

TEUCS : Transformer Encoder-based UAV Path Planning for Cattle Search Problem

Najoua Benalaya ^{*}, [†] Ichrak Amdouni^{*}, Cedric Adjih [‡], Anis Laouiti [†], Leila Azouz Saidane ^{*}

^{*}ENSI, Tunisia, University of Manouba, surname.name@ensi-uma.tn

[†]Telecom SudParis, France, najoua_benalaya@telecom-sudparis.eu, anis.laouiti@telecom-sudparis.eu

[‡]INRIA Saclay, France, cedric.adjih@inria.fr

Abstract—The UAV-based Cattle Search (UCS) problem originates from our previously developed application, where UAVs act as veterinary agents to locate cattle and collect health data [1]. The goal of UCS problem is to compute an optimized UAV path that minimizes the Expected Search Time (EST). Recent advances in learning-based combinatorial optimization show that neural models can construct near-optimal solutions by learning from large datasets. In particular, the Transformer Encoder efficiently captures spatial and sequential dependencies through self-attention mechanisms. Motivated by this, we propose TEUCS, an autoregressive Transformer Encoder trained in a supervised manner. TEUCS introduces several distinctive features compared to existing approaches. Unlike classical routing problems that aim to minimize tour length, our formulation focuses on minimizing EST under uncertainty. Moreover, TEUCS incorporates domain-specific inputs: (i) a containment probability map representing the likelihood of the target being in each grid cell, and (ii) the last visited cell. We evaluate TEUCS against linear programming and a deep reinforcement learning baseline (PPO). Results demonstrate that TEUCS achieves near-optimal performance, with a gap of 0.007% compared to the optimal linear solutions while significantly outperforming PPO baseline.

Index Terms—UCS problem, Transformers, UAVs, Search Path Planning, Seq2Seq architecture.

I. INTRODUCTION

A. Context

UAVs are increasingly equipped with high-resolution imaging systems and advanced sensing capabilities, which makes them valuable tools across various domains, including modern agriculture [2]. In this context, Drone-Vet is a UAV-based system designed for livestock health surveillance. It leverages a UAV acting as a mobile veterinary agent to locate and approach cattle in the field, and subsequently retrieve health-related data stored in a collar-mounted sensor worn by each animal [3].

Developing this application in practice presents many challenges. Animal locations are rarely known in advance due to their dynamic and unpredictable movements. At the same time, UAVs face limited flight time and energy, making full area coverage impractical. Existing tracking solutions (e.g., GPS, RFID, cameras) are costly and impractical at scale [4], highlighting the need for UAVs to autonomously plan efficient, energy-aware paths for rapid cattle detection. This leads us to define the UCS (UAV-based Cattle Search) problem which

aims at computing an optimized UAV path that minimizes the Expected Search Time (EST) to locate a cattle.

One way to design an optimized search path is to (i) discretize the environment into a finite set of cells and (ii) determine the order in which these cells should be visited to minimize the expected time to locate the cattle. Since this requires selecting and ordering elements from a discrete set, the UCS problem can be formulated as a combinatorial path optimization problem. It is related to the Optimal Search Path (OSP) problem and similar to the NP-hard Traveling Salesman Problem (TSP) [5], [6].

Recently, learning-based methods have emerged for solving combinatorial optimization (CO) problems. Machine learning improves efficiency by enriching variable representations, allowing them to capture problem structure directly rather than relying solely on iterative algorithms [7]. Various architectures have been explored, from pointer networks to graph attention networks (GATs) and Transformer models [8].

A Transformer is a deep learning architecture that processes sequential data using attention mechanisms. Initially introduced for NLP [9], it has been adapted to combinatorial optimization problems such as TSP [10]. It consists of an encoder that builds input representations and a decoder that generates output sequences. Its key component is multi-head self-attention (MHSA), which allows the model to focus on the most relevant parts of the input to construct near-optimal solutions.

Building on this, we propose TEUCS, a Transformer Encoder for solving the UCS problem. It generates search paths autoregressively, constructing the solution step by step by predicting the next cell based on previously selected ones. TEUCS is trained in a supervised manner using optimal paths obtained from a linear formulation in our previous work [3].

