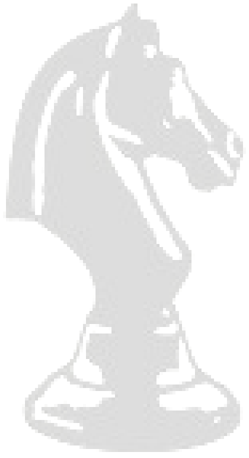
Azrieli College of Engineering Jerusalem
עזריאלי מכללה אקדמית להנדסה ירושלים

המחלקה להנדסת תוכנה

דוח אלפא פרויקט גמר – ה'תשפ"ה

רשתות נוירונים ניתנות לפירוש עבור בעיית מסילה המילטונית

Interpretable Neural Networks for Tackling Hamiltonian Path Problems



מאת
אנדרי טמס
ת.ז. 322341173
טוני בגאלי
ת.ז. 322342205

מנחה אקדמי: ד"ר לוריא צור.

28/01/2025

חתימת מנחה: [illegible]

מערכות ניהול הפרויקט:

#	מערכת	מיקום
1	מאגר קוד	Github link
2	יומן	Issues link
3	קישור לסרטון דוח אלפא	Video link

מידע נוסף:

סוג הפרויקט	מחקרי ממרצה במכללה
אם זה פרויקט ממשיך?	זה פרויקט חדש

Table of Contents

Key Terms

1. **Knight’s Tour**
The classical puzzle of moving a knight across a chessboard so that it visits every square exactly once. This can be viewed as a special case of the Hamiltonian Path Problem in graph theory.

2. **Hamiltonian Path Problem (HPP)**
A fundamental graph problem of finding a path that visits every vertex exactly once. The Knight’s Tour can be modeled as finding a Hamiltonian path (or cycle) on the board’s graph.

3. **Monte Carlo Tree Search (MCTS)**
A search algorithm that builds a game tree incrementally through random simulations (“rollouts”) and uses statistical outcomes to determine the most promising moves.

4. **Warnsdorff’s Algorithm**
A heuristic for the Knight’s Tour that always moves to the square with the fewest onward moves (lowest degree), aiming to minimize early dead ends.

5. **Brute Force / Backtracking**
A systematic search approach that tries every possible path. If a path leads to a dead end, the algorithm backtracks to explore other branches.

6. **Reinforcement Learning (RL)**
A machine-learning paradigm where an agent learns optimal actions through rewards and penalties while interacting with an environment.

7. **Q-Learning**
A popular RL method that maintains a Q-table (state-action values). The agent updates these values based on rewards received, converging toward an optimal policy over time.

Elevator speech

In this project, we aim to develop **interpretable neural networks** that tackle **Hamiltonian path problems:** A class of complex combinatorial puzzles that include the Knight's Tour and the Traveling Salesman Problem (TSP). These problems are **NP-complete** and grow exponentially in complexity.

Our goal in this project is to determine whether general-purpose reinforcement learning algorithms can discover new heuristics and algorithms that don't involve neural networks at all. Focusing on interpretability means we want to shine a light on the network's decision-making process, extracting insights into the "why, and how" behind each move rather than just offering unexplained solutions. Our motivating example is the discovery of the min-conflicts heuristic, which is a powerful and widely used heuristic to solve constraint satisfaction problems that was discovered by a neural network algorithm.

Introduction

Combinatorial optimization problems play a pivotal role in numerous real-world domains, from routing in logistics to path planning in robotics. One particularly fascinating class of these problems revolves around **Hamiltonian paths** routes that must visit every node (or square) exactly once. Among the most iconic examples is the **Knight's Tour**, where a knight in chess traverses an n-by-n board, trying to visit each square a single time. Another well-known instance is the **Traveling Salesman Problem (TSP)**, in which a traveler must visit each city once and minimize the total travel distance.

Over the years, many methods have emerged to tackle Hamiltonian path problems: classical heuristics, brute-force backtracking, and various AI-driven algorithms. Yet, as problem sizes grow, these approaches often become computationally expensive. Furthermore, sophisticated algorithms particularly machine learning methods tend to function as "black boxes," offering limited insight into **why** a given move or route is chosen.

This project addresses the gap between **efficient pathfinding** and **interpretability**. Existing solutions for problems like the Knight's Tour either rely on well-known heuristics (e.g., Warnsdorff's rule) or exploit more advanced AI techniques (e.g., Monte Carlo Tree Search, Q-Learning), but rarely provide a clear explanation of their reasoning. Such transparency is crucial in high-stakes environments where human operators need to trust and verify the solutions where it might be in autonomous vehicle navigation, warehouse robotics, or complex logistics planning.

