Predicting Opponent Moves for Improving Hearthstone AI

by Alexander Dockhorn, Max Frick, Ünal Akkaya, and Rudolf Kruse

Institute for Intelligent Cooperating Systems
Department for Computer Science, Otto von Guericke University Magdeburg
Universitaetsplatz 2, 39106 Magdeburg, Germany

Email: {alexander.dockhorn, max.frick, uenal.akkaya, rudolf.kruse}@ovgu.de

Contents

Hearthstone – an Online Card Game

- Hearthstone is an online collectible card game
- 2 players play a single game using a self-constructed deck of 30 out of more than 1000 cards
- Next to a small amount of standard-cards most cards have unique effects and

OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
INF FAKULTÄT FÜR INFORMATIK

Hearthstone – Game Components and States

Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (MCTS): Adding Monte Carlo simulations (or rollouts) after adding a new node to the tree.

Two different policies are used on each episode:

- **Tree Policy:** Selecting a node for expansion in the search tree (e.g. Greedy, UCB).
- **Default Policy:** Control the simulations outside the tree (e.g. picking actions at random)

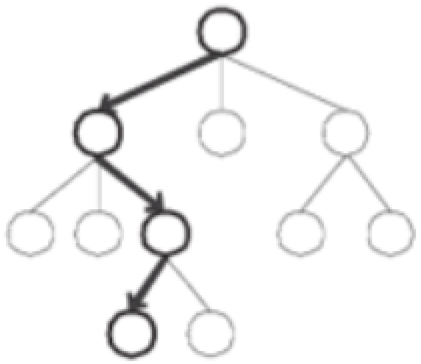
UCB1

$$a_t = \arg\max_{a \in A} \underbrace{Q_t(a)}_{\text{Exploitation}} + C \underbrace{\sqrt{\frac{\ln N(s)}{N(s,a)}}}_{\text{Exploration}}$$

Monte Carlo Tree Search - Tree Policy

1. Tree Selection:

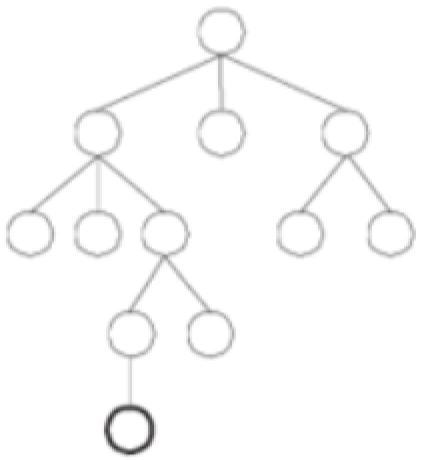
Following the **tree policy** (i.e. UCB1), navigate the tree until reaching a node with at least one child state that was not explored yet.



2. Expansion:

Add a new node in the tree, as a child of the node reached in the tree selection step.

- this node resembles an action

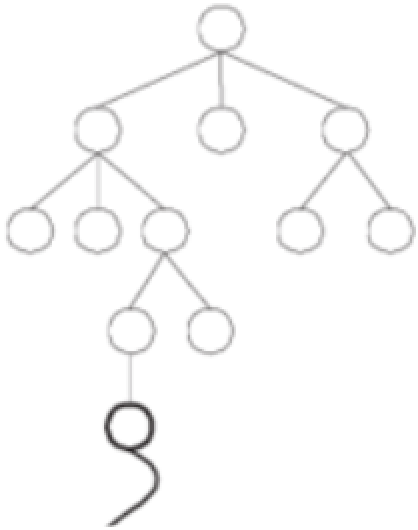


Monte Carlo Tree Search - Default Policy

3. Monte Carlo Simulation/Rollout:

Following the **default policy**, advance the state until a terminal state or a pre-defined maximum number of steps.

