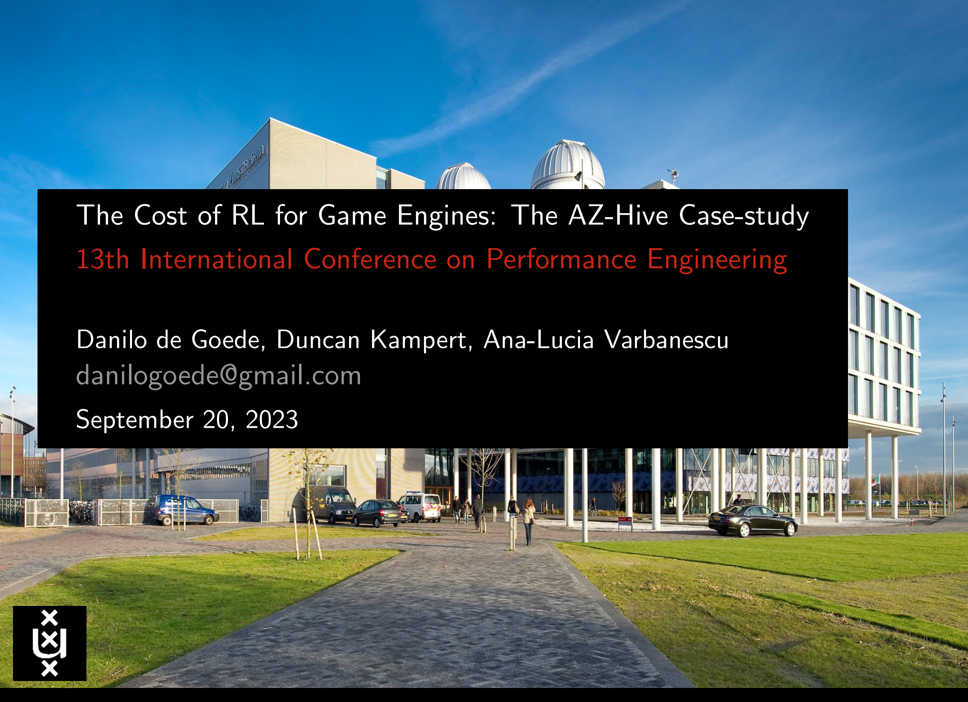


The Cost of RL for Game Engines: The AZ-Hive Case-study

13th International Conference on Performance Engineering

Danilo de Goede, Duncan Kampert, Ana-Lucia Varbanescu
danilogoede@gmail.com

September 20, 2023

Heuristics: Humans vs. AlphaZero

- Traditional approaches
  - *Hand-crafted heuristics*
- Reinforcement Learning
  - *Learn heuristics by practicing*

Shannon's chess-playing machine (1949)

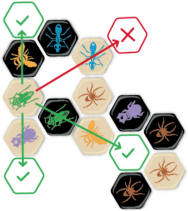
Deep Blue Vs. Garry Kasparov (1997)

Why should you (ICPE) care?

- AlphaZero *may* solve many games, but its *efficiency* is unclear.
- To increase efficiency, optimizations are needed:
  1. Add more human knowledge to reduce the design space (game heuristics/mathematics research);
  2. Improve the modeling itself (AI research);
  3. Improve training speed (performance engineering).

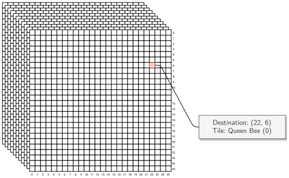
A case study: the game of Hive

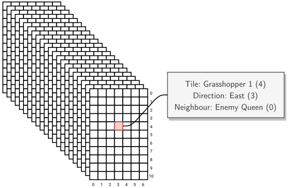
- Hive: A tile-based strategy game
- Hive is different from already solved games
  1. Lack of natural progression
  2. Dynamic structure of game states
- Hive lacks a strong computer implementation



Design space: construction

- Action encoding
  1. *Absolute coordinate*
  2. *Tile-relative*
- Board representation
  1. 2D: Original, Symmetric, Simple
  2. 3D: Binary Planes, Hybrid
- NN architecture
- Optimisations
  1. Exploit symmetries of Hive
  2. Game rule modifications
- Training hyperparameters





