

# Computing Science (CMPUT) 455

## Search, Knowledge, and Simulations

Martin Müller

Department of Computing Science  
University of Alberta  
[mmueller@ualberta.ca](mailto:mmueller@ualberta.ca)

Fall 2022

# Today's Topics - Lecture 11

---

- Defining and using knowledge in heuristic search
- State evaluation
- Move evaluation

# Coursework

---

- Work on Assignment 2
- Quiz 6
- Finish reading: van der Werf et al., *Solving Go on small boards*

## Knowledge in Heuristic Search

# Using Knowledge in Heuristic Search

---

- How can knowledge be used in heuristic search?
  - Evaluate states
  - Evaluate actions
  - Other ways
- Properties and interpretations of knowledge for heuristic search
  - Probabilities, preferences, ordering, ranking
- Representing knowledge
  - Rules, patterns, features, neural nets
- Acquiring knowledge
  - Manual vs machine learning

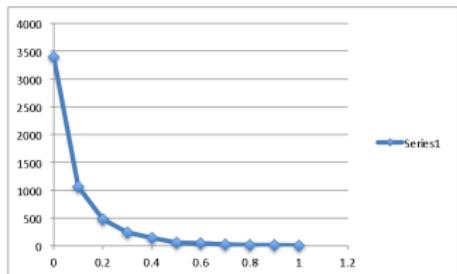
# Knowledge for Search - the Story So Far

---

- Discussed many search techniques
- Most were “knowledge-free”
  - Blind search
  - Random sampling
  - Solving games with boolean negamax or alphabeta
- Others used a “black box” heuristic evaluation function
  - Depth-limited alphabeta search
  - Move ordering heuristic
- How to build such a heuristic function?

# Treasure Hunt Example from Lecture 7

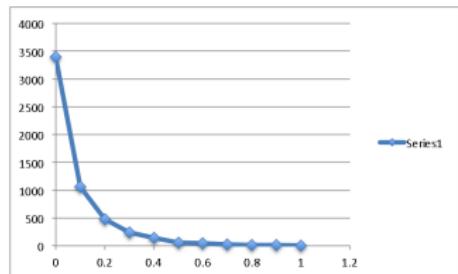
---



- Results from running `heuristic_search_on_tree.py`
- A little bit of heuristic guidance greatly reduced number of samples until treasure was found
- With perfect heuristic: walk straight to goal
- **Question:** Could we do as well in games with a perfect heuristic?

# Treasure Hunt Example from Lecture 7

---



- Results from running `heuristic_search_on_tree.py`
- A little bit of heuristic guidance greatly reduced number of samples until treasure was found
- With perfect heuristic: walk straight to goal
- **Question:** Could we do as well in games with a perfect heuristic?
- *Not quite.* Remember proof trees. Even with perfect heuristics, need to cover *all* the opponent's move choices.
  - Not sufficient to find a single path to the “goal”

# Big Questions - Knowledge for Heuristic Search

---

- What is knowledge used for?
- Where does it come from?
- How is it selected?
- How is it constructed?
- How is it learned?

# Kinds of Knowledge in Heuristic Search

---

- Many kinds of knowledge in heuristic search
- The “big two” for us:
  1. State evaluation
  2. Move evaluation
- Many other examples of using knowledge

# Other Kinds of Knowledge in Heuristic Search

---

- Time control, search depth control
- Knowledge to reduce size of state space
  - DAG vs tree
  - Benson's algorithm in Go
- Efficient state representation (we discussed)
- Knowledge about algorithm optimization and tuning
- Many more...