B. Contributions

TEUCS introduces several innovations beyond a standard Transformer encoder: (i) probability-aware embeddings to encode the belief distribution over grid cells, (ii) a last-visited token to capture sequential dependencies relevant to search, and (iii) a supervised learning framework that approximates Linear Programming optimal solutions for expected search time rather than tour length. Specifically, the main contributions of this work are:

- Transformer-based architecture adapted to solve novel NP-complete probabilistic problem of search path optimization.
- Effective sequence modeling: The model predicts cell coordinates, enabling the generation of smooth search paths. In addition, it incorporates specialized masking mechanisms to respect the constraints of the problem.
- Efficient training paradigm with the data augmentation technique by leveraging the complete dihedral group D_4 symmetries of grid-based problems.
- Developing and Comparing TEUCS performance with both linear program and Deep Reinforcement Learning models.

II. RELATED WORK

Attention-based models replace handcrafted iterative algorithms by learning high-dimensional embeddings that capture relationships between variables. Through attention, they integrate this information in parallel, enabling simple output layers (like a linear classifier) to generate solutions, with processing complexity depending on the task and model capacity.

In this area, Vinyals et al. [11] introduced Pointer Networks, an LSTM encoder–decoder model for TSP, trained on annotated optimal tours (up to 50 nodes). The model generates solutions autoregressively, each step depending on all previous outputs. Bello et al. [12] further extended Pointer Networks by integrating RL algorithm for fine-tuning, using a policy gradient method to directly optimize the model. They applied supervised pretraining on tours generated by Concorde, which expanded the problem size up to 100 nodes.

After this, more research began to shift toward attention-based architectures.

Kool et al. [10] addressed TSP and VRP using a Transformer-based model for graph data, similar to a Graph Attention Network. Trained with REINFORCE, it learns a generalized policy to generate solutions autoregressively, one element at a time. Kool et al. [13] trained Transformer encoders on solver-generated data addressing TSP and VRP instances of up to 100 nodes. Jin et al. [14] introduced Pointerformer, a memory-efficient Transformer-based model with a multi-pointer decoder, demonstrating strong scalability of DRL methods to large-scale TSP instances. Bogrybayeva et al. [15] investigated a TSP variant with drone constraints, employing an encoder–LSTM architecture trained with DRL to optimize both route length and energy cost. Luo et al. [16] were able to improve representational scalability by training large Transformer encoder-only models on synthetic datasets, which were generated by a solver, for instances beyond TSP-500. Finally, Bresson et al. [17] proposed a modified version of the Transformer architecture to tackle TSP and process a set of nodes to learn the relationship between the nodes and generate a city each time period. The model was trained with the Reinforce algorithm and improved with the beam search heuristic.

III. PROBLEM STATEMENT AND SYSTEM MODEL

In this section, we focus on the UCS problem that aims at planning an efficient search trajectory followed by a UAV to locate the stationary cattle¹. within the minimum expected search time.

A. Assumptions

A1: The herding area is modeled as a 2D space discretized into a regular grid, where each sub-area corresponds to a cell.

A2: There is only one stationary target (cattle) inside each cell.

A3: The searcher is one autonomous UAV.

A4: The searcher's movement is constrained (only move to neighbor cells).

A5: The search time is discrete. The time is divided into multiple time periods.

A6: The UAV is equipped with perfect sensors enabling it to detect the target if both of them are in the same cell at the same time.

B. System Model

• Search environment:

Consistent with search theory, the search space is first decomposed into sub-regions to enable efficient planning [18]. We model the problem as a $G \times G$ grid world composed of a finite set of cells. $\mathcal{J} = \{1, \dots, J\}$ represents the search environment; each cell has a probability of the target containment which is denoted p_j . Since the target is stationary the equation (1) is valid. Consequently, it is guaranteed that the target can always be found, albeit potentially requiring a visit to every cell in the worst-case scenario.

$$\sum_{j \in \mathcal{J}}^J p_j = 1 \quad (1)$$

The UAV initially sweeping the AoI (Area of Interest) could be a method to estimate the prior probability distribution.

• Time and motion model: The search time is discrete $\mathcal{T} = \{1, \dots, T\}$, and at each step t the UAV moves to a neighboring cell and inspects it. It may start in any cell j , and follows a computed trajectory until the target is found, ending the mission. Due to kinematic constraints, the UAV can only move to adjacent cells in the following time period $t + 1$. We also assume perfect sensing, meaning the target is always detected if it is in the same cell as the UAV.

