Monte Carlo Tree Search (MCTS)

Wojciech Jaśkowski, Marcin Szubert

Zakład Inteligentnych Systemów Wspomagania Decyzji
Instytut Informatyki
Politechnika Poznańska

21 października 2014

Monte Carlo Tree Search (MCTS)

Wojciech Jaśkowski, Marcin Szubert

Zakład Inteligentnych Systemów Wspomagania Decyzji
Instytut Informatyki
Politechnika Poznańska

21 października 2014

1. Dzień dobry Państwu
2. Chciałbym opowiedzieć o metodach MCTS, które zyskaly ostatnio duza popularnosc szczegolnie w swiecie zwiazanym ze sztuczna inteligencja w grach.

Presentation Outline

Presentation Outline

1. Sequential Decision Making
2. Games
3. Game Tree Search
4. Monte Carlo Tree Search
5. Extensions & Domains
6. Conclusions

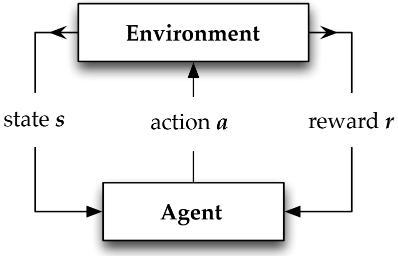
1. Plan mojej prezentacji jest nastepujacy
2. W pierwszej kolejnosci powiem o ogolnej klasie problemow dla ktorych mozna stosowac metody MCTS
3. Sa to problemy zwiazane z sekwencyjnym podejmowaniem decyzji
4. Powiem o tym jakie sa typowe podejscia do rozwiazywania takich problemow i jak na ich tle prezentuja sie metody MCTS
5. Nastepnie przejde do specyficznego przypadku problemow SPD jakim sa gry, w szczegolnosci gry kombinatoryczne
6. W szczegolnosci powiem o grze Go, ktora stanowi wyzwanie dla metod sztucznej inteligencji i przyczynila sie do
7. **Przechodząc do meritum**
8. W dalszej kolejnosci przedstawie typowe podejscia polegajace na przeszukiwaniu drzewa gry z ktorych rozwinela sie metoda MCTS

Presentation Outline

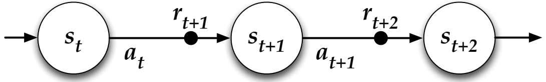
Presentation Outline

1. Sequential Decision Making
2. Games
3. Game Tree Search
4. Monte Carlo Tree Search
5. Extensions & Domains
6. Conclusions

Sequential Decision Making

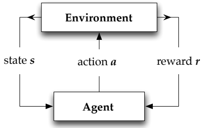


- The agent and the environment interact at discrete time steps:

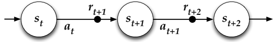


- Goal: select actions that maximize the sum of future rewards, when the consequences of those actions may not be revealed for many steps.

Sequential Decision Making



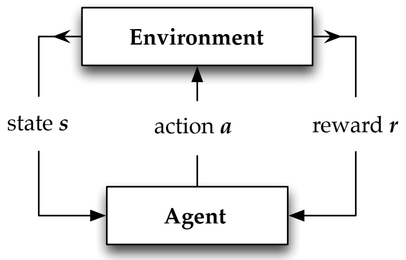
- The agent and the environment interact at discrete time steps:



- Goal: select actions that maximize the sum of future rewards, when the consequences of those actions may not be revealed for many steps.

1. W takich problemach **autonomiczny i inteligentny** agent zostaje umieszczony w nieznanym środowisku i uczy się **podejmowania następujących po sobie decyzji**
2. Uczenie **w oparciu o interakcje** zachodzące w **dyskretnych jednostkach czasu**
3. Rysunek przedstawia **typowy scenariusz** takich interakcji
4. **W pierwszym kroku** agent obserwuje bieżący stan środowiska
5. Na podstawie **dotychczas wypracowanej strategii** i bieżącego stanu środowiska, agent wykonuje akcje.
6. **W wyniku akcji** środowisko **zmienia swój stan**, a agent może otrzymać od środowiska nagrodę pełniącą rolę **wzmocnienia** (**potencjalnie negatywne**)
7. Wśród **przykładów** sekwencyjnych problemów decyzyjnych można wyróżnić m.in. kierowanie samochodem, szeregowanie zadań czy też **grę w szachy**.
8. 1.5 min =¿ 10.5 min

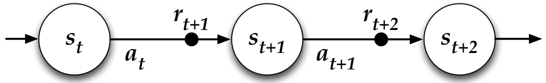
Sequential Decision Making



Example 1: Playing Chess

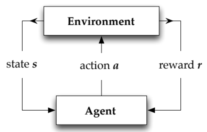
agent	player
environment	game, opponent
state	state of the board
action	game move
rewards	outcomes

- The agent and the environment interact at discrete time steps:



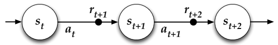
- Goal: select actions that maximize the sum of future rewards, when the consequences of those actions may not be revealed for many steps.

Sequential Decision Making



Example 1: Playing Chess
agent player
environment game, opponent
state state of the board
action game move
rewards outcomes

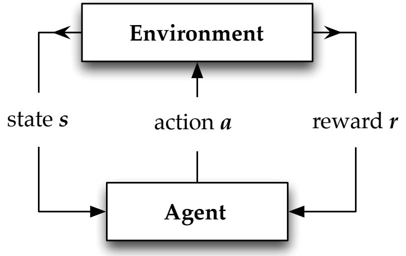
- The agent and the environment interact at discrete time steps:



- Goal: select actions that maximize the sum of future rewards, when the consequences of those actions may not be revealed for many steps.

1. W takich problemach **autonomiczny i inteligentny** agent zostaje umieszczony w nieznanym środowisku i uczy się **podejmowania następujących po sobie decyzji**
2. Uczenie **w oparciu o interakcje** zachodzące w **dyskretnych jednostkach czasu**
3. Rysunek przedstawia **typowy scenariusz** takich interakcji
4. **W pierwszym kroku** agent obserwuje bieżący stan środowiska
5. Na podstawie **dotychczas wypracowanej strategii** i bieżącego stanu środowiska, agent wykonuje akcje.
6. **W wyniku akcji** środowisko **zmienia swój stan**, a agent może otrzymać od środowiska nagrodę pełniącą rolę **wzmocnienia** (**potencjalnie negatywne**)
7. Wśród **przykładów** sekwencyjnych problemów decyzyjnych można wyróżnić m.in. kierowanie samochodem, szeregowanie zadań czy też **grę w szachy**.
8. 1.5 min =¿ 10.5 min

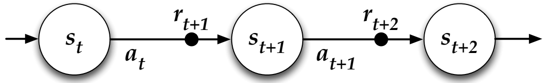
Sequential Decision Making



Example 2: Packing a Knapsack

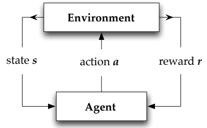
agent	camper
environment	knapsack
state	remaining capacity
action	packing an item
rewards	values of items

- The agent and the environment interact at discrete time steps:



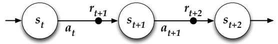
- Goal: select actions that maximize the sum of future rewards, when the consequences of those actions may not be revealed for many steps.

Sequential Decision Making



Example 2: Packing a Knapsack
agent camper
environment knapsack
state remaining capacity
action packing an item
rewards values of items

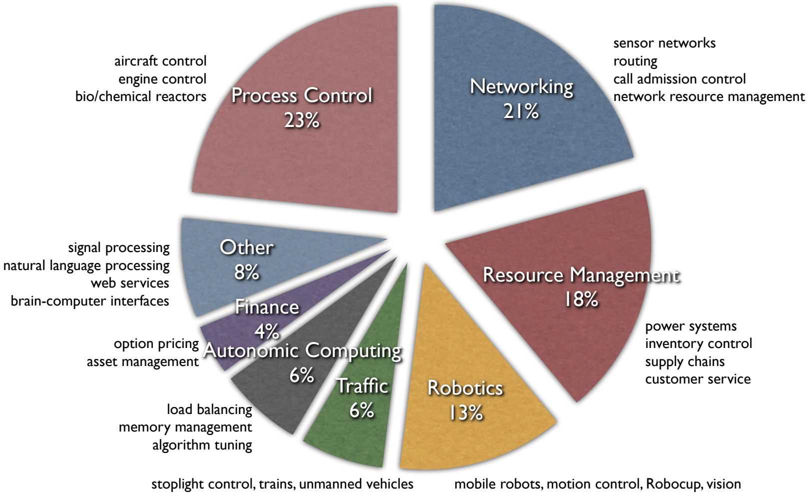
- The agent and the environment interact at discrete time steps:



- Goal: select actions that maximize the sum of future rewards, when the consequences of those actions may not be revealed for many steps.

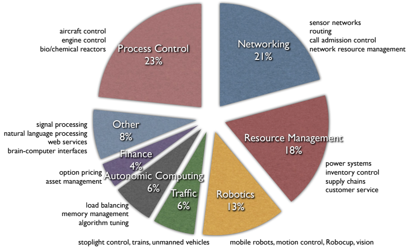
1. W takich problemach **autonomiczny i inteligentny** agent zostaje umieszczony w nieznanym środowisku i uczy się **podejmowania następujących po sobie decyzji**
2. Uczenie **w oparciu o interakcje** zachodzące w **dyskretnych jednostkach czasu**
3. Rysunek przedstawia **typowy scenariusz** takich interakcji
4. **W pierwszym kroku** agent obserwuje bieżący stan środowiska
5. Na podstawie **dotychczas wypracowanej strategii** i bieżącego stanu środowiska, agent wykonuje akcje.
6. **W wyniku akcji** środowisko **zmienia swój stan**, a agent może otrzymać od środowiska nagrodę pełniącą rolę **wzmocnienia** (**potencjalnie negatywne**)
7. Wśród **przykładów** sekwencyjnych problemów decyzyjnych można wyróżnić m.in. kierowanie samochodem, szeregowanie zadań czy też **grę w szachy**.
8. 1.5 min =¿ 10.5 min

Sequential Decision Problems



Sutton, R. (2009). Multidisciplinary Symposium on Reinforcement Learning.

Sequential Decision Problems



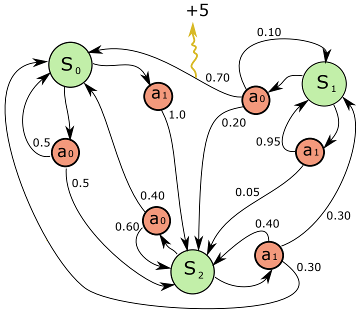
Sutton, R. (2009). Multidisciplinary Symposium on Reinforcement Learning.

1. Na zakończenie krótkiego opisu uczenia się ze wzmocnieniem
2. Warto zwrócić uwagę, że **przedstawione metody** znajdują **praktyczne zastosowanie** w wielu dziedzinach
3. Zastosowania ilustruje diagram przedstawiony na konferencji ICML przez Richarda Suttona, ktory jest jednym z najwiekszych autorytetow
4. Zastosowania są **różnorodne**:
5. Począwszy od **sterowania silnikami/samolotami** przez **zarządzanie zasobami sieciowymi, routing** i **zarządzanie zasobami w systemach produkcyjnych** aż po naturalne **zastosowania w robotyce**

Markov Decision Process (MDP)

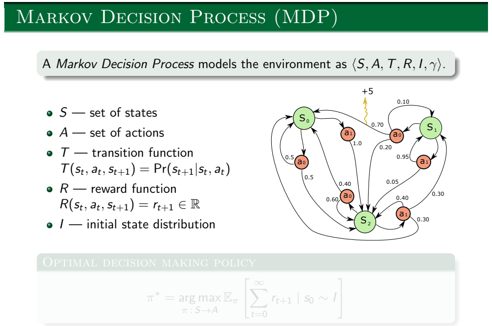
A *Markov Decision Process* models the environment as $\langle S, A, T, R, I, \gamma \rangle$.