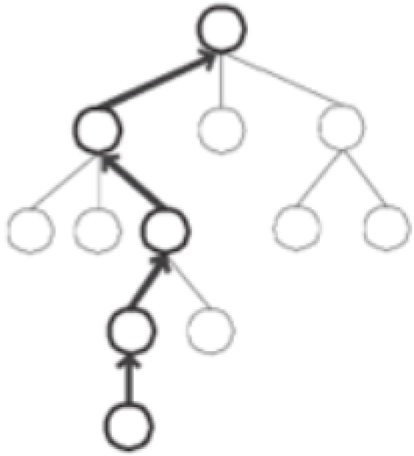
- calculate the return of this episode.



4. Back-propagation:

Update the values of $Q(s, a)$, $N(s)$ and $N(s, a)$ of the nodes visited in the tree during steps 1 and 2.

- Values of nodes visited during the simulation will not be updated.



Why we cannot use MCTS effectively?

- Monte Carlo Tree search needs a Forward Model for executing many simulations starting from the current state
  - We now the rules of the game, so the Forward Model is available
  - But we do not know the current state of the system!

- The player does not know the real state of the game, which consists of
  - The board state
  - The own and the opponent‘s cards
  - The cards in both decks
  - All previously played cards

Sources of Uncertainty

- Specifically the player does not know:
  - Which specific cards he will draw in the future
  - Which cards the opponent has on his hand or in the deck

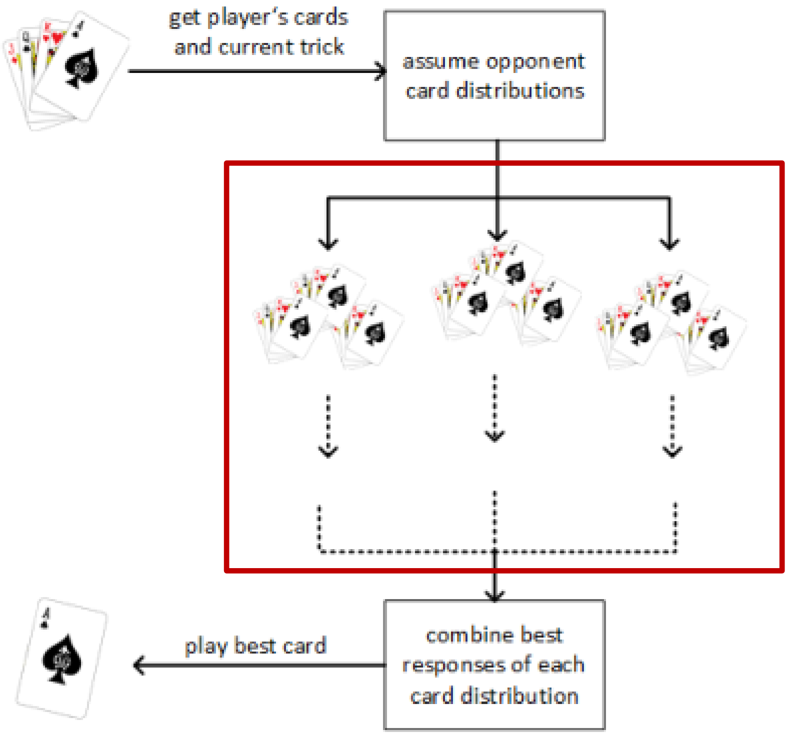
- We need to adapt Monte Carlo Tree Search as long as we do not know the current game state.

MCTS for an Unknown Card Distribution

- If the real gamestate is unknown we can still guess multiple times:
  - information about the opponent‘s hero and cards that were already played restrict the open card pool

- Use and ensemble of MCTS agents with majority vote

- **This does not suffice in Hearthstone!**
  - **Too many possible card distributions**



Method used for Doppelkopf

Dockhorn, A., Doell, C., Hewelt, M., & Kruse, R. (2017). A decision heuristic for Monte Carlo tree search doppelkopf agents. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1–8). IEEE. http://doi.org/10.1109/SSCI.2017.8285181

