Trade-Off Properties of End-to-End Neural Solvers on the Weapons–Targets Assignment Problem

Ayoub E. Sara K. Charles B.

Laval University

Abstract

This research project presents results related to the trade-offs made when using end-to-end neural solvers instead of traditional solving methods for combinatorial optimization problems. We focused on comparing a state-of-the-art GNN architecture and a traditional solver on a specific assignment problem: the Weapon–Target Assignment (WTA). This defence-related problem seeks to assign weapons (or missiles) from different weapon systems to targets in order to minimize the total expected destructive value of those targets. We used reinforcement learning to train our neural solver in order to learn an optimal policy, where the reward was the negative of the total expected destructive value of all targets after an assignment. Despite challenges, we demonstrate results related to the speed–optimality trade-offs between these two methods and discuss different use cases where one may be better suited than the other.

1 Introduction

Assignment problems are a subclass of combinatorial optimization problems studied in Computer Science and Operations Research (OR). Unlike permutation-class problems (e.g., routing) where one seeks an ordering of inputs, assignment problems map agents to tasks to optimize an objective under constraints. These problems are critical in production planning and logistics. Optimized assignments can yield large savings [1]. Many real-world instances are NP-hard; exact methods guarantee optimality on small instances but become resource-intensive as problem size grows.

Practitioners often trade optimality for computational cost and time by using handcrafted heuristics that reduce the search space [2]. With recent progress in Deep Learning (DL), data-driven approaches (supervised learning to approximate policies or reinforcement learning (RL) to learn them) have been applied to produce strong heuristics from problem instances [3].

1.1 End-to-End Neural Solvers

A rising approach directly maps a problem instance to a solution with an end-to-end policy. Their speed in real-time settings is attractive when applications cannot afford slow optimization. They also reduce dependence on domain-expert heuristics and can be adapted to varying constraints/objectives via simulation and fine-tuning [4].

However, these black-box solvers inherently expose a *Speed–Optimality–Satisfiability* trade-off: higher speed at potentially lower optimality, or lower cost with solutions that may slightly violate constraints unless explicitly enforced. Our objective is to explore these trade-offs by comparing a modern end-to-end learning architecture to a traditional baseline on a relevant assignment problem.

1.2 State of the Art

The application of neural architectures to NP-hard combinatorial optimization (CO) problems dates back to Hopfield networks, which minimized an energy function to solve small TSP instances, and to self-organizing maps [5], which provided early biologically inspired heuristics. The modern resurgence began with Pointer Networks [6], which introduced an attention-based encoder–decoder model capable of handling variable input sizes. This architecture gained traction due to its flexibility and was later improved by RL–based training [2], showing that neural models can learn effective policies directly from data.

Subsequent advances combined Graph Neural Networks (GNNs) and sequence-to-sequence frameworks to encode graph-structured inputs and generate solutions either autoregressively or in a single forward pass. Attention-based variants such as Graph Attention Networks (GATs) further enhanced representational power. These models demonstrated that neural solvers could approach, and in some cases surpass, traditional optimization methods on challenging NP-hard problems, offering a promising balance between adaptability, efficiency, and solution quality

2 Problem Description

The aim of this work is to analyze the trade-offs between traditional solvers based on handcrafted heuristics and the more recent end-to-end neural solvers, which learn data-driven solution policies either through supervised learning (approximation) or reinforcement learning (discovery). More specifically, we seek to address the following key questions:

- *How can we quantify the Speed–Optimality–Satisfiability trade-offs?*
- *How do trade-offs change with key parameters and instance sizes?*
- *How much optimality (w.r.t. an objective) should be sacrificed for a given time gain?*
- *Can a policy be fine-tuned to adapt across instance sizes, constraints, and objectives?*

2.1 Weapon–Target Assignments

We study the Weapon–Target Assignment (WTA) problem: assign munitions from multiple weapon platforms to targets so as to *minimize* the total *expected* survival (equivalently, minimize the post-engagement destructive value remaining). We consider the *static* single-stage variant (SWTA) and a defensive perspective common in the literature.

In time-critical defence settings (e.g., intercepting hypersonic RVs), milliseconds matter. Large-scale WTA instances are NP-hard, and exhaustive search is impractical. Fast, near-optimal black-box solvers therefore become appealing. Specifically, we compare: (i) a state-of-the-art neural architecture trained with RL; and (ii) a traditional MiniZinc-based exact/CP baseline (solver).

3 Proposed Approach

We first formalize the SWTA model, then present the end-to-end neural architecture. Training, baseline, and experimental protocol follow.

3.1 Model for SWTA

Sets and indices. Weapon platforms $w \in \{1, \ldots, W\}$, targets $k \in \{1, \ldots, K\}$.

Parameters.

- $M_w \in \mathbb{Z}_{\geq 0}$: number of missiles available at platform w.
- $P_{w,k} \in [0, 1]$: probability that one missile from w destroys target k.
- $V_k \geq 0$: destructive value of target k.

Decision variables.

$$x_{w,k} \in \{0, 1, \ldots, M_w\} \quad \text{number of missiles from platform } w \text{ assigned to target } k.$$

Constraints (munition limits).

$$\sum_{k=1}^{K} x_{w,k} \;\leq\; M_w, \qquad \forall\, w = 1, \ldots, W.$$

Objective (minimize expected survival value). Assuming independent kill events, the survival probability of target k after $x_{w,k}$ shots from each platform w is

$$S_k(\boldsymbol{x}) \;=\; \prod_{w=1}^{W} \left(1 - P_{w,k}\right)^{x_{w,k}}.$$

Hence the expected survival value is $V_k\, S_k(\boldsymbol{x})$, and the overall objective is

$$\min_{\boldsymbol{x}} \sum_{k=1}^{K} V_k \prod_{w=1}^{W} \left(1 - P_{w,k}\right)^{x_{w,k}}.$$

Example. If $V_k = 1000$, $P_{2,k} = 0.9$, and $x_{2,k} = 2$ with no other shots, then the post-engagement expected survival is $1000\,(1 - 0.9)^2 = 10$.

