Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search

Paper by Rémi Coulom, CG 2006

Presented by Markus Enzenberger.
Go Seminar, University of Alberta.

June 20, 2006

Introduction

- 9×9 Go less complex than chess, but:

Introduction

- 9×9 Go less complex than chess, but:
  - Difficulty of creating a static position evaluator
  - No easy way to use quiescence search
  - Most positions are very dynamic

Introduction

- 9×9 Go less complex than chess, but:
  - Difficulty of creating a static position evaluator
  - No easy way to use quiescence search
  - Most positions are very dynamic
- Monte-Carlo evaluation as an alternative

Introduction

- 9×9 Go less complex than chess, but:
  - Difficulty of creating a static position evaluator
  - No easy way to use quiescence search
  - Most positions are very dynamic
- Monte-Carlo evaluation as an alternative
  - Choose random actions; average outcomes
  - Can be improved with tree search
    - Pruning methods ($\rightarrow$ Bouzy)
    - Methods with better asymptotic behaviour for MDPs ($\rightarrow$ Chang, Fu, Marcus)

Introduction

- 9×9 Go less complex than chess, but:
  - Difficulty of creating a static position evaluator
  - No easy way to use quiescence search
  - Most positions are very dynamic
- Monte-Carlo evaluation as an alternative
  - Choose random actions; average outcomes
  - Can be improved with tree search
    - Pruning methods ($\to$ Bouzy)
    - Methods with better asymptotic behaviour for MDPs ($\to$ Chang, Fu, Marcus)
- This paper presents a new algorithm for combining Monte-Carlo evaluation with tree search

Algorithm Structure

- Simulations produce tree stored in memory

Algorithm Structure

- Simulations produce tree stored in memory
  - Nodes store average and variance of final score
  - Store only nodes with parent visited more than once
  - Nodes with visit counts above a threshold are internal nodes

Algorithm Structure

- Simulations produce tree stored in memory
  - Nodes store average and variance of final score
  - Store only nodes with parent visited more than once
  - Nodes with visit counts above a threshold are internal nodes
- Sampling of positions

Algorithm Structure

- Simulations produce tree stored in memory
  - Nodes store average and variance of final score
  - Store only nodes with parent visited more than once
  - Nodes with visit counts above a threshold are internal nodes
- Sampling of positions
  - Internal nodes according to value (selectivity)
  - Other positions according to urgency

Algorithm Structure

- Simulations produce tree stored in memory
  - Nodes store average and variance of final score
  - Store only nodes with parent visited more than once
  - Nodes with visit counts above a threshold are internal nodes
- Sampling of positions
  - Internal nodes according to value (selectivity)
  - Other positions according to urgency
- Approach similar to algorithm of Chang, Fu, Marcus
  Advantages over Bouzy's method:

Algorithm Structure

- Simulations produce tree stored in memory
  - Nodes store average and variance of final score
  - Store only nodes with parent visited more than once
  - Nodes with visit counts above a threshold are internal nodes
- Sampling of positions
  - Internal nodes according to value (selectivity)
  - Other positions according to urgency
- Approach similar to algorithm of Chang, Fu, Marcus
  Advantages over Bouzy's method:
  - Anytime algorithm
  - Converges to optimal move

Selectivity

- Allocate simulations at every node
- Search good looking moves deeper; bad moves less

Background

- Most selectivity algorithms in Monte-Carlo rely on central limit theorem (error σ^2/N)

Background

- Most selectivity algorithms in Monte-Carlo rely on central limit theorem (error σ^2/N)
  $\longrightarrow$ but in tree search probabilities are altered

Background

- Most selectivity algorithms in Monte-Carlo rely on central limit theorem (error σ^2/N)
  $\longrightarrow$ but in tree search probabilities are altered
- Don't let sampling go to zero

Background

- Most selectivity algorithms in Monte-Carlo rely on central limit theorem (error σ^2/N)
  $\longrightarrow$ but in tree search probabilities are altered
- Don't let sampling go to zero
  - *n*-armed bandit problems
    minimize selection of non-optimal moves during simulations

