



## Lecture 9 of 42

### Evaluation Functions and Expectiminimax Discussion: Games in AI

Wednesday, 13 September 2006

William H. Hsu

Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/v9v3>

Course web site: <http://www.kddresearch.org/Courses/Fall-2006/CIS730>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

#### Reading for Next Class:

Section 7.1 – 7.4, p. 194 - 210, Russell & Norvig 2<sup>nd</sup> edition



## Lecture Outline

- **Reading for Next Class: Sections 7.1 – 7.4, R&N 2<sup>e</sup>**
- **Today: Game Theory**
  - \* Alpha-beta pruning
  - \* Design and learning of evaluation functions
  - \* Knowledge engineering (KE) and knowledge representation (KR)
- **Friday: Segue to KR**
  - \* Classical knowledge representation
  - \* Limitations of the classical symbolic approach
  - \* Modern approach: representation, reasoning, learning
  - \* “New” aspects: uncertainty, abstraction, classification paradigm
- **Next Week: Start of Material on Logic**
  - \* Representation: “a bridge between learning and reasoning” (Koller)
  - \* Basis for automated reasoning: theorem proving, other inference





## Game Theory Framework: Recap

- **Perfect Play**
  - \* General framework(s)
  - \* What could agent do with perfect info?
- **Resource Limits**
  - \* Search ply
  - \* Static evaluation: heuristic *search* vs. heuristic *game tree search*
  - \* Examples
    - ⇒ Tic-tac-toe, connect four, checkers, connect-five / Go-Moku /  $wu^3 zi^3 qi^2$
    - ⇒ Chess, go
- **Games with Uncertainty**
  - \* Explicit: games of chance (e.g., backgammon, Monopoly)
  - \* Implicit: see suggestions about Angband project



## Minimax Algorithm: Review

```

function MINIMAX-DECISION(game) returns an operator
  for each op in OPERATORS[game] do
    VALUE[op] ← MINIMAX-VALUE(APPLY(op, game), game)
  end
  return the op with the highest VALUE[op]

```

```

function MINIMAX-VALUE(state, game) returns a utility value
  if TERMINAL-TEST[game](state) then ← what's this?
    return UTILITY[game](state) ← what's this?
  else if MAX is to move in state then
    return the highest MINIMAX-VALUE of SUCCESSORS(state)
  else
    return the lowest MINIMAX-VALUE of SUCCESSORS(state)

```

Adapted from slides by S. Russell  
UC Berkeley

Figure 6.3 p. 166 R&N 2e

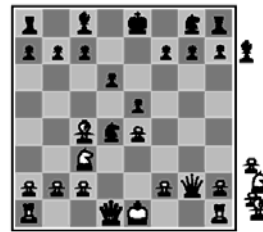




## Static Evaluation Function for Chess: Review



Black to move  
White slightly better



White to move  
Black winning

For chess, typically *linear* weighted sum of features

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$

e.g.,  $w_1 = 9$  with

$f_1(s) = (\text{number of white queens}) - (\text{number of black queens})$

etc.

Adapted from slides by S. Russell  
UC Berkeley

Figure 6.8 p. 173 R&N 2e



## Quiescence and Horizon Effect: Review

### ● Issues

#### \* Quiescence

⇒ Play has “settled down”

⇒ Evaluation function unlikely to exhibit wild swings in value in near future

#### \* Horizon effect

⇒ “Stalling for time”

⇒ Postpones inevitable win or damaging move by opponent

⇒ See: Figure 5.5 R&N

### ● Solutions?

\* Quiescence search: expand non-quiet positions further

\* No general solution to horizon problem at present

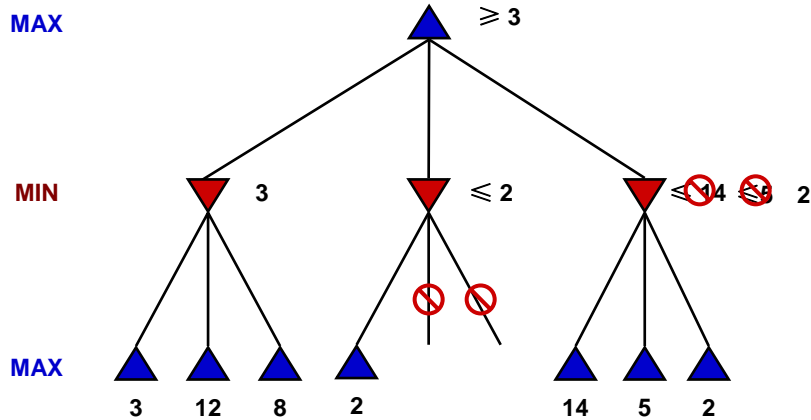
Adapted from slides by S. Russell  
UC Berkeley