3.2 End-to-End Learning Architecture

We adopt **MatNet** [8], a bipartite GNN with an encoder–decoder structure tailored to matrix-structured CO problems. The encoder stacks dual graph-attention layers (independent updates for each bipartite side) with edge-weighted attention; the decoder generates assignments autoregressively. Zero/one-hot inputs enable variable-sized instances.

4 Experimentation Protocol

4.1 Training Details

We trained our neural solver model with reinforcement learning to learn an *optimal policy* rather than fit labels (which are costly and may limit generalization) [2, 8]. We use **POMO** [9] (policy optimization with multiple optima) with REINFORCE and greedy rollout. The decoder proposes $m \times W$ parallel solutions by varying the starting missile and keeps the best under the SWTA objective; the reward is the negative objective.

4.2 Experiment Settings

We train on WTA instances of sizes 3×3, 3×4, 4×3, and 4×4 for 100 epochs using Adam (lr 4×10^{-4}), generating 10,000 instances per epoch with batch size 200. Inputs comprise the destruction-probability matrix expanded per missile and the target value vector $\boldsymbol{V}$.

Table 1: Hyperparameters

Type	Value
Batch size	200
Epochs	100
Instances per epoch	10,000
Optimizer	Adam
Learning rate	4×10^{-4}
# Encoder layers	5
# Attention heads	16

4.4 Baseline Method & WTA Instance Generator

We implement a MiniZinc model (Gecode) as a traditional baseline. Metrics: (i) error to optimal survival value; (ii) wall-clock time difference between NN inference and solver time on the same test set. A Python instance generator creates unique WTA datasets: for each instance, $P_{w,k} \sim U[0,1]$ and $V_k \sim U[0,1]$; uniqueness is enforced by hashing concatenated parameters. Instances are saved both as MiniZinc data files and as pickled Python objects. Each instance is solved once by the baseline to produce reference solutions (assignment, survival value, time), captured via the MiniZinc CLI.

5 Results & Discussion

Table 2 reports averages over test instances. As expected, the MiniZinc solver attains lower survival values (optimal) while the MatNet model typically infers faster at test time (except on 4×3 given increased computational load from input expansion). On larger scales (beyond our budget), neural methods are expected to offer greater speedups at inference despite potentially higher training cost.

Table 2: Results trained on 3×3 and out-of-distribution tested on 100 instances of each size.

	3×3		4×3		3×4		4×4	
Method	Survival	Time (s)	Survival	Time (s)	Survival	Time (s)	Survival	Time (s)
MatNet	0.62	0.18	0.49	0.25	0.81	0.20	0.71	0.20
MiniZinc	0.18	0.22	0.07	0.21	0.40	0.21	0.17	0.28

6 Conclusion

We compared an end-to-end GNN (MatNet) trained with RL against a MiniZinc solver baseline on the SWTA problem. Within our compute limits, the traditional solver achieved better results on small instances, while the neural model inferred faster overall. In real-time defense settings, rapid near-optimal inference can justify modest inefficiency. Scaling studies are needed, as prior work shows neural solvers often outperform exact methods in test-time speed on large instances. Future work will address (i) generalization through fine-tuning to new constraints and objectives, and (ii) extension to multistage or dynamic WTA with feedback from realized outcomes.

References

[1] Y. Fan, "Job Scheduling in High Performance Computing," *arXiv:2109.09269*, 2021. https://arxiv.org/pdf/2109.09269.pdf

[2] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural Combinatorial Optimization with Reinforcement Learning," *arXiv:1611.09940*, 2016. https://arxiv.org/pdf/1611.09940.pdf

[3] Y. Bengio, A. Lodi, and A. Prouvost, "Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon," *arXiv:1811.06128*, 2018. https://arxiv.org/pdf/1811.06128.pdf

[4] C. K. Joshi, Q. Cappart, L.-M. Rousseau, and T. Laurent, "Learning TSP Requires Rethinking Generalization," *arXiv:2006.07054*, 2020. https://arxiv.org/pdf/2006.07054.pdf

[5] K. A. Smith, "Neural Networks for Combinatorial Optimization: A Review of More Than a Decade of Research," *INFORMS J. on Computing*, 11(1):15–34, 1999.

[6] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer Networks," *arXiv:1506.03134*, 2015. https://arxiv.org/pdf/1506.03134.pdf

[7] W. Kool, H. van Hoof, and M. Welling, "Attention, Learn to Solve Routing Problems!" *arXiv:1803.08475*, 2018. https://arxiv.org/pdf/1803.08475.pdf

[8] Y.-D. Kwon, J. Choo, I. Yoon, M. Park, D. Park, and Y. Gwon, "Matrix Encoding Networks for Neural Combinatorial Optimization," *arXiv:2106.11113*, 2021. https://arxiv.org/pdf/2106.11113.pdf

[9] Y.-D. Kwon, J. Choo, B. Kim, I. Yoon, Y. Gwon, and S. Min, "POMO: Policy Optimization with Multiple Optima," *NeurIPS 33*, 2020.