- S — set of states
- A — set of actions
- T — transition function
  $T(s_t, a_t, s_{t+1}) = \Pr(s_{t+1} | s_t, a_t)$
- R — reward function
  $R(s_t, a_t, s_{t+1}) = r_{t+1} \in \mathbb{R}$
- I — initial state distribution



Optimal decision making policy

$$\pi^* = \underset{\pi : S \to A}{\arg\max} \, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} r_{t+1} \mid s_0 \sim I \right]$$

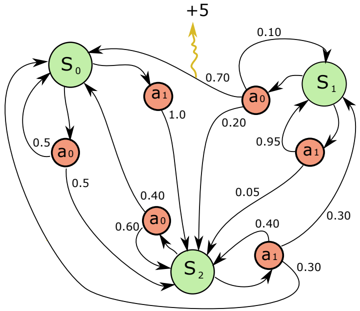


1. **Proces decyzyjny Markowa** definiowany jest jako **szóstka uporządkowana** złożona z następujących elementów:
2. zbiór stanów — bieżący stan całkowicie opisuje proces - zawiera wszelkie informacje potrzebne do podjęcia decyzji
3. zbiór akcji, który może być **funkcją bieżącego stanu środowiska**
4. funkcja tranzycji określająca **prawdopodobieństwo przejścia między stanami**, **własność Markowa**
5. funkcja nagrody zwraca **rzeczywiste wartości wzmocnienia** po tranzycji
6. rozkład I określa od którego stanu **rozpoczną się interakcje**
7. współczynnik dyskontowania okresla jak krotkowzroczny powinien byc uczen
8. **Rysunek** przedstawia **graf tranzycji** przykładowego procesu decyzyjnego Markowa
9. **W formalizmie MDP** mozna zdefiniowac optymalna strategie podejmowania decyzji
10. Jest to taka **funkcja pi przyporzadkowujaca kazdemu stanowi akcje**, ktora maksymalizuje...
11. **Wiedzac jak powinna wygladac optymalna strategia**

Markov Decision Process (MDP)

A *Markov Decision Process* models the environment as $\langle S, A, T, R, I, \gamma \rangle$.

- S — set of states
- A — set of actions
- T — transition function
  $T(s_t, a_t, s_{t+1}) = \Pr(s_{t+1} | s_t, a_t)$
- R — reward function
  $R(s_t, a_t, s_{t+1}) = r_{t+1} \in \mathbb{R}$
- I — initial state distribution



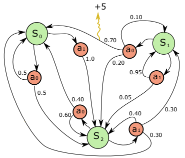
Optimal decision making policy

$$\pi^* = \underset{\pi : S \to A}{\arg\max}\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} r_{t+1} \mid s_0 \sim I \right]$$

Markov Decision Process (MDP)

A *Markov Decision Process* models the environment as $\langle S, A, T, R, I, \gamma \rangle$.

- S — set of states
- A — set of actions
- T — transition function $T(s_t, a_t, s_{t+1}) = \Pr(s_{t+1} | s_t, a_t)$
- R — reward function $R(s_t, a_t, s_{t+1}) = r_{t+1} \in \mathbb{R}$
- I — initial state distribution



Optimal decision making policy

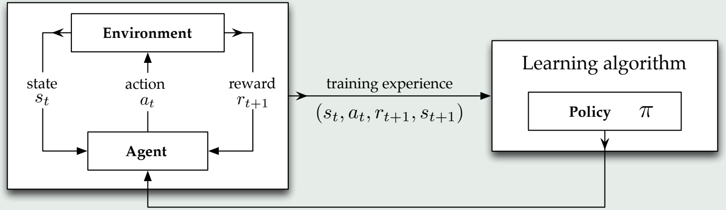
$$\pi^* = \underset{\pi : S \to A}{\arg\max}\, \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} r_{t+1} \mid s_0 \sim I \right]$$

1. **Proces decyzyjny Markowa** definiowany jest jako **szóstka uporządkowana** złożona z następujących elementów:
2. zbiór stanów — bieżący stan całkowicie opisuje proces - zawiera wszelkie informacje potrzebne do podjęcia decyzji
3. zbiór akcji, który może być **funkcją bieżącego stanu środowiska**
4. funkcja tranzycji określająca **prawdopodobieństwo przejścia między stanami**, **własność Markowa**
5. funkcja nagrody zwraca **rzeczywiste wartości wzmocnienia** po tranzycji
6. rozkład I określa od którego stanu **rozpoczną się interakcje**
7. współczynnik dyskontowania okresla jak krotkowzroczny powinien byc uczen
8. **Rysunek** przedstawia **graf tranzycji** przykładowego procesu decyzyjnego Markowa
9. **W formalizmu MDP** mozna zdefiniowac optymalna strategie podejmowania decyzji
10. Jest to taka **funkcja pi przyporzadkowujaca kazdemu stanowi akcje**, ktora maksymalizuje...
11. **Wiedzac jak powinna wygladac optymalna strategia**

Two approaches to policy optimization

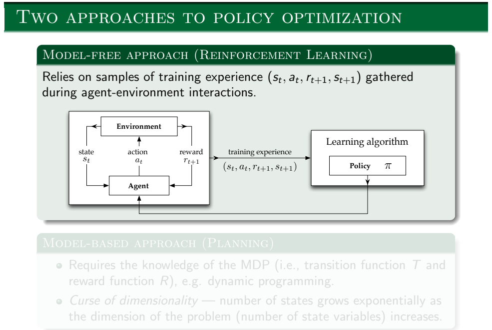
Model-free approach (Reinforcement Learning)

Relies on samples of training experience $(s_t, a_t, r_{t+1}, s_{t+1})$ gathered during agent-environment interactions.



Model-based approach (Planning)

- Requires the knowledge of the MDP (i.e., transition function T and reward function R), e.g. dynamic programming.
- *Curse of dimensionality* — number of states grows exponentially as the dimension of the problem (number of state variables) increases.


Two approaches to policy optimization
Model-free approach (Reinforcement Learning)
Relies on samples of training experience $(s_t, a_t, r_{t+1}, s_{t+1})$ gathered during agent-environment interactions.
Model-based approach (Planning)
Requires the knowledge of the MDP (i.e., transition function T and reward function R), e.g. dynamic programming.
Curse of dimensionality — number of states grows exponentially as the dimension of the problem (number of state variables) increases.

1. **Pytanie brzmi zatem** jak znaleźć optymalną strategię? - Istnieją 2 główne podejścia
2. **Pierwsze z nich** Model-Free nie wymaga znajomosci srodowiska, lecz zakłada że tego środowiska można się **nauczyć**
3. Oparte jest na **próbkach doświadczenia** uczącego **zebranych podczas interakcji**
4. Takie podejście **odpowiada naturalnemu procesowi uczenia się metodą prób i błędów**
5. **W roli algorytmów uczących** przetwarzajacych i wnioskujacych na podstawie zdobywane doświadczenie - wykorzystuje się dwa rodzaje metod
6. Metody wykorzystujące funkcję wartości jako **krok pośredni** do strategii
7. **Drugie podejście** Model-Based - wymaga znajomosci modelu środowiska (MDP)
8. **Znając funkcję tranzycji i nagrody** można **zaplanować** sekwencję przyszłych akcji.
9. Do tego rodzaju metod zalicza się zarówno zaproponowane przez **Bellmana DP**, jak i właśnie metody **MCTS** - istnieja miedzy nimi jednak **dwie duze roznice** o ktorych zaraz bede mowil
10. Roznice te powoduja, ze o ile MCTS mozna stosowac w zlozonych MDP, tak DP **cierpia z powodu klatwy wymiarowosci**
11. **Termin ten zaproponowal Bellman** na okreslenie problemow wynikajacych z wykladniczego wzrostu rozmiaru przestrzeni stanow wraz ze wzrostem liczby zmiennych ten stan opisujacych

Two approaches to policy optimization

Model-free approach (Reinforcement Learning)

Relies on samples of training experience $(s_t, a_t, r_{t+1}, s_{t+1})$ gathered during agent-environment interactions.

- *Direct policy search* methods, e.g. evolutionary algorithms.
- Methods based on *value functions*, e.g. temporal difference learning.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} r_{t+k+1} | s_t = s\right]$$

Model-based approach (Planning)

- Requires the knowledge of the MDP (i.e., transition function T and reward function R), e.g. dynamic programming.
- *Curse of dimensionality* — number of states grows exponentially as the dimension of the problem (number of state variables) increases.

Two approaches to policy optimization

Model-free approach (Reinforcement Learning)

Relies on samples of training experience $(s_t, a_t, r_{t+1}, s_{t+1})$ gathered during agent-environment interactions.

- *Direct policy search* methods, e.g. evolutionary algorithms.
- Methods based on *value functions*, e.g. temporal difference learning.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} r_{t+k+1} | s_t = s\right]$$

Model-based approach (Planning)

- Requires the knowledge of the MDP (i.e., transition function T and reward function R), e.g. dynamic programming.
- *Curse of dimensionality* — number of states grows exponentially as the dimension of the problem (number of state variables) increases.

1. **Pytanie brzmi zatem** jak znaleźć optymalną strategię? - Istnieją 2 główne podejścia
2. **Pierwsze z nich** Model-Free nie wymaga znajomosci srodowiska, lecz zakłada że tego środowiska można się **nauczyć**
3. Oparte jest na **próbkach doświadczenia** uczącego **zebranych podczas interakcji**
4. Takie podejście **odpowiada naturalnemu procesowi uczenia się metodą prób i błędów**
5. **W roli algorytmów uczących** przetwarzajacych i wnioskujacych na podstawie zdobywane doświadczenie - wykorzystuje się dwa rodzaje metod
6. Metody wykorzystujące funkcję wartości jako **krok pośredni** do strategii
7. **Drugie podejście** Model-Based - wymaga znajomosci modelu środowiska (MDP)
8. **Znając funkcję tranzycji i nagrody** można **zaplanować** sekwencję przyszłych akcji.
9. Do tego rodzaju metod zalicza się zarówno zaproponowane przez **Bellmana DP**, jak i właśnie metody **MCTS** - istnieja miedzy nimi jednak **dwie duze roznice** o ktorych zaraz bede mowil
10. Roznice te powoduja, ze o ile MCTS mozna stosowac w zlozonych MDP, tak DP **cierpia z powodu klatwy wymiarowosci**
11. **Termin ten zaproponowal Bellman** na okreslenie problemow wynikajacych z wykladniczego wzrostu rozmiaru przestrzeni stanow wraz ze wzrostem liczby zmiennych ten stan opisujacych

Two approaches to policy optimization

Model-free approach (Reinforcement Learning)

Relies on samples of training experience $(s_t, a_t, r_{t+1}, s_{t+1})$ gathered during agent-environment interactions.

- *Direct policy search* methods, e.g. evolutionary algorithms.
- Methods based on *value functions*, e.g. temporal difference learning.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} r_{t+k+1} | s_t = s\right]$$

Model-based approach (Planning)

- Requires the knowledge of the MDP (i.e., transition function T and reward function R), e.g. dynamic programming.
- *Curse of dimensionality* — number of states grows exponentially as the dimension of the problem (number of state variables) increases.

Two approaches to policy optimization

Model-free approach (Reinforcement Learning)

Relies on samples of training experience $(s_t, a_t, r_{t+1}, s_{t+1})$ gathered during agent-environment interactions.