Background

- Most selectivity algorithms in Monte-Carlo rely on central limit theorem (error σ^2/N)
  $\longrightarrow$ but in tree search probabilities are altered
- Don't let sampling go to zero
  - *n*-armed bandit problems
    minimize selection of non-optimal moves during simulations
    $\longrightarrow$ not required in tree search

Background

- Most selectivity algorithms in Monte-Carlo rely on central limit theorem (error σ^2/N)
  $\longrightarrow$ but in tree search probabilities are altered
- Don't let sampling go to zero
  - *n*-armed bandit problems
    minimize selection of non-optimal moves during simulations
    $\longrightarrow$ not required in tree search
  - Discrete stochastic optimization
    optimize final decision

Background

- Most selectivity algorithms in Monte-Carlo rely on central limit theorem (error σ^2/N)
  $\longrightarrow$ but in tree search probabilities are altered
- Don't let sampling go to zero
  - *n*-armed bandit problems
    minimize selection of non-optimal moves during simulations
    $\longrightarrow$ not required in tree search
  - Discrete stochastic optimization
    optimize final decision
    $\longrightarrow$ only wanted for root node in tree search

Crazy Stone's Algorithm

- Objective: obtain accurate backed-up value.

Crazy Stone's Algorithm

- Objective: obtain accurate backed-up value.
- Sample according to probability of being better than current best move

Crazy Stone's Algorithm

- Objective: obtain accurate backed-up value.
- Sample according to probability of being better than current best move

$$u_i = \exp(-2.4\frac{\mu_0 - \mu_i}{\sqrt{2(\sigma_0^i + \sigma_i^i)}}) + \epsilon_i$$

  - Assume Gaussian distributions
  - Resembles Boltzmann distributions often used in *n*-armed bandit problems

- ϵ_i is an empirical constant to avoid that probability goes to zero

$$\epsilon_i = \frac{0.1 + 2^{-i} + a_i}{N}$$

- ϵ_i is an empirical constant to avoid that probability goes to zero

$$\epsilon_i = \frac{0.1 + 2^{-i} + a_i}{N}$$

$$a_i = \begin{cases} 1 & : \text{move } i \text{ is atari} \\ 0 & : \text{otherwise} \end{cases}$$

Increase sampling of atari moves, because they require a follow-up move and their true value might be underestimated

- Backup method for external nodes

$$\mu = \Sigma / S$$

$$\sigma^2 = \frac{\Sigma_2 - S\mu^2 + 4P^2}{S+1}$$

P points on board; Σ_2 sum squared values of this node; Σ sum values; S number simulations
High prior variance for rarely explored nodes

Backup Method

- What is the best backup method for internal nodes?

Backup Method

- What is the best backup method for internal nodes?
- Mean

Backup Method

- What is the best backup method for internal nodes?
- Mean
  - Use same method as for external nodes
  - Best move dominates in the long term
  - Simple, but inefficient
  - For random variables, expected maximum is not sum of values weighted by probability to be the best
  - Underestimates node value

Backup Method

- What is the best backup method for internal nodes?
- Mean
  - Use same method as for external nodes
  - Best move dominates in the long term
  - Simple, but inefficient
  - For random variables, expected maximum is not sum of values weighted by probability to be the best
  - Underestimates node value
- Max

Backup Method

- What is the best backup method for internal nodes?
- Mean
  - Use same method as for external nodes
  - Best move dominates in the long term
  - Simple, but inefficient
  - For random variables, expected maximum is not sum of values weighted by probability to be the best
  - Underestimates node value
- Max
  - Low number of simulations $\rightarrow$ noisy values
  - Move with best value is likely to be most lucky move
  - Overestimates node value

- Update probability distributions

- Update probability distributions
  - Assumes independence of distributions
    $\longrightarrow$ not true in tree search

- Update probability distributions
  - Assumes independence of distributions
    $\longrightarrow$ not true in tree search
- Robust max

- Update probability distributions
  - Assumes independence of distributions
    $\longrightarrow$ not true in tree search
- Robust max
  - Backup value with maximum number of games
  - Most of the time it is move with best value
  - Otherwise better not backup less searched move

