Adversarial Search (I)

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University
Outline

- Introduction to Games
- The Minimax Algorithm
- Alpha-beta Pruning
- Imperfect, Real-time Decisions
- Games including Chance
- State-of-the-art Game Programs
- Discussion
- Summary
Games

- Here we will talk about games that are
  - deterministic,
  - 2-player and turn-taking,
  - zero-sum, and
  - with perfect information

- Except robot soccer, physical games have not attracted much interest in AI community.
  - They are too complicated and imprecise.
Games are interesting because they are very hard and e.g. The search tree has size $35^{100} \times (10^{154})$ when the game goes to 50 moves by each player in the chess game. penalize inefficiency severely. A chess program that is half as efficient probably will be beaten to the ground.
Games

- We can define a game as a search problem.
  - **Initial state** (board position, first player, etc.)
  - **Successor function** (legal moves and resulting states)
  - **Terminal test** (when the game is over)
  - **Utility function** (win, draw, lose, etc.)

- The initial state and the legal moves for each side define the game tree.
Games

“Adversarial Search,” Artificial Intelligence, Spring, 2010
Optimal Decisions in Games

- Different from the search problems that have been mentioned, there is an opponent.

- We say a strategy is optimal if it leads to outcomes at least as good as any other strategy when one is playing with an "infallible" opponent.
  - The definition of optimal play maximizes the worst-case outcome.
Optimal Decisions in Games

$\text{MinimaxValue}(n) =$

$$\begin{align*}
\text{Utility } (n) & \quad \text{if } n \text{ is a terminal state} \\
\max_{s \in \text{Successors}(n)} \text{MinimaxValue}(s) & \quad \text{if } n \text{ is a MAX node} \\
\min_{s \in \text{Successors}(n)} \text{MinimaxValue}(s) & \quad \text{if } n \text{ is a MIN node}
\end{align*}$$
Optimal Decisions in Games

What action will you (MAX) take? 😊

```
1 2 3 2
1 8 9 7 8 2 2 3 2 3
A1 A2 A3
```

```
MAX
```

```
MIN
1 2 3 2 3
```

“Adversarial Search,” Artificial Intelligence, Spring, 2010
The Minimax Algorithm

function **MINIMAX-DECISION**(state) returns an action

\[ v \leftarrow \text{MAX-VALUE}(\text{state}) \]

return the action in SUCCESSORS(state) with value \( v \)

Assume I am MAX.

function **MAX-VALUE**(state) returns a utility value

\[ \text{if TERMINAL-TEST}(\text{state}) \text{ then return UTILITY}(\text{state}) \]

\[ v \leftarrow -\infty \]

\[ \text{for } a, s \text{ in SUCCESSORS}(\text{state}) \text{ do} \]

\[ v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s)) \]

return \( v \)

function **MIN-VALUE**(state) returns a utility value

\[ \text{if TERMINAL-TEST}(\text{state}) \text{ then return UTILITY}(\text{state}) \]

\[ v \leftarrow \infty \]

\[ \text{for } a, s \text{ in SUCCESSORS}(\text{state}) \text{ do} \]

\[ v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s)) \]

return \( v \)
The Minimax Algorithm

- It uses a simple recursive computation.
- It performs a complete depth-first exploration of the game tree.

**Complexity**
- Time: $O(b^m)$
- Space $O(bm)$ or $O(m)$
The Minimax Algorithm

- What about multi-player games?
  - We can replace the single value for each node with a vector of values.
  - The backed-up value of a node $n$ is the utility vector of whichever successor has the highest value for the player choosing at $n$. 
The Minimax Algorithm

Example: a 3-player game tree
Alpha-Beta Pruning

- The problem with *minimax* is the huge number of nodes to be examined. (exponential in the number of moves)
- *Alpha-beta pruning* returns the same move as minimax would, but prunes away branches that cannot possibly influence the decision.
Alpha-Beta Pruning

Range of possible values

MAX

MIN

\([-\infty, +\infty]\)

\([-\infty, +\infty]\)

\([-\infty, 3]\)

\([-\infty, 3]\)

\(\leq 3\)
Alpha-Beta Pruning

\[ \text{MAX} \]
\[ \text{MIN} \]
\[ [-\infty,3] \]
\[ \leq 3 \]
\[ 3 \]
\[ 12 \]
\[ 8 \]

\[ [3,\infty] \]
\[ \geq 3 \]

Artificial Intelligence: A Modern Approach, 2nd ed., Figure 6.5

“Adversarial Search,” Artificial Intelligence, Spring, 2010
Alpha-Beta Pruning

This node is worse for MAX
Alpha-Beta Pruning

Artificial Intelligence: A Modern Approach, 2nd ed., Figure 6.5

“Adversarial Search,” Artificial Intelligence, Spring, 2010
Alpha-Beta Pruning
**Alpha-Beta Pruning**

\[ \text{MINIMAX \_VALUE}(\text{root}) \]
\[ = \max(\min(3,12,8), \min(2, x, y), \min(14,5,2)) \]
\[ = \max(3, \min(2, x, y), 2) \]
\[ = \max(3, z, 2) \text{ where } z \leq 2 \]
\[ = 3. \]
Alpha-Beta Pruning

- General principle
  - Consider a node $n$ to which Player has a choice of moving.

    If Player has a better choice $m$ at any point further up, then $n$ will never be reached in actual play.

  - Once we have found out enough about $n$ to reach the above conclusion, we can prune it.
**Alpha-Beta Pruning**

```plaintext
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
    α, the value of the best alternative for MAX along the path to state
    β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
    α, the value of the best alternative for MAX along the path to state
    β, the value of the best alternative
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
    return v
```

“Adversarial Search,” Artificial Intelligence, Spring 2010. Artificial Intelligence: A Modern Approach, 2nd ed., Figure 6.7
Alpha-Beta Pruning

- Demo on alpha-beta pruning
  - http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html
  - http://www.youtube.com/watch?v=ipO2FWQlGUc
Exercise

Apply the minimax algorithm & alpha-beta pruning
Adversarial Search (II)

Instructor: Tsung-Che Chiang
tcchiang@ieee.org
Department of Computer Science and Information Engineering
National Taiwan Normal University
Alpha-Beta Pruning

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

If this node is generated first, we can prune the other two.
It might be worthwhile to examine first the successors that are likely to be best.

