The game of Go, Monte Carlo Tree Search and Computer Go

Martin Müller
mmaueller@ualberta.ca

Acknowledgement: some slides adapted from Dave Silver’s presentations
Contents

• Limits of alpha-beta search
• The Game of Go
• What is Monte Carlo Tree Search?
• The Fuego project
Alphabeta Search

- Minimax principle
- My turn: choose best move
- Opponent’s turn: they choose move that’s worst for me
- Alphabeta (αβ): prune irrelevant parts of tree
\textbf{\(\alpha\beta\) Successes (1)}

- Full search: solve the game
  - checkers (Schaeffer et al 2007)
  - Nine men’s morris (Gasser 1994)
  - Gomoku (5 in a row) (Allis 1990)
  - Awari, 5x5 Go, 5x5 Amazons,.....
αβ Successes (2)

• Not solved, but super-human strength:
  • chess (Deep Blue team, 1996)
  • Othello (Buro 1996)

• Grandmaster strength:
  • shogi (Japanese chess)
  • xiangqi (Chinese chess)
αβ Failures

• Go

• General Game Playing (GGP)

• Why fail?
  • Focus on Go here
Go

• Classic Asian board game
• Simple rules, complex strategy
• Played by millions
• Hundreds of top experts - professional players
• Until recently, computers much weaker than humans
Go Rules

• Start: empty board
• Goal: surround
• Empty points
• Opponent (capture)
• Win: control more than half the board
• Komi: compensation for first player advantage
End of Game

- End: both players pass
- **Territory** - intersections surrounded by one player
- The player with more (stones+territory) wins the game
- **Komi**: adjustment for first player advantage (e.g. 7.5 points)
Why does $\alpha \beta$ Fail in Go?

- Depth and width of game tree
  - 250 moves on average
  - game length $> 200$ moves
- Lack of good evaluation function
Monte Carlo Methods

- Recently popular (mainly last 5 years)
- Hugely successful
  - Backgammon (Tesauro)
  - Scabble (Sheppard)
  - Go (many)
- Amazons, Havannah, Lines of Action, ...
Monte-Carlo Simulation

- No evaluation function? No problem!
- Simulate rest of game using random moves (easy)
- Score the game at the end (easy)
- Use that as evaluation (hmm, but...)
The GIGO Principle

• Garbage in, garbage out
• Even the best algorithms do not work if the input data is bad
• How can we gain any information from playing random moves?
Well, it Works!

• For some games, anyway

• Even random moves often preserve some difference between a good position and a bad one

• The rest is statistics...
  • ...well, not quite.
Basic Monte Carlo Search

• Play lots of random games starting with each possible move
• Keep winning statistics for each move
• Play move with best winning percentage
Simulation - Example

• Random legal moves, but...
• ...do not fill one point eyes
• End of game after both pass
• Evaluate by Chinese rules:
  +1 for win
  0 for loss
Example (for one move)
Example (for one move)
Example (for one move)

Current positions

Simulation
Example (for one move)

Current position $s$

Simulation

Outcomes

$1 \ 1 \ 0 \ 0$
Example (for one move)

\[ V(s) = \frac{2}{4} = 0.5 \]

Current position \( s \)

Simulation

Outcomes

1 1 0 0
Evaluation

• Surprisingly good e.g. in Go - much better than random or simple knowledge-based players

• Still limited
  • Prefers moves that work “on average”
  • Often these moves fail against the best response
  • “Silly threats”
How to Improve?

1. Better-than-random simulations
2. Add game tree (as in $\alpha\beta$)
3. Add statistics over move quality (RAVE, AMAF) - not today
4. Add knowledge in the game tree - not today
   1. human knowledge
   2. machine-learnt knowledge
1. Better Simulations

- Goal: strong correlation between initial position and result of simulation
- Preserve wins and losses
- How?
Knowledge in Simulations

- MoGo-style patterns
- Tactical rules
MoGo-Style Patterns

- 3x3 or 2x3 patterns
- Apply as response near last move

[Diagram showing various patterns]
Tactical rules

- Escape from threats
- Stabilize/attack weak stones
Example of Biased Simulation

valkyria-ExBoss-biased-random-game.sgf
Building a better Randomized Policy

• Two main approaches
  • Crazy Stone: use rules, patterns to set probabilities for each legal move
  • MoGo, Fuego: hierarchy of rules
    • Find set of highest priority rules
    • Choose randomly from this (often small) set
2. Add Game Tree

• Using simulations directly as an evaluation function for $\alpha\beta$ fails

• Too much noise, or too slow if running many simulations per state

• Result: Monte-Carlo was ignored for over 10 years in Go
Monte Carlo Tree Search

- Idea: use results of simulations to guide growth of the game tree

- **Exploitation:** focus on promising moves

- **Exploration:** focus on moves where uncertainty about evaluation is high

- Two contradictory goals?
UCB Formula

- Multi-armed bandits (slot machines in Casino)
- Which bandit has best payoff?
- Explore all arms, but:
  - Play promising arms more often
- Minimize regret from playing poor arms
UCT Algorithm