C. Objective

The primary objective of the established model is to learn how to construct a search path that minimizes the expected search time which is formulated as follows :

$$\text{Expected search time} = \sum_{j \in \mathcal{J}} p_j \cdot t_j \quad (2)$$

¹we refer with cattle to one or group of animals located in same cell

Besides the major learning goal, the search path involves learning sub-goals, which are:

- Avoid going out of the AoI bounds.
- Visit all the cells to guarantee the target locating.
- No revisiting cells, since the target is stationary; if it was not found in cell j , there is no need to revisit cell j once again.

IV. TEUCS: TRANSFORMER ENCODER FOR UCS PROBLEM

A. Overview

Our model is based on a Transformer [9] encoder architecture. We call our approach *TEUCS* (Transformer Encoder for UCS). It solves the optimization problem by constructing the search path autoregressively as follows:

- 1) Starting from a given cell, the model predicts at each time step the next cell to inspect.
- 2) At each step, the model receives as an input the set of candidate (unvisited) cells and their probabilities p_j values, together with the last visited cell.
- 3) Based on this input, the model selects the next cell to visit.
- 4) The selected cell is removed from the candidate set, and the process repeats until all cells are inspected.

At each step, the next cell is selected from candidate cells using an encoder with a scoring head. TEUCS is trained in a supervised manner on optimal paths generated by a MILP model [3]. Training uses partial paths, with the next cell as ground truth, making the model a next-cell predictor.

B. Model Architecture

The model architecture, shown in Fig. 1, is a stack of standard Transformer encoder layers. Each layer consists of the following blocks:

1) *Input Embeddings*: For the input, the cells are presented as a sequence of tokens: one token per unvisited grid cell and one special token for the UAV's last-visited location. For a $G \times G$ grid with $J = G^2$ cells, each cell $j \in \{1, \dots, J\}$ has integer coordinates $(x_j, y_j) \in \{0, \dots, G-1\}^2$ and an associated probability p_j of containing the target.

To respect the grid structure, we use an embedding by adding separate learned components for each feature of the cell (e.g. one embedding for x , one for y , one for p , etc.). In that way, all the information concerning one cell, is aggregated in a single token.

Thus the initial cell/token embedding $h_j^{(0)} \in \mathbb{R}^d$ is the sum of learnable components:

$$h_j^{(0)} = e_{x_j}^{(x)} + e_{y_j}^{(y)} + W_p p_j, \quad (3)$$

where $e^{(x)}, e^{(y)} \in \mathbb{R}^{G \times d}$ are learnable embedding lookup tables for the x - and y -coordinates (so $e_{x_j}^{(x)}, e_{y_j}^{(y)} \in \mathbb{R}^d$), and $W_p \in \mathbb{R}^d$ (equivalently, $W_p \in \mathbb{R}^{d \times 1}$) projects the scalar p_j into the d -dimensional space.

For the last cell of the current subpath, a last-visited token is embedded similarly with the coordinates and probability

$(x_{i_t}, y_{i_t}, p_{i_t})$ of the last visited cell i_t . To point out that this cell is the last visited cell, and not a candidate cell that can be visit, we add another learned embedding $f \in \mathbb{R}^d$. similarly, the embedding of the last inspected cell in the sub-path is formulated as follows:

$$h_{\text{last}}^{(0)} = e_{x_{i_t}}^{(x)} + e_{y_{i_t}}^{(y)} + W_p p_{i_t} + f \quad (4)$$

The input to the Transformer encoder is a $(J+1) \times d$ matrix formed by stacking the embeddings of all J cell tokens followed by the last-visited token:

$$H^{(0)} = [h_1^{(0)}, h_2^{(0)}, \dots, h_J^{(0)}, h_{\text{last}}^{(0)}] \quad (5)$$

Importantly, the conventional sinusoidal positional encodings is not used in TEUCS. The learned coordinate embeddings ($e^{(x)}$ and $e^{(y)}$) explicitly encode the absolute position of each node, thereby providing the spatial awareness of the model. Additionally, because relationships are found dynamically through the self-attention mechanism, the set of cell tokens is permutation invariant plus their order in the input sequence is arbitrary and meaningless.