# Knowledge for State Evaluation

---

## State evaluation

- What is it?
  - Mapping from (full) state to one number
- What does it measure?
  - How good is that state?
- Other terms with similar meaning:
  - Evaluation function
  - Position evaluation

# Knowledge for Move Evaluation

---

## Move evaluation

- What is it?
  - Mapping from move (action) to number
- What does it measure?
  - How good is that move?
  - Example: probability that this is the best move
- Can be used as filter:
  - Which moves are (probably) bad and should be filtered out (pruned)
- Other terms with similar meaning:
  - Action value
  - Move value
  - Q-value in machine learning

## Details on State Evaluation

---

- We know exact evaluation in **terminal states**
  - Win, loss, draw, win by 23.5 points,...
- What about heuristic evaluation in **non-terminal states?**
- In games, two kinds of evaluation are popular
- Heuristic evaluation:
  - Higher is better
  - “Just a number”, no extra meaning, used for ranking/ordering moves
- Winning probability:
  - Higher is better
  - Has an interpretation as probability

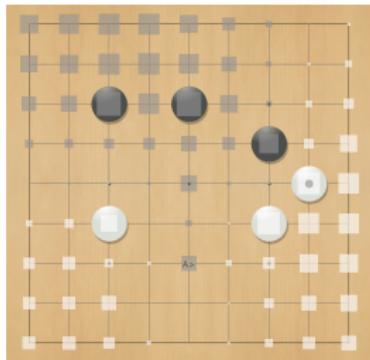
# Heuristic State Evaluation in Search

---

- Most important use of state evaluation:
  - Evaluation function in search
- Leaf nodes *of search* evaluated by this function
  - Exact evaluation for terminal states
  - **Heuristic evaluation of non-terminal states**
- Interior nodes in search evaluated by minimax rule
  - Backup the evaluation in leaf nodes:
    - to parent ...
    - to grandparent, further up ...
    - ... all the way to the root

# Heuristic Evaluation Function

---



- Heuristic evaluation:  
higher is better
- One possible interpretation:  
estimate of score of game
  - +12
  - “Black is about 12 points ahead”
- May have no “hidden” interpretation
  - Evaluation can be “just a number”
  - Used just for relative ranking of positions

## Example: Evaluation Function in Chess

---

- Queen = 9, rook = 5, bishop = 3,...
- Positional features, such as location of pieces
- Evaluation =  
sum of my material's values  
- sum of opponent's material's values
- Higher value is better
- No “deeper meaning”, just a number
- Use in search, example:
  - Move A leads to position with evaluation 1.49
  - Move B leads to position with evaluation 1.52
  - Search will select B over A

## Relative vs Absolute Evaluation

---

- For decision-making, the evaluation numbers themselves do not matter
- Only the *ordering* given by the numbers matters (higher is better)
- It decides the *preference* or *ranking* between moves
  - Example 1: multiply all evaluations by 10
  - Example 2: add 7 to all evaluations
  - ⇒ The search will make exactly the same decisions

## Relative vs Absolute Evaluation

---

- Monotonically increasing (order-preserving) function:

$$x > y \implies f(x) > f(y)$$

- Any monotonically increasing mapping of the evaluation function will give the same search behavior
- Same idea in utility theory (Lecture 3): *ordinal utility*

## Relative vs Absolute Evaluation

---

- Monotonically increasing (order-preserving) function:

$$x > y \implies f(x) > f(y)$$

- Any monotonically increasing mapping of the evaluation function will give the same search behavior
- Same idea in utility theory (Lecture 3): *ordinal utility*
- Cardinal comparisons (exact values) become important when you **aggregate** multiple evaluations
  - E.g., by taking expectations
  - Some increasing mappings still give same search behavior when evaluations aggregated
  - Others might not

## But - What Does an Evaluation Mean?

---

One interpretation:

- General motto:  
“Similar evaluation values for similar states”

More precise version:

- All states with the same evaluation are “equally good”
- They have the same (unknown to us) probability of winning
- Higher evaluation = higher probability of winning
- Evaluation function partitions set of all states  $S$ 
  - Subsets  $S_v$ , one subset for each value  $v$
  - All states in same  $S_v$  are assumed equally good for us

# Mixing Exact and Heuristic Evaluation

---

- We can mix exact and heuristic evaluations
- If we are careful, we can get true proofs of wins and losses this way
- Example: win = 10000, highest heuristic score = 5000
- If alphabeta returns 10000, it is a proven win
- Having a good heuristic can help speed up an exact proof
  - Provides good move ordering for iterative deepening search
  - A better move sorted first means more cuts in the tree search
  - Example: checkers proof - iterative search using heuristics
- Careful: remember problem with draw vs heuristic scores in proving TicTacToe

