Hex Game

by

Christoffer Eigill Prag & Yngve Magnussen

in

IKT457 Learning Systems

Faculty of Technology and Science

Universitetet i Agder

Grimstad, December 2024

Contents

1 Introduction

The use of artificial intelligence techniques has become increasingly challenging due to the growing complexity of board games. Among the various approaches to creating intelligent systems, logic-based methods stand out for their ability to provide structured reasoning and interpretability. This research explores how Graph Tsetlin Machines (GTMs) can be applied to predict the winners of board games, with a focus on the strategic connection game Hex.

The Tsetlin Machine is a logic-based artificial intelligence model that employs propositional logic to mimic human reasoning processes, enhancing memory and decision-making capabilities to achieve optimal results. By extending this logic-driven approach to graph-structured data, the Graph Tsetlin Machine (GTM) enables the representation and analysis of board game states as node-edge connections. This capability positions GTMs as uniquely suited for tasks requiring both local and global interpretability, such as predicting game outcomes based on the evolving state of a board.

This project aims to utilize data from over one million simulated Hex games to predict winners at various stages of gameplay. By constructing feature vectors for nodes and representing game dynamics through graph structures within the GTM framework, we enhance both the accuracy of predictions and the interpretability of the model's decision-making process. This approach not only provides precise predictions but also offers valuable insights into strategic gameplay.

The ultimate goal is to develop logic-based machine learning while addressing issues with scalability, resilience, and real-time inference. The purpose of this study is to show how GTMs can be used to provide understandable, practical, and efficient solutions for challenging board game situations.

Further we will describe the model architecture, feature engineering, and data processing used in the technical implementation of the Graph Tsetlin Machine for Hex game outcome prediction. The workflow and procedure utilized to get the results which includes important processes ranging from data gathering and simulation to training and evaluation. The outcomes of our tests, along with performance indicators and information on how interpretable the forecasts are.

The source code and additional resources for this project are available on GitHub [1] at: `https://github.com/yngvemag/uia-ikt457-hex-game`.

2 Implementation Strategy

This chapter offers a thorough overview of the project's approach, emphasizing the methodical actions used to meet the goals of the investigation. In order to ensure a fair and accurate depiction of gameplay situations, the chapter starts by outlining the processing and structuring of the Hex game data into training and testing datasets. The transformation of Hex boards into graph-based representations, enhanced with symbolic and feature-based features to capture the intricacies of the games, is then thoroughly explained.

The training step is then described, including how the Graph Tsetlin Machine (GTM) was set up and used to acquire strategies and logical patterns from the provided graph representations. Additionally covered is the iterative method for feature optimization and parameter adjustment. Lastly, the evaluation step is examined, during which the model's prediction skills were validated by evaluating its performance using test data that was not visible. The inclusion of visual tools to analyze findings and guide future improvements is highlighted in the chapter's conclusion. This methodical approach guarantees that the system is reliable, comprehensible, and efficient in forecasting the results of Hex games.

2.1 Hex game dataset

The dataset used in this research comes from the Kaggle dataset repository [2] "Game of Hex", which contains one million Hex games created by random play. This dataset is a great tool for investigating and testing logic-based AI models, like Graph Tsetlin Machines, because it was created especially for training machine learning models to predict the outcome of Hex games.

With columns representing individual cells on a Hex board, each game in the dataset is represented as a single row. For instance, a particular cell on the board is represented by the column $cell1_0$. These columns' values are encoded as follows:

- 1: Shows that the first player played the cell.
- -1: Denotes that the cell was played by second player.
- 0: Indicates the cell remains unplayed.

The dataset also includes a winner column (the last column), which explicitly identifies the winner of each game. As Hex cannot result in a draw, this column contains values of 1 or -1, corresponding to the first and second players, respectively.

The board itself is structured as a nxn graph, with each cell uniquely identified by its row and column indices (e.g., $cell0_0$, $cell0_1$, ..., $cell6_6$). The data is well-suited for machine learning models that require structured input, allowing each cell's state to contribute to the overall prediction.

Example data:

'-1,1,0,0,-1,-1,1,0,1,1,-1,-1,1,1,0,1,1,-1,1,1,0,1,1,-1,1,0,-1,0,-1,0,1,-1,1,0,-1,-1,1,0,-1,1,-1,-1,1,-1,0,1,-1,0,-1,1'

Player 2 will win the following board as a result.

