Trade-Off Properties of End-to-End Neural Solvers

Ayoub.E *Sara.K* *Charles.B*

Laval University

Abstract — This research project presents some 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 the results of 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 parametrize 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 demonstrated results related to the speed-optimality trade-offs between those two methods and discussed different uses cases where one may be better suited than the other.

I. INTRODUCTION

Assignment problems are considered to be a subclass of the larger set of combinatorial optimization problems, which are commonly studied in Computer Science and Operational Research (OR). Unlike the permutation subclass problems where we are interested in finding a permutation of the input (ex: Routing Problems) that optimize a certain objective function, an assignment problem is seeking to produce a mapping from agents to tasks such that an objective function is optimized, and a set of constraints are respected.

Popular assignment problems are often critical to production planning and logistics, which makes the industrial sector more efficient. Models and algorithms that seek to enhance the optimality of assignments in function of certain objectives can represent millions of dollars of savings for many corporations [1]. But few of those problems can be solved on a polynomial scale, which means that many of them encountered in real life are NP-hard problems. In other words, as we scale the size of a problem, the more difficult it becomes to obtain an optimal solution.

Traditional approaches for solving those type of problems have mainly revolved around using domain expert knowledge to produce handcrafted heuristics, which can be understood as a set of strategies that allows a strong reduction of the search space in order to find optimal solutions. Indeed, while "Exact methods" are preferred for smaller instances as they often guarantee an optimal solution, those becomes much more resources intensive as the problem gain a certain degree of complexity. Hence, actors often decide to trade off optimality for computational cost & time via heuristic methods that allow a solver to select variables & values branching to filter the search space much more rapidly [2].

But as the recent advances in Deep Learning & Natural Language Processing started to be applied to those combinatorial optimization problems, many classes of data-driven learning approach were seeing increased usage in approximating (via Supervised Learning) or learning a policy (via Reinforcement Learning) that allow to obtain strong heuristics for appropriate variable/value branching just with problem instances. This approach is very useful for augmenting the problem-solving process with valuable pieces of information like pointed out by [3].

1.1) End-to-End Neural Solvers

Another rising class of data-driven learning approach is to approximate/learn an end-to-end policy that take a problem instance as input and output directly a solution to our problem. Their degree of rapidity in real-time usage attracted interested recently as many real-life applications cannot afford to waste time on generating good solutions to more complex problem instances. Hence their potential use can generate strong utility to customers as they can first be more efficient for real-time fast query applications, and second, they do not require domain expert knowledge, which can represent a significant cost for implementing those type of solutions. Additionally, adapting those type of architecture to different constraints & objective functions of the same problem may be more practical via simulation & fine-tuning [4].

But as expected, a lot of trade-offs are needed to benefit from end-to-end neural solvers. Indeed, many of those black-box solvers use the Speed-Optimality-Satisfiability trade-off curve to become more appealing, which then let the customer select parameters of this trade-off: higher speed, but lower optimality according to your objective function(s). Lower cost, but possibly solutions that are not 100% satisfiable while being near-optimal.

Therefore, the objective of this research project is to explore the trade-off properties involved in those kinds of decisions. By using a current state-of-the-art learning architecture and a traditional baseline, we will seek to quantify this trade-off on a particular problem that can be well encoded using modern end-to-end solvers. The different sections of this paper are as follow: we will start by a review of the current **state-of-the-art** in End-to-end Neural Solvers, then it will be followed by a detailed **problem description**, a layout of our selected **approach**, the **experimental protocol** chosen, and finally the **results**, **analysis**, and **conclusion** of our research.

1.2) State-of-the-art

The trend of applying neural networks architectures to NP-Hard combinatorial optimization problems is often said to be initiated by the use of Hopfield Networks that were optimizing an energy function to solve some instances of the TSP, which worked partially for some set of instances []. As methods for encoding the problem and avoiding local minima were progressing, better results were generated using this type of methods. Another popular method of the ending 20st century were self-organizing neural networks, which are well detailed in the paper of [5].