Considering these challenges, our project seeks to develop **interpretable neural networks** and hybrid algorithms that can not only **find** valuable Hamiltonian paths quickly but also **explain** how they do it. By harnessing both classical heuristics and modern AI, we aim to balance **performance** with **explainability**, offering a novel approach that illuminates the decision-making process behind complex pathfinding tasks.

Problem description

The Hamiltonian Path Problem (HPP) involves finding a path that visits each vertex (or cell) exactly once, making it especially challenging in large or obstacle-laden environments. In this project, we extend the traditional HPP to scenarios like the Knight's Tour with blocked squares and the Traveling Salesman Problem (TSP) with various constraints.

Problem Requirements and Characterization

This project requires handling **flexible board configurations**, including adjustable dimensions and obstacle placements, with the goal of either finding a near-complete tour or determining that no valid path exists. Beyond merely generating a path, the methods employed should elucidate **why** each move is selected, shedding light on whether a given step reduces potential dead ends or applies a specific heuristic.

- **Flexible Board Configurations**
  The system should support various board sizes and customizable obstacle placements, enabling diverse test scenarios for the algorithms under investigation.
- **Decision Explanation**
  Each move is accompanied by a clear rationale, which highlights how the underlying heuristic or learning-based approach aims to minimize dead ends.
- **Performance vs. Transparency**
  While computational efficiency is relevant, **interpretability** of the pathfinding process is central to understanding and refining new heuristics or algorithms.

The Problem from a Software Engineering Perspective

Implementing an interpretable Hamiltonian path detector involves combining algorithms like MCTS and Q-Learning to handle exponential complexity, along with classical heuristics such as Warnsdorff's rule. Achieving this requires a **modular architecture** for integrating diverse methods, with **careful balance** between speed and interpretability, and efficient **obstacle handling**. Providing **insights** into each decision made with carefully optimization for exploring new heuristics or algorithms.

- **Exponential Complexity**: MCTS and Q-Learning must scale effectively.
- **Balancing Speed** & **Clarity**: Rapid solutions often lack transparency, necessitating additional explanatory mechanisms.

Similar Works & References

Several research efforts have focused on integrating neural networks, heuristic algorithms, or both to solve combinatorial optimization problems such as the TSP and the Knight's Tour. The table below highlights a few key publications, summarizing each work and its relevance to our project's emphasis on interpretability, AI-driven heuristics, and efficiency.

Article	Summary	Relevance to the Project
Learning Improvement Heuristics for Solving Routing Problems	This paper explores how neural networks can be used to solve the Traveling Salesman Problem (TSP), including methods for training and optimizing NN models for combinatorial optimization tasks.	The paper provides insights into using neural networks for combinatorial problems like TSP, which is directly related to our goal of solving Hamiltonian path-type problems using neural networks.
NASA Technical Paper on Min-Conflicts Algorithm	This document discusses the min-conflicts algorithm for constraint satisfaction problems, highlighting heuristic-based approaches to problem-solving.	The min-conflicts algorithm's heuristic nature is relevant for our project's focus on discovering and extracting efficient heuristics for problems like Knight's Tour and TSP.
A Survey on Neural Network Interpretability	This paper reviews deep learning approaches applied to NP-hard problems, offering techniques for improving the efficiency of solving such problems.	The methods explored in this paper could guide the neural network architecture and training strategies to efficiently solve Hamiltonian path problems, aligning with our goal to optimize and interpret neural network solutions.
Acquisition of Chess Knowledge in AlphaZero	The paper focuses on making AI models, especially deep learning, interpretable in the context of optimization, explaining the decision-making process.	This paper aligns with our project's emphasis on interpretability, offering methods and frameworks to explain neural network decisions in solving complex optimization problems.

Solution Description

Our system tackles Hamiltonian-type problems, such as the Knight's Tour with obstacles and the Traveling Salesman Problem (TSP), by generating a solution path and providing explanations for each move. Users input the board size, obstacle configuration, and desired mode (e.g., Warnsdorff's Algorithm, MCTS, Q-Learning, ...), and the system outputs a complete/near-complete path or indicates if no solution is possible.