- Player1: blue (-1)
- Player2: orange (1)
- Empty: gray (0)

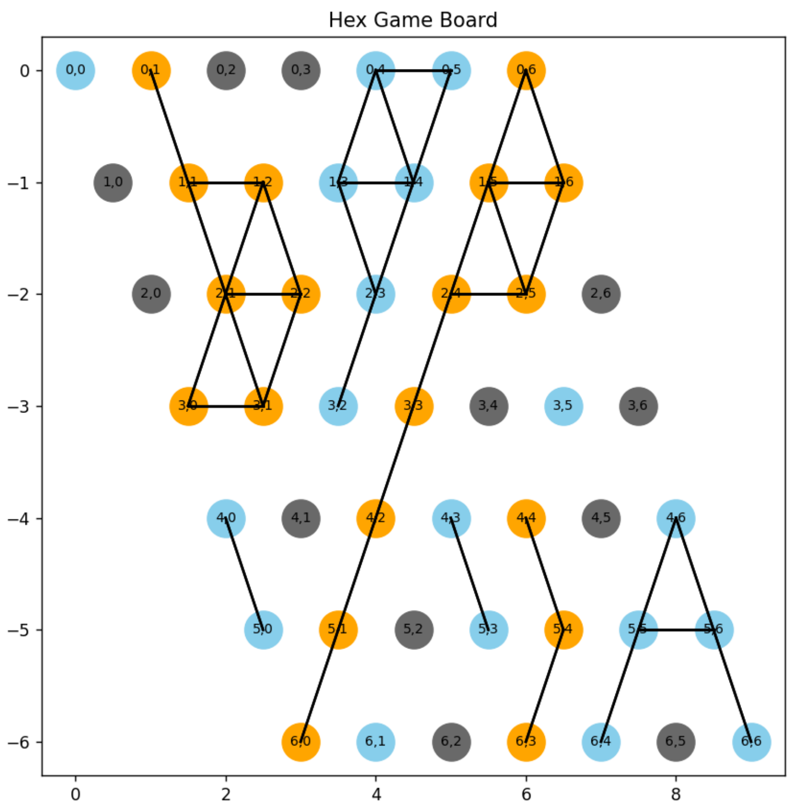

Figure 2.1: Example hex-game

2.2 Initialization, Training, and Evaluation

This project leverages Graph Tsetlin Machines (GTMs) to predict the outcomes of Hex board games. The workflow is designed to ensure a systematic approach, combining initialization, graph construction, training, and evaluation. Each phase builds on the previous steps to create a robust model capable of identifying winning patterns in the Hex gameplay.

2.2.1 Main Workflow Overview

The project begins with the central script, main.py, which orchestrates all the phases in the project.