But if we focus only on the recent pick up of interest of DL for C.O, this can be traced back to the introduction of pointer network by Vinyals et al. (2015) [6], where a pair of RNNs encoder-decoder were used to first encode the problem structure (in this case cities for the TSP) via embeddings, and then decode sequentially the solutions via an attention mechanism that generate conditional distribution over the input nodes, hence generating a permutation over its respective inputs. The fact that it was possible to use the architecture over different sizes of input graph made it really appealing to researchers around the world, hence becoming a popular architecture that was improved by many like [2], that used Reinforcement Learning training instead of Supervised Learning, proving that we can extract better results from it.

As time progressed, Graph Neural Networks (GNNs) and Sequence-to-Sequence (seq2seq) learning approaches were becoming common for the encoder-decoder pair in order to approximate/learn the desired policy that output solutions directly from a specific problem instance. Some variants leveraging the attention mechanism like Graph Attention Networks [7] are also considered state-of-the-art, some generating solutions autoregressively in a step by-step fashion, others making use of non-autoregressive approaches. But all proved that it was possible to match or exceed the results obtained by traditional solver approach on certain unsolved NP-Hard problems.

II. PROBLEM DESCRIPTION

Across this research project, we are interested in analyzing the Trade-off Properties between Traditional Solvers with handcrafted heuristics and End-to-End Neural Solvers that learn a policy from data in order to directly output solutions to a problem. In a more precise way, here are some questions that we find are worth some answers:

- *How can we quantify the Speed-Optimality-Satisfiability Trade-offs?*
- *How to improve those trade-offs by changing some key parameters?*
- *How do those parameters change with different instances sizes?*
- *How much optimality (according to an objective function) do we need to sacrifice for a certain time gain?*
- *Can we fine-tune a policy network to adapt solutions to different instances sizes, constraints & objective functions?*

2.1) WTA Problem

To carry on our study, we decided to choose a problem that is exactly benefiting from improvement in this trade-off, which means that optimal solutions cannot be allowed to take too much time to be generated while any poor solution according to the objective function can have catastrophic consequences for a country. Hence, we choose the Weapon-Target Assignment (WTA) problem, a popular defense-related problem common to Operations Research that seeks to assign weapons to targets in order to defend strategic assets (defensive perspective) or degrade capabilities of an enemy (offensive perspective). Two main versions of this bipartite assignment problem exist:

- **Static**: All the inputs to the problem are fixed. Therefore, all targets & weapons are known, and all weapons engage targets in a single stage.

- **Dynamic**: Multi-stage problem where some weapons are engaged at the targets at a stage 1, then the outcome of this engagement is analyzed and strategy for future stage(s) is decided.

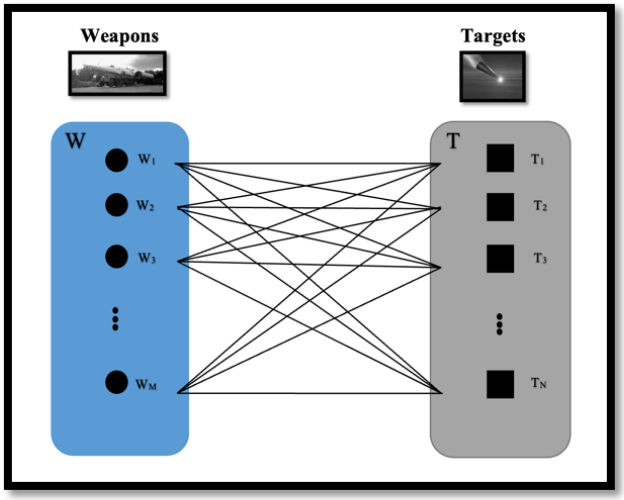

Figure 1: WTA as a Bipartite Assignment Problem

In order to avoid complications of a multi-stage problem like computing a transition probability matrix that k targets survive having used i weapons, we choose the static version of the WTA, hence the acronym SWTA. Additionally, we will select the defensive perspective of the problem as it is the most common in the literature and most intuitive for us. Hence, the problem can be seen as efficiently assigning weapons to targets so that the total expected destructive value of the incoming targets is minimized.
(More details on the problem model are presented in the next section)