In the best case (in fact, impossible), alpha-beta needs to examine only $O(b^{m/2})$ nodes instead of $O(b^m)$ for minimax.

If successors are examined in random order, the total number of nodes examined will be roughly $O(b^{3m/4})$ for moderate $b$. 
Alpha-Beta Pruning

Exercise: perfect ordering
Alpha-Beta Pruning

A simple explanation of $O(b^{m/2})$ complexity

**Fig. 7.** Alpha and beta cutoffs in a ternary tree which is perfectly ordered. "*" means cutoff.

“Adversarial Search,” Artificial Intelligence, Spring, 2010

Alpha-Beta Pruning

- For chess,
  - A fairly simple order function (such trying captures first, then threats, then forward, ...) gets you to within about a factor of 2 of $O(b^{m/2})$ result.
  
  - Adding dynamic move-ordering schemes (such as trying first the best moves at last time) brings us quite close to the theoretical limit.
Alpha-Beta Pruning

- In games, repeated states occur frequently because of different permutations of the move sequence that end up in the same position (transpositions).

- It is worthwhile to store the evaluation of positions in a hash table (transposition table).
  - There could be a dramatic effect, sometimes as much as doubling the reachable search depth in chess.
  - There are various strategies for choosing valuable states to store.
Imperfect, Real-time Decisions

One problem of alpha-beta is that it still has to search to terminal states. ⇒ The depth is usually not practical.

We should cut off the search earlier and apply a heuristic evaluation function.

- terminal test → cut-off test
- utility function → heuristic evaluation function

```
function MAX-VALUE(state, α, β) returns a utility value
  inputs: state, current state in game
  α, the value of the best alternative for MAX along the path to state
  β, the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
```
Imperfect, Real-time Decisions

- An **evaluation functions** returns an **estimate** of the expected utility of the game from a given position.

- The performance of a game-playing program is dependent on the quality of its evaluation function.
Imperfect, Real-time Decisions

- How exactly do we design good evaluation functions?
  - It should order the terminal states in the same way as the true utility function.
  - It must not take too long time.
  - For non-terminal states, it should be strongly correlated with the actual chances of winning.
Imperfect, Real-time Decisions

- Most evaluation functions work by calculating various **features** of the state.
  - e.g. number of pawns possessed by each side

- The features define various categories of states.

- The evaluation function cannot know exactly which state will lead to a win.

  But it can return a value that reflects the proportion of states with each outcome.
Imperfect, Real-time Decisions

Example:

72% win 20% loss 8% draw
\[
0.72 \times 1 + 0.2 \times (-1) = 0.52
\]

The evaluation function need not return actual expected value, as long as the ordering of the states is the same.
Imperfect, Real-time Decisions

- The above method requires too many categories and hence too much experience to estimate all the probabilities of winning.
- Another common method is to compute separate numerical contributions from each feature and then sum them.

\[ Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s). \]
Imperfect, Real-time Decisions

- Adding up the values of features involves a very strong assumption: the contribution of each feature is independent.
  - Bishops are more powerful in the endgame, when they have much space to maneuver.

- Current programs also use nonlinear combinations.
  - e.g. A pair of bishops might be worth slightly more than twice the value of a single bishop.
Imperfect, Real-time Decisions

**Cutting off search**

- The most straightforward approach is set a fixed depth limit.
- A more robust approach is to use iterative deepening.

However, they can lead to errors without looking at the (near) future.
Imperfect, Real-time Decisions

- Two slightly different chess positions with very different results

Chess pieces:
- King
- Queen
- Rook
- Bishop
- Knight
- Pawn

“Adversarial Search,” Artificial Intelligence, Spring, 2010

Artificial Intelligence: A Modern Approach, 2nd ed., Figure 6.8
Symbols from Wikipedia (http://en.wikipedia.org/wiki/Chess)
Imperfect, Real-time Decisions

- The evaluation function should be applied only to positions that are quiescent - unlikely to exhibit wild swings in value in the near future.

- Quiescence search is to expand nonquiescent positions to quiescent ones.
  - Sometimes it considers only certain types of moves, such as capture moves, that will quickly resolve the uncertainties.
Imperfect, Real-time Decisions

- The horizon effect is more difficult to eliminate.
- It arises when facing an unavoidable serious-damage move by the opponent.

Example:

Black can forestall the queening move for 14 ply by checking White with the rook, but inevitably the pawn will become a queen.

The stalling moves push the inevitable queening move “over the search horizon” to a place where it cannot be detected.
Imperfect, Real-time Decisions

Another example (http://en.wikipedia.org/wiki/Horizon_effect)

- Assume a situation where black is searching the game tree to six plies depth and see that it is going to lose queen.
- Also, suppose there is another combination of moves where by sacrificing a rook, the loss of the queen is pushed to the eighth ply.
- Since the loss of the queen was pushed over the horizon of search, sacrificing of the rook seems to be better than losing the queen, so the sacrificing move is returned as the best option.
Imperfect, Real-time Decisions

- The use of singular extensions has been quite effective in avoiding the horizon effect.
  - A singular extension is a move that is “clearly better” than all other moves in a given position.
  - A singular extension search can go beyond the normal depth limit without much cost because its branching factor is 1.
  - Quiescence search can be viewed as a variant.
Games including Chance

- In real life, there are many unpredictable external events.
- Many games mirror this by including a random element, such as the throwing of dice.
- Backgammon is a typical example.

http://aimm02.cse.ttu.edu.tw/rl/
Games including Chance

White has rolled 6-5.