System Design

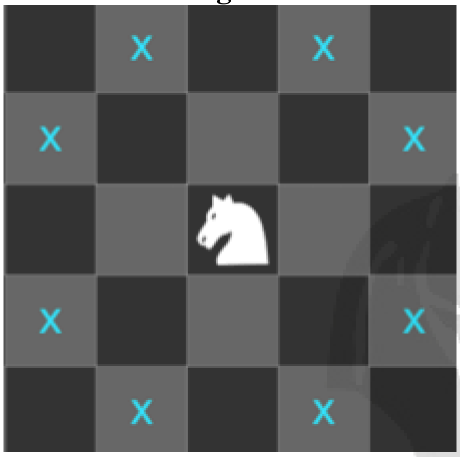
- **KnightTourBoardSetup**: Handles board initialization, obstacle placement, and move validation.
- **Algorithms:**
    - **Warnsdorff's Algorithm:** A heuristic that selects moves with the fewest onward options.
    - **MCTS (Monte Carlo Tree Search):** Simulates future paths and picks the move with the highest potential.
    - **Q-Learning:** Uses reinforcement learning to develop an optimal policy for move selection.
    - **Neural Networks (NN):** Under development to enhance pathfinding and interpretability on larger boards.
    - **AlphaZero:** An advanced RL framework combining deep neural networks with MCTS. It continuously refines its policy and value networks through self-play, aiming to discover superior heuristics for complex decision tasks—here, potentially applied to Knight's Tour–like challenges.

We plan to expand neural network integration for both the Knight's Tour and TSP setups, leveraging them for improved efficiency and explainability. Additionally, MCTS will be refined to better handle dynamic constraints, ensuring adaptability across various Hamiltonian-type problems.

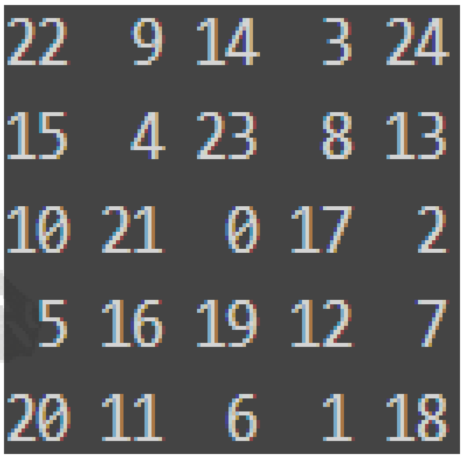
What have we done so far?

1. **Core Environment and Board Setup**
    - We developed a **KnightGame** class to generate an n-by-n board, place obstacles, and track the knight's moves with each visited square.
    - This provides a consistent environment for all our different methods.

Possible Starting Point



Possible Solution



22	9	14	3	24
15	4	23	8	13
10	21	0	17	2
5	16	19	12	7
20	11	6	1	18

2. **Multiple Approaches to the Knight's Tour**
    - **Manual Mode**: Allows a user to move the knight step-by-step, with obstacle placement including an undo feature for changing choices.
    - **Warnsdorff's Algorithm**: Implemented this classic heuristic to always choose the next square with the fewest onward moves, typically yielding efficient tours on less obstructed boards.
    - **Brute Force (Backtracking)**: Explores all possible paths systematically. This is computationally expensive but serves as a baseline to verify correctness for smaller boards.
    - **Monte Carlo Tree Search (MCTS)**: We integrated an MCTS approach that simulates moves, expands the game tree, and backpropagates results to choose promising next moves.
    - **Q-Learning**: Implemented a tabular Q-Learning agent. It explores possible moves, collects rewards, and updates a Q-table to gradually learn better strategies.
    - **(Comparing Option) MCTS vs. Warnsdorff**: We can compare their performance on the same initial board to see which approach covers more squares or discovers a better/full tour.

3. **Integration and Code Architecture**

   - We have a **main script** that presents a menu-driven interface. Each option corresponds to one of the approaches above. This has helped us validate each algorithm's behavior on the same KnightGame environment.
   - The code is structured into separate classes and files (e.g., KnightGame, MCTSPlay, MCTSNode, WarnsdorffsAlgorithm, QLearningAgent), making it more maintainable and easier to extend.

4. **Preliminary Observations**

   - **Warnsdorff's Algorithm** tends to find long tours quickly on open boards but can fail early if obstacles are poorly placed.
   - **MCTS** can handle obstacles by random exploration, though it may require enough iterations to converge on a good path.
   - **Q-Learning** can learn effective moves with sufficient episodes but needs well-tuned hyperparameters (learning rate, exploration rate, etc.).
   - **Brute Force** confirms our solutions on small boards, but it becomes computationally expensive as the board size grows.

5. **Future Directions**

   - **Adaptive AI Methods**: Explore a hybrid approach combining MCTS and Q-Learning to leverage both statistical search and learning heuristics.
   - **Algorithmic Enhancements**: Investigate enhancements like dynamic alpha/beta pruning or advanced heuristics for MCTS and Q-Learning.
   - **Neural Networks**: Integrate a neural network (NN) to guide or enhance decision-making within MCTS or Q-Learning, allowing more sophisticated pattern recognition on larger or more complex boards.