2) *Self Attention Mechanism*: In TEUCS, self-attention acts as weighted message passing between cell tokens: each cell aggregates others' values based on query-key compatibility scores computed as follows:

$$u_{ij} = \frac{q_i^\top k_j}{\sqrt{d_k}}. \quad (6)$$

From the compatibility score, the attention weight α_{ij} is driven by applying a softmax. According to this, the cell embeddings are updated as follows:

$$H_i^{(l)} = H_i^{(l-1)} + \sum_j \alpha_{ij} V_j \quad (7)$$

Where $H_i^{(l-1)}$ is the embeddings of cell i from layer $l-1$. Strong relevance between the two cells i and j is indicated by a high score α_{ij} . For example, after softmax normalization, a large attention weight will result if the last-visited node i and another node j have a relevant interaction for the search task because j has a high target probability p_j or is spatially proximate. This is because their projected queries and keys vectors will be mapped to similar directions in space. As a result, a large amount of the value vector v_j will be incorporated into the updated representation for token i , thereby incorporating the pertinent data from j .

3) *Multi-head self-attention (MHSA)*: For head u , queries, keys, and values are projected to dimension d_h where d_h is equal to the embedding dimension d divided by the number of heads h . The attention is applied to produce $Z^{(u)} \in \mathbb{R}^{J \times d_h}$ (weighted sum of all Value vectors for each token).

The MHSA sub-layer is wrapped with a residual connection and then by layer normalization to stabilize the training.

This sequential architecture guarantees that every layer optimizes the representations H from the previous layer, as opposed to learning to predict a brand new transformation

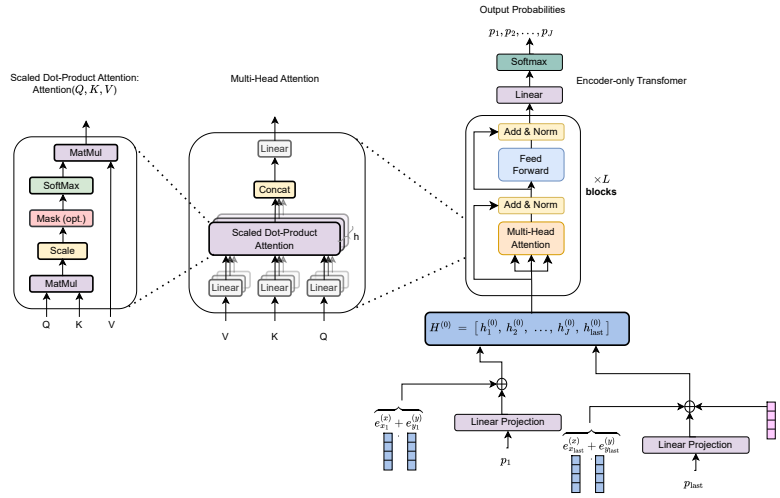


Fig. 1: TEUCS Architecture

from nothing. Next, an FFN sub-layer is added to work on each token.

4) *Output Head and Logits*: The L encoder layers transform each token's representation by aggregating contextual information from candidate cells tokens in the through MHSA. This produces updated representations in $\mathbb{R}^d$ that encode both the token's own features and its relational context within the sequence. The final state of the tokens, precisely the special last-visited token, provides a rich, global contextual representation of the entire search state:

$$\text{Context Vector: } C = H_{last}^{(L)} \in \mathbb{R}^d \quad (8)$$

The context vector is then passed through a linear projection layer to produce a set of logits, one for each cell in the instance. The logits represent the estimated preference score of visiting that cell next and is given as follows:

$$\text{Logits: } z = W_o c + b_o \in \mathbb{R}^J \quad (9)$$

$W_o \in \mathbb{R}^{J \times d}$ is a weight learnable matrix that maps context vector c to the logits z and $b_o \in \mathbb{R}^J$ is a bias vector. the model predicts the next cell by exploiting the embeddings of the last inspected cell. At each time period t , a sub-path of n cells is predicted and a set of candidate cells is retained.