Four legal moves:

\((5 \rightarrow 10, 5 \rightarrow 11)\)
\((5 \rightarrow 11, 19 \rightarrow 24)\)
\((5 \rightarrow 10, 10 \rightarrow 16)\)
\((5 \rightarrow 11, 11 \rightarrow 16)\)
Games including Chance

- A game tree in backgammon must include chance nodes in addition to MAX and MIN nodes.
Games including Chance

- We can only calculate the expected value. The minimax value is generalized to the expectiminimax value:

\[
\begin{align*}
\text{Utility}(n) & \quad \text{if } n \text{ is a terminal state} \\
\max_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \quad \text{if } n \text{ is a MAX node} \\
\min_{s \in \text{Successors}(n)} \text{EXPECTIMINIMAX}(s) & \quad \text{if } n \text{ is a MIN node} \\
\sum_{s \in \text{Successors}(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) & \quad \text{if } n \text{ is a chance node}
\end{align*}
\]

where \( P(s) \) is the probability that the dice roll occurs.
Games including Chance

- Applying the cut-off and heuristic evaluation function is more difficult.

![Game Tree Diagram](Image)

“Adversarial Search,” Artificial Intelligence, Spring, 2010

Artificial Intelligence: A Modern Approach, 2nd ed., Figure 6.12
Games including Chance

- The program behaves totally different if we make a change in the scale of some evaluation values.

- To avoid this sensitivity, the evaluation function must be a positive linear transformation of the probability of winning from a position.
Games including Chance

- Considering the chance node, the complexity becomes $O(b^mn^m)$, where $n$ is the number of distinct rolls.
  - The extra cost is high. For example, in backgammon, $n$ is 21 and $b$ is usually round 20. (The value of $b$ can be up to 4,000 when the player rolls doubles.)
  - Three plies is probably all we could manage.
Games including Chance

- The advantage of alpha-beta pruning is that it ignores future that is not going to happen and concentrates on likely sequences.
Games including Chance

- In games with dice, there are no likely sequences of moves, because for those moves to take place, the dice would first have to come out to make them legal.

Can we prune the dashed move?
Games including Chance

- We can still do something like alpha-beta pruning.
  - If we put bounds on the possible values of the utility function, we can place an upper bound on the value of a chance node without looking at all its children.
  - The analysis for MIN and MAX nodes is unchanged.
Games including Chance

Suppose the value of the terminal states is in the interval $[0, 10]$, which moves can we prune?

```
\begin{center}
\begin{tikzpicture}
\node (MAX) at (0,0) {MAX};
\node[fill=gray!30] (a) at (-1,1) {8} edge from parent node[draw=none]{0.5};
\node[fill=gray!30] (b) at (1,1) {8} edge from parent node[draw=none]{0.5};
\node (c) at (0,2) {MIN};
\node[fill=gray!30] (d) at (-1,3) {8} edge from parent node[draw=none]{0.3};
\node[fill=gray!30] (e) at (-1,4) {8} edge from parent node[draw=none]{0.7};
\node[fill=gray!30] (f) at (1,3) {8} edge from parent node[draw=none]{0.1};
\node[fill=gray!30] (g) at (1,4) {8} edge from parent node[draw=none]{0.9};
\node (a1) at (-2,5) {8} edge from parent node[draw=none]{8};
\node (a2) at (-1.5,5) {8} edge from parent node[draw=none]{10};
\node (b1) at (-1,5) {8} edge from parent node[draw=none]{6};
\node (b2) at (-0.5,5) {8} edge from parent node[draw=none]{5};
\node (c1) at (0,5) {8} edge from parent node[draw=none]{10};
\node (c2) at (0.5,5) {8} edge from parent node[draw=none]{3};
\node (d1) at (1,5) {8} edge from parent node[draw=none]{6};
\node (d2) at (1.5,5) {8} edge from parent node[draw=none]{2};
\node (a) at (-2,5) {8} edge from parent node[draw=none]{a};
\node (b) at (-1.5,5) {8} edge from parent node[draw=none]{b};
\node (c) at (-1,5) {8} edge from parent node[draw=none]{c};
\node (d) at (0,5) {8} edge from parent node[draw=none]{d};
\node (e) at (0.5,5) {8} edge from parent node[draw=none]{e};
\node (f) at (1,5) {8} edge from parent node[draw=none]{f};
\node (g) at (1.5,5) {8} edge from parent node[draw=none]{g};
\node (h) at (2,5) {8} edge from parent node[draw=none]{h};
\end{tikzpicture}
\end{center}
```
Card Games

- In many card games, each player receives a hand of cards that is not visible to the other players at the beginning of the game.
  - e.g. bridge, whist, heart, and some forms of poker
Card Games

- It **might seem** that these card games are **just like** dice games with all the dice being **rolled at the beginning**: the cards are dealt randomly and determine the moves available to each player.

⇒ It is not true.
Card Games

An example: 4-card two handed bridge
Assume all cards are visible.

MAX

MIN

Suppose MAX leads the ♣9. MIN must play the ♣10.

Then, MIN leads the ♠2. MAX must play the ♥6.

Then MAX wins the remaining two tricks.

⇒ Draw game

(Actually, we can show that lead of the ♣9 is an optimal choice.)
An example: 4-card two handed bridge
Assume all cards are visible.

Suppose MAX leads the ♣9. MIN must play the ♣10.

Then, MIN leads the ♠2. MAX must play the ♥6.

Then MAX wins the remaining two tricks.

⇒ Draw game

(Again, we can show that lead of the ♣9 is an optimal choice.)
Card Games

An example: 4-card two handed bridge
Assume one card is invisible. But we know that it is either ♥4 or ♦4.