- Update probability distributions
  - Assumes independence of distributions
    → not true in tree search
- Robust max
  - Backup value with maximum number of games
  - Most of the time it is move with best value
  - Otherwise better not backup less searched move
- Mix

- Update probability distributions
  - Assumes independence of distributions
    $\longrightarrow$ not true in tree search
- Robust max
  - Backup value with maximum number of games
  - Most of the time it is move with best value
  - Otherwise better not backup less searched move
- Mix
  - Linear combination between Robust max and Mean
  - Refinements for situations where mean is superior to max

Mix algorithm

```
float MeanWeight = 2 * WIDTH * HEIGHT;
if (Simulations > 16 * WIDTH * HEIGHT)
 MeanWeight *= float(Simulations) / (16 * WIDTH * HEIGHT);

float Value = MeanValue;
if (tGames[1] && tGames[0])
{
 float tAveragedValue[2];
 for (int i = 2; --i >= 0;)
  tAveragedValue[i] =
   (tGames[i] * tValue[i] + MeanWeight * Value) / (tGames[i] + MeanWeight);

 if (tGames[0] < tGames[1])
 {
  if (tValue[1] > Value)
   Value = tAveragedValue[1];
  else if (tValue[0] < Value)
   Value = tAveragedValue[0];
 }
 else
  Value = tAveragedValue[0];
}
else
 Value = tValue[0].
return Value;
```

Fig. 1. Value-backup algorithm. The size of the goban is given by “WIDTH” and “HEIGHT”. “Simulations” is the number of random games that were run from this node, and “MeanValue” the mean value of these simulations. Move number 0 is the best move, move number 1 is the second best move or the move with the highest number of games, if it is different from the two best moves. tValue[i] are the the backed-up values of the moves and tGames[i] their numbers of simulations.

Value Backup

Backup experiments

- Run number of simulations (S) for different backup methods
- Compute mean error and mean squared error
- "True value":
  value with best backup method at $2S$ simulations

Results

Simulations	Mean $\sqrt{\langle\delta^2\rangle}$	Mean $\langle\delta\rangle$	Max $\sqrt{\langle\delta^2\rangle}$	Max $\langle\delta\rangle$	Robust Max $\sqrt{\langle\delta^2\rangle}$	Robust Max $\langle\delta\rangle$	Mix $\sqrt{\langle\delta^2\rangle}$	Mix $\langle\delta\rangle$
128	6.44	-3.32	41.70	37.00	39.60	35.30	5.29	-1.43
256	7.17	-4.78	25.00	22.00	23.60	20.90	4.72	-1.89
512	7.56	-5.84	14.90	12.70	13.90	11.90	4.08	-1.70
1,024	6.26	-4.86	9.48	7.91	8.82	7.41	3.06	0.13
2,048	4.38	-3.15	6.72	5.37	6.11	4.91	2.63	0.77
4,096	2.84	-1.55	4.48	3.33	3.94	2.91	2.05	0.69
8,192	2.23	-0.62	2.78	1.47	2.42	1.07	1.85	0.32
16,384	2.34	-0.57	2.45	0.01	2.40	-0.30	2.10	-0.19
32,768	2.15	-0.52	2.19	0.10	2.26	-0.12	1.93	-0.02
65,536	2.03	-0.50	1.83	0.23	1.88	0.01	1.70	0.01
131,072	2.07	-0.54	1.80	0.25	1.94	0.02	1.80	-0.02
262,144	1.85	-0.58	1.49	0.25	1.51	0.07	1.39	-0.02

Uncertainty Backup

- Use data of backup experiments
- Approximate with

$$\frac{\sigma^2}{\min(500, S)}$$

Random Simulations

- Simplest method:
  select moves uniformly at random, if legal and not eye-filling

Random Simulations

- Simplest method:
  select moves uniformly at random, if legal and not eye-filling
- Improvement:
  probability distribution uses domain specific knowledge
  assign urgencies and select moves with probability proportional to urgency

