

Artificial Intelligence for deterministic 2 players games with UCT

Pierre Gueth

2 octobre 2013

UCT and other AI

Tree search algorithm

- Brute force approach
- Exhaustive search of every game possible
- High complexity
- Works only for small game (tic-tac-toe)

MinMax

- Standard AI for chess
- Tree search with limited depth
- Need score function
- Constant time consumption
- Doesn't work for open game like Go

UCT

- Collaboration between INRIA and Taiwan university [LEE2009]
- Used in MOGO, first Go AI to beat professional player in 2008
- Monte Carlo / Tree search hybrid
- No need for score function, only game completion
- Flexible time consumption and high scalability

UCT and other AI

Tree search algorithm

- Brute force approach
- Exhaustive search of every game possible
- High complexity
- Works only for small game (tic-tac-toe)

MinMax

- Standard AI for chess
- Tree search with limited depth
- Need score function
- Constant time consumption
- Doesn't work for open game like Go

UCT

- Collaboration between INRIA and Taiwan university [LEE2009]
- Used in MOGO, first Go AI to beat professional player in 2008
- Monte Carlo / Tree search hybrid
- **No need for score function**, only game completion
- **Flexible time consumption and high scalability**

Which game can UCT play ?

- Two players alternating
- Deterministic game states
- No cyclic game states
- Each game should end by a win of either player or a draw

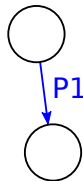
Which game can UCT play ?



- Two players alternating
- Deterministic game states
- No cyclic game states
- Each game should end by a win of either player or a draw

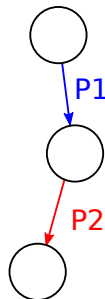
Which game can UCT play ?

- Two players alternating
- Deterministic game states
- No cyclic game states
- Each game should end by a win of either player or a draw



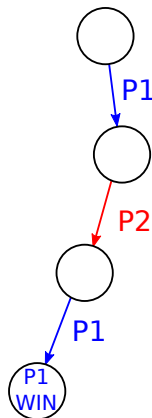
Which game can UCT play ?

- Two players alternating
- Deterministic game states
- No cyclic game states
- Each game should end by a win of either player or a draw



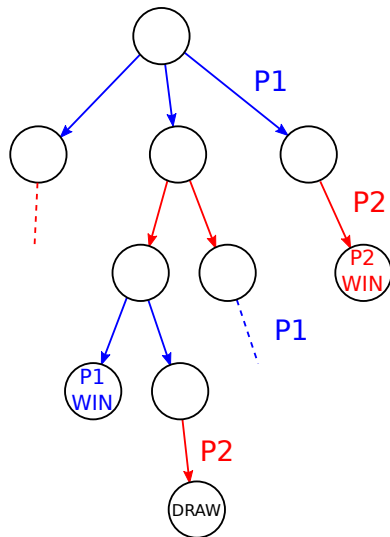
Which game can UCT play ?

- Two players alternating
- Deterministic game states
- No cyclic game states
- Each game should end by a win of either player or a draw



Which game can UCT play ?

- Two players alternating
- Deterministic game states
- No cyclic game states
- Each game should end by a win of either player or a draw



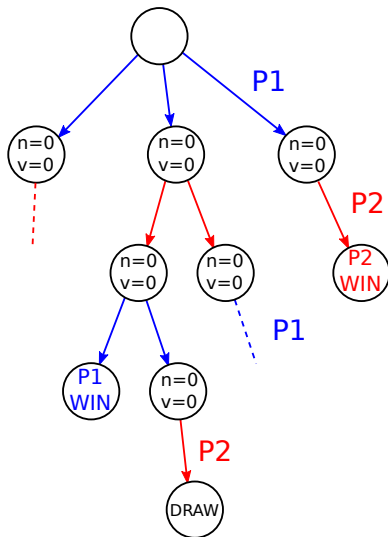
UCT principle

- ❶ Play random moves for both players until game reaches its end
- ❷ Update upstream game state value
- ❸ Select initials moves and go back to step 1

Each state has two variables :

- n = number of played games that pass by this state
- v = number of won games that pass by this state

The **probability of winning** by reaching each state is estimated by v/n



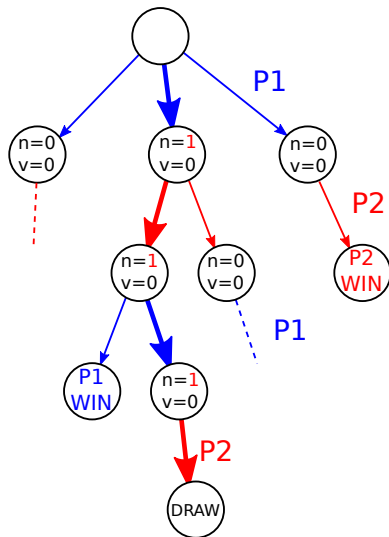
UCT principle

- 1 Play random moves for both players until game reaches its end
- 2 Update upstream game state value
- 3 Select initials moves and go back to step 1