MAX: ♥6 ♦6 ♣9 ♣8
MIN: ? ♠2 ♣10 ♣5

MAX’s reasoning
The ♣9 is an optimal choice against MIN’s first and second hands, so it must be optimal now because I know that MIN has one of the two hands.

Is it reasonable?
An example: 4-card two handed bridge
Assume one card is invisible. But we know that it is either ♥4 or ♦4.

Max: ♥6 ♦6 ♣9 ♦8

MIN: ? ♠2 ♣10 ♠5

Suppose MAX leads the ♣9. MIN must play the ♣10.

Then, MIN leads the ♠2. …???

Which card should MAX play?
♥6? MIN might have ♥4.
♦6? MIN might have ♦4.
Card Games

The problem with MAX's algorithm is that it assumes that in each possible deal, play will proceed as if all the cards are visible.
Card Games

- In games such as bridge, it is often a good idea to play a card that will help one discover things about opponents’ or partner’s cards.
  - Such an algorithm searches in the space of belief states.
- In games of imperfect information, it’s best to give away as little information to the opponent as possible.
  - Often the best way is to act unpredictably.
State-of-the-Art

Chess:


- Deep Blue is a parallel computer with
  - 30 IBM RS/6000 processors for software search and
  - 480 VLSI chess processors for hardware search

- 126 ~ 330 million nodes per second
- Up to 30 billion positions per move, reaching depth 14 routinely
State-of-the-Art

Chess:

- Standard iterative-deepening alpha-beta search with a transposition table
- Ability to generate extensions up to 40 plies
- Over 8000 features in the evaluation function
- A database of 700,000 grandmaster games
- A large endgame database of solved positions (5~6 pieces)

Fritz vs. V. Kramnik

- 2002 - 4:4
- 2006 - 2 wins, 4 draws
State-of-the-Art

Checkers: http://www.youtube.com/watch?v=QVlm2tuf_kg

- 1952, Arthur Samuel of IBM developed a program that learned its own evaluation function by playing itself thousands of times.
  - It defeated a human champion in 1962.
- Chinook (by J. Schaeffer) came in second in 1990.
  - regular PC, alpha-beta, a database of 444 billion positions with 2~8 pieces.
  - Chinook became the official world champion in 1994.
  - Schaeffer believes that with enough computing power, checkers would be completely solved.
State-of-the-Art

- **Othello (Reversi):**
  - It has a smaller search space than chess, usu. 5 to 15 legal moves.
  - In 1997, the Logistello program defeated the human world champion by six games to none.
  - It is generally acknowledged that humans are no match for computers at Othello.
State-of-the-Art

- Backgammon:
  - Most work has gone into improving the evaluation function.
  
  - G. Tesauro combined reinforcement learning with neural network to develop the evaluation function that is used with a search to depth 2 or 3.
  
  - Tesauro’s program (TD-GAMMON) is reliably ranked among the top 3 players in the world.
    - More than a million training games against itself
    - The program’s opinion of the opening moves have in some cases radically altered the received wisdom.
State-of-the-Art

Go:

- The branching factor starts at 361 (19\(\times\)19), which is too daunting for regular search methods.
- Most of the best programs combine pattern recognition with limited search.
- Success may come from integrating local reasoning about many loosely connected subgames.
- Go is an area that is likely to benefit from intensive investigation using more sophisticated reasoning methods.
State-of-the-Art

2009.2.10

人腦輸電腦 周俊勳不敵魔圍棋

記者黃德芬／南市報導

「2009歐洲魔圍棋挑戰台灣職業棋士邀請賽」10日在台南大學開幕，首日由國民紅面棋士周俊勳再次與魔圍棋 MOGO進行電腦與人腦的比賽，結果下午第1場 19路比賽時，周俊勳竟敗給 MOGO，讓人跌破眼鏡，也創下電腦贏過9段職業棋士的世界紀録，立即在網絡上引起全球圍棋迷的熱烈討論。

面對這項結果，周俊勳坦言，原本他認爲雖然讓7子，但應該能夠輕鬆獲勝，沒想到 MOGO竟然下出業餘高段的水準，甚至有職業水準，輕易獲得勝利，他確實有些錯愕及難過，對於13日的比賽也就更為期待，他會調整布局，全力應戰。

周俊勳去年曾在「2008人工智慧型計算論壇暨全球9路電腦圍棋賽」中首次與 MOGO交鋒，比賽結果仍是人腦略勝一籌，在 2場 9路、1場 19路比賽中，周俊勳 3戰全勝。

之後 MOGO研發團隊以去年比賽獲得的大量分析數據進行程式改良及加強，果然「魔」力大增，昨天上午 2場 9路比賽，在第 1場即讓周俊勳備感壓力，在第 10手棋出乎意料下得非常好，讓周俊勳足足思考 10分鐘之久，陷入苦戰，所幸最後仍 2戰皆勝；下午進行 2場 19路比賽，周俊勳各讓 7子，結果魔圍棋、棋王各勝 1場，令人意外。


“Adversarial Search,” Artificial Intelligence, Spring, 2010
State-of-the-Art
State-of-the-Art

Bridge:

- Optimal play can include elements of
  - information-gathering,
  - communication,
  - bluffing, and
  - careful weighing of probabilities.

- The GIB program (Ginsberg, 1999) was ranked at the 12th place in a field of 35 in 1998.
- Jack is the six times World Champion Computer Bridge. See [http://www.jackbridge.com/eindex.htm](http://www.jackbridge.com/eindex.htm).
State-of-the-Art

Prof. Shun-Shii Lin’s Achievement

- The 2nd prize in TAAI 19×19 Computer Go Competition, 2009
- The 4th prize in World 9×9 Computer Go Championship, 2008
- The 4th prize of Chinese Chess Tournament in Computer Olympiad, 2007
- The 3rd prize of Chinese Chess Tournament in Computer Olympiad, 2006
Monte-Carlo Go (MoGo)

- It was developed by INRIA in France.
  - Since August 2006 it has been consistently ranked no. 1 on the Computer Go server (http://cgos.boardspace.net/).