- *Direct policy search* methods, e.g. evolutionary algorithms.
- Methods based on *value functions*, e.g. temporal difference learning.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} r_{t+k+1} | s_t = s\right]$$

Model-based approach (Planning)

- Requires the knowledge of the MDP (i.e., transition function T and reward function R), e.g. dynamic programming.
- *Curse of dimensionality* — number of states grows exponentially as the dimension of the problem (number of state variables) increases.

1. **Pytanie brzmi zatem** jak znaleźć optymalną strategię? - Istnieją 2 główne podejścia
2. **Pierwsze z nich** Model-Free nie wymaga znajomosci srodowiska, lecz zakłada że tego środowiska można się **nauczyć**
3. Oparte jest na **próbkach doświadczenia** uczącego **zebranych podczas interakcji**
4. Takie podejście **odpowiada naturalnemu procesowi uczenia się metodą prób i błędów**
5. **W roli algorytmów uczących** przetwarzajacych i wnioskujacych na podstawie zdobywane doświadczenie - wykorzystuje się dwa rodzaje metod
6. Metody wykorzystujące funkcję wartości jako **krok pośredni** do strategii
7. **Drugie podejście** Model-Based - wymaga znajomosci modelu środowiska (MDP)
8. **Znając funkcję tranzycji i nagrody** można **zaplanować** sekwencję przyszłych akcji.
9. Do tego rodzaju metod zalicza się zarówno zaproponowane przez **Bellmana DP**, jak i właśnie metody **MCTS** - istnieja miedzy nimi jednak **dwie duze roznice** o ktorych zaraz bede mowil
10. Roznice te powoduja, ze o ile MCTS mozna stosowac w zlozonych MDP, tak DP **cierpia z powodu klatwy wymiarowosci**
11. **Termin ten zaproponowal Bellman** na okreslenie problemow wynikajacych z wykladniczego wzrostu rozmiaru przestrzeni stanow wraz ze wzrostem liczby zmiennych ten stan opisujacych

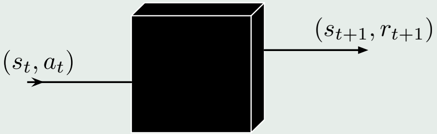
Planning in Large MDPs

Classical Planning Assumptions

- MDP is given explicitly by tables of rewards and transition probabilities.
- The output is a total mapping π from states to actions.

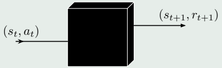
Sample-Based / Simulation-Based Planning

- A natural way to specify a large MDP is to use a *generative model*, or *simulator*, of the MDP.



- Planning algorithms can employ a simulator to generate sample sequences of experience.

Planning in Large MDPs
Classical Planning Assumptions
- MDP is given explicitly by tables of rewards and transition probabilities.
- The output is a total mapping π from states to actions.
Sample-Based / Simulation-Based Planning
- A natural way to specify a large MDP is to use a *generative model*, or *simulator*, of the MDP.



- Planning algorithms can employ a simulator to generate sample sequences of experience.

1. Aby możliwe było planowanie w złożonych MDP o dużych przestrzeniach stanów - a to umożliwiają metody MCTS, należy zrewidowować założenia jakie przyjmują klasyczne metody
2. W odniesieniu do dwoch zalozen klasycznych metod planowania, pokaze dwie kluczowe koncepcyjne roznice miedzy klasycznymi metodami typu DP a nowoczesnymi metodami MCTS
3. Po pierwsze

Planning in Large MDPs

Classical Planning Assumptions

- MDP is given explicitly by tables of rewards and transition probabilities.
- The output is a total mapping π from states to actions.

Online Planning (Search)

- *Offline algorithms* have to find a policy for the entire state space.
- *Online algorithms* focus on computing good local policies at each decision step during the execution.

Offline Planning	Policy construction						Policy execution	
Online Planning								

Planning in Large MDPs

Classical Planning Assumptions

- MDP is given explicitly by tables of rewards and transition probabilities.
- The output is a total mapping π from states to actions.

Online Planning (Search)

- *Offline algorithms* have to find a policy for the entire state space.
- *Online algorithms* focus on computing good local policies at each decision step during the execution.

Offline Planning Policy construction Policy execution

Online Planning

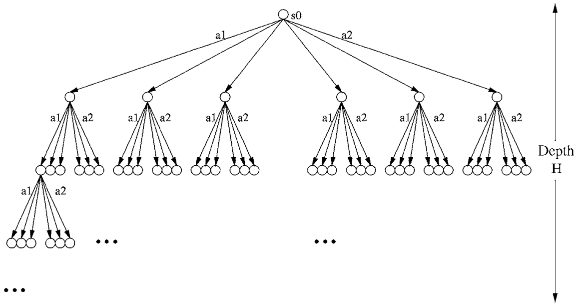
1. Aby możliwe było planowanie w złożonych MDP o dużych przestrzeniach stanów - a to umożliwiają metody MCTS, należy zrewidowować założenia jakie przyjmują klasyczne metody
2. W odniesieniu do dwoch zalozen klasycznych metod planowania, pokaze dwie kluczowe koncepcyjne roznice miedzy klasycznymi metodami typu DP a nowoczesnymi metodami MCTS
3. Po pierwsze

Planning in Large MDPs

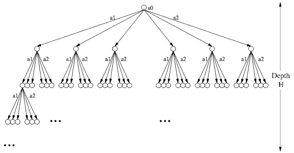
Online Sample-Based Planning (Search)

Given current state s_0:

- use the simulator to draw samples for many state-action pairs,
- organize sampled states into a look-ahead search tree rooted at s_0,
- compute the next action to take from s_0.



A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes, Kearns M., Mansour Y., Ng, A.Y., 2002

Planning in Large MDPs
Online Sample-Based Planning (Search)
Given current state s_0:
- use the simulator to draw samples for many state-action pairs,
- organize sampled states into a look-ahead search tree rooted at s_0,
- compute the next action to take from s_0.

A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes, Kearns M., Mansour Y., Ng, A.Y., 2002

Presentation Outline

1. Sequential Decision Making
2. Games
3. Game Tree Search
4. Monte Carlo Tree Search
5. Extensions & Domains
6. Conclusions

Presentation Outline

1. Sequential Decision Making
2. Games
3. Game Tree Search
4. Monte Carlo Tree Search
5. Extensions & Domains
6. Conclusions

Games

Game Theory

Game theory extends decision making to situations in which multiple agents interact. A game can be defined as a set of established rules that allows the interaction of players to produce specified outcomes.

Combinatorial Games

Games with two players that are zero-sum, perfect information, deterministic, discrete and sequential.

	deterministic	*chance*
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	battleships	bridge, poker, scrabble, nuclear war

Games

Game Theory

Game theory extends decision making to situations in which multiple agents interact. A game can be defined as a set of established rules that allows the interaction of players to produce specified outcomes.

Combinatorial Games

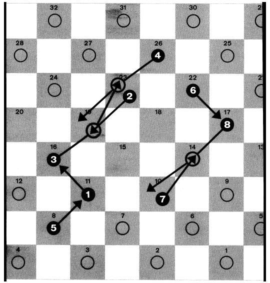
Games with two players that are zero-sum, perfect information, deterministic, discrete and sequential.

	deterministic	*chance*
perfect information	chess, checkers, go, othello	backgammon, monopoly
imperfect information	battleships	bridge, poker, scrabble, nuclear war

Why Combinatorial Games?

> Games provide a convenient vehicle for the development of learning procedures as contrasted with a problem taken from life, since many of the complications of detail are removed.
>
> *Some Studies in Machine Learning Using the Game of Checkers, Samuel A., 1959*

- Closed micro-worlds with simple rules.
- Clear benchmarks of performance both between different programs and against human intelligence.
- Excellent testbeds for AI:
  - Chess is the *Drosophila* of AI.
  - Games are to AI as grand prix racing is to automobile design.

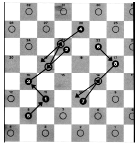


Why Combinatorial Games?

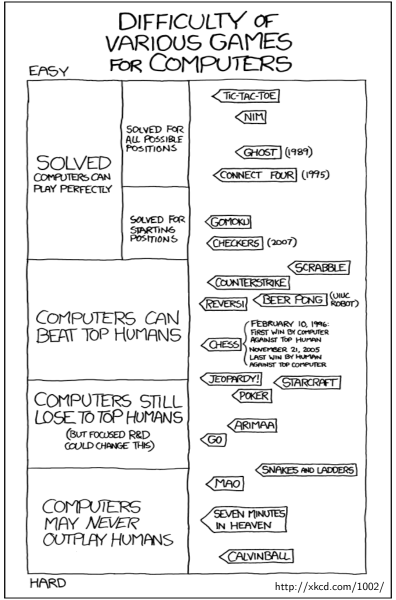
Games provide a convenient vehicle for the development of learning procedures as contrasted with a problem taken from life, since many of the complications of detail are removed.

Some Studies in Machine Learning Using the Game of Checkers, Samuel A., 1959

- Closed micro-worlds with simple rules.
- Clear benchmarks of performance both between different programs and against human intelligence.
- Excellent testbeds for AI:
  - Chess is the *Drosophila* of AI.
  - Games are to AI as grand prix racing is to automobile design.

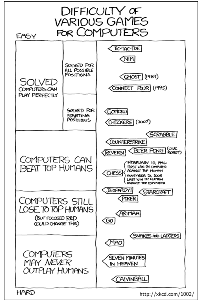


Game Complexity



Game	State-space complexity	Branching factor
Tic-tac-toe	10^3	4
Connect 4	10^{13}	4
Checkers	10^{20}	2.8
Othello	10^{28}	10
Chess	10^{50}	35
Go	10^{171}	250

Game Complexity

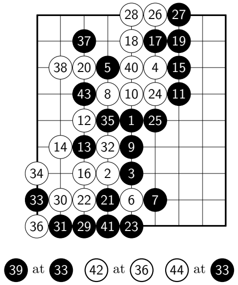


Game	State-space complexity	Branching factor
Tic-tac-toe	10^3	4
Connect 4	10^{13}	4
Checkers	10^{20}	2.8
Othello	10^{28}	10
Chess	10^{50}	35
Go	10^{171}	250

1. Warto zwrocic jednak uwage, ze wiele gier rozwazanych w historii AI nie stanowi juz wyzwania
2. O trudnosci danej gry swiadcza m.in. dwie cechy - liczba stanow i branching factor
3. Ze wszystkich gier przedstawionych na slajdzie, najwiekszym wyzwaniem jest gra Go

The Trouble with Go

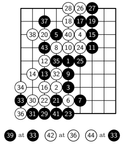
- Enormous combinatorial complexity (large state and action space).
- Long term influence of moves (delayed reward, temporal credit assignment).
- No good heuristics for evaluating a state (in contrast to chess or checkers).



> Many real-world, sequential decision making problems are difficult for exactly the same reasons. Therefore, progress in Go can lead to advances that are significant beyond computer Go and may ultimately contribute to advancing the field of AI as a whole.
>
> *The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions,*
> *Gelly S., Kocsis L, Shoenauer M., Sebag M., Silver D., Szepesvari C., 2012*

The Trouble with Go

- Enormous combinatorial complexity (large state and action space).
- Long term influence of moves (delayed reward, temporal credit assignment).
- No good heuristics for evaluating a state (in contrast to chess or checkers).



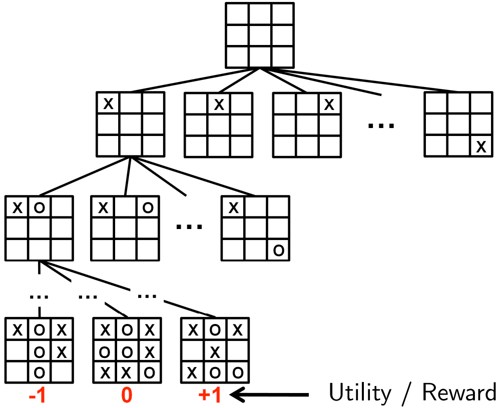
Many real-world, sequential decision making problems are difficult for exactly the same reasons. Therefore, progress in Go can lead to advances that are significant beyond computer Go and may ultimately contribute to advancing the field of AI as a whole.