To explain why the trade-off matter when it comes to the defensive WTA, let us imagine a country A defending against incoming swarms of ICBMs (Inter-Continental Ballistic Missiles) from country B with their Re-Entry vehicles manoeuvring in the atmosphere at hypersonic speeds coming directly towards your key communication, energetic & military infrastructures. In those types of situations, every millisecond counts when it comes to generating an optimal assignment matrix of the available weapons (ABM, laser, EW...) to the closing-in targets in function of the problem parameters. Hence, for large instances of weapon-target pairs that will characterize modern warfare, generating solutions via traditional solvers is impractical due to the time necessary to explore the huge search space of this NP-Hard problem. Therefore, fast black-box solvers that can output real-time near optimal solutions to large instances of this type of bipartite graph problem may seem like a good option.

In order to explore the speed-optimality trade-off and the related questions seen above, we decided to compare solving instances of the problem using 2 different approaches: first, a state-of-the-art deep learning architecture parametrized via reinforcement learning and then, a traditional solving method using MiniZinc solver "Chuffed" to output solutions to our optimization model.

III. PROPOSED APPROACH

Across this section, we first detail our model for the Static WTA, then present the End-to-End Neural Solver Architecture selected. The next section will then focus on the training algorithm used to parametrize those networks, supplemented by details on our baseline approach and our experimental protocol.

3.1) Model for the SWTA

A classical assignment problem can be understood as a situation where N tasks are assigned to M agents. Each agent can only perform one task, but a task may be performed by multiple agents. Any assignment comes with an incurred cost (or reward), which then need to be minimised (or maximised) effectively while respecting a specific set of constraints. In this paper, we are undertaking the Weapon-Target Assignment problem, which is defined over a set of W Weapon Platforms each having their own M_w missiles available. The missiles can be individually assigned to any or none of K targets which need to be destroyed. Each Weapon Platform has a constant probability $P_{w,\,k}$ of destroying each of the targets. The optimal assignment will maximize the total destructive value over the targets, or equivalently minimize the total survival value, where the complement of the destructive probability is used.

Therefore, for any specific instance of the problem, the corresponding inputs to the model are the following:

- Matrix $P_{w,\,k}$ -- Probabilities for Weapon Platform w to destroy target k
- Vector V_k -- Expected Destructive Value for all k targets

Those parameters can be viewed from multiple perspective. For example, we can consider that the enemy targets are all of the same type (ex: ICBMs of type A), but what determines their destructive value V_k is their position, velocity, and altitude compared to allied strategic assets. In the same logic, we can consider that the Weapon Systems are all of the same type (ex: ABM of type A), but what determines their probability P_{ki} of destructing target k is their geographic location compared to those targets. In other more complex situation, different type of targets and weapons systems may be considered, but the parameters nature will stay the same.

The outputs are the assignment matrix whose dimension is the number of Weapon Platforms by the number of targets. Each value inside the matrix represents the number of missiles of a specific Weapon Platform has been assigned to a specific target. Those are the decision variables of our problem. Ideally, when it comes to our deep learning architecture, this matrix should be produced sequentially, vector by vector, where a conditioned distribution for a specific missile is generated over all possible targets k, and the highest probability correspond to the optimal assignment. This would be repeated for each missile for the current Weapon Platform. And to

ensure near-optimality and satisfiability, multiple matrices can be generated starting from different missile for all Weapon Platforms. Then, all those possible solutions could be tested in order to choose the optimal one.