- Strategies
  - evaluating the positions using Monte-Carlo methods
  - exploration-exploitation in the search tree using a UCT algorithm
    - asymmetric growth of the tree
    - efficient imprecision management
    - any time

http://www.inria.fr/saclay/resources/computer-culture/mogo-champion-program-for-go-games
Monte-Carlo Go (MoGo)

- **K**-armed bandit problem
  - **K** gambling machines
  - $X_{i,n}$ is the reward obtained by playing the $i^{th}$ machine at the $n^{th}$ time
  - $X_{i,1}, X_{i,2}, \ldots$ are i.i.d. with a certain but unknown expectation $\mu_i$.
  - $X_{i,s}$ and $X_{j,t}$ are also independent.
  - A policy determines the next machine to play based on the sequence of past plays and obtained rewards.
Monte-Carlo Go (MoGo)

- K-armed bandit problem
- Regret

\[
\mu^* n - \sum_{j=1}^{K} \mu_j E[T_j(n)] \quad \text{where} \quad \mu^* = \max_{1 \leq i \leq K} \mu_i
\]

- \(n\) is the number of plays
- \(T_j(n)\) is the number of times machine \(i\) has been played after the first \(n\) plays.

Monte-Carlo Go (MoGo)

- **K-armed bandit problem**
- Under the policies satisfying

\[
\mathbb{E}[T_j(n)] \leq \left( \frac{1}{D(p_j \| p^*)} + o(1) \right) \ln n
\]

\[
D(p_j \| p^*) \overset{\text{def}}{=} \int p_j \ln \frac{p_j}{p^*}
\]

, the optimal machines is played exponentially more often than any other machine. This regret is the best possible. (Lai and Robbins 1985)

Monte-Carlo Go (MoGo)

- **UCB1 algorithm** *(Auer et al., 2002)*
  - It ensures the optimal machine is played exponentially more often than any other machines.

**Deterministic policy:** ucb1.

**Initialization:** Play each machine once.

**Loop:**

- Play machine $j$ that maximizes $\bar{x}_j + \sqrt{\frac{2 \ln n}{n_j}}$, where $\bar{x}_j$ is the average reward obtained from machine $j$, $n_j$ is the number of times machine $j$ has been played so far, and $n$ is the overall number of plays done so far.

Monte-Carlo Go (MoGo)

- **UCT**: UCB1 for tree search (Kocsis et al., 2006)
  - UCT is the extension of UCB1 to minimax tree search.
  - The idea is to consider each node as an independent bandit, with its child-nodes as independent arms.
  - It plays sequences of bandits within limited time.

Monte-Carlo Go (MoGo)

- **UCT**: UCB1 for tree search (Kocsis et al., 2006)

Monte-Carlo Go (MoGo)

- **UCT**: UCB1 for tree search (Kocsis et al., 2006)
  - UCT vs. alpha-beta search

(1) UCT works in an anytime manner.
(2) UCT handles uncertainty in a smooth way.
(3) UCT explores more deeply the good moves.
Monte-Carlo Go (MoGo)

- **UCT**: UCB1 for tree search (Kocsis et al., 2006)

```
1: function playOneSequence(rootNode);
2:   node[0] := rootNode; i = 0;
3:   while(node[i] is not leaf) do
4:     node[i+1] := descendByUCB1(node[i]);
5:     i := i + 1;
6:   end while ;
7:   updateValue(node, -node[i].value);
8:   end function;
```
Monte-Carlo Go (MoGo)

**UCT**: UCB1 for tree search (Kocsis et al., 2006)

```plaintext
9: function descendByUCB1(node)
10:     nb := 0;
11:     for i := 0 to node.childNode.size() - 1 do
12:         nb := nb + node.childNode[i].nb;
13:     end for;
14:     for i := 0 to node.childNode.size() - 1 do
15:         if node.childNode[i].nb = 0
16:             do v[i] := ∞;
17:         else v[i] := -node.childNode[i].value /
18:                 node.childNode[i].nb
19:                 +sqrt(2*log(nb)/(node.childNode[i].nb))
20:             end if;
21:     end for;
22:     index := argmax(v[j]);
23:     return node.childNode[index];
24: end function;
```

Monte-Carlo Go (MoGo)

- **UCT**: UCB1 for tree search (Kocsis et al., 2006)

```plaintext
22: function updateValue(node, value)
23:     for i := node.size()-2 to 0 do
24:         node[i].value := node[i].value + value;
25:         node[i].nb := node[i].nb + 1;
26:         value := -value;
27:     end for;
28: end function;
```
Monte-Carlo Go (MoGo)

MoGo: UCT for Computer-Go (Gelly et al., 2006)
- Each node of the search tree is a Go board situation.

Hypothesis:
- Each Go board situation is a bandit problem.
- Each legal move is an arm with unknown reward but of a certain distribution.
Monte-Carlo Go (MoGo)

MoGo: UCT for Computer-Go (Gelly et al., 2006)

```
1: function playOneSequenceInMoGo(rootNode)
2:     node[0] := rootNode; i := 0;
3:     do
4:         node[i+1] := descendByUCB1(node[i]); i := i + 1;
5:         while node[i] is not first visited;
6:         createNode(node[i]);
7:         node[i].value := getValueByMC(node[i]);
8:         updateValue(node,-node[i].value);
9:     end function;
```


“Adversarial Search,” Artificial Intelligence, Spring, 2010
Monte-Carlo Go (MoGo)

MoGo: UCT for Computer-Go (Gelly et al., 2006)

Monte-Carlo Go (MoGo)

- **MoGo**: UCT for Computer-Go (Gelly et al., 2006)
- Improving simulation with domain knowledge
  - Local patterns are introduced to have some more reasonable moves during random simulations.