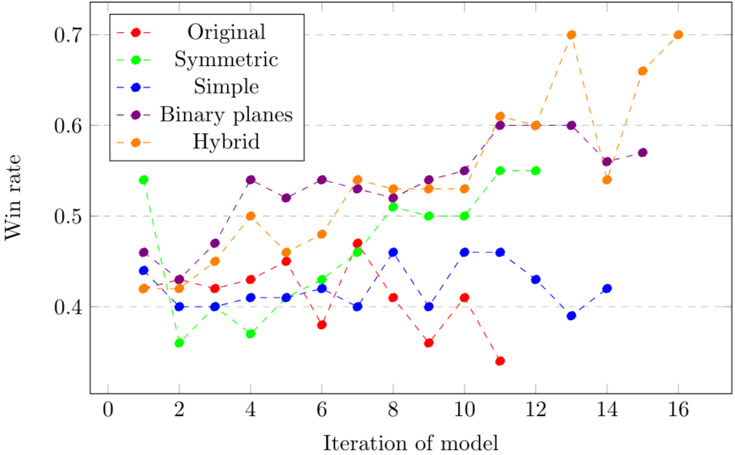
The first 3 dimensions already span a design space of 60 unique configurations

Design space: exploration

We select "best" configuration based on playing strength.

- Playing strength estimation
  - Challenge: Hard to quantify the strength of a move
  - Solution: Compare against other implementations
  - Metric of success: Win rate against a random agent
- Experimental set-up
  - Train 5 configurations for 4 hours
  - Pit every accepted model against the random agent

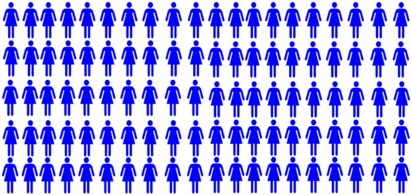
Exploration cost



[1] *Note:* Every line here takes 20 hours. This plot took 100 hours

Computational cost

- Exploration necessary to find the best configuration
- No rules for "good" design decisions
- Exploration ultimately means trying all solutions
  - This is extremely compute-intensive
- Full exploration of the design space we proposed:
  1. Time: 381 node-years
  2. Energy: 602.78 MWh



Enough energy for $\sim$100 Dutch persons for 1 year

Conclusion

"**AlphaZero is a usable framework** to enable self-play reinforcement learning for a Hive playing engine. However, there are **no rules** to discern between good and bad **design decisions**. Consequently, the **cost** of exploring the design space can quickly become **prohibitive**."

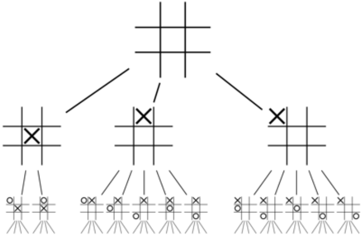
Backup slides

Training infrastructure: building blocks

1. NN: takes a board state s, and produces:
   - A policy vector $\vec{p}$: What moves are *probably* good?
   - A value $v \in [-1, 1]$: Who is winning from this position?

Training infrastructure: building blocks

1. NN: takes a board state s, and produces:
   - A policy vector $\vec{p}$: What moves are *probably* good?
   - A value $v \in [-1, 1]$: Who is winning from this position?
2. MCTS: Finds promising moves by exploring a game tree
   - Outputs an 'improved policy vector' $\vec{\pi}$: What moves are good?



Training infrastructure: core idea

Algorithm 1 Training the NN through self-play

procedure SELFPLAY
  for *iter* $\leftarrow$ 1 to *numIterations* **do**
    for *ep* $\leftarrow$ 1 to *numGames* **do**
      $gameData \leftarrow \text{playGame}(f_\theta)$
      $trainData.\text{append}(gameData)$
    $f_{\theta,\text{new}} \leftarrow \text{trainNN}(trainData)$ $\quad \triangleright\ \mathcal{L} = (z - v)^2 - \vec{\pi}^T \log \vec{p} + c\|\theta\|^2$
    if $f_{\theta,\text{new}}$ outperforms f_θ **then** $f_\theta \leftarrow f_{\theta,\text{new}}$