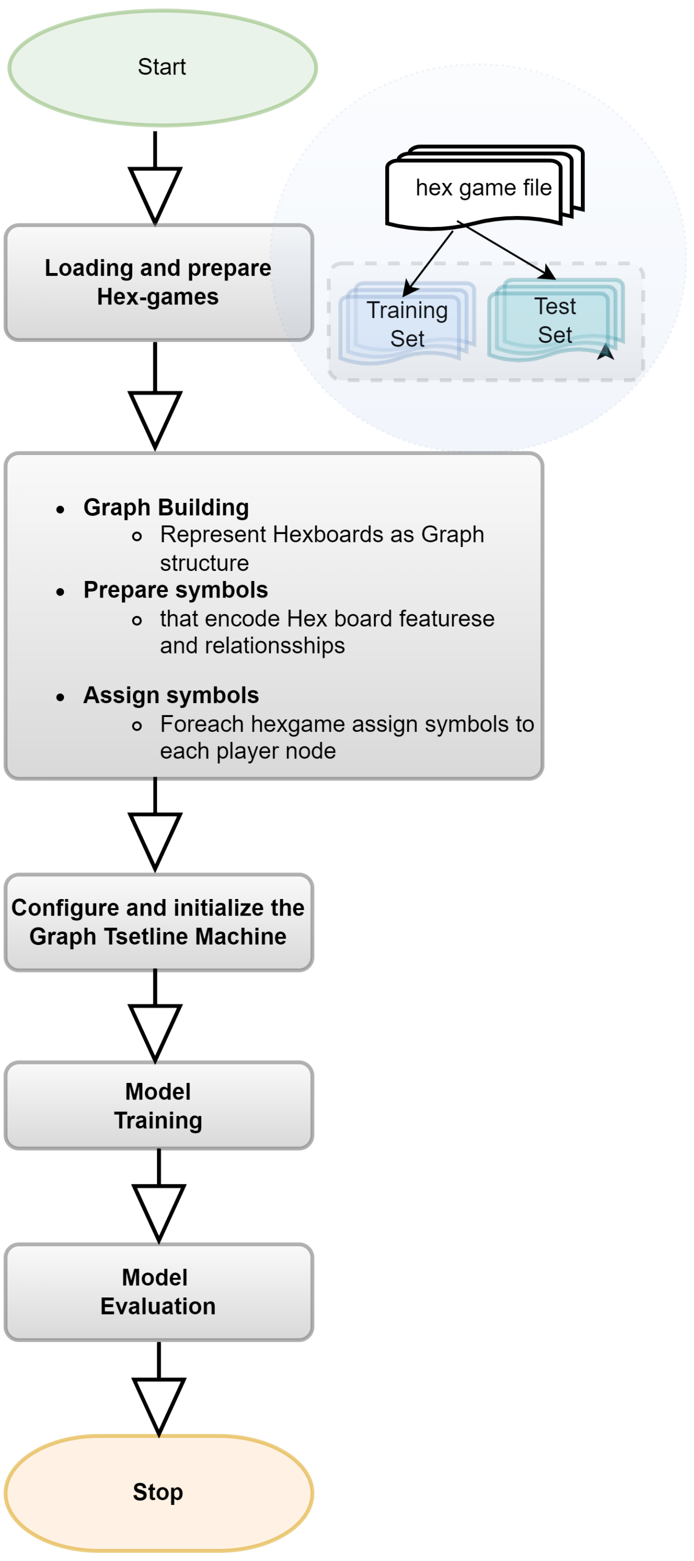
Start
hex game file
Training Set
Test Set
Loading and prepare Hex-games
Graph Building
Represent Hexboards as Graph structure
Prepare symbols
that encode Hex board featurese and relationsships
Assign symbols
Foreach hexgame assign symbols to each player node
Configure and initialize the Graph Tsetline Machine
Model Training
Model Evaluation
Stop

Figure 2.2: Main workflow

Loading and prepare hex-games

The Hex game data for our project is loaded from structured CSV files, where each row denotes a full game, complete with the winning player and the condition of each cell on the board. The data is prepared for usage by being divided into training and testing datasets, making sure that each set offers a fair representation of the game's circumstances. This enables us to successfully train the model and validate it using data that hasn't been seen yet.

Once the data are loaded, we analyze the layout of the board to extract key details about the state of each cell: whether it is occupied by Player 1 (1), Player 2 (-) or left empty (0). We also compute adjacency relationships to reflect the connections between cells in the Hex board, capturing horizontal, vertical, and diagonal links. These relationships are critical for constructing the graph-based representations used by the Graph Tsetlin Machine.

Feature vectors and symbolic representations are then generated for each board, encapsulating relevant gameplay information. These features are carefully designed to capture the patterns and nuances of Hex gameplay, enabling the GTM to process the data efficiently. This workflow ensures that the data is transformed into a structured format, ready for graph construction and model training.

Setting Up Graphs

Hex boards are methodically converted into graph-based representations in our project in order to efficiently model the games. Each Hex board is depicted as a graph, with nodes standing in for distinct board cells. Adjacency relationships, such as horizontal, vertical, and diagonal connections, are represented by the edges connecting the nodes. We accomplish this by building and initializing these graph structures using $init_graphs.py$.

Every node is enhanced with both static symbols and dynamic features throughout the graph creation process. Information such as the player occupying the cell (1 for Player 1, -1 for Player 2, or 0 for unoccupied) and directional associations with nearby nodes are examples of static symbols, which are described in symbols.py. Additional context is provided through dynamic features, which are produced in $hexboard_features.py$ and include adjacency advantages and steps taken to win. In order to prepare the data for the GTM and capture the subtleties of Hex gaming, these qualities are essential. The resultant graph objects are prepared for training since they have all node and edge characteristics set up.