Left: beginning of one random game simulated by pure random mode. Moves are sporadically played with little sense.

Right: beginning of one random game simulated by the pattern-based random mode. From move 5 to move 29 one complicated sequence is generated.

Monte-Carlo Go (MoGo)

MoGo: UCT for Computer-Go (Gelly et al., 2006)

Improving simulation with domain knowledge
- Local patterns are introduced to have some more reasonable moves during random simulations.


Figure 5: Patterns for Hane. True is returned if any pattern is matched. In the right one, a square on a black stone means true is returned if and only if the eight positions around are matched and it is black to play.
Monte-Carlo Go (MoGo)

MoGo: UCT for Computer-Go (Gelly et al., 2006)

- Improving simulation with domain knowledge
  - Local patterns are introduced to have some more reasonable moves during random simulations.

<table>
<thead>
<tr>
<th>Random mode</th>
<th>Win. Rate for B. Games</th>
<th>Win. rate for W. Games</th>
<th>Total Win. Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure</td>
<td>46% (250)</td>
<td>36% (250)</td>
<td>41.2% ± 4.4%</td>
</tr>
<tr>
<td>Sequence-like</td>
<td>77% (400)</td>
<td>82% (400)</td>
<td>80% ± 2.8%</td>
</tr>
</tbody>
</table>

Table 3: Different modes with 70000 random simulations/move in 9x9.

Monte-Carlo Go (MoGo)

- **MoGo**: UCT for Computer-Go (Gelly et al., 2006)
  - For the nodes far from the root, whose number of simulation is very small, UCT tends to be too much exploratory.
  - This is because all the possible moves in one position are supposed to be explored before using the UCB1 formula.

```plaintext
14: for i := 0 to node.childNode.size() - 1 do
15:   if node.childNode[i].nb = 0
       do v[i] := ∞;
16:   else v[i] := -node.childNode[i].value /node.childNode[i].nb +sqrt(2*log(nb)/(node.childNode[i].nb)
17:   end if;
```

Monte-Carlo Go (MoGo)

- **MoGo**: UCT for Computer-Go (Gelly et al., 2006)
  - Exploring order of unvisited nodes: first-play urgency
    - A fixed constant named first-play urgency (FPU) was set.
    - The FPU is set to \( \infty \) in the original UCB1.
    - Smaller FPU ensures earlier exploitation.
    - Any node, after being visited at least once, has its urgency updated according to UCB1 formula.
Monte-Carlo Go (MoGo)

- **MoGo**: UCT for Computer-Go (Gelly et al., 2006)
- Exploring order of unvisited nodes: first-play urgency

<table>
<thead>
<tr>
<th>FPU</th>
<th>Winning Rate for Black Games</th>
<th>Winning Rate for White Games</th>
<th>Total Winning Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4</td>
<td>37% ± 9% (100)</td>
<td>38% ± 10% (100)</td>
<td>37.5% ± 7%</td>
</tr>
<tr>
<td>1.2</td>
<td>46% ± 10% (100)</td>
<td>36% ± 10% (100)</td>
<td>41% ± 7%</td>
</tr>
<tr>
<td>1.1</td>
<td>45% ± 6% (250)</td>
<td>41% ± 6% (250)</td>
<td>43.4% ± 4.4%</td>
</tr>
<tr>
<td>1.0</td>
<td>49% ± 6% (300)</td>
<td>42% ± 6% (300)</td>
<td>45% ± 4%</td>
</tr>
<tr>
<td>0.9</td>
<td>47% ± 8% (150)</td>
<td>32% ± 8% (150)</td>
<td>40% ± 5.6%</td>
</tr>
<tr>
<td>0.8</td>
<td>40% ± 14% (50)</td>
<td>32% ± 13% (50)</td>
<td>36% ± 9.6%</td>
</tr>
</tbody>
</table>

Monte-Carlo Go (MoGo)

- **MoGo**: UCT for Computer-Go (Gelly et al., 2006)
- Exploring order of unvisited nodes: parent information
  - One assumption is that given a situation, good moves may sometimes still be good ones on the following move.
  - MoGo typically use the estimated values of a move m in the grandfather of the node.
Multiplayer Games

- Multiplayer games usually involve alliances.
  - Alliances are made and broken as the game proceeds.
  - In some cases, there is a social stigma to breaking an alliance.

KOEI San5
http://www.koei.com.tw/
Multiplayer Games

- If the game is not zero-sum, then collaboration can also occur with just two players.
### Prisoner’s Dilemma

<table>
<thead>
<tr>
<th></th>
<th>B cooperates</th>
<th>B defects</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A cooperates</strong></td>
<td>$A = 3 / B = 3$ (R/R)</td>
<td>$A = 0 / B = 5$ (S/T)</td>
</tr>
<tr>
<td><strong>A defects</strong></td>
<td>$A = 5 / B = 0$ (T/S)</td>
<td>$A = 1 / B = 1$ (P/P)</td>
</tr>
</tbody>
</table>

No matter what the other does, the selfish choice of defection yields a higher payoff than cooperation.

*What will you do?*
Iterated Prisoner’s Dilemma

- If the number of rounds is fixed, one chooses to always defect.
- In the real world, two individuals may meet more than once. If an individual can recognize a previous interactant and remember some aspects of the prior outcomes, then the strategic situation becomes an iterated Prisoner’s Dilemma.