Deckbuilding

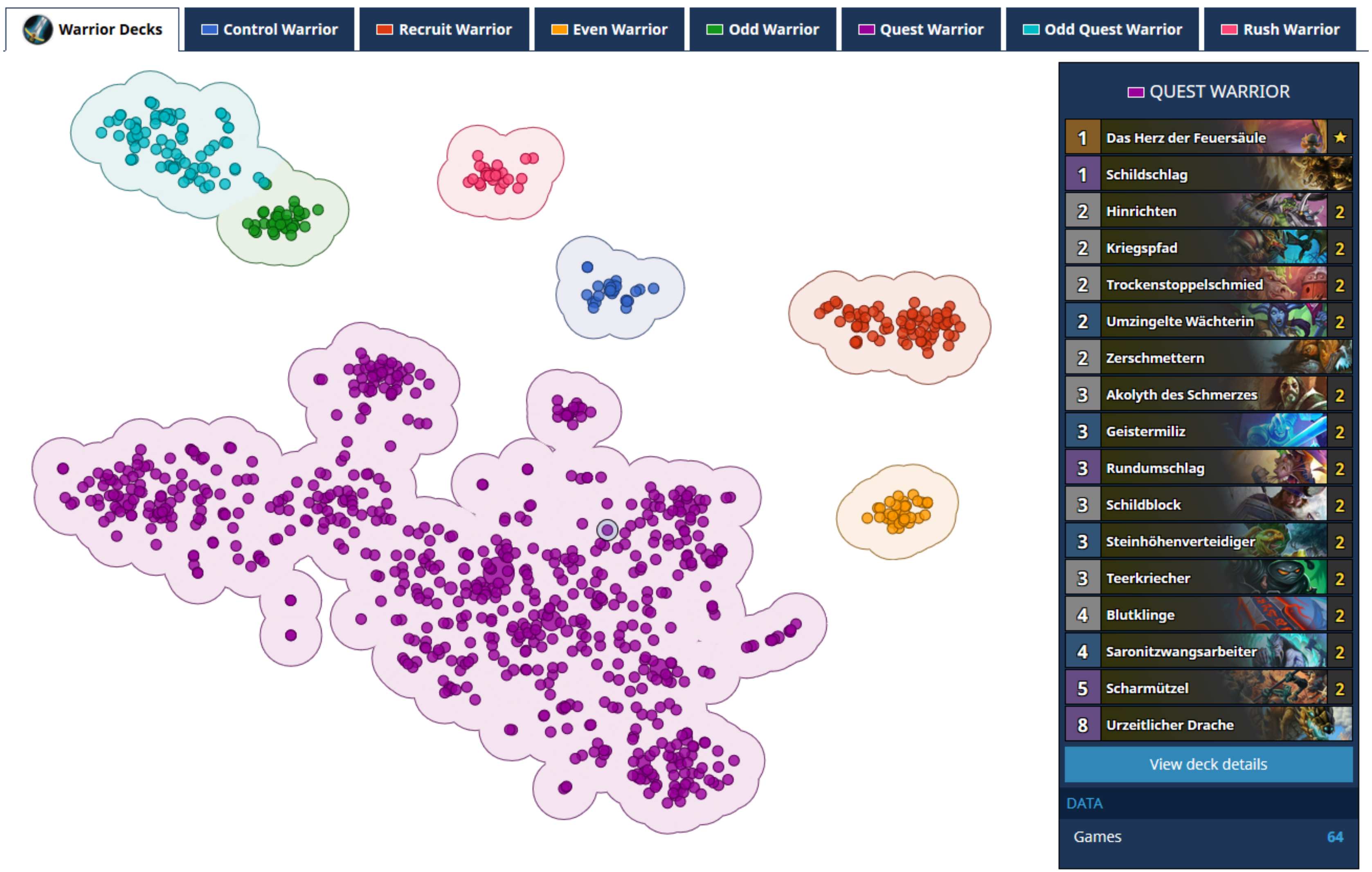
- By common knowledge popular decks often employ critical properties such as:
  - A stable mana curve
  - Strong synergies
  - A strategy to reach the win-condition
- Most decks can be categorized in deck types, which are common to the meta-game
  - Those decks often consist of similar cards, but do not necessarily contain the same 30 cards

Mana Curves

- The players can play their cards using mana
- Each turn the pool of available mana increases by one
- The mana curve of a deck describes how much cards per cost are included in the deck

