UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Logic-based AI for Interpretable Board Game Winner Prediction with Tsetlin Machine

Giri, C.; Granmo, O.-C.; van Hoof, H.; Blakely, C.D.

[Link to publication](#)

Citation for published version (APA):

Logic-based AI for Interpretable Board Game Winner Prediction with Tsetlin Machine

Charul Giri
Centre for AI Research
University of Agder
Grimstad, Norway
charul.giri@uia.no

Ole-Christoffer Granmo
Centre for AI Research
University of Agder
Grimstad, Norway
ole.granmo@uia.no

Herke van Hoof
Informatics Institute
University of Amsterdam
Amsterdam, the Netherlands
h.c.vanhoof@uva.nl

Christian D. Blakely
AI and Real-Time Analytics
PwC Switzerland
Zurich, Switzerland
christian.blakely@ch.pwc.com

Abstract—Hex is a turn-based two-player connection game with a high branching factor, making the game arbitrarily complex with increasing board sizes. As such, top-performing algorithms for playing Hex rely on accurate evaluation of board positions using neural networks. However, the limited interpretability of neural networks is problematic when the user wants to understand the reasoning behind the predictions made. In this paper, we propose to use propositional logic expressions to describe winning and losing board game positions, facilitating precise visual interpretation. We employ a Tsetlin Machine (TM) to learn these expressions from previously played games, describing where pieces must be located or not located for a board position to be strong. Extensive experiments on 6×6 boards compare our TM-based solution with popular machine learning algorithms like XGBoost, InterpretML, decision trees, and neural networks, considering various board configurations with 2 to 22 moves played. On average, the TM testing accuracy is 92.1%, outperforming all the other evaluated algorithms. We further demonstrate the global interpretation of the logical expressions, and map them down to particular board game configurations to investigate local interpretability. We believe the resulting interpretability establishes building blocks for accurate assistive AI and human-AI collaboration, also for more complex prediction tasks.

Index Terms—Tsetlin Machine, Winner Prediction, Interpretable AI, Hex

I. Introduction

Board game winner prediction is a critical part of state-of-the-art AIs for board game playing, such as AlphaZero [22]. To guide a Monte-Carlo tree search towards a winning move, for instance, one needs to assess intermediate board configurations. Lately, state-of-the-art solutions use deep neural network architectures to evaluate board configurations and to propose which move sequences to explore. Such black-box algorithms achieve unsurpassed accuracy in board game position evaluation. However, their complex nature makes them hard to interpret as they do not reveal their internal decision mechanism. There are approximate post-hoc methods for explaining specific outputs such as LIME, yet, being approximate, they can be arbitrarily erroneous [18].

As AI integrates into our daily lives, interpretability becomes increasingly crucial. Whereas black-box methods can have significant ethical, legal, and social implications by slipping through biased and wrong decisions [14], algorithms that explain their output facilitate societal acceptance by being

transparent [15]. An AI-powered recruitment tool developed by Amazon was, for instance, found to be gender-biased and therefore scrapped [7].

Lack of interpretation is problematic because it hinders quality assurance and human-computer collaboration. In contrast, an interpretable system enables human assessment of how the AI reasons, for overseeing the AI decision process. If the AI, in addition, can provide transparent decision alternatives, it becomes possible to augment AI-based decision-making with human intelligence, through collaboration. Indeed, humans and AI collaborating may be more capable than each alone. As exemplified by Hipp et al. [12], the combination of the tactical superiority of computers and the strategical superiority of humans led to the victory of amateur players assisted by AI in Cyborg Chess[1], facing grandmasters and supercomputers.

The above findings are in contrast to the traditional view that complex black-box models are necessary to maximize accuracy, creating a trade-off between interpretability and accuracy. Further, Rudin suggests that interpretable models can perform competitively if the representation of the problem is adequate [18].

Finding competitive interpretable representations for complex board games like Hex is an unresolved challenge. Contemporary interpretable techniques, like decision trees and XGBoost, employ greedy learning strategies or rely on model simplifications. They, therefore, struggle with learning complex patterns such as board configurations that lead to a win or loss.

In this paper, we propose a competitive interpretable representation for the board game Hex, learned by a Tsetlin Machine (TM) [9]. TM is a recent technique for pattern recognition that uses a team of Tsetlin automata [23] to learn patterns expressed in propositional logic. TMs have obtained competitive accuracy, memory footprint, and learning speed on several benchmark datasets [1]. They have been particularly successful in natural language processing, including explainable aspect-based sentiment analysis [24]. Being based on

---

[1]Cyborg Chess, also called Advanced Chess, is a form of chess where the human player is assisted by a computer chess program to explore potential moves more effectively.

finite-state automata, they further support Markov chain-based convergence analysis [26].

The contributions of the paper can be summarized as follows. We propose an approach to representing board game configurations in Hex using propositional logic. The expressions describe the presence and absence of pieces with conjunctive clauses (AND-rules). This representation allows us to capture board configurations of arbitrary complexity, inspired by human logical reasoning [17]. Our approach thus falls in the domain of rule-based classifiers, which provide humans with easy-to-understand interpretation of the classifiers' decision making. We further demonstrate how our representation can be aggregated and visualized for global and local interpretation.

II. BOARD GAME WINNER PREDICTION IN HEX

A. Problem Overview

Hex is an abstract two-player strategy board game played on an $n \times n$ rhombus board. In brief, the players alternate with placing pieces on empty board slots seeking to form a connection from one edge to the opposite edge. The goal of the game is to form an unbroken chain of pieces from one end of the board to the other. The first player (Black) needs to connect the top edge with the bottom edge, while the second player (White) must form a connection between the left and the right edges.