After all the assignments are made, we can use the assignment matrix to compute the value of our objective function, which is represented by the total over all targets of their expected survival value after the missiles were fired. For example, consider a target k_1 with destructive value V_k of 1000, and 2 missiles from Weapon Platform w_2 were assigned to it. We can take the probabilities for w_2 to destroy target k_1, say $P_{w,k} = 0.9$, and compute the new expected survival value for target $k_1 = 1000 * (1 - 0.9)^2 = 10$ units.

Below is the formal definition of the model used for the SWTA problem as described above:

- *Parameters & Constants:*
  - Number of Weapon-Platform W
  - Number of targets
  - Matrix of destruction probability P_{WxK}
  - Vector of target values V_K
  - Vector of missiles in each Weapon-Platform M_W

- *Decision Variables & Domains:*

X, the assignment matrix of weapon types to targets.

$$dom(x_{w,k}) = \{0, \ldots, M_w\}\, \forall w \in \{0, \ldots, W\}, \forall k \in \{0, \ldots, K\}$$

- *Constraints*

The number of missiles assigned for a weapon platform should not exceed the number of missiles for that weapon platform.

$$\sum_{i=1}^{k} x_{w,k} \leq M_w \;\; \forall w \in \{1, \ldots, W\}$$

- *Objective function:*

We minimize the survival value, which works by using the survival probability $1 - P_{w,k}$.

$$\min \quad \sum_{k=1}^{K} \left(V_k \cdot \prod_{w=1}^{W} (1 - P_{w,k})^{x_{w,k}} \right)$$

3.2) End-to-End Learning Architecture

As this problem can be represented as a bipartite graph structure with weighted edges, this makes it well suited for being encoded using many types of Graph Neural Networks (GNNs) and recent state-of-the-art seq-to-seq architectures. We adopted Mat-Net GNN [8] model architecture on WTA problem as one of the most recent approaches introduced for solving bipartite CO problems.

Mat-Net has an encoder-decoder structure to capture a suitable representation of the problem matrix and optimizes the solution using reinforcement through this representation. An encoder is a neural network that embeds an input matrix to a latent space with different dimensions or properties, which is more efficient for some desired processing. A decoder, as its name suggests, is a neural network that undoes the encoder embedding by mapping the learnt representation to the original space, making it possible to read and understand the results.

Mat-Net's encoder is inspired from the node-embedding framework of Graph Attention Networks (GATs), which consists of multiple graph attentional layers stacked together. A classic attention layer as in GAT is a neural network layer that gets the vector representations of a graph's nodes $\hat{h}_v$ and uses an update function based on some form of aggregation (massage passing) between neighboring nodes as formulated below to update the representation to vector $\hat{h}_v{}'$.

$$\hat{h}'_v = \mathcal{F}\big(\hat{h}_v, \{\hat{h}_w \mid w \in \mathcal{N}_v\}\big)$$

The update function F in attention layer is a learnable function composed of multiple attention heads which aggregate the attention score (a predefined function) of each node v and its neighbors $w \in N_v$ representation vectors. In other words, we want to learn how important node v features are for node w and this is what is called the attention coefficient, which is a weight for each edge of the neighbors that tells us how much attention we should pay to that specific node.

What makes Mat-Net different is adopting two independent update functions for the two sets of nodes in the bipartite problem in each attention layer, and also considering the graph's edge weights in computing the attention score as formulated below.

$$\begin{aligned}
\hat{h}'_{a_i} &= \mathcal{F}_A\big(\hat{h}_{a_i}, \{(\hat{h}_{b_j}, e(a_i, b_j)) \mid b_j \in B\}\big) \quad \text{for all } a_i \in A, \\
\hat{h}'_{b_j} &= \mathcal{F}_B\big(\hat{h}_{b_j}, \{(\hat{h}_{a_i}, e(a_i, b_j)) \mid a_i \in A\}\big) \quad \text{for all } b_j \in B.
\end{aligned}$$

In the above formula, A and B are the two sets of nodes and e is the edge weight between a pair of nodes. In our model to initializes the representation vectors for the two sets, missiles & targets, zero-vectors and one-hot vectors are fed as input to the model, similar to Mat-Net original paper. This allows the model to support variable-sized inputs by processing different problem instances uniformly.