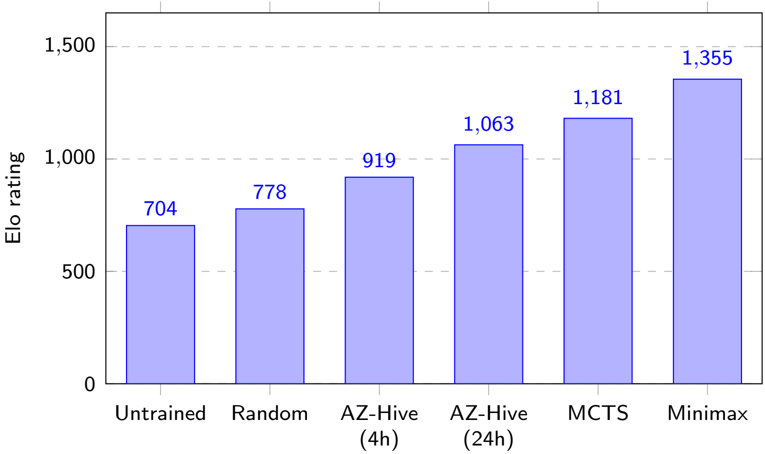
procedure PLAYGAME
  while !gameEnded(s) **do**
    $\vec{\pi} \leftarrow \text{MCTS}(s, f_\theta)$
    $gameData.\text{append}((s, \vec{\pi}, z))$
    $bestAction \sim \vec{\pi}$
    $s \leftarrow \text{playMove}(s, bestAction)$
  return gameData

AZ-Hive Implementation

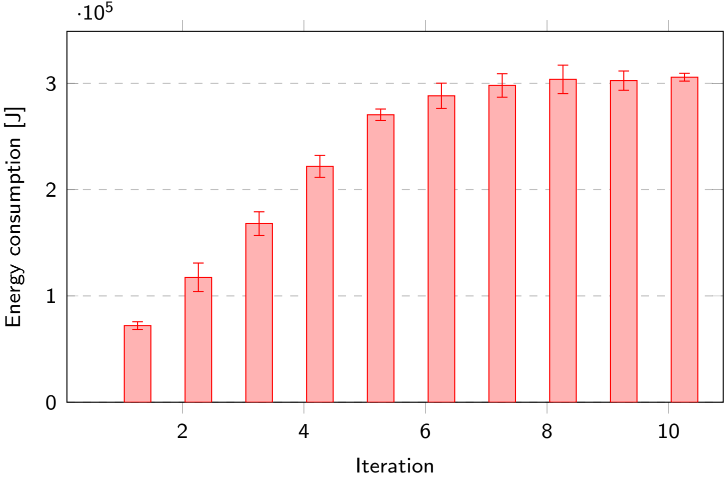
What do we need for AlphaZero?

- Implementation of the game to explore valid moves
- Communication between the training infrastructure and the game implementation

Competitive, but not the best...



Energy consumption analysis

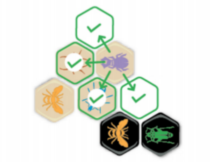


Hive: game rules

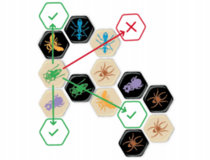
- First move of both players: place a tile (adjacent)
- Then, players take turns to *place* or *move* a tile.
  - Newly placed tiles may not be adjacent to any tile of the enemy
  - When moving a tile, it may not break the *Hive*

Hive: game rules (movement)

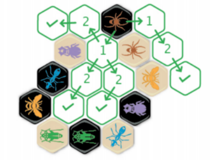
(a) Queen Bee



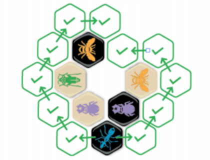
(b) Beetle



(c) Grasshopper

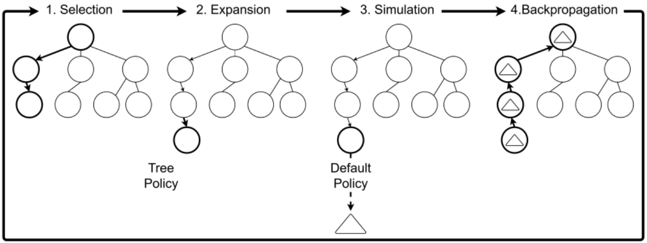


(d) Spider



(e) Soldier Ant

MCTS



Selection rule:

$$\mathrm{UCT}(j) = \frac{w_j}{n_j} + C_p \sqrt{\frac{2 \ln n}{n_j}} \tag{1}$$

MCTS (AlphaZero)

- Rather than random playouts, we use a NN to guide the search
- Selection rule:

$$\text{UCT}_{\text{modified}} = Q(s,a) + C_p \cdot P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \qquad (2)$$