Predicting the conclusion of the game from any intermediate board state is a challenging task. For example, a 11×11 Hex board has 2.4×10^{56} legal positions, and a typical board configuration provides almost three times as many move options compared to Chess [5]. The resulting large branching factor makes it only possible to do relatively shallow game tree searches. As the game approaches the end, predicting the winner through search becomes more feasible, however, from the opening and intermediate board states, only state-of-the-art neural networks are capable of sufficiently accurate winner prediction. Note that unlike Go, Hex cannot end in a draw, producing a two-class prediction problem.

B. Hex Game Strategies

To facilitate the interpretability study later in the paper, we here provide a brief overview of basic playing strategies for Hex. In Hex, a good offense is equal to a good defense. If you have completed your connection, it will automatically mean that you have prevented the opponent from completing her connection. And, if you have prevented your opponent's connection, you have, as a consequence, simultaneously completed your own connection. This is why a game of Hex never results in a draw [19].

Fig. 1 shows a completed Hex game played on an 11×11 board. As seen, Black wins the game by forming an unbroken chain from the top edge to the bottom edge. Notice how the continuous chain from top to bottom prevents White from forming a continuous chain from left edge to right edge.

In Hex, the first player always has the advantage. This is because in a finite game, there exists a strategy whereby a first
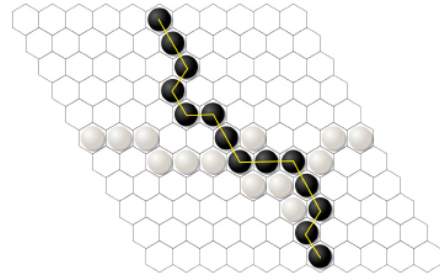

Fig. 1: Completed Hex game on an 11×11 board where Black is the winner
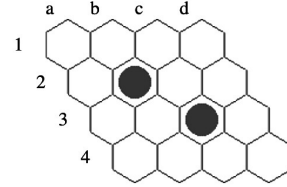

Fig. 2: Bridge pattern

mover cannot lose [20]. The following swap rule balances this bias.

Swap Rule: After the first player (Black) has made its move, the second player can choose either to swap sides, taking the first move as its own, or to respond to the first player, continuing playing as White.

Virtual connections and bridge patterns form the basis for playing competitively, illustrated in Fig. 2.

Virtual Connection: A virtual connection secures a chain without putting the pieces directly adjacent to each other, lining out a strong chain. The most common type of virtual connection is a bridge pattern.

Bridge Pattern: A bridge pattern is created when two pieces of the same color are placed so that they are separated only by two neighbouring empty hex cells. In Fig. 3, for example, the link from b2 to c3 is secured by Black, despite the pieces not being adjacent to each other. If White tries to break the link by putting a pieces in b3, for instance, Black can still connect through c2, and vice-versa [4].
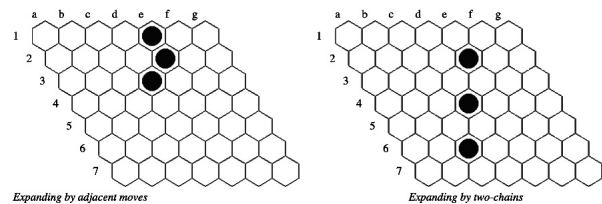

Fig. 3: Expansion through bridge pattern

C. Current Approaches

In 2000, the Hexy [2] program won the first Computer Olympiad for Hex. It exploited alpha-beta search and H-search

to find and combine virtual connections to form more complex patterns. Later, MoHex [3] became champion in the 2009 Hex Computer Olympiad. MoHex is a Monte Carlo tree search (MCTS) based approach. It combines MCTS with H-search and ICA to prune the game tree and speed up the tree search. It further exploits Depth-first Proof Number search to find the best moves in parallel. MoHex 2.0 [13] is an improvement over MoHex, using the MiniMax algorithm to learn patterns for weighing the MCTS simulations.

From 2009 to 2016, different versions of MoHex won all of the Computer Olympiads. In 2016, however, inspired by the success of AlphaGo [21], researchers started applying neural networks to the game of Hex. NeuroHex [25] applied Deep Q-Learning with experience replay. Another program called MoHexNet [11] used depth-1 trees found by NeuroHEX, and combined the depth-1 trees with MCTS. As a result, MoHexNet won the 2016 Computer Olympiad. Since then, recent work introduced various ways of improving MoHexNet, leading to MoHex-CNN [10] and MoHex-3HNN [8], which won the Computer Olympiads in the following years.

Extensive research has been carried out on the policy evaluation methods for predicting next best moves, playing out the entire game with MCTS. While a MCTS search by itself is interpretable, the neural network-based evaluation of board positions are difficult to interpret due to the lack of transparency of neural networks. In our present paper, we focus on predicting the winner directly, from any board position, without playing out the game. Our goal is to address the lacking transparency of the neural network-based approaches, hindering for instance human-AI collaboration.

Our paper sheds light on this matter by introducing an interpretable method for predicting the winner for 6×6 Hex boards. Also, Hex, just like Go, is a pattern oriented game. So this research can be regarded as a step in the direction of developing AI assistance for the game of Go as well.

III. INTERPRETABLE WINNER PREDICTION WITH LOGICAL RULES

In this section, we present the basics of the Tsetlin Machine, to lay the foundation for our board game winner prediction approach. We further describe our method for representing winning and losing board game configurations.

A. Pattern Representation

A TM in its simplest form takes a feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_o] \in \{0,1\}^o$ of o propositional values as input and assigns the vector a class $\hat{y} \in \{0,1\}$. To minimize
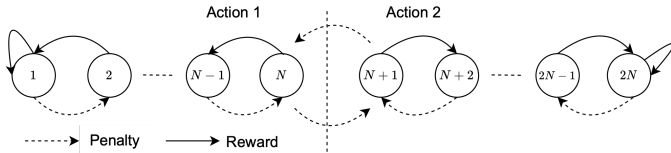

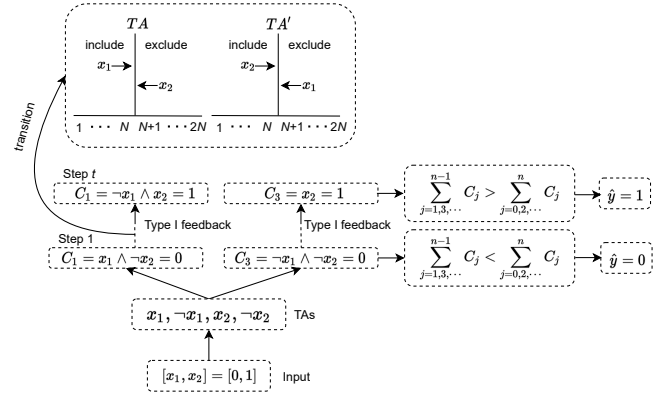Fig. 4: A two-action Tsetlin Automaton with $2N$ states.