• Kocsis and Szepesvari (2006)
• Apply UCB in each node of a game tree
• Which node to expand next?
• Start at root (current state)
• While in tree, choose child $n$ that maximizes

$$\text{UCTValue}(\text{parent}, n) = \text{winrate}(n) + C \times \sqrt{\frac{\ln(\text{parent.visits})}{n\.visits}}$$
• $\text{UCTValue}(parent, n) =$
  
  $\text{winrate}(n) + C \times \sqrt{\ln(parent.\text{visits})/n.\text{visits}}$

• $\text{winrate}(n)$.. *exploitation* term - average success of $n$ so far

• $1/n.\text{visits}$.. part of *exploration* term - explore nodes with very few visits - reduce uncertainty

• $\ln(parent.\text{visits})$.. part of *exploration* term - explore all nodes at least a little bit

• $C$.. *exploration constant* - how important is exploration relative to exploitation?
Monte Carlo Tree Search

Current state → Tree Policy

Default Policy
Monte Carlo Tree Search
Monte Carlo Tree Search

Current state

Tree Policy

Default Policy

1/1

2/3

0/1
Monte Carlo Tree Search

Current state → 2/4

Tree Policy

Default Policy
Monte Carlo Tree Search

Current state

Tree Policy

Default Policy
Summary of Monte Carlo Tree Search

• Amazingly successful in games where alphabeta failed
• Top in Backgammon, Go, General Game Playing, Hex, Amazons, Lines of Action, Havannah,...
• Similar methods work in multiplayer games (e.g. card games), planning and puzzles
Summary(2)

- Very successful in practice
- Scales OK to parallel machines
- Reasons for why and how it works still poorly understood
- Some limitations (see later)
The Fuego Project

- Developed at UofA (Enzenberger, Müller, Arneson,...)
- Open-source program hosted on sourceforge
- http://fuego.sourceforge.net/
- Goals:
  - General game-independent framework
  - Strong programs, e.g. Go, Hex, Amazons
Fuego Structure

- Game-independent kernel: smartgame library
  - MCTS, alphabeta, common data structures, utility classes
- General Go engine
  - Go board, rules, blocks, static safety algorithms
- Fuego - Monte Carlo Go program
Fuego Go Successes

• 2009: First program to beat top human professional Chou Chun-Hsun in 9x9 game with no handicap

• Won 2009 Computer Olympiad 9x9 Go

• 2nd place in 2009 Olympiad 19x19

• 8-core Fuego ranked 3rd on 9x9 CGOS all-time

• 80-core Fuego about 200 Elo stronger
Analysis and Update

• (Game demo here)

• According to expert analysis, Chou did not make a mistake in this game, but Fuego played flawlessly

• A milestone for Computer Go

• In game 2 with Black, Fuego played a dubious move 3 and lost easily

• This year, we had 1 win 3 losses against professionals :(
Projects using Fuego(1)

- **Bluefuego**: MPI library for Fuego
  - Developed by IBM
  - Distributed memory - connects (many) copies of Fuego
  - Scales to hundreds of cores

- **MoHex**: strongest Hex program
  - Developed by Ryan Hayward’s group, CS, University of Alberta
  - Uses SmartGame kernel, MCTS engine
Projects using Fuego(2)

- **Explorer**: "classical" Go program
  - Strong solvers for Tsume Go, safe territory, endgame
  - Uses Fuego’s SmartGame, Go libraries
- **FuegoEx**: add Explorer knowledge to Fuego
  - Tactical search (block capture)
  - Pattern matching - 4000 handmade large irregular patterns
  - Filter: prune blunders from MCTS
Projects using Fuego(3)

- **RLGO** (Dave Silver)
  - Reinforcement-learning based Go program
  - Uses SmartGame, Go, GoUct
- **TsumeGo Explorer** (Kishimoto + Müller)
  - World’s best Life and Death solver for enclosed areas
  - Uses SmartGame, Go
- **Arrow** (Müller) Amazons-playing program
  - uses classical alpha-beta search other basic functionality from SmartGame
  - Arrow2 (Huntley, VanEyck) MCTS-based
Parallel Search

- Shared memory parallelization (Enzenberger)
- Good speedup up to about 8 cores
- Lockfree shared game tree
- Memory-limited
- Distributed memory - BlueFuego (IBM)
Research Challenges

• How to improve simulations?
  • offline
  • online (during a game)
• How to achieve “locally strong” play?
  • Global search cannot see enough
• How to scale to massively parallel systems?
Summary

• Monte-Carlo methods have revolutionized search and games

• Still not well understood

• Lots of good research to be done

• Ideas transfer to planning, optimization,...

• Challenge in Go:
  • How to scale up to full 19x19 board?