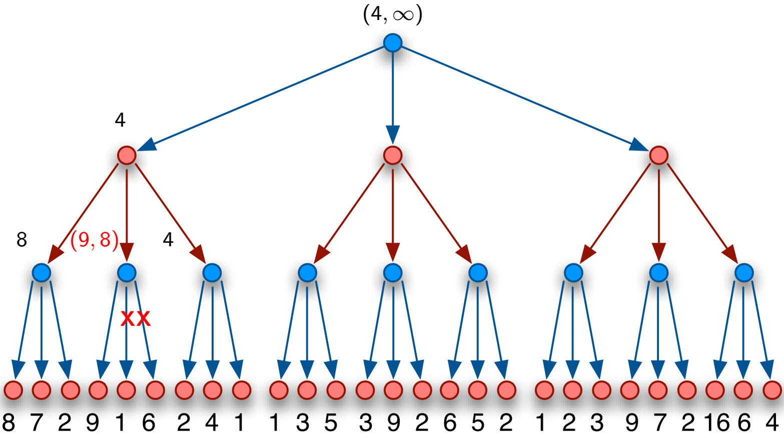
Alpha-Beta Pruning Example



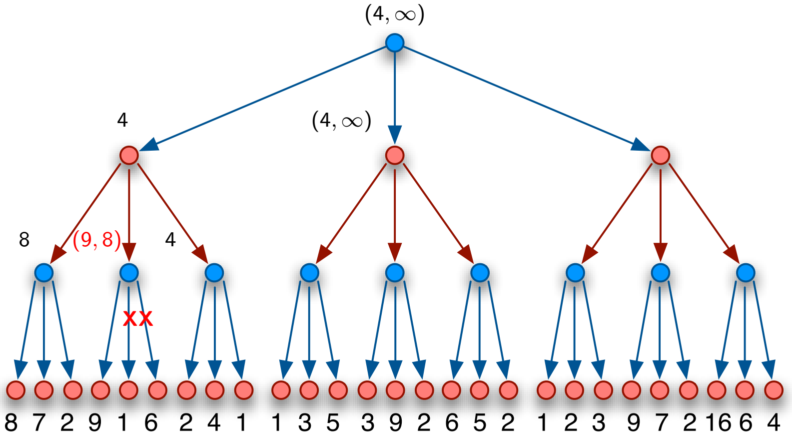
Alpha-Beta Pruning Example



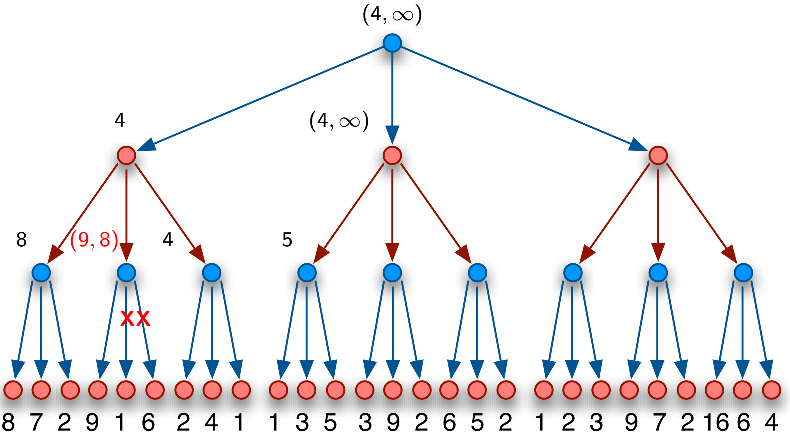
Alpha-Beta Pruning Example



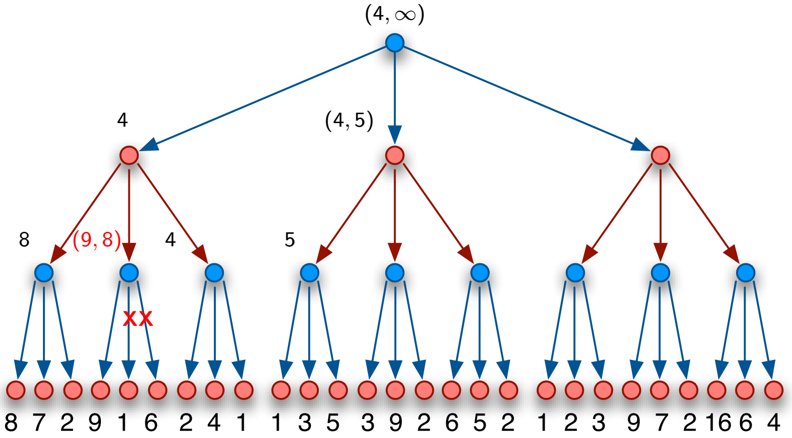
Alpha-Beta Pruning Example



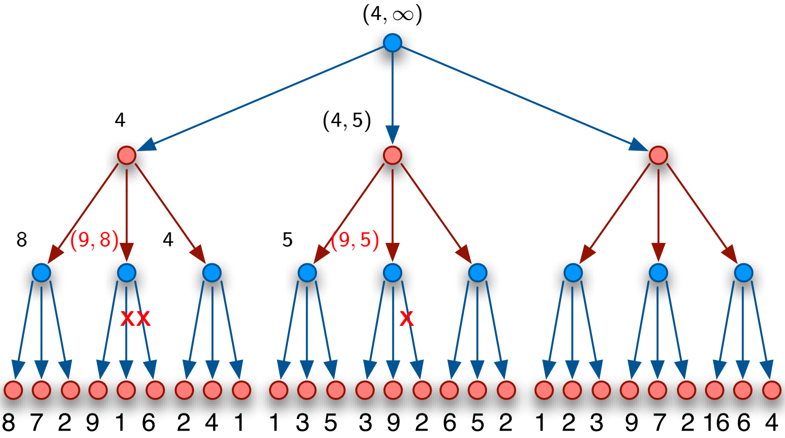
Alpha-Beta Pruning Example



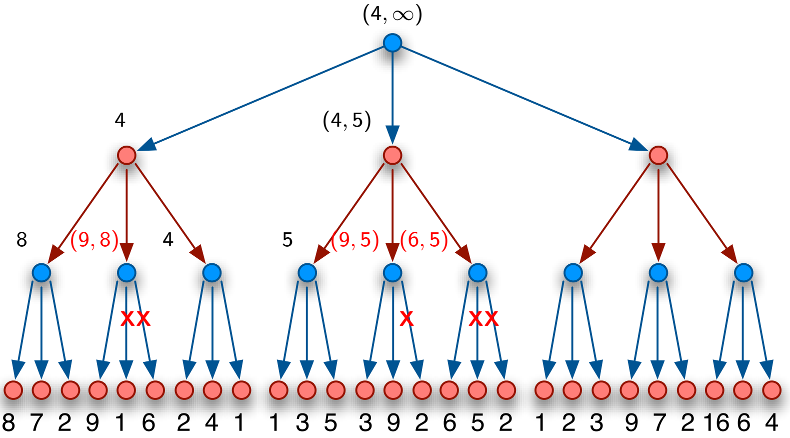
Alpha-Beta Pruning Example



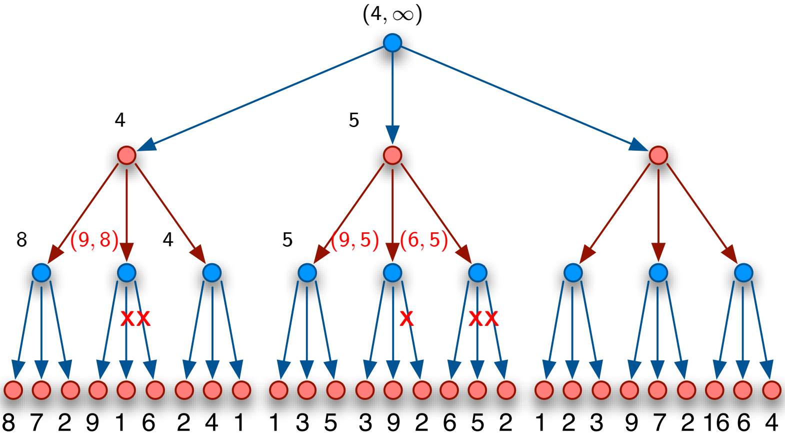
Alpha-Beta Pruning Example



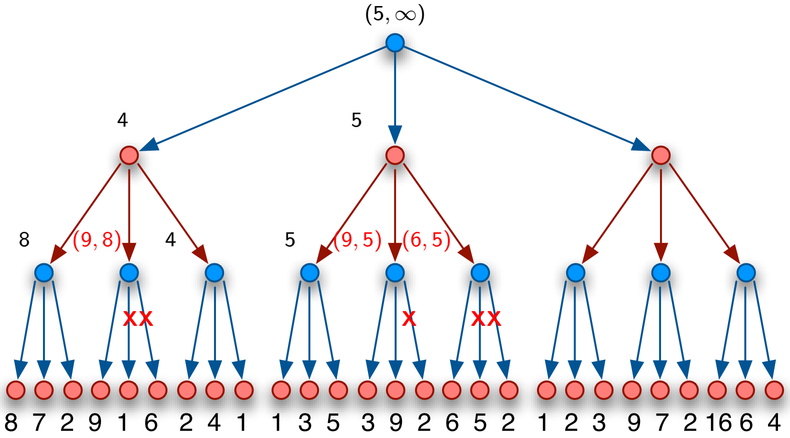