## Alpha-Beta ( $\alpha$ - $\beta$ ) Pruning Example [1]: Review

What are  $\alpha$ ,  $\beta$  values here?



Adapted from slides by S. Russell  
UC Berkeley

Figure 6.5 p. 168 R&N 2e



## Alpha-Beta ( $\alpha$ - $\beta$ ) Pruning: Modified Minimax Algorithm

Basically MINIMAX + keep track of  $\alpha$ ,  $\beta$  + prune

```

function MAX-VALUE(state, game,  $\alpha$ ,  $\beta$ ) returns the minimax value of state
  inputs: state, current state in game
         game, game description
          $\alpha$ , the best score for MAX along the path to state
          $\beta$ , the best score for MIN along the path to state

  if CUTOFF-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
     $\alpha \leftarrow \text{MAX}(\alpha, \text{MIN-VALUE}(s, \textit{game}, \alpha, \beta))$ 
    if  $\alpha \geq \beta$  then return  $\beta$ 
  end
  return  $\alpha$ 

function MIN-VALUE(state, game,  $\alpha$ ,  $\beta$ ) returns the minimax value of state
  if CUTOFF-TEST(state) then return EVAL(state)
  for each s in SUCCESSORS(state) do
     $\beta \leftarrow \text{MIN}(\beta, \text{MAX-VALUE}(s, \textit{game}, \alpha, \beta))$ 
    if  $\beta \leq \alpha$  then return  $\alpha$ 
  end
  return  $\beta$ 

```

Adapted from slides by S. Russell  
UC Berkeley





## Properties of $\alpha - \beta$

- Pruning **does not** affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity =  $O(b^{m/2})$   
 → **doubles** depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of **metareasoning**)

Adapted from slides by S. Russell  
UC Berkeley



## Why is it called $\alpha - \beta$ ?

- $\alpha$  is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If  $v$  is worse than  $\alpha$ , *max* will avoid it  
 → **prune that branch**
- Define  $\beta$  similarly for *min*

MAX

MIN

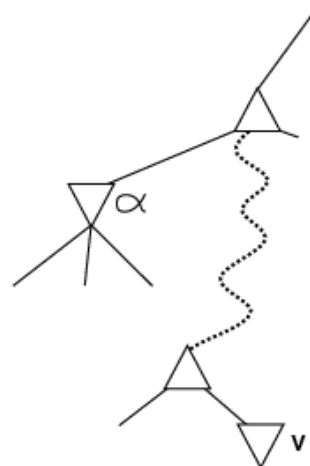
..

..

..

MAX

MIN



Adapted from slides by S. Russell  
UC Berkeley





## The $\alpha$ - $\beta$ algorithm

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game
   $v \leftarrow \text{MAX-VALUE}(\textit{state}, -\infty, +\infty)$ 
  return the action in SUCCESSORS(state) with value  $v$ 

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$ 
    if  $v \geq \beta$  then return  $v$ 
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return  $v$ 
```

Adapted from slides by S. Russell  
UC Berkeley



## The $\alpha$ - $\beta$ algorithm