- After some simulations, MCTS outputs a vector $\vec{\pi}$ s.t.

$$\pi_a \propto N(s,a)^{\frac{1}{\tau}} \qquad (3)$$

Self-play reinforcement learning

- 3 stages:
  1. Generate training data through self-play

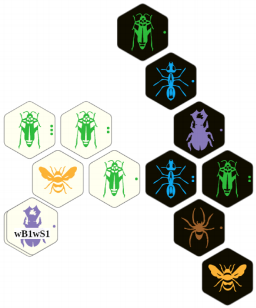
$$D_T = \{(s_t, \vec{\pi}_t, z_t) \mid t \in \mathbb{N}\} \tag{4}$$

  2. Train the NN using this data

$$l = (z - v)^2 - \vec{\pi}^T \log \vec{p} + c\|\theta\|^2 \tag{5}$$

  3. Pit the new model against its previous iteration

Representing the Hive as a board



(a) Game state with stacked tiles.

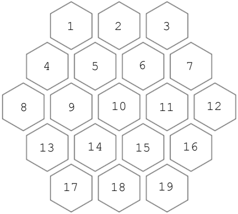
0	0	0	0	0	0	0	0	0	0
0	..	..	..	..	..	..	..	..	0
0	..	0	9	0	0	0	0	..	0
0	..	0	0	10	0	0	0	..	0
0	..	2	2	0	8	0	0	..	0
0	..	0	5	2	10	9	0	..	0
0	..	0	34	0	0	7	0	..	0
0	..	0	0	0	0	0	6	..	0
0	..	..	..	..	..	..	..	..	0
0	0	0	0	0	0	0	0	0	0

(b) Encoding of state with stacked tiles.

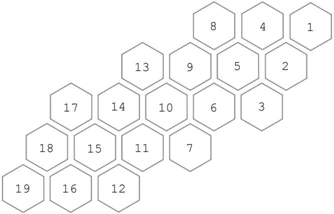
Invariance under rotation & reflection

The naive solution does not keep adjacency properties of the Hive

1	2	3	0	0
4	5	6	7	0
8	9	10	11	12
0	13	14	15	17
0	0	17	18	19



0	0	8	4	1
0	13	9	5	2
17	14	10	6	3
18	15	11	7	0
19	16	12	0	0



Invariance under rotation & reflection (cont'd)

We can perform a 60 degrees rotation by:

1. Performing a clockwise rotation of 90 degrees
2. Shifting the rows to restore the adjacency properties of the board

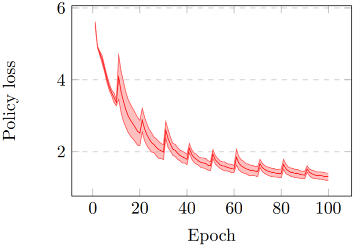
Neural network architecture

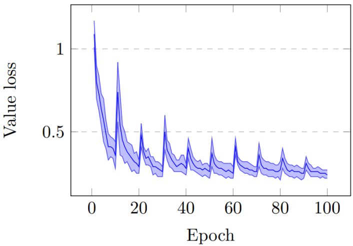
- Convolutional Neural Network (CNN) of either 4, 6, or 8 layers
- Input: The state of a board (depends on board representation)
- Each convolutional block applies the following operations:
  1. Convolution (256 filters of size 3×3 with stride 1)
  2. Batch normalisation
  3. Rectifier nonlinearity (ReLU) activation function
- The output of the convolutional layers is passed into two heads:
  1. Policy head: 2 fully connected layers + softmax
  2. Value head: 2 fully connected layers + Hyperbolic tangent activation function (tanh)

Hyperparameters

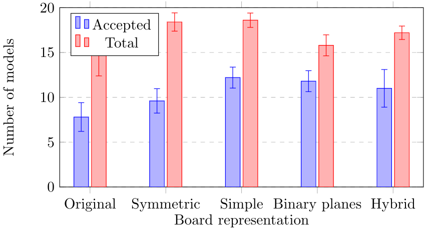
Hyperparameter name	Value of hyperparameter
numIters	20
numEpisodes	100
numMCTSSims	25
updateThreshold	0.5
cpuct	0.8
epochs	10
batchSize	64
numItersForTrainExamplesHistory	20