Fig. 5: TM learning dynamics for an XOR-gate training sample, with input $(x_1 = 0, x_2 = 1)$ and output target $y = 1$.

classification error, the TM produces n self-contained patterns. The input vector $\mathbf{x}$ first provides the literal set $L = \{l_1, l_2, \ldots, l_{2o}\} = \{x_1, x_2, \ldots, x_o, \neg x_1, \neg x_2, \ldots, \neg x_o\}$, consisting of the input features and their negations. By selecting subsets $L_j \subseteq L$ of the literals, the TM can build arbitrarily complex patterns through ANDing the literal selection to form conjunctive clauses:

$$C_j(\mathbf{x}) = \bigwedge_{l_k \in L_j} l_k. \tag{1}$$

Above, $j \in \{1, 2, \ldots, n\}$ refers to a particular clause C_j and $k \in \{1, 2, \ldots, 2o\}$ refers to a particular literal l_k. As an example, the clause $C_j(\mathbf{x}) = x_1 \wedge \neg x_2$ consists of the literals $L_j = \{x_1, \neg x_2\}$ and evaluates to 1 when $x_1 = 1$ and $x_2 = 0$.

B. Representing Board Game Configurations

We represent a board game configuration as a clause, with each literal referring to whether a piece of a certain color is present or absent in a specific board location. The following is an example of such a clause, capturing the 6×6 Hex board pattern visualized in Fig. 6:

$x_{10} \wedge x_{21} \wedge \neg x_6 \wedge \neg x_{14} \wedge \neg x_{17} \wedge \neg x_{33} \wedge \neg x_{38} \wedge \neg x_{51} \wedge \neg x_{59} \wedge \neg x_{60} \wedge \neg x_{66} \wedge \neg x_{71} \wedge \neg x_{72}$.

Features $x_1, x_2, \ldots, x_{36}$ refer to the Black pieces, index starting from the top left of the board and increasing along the rows. For instance, literal x_{10} in the clause specifies that there must be a Black piece in position *d2*, while literal $\neg x_6$ signify that a Black piece cannot be located in location *f1*. Correspondingly, features $x_{31}, x_{32}, \ldots, x_{72}$ represent the White pieces. This particular example pattern provides evidence for Black winning because the positions *c4* and *d2* are forming a bridge connection. However, the final prediction is not based on a single individual pattern but by sharing the knowledge of multiple learned patterns.
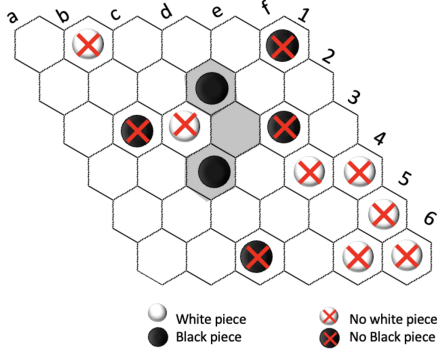
Fig. 6: Pattern structure for a sample clause

White piece / No white piece
Black piece / No Black piece

C. Tsetlin Automata Teams

The fundamental learning unit of TM is a Tsetlin Automaton(TA). The TM assigns one TA per literal l_k per clause C_j to build the clauses. The TA assigned to literal l_k of clause C_j decides whether l_k is *Excluded* or *Included* in C_j. Fig. 4 depicts a two-action TA with $2N$ states. For states 1 to N, the TA performs action *Exclude* (Action 1), while for states $N+1$ to $2N$ it performs action *Include* (Action 2). As feedback to the action performed, the environment responds with either a Reward or a Penalty. If the TA receives a Reward, it moves deeper into the side of the action. If it receives a Penalty, it moves towards the middle and eventually switches action.

With n clauses and $2o$ literals, we get $n \times 2o$ TAs. We organize the states of these in a $n \times 2o$ matrix $A = [a_k^j] \in \{1, 2, \ldots, 2N\}^{n \times 2o}$. We will use the function $g(\cdot)$ to map the automaton state a_k^j to Action 0 (*Exclude*) for states 1 to N and to Action 1 (*Include*) for states $N+1$ to $2N$: $g(a_k^j) = a_k^j > N$.

We can connect the states a_k^j of the TAs assigned to clause C_j with its composition as follows:

$$C_j(\mathbf{x}) = \bigwedge_{l_k \in L_j} l_k = \bigwedge_{k=1}^{2o} \left[g(a_k^j) \Rightarrow l_k \right]. \quad (2)$$

Here, l_k is one of the literals and a_k^j is the state of its TA in clause C_j. The logical *imply* operator $\Rightarrow$ implements the *Exclude/Include* action. That is, the *imply* operator is always 1 if $g(a_k^j) = 0$ (*Exclude*), while if $g(a_k^j) = 1$ (*Include*) the truth value is decided by the truth value of the literal.