The dual attention layer structure defined above along with the overall architecture of the Mat-Net model is presented in the following picture to make the whole structure clearer.

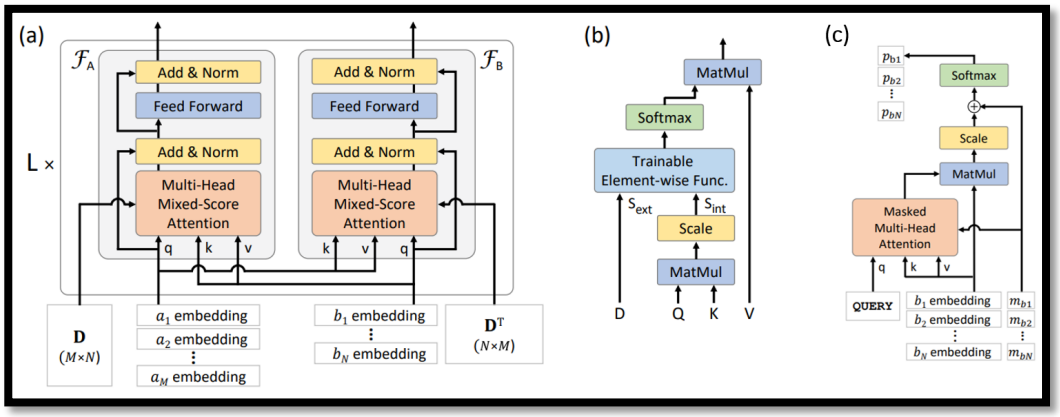

Figure 1: Mat-Net model structure: a) overall architecture, b) encoder multi-head mixed score attention, c) decoder architecture

IV. EXPERIMENTATION PROTOCOL

4.1) Training Details

As mentioned earlier, we used reinforcement learning to parametrize our model instead of supervised learning. Similar to the original paper, this choice was made because we aimed to learn an optimal policy and not approximate one that depends on the labels, which are resource intensives to produce and often limit the policy learning according to those specific set of examples that may not be representative of the true underlying distribution. In supervised case even if the optimal solutions are used as labels for training, the generalization capabilities will be inferior to an RL agent exploring states and their respective rewards, based on [2,8].

The POMO [9] training algorithm was adopted to perform RL on the model and the decoder, which uses reinforce gradient estimator with greedy rollout. POMO allows the decoder to produce MxW_P solutions in parallel, each starting from a different missile, and chooses the best solution according to objective function. The objective function for each missile target assignment, was calculated based on the corresponding formula in 3.1, and its negative value was given to the RL agent as the reward to evaluate and compare its assignments in order to maximize the reward or in other terms minimize the overall survival-destruction values of the missiles. The agent solutions were generated autoregressively, which means at each decoding iteration of a specific missile-weapon, we generate a distribution over all targets k, and we select the highest probability between 0 and 1. Then, all other numbers of the vector are set to 0.

4.2) Experiment settings

We train each of the models for WTA 3x3, 3x4, 4x3 and 4x4 for 100 epochs, using Adam optimizer with learning rate set to 4 × 10−4 without a decay. There are 10,000 randomly generated problem instances in each epoch, processed in batch size of 200. A summary of our hyperparameters is shown in table (I).

The input to the model is a matrix similar to the destructive probability P_{WxK}, but with redundant rows for each missile of the weapon platforms, resulting to a matrix of size P_{2WxK} as we considered there are two missiles available from each platform. The other input used for calculating the objective function is vector V_k. More details on objective function and other model data are provided in the earlier section 3.1.