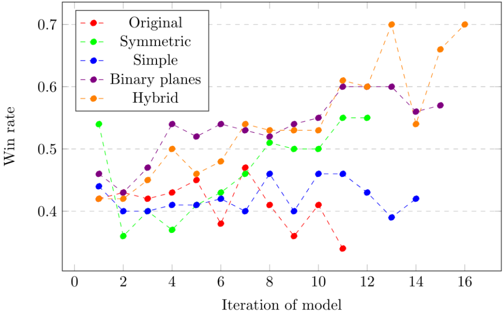
Empirical evaluation: Training infrastructure correctness (SQ1)



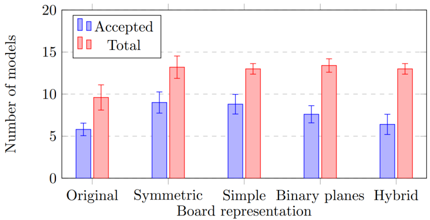


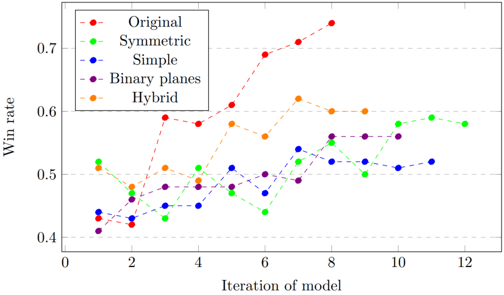
Empirical Analysis: tile-relative



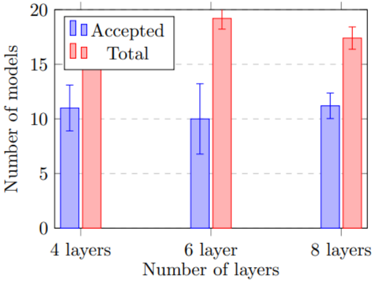


Empirical Analysis: absolute coordinate (SQ2)

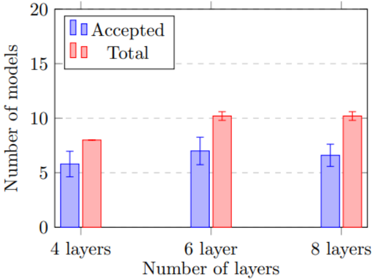




Empirical Analysis: NN architecture + invariance (SQ2)

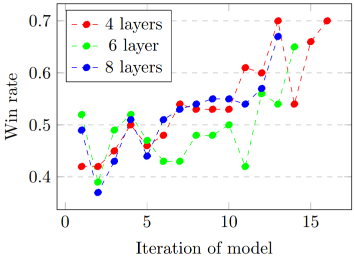


(a) No exploitation

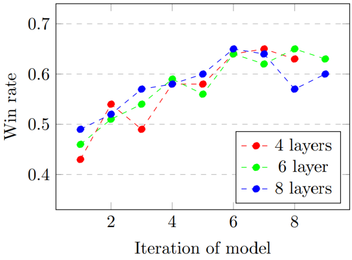


(b) Exploiting invariance

Empirical Analysis: NN architecture + invariance (SQ2)



(a) No exploitation



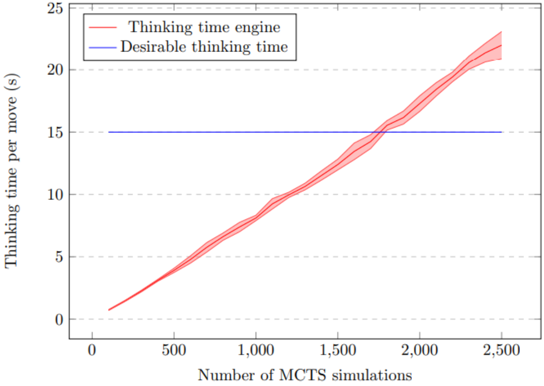
(b) Exploiting invariance

Tournament play with traditional approaches

- How does our engine compare against traditional approaches?
- Set up a tournament against Minimax and classic MCTS
- Maintain Elo rating $R(\cdot)$ throughout the tournament

$$P(\text{A defeats B}) = \frac{1}{1 + 10^{c_{\text{elo}} \cdot (R(B) - R(A))}} \tag{6}$$

Empirical Analysis: Suitability real-life scenario (SQ4)



Contributions

- Designed and implemented the training infrastructure
- Developed a Hive-playing engine that learns the game without any human knowledge
- Defined and partially explored a design space, and analyzed a subset of the configurations
- Analyzed the performance and usability of our engine in a real-life scenario