Training the Model

For training the model, we utilize the Graph Tsetlin Machine (GTM), implemented in tm.py. The training process begins by configuring the GTM with hyperparameters, such as the number of clauses, threshold (T), specificity parameter (s), and depth. These settings are optimized based on the complexity of the Hex game and the nature of the graph-based features.

The GTM receives the training data, which consists of graph representations of Hex boards. The model learns logical patterns and gameplay strategies by processing the nodes' and edges' symbolic and feature-based attributes. Even for big datasets, the training process is made efficient by utilizing GPU acceleration through kernels.py. We track performance metrics like accuracy, precision, and F1-score during training in order to assess the model's development and modify the training settings as necessary.

Evaluating Performance

To evaluate the performance of the trained GTM, we apply the model to unseen test graphs that were prepared in a manner consistent with the training graphs. Using main.py, the model predicts the outcomes of these test games based on the graph representations.

The evaluation metrics, such as accuracy, precision, recall, and F1-score, are then calculated by comparing the predictions with the actual results. These measurements shed light on the model's generalizability and overall efficacy in spotting trends and tactics. We also examine the data using visual tools, which help us pinpoint areas that need optimization or enhancement. This assessment stage guarantees that the GTM is reliable and able to forecast Hex game results with accuracy.

3 Implementation

This chapter provides a detailed explanation of the technical implementation of the project, including the structure of the system, the model training pipeline, and the evaluation methodology. Each section covers key components such as the project structure, data processing, graph construction, symbol preparation, training, and evaluation.

3.1 Project Structure and Implementation

The project is organized to facilitate ease of development, testing, and extensibility. It is structured into modular files, each responsible for specific functionalities, ensuring clear separation of concerns.

3.1.1 Project Files

The project comprises several key files, each playing a distinct role in the implementation:

- **main.py:** The central entry point for the project, coordinating all major processes. This file handles data loading, initialization of graphs, configuration of hyperparameters, training of the Graph Tsetlin Machine (GTM), and evaluation of model performance. It also facilitates interaction with the visual tools to analyze gameplay results and debug the pipeline.
- **init_graphs.py:** Responsible for creating and initializing graph-based representations of Hex boards. This file converts board data into graph structures, with nodes representing board cells and edges encoding adjacency relationships (horizontal, vertical, and diagonal). It ensures that all nodes and edges are getting relevant symbolic and feature-based properties for GTM processing.
- **hexboard.py:** Defines the structure of the Hex board, offering utility methods to parse, process, and validate board data. This file is integral for reading game data from CSV files and preparing datasets for training and testing. It generates adjacency matrices and provides methods for extracting graph-relevant attributes such as labels, symbols, and node properties. Additionally, it interacts with 'hexboard_features.py' to encode gameplay states.
- **hexboard_features.py:** Handles the extraction and assignment of features for nodes in the graph representation. This includes both symbolic features, such as player-specific symbols and directional relationships, and vector features, which provide dynamic gameplay context like adjacency advantages or steps toward victory. These features are critical for capturing the complexities of Hex gameplay and are computed based on the adjacency matrix and node data.

- **symbols.py:** Provides predefined static symbols to encode player-specific and directional information. Symbols in this file represent fixed relationships, such as player moves (‘1‘ for Player 1, ‘-1‘ for Player 2, and ‘0‘ for unoccupied) and directional connections (e.g., North, East, Northwest). These symbols are used in conjunction with dynamic features from ‘hexboard_features.py‘ to build meaningful graph representations.

3.2 Initialization

The initialization phase prepares the data and graph structures that serve as inputs for model training.

3.2.1 Data Loading

In this project, Hex game data is loaded from structured CSV files. Each row represents a complete game, detailing the state of every cell on the board and the game winner. The data is split into training and testing sets, ensuring balanced representation of gameplay scenarios. The **‘hexboard.py‘** file parses the data, while **‘hexboard_features.py‘** generates feature vectors encapsulating gameplay details such as player positions, adjacency relationships, and game progress.