The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions, Gelly S., Kocsis L, Shoenauer M., Sebag M., Silver D., Szepesvari C., 2012

Presentation Outline

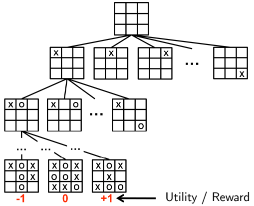
Odniose sie teraz do algorytmow planowania online zaprezentowanych w punkcie 1 i pokaze przyklady takich algorytmow stosowane w kontekscie gier do przeszukiwania drzewa gry

Game Tree



- Game tree organizes the possible future action sequences into a tree.
- The root represents the initial state, while each other node represents non-empty finite action sequence of two players.

Game Tree

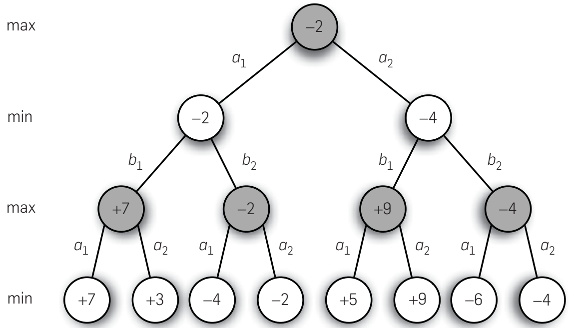


- Game tree organizes the possible future action sequences into a tree.
- The root represents the initial state, while each other node represents non-empty finite action sequence of two players.

1. Drzewo gry jest szczegolnym przypadkiem drzewa stanow w sekwencyjnym procesie decyzyjnym
2. Przedstawia ono mozliwe sekwencje przyszlych decyzji rozpoczynajace sie w biezacym stanie srodowiska
3. **Gry dwuosobowe**
4. Liście niosą ze sobą nagrody
5. Ten sam stan moze wystepowac w wielu miejscach w grze - DAG

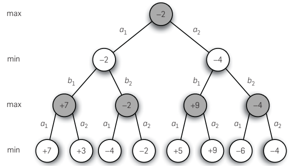
Minimax Search for Online Planning

- Expand a complete search tree, until terminal states have been reached and their utilities can be computed.
- Go back up from the leaves towards the current state of the game.
  - At each *min* node, back up the *worst* value among children.
  - At each *max* node, back up the *best* value among children.



Minimax Search for Online Planning

- Expand a complete search tree, until terminal states have been reached and their utilities can be computed.
- Go back up from the leaves towards the current state of the game.
  - At each *min* node, back up the *worst* value among children.
  - At each *max* node, back up the *best* value among children.



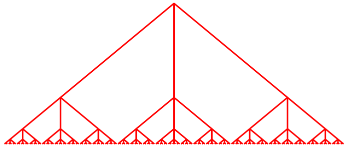
1. The algorithm called the Minimax algorithm was invented by Von Neumann and Morgenstern in 1944, as part of game theory.
2. The root of the tree is the current board position, it is MAXs turn to play
3. MAX generates the tree as much as it can, and picks the best move assuming that MIN will also choose the moves for herself.

Dealing with Huge Trees

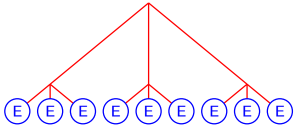
While minimax search leads to *optimal actions*, it is intractable for most interesting games; the computation time is proportional to the size of the game tree, which grows exponentially with its depth.

- Size of the full game tree — $O(b^m)$.
- Impractical to search to the end of the game.

- A subtree of limited depth.
- Heuristic evaluation function estimates values of leaf nodes.



Full tree



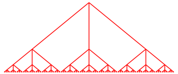
Depth limit +
Position evaluation function (E)

Dealing with Huge Trees

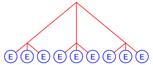
While minimax search leads to *optimal actions*, it is intractable for most interesting games; the computation time is proportional to the size of the game tree, which grows exponentially with its depth.

- Size of the full game tree — $O(b^m)$.
- Impractical to search to the end of the game.

- A subtree of limited depth.
- Heuristic evaluation function estimates values of leaf nodes.

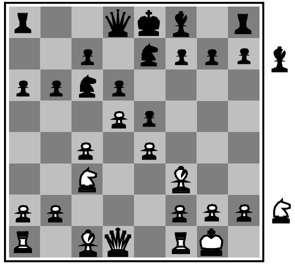


Full tree



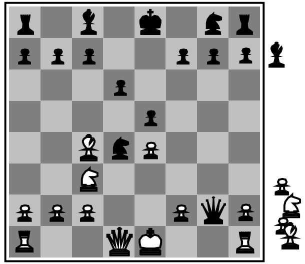
Depth limit +
Position evaluation function (E)

How to Evaluate a Game Position (State)?



Black to move

White slightly better



White to move

Black winning

- If the features of the board can be judged independently, then a good choice is a weighted linear function:

$$E(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

How to Evaluate a Game Position (State)?

Black to move

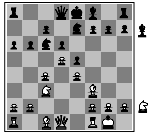
White slightly better

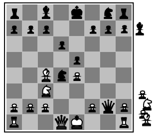
White to move

Black winning

- If the features of the board can be judged independently, then a good choice is a weighted linear function:

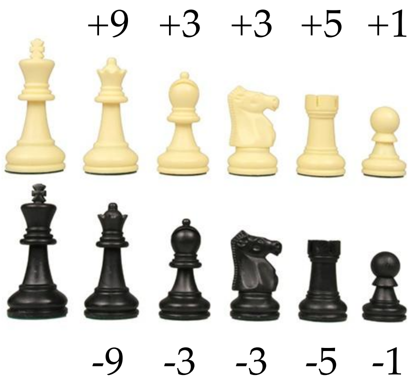
$$E(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$





How to Evaluate a Game Position (State)?

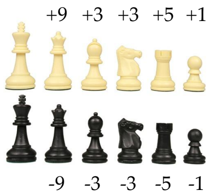
- Beginners evaluate position by giving each piece a value ...



- ... and summing up values of pieces in a given state.

How to Evaluate a Game Position (State)?

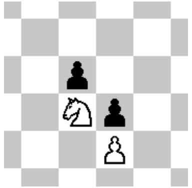
- Beginners evaluate position by giving each piece a value ...



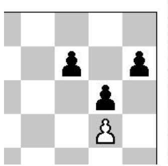
- ... and summing up values of pieces in a given state.

How to Evaluate a Game Position (State)?

- Experts evaluate position using sophisticated features, but:
  - hard to extract these features,
  - hard to quantify their weights.



knight on outpost

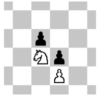


weak kingside pawn structure

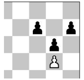
- Deep Blue employed more than 8000 features.
- Evaluation functions can be learned by e.g. temporal difference learning or evolutionary algorithms.

How to Evaluate a Game Position (State)?

- Experts evaluate position using sophisticated features, but:
  - hard to extract these features,
  - hard to quantify their weights.



knight on outpost



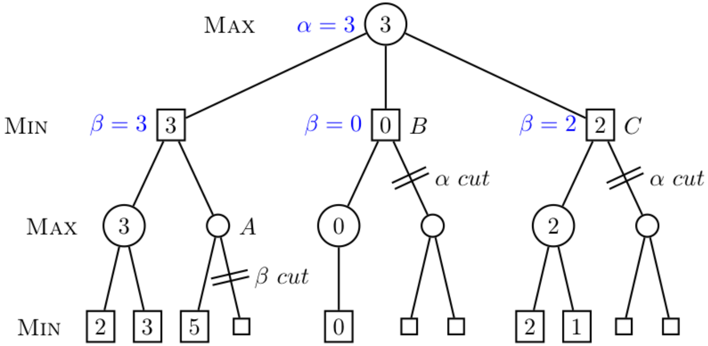
weak kingside pawn structure

- Deep Blue employed more than 8000 features.
- Evaluation functions can be learned by e.g. temporal difference learning or evolutionary algorithms.

Minimax Search Enhancements

Alpha-Beta Pruning

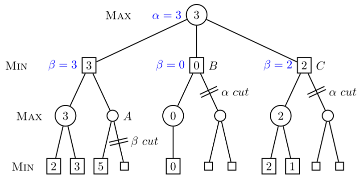
- Branch and Bound pruning of nodes outside window $[\alpha, \beta]$.
- Can double the search depth with optimal ordering.



Minimax Search Enhancements

Alpha-Beta Pruning

- Branch and Bound pruning of nodes outside window $[\alpha, \beta]$.
- Can double the search depth with optimal ordering.



1. **Kończąc temat** podstawowego algorytmu przeszukiwania drzewa gry, minimax
2. Warto wspomnieć o tym, że **zaproponowanych zostało wiele usprawnień**
3. Jednym z najpopularniejszych jest **mechanizm przycinania drzewa alpha-beta**
4. Podczas przeszukiwania drzewa utrzymywany jest przedział wartosci [alpha, beta]
5. alpha = maksymalny wynik gracza MAX
6. beta = minimalny wynik gracza MIN
7. Pozwala to odcinać gałęzie drzewa które nie wpłyną na optymalną grę
8. DZięki tym oszczędnościom możliwe jest zwiększenie głębokości przeszukiwania maksymalnie dwukrotnie
9. **Programs based on variants of minimax search with alpha-beta pruning have outperformed human world champions in chess, checkers, and othello.**
10. **Ale nie dla Go.** gdzie podstawowy problem polega na tym, ze trudno jest zaprojektowac sensowna funkcje oceny heurystycznej

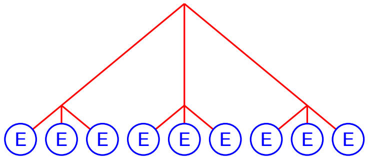
Presentation Outline

Odpowiedzia na ten problem sa wlasnie szeroko rozumiane metody Monte Carlo

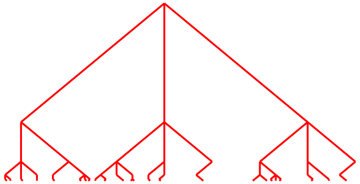
Monte Carlo Simulations

Motivation

- **problem**: evaluation functions might be hard to learn or design
- **solution 1**: construct a **playout policy** to estimate values of states
- **drawbacks**: sensitive to the choice of policy and systematic errors
- **solution 2**: Monte-Carlo adds explicit **randomization**.
  - reduce influence of systematic errors
  - distinctions between states: "easy to win" vs. "hard to win"



Classical Approach: Depth limit + Position evaluation function (E)

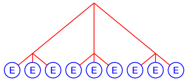


Monte Carlo Approach: simulated playouts

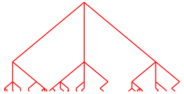
Monte Carlo Simulations

Motivation

- problem: evaluation functions might be hard to learn or design
- solution 1: construct a playout policy to estimate values of states
- drawbacks: sensitive to the choice of policy and systematic errors
- solution 2: Monte-Carlo adds explicit randomization.
  - reduce influence of systematic errors
  - distinctions between states: "easy to win" vs. "hard to win"



Classical Approach: Depth limit + Position evaluation function (E)



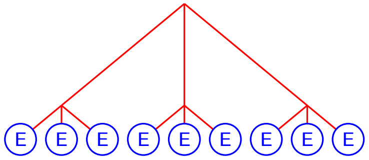
Monte Carlo Approach: simulated playouts

