Games – adversarial search

In which we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

April 8, 2010

Spis treści

Introduction to lecture

Competitive environments – goals in conflict

- In this part we cover competitive environments, in which the agents' **goals are in conflict**, giving rise to adversarial search problems often knows as games.
- We begin with a definition of the **optimal move** and an algorithm for finding it.
- We then look at techniques for choosing a good move when time is limited.
- We also discuss games that includes an element of chance.
- We also discuss games that includes an elements of imperfect information.

Introduction to lecture

Competitive environments – goals in conflict

- In this part we cover competitive environments, in which the agents' **goals are in conflict**, giving rise to adversarial search problems often knows as games.
- We begin with a definition of the **optimal move** and an algorithm for finding it.
- We then look at techniques for choosing a good move when time is limited.
- We also discuss games that includes an element of chance.
- We also discuss games that includes an elements of imperfect information.

Introduction to lecture

Competitive environments – goals in conflict

- In this part we cover competitive environments, in which the agents' **goals are in conflict**, giving rise to adversarial search problems often knows as games.
- We begin with a definition of the **optimal move** and an algorithm for finding it.
- We then look at techniques for choosing a good move when time is limited.
- We also discuss games that includes an element of chance.
- We also discuss games that includes an elements of imperfect information.

Introduction to lecture

Competitive environments – goals in conflict

- In this part we cover competitive environments, in which the agents' **goals are in conflict**, giving rise to adversarial search problems often knows as games.
- We begin with a definition of the **optimal move** and an algorithm for finding it.
- We then look at techniques for choosing a good move when time is limited.
- We also discuss games that includes an element of chance.
- We also discuss games that includes an elements of imperfect information.

Introduction to lecture

Competitive environments – goals in conflict

- In this part we cover competitive environments, in which the agents' **goals are in conflict**, giving rise to adversarial search problems often knows as games.
- We begin with a definition of the **optimal move** and an algorithm for finding it.
- We then look at techniques for choosing a good move when time is limited.
- We also discuss games that includes an element of chance.
- We also discuss games that includes an elements of imperfect information.

Introduction to lecture

Competitive environments – goals in conflict

- In this part we cover competitive environments, in which the agents' **goals are in conflict**, giving rise to adversarial search problems often knows as games.
- We begin with a definition of the **optimal move** and an algorithm for finding it.
- We then look at techniques for choosing a good move when time is limited.
- We also discuss games that includes an element of chance.
- We also discuss games that includes an elements of imperfect information.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum.
In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum.
In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum. In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum.
In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum.

In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum.
In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum. In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum.
In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum. In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

AI games

In AI, the most common games are of a rather specialized kind, what game theorist call deterministic, turn-taking, two-player, zero-sum. In our terminology, this means

- deterministic,
- fully observable,
- agents act alternately,
- utility values at the end of the game are always opposite.

Game in AI

Why games?

Games are interesting because they are to hard to solve.

Exmaple: chess

- An average branching factor of about 35.
- Games often go to 50 moves.
- So the search tree has about 35^{100} or 10^{154} nodes.
- Fortunately the search graph has „only" about 10^{40} distinc nodes.

Game in AI

Why games?

Games are interesting because they are to hard to solve.

Exmaple: chess

- An average branching factor of about 35.
- Games often go to 50 moves.
- So the search tree has about 35^{100} or 10^{154} nodes.
- Fortunately the search graph has „only” about 10^{40} distinc nodes.

Game in AI

Why games?

Games are interesting because they are to hard to solve.

Exmaple: chess

- An average branching factor of about 35.
- Games often go to 50 moves.
- So the search tree has about 35^{100} or 10^{154} nodes.
- Fortunately the search graph has „only" about 10^{40} distinc nodes.

Game in AI

Why games?

Games are interesting because they are to hard to solve.

Exmaple: chess

- An average branching factor of about 35.
- Games often go to 50 moves.
- So the search tree has about 35^{100} or 10^{154} nodes.
- Fortunately the search graph has „only” about 10^{40} distinc nodes.

Game in AI

Why games?

Games are interesting because they are to hard to solve.