Each state has two variables :

- n = number of played games that pass by this state
- v = number of won games that pass by this state

The **probability of winning** by reaching each state is estimated by v/n



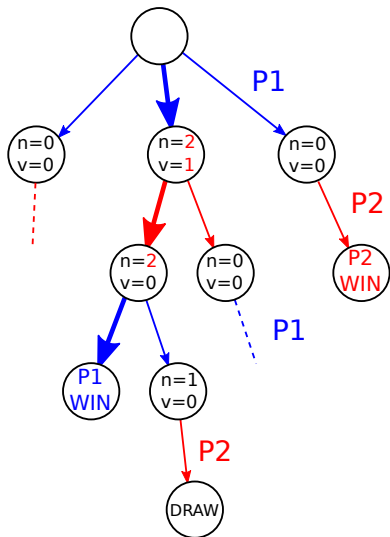
UCT principle

- 1 Play random moves for both players until game reaches its end
- 2 Update upstream game state value
- 3 Select initials moves and go back to step 1

Each state has two variables :

- n = number of played games that pass by this state
- v = number of won games that pass by this state

The **probability of winning** by reaching each state is estimated by v/n



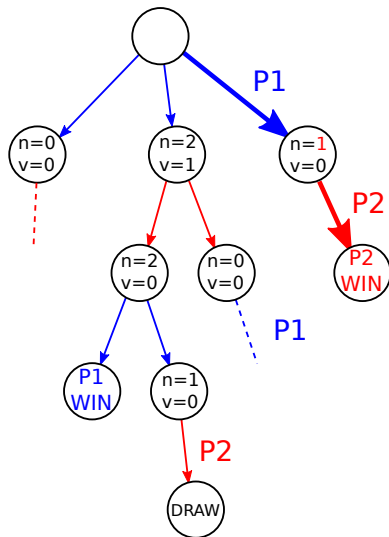
UCT principle

- 1 Play random moves for both players until game reaches its end
- 2 Update upstream game state value
- 3 Select initials moves and go back to step 1

Each state has two variables :

- n = number of played games that pass by this state
- v = number of won games that pass by this state

The **probability of winning** by reaching each state is estimated by v/n



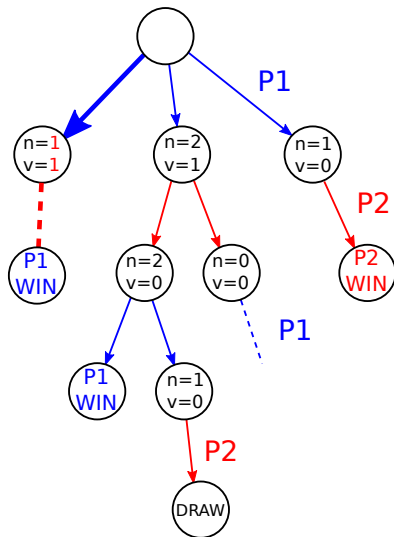
UCT principle

- 1 Play random moves for both players until game reaches its end
- 2 Update upstream game state value
- 3 Select initials moves and go back to step 1

Each state has two variables :

- n = number of played games that pass by this state
- v = number of won games that pass by this state

The **probability of winning** by reaching each state is estimated by v/n



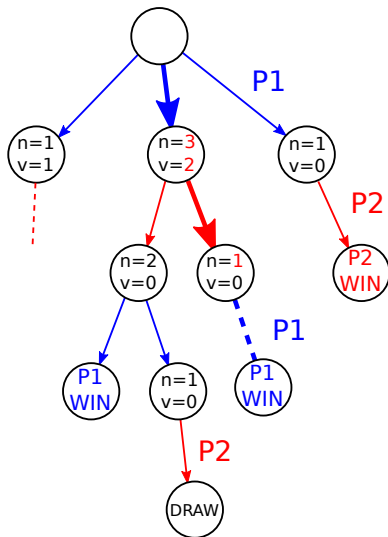
UCT principle

- ❶ Play random moves for both players until game reaches its end
- ❷ Update upstream game state value
- ❸ Select initials moves and go back to step 1

Each state has two variables :

- n = number of played games that pass by this state
- v = number of won games that pass by this state