Alpha-Beta Pruning Example



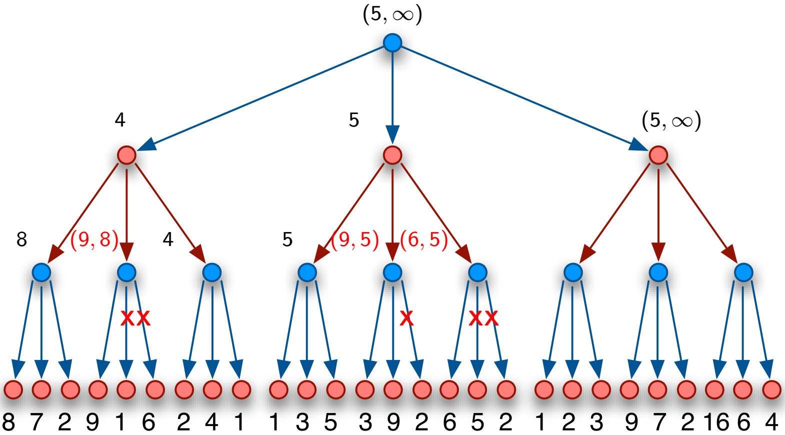
Alpha-Beta Pruning Example



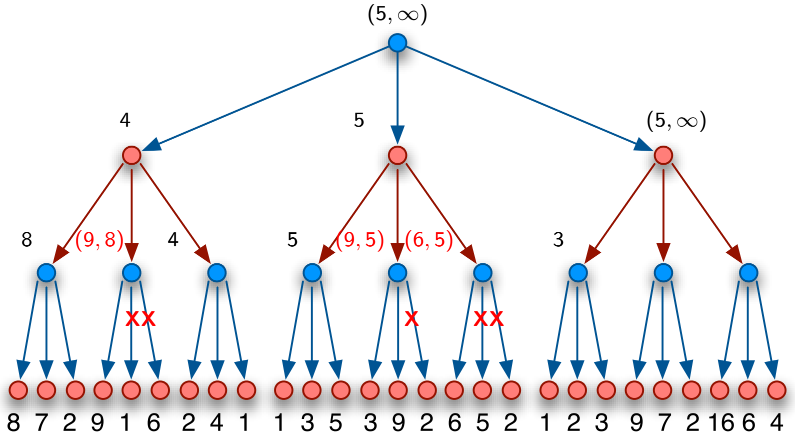
Alpha-Beta Pruning Example



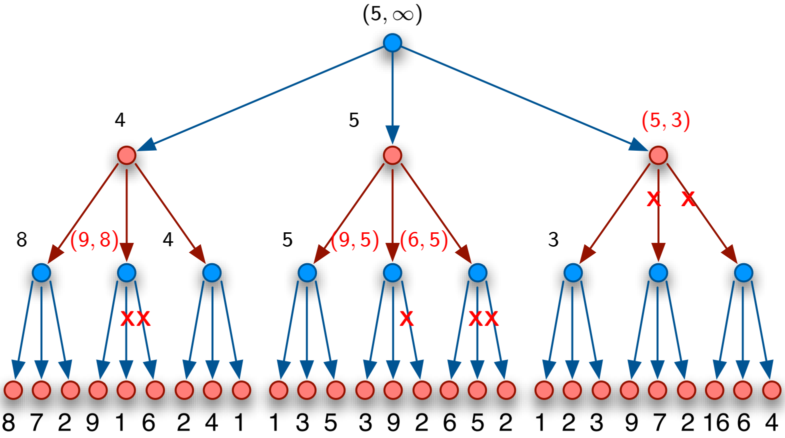
Alpha-Beta Pruning Example



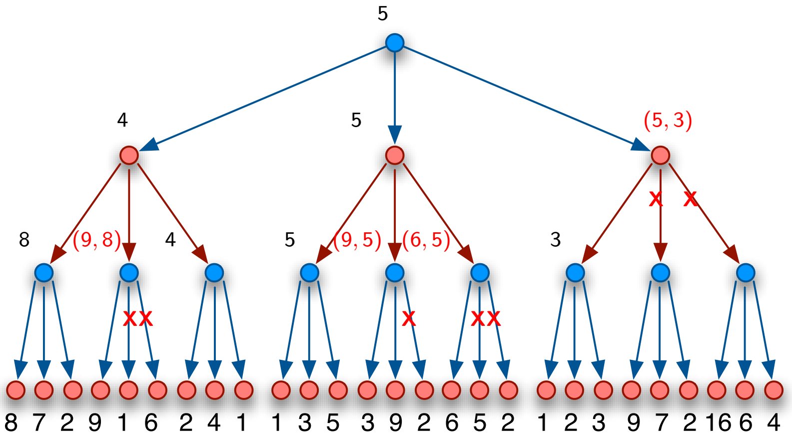
Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Alpha-Beta Pruning Example



Effectiveness of Alpha-Beta Pruning

- The performance of Alpha-Beta pruning depends strongly on the order in which the tree is searched.
- Ideally, one would want to examine the best successors first.
(Clearly, this is not achievable, since if we knew the best successors a priori, we would have solved the problem!)
- If this can be done, alpha-beta searches only need $O(b^{d/2})$ time
(compare with standard minmax, a depth-first search requiring $O(b^d)$ time).
- Effectively, this allows to double the search depth!
- State-of-the-art algorithms, such as NegaScout and MTD(f) are based on alpha-beta pruning, combined with null-window searches, which can yield quickly bounds on the value of the game.