Exmaple: chess

- An average branching factor of about 35.
- Games often go to 50 moves.
- So the search tree has about 35^{100} or 10^{154} nodes.
- Fortunately the search graph has „only” about 10^{40} distinc nodes.

Game in AI

Why games?

Games are interesting because they are to hard to solve.

Exmaple: chess

- An average branching factor of about 35.
- Games often go to 50 moves.
- So the search tree has about 35^{100} or 10^{154} nodes.
- Fortunately the search graph has „only" about 10^{40} distinc nodes.

Game in AI

Why games?

Games, like the real world, therefore require **the ability to make *some* decision even when calculating the *optimal* decision is infeasible.** Games also penalize inefficiency severely.

Game in AI

Definition

A game can be formally defined as a kind of search problem with the following elements

- The **initial state**, which specifies how the game is set up at the start.
- Function `PLAYER(s)`, which defines which player has the move in a state `s`.
- Function `ACTIONS(s)`, which returns the set of legal moves in a state.
- The **transition model**, `RESULT(s,a)`, which defines the result of a move.
- A **terminal test**, `TERMINAL-TEST(s)`, which is true when the game is over and false otherwise (equivalent of a goal test function).
- Function `UTILITY(s,p)`, also called an objective function or payoff function) which returns final numeric value for a game that ends in terminal state `s` for a player `p` (equivalent of a cost function).

Game in AI

Definition

A game can be formally defined as a kind of search problem with the following elements

- The **initial state**, which specifies how the game is set up at the start.
- Function `PLAYER(s)`, which defines which player has the move in a state `s`.
- Function `ACTIONS(s)`, which returns the set of legal moves in a state.
- The **transition model**, `RESULT(s,a)`, which defines the result of a move.
- A **terminal test**, `TERMINAL-TEST(s)`, which is true when the game is over and false otherwise (equivalent of a goal test function).
- Function `UTILITY(s,p)`, also called an objective function or payoff function) which returns final numeric value for a game that ends in terminal state `s` for a player `p` (equivalent of a cost function).

Definition

A game can be formally defined as a kind of search problem with the following elements

- The **initial state**, which specifies how the game is set up at the start.
- Function `PLAYER(s)`, which defines which player has the move in a state `s`.
- Function `ACTIONS(s)`, which returns the set of legal moves in a state.
- The **transition model**, `RESULT(s,a)`, which defines the result of a move.
- A **terminal test**, `TERMINAL-TEST(s)`, which is true when the game is over and false otherwise (equivalent of a goal test function).
- Function `UTILITY(s,p)`, also called an objective function or payoff function) which returns final numeric value for a game that ends in terminal state `s` for a player `p` (equivalent of a cost function).

Game in AI

Definition

A game can be formally defined as a kind of search problem with the following elements

- The **initial state**, which specifies how the game is set up at the start.
- Function `PLAYER(s)`, which defines which player has the move in a state `s`.
- Function `ACTIONS(s)`, which returns the set of legal moves in a state.
- The **transition model**, `RESULT(s,a)`, which defines the result of a move.
- A **terminal test**, `TERMINAL-TEST(s)`, which is true when the game is over and false otherwise (equivalent of a goal test function).
- Function `UTILITY(s,p)`, also called an objective function or payoff function) which returns final numeric value for a game that ends in terminal state `s` for a player `p` (equivalent of a cost function).

Game in AI

Definition

A game can be formally defined as a kind of search problem with the following elements

- The **initial state**, which specifies how the game is set up at the start.
- Function `PLAYER(s)`, which defines which player has the move in a state `s`.
- Function `ACTIONS(s)`, which returns the set of legal moves in a state.
- The **transition model**, `RESULT(s,a)`, which defines the result of a move.
- A **terminal test**, `TERMINAL-TEST(s)`, which is true when the game is over and false otherwise (equivalent of a goal test function).
- Function `UTILITY(s,p)`, also called an objective function or payoff function) which returns final numeric value for a game that ends in terminal state `s` for a player `p` (equivalent of a cost function).