Urgencies

- Illegal or completely surrounded by own stones (not atari)
  $\longrightarrow$ urgency $= 0$ (final)
  (full false-eye detection to slow)

Urgencies

- Illegal or completely surrounded by own stones (not atari)
  $\longrightarrow$ urgency $= 0$ (final)
  (full false-eye detection to slow)
- otherwise: $\longrightarrow$ urgency $= 1$

Urgencies

- Illegal or completely surrounded by own stones (not atari)
  $\longrightarrow$ urgency $= 0$ (final)
  (full false-eye detection to slow)
- otherwise: $\longrightarrow$ urgency $= 1$
- only liberty of own block (size S)
  $\longrightarrow$ urgency $+= 1000 \times S$
  unless hopeless extension
  - at most one contiguous empty point
  - no opponent string in atari
  - no own string not atari

Urgencies

- only liberty of opponent block
  $\longrightarrow$ urgency $+= 10000 \times S$
  unless hopeless extension

Urgencies

- only liberty of opponent block
  $\rightarrow$ urgency $+= 10000 \times S$
  unless hopeless extension
  if opponent block adjacent to own block in atari
  $\rightarrow$ urgency $+= 100000 \times S$

Useless Moves

Some moves are, if selected, replaced by a different move

- Create eye: if surrounded by own stones of a single string and one empty point, which is liberty of the same string
  $\rightarrow$ play on this liberty

Useless Moves

Some moves are, if selected, replaced by a different move

- Create eye: if surrounded by own stones of a single string and one empty point, which is liberty of the same string
  $\rightarrow$ play on this liberty
- Surrounded by opponent stones and one empty point, move does not change atari status of opponent blocks
  $\rightarrow$ play on empty point

- Move creates string in atari (more than one stone)
  - if urgency ≥ 1000:
    $\rightarrow$ urgency $= 1$, repeat move selection
  - string had a string in atari adjacent
    $\rightarrow$ capture string in atari instead
  - string had two liberties before move
    $\rightarrow$ play other liberty instead

Performance

- Athlon 3400+
- 64-bit GCC 4.0.3
- 17000 games/sec on empty 9×9

Game Results

Player	Opponent	Winning Rate	Komi
CrazyStone (5 min / game)	Indigo 2005 (8 min / game)	61% (±4.9)	6.5
Indigo 2005 (8 min / game)	GNU Go 3.6 (level 10)	28% (±4.4)	6.5
CrazyStone (4 min / Game)	GNU Go 3.6 (level 10)	25% (±4.3)	7.5
CrazyStone (8 min / Game)	GNU Go 3.6 (level 10)	32% (±4.7)	7.5
CrazyStone (16 min / Game)	GNU Go 3.6 (level 10)	36% (±4.8)	7.5

Table 2. Match results, with 95% confidence intervals

Conclusion

- New efficient backup method for MC tree search

Conclusion

- New efficient backup method for MC tree search
- Good perfomance in 9×9 Go tournaments

Conclusion

- New efficient backup method for MC tree search
- Good perfomance in 9×9 Go tournaments
- Future research

Conclusion

- New efficient backup method for MC tree search
- Good perfomance in 9×9 Go tournaments
- Future research
  - Improve selectivity and uncertainty backup

Conclusion

- New efficient backup method for MC tree search
- Good perfomance in 9×9 Go tournaments
- Future research
  - Improve selectivity and uncertainty backup
  - Use stochastic optimization algorithms at root node

Conclusion

- New efficient backup method for MC tree search
- Good perfomance in 9×9 Go tournaments
- Future research
  - Improve selectivity and uncertainty backup
  - Use stochastic optimization algorithms at root node
  - Overcome tactical weaknesses by game specific knowledge in random simulations

Conclusion

- New efficient backup method for MC tree search
- Good perfomance in 9×9 Go tournaments
- Future research
  - Improve selectivity and uncertainty backup
  - Use stochastic optimization algorithms at root node
  - Overcome tactical weaknesses by game specific knowledge in random simulations
  - Scale to 19×19
    Use high-level goals instead of global search