The **‘Hexboard‘** class, implemented in **’hexboard.py’** file, defines the structure and functionality of the Hex board. It plays a critical role in the data preparation pipeline by parsing, validating, and processing Hex game data. Key responsibilities include:

- **Data Parsing**: Reads Hex game data from structured CSV files, where each row represents a complete game. This includes the state of every cell on the board and the identity of the game winner.
- **Validation**: Validates the input data to ensure it adheres to the expected format and contains no inconsistencies or missing values.
- **Board Size Calculation**: Dynamically calculates the size of the board from the input data, enabling the system to support different board dimensions seamlessly
- **Dataset Preparation**: Splits the data into training and testing sets, ensuring balanced representation of gameplay scenarios for effective model training and evaluation

The **‘HexBoard‘** class integrates with **‘hexboard_features.py‘** to compute feature vectors for each board. These vectors encapsulate gameplay details, such as the positions of players, adjacency relationships, and progress toward victory. This modular design enables flexibility in handling various board sizes and provides a robust foundation for generating graph-based representations.

3.2.2 Graph Building

Hex boards are transformed into graph-based representations using the **'init_graphs.py'** file. Each cell on the Hex board is represented as a node in the graph, and connections (edges) are created based on adjacency relationships such as horizontal, vertical, and diagonal links. Node properties, including features and symbols, are assigned during this phase to enrich the graph structure with meaningful gameplay information.

- **Node Creation:** Each Hex board is represented as a graph, where each cell on the board corresponds to a graph node. These nodes are assigned properties such as player occupation (e.g., Player 1, Player 2, or unoccupied), symbolic relationships (e.g., directional neighbor symbols), and dynamic features like completed steps or adjacency advantages.
- **Edge Building:** Connections, or edges, are established between nodes based on the Hex board's layout, capturing horizontal, vertical, and diagonal relationships. Each edge is associated with a specific type, such as directional connections, to provide additional context about gameplay interactions.
- Symbol **and Feature Assignment:** Static symbols, defined in symbols.py, are assigned to nodes to represent fixed attributes like player-specific information and directional relationships. Dynamic features, extracted using **'hexboard_features.py'**, are also added to nodes, enriching the graph's representational power with context-dependent details like adjacency advantages.
- **Graph Object Preparation:** The fully constructed graph is then prepared using the Graphs class, which manages the configuration of node and edge properties to ensure compatibility with the Graph Tsetlin Machine (GTM).
- **Integration with Training and Testing:** Training graphs are initialized with these feature-based and symbolic attributes when they are prepared. Then these graphs are sent to the GTM for testing and learning, which helps the model see trends and accurately forecast the game results. This methodical technique guarantees that the graph representation accurately depicts the intricacies of Hex gameplay while facilitating the model's quick processing.

3.2.3 Symbols Preparation

Static symbols, defined in **'symbols.py'**, are used to represent player-specific and directional relationships on the Hex board. Dynamic features, such as adjacency advantages and steps toward victory, are extracted in hexboard_features.py. These symbols and features are crucial for capturing the intricacies of Hex gameplay and enabling the Graph Tsetlin Machine to learn logical patterns effectively.

Player Symbols

Player symbols encode the ownership or state of each cell on the Hex board. The values are:

```
PLAYER_SYMBOLS = {
    1: "X",
    -1: "O",
    0: "_"
}
```

These symbols allow the model to differentiate between players and analyze game strategies based on cell ownership.

Directional Symbols

Directional symbols represent the relationships between neighboring cells based on their relative positions. For instance:

```
DIRECTIONS = {
    (-1, 0): ["NW"],     # Top Left
    (-1, 1): ["NE"],     # Top Right
    (0, -1): ["W"],      # Left
    (0, 1): ["E"],       # Right
    (1, -1): ["SW"],     # Bottom Left
    (1, 0): ["SE"]       # Bottom Right
}
```

These symbols indicate the direction of the connection between two nodes. These symbols help the model understand the relationships and connectivity patterns within the hexgame.

Edge Symbols

Edge symbols encode the relationships between nodes and their edges, highlighting gameplay-related connections. Examples include:

```
EDGE_SYMBOLS = {
    1: "X-EDGE-X",
    -1: "O-EDGE-O"
}
```

If a node in the graph has this symbol attached, it means that this node is connected to other nodes in the graph and the connection between nodes are fully connected from one side to the other side.

Blocked Symbols

Blocked symbols are used to mark cells or paths that are unavailable for gameplay.

```
BLOCKED_SYMBOL = ['B']
```

This represent a blocked node or area that cannot be occupied or traversed. This is crucial for modeling scenarios where certain board regions are restricted.