D. Classification

The odd-numbered half of the clauses vote for class $\hat{y} = 0$ and the even-numbered half vote for $\hat{y} = 1$. Classification is performed as a majority vote:

$$\hat{y} = 0 \leq \sum_{j=1,3,\ldots}^{n-1} \bigwedge_{k=1}^{2o} \left[g(a_k^j) \Rightarrow l_k \right] - \sum_{j=2,4,\ldots}^{n} \bigwedge_{k=1}^{2o} \left[g(a_k^j) \Rightarrow l_k \right]. \quad (3)$$

As such, the odd-numbered clauses have positive polarity, while the even-numbered ones have negative polarity. As an example, consider the input vector $\mathbf{x} = [0, 1]$ in the lower part of Fig. 5. The figure depicts two clauses of positive polarity,

INPUT	CLAUSE	1		0	
	LITERAL	1	0	1	0
INCLUDE LITERAL	P(REWARD)	$\frac{s-1}{s}$	NA	0	0
	P(INACTION)	$\frac{1}{s}$	NA	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(PENALTY)	0	NA	$\frac{1}{s}$	$\frac{1}{s}$
EXCLUDE LITERAL	P(REWARD)	0	$\frac{1}{s}$	$\frac{1}{s}$	$\frac{1}{s}$
	P(INACTION)	$\frac{1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$	$\frac{s-1}{s}$
	P(PENALTY)	$\frac{s-1}{s}$	0	0	0

TABLE I: Type I Feedback

INPUT	CLAUSE	1		0	
	LITERAL	1	0	1	0
INCLUDE LITERAL	P(REWARD)	0	NA	0	0
	P(INACTION)	1.0	NA	1.0	1.0
	P(PENALTY)	0	NA	0	0
EXCLUDE LITERAL	P(REWARD)	0	0	0	0
	P(INACTION)	1.0	0	1.0	1.0
	P(PENALTY)	0	1.0	0	0

TABLE II: Type II Feedback

$C_1(\mathbf{x}) = x_1 \wedge \neg x_2$ and $C_3(\mathbf{x}) = \neg x_1 \wedge \neg x_2$ (the negative polarity clauses are not shown). Both of the clauses evaluate to zero, leading to class prediction $\hat{y} = 0$.

E. Learning

The upper part of Fig. 5 illustrates learning. A TM learns online, processing one training example $(\mathbf{x}, y)$ at a time. Based on $(\mathbf{x}, y)$, the TM rewards and penalizes its TAs, which amounts to incrementing and decrementing their states. There are two kinds of feedback: Type I Feedback encourages clauses to encode frequently-encountered patterns and Type II Feedback increases the discrimination power of the patterns.

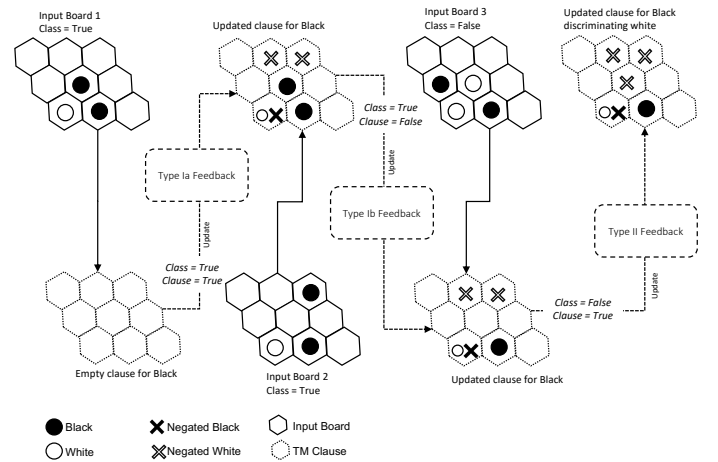Black / White / Negated Black / Negated White / Input Board / TM Clause

Fig. 7: TM learning dynamics for three Hex board positions

Type I feedback is given stochastically to clauses with positive polarity when $y = 1$ and to clauses with negative polarity when $y = 0$. Conversely, Type II Feedback is given stochastically to clauses with positive polarity when $y = 0$ and to clauses with negative polarity when $y = 1$. The probability of a clause being updated is based on the vote sum v: $v = \sum_{j=1,3,\ldots}^{n-1} \bigwedge_{k=1}^{2o} \left[g(a_k^j) \Rightarrow l_k \right] - \sum_{j=2,4,\ldots}^{n} \bigwedge_{k=1}^{2o} \left[g(a_k^j) \Rightarrow l_k \right]$. The voting error is calculated as:

$$\epsilon = \begin{cases} T - v & y = 1 \\ T + v & y = 0. \end{cases} \quad (4)$$

Here, T is a user-configurable voting margin yielding an ensemble effect. The probability of updating each clause is $P(\text{Feedback}) = \frac{\epsilon}{2T}$.

A random sampling from $P(\text{Feedback})$ decides which clauses to update. The following TA state updates of the chosen clauses' can be formulated as matrix additions, subdividing Type I Feedback into feedback Type Ia and Type Ib:

$$A_{t+1}^* = A_t + F^{II} + F^{Ia} - F^{Ib}. \quad (5)$$

Here, $A_t = [a_k^j] \in \{1, 2, \ldots, 2N\}^{n \times 2o}$ contains the states of the TAs at time step t and A_{t+1}^* contains the updated state for time step $t + 1$ (before clipping). The matrices $F^{Ia} \in \{0, 1\}^{n \times 2o}$ and $F^{Ib} \in \{0, 1\}^{n \times 2o}$ contains Type I Feedback. A zero-element means no feedback and a one-element means feedback. As shown in Table I, two rules govern Type I feedback:

- **Type Ia Feedback** is given with probability $\frac{s-1}{s}$ whenever both clause and literal are 1-valued.[2] It penalizes *Exclude* actions and rewards *Include* actions. The purpose is to remember and refine the patterns manifested in the current input **x**. This is achieved by increasing selected TA states. The user-configurable parameter s controls pattern frequency, i.e., a higher s produces less frequent patterns.
- **Type Ib Feedback** is given with probability $\frac{1}{s}$ whenever either clause or literal is 0-valued. This feedback rewards *Exclude* actions and penalizes *Include* actions to coarsen patterns, combating overfitting. Thus, the selected TA states are decreased.