Card Synergies

Meta-Decks


Warrior Decks
Control Warrior
Recruit Warrior
Even Warrior
Odd Warrior
Quest Warrior
Odd Quest Warrior
Rush Warrior
QUEST WARRIOR
1 Das Herz der Feuersäule
1 Schildschlag
2 Hinrichten 2
2 Kriegspfad 2
2 Trockenstoppelschmied 2
2 Umzingelte Wächterin 2
2 Zerschmettern
3 Akolyth des Schmerzes 2
3 Geistermiliz 2
3 Rundumschlag 2
3 Schildblock 2
3 Steinhöhenverteidiger 2
3 Teerkriecher 2
4 Blutklinge 2
4 Saronitzwangsarbeiter 2
5 Scharmützel 2
8 Urzeitlicher Drache
View deck details
DATA
Games 64

Finding Probable Card-Combinations

- Meta-decks and card synergies tell us that we have strong dependencies between cards in a deck
- We want to predict likely cards by observing frequent card-combinations in a database of played games
- Counting bi-grams of co-occuring cards:
  - **isolated**: cards need to be played at the same turn
  - **successive**: cards need to be played in successive turns
  - **combined**: isolated and successive counts are added

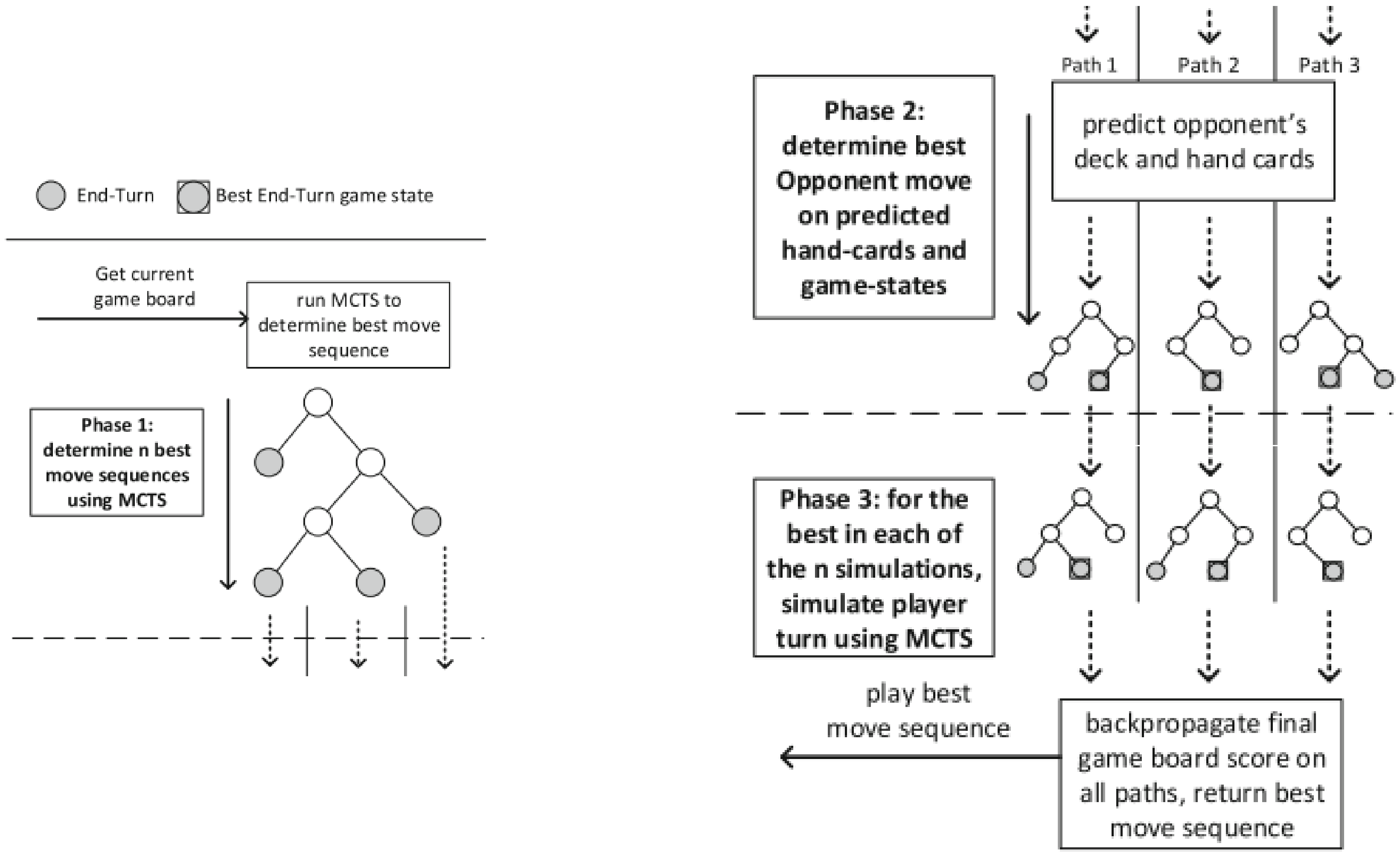
Adapting MCTS Gamestate Sampling

- A set of hand-cards is determined using the bi-gram database and all previously seen cards
    1. determine the number of co-occurences of each card
    2. filter possible cards by the rules of the deckbuilding process
    3. sample the opponent‘s hand cards based on the normalized co-occurence values

- Generate multiple set of opponent‘s hand cards and run MCTS on each of them

- Use (weighted) majority vote to determine the best card to be played

The overall Process



Evaluation

- We tested our agent against multiple other agents playing multiple decks
  - Random = randomly choose the next action
  - flatMC = flat Monte Carlo algorithm, simulate n times for each action
  - plainMCTS = MCTS using a randomly guessed game states
  - foMCTS = MCTS using the true game state (cheating)
  - Exh.s. = exhaustive search for best action, does not consider the opponent

Wins in %	Aggro	Mid	Control