1. **Zanim przejde do metody MCTS**, powiem o historycznie wcześniejszych metodach wykorzystujacych tzw. symulacje Monte Carlo do ewaluacji stanu
2. **Bezposrednia motywacja** dla tych metod sa wlasnie sytuacje gdzie **klasyczne podejscie** sie nie sprawdza
3. **Alternatywa dla uzycia funkcji ewaluacji** na plytkim poziomie przeszukiwania jest skonstruowanie **Tzw. Playout Policy** i rozwiniecie drzewa do konca wg. tej strategii
4. **Użycie końcowych wyników do estymacji wartości bieżącego stanu**
5. **Jesli uzylibysmy deterministycznej strategii** i sama gra jest deterministyczna wowczas...
6. **Estymacja byłaby obciazona systematycznym bledem**
7. **Dlatego tez metody MC** opieraja sie z reguly na **losowych strategiach**, wrecz jednorodnie losowych

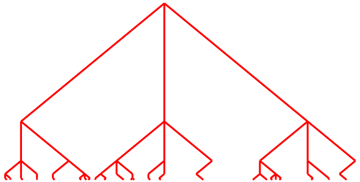
Monte Carlo Simulations

Motivation

- **problem**: evaluation functions might be hard to learn or design
- **solution 1**: construct a **playout policy** to estimate values of states
- **drawbacks**: sensitive to the choice of policy and systematic errors
- **solution 2**: Monte-Carlo adds explicit **randomization**.
  - reduce influence of systematic errors
  - distinctions between states: "easy to win" vs. "hard to win"



Classical Approach: Depth limit + Position evaluation function (E)

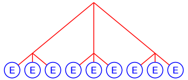


Monte Carlo Approach: simulated playouts

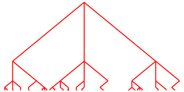
Monte Carlo Simulations

Motivation

- problem: evaluation functions might be hard to learn or design
- solution 1: construct a playout policy to estimate values of states
- drawbacks: sensitive to the choice of policy and systematic errors
- solution 2: Monte-Carlo adds explicit randomization.
  - reduce influence of systematic errors
  - distinctions between states: "easy to win" vs. "hard to win"



Classical Approach: Depth limit + Position evaluation function (E)



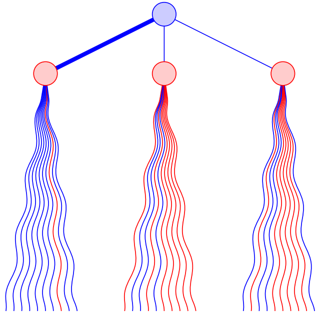
Monte Carlo Approach: simulated playouts

1. **Zanim przejde do metody MCTS**, powiem o historycznie wcześniejszych metodach wykorzystujacych tzw. symulacje Monte Carlo do ewaluacji stanu
2. **Bezposrednia motywacja** dla tych metod sa wlasnie sytuacje gdzie **klasyczne podejscie** sie nie sprawdza
3. **Alternatywa dla uzycia funkcji ewaluacji** na plytkim poziomie przeszukiwania jest skonstruowanie **Tzw. Playout Policy** i rozwiniecie drzewa do konca wg. tej strategii
4. **Użycie końcowych wyników do estymacji wartości bieżącego stanu**
5. **Jesli uzylibysmy deterministycznej strategii** i sama gra jest deterministyczna wowczas...
6. **Estymacja byłaby obciazona systematycznym bledem**
7. **Dlatego tez metody MC** opieraja sie z reguly na **losowych strategiach**, wrecz jednorodnie losowych

Flat Monte Carlo

Move Selection

- N playouts for each move — sample possible continuation using a randomized playing policy for both players.
- Pick the most valuable move — the value of the move is the average of the evaluations obtained at the end of the lines of play.

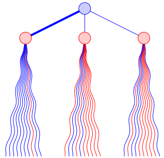


9/10 3/10 4/10

Flat Monte Carlo

Move Selection

- N playouts for each move — sample possible continuation using a randomized playing policy for both players.
- Pick the most valuable move — the value of the move is the average of the evaluations obtained at the end of the lines of play.



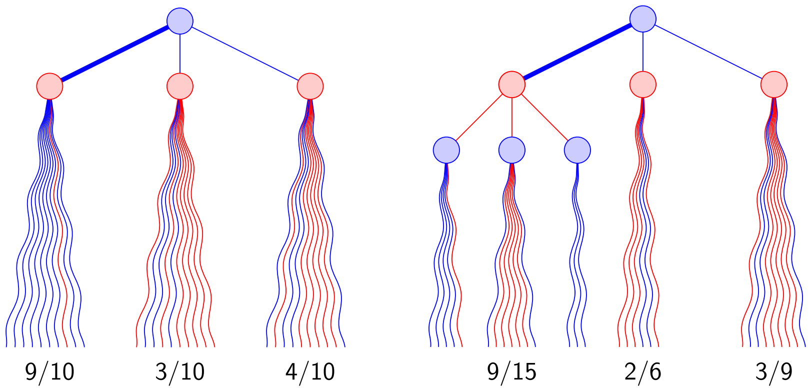
9/10 3/10 4/10

1. **Konkretnym algorytmem planowania** opartym na tym pomysle jest Flat MC

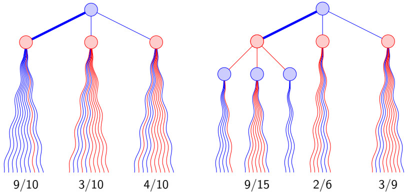
Problem with Flat Monte Carlo

Short-sighted evaluation

For instance, random simulations for a move may look good at first, but if it turns out that this move can be followed up by a killer opponent move, its evaluation may decrease when it is searched deeper.



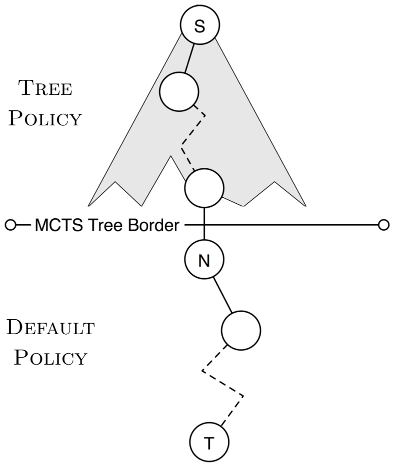
Problem with Flat Monte Carlo
Short-sighted evaluation
For instance, random simulations for a move may look good at first, but if it turns out that this move can be followed up by a killer opponent move, its evaluation may decrease when it is searched deeper.



Monte Carlo Simulations + Tree Search

Monte Carlo Tree Search

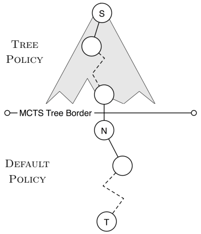
- Combines Monte-Carlo simulation with game tree search.
- Run a number of simulations and **selectively** build up a search tree.
- Gradually adapt and improve the **simulation policy**:
  - tree policy — intelligent moves,
  - default policy — random moves.
- The values of nodes are estimated by Monte Carlo simulations.



Monte Carlo Simulations + Tree Search

Monte Carlo Tree Search

- Combines Monte-Carlo simulation with game tree search.
- Run a number of simulations and **selectively** build up a search tree.
- Gradually adapt and improve the **simulation policy**:
  - tree policy — intelligent moves,
  - default policy — random moves.
- The values of nodes are estimated by Monte Carlo simulations.

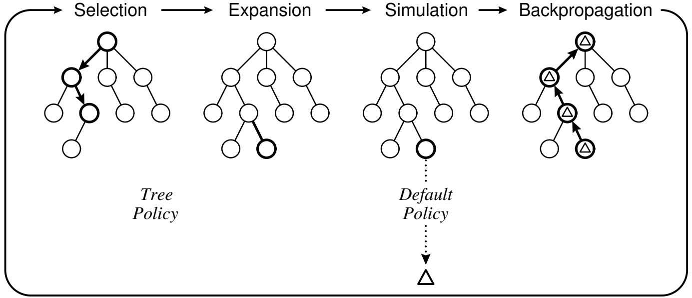


1. Monte-Carlo tree search (MCTS) combines Monte-Carlo simulation with game tree search. It proceeds by selectively growing a game tree. As in minimax search, each node in the tree corresponds to a single state of the game. However, unlike minimax search, the values of nodes (including both leaf nodes and interior nodes) are now estimated by Monte- Carlo simulation.
2. One of the key ideas of MCTS is to gradually adapt and improve this simulation policy. As more simulations are run, the game tree grows larger and the Monte-Carlo values at the nodes become more accurate, providing a great deal of useful information that can be used to bias the policy to- wards selecting actions which lead to child nodes with high values.
3. **The algorithm progressively builds a partial game tree, guided by the results of previ- ous exploration of that tree.**
4. **The tree is used to estimate the values of moves, with these estimates (particularly those for the most promising moves) becoming more accurate as the tree is built.**

THE FAMILY OF MCTS ALGORITHMS

Algorithm 1 General MCTS approach.

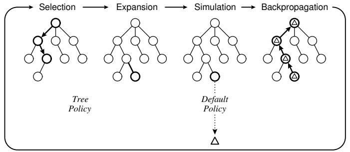
function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))



THE FAMILY OF MCTS ALGORITHMS

Algorithm 1 General MCTS approach.

function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))

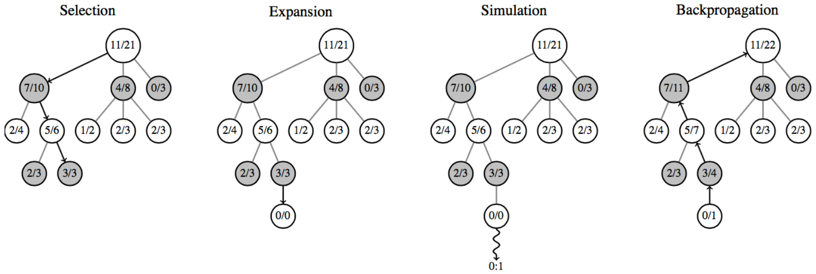


1. Family of MCTS algorithms
2. Zmienne - tree policy, default policy

The Family of MCTS Algorithms

Algorithm 1 General MCTS approach.