The matrix $F^{II} \in \{0, 1\}^{n \times 2o}$ contains Type II Feedback to the TAs, given per Table III.

- **Type II Feedback** penalizes *Exclude* actions to make the clauses more discriminative, combating false positives. That is, if the literal is 0-valued and the clause is 1-valued, TA states below $N + 1$ are increased. Eventually the clause becomes 0-valued for that particular input, upon inclusion of the 0-valued literal.

The final updating step for training example $(\mathbf{x}, y)$ is to clip the state values to make sure that they stay within value 1 and $2N$:

$$A_{t+1} = clip \left(A_{t+1}^*, 1, 2N \right). \quad (6)$$

[2]Note that the probability $\frac{s-1}{s}$ is replaced by 1 when boosting true positives.

(a) Type Ia Feedback

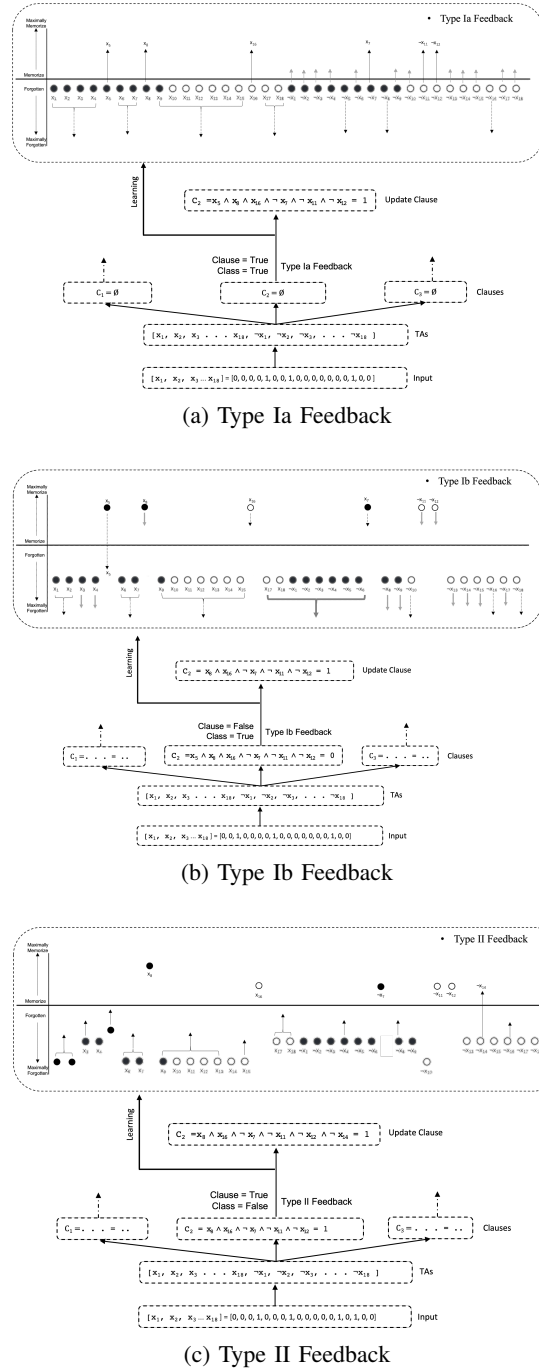(b) Type Ib Feedback

(c) Type II Feedback

Fig. 8: Visualization of Tsetlin Automata state updates

For example, both of the clauses in Fig. 5 receives Type I Feedback over several training examples, making them resemble the input associated with $y = 1$.

F. Learning Board Configuration Patterns

Fig. 7 demonstrates how Type Ia, Type Ib, and Type II Feedback operates on board game configurations to update a clause. We use 3×3 boards to visualize learning. The starting point is an empty clause, which is updated based on an input board that led to a Black win (upper left). Since the clause

CLASS	GAME BOARDS
BLACK	175968
WHITE	111826
TOTAL	287794

TABLE III: Dataset distribution

METHOD	HYPERPARAMETER	TRAINING ACCURACY	TESTING ACCURACY
NAÏVE BAYES	DEFAULT	72.02	72.02
DECISION TREE	MAX DEPTH=100	100	87.1
K-NEAREST NEIGHBOUR	K=1	100	84
XGBOOST		84.88	88.90
INTERPRETML(EBC)	DEFAULT	74.56	74.33
NEURAL NETWORK		97.6	90.7
TSETLIN MACHINE	CLAUSES = 10000 T = 8000 S = 100 MAX WEIGHT=255	95.6	92.1

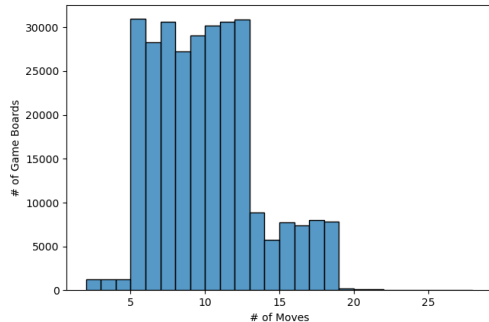TABLE IV: Comparison with other machine learning techniques

Fig. 9: Distribution of games per number of moves played

is empty, it evaluates to True. Type Ia Feedback then makes the clause resemble the input board by introducing literals from the board, building a pattern for winning positions. The next new winning board configuration given as input does not match the clause, leading to Type Ib Feedback that removes literals from the clause. Finally, a losing board position is given as input. Since the clause evaluates to True, Feedback Type II is given, refining the clause to become more discriminative (upper left). It is the states of the underlying Tsetlin Automata that drive these changes, illustrated by Fig. 8).

IV. EMPIRICAL EVALUATION

In this section we evaluate our proposed method and demonstrate how to interpret the board configuration patterns produced.