Game in AI

Definition

A game can be formally defined as a kind of search problem with the following elements

- The **initial state**, which specifies how the game is set up at the start.
- Function `PLAYER(s)`, which defines which player has the move in a state `s`.
- Function `ACTIONS(s)`, which returns the set of legal moves in a state.
- The **transition model**, `RESULT(s,a)`, which defines the result of a move.
- A **terminal test**, `TERMINAL-TEST(s)`, which is true when the game is over and false otherwise (equivalent of a goal test function).
- Function `UTILITY(s,p)`, also called an objective function or payoff function) which returns final numeric value for a game that ends in terminal state `s` for a player `p` (equivalent of a cost function).

Game in AI

Definition

A game can be formally defined as a kind of search problem with the following elements

- The **initial state**, which specifies how the game is set up at the start.
- Function `PLAYER(s)`, which defines which player has the move in a state `s`.
- Function `ACTIONS(s)`, which returns the set of legal moves in a state.
- The **transition model**, `RESULT(s,a)`, which defines the result of a move.
- A **terminal test**, `TERMINAL-TEST(s)`, which is true when the game is over and false otherwise (equivalent of a goal test function).
- Function `UTILITY(s,p)`, also called an objective function or payoff function) which returns final numeric value for a game that ends in terminal state `s` for a player `p` (equivalent of a cost function).

Definition

The initial state, `ACTIONS` and `RESULTS` define the **game tree** for the game – a game where

- the nodes are game states and
- the edges are moves.

Example

Game tree for the game of tic-tac-toe.

Game in AI

Definition

The initial state, `ACTIONS` and `RESULTS` define the **game tree** for the game – a game where

- the nodes are game states and
- the edges are moves.

Example

Game tree for the game of tic-tac-toe.

Optimal decisions in games

What is an optimal solution in games?

- In a normal search problem, the optimal solution would be a **sequence of actions** leading to a goal state.
- In adversarial search, opponent has something to say about it.
- Therefore we must find a **contingent strategy**, which specifies our move in the initial state and then moves in the states resulting from every possible response by our opponent.

Optimal decisions in games

What is an optimal solution in games?

- In a normal search problem, the optimal solution would be a **sequence of actions** leading to a goal state.
- In adversarial search, opponent has something to say about it.
- Therefore we must find a **contingent strategy**, which specifies our move in the initial state and then moves in the states resulting from every possible response by our opponent.

Optimal decisions in games

What is an optimal solution in games?

- In a normal search problem, the optimal solution would be a **sequence of actions** leading to a goal state.
- In adversarial search, opponent has something to say about it.
- Therefore we must find a **contingent strategy**, which specifies our move in the initial state and then moves in the states resulting from every possible response by our opponent.

Optimal decisions in games

What is an optimal solution in games?

- In a normal search problem, the optimal solution would be a **sequence of actions** leading to a goal state.
- In adversarial search, opponent has something to say about it.
- Therefore we must find a **contingent strategy**, which specifies our move in the initial state and then moves in the states resulting from every possible response by our opponent.

Optimal decisions in games

How to find optimal strategy

- One ply tree.
- Expansion of the previous tree: two ply (one move deep).
- Example with two moves deep tree.

Optimal decisions in games

How to find optimal strategy

- One ply tree.
- Expansion of the previous tree: two ply (one move deep).
- Example with two moves deep tree.

Optimal decisions in games

How to find optimal strategy

- One ply tree.
- Expansion of the previous tree: two ply (one move deep).
- Example with two moves deep tree.

Optimal decisions in games

How to find optimal strategy

- One ply tree.
- Expansion of the previous tree: two ply (one move deep).
- Example with two moves deep tree.

Minimax algorithm

```
function MINIMAX(state) return an action
{
  v := MAX(state)
  return the action in ACTIONS(state) with value v
}

function MAX(state) return a utility value
{
  if(TERMINAL-TEST(state)) then
    return UTILITY(state)
  v := -infty
  for each a in ACTIONS(state) do
    v := max(v,MIN(RESULT(state,a)))
  return v
}

function MIN(state) return a utility value
{
  if(TERMINAL-TEST(state)) then
    return UTILITY(state)
  v := +infty
  for each a in ACTIONS(state) do
    v := min(v,MIN(RESULT(state,a)))
  return v
}
```

Optimal decisions in games

Optimal decision in multiplayer games

Let us examine how to extend the minimax idea to multiplayer games.