Overall, the core functionality generating boards, placing obstacles, and running multiple solution strategies have been established. We have gained valuable insights into the strengths and weaknesses of each approach, setting the stage for deeper experimentation and optimization.

Helpful Generative AI Tools

ChatGPT

- **Idea Generation & Research:** I found ChatGPT incredibly helpful when I wanted to brainstorm new angles or get clarification on tricky algorithmic concepts like MCTS or Q-Learning. It felt like having a second pair of eyes reviewing my thoughts in real time.

- **Explanations & Documentation:** ChatGPT occasionally referenced or summarized articles and projects like ours, giving me quick insights into how others approached related Hamiltonian path or reinforcement learning issues. This context helped me cross-check and refine my methods and ultimately produce more thorough documentation.

GitHub Copilot

- **Collaboration & Team Efficiency:** While collaborating with my teammate on a shared repository, Copilot's suggestions provided a consistent coding style and helped reduce repetitive boilerplate. This made code reviews smoother and pull requests more focused on logic rather than syntax details.
- **Inline Code Suggestions:** During live coding sessions, Copilot often gave me snippets for routine tasks like setting up loops or initializing data structures. My teammate and I appreciated how it kept our workflow moving without constant context switching.

Cody AI (in Visual Studio)

- **Code Assistance:** In Visual Studio, Cody AI seamlessly generated bits of boilerplate, such as class outlines or basic method stubs. That freed us to focus on the actual logic rather than repetitive code overhead.
- **Contextual Recommendations:** Cody AI often anticipated my next steps based on the existing code context, suggesting lines or refactors that helped me keep a steady flow while coding.

Appendices

Risk Table

Risk ID	Risk Description	Likelihood (L/M/H)	Impact (L/M/H)	Mitigation Strategy	Contingency/Response Plan
R-01	**MCTS Complexity and Time Consumption** Monte Carlo Tree Search may become too computationally expensive for larger boards or extensive simulations, potentially slowing progress.	Medium	High	- Start with smaller board sizes for initial testing. - Optimize or prune search if runtime grows too large.	- Switch to a more efficient approach or a reduced-iteration MCTS if performance is unacceptable.
R-02	**Q-Learning Tuning Challenges** Obtaining effective learning parameters may be difficult, causing suboptimal behavior or insufficient discovery of new heuristics.	Medium	Medium	- Experiment with different learning rates, exploration strategies, and reward structures. - Regularly evaluate performance against known heuristics (e.g., Warnsdorff).	- Revert to simpler approaches or hybrid strategies if tuning fails to improve learning outcomes significantly.
R-03	**Warnsdorff Algorithm Limitations** Relying solely on Warnsdorff's heuristic might not yield the insights or results needed, especially in heavily obstructed or specialized boards.	Medium	Medium	- Investigate multiple heuristics beyond Warnsdorff (e.g., alternative move-order strategies). - Compare different heuristics in parallel.	- Abandon or augment Warnsdorff if it proves consistently weak, adopting a newly discovered or proven heuristic.
R-04	**Team Communication Gaps** With two members working on the project, misunderstandings could arise about new ideas, potential solutions, or analysis of AI behavior.	Low	High	- Hold frequent check-ins and brainstorming sessions. - Document all algorithmic changes and testing outcomes in a shared repository.	- If misalignment is detected, schedule dedicated sync sessions to realign goals and maintain consistent progress.
R-05	**Ensuring Correct Learning & Transparency** Even if the AI (or backtracking/heuristics) performs well, we must illuminate *why* it works and *how* new algorithms emerge, aligning with our core research goal of discovering novel heuristics.	Medium	High	- Capture the system's decision-making rationale (e.g., state-action pairs, path expansions). - Integrate logging and post-run analysis tools for interpretability.	- If clarity is insufficient, develop more robust explanation frameworks or conduct deeper manual trace analyses.
R-06	**Neural Network Complexity** While the project aims to avoid NN reliance, exploring NNs may still occur. Implementing them can be difficult and time-consuming, distracting from the core question about alternative RL-based heuristics.	Low	Medium	- Limit NN experiments to a secondary branch of the project. - Clearly separate NN-focused efforts from the main goal of discovering non-NN heuristics.	- If NN complexity overruns the schedule, refocus on purely non-NN RL approaches to stay aligned with the main goal.