A. Dataset

The dataset we use for evaluation consists of $287,000$ game board configurations, which were created through self-play using a modified version of MoHex2.0, with MCTS and data augmentation. The games lasted for up to 28 moves, with the majority of games ending before 20 moves (Fig. **??**).

Since TM requires input features to be binarized, each board is represented with a 72 bit vector, as described in the previous section.

B. Results

Our experimental setup is as follows. As previously explained in Section III, a TM has two hyper-parameters T and s, along with the number of clauses n. For our experiments we initialized our TM with $10,000$ clauses and set the other hyperparameter T and s to 8000 and 100, respectively. As a general rule, TMs perform well with a T value equal to 80%

of the total number of clauses per class, leaving only s to be tuned. We found an appropriate s value through a limited manual trial and error search, so performance may possibly improve trough a more thorough automated grid search. We used 67% of the data to train the TM for 200 epochs, and the remaining 33% of the data for evaluating prediction accuracy.

Table IV contains the results of our experiments, including a comparison with six popular machine learning algorithms. For a fair comparison, we used the same training and testing datasets for all of the algorithms. Each algorithm was first trained using default sklearn parameters, and the top-performing ones were then fine-tuned to maximize performance. As seen in the table, we obtained an average testing accuracy of 92.1% with our approach, outperforming the other interpretable methods like Decision Trees, XGBoost [6], and InterpretML [16] by a wide margin.
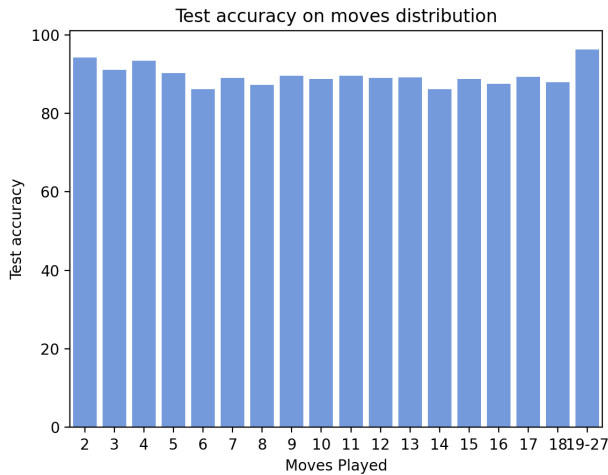
Also, our technique performs better than a fine-tuned black-box neural network (NN) configuration. The NN consisted of five layers with Sigmoid activation functions, trained with the Adam optimizer. We further utilized binary Crossentropy loss function, and exploited L_2 regularization and dropout to reduce overfitting. Still, the maximum accuracy we obtained with the NN was about 1.5% lower than our TM-based approach.

To investigate whether accuracy varies depending on the stage of a game (e.g., beginning, middle, or end), we grouped the data according to the number of moves played (Fig. 9). Fig. 10 summarises how training and testing accuracy varies by the number of moves played. Observe how the performance of TM remains more or less unaffected by number of moves played, indicating a strong ability to predict the winner at all stages of a game. That is, our method is not only able to predict the winner accurately later in a game, when more information is available (intermediate and ending games), it can also predict the winner very early in the game, signifying the importance of the first moves played.

Earlier in this paper we discussed that TM learns patterns by forming clauses, so we performed a precision analysis of the clauses. Fig. 11 illustrates the precision histograms for positive polarity and negative polarity clauses. It can be observed that a majority of the clauses fall in a fairly precise group, which signify that the predictions are made with a higher confidence.
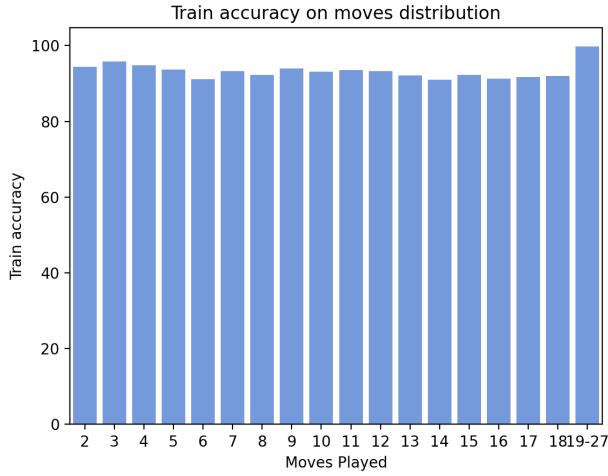
(a) Training accuracy per #moves played

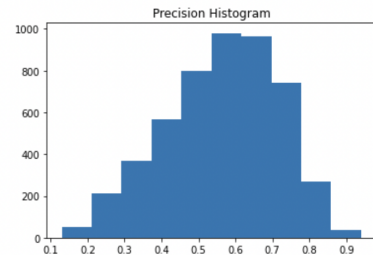(b) Testing accuracy per #moves played

Fig. 10: Accuracy at different stages of a game

(a) Positive Polarity Clauses

(b) Negative Polarity Clauses

Fig. 11: Clause precision

(a) Positive polarity clauses

(b) Negative polarity clauses

Fig. 12: Ten most impactful clauses per polarity
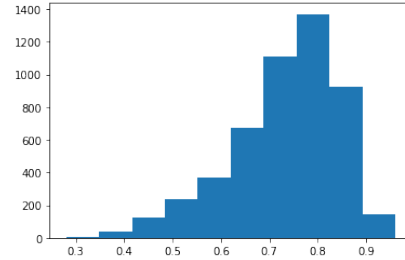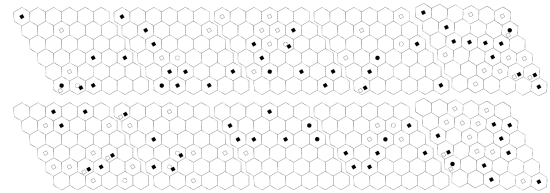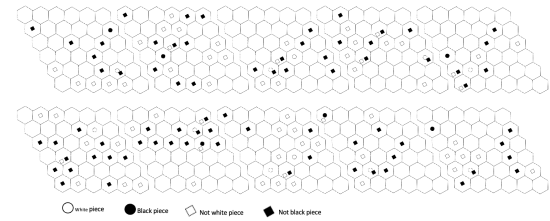
C. Interpretation

We now investigate how to visualize the clauses for global and local interpretation.