“Adversarial Search,” Artificial Intelligence, Spring, 2010
Iterated Prisoner’s Dilemma

- Robert Axelrod’s IPD tournament
  - First round (14 entries)
    - The best strategy was “tit for tat”: cooperate at the first round, and do what the opponent does in the previous round.
    - Altruistic strategies did well and greedy strategies did poorly.
  - Second round (62 entries)
    - Tit for tat won the first place again.
    - Among the top 15 entries, only one is not nice.
    - Among the last 15 entries, only one is nice.
Iterated Prisoner's Dilemma

- Common benchmark strategies in IPD

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Full Name</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLC</td>
<td>Always Cooperate</td>
<td>Cooperates indefinitely.</td>
</tr>
<tr>
<td>ALLD</td>
<td>Always Defect</td>
<td>Defects indefinitely.</td>
</tr>
<tr>
<td>TFT</td>
<td>Tit-for-Tat</td>
<td>Cooperates initially but subsequently repeats opponent’s previous move.</td>
</tr>
<tr>
<td>Pavlov</td>
<td>Pavlov (WSLS)</td>
<td>If previous move is SUCCESSFUL (T or R), increase the probability of executing the same move. Decrease the probability otherwise.</td>
</tr>
<tr>
<td>TFTT</td>
<td>Tit-for-Two-Tats</td>
<td>Resembles TFT but forgives opponent for 1 defect. Strategy defects only after 2 consecutive defects by opponent.</td>
</tr>
<tr>
<td>RAND</td>
<td>Random</td>
<td>Defects or cooperates with probability ½.</td>
</tr>
<tr>
<td>STFT</td>
<td>Suspicious TFT</td>
<td>Defects initially but subsequently repeats opponent’s previous move.</td>
</tr>
</tbody>
</table>

Iterated Prisoner’s Dilemma

- Good properties of successful strategies in the IPD:
  - Nice (cooperate first)
  - Retaliating (defect if the opponent defects)
  - Forgiving (cooperate if the opponent apologizes)
  - Non-envious (do not exploit the opponent)

- In the IPD, the optimal strategy depends upon the strategies of likely opponents.
Iterated Prisoner’s Dilemma

- Evolving IPD strategies by GA (Axelrod 1987)
  - Encoding
    - The strategy is deterministic.
    - Use the outcomes of the three previous moves to make a choice in the current move.
    - Since there are 4 possible outcomes (R, T, S, and P) in each move, there are $4 \times 4 \times 4 = 64$ histories of previous three moves.

Iterated Prisoner’s Dilemma

- Evolving IPD strategies by GA (Axelrod 1987)
  - Evaluation
    - Each individual play an 151-move IPD with eight representative strategies in the second round tournament with 62 entries.
  - Mating selection
    - Individual who is one standard deviation above average: two matings
    - Average individual: one mating
    - Individual who is one std. below average: no mating
    - Random pairing
Iterated Prisoner’s Dilemma

- Evolving IPD strategies by GA (Axelrod 1987)
  - One-point crossover
  - Flip mutation
  - Generational GA
  - Random initial population
  - Parameters
    - Population size: 20
    - Generation number: 50
    - Number of runs: 40
Iterated Prisoner’s Dilemma

- Evolving IPD strategies by GA (Axelrod 1987)
  - The GA evolved populations whose median member was just as successful as tit-for-tat.
  - Five behavioral patterns found:
    - Don’t rock the boat (C after RRR)
    - Be provocable (D after RRS)
    - Accept an apology (C after TSR)
    - Forget (C after SRR)
    - Accept a rut (D after PPP)
  - In 11 of 40 runs, the median rule actually does substantially better than tit-for-tat.
Iterated Prisoner’s Dilemma

- Evolving IPD strategies by GA (Axelrod 1987)
  - These strategies manage to exploit one of the eight representative strategies at the cost of achieving somewhat less cooperation with two others.
  - They break the most important advice (to be nice).
    - They always defect on the first one or two moves and use the choices of the other player to discriminate what should be done next.
    - They have responses that allowed them to “apologize” to unexploitable players and keep defecting those who are exploitable.
Iterated Prisoner’s Dilemma

- Evolving IPD strategies by GA (Axelrod 1987)
  - While these rules are effective, we cannot say that they are better than tit-for-tat.
  - They are probably not very robust in other environments.
  - In an ecological simulation these rules would be destroying the basis of their own success.
Iterated Prisoner’s Dilemma

- Evolving IPD strategies by GA (Axelrod 1987)
  - Sexual vs. asexual reproduction
    - The asexual runs were only half as likely to evolve population in which the median member was substantially more effective than tit-for-tat.
  - Changing environment
    - Each individual plays IPD with others in the population.
    - The evolution starts with a pattern of decreased cooperation and decreased effectiveness.
    - After 10~20 generations, a complete reversal takes place. As the reciprocators do well, they spread in the population resulting in more and more cooperation and greater effectiveness.
Iterated Prisoner’s Dilemma

The power of teaming

- A team from Southampton University submitted 60 programs to the 20th IPD competition.
  - These programs try to recognize each other through the first 5~10 rounds.
  - Once the recognition is made, one program always cooperates and the other always defects.
  - If the opponent is a non-Southampton player, it continuously defects.
- They took the top 3 positions in the competition.
Iterated Prisoner’s Dilemma

  - Entries will be evaluated by running a series of evolutionary simulations, in which species of IPD players will compete for survival.
  - In each simulation, an initial population of players will consist of fixed number of players of each species (or coalition of species). This number will be at least 10, and may be more if the number of entries is not too high.
Iterated Prisoner’s Dilemma

  
  - In each generation, each player will play each other player in a round-robin IPD tournament. The fitness of each player will be their total score in the tournament.
  