```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  inputs: state, current state in game
          $\alpha$ , the value of the best alternative for MAX along the path to state
          $\beta$ , the value of the best alternative for MIN along the path to state
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for  $a, s$  in SUCCESSORS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s, \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$ 
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return  $v$ 
```

Adapted from slides by S. Russell  
UC Berkeley





## Resource limits

Suppose we have 100 secs, explore  $10^4$  nodes/sec  
→  $10^6$  nodes per move

Standard approach:

- **cutoff test:**  
e.g., depth limit (perhaps add **quiescence search**)
- **evaluation function**  
= estimated desirability of position

Adapted from slides by S. Russell  
UC Berkeley



## Evaluation functions

- For chess, typically **linear** weighted sum of **features**  
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$
- e.g.,  $w_1 = 9$  with  
 $f_1(s) = (\text{number of white queens}) - (\text{number of black queens}), \text{ etc.}$

Adapted from slides by S. Russell  
UC Berkeley





## Deterministic games in practice

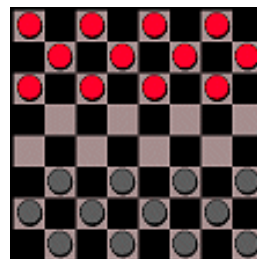
- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions.
- **Chess:** Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply.
- **Othello:** human champions refuse to compete against computers, who are too good.
- **Go:** human champions refuse to compete against computers, who are too bad. In go,  $b > 300$ , so most programs use pattern knowledge bases to suggest plausible moves.

Adapted from slides by S. Russell  
UC Berkeley



## Digression: Learning Evaluation Functions

- **Learning = Improving with Experience at Some Task**
  - \* Improve over task  $T$ ,
  - \* with respect to performance measure  $P$ ,
  - \* based on experience  $E$ .
- **Example: Learning to Play Checkers**
  - \*  $T$ : play games of checkers
  - \*  $P$ : percent of games won in world tournament
  - \*  $E$ : opportunity to play against self
- **Refining the Problem Specification: Issues**
  - \* What experience?
  - \* What *exactly* should be learned?
  - \* How shall it be *represented*?
  - \* What specific algorithm to learn it?
- **Defining the Problem Milieu**
  - \* Performance element: How shall the results of learning be applied?
  - \* How shall performance element be evaluated? Learning system?





## Example: Learning to Play Checkers

- **Type of Training Experience**
  - \* Direct or indirect?
  - \* Teacher or not?
  - \* Knowledge about the game (e.g., openings/endgames)?
- **Problem: Is Training Experience *Representative* (of Performance Goal)?**
- **Software Design**
  - \* Assumptions of the learning system: *legal* move generator exists
  - \* Software requirements: generator, evaluator(s), parametric target function
- **Choosing a Target Function**
  - \* *ChooseMove*:  $Board \rightarrow Move$  (action selection function, or *policy*)
  - \*  $V: Board \rightarrow R$  (board evaluation function)
  - \* Ideal target  $V$ ; approximated target  $\hat{V}$
  - \* Goal of learning process: operational description (approximation) of  $V$



## A Target Function for Learning to Play Checkers

- **Possible Definition**
  - \* If  $b$  is a final board state that is won, then  $V(b) = 100$
  - \* If  $b$  is a final board state that is lost, then  $V(b) = -100$
  - \* If  $b$  is a final board state that is drawn, then  $V(b) = 0$
  - \* If  $b$  is not a final board state in the game, then  $V(b) = V(b')$  where  $b'$  is the best final board state that can be achieved starting from  $b$  and playing optimally until the end of the game
  - \* *Correct values, but not operational*
- **Choosing a Representation for the Target Function**
  - \* Collection of rules?
  - \* Neural network?
  - \* Polynomial function (e.g., linear, quadratic combination) of board features?
  - \* Other?
- **A Representation for Learned Function**
  - \*  $\hat{V}(b) = w_0 + w_1 bp(b) + w_2 rp(b) + w_3 bk(b) + w_4 rk(b) + w_5 bt(b) + w_6 rt(b)$
  - \*  $bp/rp$  = number of black/red pieces;  $bk/rk$  = number of black/red kings;  $bt/rt$  = number of black/red pieces *threatened* (can be taken next turn)

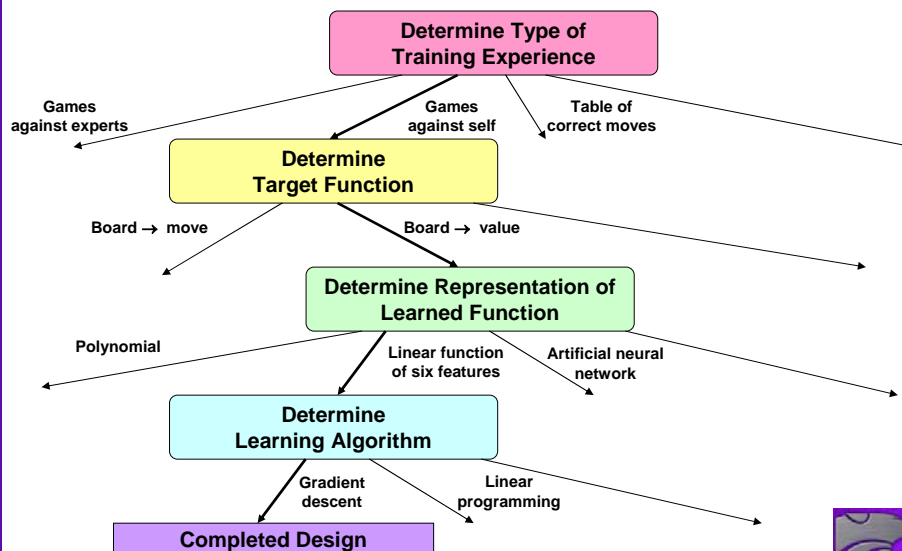


## Training Procedure for Learning to Play Checkers

- **Obtaining Training Examples**
  - \*  $V(b)$  the target function
  - \*  $\hat{V}(b)$  the learned function
  - \*  $V_{train}(b)$  the training value
- **One Rule For Estimating Training Values:**
  - \*  $V_{train}(b) \leftarrow \hat{V}(Successor(b))$
- **Choose Weight Tuning Rule**
  - \* **Least Mean Square (LMS) weight update rule:**  
REPEAT
    - ⇒ Select a training example  $b$  at random
    - ⇒ Compute the  $error(b)$  for this training example
    - ⇒ For each board feature  $f_p$ , update weight  $w_i$  as follows:  
$$error(b) = V_{train}(b) - \hat{V}(b)$$
where  $c$  is a small, constant factor to adjust the learning rate  
$$w_i \leftarrow w_i + c \cdot f_i \cdot error(b)$$

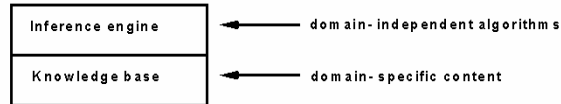


## Design Choices for Learning to Play Checkers





## Knowledge Bases



Knowledge base = set of sentences in a formal language

Declarative approach to building an agent (or other system):

TELL it what it needs to know

Then it can ASK itself what to do—answers should follow from the KB

Agents can be viewed at the knowledge level

i.e., what they know, regardless of how implemented

Or at the implementation level

i.e., data structures in KB and algorithms that manipulate them

Adapted from slides by S. Russell  
UC Berkeley



## Simple Knowledge-Based Agent