Table I: Hyperparameters

Type	Value
Batch size	200
No. of epoch	100
No. instances generated / epoch	10*1000
Optimizer	Adam
No. of heads	16
No. of Encoder layers	5
Learning rate	4×10^{-4}

4.4) Baseline Method & WTA Instance Generator

To evaluate our neural-based approach, we compare its performance with the performance of a traditional solver approach. The performance metrics which are of interest are *i)* the error margin of the approximate solution predicted in comparison to the optimal solution and *ii)* the execution time difference between the inference of the network on a set of test examples and the solving process of its solver counterpart on the same set of examples. As such, we developed a baseline method which is implemented as a MiniZinc model. The MiniZinc model uses the Gecode solver to find the optimal assignment of weapons to targets. The baseline is then used to produce ground truth solutions of WTA instances to allow the measurement of performance metric. The measurement is done by averaging over the WTA instances present in the test set both the survival values predicted by the network and the reference survival value found by the baseline. By observing the difference between these two values, we can determine whether or not the network has correctly learned from its training how to predict a solution which approaches the optimal survival value. In addition, the time taken by both the solving procedure of the baseline method and the network approach is recorded to measure performance metric.

Given that we had to produce a considerable number of WTA instances in order to train and test the network, we automated the process of creating these instances by developed a WTA instance generator. The generator is implemented in Python and takes as input the number of instances N to generated and the problem parameters which will be shared by all the instances. These fixed problem parameters are the number of weapon types, the number of munitions for each weapon type, and the number of targets. For each of the N instances, the generation process samples from a uniform distribution in the range [0, 1] the destruction probabilities for each pair of weapon type and target. Similarly, the target values are also samples from the same distribution over [0, 1] the value of each target. To ensure that each instance generated is unique, we compute the hash of the array resulting from the concatenation of all the generated parameters and store it. For each subsequent generated instance, we compute its hash in the same fashion and iterate until we find parameters which have not yet been used. The uniqueness of each instance ensures that there is no overlap between the training and test set, and as such confirms that we correctly measure its generalization performance using the test set. After the instances are created, they are individually saved in the MiniZinc datafile format and collectively saved as a python dictionary persisted using the *pickle* module. The datafile format is used by the baseline method and the python objects are used by the network for training and testing. Additionally, after being generated, each instance is solved exactly using the baseline method to produce the reference solutions. Using the MiniZinc command line toolkit to automate the solving of these instances, we can capture the output of the MiniZinc solver and save the solution as a csv file containing the final assignment of weapons to targets, the survival value, and the time taken by the solver.

V. RESULTS & DISCUSSION

Table 2 displays the results we got from our experiments, which are average values over all instances for different number of target and weapons. As we can see, the MiniZinc solver outperformed on results, which was expected due to Mat-Net being an approximate method. On the other hand, the execution time of Mat-Net is lower in test time, except for the WTA- 4x3 problem. This is because the enlargement of the input matrix results to larger network and more computational cost with a nonlinear relation. This is not supervising since even though, we couldn't do larger tests, neural network-based methods' time is expected to outperform the exact solvers in higher scales, and that is where they really show their potential. Therefore, in small samples it is normal if an exact solver finds the solution faster.

Table II: Results for Mat-Net trained on 3x3 instances and out-of-distribution tested on 100 3x4 instances, 100 4x3 instances, 100 4x4 instances

Method	**WTA - 3x3**		**WTA- 4x3**		**WTA - 3x4**		**WTA - 4 x 4**	
	Average Survival Value	Average Time (s)	Average Survival Value	Average Time (s)	Average Survival Value	Average Time (s)	Average Survival Value	Average Time (s)
Mat-Net	0.62	0.18	0.49	0.25	0.81	0.20	0.71	0.20
MiniZinc Solver	0.18	0.22	0.07	0.21	0.40	0.21	0.17	0.28

To understand how the results were computed for the MiniZinc baseline method, we refer to Section 4.4. The baseline results are reproducible using the python notebook *WTA_generator.ipynb* in addition to the MiniZinc model *WTA.mzn* provided as supplementary material to this paper.