Neighbor Symbol

Combines player symbols and direction symbols to represent relationships.

```
# Combines player symbols and direction symbols to represent relationships.
# Neighors for player 'X': XNW, XNE, ,XW, XE, XSW, XSE
# Neighors for player 'O': ONW, ONE, ,OW, OE, OSW, OSE
NEIGHBOR_SYMBOLS = [f'{p}{n}' for p in PLAYER_SYMBOLS.values() for n in DIRECTION_SYMBOLS]
```

For instance:

- XOE: Represents Player1 occupying a neighboring node to the east of a given node.
- XNW: Represents Player1 occupying a neighboring node to the north west of a given node.

Edge to edge advantage

This symbol represents the positional advantage gained by connecting specific edges of the Hex board. This symbol only occure when a player nodes are fully connected.

```
ADVANTAGE_COUNT = 5
# Tracks strategic "edge-to-edge" connections.
EDGE_TO_EDGE_ADVANTAGE = [f'E{i+1}E' for i in range(ADVANTAGE_COUNT)]
```

Steps Completed Symbol

The steps completed symbol tracks progress toward completing a winning path. For example:

A numeric value indicating how many steps a player has completed toward creating a continuous path from one side of the board to the other. This symbol is dynamic and updates based on gameplay, providing critical feedback for the model to assess game states effectively.

3.2.4 Feature vector Symbols

The feature vector symbols are dynamically generated attributes assigned to each node on the Hex board graph. These symbols capture context-specific gameplay details that go beyond static relationships, enabling a more granular and flexible representation of the board's state. The process of extracting and assigning feature vector symbols is as follows.

The **'get_feature_vector(node_id)'** function in the 'Hexboard' class retrieves a sparse feature vector associated with a specific node (cell) on the Hex board. This vector represents various gameplay attributes relevant to the node, such as adjacency relationships, positional advantages, or progress toward a winning path. Before processing, the feature vector is checked for non-zero entries (node_features.nnz > 0). This ensures that only nodes with meaningful feature data are considered for further processing, optimizing the graph representation by ignoring redundant or inactive nodes.

For nodes with non-zero feature vectors, the indices of these features are retrieved using node_features.indices. Each index corresponds to a specific gameplay attribute or condition that the node satisfies. For example, an index might represent an adjacency advantage or an intermediate step toward a victory condition.

For each non-zero feature index, a feature symbol in the format Feature:$< index >$ is dynamically generated and assigned to the node. This is achieved through the **graphs.add_graph_node_property(graph_id, node_id,** $f''Feature : feature''$**)** method. These symbols effectively encode the gameplay state of the node and enrich the graph representation.

From init_graph.py:

```
if HexboardFeatures.FEATURE_COUNT > 0:
            node_features = hexboard.get_feature_vector(node_id)
            if node_features.nnz > 0:  # Check if the node has any non-zero features
                feature_indices = node_features.indices  # Get the non-zero feature indices
                for feature in feature_indices:
                    graphs.add_graph_node_property(graph_id, node_id, f"Feature:{feature}")
```

Player Steps Completed

This feature evaluates the progress a player has made toward their winning condition by measuring the number of steps completed along the path to their goal. For Player 1, who aims to connect vertically, this metric considers the number of unique rows spanned by their connected nodes. For Player 2, who aims to connect horizontally, it counts the unique columns spanned. This feature quantifies the proximity of a player to their win condition.

Distance to goal

This feature calculates the shortest distance from a player's node to the goal line. For Player 1, it measures the vertical distance to the top or bottom of the board, while for Player 2, it measures the horizontal distance to the left or right edge. The smaller the distance, the closer the node is to fulfilling the player's objective.

Cluster size

This feature identifies all nodes connected to a given node (forming a cluster) and returns the size of the cluster. A larger cluster implies a stronger network of nodes, providing a better base for expansion or defense.

Node Degree

The degree of a node represents the number of direct neighbors it has in the internal graph. This feature indicates how well-connected a specific node is within its cluster, which can reflect its strategic importance in terms of connectivity.

Average Degree in Cluster

This feature computes the average degree of all nodes in the cluster containing the given node. It captures the overall connectivity within the cluster, highlighting the density and robustness of the connections.

Edge Connectivity

This feature counts the number of a player's nodes that are directly connected to the edges of the board. Edge connectivity is a strategic advantage, as control of edges often plays a critical role in achieving the win condition.

Path Robustness

Path robustness measures the number of distinct connected paths that span a player's start and end edges. It reflects the redundancy and resilience of a player's strategy, as more paths reduce the risk of being blocked by the opponent.