User Requirements Document (URD)

Functional Requirements

Req. ID	Requirement Description	Priority (L/M/H)
FR-01	The simulation should allow creating an n-by-n board and placing obstacles as specified, making sure it shows why and how each decision made by the algorithm to find its hidden heuristics.	High
FR-02	Different approaches and algorithms should be easy to implement in the simulation code to be able to build up our approaches.	Medium
FR-03	The ability to tune algorithms just like Q-learning, for debugging purposes and insuring algorithms efficiency.	High
FR-04	The system shall display or output a result if no valid tour is found under current settings with clear view of its approach and path taken.	Low
FR-05	Backtracking algorithms should be traceable and easy to follow to understand its prioritizing path decisions.	Medium

Non-Functional Requirements

Req. ID	Requirement Description	Priority (L/M/H)
NFR-01	Performance: The system should handle large boards within a reasonable time limit.	High
NFR-02	Memory Usage: Because we employ complex algorithms such as Q-Learning, we must manage memory carefully so that each training episode remains both feasible in length and efficient in resource consumption.	Medium
NFR-03	Computational Resources (CPU/GPU): We can leverage different approaches, such as parallelizing MCTS threads, exploring multiple paths simultaneously and reducing overall runtime. However, this must be implemented with caution, as performance gains will vary based on each system's hardware specifications.	Medium
NFR-04	Reliability: Critical functions (board generation, move validation) must be tested to minimize errors and ensure consistent results.	High
NFR-05	**Maintainability:** Given that AI can learn different strategies over time, these approaches should be learned so it must be easily accessible for future reference, especially when refining or replicating high-performing training runs for tracing different new heuristics.	Medium
NFR-06	**Expandability:** New algorithms and heuristics should be **traceable** and **easily accessible**, ensuring that they can be added to the existing system without major refactoring and located or referenced quickly for further analysis.	Medium

Project planning

Step	Task	Timeframe	Description & Goals
1	**Investigating MCTS with Warnsdorff-Like Behavior**	~1 Month	- **Verify Hidden Heuristics**: Dive into the MCTS code to confirm whether it truly follows Warnsdorff's rule or if it's discovering an even more efficient heuristic. - **Comparative Analysis**: Compare its results directly to the original Warnsdorff's algorithm, examining why the MCTS-based approach is performing better. - **Algorithmic Insights**: Attempt to identify or extract any "hidden" algorithm that might be driving these unexpected improvements.
2	**Tune Q-Learning & Integrate Neural Network Approaches**	~1 Month	- **Q-Learning Improvements**: Adjust hyperparameters, reward structures, or exploration strategies to enhance the Q-Learning agent's performance, which so far has been suboptimal. - **Neural Network Exploration**: Incorporate a basic NN to see if it can boost results or uncover unique strategies. - **Heuristic Discovery**: Investigate whether the NN-augmented or tuned Q-Learning reveals hidden heuristics worth spotlighting.
3	**Begin AlphaZero-Style Methods**	~1 Month	- **AlphaZero Integration**: Combine a valuable network with MCTS in a self-play setting. - **Heuristic Generation**: Observe if AlphaZero's iterative approach yields new heuristics or improves upon existing methods. - **Performance Tracking**: Evaluate how well this approach scales relative to the classic Warnsdorff or MCTS-only solutions.
4	**Performance & Further Enhancements**	If Time Allows	- **Code Optimization**: Reduce runtime overhead across all algorithms (MCTS, Q-Learning, neural models). - **Hybrid Approaches**: Investigate combining different heuristics or algorithms for a more robust final solution. - **Final Integration**: If feasible, finalize an approach that balances interpretability (uncovering hidden heuristics) and performance (covering larger boards efficiently).

References

1. Wu, Y., Song, W., Cao, Z., Zhang, J., & Lim, A. (2019). *Learning Improvement Heuristics for Solving Routing Problems*. Retrieved from https://arxiv.org/pdf/1912.05784

2. Minton, S., Johnston, M. D., Philips, A. B., & Laird, P. (1992). *Minimizing conflicts: A heuristic repair method for constraint-satisfaction and scheduling problems*. NASA Technical Memorandum 108121. Retrieved from https://ntrs.nasa.gov/api/citations/19930006097/downloads/19930006097.pdf

3. Zhang, Y., Tiňo, P., Leonardis, A., & Tang, K. (2021). *A Survey on Neural Network Interpretability*. Retrieved from https://arxiv.org/pdf/2012.14261

4. Cruz, C. M., & Mitra, B. (2023). *Exploring neural activity and behavior through real-time analysis. Proceedings of the National Academy of Sciences*, 120(2). Retrieved from https://www.pnas.org/doi/pdf/10.1073/pnas.2206625119