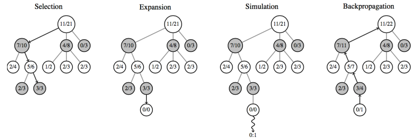
function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))



The Family of MCTS Algorithms

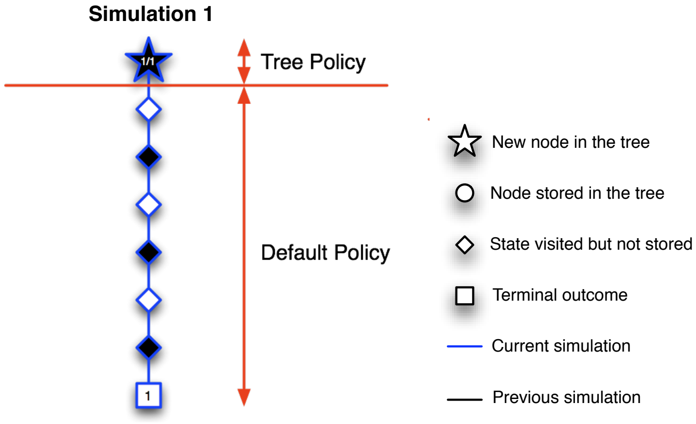
Algorithm 1 General MCTS approach.

function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))

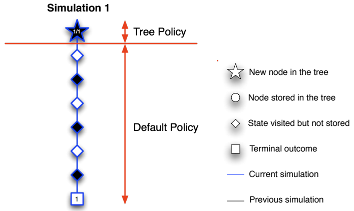


1. Family of MCTS algorithms
2. Zmienne - tree policy, default policy

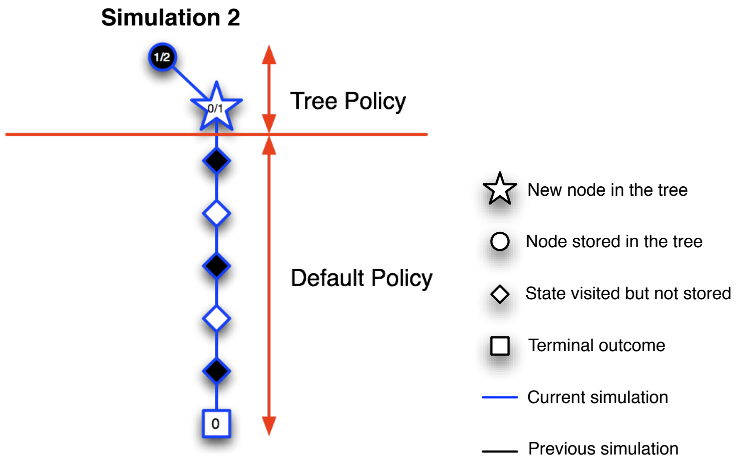
MCTS: Step by step



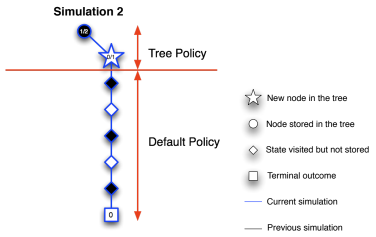
MCTS: Step by step



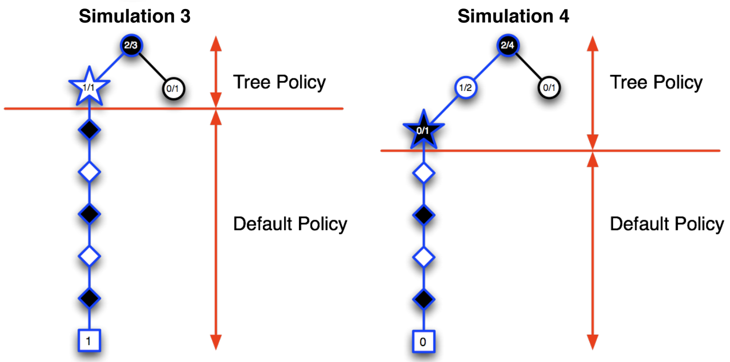
MCTS: Step by step



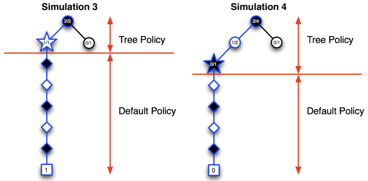
MCTS: Step by step



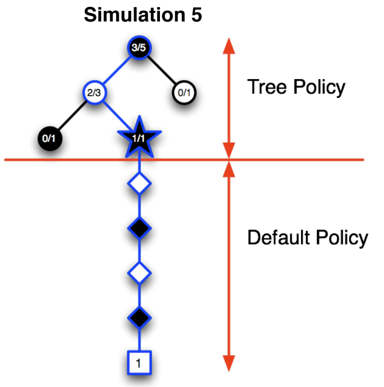
MCTS: Step by step



MCTS: Step by step



MCTS: Step by step



New node in the tree

Node stored in the tree

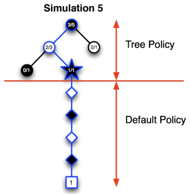
State visited but not stored

Terminal outcome

Current simulation

Previous simulation

MCTS: Step by step



New node in the tree
Node stored in the tree
State visited but not stored
Terminal outcome
Current simulation
Previous simulation

Presentation Outline

Presentation Outline

1. Sequential Decision Making
2. Games
3. Game Tree Search
4. Monte Carlo Tree Search
5. Extensions & Domains
6. Conclusions

UCT Algorithm

Upper Confidence Bounds for Trees (UCT)

The most popular algorithm in the MCTS family which is *consistent*, i.e., given enough time, the algorithm will find the *optimal* values for all nodes of the tree, and can therefore select the optimal action at the root state.

Algorithm 1 General MCTS approach.

function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))

Exploitation-Exploration balance

The algorithm must balance between testing an alternative that looks currently the best (to obtain a precise estimate) and the exploration of other alternatives (to ensure that some good alternative is not missed).

UCT Algorithm

Upper Confidence Bounds for Trees (UCT)

The most popular algorithm in the MCTS family which is *consistent*, i.e., given enough time, the algorithm will find the *optimal* values for all nodes of the tree, and can therefore select the optimal action at the root state.

Algorithm 1 General MCTS approach.

function MCTSSEARCH(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TREEPOLICY(v_0)
    $\Delta \leftarrow$ DEFAULTPOLICY($s(v_l)$)
    BACKUP(v_l, Δ)
  return a(BESTCHILD(v_0))

Exploitation-Exploration balance

The algorithm must balance between testing an alternative that looks currently the best (to obtain a precise estimate) and the exploration of other alternatives (to ensure that some good alternative is not missed).

1. W jaki sposob przechodzi po drzewie, w ktora strone rozbudowywac drzewo
2. In order to find the best move in the root, one has to determine the best moves in the internal nodes as well.
3. Since the estimates of the values of moves rely on the estimates of the values of the (best) successor nodes, we must have small estimation errors for the latter ones.
4. The problem reduces to getting the estimation error decay quickly.

UCT Algorithm

Upper Confidence Bounds for Trees (UCT)

The most popular algorithm in the MCTS family which is *consistent*, i.e., given enough time, the algorithm will find the *optimal* values for all nodes of the tree, and can therefore select the optimal action at the root state.

Algorithm 1 General MCTS approach.

function MctsSearch(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TreePolicy(v_0)
    $\Delta \leftarrow$ DefaultPolicy($s(v_l)$)
    Backup(v_l, Δ)
  return a(BestChild(v_0))

Exploitation-Exploration balance

The algorithm must balance between testing an alternative that looks currently the best (to obtain a precise estimate) and the exploration of other alternatives (to ensure that some good alternative is not missed).

UCT Algorithm

Upper Confidence Bounds for Trees (UCT)

The most popular algorithm in the MCTS family which is *consistent*, i.e., given enough time, the algorithm will find the *optimal* values for all nodes of the tree, and can therefore select the optimal action at the root state.

Algorithm 1 General MCTS approach.

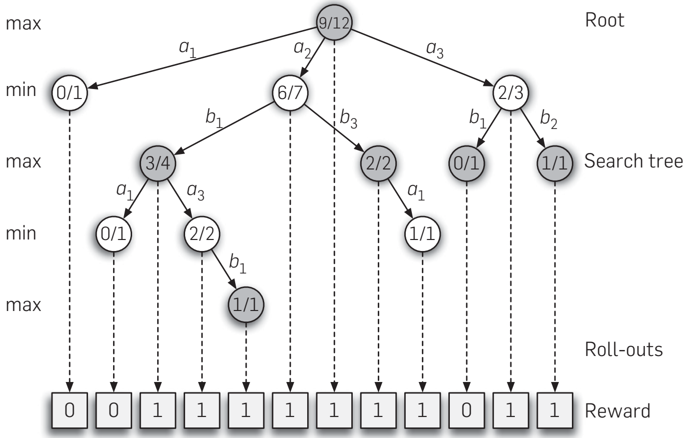
function MctsSearch(s_0)
  create root node v_0 with state s_0
  while within computational budget **do**
    $v_l \leftarrow$ TreePolicy(v_0)
    $\Delta \leftarrow$ DefaultPolicy($s(v_l)$)
    Backup(v_l, Δ)
  return a(BestChild(v_0))

Exploitation-Exploration balance

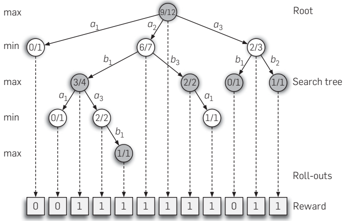
The algorithm must balance between testing an alternative that looks currently the best (to obtain a precise estimate) and the exploration of other alternatives (to ensure that some good alternative is not missed).

1. W jaki sposob przechodzi po drzewie, w ktora strone rozbudowywac drzewo
2. In order to find the best move in the root, one has to determine the best moves in the internal nodes as well.
3. Since the estimates of the values of moves rely on the estimates of the values of the (best) successor nodes, we must have small estimation errors for the latter ones.
4. The problem reduces to getting the estimation error decay quickly.

MCTS Tree



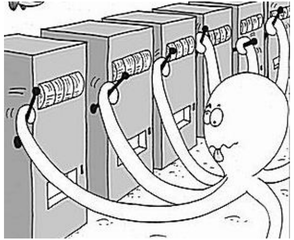
MCTS Tree



1. Problem balansowania miedzy eksploracja a eksploatacja pojawia sie w kazdym wezle w drzewie

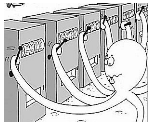
Selection? Multi-armed bandit!

- sequential decision problem
- K actions
- slot machines with unknown μ and distributions
- rewards: $X_{i,n}$ for $1 \leq i \leq K$ and $1 \leq n$
- **goal**: maximize cumulative reward
- exploitation-exploration dilemma
- **solution**: policy indicating which arm to play based on past rewards



Selection? Multi-armed bandit!

- sequential decision problem
- K actions
- slot machines with unknown μ and distributions
- rewards: $X_{i,n}$ for $1 \leq i \leq K$ and $1 \leq n$
- **goal**: maximize cumulative reward
- exploitation-exploration dilemma
- **solution**: policy indicating which arm to play based on past rewards



1. Okazuje sie, ze **kwestia znalezienia rownowagi** pomiedzy eksploatacja a eksploracja
2. Byla dosc intensywnie studiowana w kontekscie prostego sekwencyjnego problemu decyzyjnego
3. zwanego problemem wielorekiego bandyty
4. where i idicates the arm played
5. (must be estimated based on past observations)

Solution to the bandit problem

- Goal: minimize the *regret*:

$$R(n) = n\mu^* - \sum_{j=1}^{K} \mu_j \mathbb{E}[T_j(n)]$$

- No policy with regret that grows slower than $O(\ln(n))$ for a large class of reward distributions [Lai & Robbins, 1985].
- UCB1 [Agrawal, 1995] — *Optimism in the face of uncertainty*:

$$UCB1(j) = \bar{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

- Bounds (following from *Hoeffding's tail inequality*):

$$P(|\bar{X}_j - \mu_j| \geq \sqrt{\frac{2 \ln n}{n_j}}) \leq n^{-4}$$

Solution to the bandit problem

- Goal: minimize the *regret*:

$$R(n) = n\mu^* - \sum_{j=1}^{K} \mu_j \mathbb{E}[T_j(n)]$$

- No policy with regret that grows slower than $O(\ln(n))$ for a large class of reward distributions [Lai & Robbins, 1985].
- UCB1 [Agrawal, 1995] — *Optimism in the face of uncertainty*:

$$UCB1(j) = \bar{X}_j + \sqrt{\frac{2 \ln n}{n_j}}$$

- Bounds (following from *Hoeffding's tail inequality*):

$$P(|\bar{X}_j - \mu_j| \geq \sqrt{\frac{2 \ln n}{n_j}}) \leq n^{-4}$$

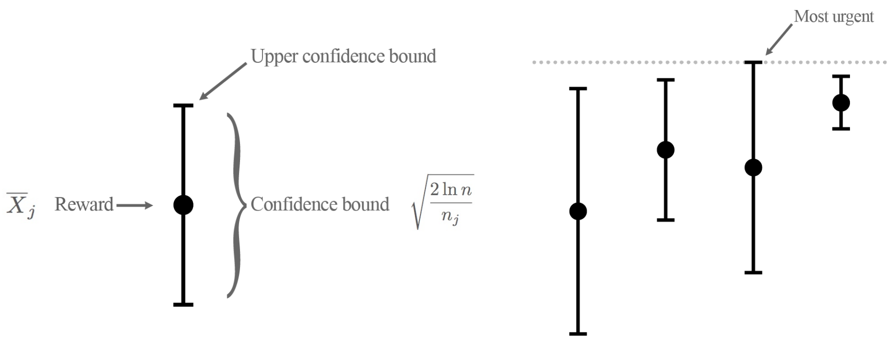
1. Proposed an algorithm, which, however was computationally "hard"
2. **Thus: "solving" means** having algorithm not worse asympotically
3. Prosta strategie ktora w tym rozumieniu jest optymalna zaproponowal Agrawal
4. Strategie te **mozna nazwac strategia optymizmu w obliczu niepewnosci**
5. Sprowadza sie ona do wyboru akcji ktora biorad pod uwage niepewnosc dotychczasowej estymaty moze miec optymistycznie najwyzsza wartosc oczekiwana
6. Mozna to zapisac w nastepujacy sposob, dla kazdej akcji j defniujac tzw. UCB - gorne ograniczenie ufnosci
7. Ograniczenie to oblicza sie jako empiryczna wartosc oczekiwana powiekszona o tzw. optymistyczna poprawke
8. W konsekwencji otrzymujemy gorna granice przedzialu ufnosci w ktorym z duzym prawdopodobienstwem (co wynika z nierownosci Hoeffdinga) znajduje sie wartosc oczekiwana danej akcji
9. Wybierajac akcje w ten sposob, za kazdym razem albo wybierzemy wartosc optymalna albo zredukujemy niepewnosc i szerokosc tego przedzialu dla akcji suboptymalnej
10. Tak wiec, suboptymalne akcje w koncu przestana byc wybierane