```

function KB-AGENT(percept) returns an action
  static: KB, a knowledge base
           t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
  
```

The agent must be able to:

Represent states, actions, etc.

Incorporate new percepts

Update internal representations of the world

Deduce hidden properties of the world

Deduce appropriate actions

Adapted from slides by S. Russell  
UC Berkeley

Figure 6.1 p. 152 R&N





## Summary Points

- **Fun and Games? Illustrating Key Features of AI Problems**
- **Games as Search Problems**
  - \* **Frameworks**
  - \* **Concepts**
    - ⇒ **Utility and representations (e.g., static evaluation function)**
    - ⇒ **Reinforcements: possible role for machine learning**
    - ⇒ **Game tree**
- **Expectiminimax: Extension to Account for Uncertainty**
- **Algorithms for Game Trees**
  - \* **Propagation of credit**
  - \* **Imperfect decisions**
  - \* **Issues**
    - ⇒ **Quiescence**
    - ⇒ **Horizon effect**
  - \* **Need for pruning**



## Terminology

- **Game Tree Search**
  - \* **Concepts**
    - ⇒ **Utility and representations (e.g., static evaluation function)**
    - ⇒ **Reinforcements: possible role for machine learning**
    - ⇒ **Game tree: node/move correspondence, search ply**
  - \* **Mechanics**
    - ⇒ **Minimax and alpha-beta pruning**
    - ⇒ **Expectiminimax**
    - ⇒ **Features in static evaluation**
- **Convergence and Analysis**
  - \* **Propagation of credit**
  - \* **Quiescence and the horizon effect**
- **Convergence and Analysis**