Threat Proximity

This feature identifies potential threats by evaluating if placing a piece at a given empty cell would immediately connect two or more nodes for a player. High threat proximity indicates positions where the player can make significant progress with minimal moves.

Critical Gaps

Critical gaps measure the number of empty cells that, if occupied, would bridge two separate clusters of a player's nodes. These gaps are strategically valuable, as they can be used to create large connected clusters or block an opponent's progress.

Center Control

Center control evaluates the number of a player's nodes located in the central region of the board. The center is often considered a key strategic area, offering better connectivity and flexibility for building or disrupting paths.

må flyttes til disksjon etterhvert This features can be combined or used invidiously, but we still achieve 100%, but how fast depend on the features

3.3 Model Training

The training phase leverages the 'tm.py' file to implement the Graph Tsetlin Machine (GTM). Training is performed on the prepared graph representations of Hex boards, with the model learning strategies and patterns from the data. GPU acceleration, facilitated by 'kernels.py', ensures efficient training, even with large datasets. Hyperparameters such as the number of clauses, threshold (`T`), specificity parameter (`s`), and depth are configured to optimize the model for Hex gameplay.

3.4 Model Evaluation

The evaluation phase validates the trained model using unseen test graphs. Predictions are compared against ground truth outcomes, and metrics such as accuracy, precision, recall, and F1-score are computed to assess performance. The evaluation process also involves visual tools for analyzing model predictions and identifying areas for improvement. This ensures that the Graph Tsetlin Machine is robust and capable of effectively predicting Hex game outcomes.

4 Results & Discussion

This chapter presents the results of our experiments and examines them in light of the goals of the research. The results are discussed with a focus on evaluating the performance of the Graph Tsetlin Machine (GTM) in predicting Hex game outcomes. We emphasize important measures to measure the model's accuracy. Additionally, an analysis is conducted on how different symbols, feature combinations, and hyperparameter settings affect the model's performance.

4.1 Results

4.1.1 Test and Training Setup

The Hex game data was split into training and testing datasets, ensuring a balanced representation of various game scenarios. The training dataset consisted of 80% of the available data, while the remaining 20% was reserved for testing. This split ensured that the model was trained on a diverse set of board configurations and validated against unseen data to evaluate its generalization capabilities.

To capture the complexities of Hex gameplay, various symbols and features were integrated into the graph representation. Symbols, such as player-specific markers and directional relationships, were paired with vector features. These combinations were tested to identify configurations that gave the best accuracy for the model.

4.1.2 Test Results

Test result on 7x7 board

In our final stage of coding, we performed the following 7x7 board size test with 100,000 games. 80,000 games were used for training, and 20,000 were reserved for testing on unseen data. Following hyperparameters were used during testing:

- Number of clauses: 10,000
- Threshold (T): 5,000
- Specificity parameter (s): 2.0
- Depth: 2

- Hypervector size: 128
- Hypervector bits: 2
- Message size: 512

During training, the model showed consistent improvement over the epochs, with accuracy starting at 95.31% in the first epoch and reaching 99.00% by the final epoch. A detailed breakdown of accuracy over selected epochs is provided in Table 4.1.

Table 4.1: Training Accuracy Over Selected Epochs

Epoch	**Accuracy (%)**
0	95.32
10	98.13
20	98.84
50	97.69
70	98.30
99	99.00

Upon completion of training, the model achieved a test accuracy of approximately 99%.

Test result on 9x9 board

To further evaluate the performance of the Graph Tsetlin Machine (GTM), we also tested the model using a 9x9 Hex board configuration. The tests included scenarios where the winning move was set to occur either 2 moves away or 5 moves away from the current state. The same hyperparameter settings were used for these experiments to ensure consistency and comparability with the larger board size results.

The hyperparameter configuration for these tests was:

- Number of clauses: 10,000
- Threshold (T): 5,000
- Specificity parameter (s): 2.0
- Depth: 2
- Hypervector size: 128
- Hypervector bits: 2
- Message size: 512

Results 2 Moves Away

The model train and tested on 5000 games converged quickly in the 2-move scenario, achieving 100% training accuracy by epoch 18. The test accuracy for this configuration was measured at 98.64%.