Random	95	100	100
flatMC	81	73	94
plainMCTS	59	47	58
foMCTS	46	36	60
exh.s.	65	47	70

(a) predMCTS Aggro Deck

Wins in %	Aggro	Mid	Control
Random	99	98	100
flatMC	88	85	99
plainMCTS	71	55	76
foMCTS	59	50	76
exh.s.	62	70	85

(b) predMCTS Mid-Range Deck

Wins in %	Aggro	Mid	Control
Random	97	97	100
flatMC	73	54	89
plainMCTS	36	31	68
foMCTS	41	16	51
exh.s.	45	20	61

(c) predMCTS Control Deck

Conclusions

- Applying MCTS to Hearthstone is very limited due to missing information
  - Guessing a gamestate lets us apply MCTS with weak play strength

- Combining multiple of such guesses by creating an ensemble of MCTS helps but does not solve the problem efficient enough

- In this work we proposed a card-prediction to enhance the accuracy of sampling possible card distributions
  - An ensemble using such predicted gamestates is nearly as powerful as knowing the real gamestate

Limitations and Open Research Questions

- Our tests using an MCTS agent with full information on the current game-state show that a perfect prediction would yield slightly better results
  - Further improving the prediction accuracy promises to yield a stronger agent

➢ Explore the tradeoff between a complex prediction or more simulations/predicted card sets

➢ Can fuzzy sets and dempster-shafer theory help us to solve this problem more efficiently?

- We want to further explore meta-decks
  - How do they develop over time?
  - Can we detect changes in the meta and react accordingly?

➢ Important for deck-building and balancing!

Thank you for your attention!

Interested in trying it yourself? Check out our Hearthstone Competition at:
http://www.is.ovgu.de/Research/HearthstoneAI.html

by Alexander Dockhorn, Max Frick, Ünal Akkaya, and Rudolf Kruse

Institute for Intelligent Cooperating Systems
Department for Computer Science, Otto von Guericke University Magdeburg
Universitaetsplatz 2, 39106 Magdeburg, Germany

Email: {alexander.dockhorn, max.frick, uenal.akkaya, rudolf.kruse}@ovgu.de