First, we need to replace the single value for each node with a vector of values. For terminal states, this vector gives the utility of the state from each player's viewpoint.

Example with two moves deep tree and two players.

Optimal decisions in games

Optimal decision in multiplayer games

Let us examine how to extend the minimax idea to multiplayer games.

First, we need to replace the single value for each node with a vector of values. For terminal states, this vector gives the utility of the state from each player's viewpoint.

Example with two moves deep tree and two players.

Optimal decisions in games

Optimal decision in multiplayer games

Let us examine how to extend the minimax idea to multiplayer games.

First, we need to replace the single value for each node with a vector of values. For terminal states, this vector gives the utility of the state from each player's viewpoint.

Example with two moves deep tree and two players.

Optimal decisions in games

Optimal decision in multiplayer games

Example with one move deep tree for three players.

Optimal decisions in games

Optimal decision in multiplayer games

The minimax algorithm performs a complete depth-first exploration of the game tree. This is a real problem, because the number of game states it has to examine is exponential in the depth of the tree. Unfortuantely, we can't eliminate the exponent, but it turns out we can effectively cut it in half. The trick is that it is possible to compute the correct minimax decision without looking at every node in the game tree. The technique we examine is called alpha-beta pruning or alpha-beta cutoff. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

Optimal decisions in games

alpha-beta cutoff – intuition

Intuition

Optimal decisions in games

alpha-beta cutoff – example

Example

Optimal decisions in games

Definition of alpha and beta parameter

- α the value of the best choice we have found so far at any choice point along the path for MAX (so, α is the highest value so far).
- β the value of the best choice we have found so far at any choice point along the path for MIN (so, β is the lowest value so far).

Optimal decisions in games

alpha-beta cutoff with alpha and beta parameter

Example with alpha and beta

Optimal decisions in games

alpha-beta cutoff and move ordering

The efectiveness of alpha-beta pruning is highly dependent on the order in which the states are examined.

Optimal decisions in games

alpha-beta cutoff and move ordering

The efectiveness of alpha-beta pruning is highly dependent on the order in which the states are examined.

Minimax with alpha-beta cutoff algorithm

```
function MINIMAX-ALPHA-BETA-CUTOFF(state) return an action
{
  v := MAX(state,-infty,+infty)
  return the action in ACTIONS(state) with value v
}

function MAX(state,alpha,beta) return a utility value
{
  if(TERMINAL-TEST(state)) then
    return UTILITY(state)
  v := -infty
  for each a in ACTIONS(state) do
    v := max(v,MIN(RESULT(state,a),alpha,beta))
    if (v >= beta) then
      return v
    alpha := maximum(alpha,v)
  return v
}

function MIN(state,alpha,beta) return a utility value
{
  if(TERMINAL-TEST(state)) then
    return UTILITY(state)
  v := +infty
  for each a in ACTIONS(state) do
    v := min(v,MIN(RESULT(state,a),alpha,beta))
    if (v <= alpha) then
      return v
    beta := minimum(beta,v)
  return v
}
```

Imperfect real-time decisions

Imperfect real-time decisions

- **Evaluation function.** This function returns an **estimate** of the expected utility of the game from a given position, just as the heuristic functions in previous part of this lecture.
- **Cutting off search** (depth limit). Replace `if(TERMINAL-TEST(state)) then return UTILITY(state)` with `if(CUTOFF-TEST(state,depth)) then return EVAL(state)`
- **Forward pruning** – some moves at a given node are pruned immediately without further consideration.
- **Search versus lookup.**

Imperfect real-time decisions

Imperfect real-time decisions

- **Evaluation function.** This function returns an **estimate** of the expected utility of the game from a given position, just as the heuristic functions in previous part of this lecture.
- **Cutting off search** (depth limit). Replace
  `if(TERMINAL-TEST(state)) then return UTILITY(state)`
  with
  `if(CUTOFF-TEST(state,depth)) then return EVAL(state)`
- **Forward pruning** – some moves at a given node are pruned immediately without further consideration.
- **Search versus lookup.**