1) Global Interpretability: Global interpretability [15] describes how a model behaves in general. It takes a holistic view of the features, providing insight into what features are informative overall and how they interact in the model.

In a game of Hex, analysing the interaction of features is crucial for understanding what constitutes a winning pattern. Each clause a TM learns captures one particular feature interaction, as displayed in Fig. 12. The figure contains the ten most impactful positive polarity clauses (Fig. 12a) and the ten most impactful negative ones (Fig. 12b), visualized as board patterns. The positive polarity clauses capture winning patterns for Black, while the negative polarity clauses represent losing patterns. The clauses C_j have been selected by combining precision and data coverage:

$$Score(j) = Precision(j)^{\alpha} \times Coverage(j) \qquad (7)$$

calculated from the True Positive (TP), False Negative(FN), and False Positive (FP) rates:

$$Precision = \frac{TP}{TP+FP} \qquad (8)$$

$$Coverage = \frac{TP}{TP+FN}. \qquad (9)$$

The α parameter is for weighing precision against coverage. That is, a higher α puts more emphasis on clause precision. Since a majority of the clauses provides high coverage, they tend to have relatively lower precision. Therefore, we use $\alpha = 10$ to single out clauses with high precision, yet providing reasonable coverage.
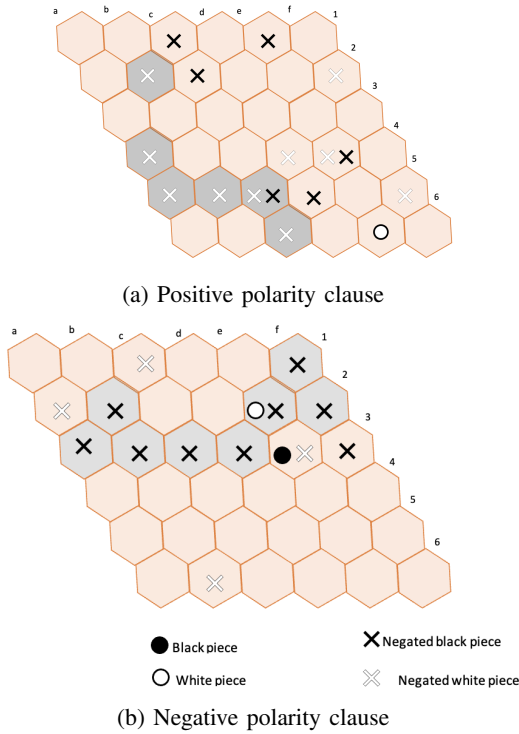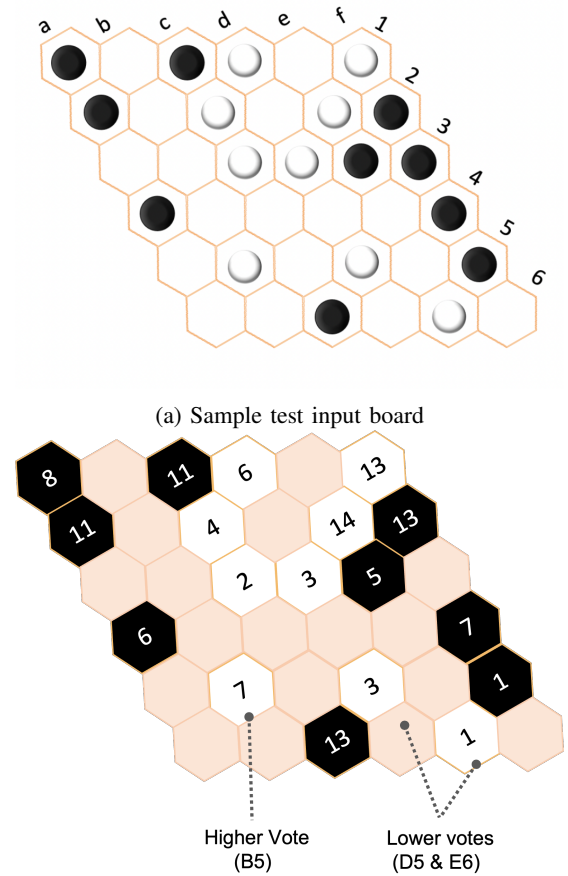
(a) Positive polarity clause

Black piece   ✗ Negated black piece

White piece   ✗ Negated white piece

(b) Negative polarity clause

Fig. 13: Example of clause interpretation

(a) Sample test input board

Higher Vote (B5)   Lower votes (D5 & E6)

(b) Local interpretation of clauses for board in Fig. 14a

Fig. 14: Local aggregated interpretation

Consider, for example, the negative polarity clause in Fig. 13b. This clause predicts loss for Black due to the string of negated Black pieces from *a3* to *f2*. Hence, the clause states that Black at this point not yet has formed a continuous chain from top to bottom.

Now, observe the positive polarity clause in Fig. 13a. This clause predicts a win for Black. As can be seen from the board, White lacks pieces from *a4* to *c6*. Accordingly, the pattern opens up for a potential chain for Black. The missing white pieces also provide evidence for a possible bridge between *a4* and *b2*, increasing the probability of a Black chain forming before the White one.

2) Local Interpretability: Local interpretability explains specific data points, as opposed to global interpretation which characterises the entire model at once [15]. In our case, global interpretation also forms the basis for local interpretation. That is, by combining the pattern information from the individual clauses that evaluate to True for a specific board configuration, more insights about the board configuration is obtained.