Upper Confidence Bounds for Trees (UCT)

- It was found to work also for non-stationary distributions in trees [Kocsis & Szepesvari, 2006]

$$UCB1(j) = \bar{X}_j + C\sqrt{\frac{2 \ln n}{n_j}}$$

- theoretically, $C = 1$; in practice chosen experimentally

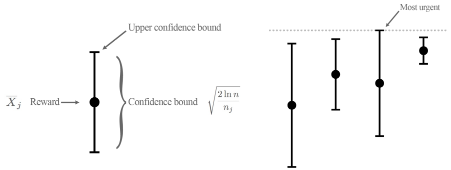


Upper Confidence Bounds for Trees (UCT)

- It was found to work also for non-stationary distributions in trees [Kocsis & Szepesvari, 2006]

$$UCB1(j) = \bar{X}_j + C\sqrt{\frac{2 \ln n}{n_j}}$$

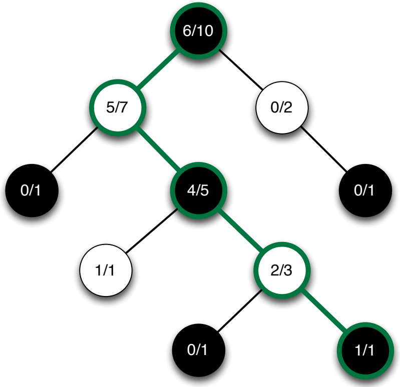
- theoretically, $C = 1$; in practice chosen experimentally



1. Specyficzna klasa rozkladow niestacjonarnych
2. Kocsis i S udowodnili zbieznosc algorytmu UCT do wartosci minimaxowych
3. Not: highest rewards, highest/lowest bound, but most urgent
4. More visits = tighter bound

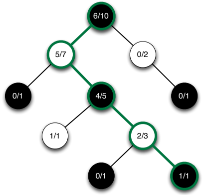
Exploitation & exploration

$$Q_{UCT}(s,a) = Q(s,a) + C\sqrt{\frac{2\ln n(s)}{n(s,a)}}$$



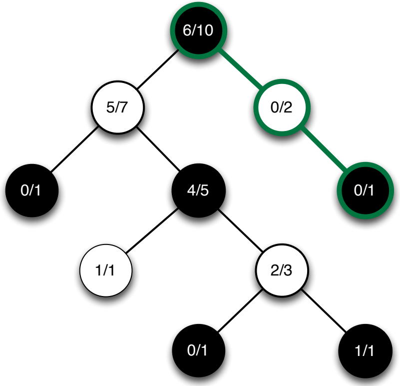
Exploitation & exploration

$$Q_{UCT}(s,a) = Q(s,a) + C\sqrt{\frac{2\ln n(s)}{n(s,a)}}$$



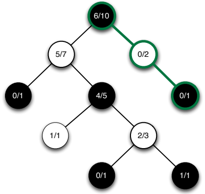
Exploitation & exploration

$$Q_{UCT}(s,a) = Q(s,a) + \boxed{C\sqrt{\frac{2\ln n(s)}{n(s,a)}}}$$



Exploitation & exploration

$$Q_{UCT}(s,a) = Q(s,a) + \boxed{C\sqrt{\frac{2\ln n(s)}{n(s,a)}}}$$



Presentation Outline

Presentation Outline

1. Sequential Decision Making
2. Games
3. Game Tree Search
4. Monte Carlo Tree Search
5. Extensions & Domains
6. Conclusions

Strengths — Anytime & Aheuristic

Anytime

- Backpropagates the outcome of each game immediately.
- Can be stopped at any time returning the currently best action.
- More computing power generally leads to better performance.

Aheuristic

- No specific domain knowledge required:
  - Available actions for a given state (legal moves).
  - Whether a given state is terminal (game over).
- Intelligent moves with no tactical knowledge.
- Ideal for General Game Playing.

Strengths — Anytime & Aheuristic

Anytime
- Backpropagates the outcome of each game immediately.
- Can be stopped at any time returning the currently best action.
- More computing power generally leads to better performance.

Aheuristic
- No specific domain knowledge required:
  - Available actions for a given state (legal moves).
  - Whether a given state is terminal (game over).
- Intelligent moves with no tactical knowledge.
- Ideal for General Game Playing.

1. allowing the algorithm to run for additional iterations often improves the result.
2. significant improvements in performance may often be achieved using domain-specific knowledge.

Strengths — Anytime & Aheuristic

Anytime

- Backpropagates the outcome of each game immediately.
- Can be stopped at any time returning the currently best action.
- More computing power generally leads to better performance.

Aheuristic

- No specific domain knowledge required:
  - Available actions for a given state (legal moves).
  - Whether a given state is terminal (game over).
- Intelligent moves with no tactical knowledge.
- Ideal for General Game Playing.

Strengths — Anytime & Aheuristic

Anytime
- Backpropagates the outcome of each game immediately.
- Can be stopped at any time returning the currently best action.
- More computing power generally leads to better performance.

Aheuristic
- No specific domain knowledge required:
  - Available actions for a given state (legal moves).
  - Whether a given state is terminal (game over).
- Intelligent moves with no tactical knowledge.
- Ideal for General Game Playing.

1. allowing the algorithm to run for additional iterations often improves the result.
2. significant improvements in performance may often be achieved using domain-specific knowledge.

Game Description Language

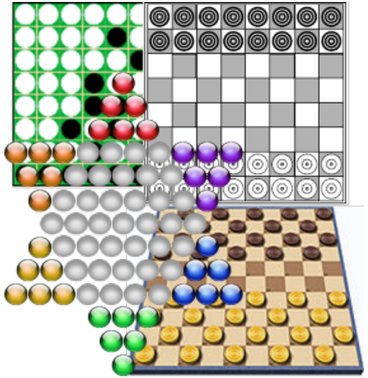
```
role(white)
role(black)

base(cell(M,N,Z)) :-
  index(M) &
  index(N) &
  filler(Z)

base(control(W)) :- role(W)

input(W,mark(X,Y)) :-
  role(W) &
  index(X) &
  index(Y)

input(W,noop) :- role(W)

init(cell(X,Y,b)) :-
  index(X) &
  index(Y)

init(control(white))

legal(P,mark(X,Y)) :-
  true(cell(X,Y,b)) &
  true(control(P))

legal(x,noop) :-
  true(control(black))

legal(o,noop) :-
  true(control(white))

next(cell(M,N,x)) :-
    does(white,mark(M,N))

next(cell(M,N,0)) :-
    does(black,mark(M,N))

next(cell(M,N,Z)) :-
    does(P,mark(M,N)) &
    true(cell(M,N,Z)) & Z!=b

next(cell(M,N,b)) :-
    does(P,mark(J,K)) &
    true(cell(M,N,b)) &
    distinct(M,J)

next(cell(M,N,b)) :-
    does(P,mark(J,K)) &
    true(cell(M,N,b)) &
    distinct(N,K)

next(control(white)) :-
    true(control(black))

next(control(black)) :-
    true(control(white))

goal(white,100) :- line(x) & ~line(o)
goal(white,50) :- ~line(x) & ~line(o)
goal(white,0) :- ~line(x) & line(o)
goal(black,100) :- ~line(x) & line(o)
goal(black,50) :- ~line(x) & ~line(o)
goal(black,0) :- line(x) & ~line(o)

terminal :- line(P)
terminal :- ~open

row(M,P) :-
    true(cell(M,1,P)) &
    true(cell(M,2,P)) &
    true(cell(M,3,P))

column(N,P) :-
    true(cell(1,N,P)) &
    true(cell(2,N,P)) &
    true(cell(3,N,P))

diagonal(P) :-
    true(cell(1,1,P)) &
    true(cell(2,2,P)) &
    true(cell(3,3,P))

diagonal(P) :-
    true(cell(1,3,P)) &
    true(cell(2,2,P)) &
    true(cell(3,1,P))

line(P) :- row(M,P)
line(P) :- column(N,P)
line(P) :- diagonal(P)

open :- true(cell(M,N,b))

index(1)      filler(x)
index(2)      filler(o)
index(3)      filler(b)
```


Game Description Language

role(white)
role(black)
base(cell(M,N,Z)) :- index(M) & index(N) & filler(Z)
base(control(W)) :- role(W)
input(W,mark(X,Y)) :- role(W) & index(X) & index(Y)
input(W,noop) :- role(W)
init(cell(X,Y,b)) :- index(X) & index(Y)
init(control(white))
legal(P,mark(X,Y)) :- true(cell(X,Y,b)) & true(control(P))
legal(x,noop) :- true(control(black))
legal(o,noop) :- true(control(white))
next(cell(M,N,x)) :- does(white,mark(M,N))
next(cell(M,N,0)) :- does(black,mark(M,N))
next(cell(M,N,Z)) :- does(P,mark(M,N)) & true(cell(M,N,Z)) & Z!=b
next(cell(M,N,b)) :- does(P,mark(J,K)) & true(cell(M,N,b)) & distinct(M,J)
next(cell(M,N,b)) :- does(P,mark(J,K)) & true(cell(M,N,b)) & distinct(N,K)
next(control(white)) :- true(control(black))
next(control(black)) :- true(control(white))
goal(white,100) :- line(x) & ~line(o)
goal(white,50) :- ~line(x) & ~line(o)
goal(white,0) :- ~line(x) & line(o)
goal(black,100) :- ~line(x) & line(o)
goal(black,50) :- ~line(x) & ~line(o)
goal(black,0) :- line(x) & ~line(o)
terminal :- line(P)
terminal :- ~open
row(M,P) :- true(cell(M,1,P)) & true(cell(M,2,P)) & true(cell(M,3,P))
column(N,P) :- true(cell(1,N,P)) & true(cell(2,N,P)) & true(cell(3,N,P))
diagonal(P) :- true(cell(1,1,P)) & true(cell(2,2,P)) & true(cell(3,3,P))
diagonal(P) :- true(cell(1,3,P)) & true(cell(2,2,P)) & true(cell(3,1,P))
line(P) :- row(M,P)
line(P) :- column(N,P)
line(P) :- diagonal(P)
open :- true(cell(M,N,b))
index(1) filler(x)
index(2) filler(o)
index(3) filler(b)

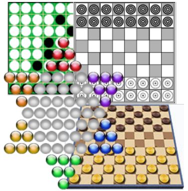
General Game Playing Competitions

- Held annually by Stanford / AAAI since 2005.
- UCT-based CadiaPlayer won in 2007.
- Currently all players use some version of MCTS.