5) *Masking Mechanism*: To enforce the model to respect the non-revisiting constraint, two-masking strategies are employed:

- At the encoder layers: The self-attention mechanism is masked to prevent any token from attending to previously visited nodes, ensuring the representations are built without their influence.
- At the output head: the logits z_j associated with cells J in the sub-path S_t are masked by assigning $-\infty$ to it. By applying a softmax to the masked logits, the final next cell inspection policy π yields a probability distribution

that ensures there is no chance of choosing any previously visited location:

$$\pi(j | s_t) = \begin{cases} \text{softmax}(z_j) = \frac{\exp(z_j)}{\sum_{k \notin S_t} \exp(z_k)}, & \text{if } j \notin S_t, \\ 0, & \text{if } j \in S_t. \end{cases} \quad (10)$$

C. Training and Inference Tuning

1) *The Training Phase*: To optimize the model training we employed the following strategies:

- **Teacher Forcing**: In the training procedure, the model handles sub-paths of different length from the training set, and in each forward pass, the model receives the last visited cell from the current partial path S_t and must decide what cell to inspect next. To facilitate the learning, the model is trained via the “teacher forcing” techniques, where it has to predict or decide which cell to inspect a time period t given the perfect guide; precisely, the last inspected cell at $t - 1$ in the optimal path.

- **Data Augmentation**: We augment the dataset using the eight symmetries of square grids (rotations: 90° , 180° , 270° and reflections), increasing its size by $8\times$. In addition, the optimal search paths are subject to the same coordinate transformation, and thus remain valid and optimal. Each transformation preserves spatial relationships and optimal paths, enabling the model to learn rotation- and reflection-invariant features, reduce overfitting, and improve generalization.

- **Loss Function**: The model learns how to identify the optimal next cell, by minimizing a *masked multi-class cross-entropy loss*. For a given state $s_t = (i_t, S_t, P)$ representing the inspected cell at time period t , the achieved sub-path, and problem instance, the encoder produces raw logits $z_j \in \mathbb{R}^J$ over all possible cells in the instance. Then a mask is applied to z_j to form $\tilde{z}_j$ so the already visited cell logits be ignored

during the decision making process. The target is the optimal next cell i_{t+1}^* extracted from LP trajectories. Hence, the objective is to maximize the probability of selecting the optimal next cell and its logit is: $\tilde{z}_{i_{t+1}^*}$, which is formulated as follows:

$$P(i_{t+1}^*) = \frac{\exp(\tilde{z}_{i_{t+1}^*})}{\sum_{k=1}^J \exp(\tilde{z}_k)} \quad (11)$$

Optimization algorithms are typically framed as minimization approaches. Therefore, the negative logarithm of the probability in the formula (11) is used to adapt the objective to the nature of optimization algorithm. Then, cross-entropy loss $\mathcal{L}$ is computed for the optimal next cell i_{t+1}^* , is formulated as:

$$\mathcal{L} = -\log \frac{\exp(\tilde{z}_{i_{t+1}^*})}{\sum_{k=1}^J \exp(\tilde{z}_k)} = -\tilde{z}_{i_{t+1}^*} + \log \sum_{k=1}^J \exp(\tilde{z}_k) \quad (12)$$

2) *The Inference Phase:* The model outputs the cell with the highest probability of selection $i_{t+1} = \arg \max_{j \notin S_t} P(j)$ repeatedly until all cells are inspected. The model begins by predicting the next cell by continuously modifying the mask to eliminate grid cells that have already been visited. This procedure continues until the complete sequence is produced.

V. SIMULATION AND RESULTS

By leveraging the optimal search path generated by the MILP for UCS in [3], we were able to generate a dataset of 46×10^4 problem instances with its respective optimal search path. With the data augmentation the overall increase in the dataset to 3.68 million samples.

A. Simulation Setup and Hyperparameters

All experiments were performed on a computing server with Intel (R) Xeon (R), RAM 24 GB, with 16 cores equipped with an NVIDIA RTX A5000 GPU. We implemented the core transformer architecture using the PyTorch Transformer module in Python. The model was configured with the following hyperparameters:

- Embedding dimension d : 128
- Number of layers (L): 6
- Number of attention heads: 8
- Feedforward network dimension: 512
- Dropout rate: 0.1

We used the Adam optimizer to train the model for two hours with 50 epochs and a batch size of 1024. The training plan started with a learning rate of 10^{-4} , which was changed during training by a learning rate scheduler.