Table 4.2: Training Accuracy for 9x9 Board (2 Moves Away)

Epoch	**Accuracy (%)**
0	97.07
5	99.66
10	99.98
15	99.98
18	100.00
20	100.00

Results 5 Moves Away

For the 5-move scenario, the model train and tested on 5000 games demonstrated a slightly slower convergence compared to the 2-move scenario. However, it still achieved 100% training accuracy by epoch 45. The test accuracy for this configuration reached 99.84%.

Table 4.3: Training Accuracy for 9x9 Board (5 Moves Away)

Epoch	**Accuracy (%)**
0	86.96
5	97.84
10	99.29
20	99.84
30	99.95
45	100.00

From these result we saw that the 2-move scenario showed a faster convergence to 100% than the 5-move game sets. With the 5-move sets it gradually converged and required more epochs to reach same accuarcy as the 2-move.

4.2 Discussion

Our experiments show that the Graph Tsetlin Machine (GTM) is capable of predicting outcomes in hex games. By implementing symbolic and feature-based graph representation of the board, the GTM is able to identify key patterns to predict a win for both the 7x7 and 9x9 board sizes. This indicates that the model has a high ability to generalize patterns from training to make predictions on unseen data. We also observed very fast convergence during training, with significant improvements in accuracy within the first epochs. For larger and more complex datasets, this efficiency is important.

However, when we configured the GTM with large number of clauses we that he model failed to predict with high accuracy. We suspect that the model overfitted spesific training pattern that led to this issue.

We also experience different effect with symbols. For instance did we test with different directional symbols. Initially we tested wih directional symbols such as "UP", "DOWN", "LEFT" and right. These symbols represented basic directions, with combinations like "UP + RIGHT" for diagonal relationships. However, this approach showed limited effectiveness, as the Hex board's unique hexagonal structure was not fully captured by these simplified directional representations. We changed our directional symbols to "NW", "NE", "W", "E", "SW", and "SE". These symbols are more detailed and we there is no need for combination of symbols to describe direction. This change improved the accuracy. This demonstrate that choice of feature and symbols have a huge impact how the GTM precict outcomes.

Feature vectors are essential to our method because they enhance the graph representation of the Hex board by offering dynamic, game-specific features. Adjacency advantages, completed steps toward victory, and positional relevance are examples of changing features of gameplay that feature vectors represent, in contrast to static symbols that encode fixed properties like player locations or directional relationships. Because of these characteristics, the model is able to comprehend deeper strategies that are inherent in Hex games and go beyond surface-level patterns. The process of designing and selecting feature vectors has been one of the most challenging aspects of this project. Identifying the features that contribute to the model's predictive performance. Certain features, while theoretically relevant, added little value to the model and the model did not perform better. It is also hard to find which features that will perform well together, so finding the right combination of features, hyperparamters required a lot of testing.

5 Conclusions

Hex is a complex game that requires strategic understanding of connectivity patterns. The GTM's clause-based logic aligns well with these demands, providing interpretable reasoning for its predictions. The use of symbols and dynamic features, such as adjacency advantages and steps toward victory, enabled the model to mimic key human strategies, such as blocking opponent paths and creating winning connections.

One important conclusion from our research is that in order to get best performance, it is crucial to carefully configure the hyperparameters, symbols, and feature-vector symbols. Every element is important in forming the model's comprehension of the layout of the board and gameplay patterns:

- Hypervectors impact the model's capacity to generalize across many contexts and specify the underlying representation of game states.
- Symbols improve the model's strategic reasoning by offering interpretable and context-specific information, such as player positions and directional linkages.reasoning.
- Feature-vector symbols add dynamic properties to the graph, capturing subtleties such as adjacency advantages and victory-oriented tasks performed.

To sum up, our existing setup offers a strong foundation for analyzing and forecasting Hex gameplay results utilizing the GTM. The model's efficiency and interpretability make it a promising tool for broader graph-based decision-making tasks as well as game analysis. Additional improvements including broadening the feature collection, improving the choice of hyperparameters, and extending the method to bigger boards and more intricate datasets might be investigated in future research.

References

[1] Y. Magnussen and C. E. Prag, *Hex game project repository*, `https://github.com/yngvemag/uia-ikt457-hex-game`, Accessed: December 19, 2024, 2024.

[2] C. Hollingsworth, *Game of hex dataset*, Available at `https://www.kaggle.com/datasets/cholling/game-of-hex`, Accessed on 12/04/2024, 2023.