General Game Playing Competitions

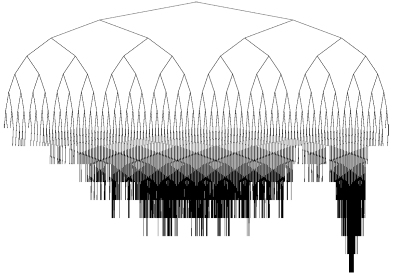
- Held annually by Stanford / AAAI since 2005.
- UCT-based CadiaPlayer won in 2007.
- Currently all players use some version of MCTS.



Strengths — Asymmetry & Parallelisation

Asymmetric Tree Growth

- Tree grows towards more promising areas.
- No fixed ply — tree expands to fit search space.
- Can go deeper than tradition game tree search.

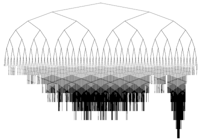


Easy parallelisation due to independent nature of each simulation.

Strengths — Asymmetry & Parallelisation

Asymmetric Tree Growth

- Tree grows towards more promising areas.
- No fixed ply — tree expands to fit search space.
- Can go deeper than tradition game tree search.

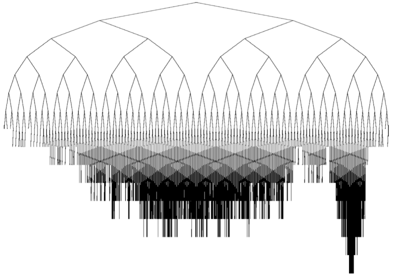


Easy parallelisation due to independent nature of each simulation.

Strengths — Asymmetry & Parallelisation

Asymmetric Tree Growth

- Tree grows towards more promising areas.
- No fixed ply — tree expands to fit search space.
- Can go deeper than tradition game tree search.

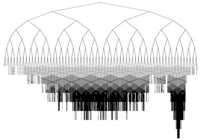


Easy parallelisation due to independent nature of each simulation.

Strengths — Asymmetry & Parallelisation

Asymmetric Tree Growth

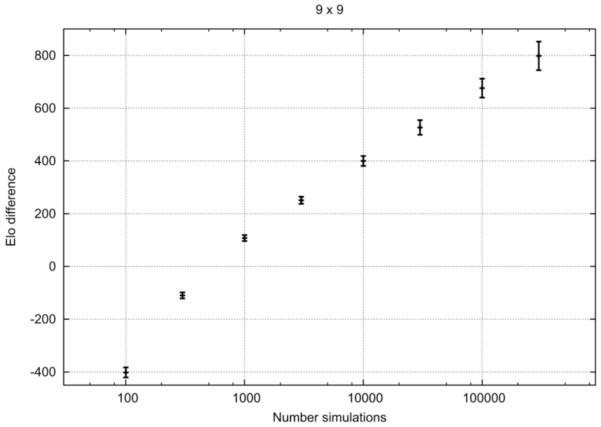
- Tree grows towards more promising areas.
- No fixed ply — tree expands to fit search space.
- Can go deeper than tradition game tree search.



Easy parallelisation due to independent nature of each simulation.

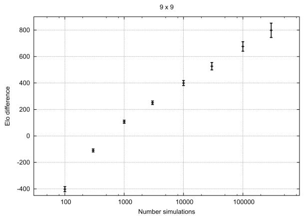
Convergence

- Converges to optimal (minimax) values given infinite time.
- Convergence speed might be improved by some modifications, but still 90% studies use "pure" UCT.



Convergence

- Converges to optimal (minimax) values given infinite time.
- Convergence speed might be improved by some modifications, but still 90% studies use "pure" UCT.



Weaknesses

- Memory intensive — tree must be kept in memory.
- Needs a lot of samples (simulation must be cheap).
- Tuning only by empirical studies — the dynamics of search are not yet fully understood.

> Intuitively, Monte-Carlo search methods work best when the estimated values from shallow searches are similar to the estimated values from deeper searches, in other words the mean reward of simulations is somewhat indicative of the optimal value, at all stages of the search.
>
> *The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions, Gelly S., Kocsis L, Shoenauer M., Sebag M., Silver D., Szepesvari C., 2012*

Weaknesses

- Memory intensive — tree must be kept in memory.
- Needs a lot of samples (simulation must be cheap).
- Tuning only by empirical studies — the dynamics of search are not yet fully understood.

> Intuitively, Monte-Carlo search methods work best when the estimated values from shallow searches are similar to the estimated values from deeper searches, in other words the mean reward of simulations is somewhat indicative of the optimal value, at all stages of the search.
>
> *The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions, Gelly S., Kocsis L, Shoenauer M., Sebag M., Silver D., Szepesvari C., 2012*

1. Czyli gdy latwo szybko przegrac - wtedy symulacje skupiaja sie na takich wlasnie ruchach

Applications of MCTS

Computer Go
MoGo
Fuego
CrazyStone
Leela
Many Faces of Go
SteenVreter
Zen

Realtime Games
Ms-PacMan
Real Time Strategy (RTS) Games
Tron
Dead End

Nondeterministic Games
Bridge
Poker
Magic: The Gathering
Backgammon

Solitaire (Puzzle) Games
Sudoku
Kakuro
Crosswords
Morpion Solitaire
SameGame
Bubble Breaker

Connection Games
Hex
Y
Havannah
Renkula
Lines of Action

Combinatorial Games
Amazons
Arimaa
Khet
Shogi
Mancala
Kriegspiel
Clobber
Othello
Blokus
Focus
Connect Four
Sum of Switches

Multiplayer Games
Settlers of Catan

General Game Playing
CadiaPlayer
Ary
Centurio

NON-GAME DOMAINS

Combinatorial Optimisation
Security
Mixed Integer Programming
Travelling Salesman Problem
Physics Simulations
Function Approximation

Constraint Satisfaction

Scheduling
Printer Scheduling
Production Management
Bus Regulation

Sample-Based Planning
Large State Spaces
Feature Selection

Procedural Content Generation
Language
Game Design
Art

Applications of MCTS

Computer Go
MoGo
Fuego
CrazyStone
Leela
Many Faces of Go
SteenVreter
Zen

Realtime Games
Ms-PacMan
Real Time Strategy (RTS) Games
Tron
Dead End

Nondeterministic Games
Bridge
Poker
Magic: The Gathering
Backgammon

Solitaire (Puzzle) Games
Sudoku
Kakuro
Crosswords
Morpion Solitaire
SameGame
Bubble Breaker

Connection Games
Hex
Y
Havannah
Renkula
Lines of Action

Combinatorial Games
Amazons
Arimaa
Khet
Shogi
Mancala
Kriegspiel
Clobber
Othello
Blokus
Focus
Connect Four
Sum of Switches

Multiplayer Games
Settlers of Catan

General Game Playing
CadiaPlayer
Ary
Centurio

NON-GAME DOMAINS

Combinatorial Optimisation
Security
Mixed Integer Programming
Travelling Salesman Problem
Physics Simulations
Function Approximation

Constraint Satisfaction

Scheduling
Printer Scheduling
Production Management
Bus Regulation

Sample-Based Planning
Large State Spaces
Feature Selection

Procedural Content Generation
Language
Game Design
Art

1. Application to many domains (not just games)

Extensions of MCTS

Bandit-Based
UCB1-Tuned
Bayesian UCT
EXP3
HOOT

Selection
FPU
Decisive Moves
Move Groups
Transpositions
Progressive Bias
Opening Books
MCPG
Search Seeding
Parameter Tuning
History Heuristic
Progressive History

AMAF
Permutation
α-AMAF
Some-First
Cutoff
RAVE
Killer RAVE
RAVE-max
PoolRAVE

Game-Theoretic
MCTS-Solver
MC-PNS
Score Bounded MCTS

Pruning
Absolute
Relative
Domain Knowledge

Simulation
Rule-Based
Contextual
Fill the Board

History Heuristics
Evaluation
Balancing
Last Good Reply
Patterns

Backpropagation
Weighting
Score Bonus
Decay
Transposition Tables

Learning
MAST
PAST
FAST

Parallelisation
Leaf
Root
Tree
UCT-Treesplit
Threading
Synchronisation

Considerations
Consistency
Parameterisation
Comparing
Enhancements

Extensions of MCTS

Bandit-Based
UCB1-Tuned
Bayesian UCT
EXP3
HOOT

Selection
FPU
Decisive Moves
Move Groups
Transpositions
Progressive Bias
Opening Books
MCPG
Search Seeding
Parameter Tuning
History Heuristic
Progressive History

AMAF
Permutation
α-AMAF
Some-First
Cutoff
RAVE
Killer RAVE
RAVE-max
PoolRAVE

Game-Theoretic
MCTS-Solver
MC-PNS
Score Bounded MCTS

Pruning
Absolute
Relative
Domain Knowledge

Simulation
Rule-Based
Contextual
Fill the Board

History Heuristics
Evaluation
Balancing
Last Good Reply
Patterns

Backpropagation
Weighting
Score Bonus
Decay
Transposition Tables

Learning
MAST
PAST
FAST

Parallelisation
Leaf
Root
Tree
UCT-Treesplit
Threading
Synchronisation

Considerations
Consistency
Parameterisation
Comparing
Enhancements

1. Hot research topic in AI

Timeline of Events

1990	Abramson demonstrates that Monte Carlo simulations can be used to evaluate value of state
1993	Brugmann applies Monte Carlo methods to the field of computer Go.
1998	MAVEN defeats the world scrabble champion.
2002	Auer et al. propose UCB1 for multi-armed bandit, laying the theoretical foundation for UCT.
2006	Coulom describes Monte Carlo evaluations for tree-based search, coining the term **Monte Carlo tree search**.
2006	Kocsis and Szepesvari associate UCB with tree-based search to give the **UCT algorithm**.
2006	Gelly et al. apply UCT to computer Go with remarkable success.
2007	CadiaPlayer becomes world champion General Game Player.
2008	MoGo achieves dan (master) level at 9x9 Go.
2009	FueGo beats top human professional at 9x9 Go.
2013	Crazy Stone beats professional human player at 19x19 Go with four handicap stones.

Timeline of Events

1990	Abramson demonstrates that Monte Carlo simulations can be used to evaluate value of state
1993	Brugmann applies Monte Carlo methods to the field of computer Go.
1998	MAVEN defeats the world scrabble champion.
2002	Auer et al. propose UCB1 for multi-armed bandit, laying the theoretical foundation for UCT.
2006	Coulom describes Monte Carlo evaluations for tree-based search, coining the term **Monte Carlo tree search**.
2006	Kocsis and Szepesvari associate UCB with tree-based search to give the **UCT algorithm**.
2006	Gelly et al. apply UCT to computer Go with remarkable success.
2007	CadiaPlayer becomes world champion General Game Player.
2008	MoGo achieves dan (master) level at 9x9 Go.
2009	FueGo beats top human professional at 9x9 Go.
2013	Crazy Stone beats professional human player at 19x19 Go with four handicap stones.

1. MCTS revolutionized Go
2. In the past, there have been two primary techniques for decision-making in adversarial games: minimax alpha-beta search and knowledge-based approaches.
3. Monte-Carlo tree search represents a new paradigm for planning in this challenging domain, which may prove to have implications well beyond the two-player games for which it was originally developed.

Q & A

THANK YOU

Q & A

THANK YOU