# What does Winning Probability Mean?

---

- Different interpretations
- Clearest case:
  - game with chance element
  - e.g. dice rolls
- The winning probability **is** the minimax score!
- Example from backgammon
  - Endgame state  $s$
  - Need to roll two sixes to win, lose otherwise
  - Probability of rolling two sixes =  $1/6 \times 1/6 = 1/36$
  - State evaluation  $v(s) = 1/36$
  - Use as evaluation in search:  
move to  $s$  if it has the highest winning probability

# Winning Probability in Games with No Chance

---

- There are no probabilities in the game itself
- A perfect player would know with certainty
  - True winning probability is either 0% or 100%
- Probability can arise from
  - Our imperfect understanding of the game
  - Generalization and abstraction in our evaluation
  - Randomness in our strategy
  - Randomness or imperfect understanding of the opponent's strategy

# Example - Winning Probabilities in Value Net

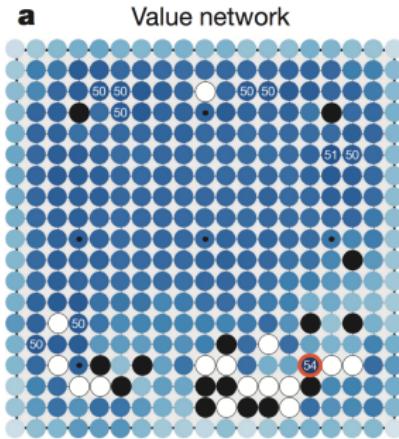


Image source: Silver et al,

Mastering the game of Go with  
deep neural networks and tree  
search, Nature

- Can use machine learning to *learn estimates of win probabilities*
- Example: AlphaGo's value network
- Deep neural net
- Map: from state to learned win probability

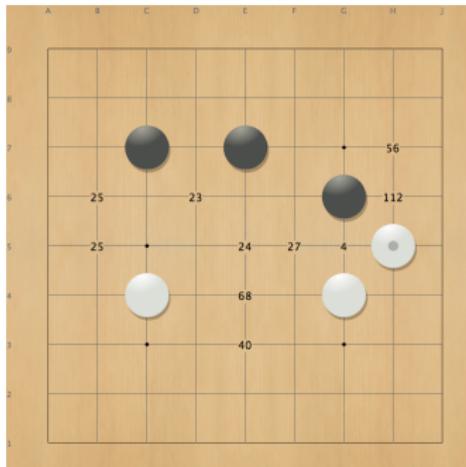
# Using Knowledge for Move Evaluation

---

- Given a state
- Also given the possible moves from that state
- Put a numeric value on each move
- Main use: action selection in search, in simulation
- Can also be used for
  - Move ordering in search
  - Move pruning
- Again, we can have both types of evaluation
  - Without a probabilistic interpretation
  - With a probabilistic interpretation

# Move Evaluation as “A Number”

---



- No interpretation
- Bigger is better
- Example: Go program Explorer (ca. 1989 - 1995)
- Where did its evaluations come from?
- Large number of hand-made heuristics for different types of moves

## Move Evaluation as Probability

---

- Assume move  $i$  has probability  $p_i$ :
- Interpretation 1:
  - $p_i$  is probability that move  $i$  is a win
- Interpretation 2:
  - $p_i$  is probability that move  $i$  is the best move
- Both make sense
- Which one you use depends on how you compute or estimate those numbers

# Relation between State and Move Evaluation (1)

---

Case 1:

- We have state evaluation
- We need move evaluation
- Easy - do a 1 ply search
- Evaluation of move:
  - Evaluation of state after making that move
- Main disadvantage: slow if branching factor is large
  - Example -  $19 \times 19$  Go: over 300 moves, evaluations, undo to make a single decision

## Relation between State and Move Evaluation (2)

---

Case 2:

- We have only move evaluation
- We need state evaluation
- No easy solution
- We could try to do “greedy rollout” by following the sequence of best-evaluated moves
- Slow, noisy result
- Not used in practice (compare with randomized simulations later)
- Still have to evaluate the terminal state to get a value