Imperfect real-time decisions

Imperfect real-time decisions

- **Evaluation function.** This function returns an **estimate** of the expected utility of the game from a given position, just as the heuristic functions in previous part of this lecture.
- **Cutting off search** (depth limit). Replace `if(TERMINAL-TEST(state)) then return UTILITY(state)` with `if(CUTOFF-TEST(state,depth)) then return EVAL(state)`
- **Forward pruning** – some moves at a given node are pruned immediately without further consideration.
- **Search versus lookup.**

Imperfect real-time decisions

Imperfect real-time decisions

- **Evaluation function.** This function returns an **estimate** of the expected utility of the game from a given position, just as the heuristic functions in previous part of this lecture.
- **Cutting off search** (depth limit). Replace
  `if(TERMINAL-TEST(state)) then return UTILITY(state)`
  with
  `if(CUTOFF-TEST(state,depth)) then return EVAL(state)`
- **Forward pruning** – some moves at a given node are pruned immediately without further consideration.
- **Search versus lookup.**

Imperfect real-time decisions

Imperfect real-time decisions

- **Evaluation function.** This function returns an **estimate** of the expected utility of the game from a given position, just as the heuristic functions in previous part of this lecture.
- **Cutting off search** (depth limit). Replace
  `if(TERMINAL-TEST(state)) then return UTILITY(state)`
  with
  `if(CUTOFF-TEST(state,depth)) then return EVAL(state)`
- **Forward pruning** – some moves at a given node are pruned immediately without further consideration.
- **Search versus lookup.**

Stochastic games

Backgammon and chance nodes

Although one player (say white) knows his own legal moves are, he does not know what opponent is going to roll and thus does not know what opponent's legal moves will be. That means white cannot construct a standard game tree of the sort we saw in chess or tic-tac-toe. A game tree in backgammon must include **chance nodes** in addition to MAX and MIN nodes. Terminal nodes and MAX and MIN nodes work exactly the same way as befor (because the dice roll is known). For chance nodes, we compute the expected value, which is the sum of the value over all outcomes, weighted by the probability of each chance action. This leads to generalize minimax for deterministic games to an **expectiminimax** for game with chance node.

Partially observable games

Partially observable games – battleships

In deterministic partially observable games, uncertainty about the state of the board arises entirely from the lack of the access to the choices made by the opponent. This class includes games such as Battleships.

Partially observable games

Partially observable games – card games

Card games provide many examples of stochastic partial observability, where the missing information is generated randomly. For example, in many games, cards are dealt randomly at the beginning of the game, with each player receiving a hand that is not visible to the other players.
At first sight, it might seem that these card games are just like dice games: the cards are dealt randomly and determine the moves available to each player, but all the „dice" are rolled at the beginning.
Even though this analogy turns out to be incorrect, it suggests an effective algorithm: consider all possible deals of the invisible cards; solve each one as if it wear a fully observable game; and then choose the move that has the best outcome averaged over all the deals.

Partially observable games

Partially observable games – card games

Card games provide many examples of stochastic partial observability, where the missing information is generated randomly. For example, in many games, cards are dealt randomly at the beginning of the game, with each player receiving a hand that is not visible to the other players.
At first sight, it might seem that these card games are just like dice games: the cards are dealt randomly and determine the moves available to each player, but all the „dice" are rolled at the beginning.
Even though this analogy turns out to be incorrect, it suggests an effective algorithm: consider all possible deals of the invisible cards; solve each one as if it wear a fully observable game; and then choose the move that has the best outcome averaged over all the deals.

Partially observable games

Partially observable games – card games

Card games provide many examples of stochastic partial observability, where the missing information is generated randomly. For example, in many games, cards are dealt randomly at the beginning of the game, with each player receiving a hand that is not visible to the other players. At first sight, it might seem that these card games are just like dice games: the cards are dealt randomly and determine the moves available to each player, but all the „dice" are rolled at the beginning.
Even though this analogy turns out to be incorrect, it suggests an effective algorithm: consider all possible deals of the invisible cards; solve each one as if it wear a fully observable game; and then choose the move that has the best outcome averaged over all the deals.