Exploration and Exploitation in Go: UCT for Monte-Carlo Go

Sylvain Gelly¹, Yizao Wang¹,²

1: Université Paris-Sud, INRIA, CNRS, TAO Group, FRANCE
2: Applied Mathematics Center, Ecole Polytechnique, FRANCE

December 9, 2006
A quick introduction to game of Go
A Quick introduction to Go game

- Go-board (Goban): 19 × 19 intersections;
- Back and White play alternatively. Black starts the game;
- Adjacent stones are called a string. Liberties are the empty intersections next to the string;
- Stones do not move, there are only added and removed from the board. A string is removed if its number of liberties is 0;
- Score: territories (number of occupied or surrounded intersections).
Beginning of Computer-Go, 1970s

Classical methods
- Expert knowledge based evaluation function;
- Minimax tree search;

Comparison with chess
- Chess: Deeper Blue won against Kasparov, 1997;
- Go: The strongest programs are about 10kyu in 2006 (amateurs of good level can win with 9 stones handicap)
Difficulties in computer-Go

- Huge branching factor $\approx 200$, chess $\approx 40$
  (John Tromp and Gunnar Farnebäck, 2006)
- Legal positions number
  $2.0 \times 10^{170}$ on $19 \times 19$, $1.0 \times 10^{38}$ on $9 \times 9$
- Good evaluation function difficult to build (Stern et al. 2004, Wu L. and P. Baldi 2006). Must take into account local, and global information
MoGo player

Two components

- random filling of Go-board evaluation function;
- bandit based tree search.
Monte-Carlo evaluation function

(B. Bruegmann, 1993)

- Let $p$ the position to evaluate;
- let $\pi$ a (stochastic) player;
- from $p$, $\pi$ plays against itself until the end of the game;
- the final score is then allocated to $p$;
- possibly iterate and average.
Monte-Carlo evaluation function

(B. Bruegmann, 1993)

- Let $p$ the position to evaluate;
- let $\pi$ a (stochastic) player;
- from $p$, $\pi$ plays against itself until the end of the game;
- the final score is then allocated to $p$;
- possibly iterate and average.

→ score
Tree based search: Outline

- Exploration-Exploitation: from alpha-beta to UCT
- Extensions to UCT
- Algorithmic issues
Tree based search

Minimax
We want to approximate the min-max value of the position. Not necessarily the best strategy: we could try to model the opponent.
Alpha-Beta algorithm

Alpha-Beta computes the minimax value in a tree (exact given an evaluation function).
Game as a multi-armed bandit

- each position is a bandit;
- each move is an arm;
- play the best move $\leftrightarrow$ maximize the reward.

UCT in Go

MoGo was the first Go program to use UCT (July 2006).
UCB algorithm (P. Auer et al.) 2002

- let $\hat{X}_i$ the empirical average rewards for $ith$ arm;
- let $T_i$ the number of trials for arm $i$;
- let $T = \sum_i T_i$

Then iteratively:
- if one arm has not been played, play it;
- else, play the arm maximizing $\hat{X}_i + \sqrt{2\frac{\log T}{T_i}}$. 

Sylvain Gelly¹, Yizao Wang¹,²

Exploration and Exploitation in Go: UCT for Monte-Carlo Go
UCB and UCT algorithms

UCT algorithm (L. Kocsis and C. Szepesvari. 2006)

- start from the root;
- until stopping criterion (e.g. the end of the game):
  - choose a move according to UCB;
  - update the position.
- score the game;
- update all visited nodes with this score (without discount).
Efficient memory management

Tree management after CrazyStone (R. Coulom 2006).
- Stop as soon as UCT gets an unseen position;
- add this node to the tree;
- evaluate the position.
Game of Go and Computer-Go
Evaluation function
Tree based search
Conclusion
Exploration-Exploitation: from alpha-beta to UCT
Nodes with small number of trials

Exploration and Exploitation in Go: UCT for Monte-Carlo Go
**Value of a position/move**

\[
value(move) = value(position_t + move)
\]

\[
value(position) = \frac{1}{T} \sum_i T_i value(move_i)
\]

\[
value(position) \rightarrow value(bestMove)
\]

**Why it is efficient compared to alpha-beta?**

- Alpha-Beta never reconsider a cut; dangerous with random non accurate evaluation function;
- \(value(node) \rightarrow max(node)\) as confidence increases.
- Efficient tree exploration
  - breadth first search;
  - move ordering efficiently managed;
  - asymmetric growing;
- Anytime.
Improvements

Specific extensions required

- from asymptotic quality to efficient move selection;
- \# trials < \# moves;
- arms are not independent.
Improvements in UCT when \( \# \) trials \( \approx \) nb arms

**First Play Urgency**

Starting with playing all arms is not optimal; Let \( c \) a default constant. Let \( X_i' \) such that

- \( X_i' = \hat{X}_i + \sqrt{2 \frac{\log T_i}{T_i}} \) if \( T_i > 0 \);
- \( X_i' = c \) if \( T_i = 0 \);

Choose the highest \( X_i' \).

Empirically \( c \approx 1 \rightarrow +50 \) ELO.
Exploiting dependencies

Share information between arms

There is no independence between arms vertically and horizontally. The goal is to improve the performance when $T$ is small.

- Average results from neighbor moves (add a term $\frac{1}{|N_i|} \sum_{j \in N_i} \hat{X}_j$);
- use results from ancestors: set a $c_i$ for each move according to $\hat{X}_i$ of its grandfather.
MoGo Curriculum Vitae

- **July 2006**: first participation in tournaments: 1650 ELO on CGOS (Computer Go Server) (9x9);
- **Aug. 2006 (beg.)**: ranked best program on CGOS: 1920 ELO;
- **Aug. 2006 (end)**: MoGo reached 2000 ELO;
- **Oct. 2006**: MoGo won the 2 KGS tournaments (9x9 and 13x13);
- **Nov. 2006**: MoGo won the 2 KGS tournaments (9x9 and 13x13);
- **Nov. 2006**: MoGo reached 2200 ELO on CGOS;
- **Dec. 2006**: MoGo 2nd on KGS formal tournament (19x19).
Conclusion & Perspectives

Conclusion

- MoGo first Go program using UCT
- Specific adaptation of UCT improving non asymptotic behavior
- Algorithm issues: parallelization of UCT (12000 nodes/second, 400000 nodes/move)

Perspectives

- Shifting towards exploitation for $19 \times 19$ Go boards
- Exploiting arm dependencies
- Using a mixture of evaluation functions

Sylvain Gelly¹, Yizao Wang¹,²

Exploration and Exploitation in Go: UCT for Monte-Carlo Go
Thank you

- Come to see the poster
- play against MoGo on KGS

Sylvain Gelly¹, Yizao Wang¹,²

Exploration and Exploitation in Go: UCT for Monte-Carlo Go