  - 100 simulations, each for 1000 generations, will be run. The winner will be the species that survives 1000 generations most often. Ties will be broken using the mean number of generations survived (to 2 decimal places).
Iterated Prisoner’s Dilemma


```java
public interface Genotype {
    // Classes implementing Genotype should have a constructor with no parameters
    public Genotype copy();
    public void mutate();
    public Phenotype develop();
}

public interface Phenotype {
    public Move getFirstMove(Class opponentClass);
    public Move getNextMove(Class opponentClass, Move oppLastMove);
}
```
Iterated Prisoner's Dilemma

IPD competition [http://philiphingston.com/ipd/cec2010.html]

```java
class CliquePhenotype implements Phenotype {
    public Move getFirstMove(Class opponentClass) {
        return opponentClass == ipd4species.players.Clique.class? Move.COOPERATE: Move.DEFECT;
    }

    public Move getNextMove(Class opponentClass, Move oppLastMove) {
        return opponentClass == ipd4species.players.Clique.class? Move.COOPERATE: Move.DEFECT;
    }
}

class TitForTatPhenotype implements Phenotype {
    public Move getFirstMove(Class opponentClass) {
        return Move.COOPERATE;
    }

    public Move getNextMove(Class opponentClass, Move oppLastMove) {
        return oppLastMove;
    }
}
```

"Adversarial Search, Artificial intelligence, Spring, 2010"
Other Game Competitions

Ms Pac-Man

http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html

Unlike Pac-Man, Ms. Pac-Man is a non-deterministic game, and rather difficult for most human players. As far as we know, nobody really knows how hard it is to develop an AI player for the game.

The world record for a human player (on the original arcade version) currently stands at 921,360. Can anyone develop a software agent to beat that?

http://www.youtube.com/watch?v=Zo0YujjX1PI
Other Game Competitions

- Unreal Tournament 2004 Deathmatch


The game used for the competition will be based on a modified version of the deathmatch game type for the First-Person Shooter, Unreal Tournament 2004.

This modified version provides a socket-based interface (called Gamebots) that allows control of bots from an external program.

A particularly easy way to interface to the game is to use the Pogamut library, which is written in Java and is available as a Netbeans plugin.
Other Game Competitions

- Unreal Tournament 2004 Deathmatch
  
  [Source](https://artemis.ms.mff.cuni.cz/pogamut/tiki-index.php?page=Agent+tutorial)

```
Example: Selecting the best weapon and shooting target
This example makes agent shoot enemy when it sees one, and stop shooting once it vanishes from the sight.

```java
protected void doLogic() {
    // other commands

    // if shooting and see nothing, stop
    if ((this.memory.isShooting()) && (!this.memory.getSeeAnyEnemy()) this.body.stopShoot();

    // if not shooting and see enemy, attack
    if (!this.memory.isShooting()) && (this.memory.getSeeAnyEnemy()) this.stateAttack();

    // other commands
}

protected void stateAttack() {
    Player target = this.memory.getSeeEnemy();
    if (target == null) return;

    this.body.changeToBestWeapon();
    this.body.shoot(target);
```
Other Game Competitions

■ Car Racing

http://cig.dei.polimi.it/
http://cig.dei.polimi.it/?page_id=134

The goal of the championship is to design a controller for a racing car that will compete on a set of unknown tracks first alone (against the clock) and then against other drivers. The controllers perceive the racing environment through a number of sensors that describe the relevant features of the car surroundings, of the car state, and the game state.

The controller can perform the typical driving actions (clutch, changing gear, accelerate, break, steering the wheel, etc.)
Other Game Competitions

Mario AI Championship

http://www.marioai.org/
http://www.youtube.com/watch?v=DlkMs4ZHHR8&feature=fvw

One of the main purposes of this competition is to be able to compare different controller development methodologies against each other, both those based on learning techniques such as artificial evolution and those that are completely hand-coded. So we hope to get submissions based on evolutionary neural networks, genetic programming, fuzzy logic, temporal difference learning, human ingenuity, hybrids of the above, etc. The more the merrier! (And better for science.)

We believe that playing this game well is a challenge worthy of the best players, the best programmers and the best learning algorithms alike.

Ready to go? Let's get started!
Other Game Competitions

- Starcraft RTS AI Competition
  
  http://code.google.com/p/bwapi/wiki/AIModule

Realtime Strategy (RTS) games are one of the major computer game genres and one of the few for which AI-based players (bots) have little chance to win against expert human players — if they are not allowed to cheat. StarCraft (by Blizzard) is one of the most popular RTS games of all time, and is known to be extremely well balanced.
More about AI in Games

- **Conferences**
  - IEEE Symposium on Computational Intelligence and Games (CIG)
  - IEEE Congress on Evolutionary Computation (CEC)
  - ACM Genetic and Evolutionary Conference (GECCO)
  - Game Developers Conference (GDC)

- **Journals**
  - IEEE Transactions on Computational Intelligence and AI in Games

- **Websites**
  - Game AI for developers (http://aigamedev.com/)
Discussion

- Minimax selects an optimal move provided that the leaf node evaluations are exactly correct.
- In reality, evaluations are usually associated with errors.
Discussion

Choosing the right-hand action might not be good.
Discussion

- The most obvious problem of the alpha-beta algorithm is that it calculates bounds on the values of all the legal moves.
- In a “clear favorite” situation, it would be better to reach a quick decision.
- A good search algorithm should select node expansions of high utility.
Discussion

- To play a game, human often has a particular goal in mind.
- This kind of goal-directed reasoning or planning sometimes eliminates combinatorial search altogether.
- A fully integrated system (goal-direct reasoning + tree/graph search) would be a significant achievement.
Summary

- A game can be defined by
  - the initial state,
  - the legal actions in each state,
  - a terminal test, and
  - a utility function.

- In 2-player zero-sum games with perfect information, the **minimax** algorithm can select optimal moves.
Summary

- The **alpha-beta** search algorithm computes the same optimal moves as minimax, but achieves much greater efficiency.
- Usually, we need to *cut* the search *off* and apply an **evaluation function**.
- *Games of chances* can be handled by taking the average utility of the children nodes the *chance* nodes.
Summary

- Optimal play in games of imperfect information requires reasoning about the current and future belief states of each player.

- Programs can match or beat the best human players in checkers, Othello, and backgammon and are close in bridge. Programs remain at the amateur level in Go.
References