The Mat-Net architecture was developed by altering the code provided by the original paper authors at https://github.com/yd-kwon/MatNet.git for ATSP problem. The main changes we applied was to redesign the input generator to generate WTA samples in the form of weapon-target destruction probability matrix and target value vector, and the objective function to calculate the RL reward as of the corresponding formula in section 3.1, which does a weighted sum over survival possibility and value of targets. Our version of Mat-Net code is uploaded along the report. To train or test a model accordingly train.py or test.py must be run using python. The default parameters are set for WTA 3x3, which needs to be changed to use this architecture for other instance sizes.

VI. CONCLUSION

We initiated this research project to investigate some trade-offs made when using end-to-end neural solvers instead of traditional solving methods for combinatorial optimization problems. Despite numerous challenges, we produced results comparing a state-of-the-art GNN architecture and a traditional Mini-Zinc solver for solving the Weapon-Target Assignment (WTA) problem. Due to limited time and resources, we were restricted in our ability to analyse further those trade-offs and scale our experimentation to large instance sizes, which should have required large computing power and time.

As we can see from the results, in case we care for accuracy more than time, the Mini-Zinc solver is a more reliable choice that optimizes the objective function with guarantee to respect the constraints. On the other hand, the Mat-Net model, performs faster in most cases. Therefore, in real time applications for example in real weapon firing situation that we have access to a large number of munitions in comparison with targets, it makes sense to adapt such a method and immediately hit the targets at the cost of wasting some missiles. Unfortunately, we did not have enough computational resources and time to do experiments in higher dimensions but based on results mentioned in other papers, despite the longer training time, it is expected that an approximate end-to-end model such as Mat-Net, significantly outperforms the exact optimizers at test on large enough datasets.

Although recent research including our work shows that the end-to-end neural solvers have the potential to compete with conventional CO solvers, more investigation is needed in the area to improve these newly introduced methods. An interesting area for further research is neural solvers' generalization or transfer learning in terms of the possibility of fine-tuning them for different constraints and objective functions than initially trained for, which could prove that the learned policy can be slightly adapted to different versions of the problem only with some light additional training. Another future work possibility specific to WTA problem is to design the model as a multistage process in which the missiles are fired at targets at each stage, and instead of considering the destructive probability, the real results are considered to decide whether a target is hit and done or to assign another stage to its process to fire at it.

VIII. REFERENCES

[1] Job Scheduling in High Performance Computing, Yuping Fan. Illinois Institute of Technology. https://arxiv.org/pdf/2109.09269.pdf

[2] Neural Combinatorial Optimization With Reinforcement Learning. Irwan Bello, Hieu Pham, Quoc V, Le, Mohammad Norouzi, Samy Bengio.Google Brain. https://arxiv.org/pdf/1611.09940.pdf

[3] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. *Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon.* https://arxiv.org/pdf/1811.06128.pdf

[4] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, Thomas Laurent. *Learning TSP Requires Rethinking Generalization.* https://arxiv.org/pdf/2006.07054v4.pdf

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

[6] Oriol Vinyals, Meire Fortunato, Navdeep Jaitly. *Pointer Networks*. https://arxiv.org/pdf/1506.03134.pdf

[7] Wouter Kool, Herke Van Hoof, Max Welling. *Attention Learn to Solve Routing Problems!* Https://Arxiv.Org/Pdf/1803.08475v3.Pdf

[8] Yeong-Dae Kwon, Jinho Choo, Iljoo Yoon, Minah Park, Duwon Park, Youngjune Gwon Samsung SDS. *Matrix Encoding Networks for Neural Combinatorial Optimization.*
https://arxiv.org/pdf/2106.11113.pdf

[9] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems 33*, 2020.