B. Experiments and Results

Concerning the PPO algorithm used as baseline comparison, we consider the algorithm that we have previously designed for the UCS problem [19]. We evaluated 1000 problem instances. For each map, TEUCS predicts a full search path from every possible starting cell. Restricting to the LP-optimal starting cell, the model matched the LP-optimal trajectory exactly in 876 of 1000 instances (87.6%).

Fig. 3 illustrates the computed trajectories for a problem instance (probability map) for which the solution of TEUCS, obtained by predicting the sequence of cells, appears to be exactly as the optimal search path.

Fig. 2 illustrates the average expected search time for 9 different maps with all possible starting cells. Looking at the generated solutions, TEUCS closely matches the LP for all starting cells. The gap between TEUCS and the benchmark is remarkably small, with minimum value of 0.007 in cell (16), and for a maximum value of 0.25 for starting from cell (10). For some starting cells (14) and (2), the expected search time is lower than the optimal search time, and this does not mean that TEUCS is really better, but the model fails few times to respect the cells adjacency constraint in the case where the candidate cells have extremely low probability of p_j in the range of ($p_j \approx 10^{-6}$). Compared to PPO, TEUCS achieves superior performance by consistently achieving the lower expected search time for 11 starting cells where the highest gap between both models is 0.26 starting from cell (8).

VI. CONCLUSION AND FUTURE SCOPE

This study introduces TEUCS, a Transformer Encoder that generates UAV search paths autoregressively under UCS constraints. At each step, it encodes all unvisited cells and adds a last-visited token for context, while masking already visited cells to avoid revisits. By combining spatial embeddings with containment probabilities, the model captures key factors affecting expected search time. Comparative study show that TEUCS achieves near-optimal performance while significantly outperforming PPO baseline. As future work, we plan to extend TEUCS to handle larger-scale instances. We want to explore hybrid training strategies combining supervised learning with reinforcement learning. We also aim to adapt the problem formulation and constraints to better reflect realistic operational environments, including dynamic belief updates and practical deployment considerations.

REFERENCES

- [1] Najoua Benalaya, Ichrak Amdouni, Cedric Adjih, Anis Laouiti, and Leila Azouz Saidane. Linear programming for uavs search path planning in livestock health monitoring. *Computers and Electronics in Agriculture*, 240:111232, 2026.
- [2] Praveen Kumar Reddy Maddikunta, Saqib Hakak, Mamoun Alazab, Sweta Bhattacharya, Thippa Reddy Gadekallu, Wazir Zada Khan, and Quoc-Viet Pham. Unmanned aerial vehicles in smart agriculture: Applications, requirements, and challenges. *IEEE Sensors Journal*, 21(16):17608–17619, 2021.
- [3] Najoua Benalaya, Cedric Adjih, Ichrak Amdouni, Anis Laouiti, and Leila Saidane. Uav search path planning for livestock monitoring. In *PEMWN*, 2022.
- [4] Chiara Aquilani, Andrea Confessore, Riccardo Bozzi, Francesco Sirtori, and Carolina Pugliese. Review: Precision livestock farming technologies in pasture-based livestock systems. *Animal*, 16(1):100429, 2022.
- [5] K. E. Trummel and J. R. Weisinger. The complexity of the optimal searcher path problem. *Operations Research*, 34(2):324–327, 1986.
- [6] Richard M Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2009.
- [7] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d’horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