# Acquiring Evaluation Knowledge

---

- Where do evaluations come from?
- (now) Machine learning
- (old) Local goal-directed search
- (old) Handcoded rules
- First, discuss how to represent knowledge in a program

# Representing Knowledge for Evaluation

---

- Many ways to represent knowledge
- Handcoded rules
- Simple features
- Pattern databases
- Neural nets

## Handcoded Rules

---

```
def selfatari(board, move, color):
    maxoldliberty = maxliberty(board, move,
                               color, 2)
    if maxoldliberty > 2:
        return False
    cboard = board.copy()
    isLegal = cboard.move(move, color)
    if isLegal:
        newliberty = cboard.liberty(move, color)
        if newliberty == 1:
            return True
    return False
```

- Most direct way
- Example: heuristic move filter, “avoid selfatari”

# Simple Features

---

```
enum FeBasicFeature{  
    FE_PASS_NEW,  
    FE_PASS_CONSECUTIVE,  
    FE_CAPTURE_ADJ_ATARI,  
    ...  
    FE_CAPTURE_MULTIPLE,  
    FE_EXTENSION_NOT_LADDER,  
    FE_EXTENSION_LADDER,  
    ...  
    FE_TWO_LIB_SAVE_LADDER,  
    FE_TWO_LIB_STILL_LADDER,  
    ...  
    FE_SELFATARI,  
    FE_ATARI_LADDER,  
    ...  
    FE_DOUBLE_ATARI,  
    FE_DOUBLE_ATARI_DEFEND,  
    FE_LINE_1,  
    FE_LINE_2,  
    FE_LINE_3,  
    ...  
}
```

- Idea: each feature is a boolean statement about a state, or a move
- Each feature is simple and easy to compute
- With machine learning, we can construct an evaluation function from a combination of many simple features
- Move feature vector:  
 $(0,0,1,\dots,1,1,0,\dots,1,0,\dots,0,0,\dots)$
- Examples: next few slides

# Remi Coulom's Simple Features (1)

---

Feature	Level	$\gamma$	Description
Pass	1	0.17	Previous move is not a pass
	2	24.37	Previous move is a pass
Capture	1	30.68	String contiguous to new string in atari
	2	0.53	Re-capture previous move
	3	2.88	Prevent connection to previous move
	4	3.43	String not in a ladder
	5	0.30	String in a ladder
Extension	1	11.37	New atari, not in a ladder
	2	0.70	New atari, in a ladder
Self-atari	1	0.06	
Atari	1	1.58	Ladder atari
	2	10.24	Atari when there is a ko
	3	1.70	Other atari
Distance to border	1	0.89	
	2	1.49	
	3	1.75	
	4	1.28	

Source: Remi Coulom, Computing Elo Ratings of Move Patterns in the Game of Go

## Remi Coulom's Simple Features (2)

---

Distance to	2	4.32	$d(\delta x, \delta y) =  \delta x  +  \delta y  + \max( \delta x ,  \delta y )$
previous move	3	2.84	
	4	2.22	
	5	1.58	
	...	...	
	16	0.33	
	$\geq 17$	0.21	
Distance to	2	3.08	
the move before	3	2.38	
the previous move	4	2.27	
	5	1.68	
	...	...	
	16	0.66	
	$\geq 17$	0.70	

Source: Remi Coulom, Computing Elo Ratings of Move Patterns in the Game of Go

# Pattern Databases

---

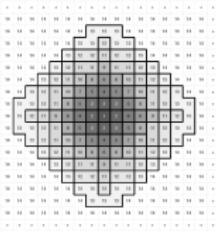
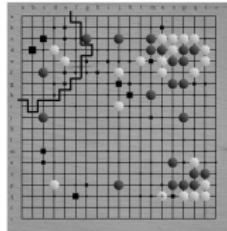


Image source: Stern et al, Bayesian  
Pattern Ranking for Move Prediction in  
the Game of Go

- Large patterns can be learned from master games, if they are frequently used
- In Go, typically we have many different sizes of pattern:
- From small 3x3 patterns to full board
- A main question is how to evaluate such patterns
- Measure how often the move in the center is played immediately, or later

# Neural Nets

---

Convolutional Neural Network (CNN)