Since TM clauses work together to form an accurate prediction, it can in some cases be insufficient to interpret a single clause alone. Instead, we aggregate the information from all the clauses that took part in classifying the input. Then we capture exactly which literals the TM relied on for the given classification. Consider the example input board in Fig. 14a. White went on and won the game from this position, as predicted by the TM. Fig. 14b interprets this TM prediction through aggregating the True clauses. That is, each Hex cell reports the number of times a Black or White piece is assigned

to the cell by one of the clauses that are True for the given board position (only considering non-negated features).

We can see that a higher count is given to Hex cell *b5*. The high count captures that *b5* forms a bridge to *c3* as well as a secure edge link to *a5* and *a6*, which completes the continuous chain for White. It is impossible for Black to break this chain. If Black tries to block White by playing *b4*, White can continue the chain by playing *c4*, or the other the way around. In the next step, if Black tries to block White by playing *a6*, White can complete the chain by playing *a5*, and vice-versa.

Also notice the pattern forming in the lower right of Fig. 14b, with a lower count. Even though White can build a chain here as an alternative, it is a weaker option since Black can block White's path from *f1* to *f6* simply by playing *f6*.

In conclusion, this example demonstrate that the TM not only recognizes individual patterns but is also able to aggregate the patterns to assess chance of winning.

V. CONCLUSION

In this paper, we presented an interpretable winner prediction approach for the game of Hex. By using simple AND rules in propositional logic, we can specify strong and weak board

configurations. Our empirical results dispute the widespread assumption that one cannot escape trading off interpretability against predictive performance. Despite being inherently interpretable, our approach outperforms several other popular ML methods, achieving an accuracy of 92% on test data. Straightforward board configuration patterns enable global and local interpretation, through visualization. By aggregating the active clauses, we can further assess the importance of specific piece formations, revealing the reasons behind our model's predictions.

In our further work we plan to go from winner prediction to predicting the winning move, to solve the game of Hex on larger boards in an interpretable manner.

REFERENCES

[1] Kuruge Darshana Abeyrathna, Bimal Bhattarai, Morten Goodwin, Saeed Rahimi Gorji, Ole-Christoffer Granmo, Lei Jiao, Rupsa Saha, and Rohan K Yadav. Massively parallel and asynchronous tsetlin machine architecture supporting almost constant-time scaling. In *International Conference on Machine Learning*, pages 10–20. PMLR, 2021.

[2] Vadim V Anshelevich. The game of hex: An automatic theorem proving approach to game programming. In *AAAI/IAAI*, pages 189–194, 2000.

[3] Broderick Arneson, Ryan Hayward, and Philip Henderson. Mohex wins hex tournament. *ICGA journal*, 32(2):114, 2009.

[4] Cameron Browne. *Hex Strategy: Making the right connections*. CRC Press, 2009.

[5] Stephan Chalup, Drew Mellor, and Frances Rosamond. The machine intelligence hex project. *Computer Science Education*, 15, 07 2005.

[6] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[7] Jeffrey Dastin. Amazon scraps secret ai recruiting tool that showed bias against women, 2018.

[8] Chao Gao, Kei Takada, and Ryan Hayward. Hex 2018: Mohex3hnn over deepezo. *J. Int. Comput. Games Assoc.*, 41(1):39–42, 2019.

[9] Ole-Christoffer Granmo. The tsetlin machine - A game theoretic bandit driven approach to optimal pattern recognition with propositional logic. *CoRR*, abs/1804.01508, 2018.

[10] Ryan Hayward and Noah Weninger. Hex 2017: Mohex wins the 11x11 and 13x13 tournaments. *ICGA Journal*, 39(3-4):222–227, 2017.

[11] Ryan Hayward, Noah Weninger, Kenny Young, Kei Takada, and Tianli Zhang. Mohex wins 2016 hex 11x11 and 13x13 tournaments.

[12] Jason Hipp, Thomas Flotte, James Monaco, Jerome Cheng, Anant Madabhushi, Yukako Yagi, Jaime Rodriguez-Canales, Michael Emmert-Buck, Michael C Dugan, Stephen Hewitt, et al. Computer aided diagnostic tools aim to empower rather than replace pathologists: Lessons learned from computational chess. *Journal of pathology informatics*, 2, 2011.

[13] Shih-Chieh Huang, Broderick Arneson, Ryan B Hayward, Martin Müller, and Jakub Pawlewicz. Mohex 2.0: a pattern-based mcts hex player. In *International Conference on Computers and Games*, pages 60–71. Springer, 2013.

[14] Davinder Kaur, Suleyman Uslu, Arjan Durresi, Sunil Badve, and Murat Dundar. Trustworthy explainability acceptance: A new metric to measure the trustworthiness of interpretable ai medical diagnostic systems. In *Conference on Complex, Intelligent, and Software Intensive Systems*, pages 35–46. Springer, 2021.

[15] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[16] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

[17] I. Noveck, R. B. Lea, George M. Davidson, and D. O'brien. Human reasoning is both logical and pragmatic. *Intellectica*, 11:81–109, 1991.

[18] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[19] Mark Schachner. The game of hex: a study in graph theory and algebraic topology. *Mathematics REU*, 2019.

[20] Ulrich Schwalbe and Paul Walker. Zermelo and the early history of game theory. *Games and economic behavior*, 34(1):123–137, 2001.

[21] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[22] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[23] Michael Lvovitch Tsetlin. On behaviour of finite automata in random medium. *Avtomat. i Telemekh*, 22(10):1345–1354, 1961.

[24] Rohan Kumar Yadav, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. Human-level interpretable learning for aspect-based sentiment analysis. In *The Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*. AAAI, 2021.

[25] Kenny Young, Gautham Vasan, and Ryan Hayward. Neurohex: A deep q-learning hex agent. In *Computer Games*, pages 3–18. Springer, 2016.

[26] Xuan Zhang, Lei Jiao, Ole-Christoffer Granmo, and Morten Goodwin. On the Convergence of Tsetlin Machines for the IDENTITY- and NOT Operators. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.