The **probability of winning** by reaching each state is estimated by v/n



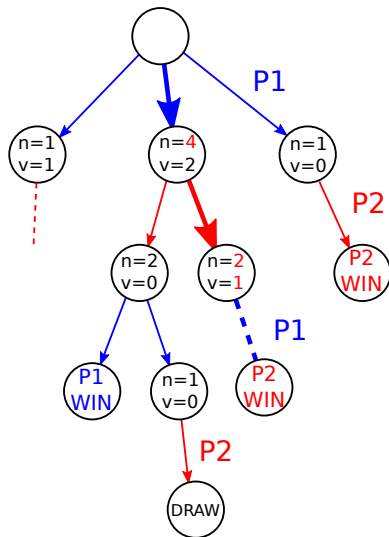
UCT principle

- ❶ Play random moves for both players until game reaches its end
- ❷ Update upstream game state value
- ❸ Select initials moves and go back to step 1

Each state has two variables :

- n = number of played games that pass by this state
- v = number of won games that pass by this state

The **probability of winning** by reaching each state is estimated by v/n



Simulated games selection

- **Exploration**

Each move has the same probability of being selected.
This ensures that every strategies are considered.

- **Exploitation**

The probability of selecting a move is proportional to the winning probability.

This ensures that the computation power is spent on the strategy that is more likely to win.

Note that when n is small the winning probability is estimated with a large relative error.

Simulated games selection

- **Exploration**

Each move has the same probability of being selected.
This ensures that every strategies are considered.

- **Exploitation**

The probability of selecting a move is proportional to the winning probability.

This ensures that the computation power is spent on the strategy that is more likely to win.

Note that when n is small the winning probability is estimated with a large relative error.

Trade-off

- For small values of n , use the exploration strategy.
- For large values of n , use the exploitation strategy.

C++ implementation

- I implemented UCT in C++
- Use Qt, CMake but mainly based on standard library
- ~ 3000 C++ lines

[https ://github.com/elcerdo/uct](https://github.com/elcerdo/uct)

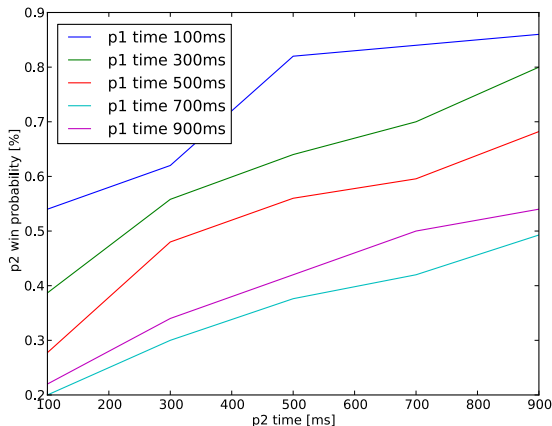
Easy implementation of new games

```
class Board {  
public:  
    virtual ~Board() =0;  
  
    virtual Board *deepcopy() const =0;  
    virtual Move *parse_move_string(Token player, const char *string) const =0;  
    virtual void print() const =0;  
    virtual bool is_move_valid(const Move &move) const =0;  
    virtual Moves get_possible_moves(Token player) const =0;  
    virtual void play_move(const Move &move) =0;  
    virtual bool play_random_move(Token player) =0;  
    virtual Token check_for_win() const =0;  
    virtual Token play_random_game(Token next_player);  
};
```

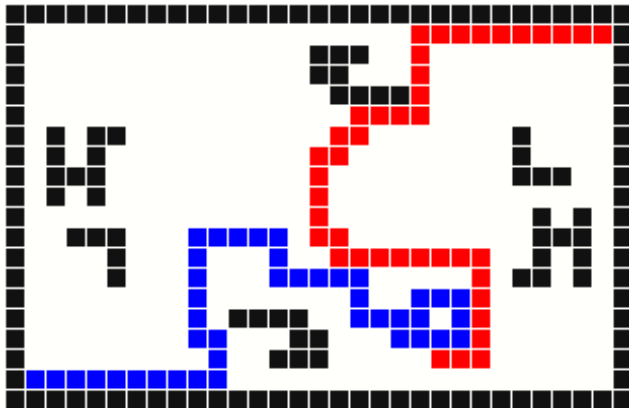
- Derive abstract board class
- Implement a few virtual methods
- That's it!

UCT handles the hard part of choosing the best move for you.

AI strength vs. computation time : Othello



AI win probability increase linearly with computation time for computation time below 1s.



UCT finished 50th over more than 3000 contestants.

Demonstration
Can you beat the AI ?

Thanks for your attention

Pierre Gueth

Wide technical and academical knowledge

- Classe préparatoire PT
- ENS Cachan
aggrégation physique appliquée / EEA
- Thèse au laboratoire CREATIS (Université Lyon I)
Imagerie médicale ultrasonore
Estimation de mouvement
- Post doc au Centre Léon Bérard (Lyon)
Simulation Monte-Carlo Protonthérapie (GATE)
Imagerie γ -prompt

Numerous computer science side project

- Freesiege, Blocks, ...
- Monte-Carlo, UCT, ...
- Autojump, cluster submission tools