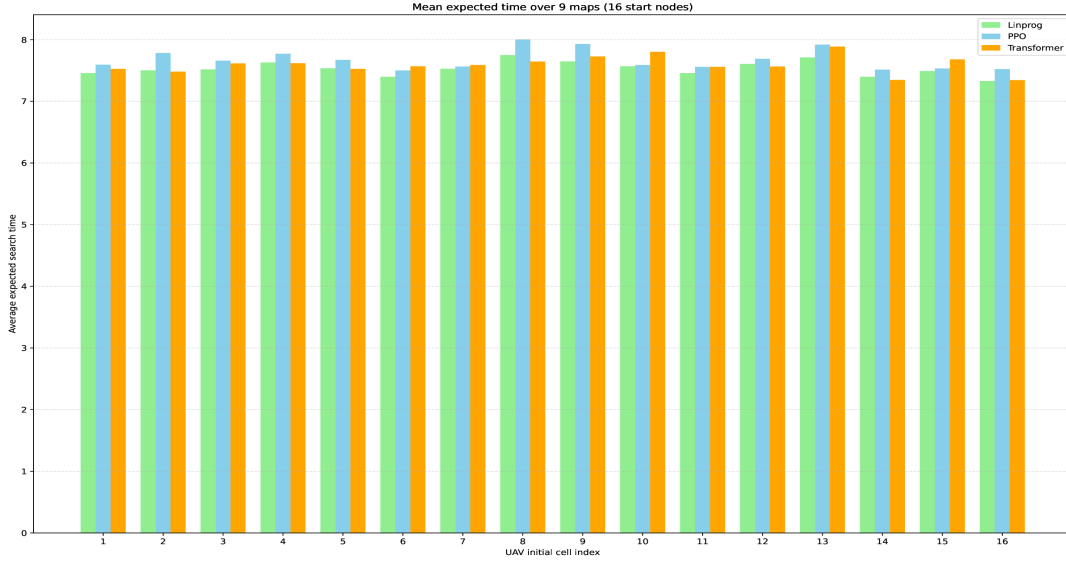


Fig. 2: The average expected search time given different initial positions for the Linear program, PPO and TEUCS

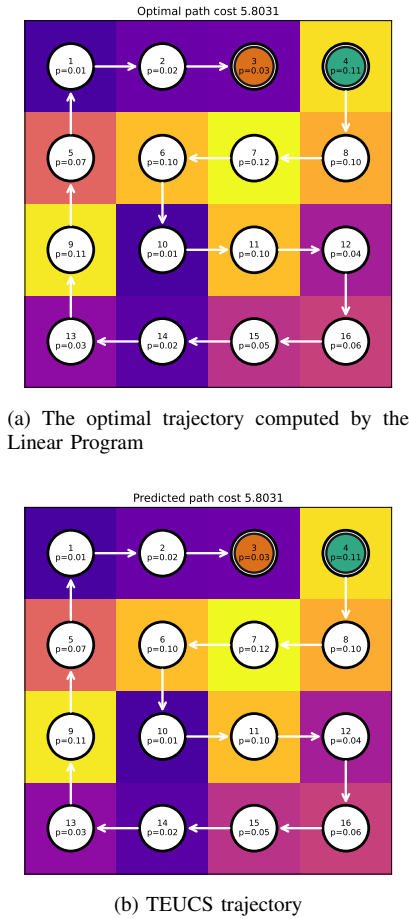


Fig. 3: The computed search trajectories

- [8] Xuan Wu, Di Wang, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L Maskell, and You Zhou. Neural combinatorial optimization algorithms for solving vehicle routing problems: A comprehensive survey with perspectives. *arXiv preprint arXiv:2406.00415*, 2024.
- [9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [10] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.
- [11] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.
- [12] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.
- [13] Wouter Kool, Herke van Hoof, Joaquim Gromicho, and Max Welling. Deep policy dynamic programming for vehicle routing problems. In *International conference on integration of constraint programming, artificial intelligence, and operations research*, pages 190–213. Springer, 2022.
- [14] Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *AAI Conference on Artificial Intelligence*, 2023.
- [15] Aigerim Bogrybayeva, Taehyun Yoon, Hanbum Ko, Sungbin Lim, Hyokun Yun, and Changhyun Kwon. A deep reinforcement learning approach for solving the traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies*, 148:103981, 2023.
- [16] Fu Luo, Xi Lin, Fei Liu, Qingfu Zhang, and Zhenkun Wang. Neural combinatorial optimization with heavy decoder: Toward large scale generalization. *Advances in Neural Information Processing Systems*, 36:8845–8864, 2023.
- [17] Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem. *CoRR*, abs/2103.03012, 2021.
- [18] Bernard O Koopman. The theory of search. ii. target detection. *Operations research*, 4(5):503–531, 1956.
- [19] Najoua Benalaya, Ichrak Amdouni, Cedric Adjih, Anis Laouti, and Leila Azouz Saidane. Deep reinforcement learning approach for uav search path planning in discrete time and space. In *IWCMC*, 2024.