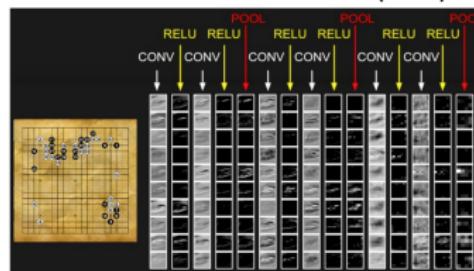


Image source:

<https://www.slideshare.net/>

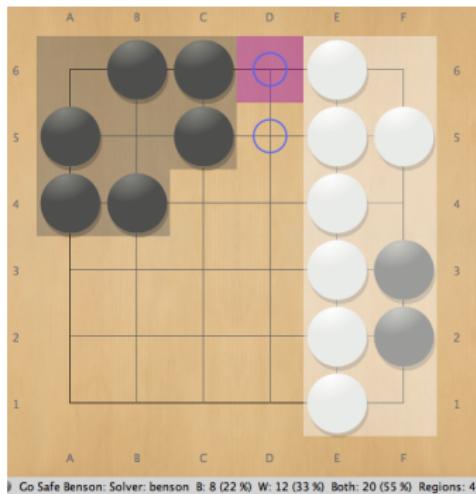
ShaneSeungwhanMoon/

how-alphago-works

- Represent knowledge in (large number of) weights of the neural net
- Lower levels of net encode local knowledge (e.g. 3x3, 5x5)
- Higher levels can express global evaluation
- Much more on nets later in the course

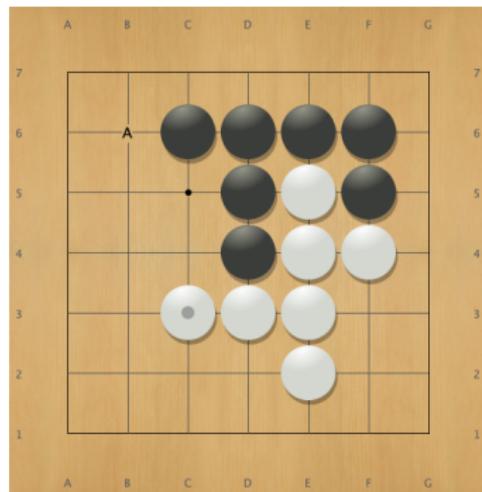
# Example of Exact Knowledge: Benson's Algorithm in Go

---



- Benson's algorithm finds stones and territories that are *unconditionally alive*
- No matter what opponent plays: they can never capture our stones
- A generalization of “two eyes”
- Can be used as a filter in a program - do not generate moves in safe territory

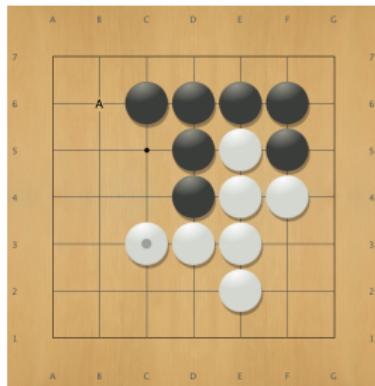
# Example of Heuristic Knowledge: Maximizing Average Liberties in Go



- GoHeuristicPlayer in our collection of simple Go programs
- Simple heuristic: build strong blocks with many liberties
- Also try to reduce the opponent's liberties
- Connect own blocks and cut opponent blocks

Link: <https://webdocs.cs.ualberta.ca/~mmueller/courses/cmput455/html/Go-programs.html#GoHeuristicPlayer>

# GoHeuristicPlayer Algorithm



- Simple Evaluation function for board states
  - Count number of liberties for each block
  - evaluation =  
average liberties of our blocks  
- average liberties of opponent blocks
- 1-ply search
  - Try all legal moves
  - Evaluate the board after each move
  - Choose the move with best evaluation
- Always beats random
- Loses against simple simulation-based players (see later)

# Summary

---

- Many kinds of knowledge
- Used for evaluating states and moves
- Heuristic rules, patterns, neural networks
- Simple heuristic player for Go, maximizes difference in average number of liberties
- Exact knowledge, e.g. safe stones