\subsection{Unsupervised Learning}

Most successful machine learning techniques fall under supervised learning where a mapping from training inputs to outputs is learned. However, Bello et al. \cite{bello2016neural} argue that supervised learning does not apply to most combinatorial optimization problems because one does not have access to optimal labels. Unlike supervised learning, one can unsupervised learning to train a verifier to compare the quality of a set of solutions and provide some reward feedback to a learning algorithm. A lot of these works utilize reinforcement learning algorithms such as deep Q-learning and policy gradient.

One of the first approaches to use unsupervised learning to tackle COPs is by Bello et al. \cite{bello2016neural}. They present a framework to tackle a popular COP called TSP using neural networks and reinforcement learning, specifically policy gradients \cite{williams1992simple}.

% They propose two approaches based on policy gradient: RL pretraining and active search. RL pretraining uses a training set to optimize a recurrent neural network (RNN) that parameterizes a stochastic policy over solutions using the expected reward as objective. The policy is then fixed at test time and inference is performed using greedy decoding or sampling. Active search does not involve pretraining. It starts from a random policy and iteratively optimizes the RNN parameters on a single test instance, using the expected reward objective. It keeps track of the best solution sampled during the search. Bello et al. consider 2D Euclidean graphs with up to 100 nodes. They compare their result with a supervised learning approach to solve TSP such as Vinyals et al. \cite{vinyals2015pointer}. They not only obtain solutions close to optimal results, but they also illustrate that their approach is flexible by testing the same method on the KnapSack problem for which they get the optimal result for instances with up to 200 items.

Bello et al. \cite{bello2016neural} follow the approach of Vinyals et al. to generalize beyond a pre-specified graph size. Vinyals et al. \cite{vinyals2015pointer} makes use of a set of non-parametric softmax modules, resembling the attention mechanism from Bahdanau et al. \cite{bahdanau2014neural}. Vinyals et al. is further inspired by Sutskever et al. \cite{sutskever2014sequence} as described in the supervised learning section above. We will not go into the detail of sequence-to-sequence learning or pointer network again in this section, however, we will outline some crucial differences between these two approaches.

Vinyals et al. \cite{vinyals2015pointer} propose training a pointer network using a supervised loss function comprising of conditional log-likelihood \cite{vinyals2015pointer}. This approach factors into a cross-entropy objective between the network's output probabilities and the targets provided by a TSP solver. However, Bello et al. \cite{bello2016neural} claims such an approach is undesirable for solving NP-hard problems because of three main reasons. First, the performance of the model is tied to the quality of the supervised labels. Second, getting high-quality labeled data is expensive and maybe infeasible for new problem statements. Finally, a person who is trying to solve such a problem cares more about finding a competitive solution more than replicating the results of another algorithm.

In contrast, Bello et al. propose model-free policy-based reinforcement learning (RL) to optimize the parameters of a pointer network because they believe that RL provides an appropriate paradigm for training neural networks for COPs, especially because these problems have relatively simple reward mechanisms that could be used at test time. Their training objective is the expected tour length. They use a well-known REINFORCE algorithm \cite{williams1992simple} to formulate the gradient of their network. They also introduce an auxiliary network called an actor-critic \cite{joel2002actor} which they use as a parametric baseline to estimate the expected tour length found by their policy. They believe that it typically improves learning. The critic is trained with stochastic gradient descent on a mean squared error objective between its predictions and the actual tour lengths sampled by the most recent policy. Their critic architecture comprises of 3 neural network modules: an LSTM encoder, an LSTM process block, and a 2-layer ReLU neural network decoder. Their actor-critic training algorithm is closely related to the asynchronous advantage actor-critic (A3C) proposed in Mnih et al. \cite{mnih2016asynchronous}.

Bello et al. consider two search strategies called sampling and active search. Under sampling, they simply sample multiple candidate tours from their stochastic policy and select the shortest one. Under active search, they refine the parameters of the stochastic policy during inference to minimize the tour length on a single test input. They claim that this approach proves to be competitive when starting from a trained model, however, it also produces satisfying solutions when starting from an untrained model. They refer to these approaches as RL pretraining-Active Search and Active Search respectively.

They test their model's performance with 3 benchmark tasks: Euclidean TSP20, 50, and 100. They compare their model against 3 different baselines: Christofides \cite{christofides1976worst}, vehicle routing solver OR-Tools \cite{google}, and optimality \cite{applegate2006traveling}. They note that training with RL significantly improves over supervised learning \cite{vinyals2015pointer}. All their methods comfortably surpass Christofides' heuristic \cite{christofides1976worst}. They also found out that both of their greedy approaches are time-efficient and just a little worse than optimality. Besides, they found out that searching at inference time proves crucial to get closer to optimality. However, that comes at the cost of running times. They conclude that RL pretraining-Sampling and RL pretraining-Active Search are the most competitive Neural Combinatorial Optimization methods as they recover the optimal solution in a significant number of their test cases. For a small solution space, they report that RL pretraining-Sampling outperforms RL pretraining-Active Search in both quality and speed. However, for larger solution space, RL pretraining-Active Search proves superior.

Moreover, they also applied their approach to the KnapSack problem to test the flexibility of their model. They found out that RL pretraining-Greedy yields solutions that are just 1\% less than optimal on average and Active Search solves all instances of KnapSack problems of size 50, 100, and 200 optimally.

Nazari et al. \cite{nazari2018reinforcement} generalize the framework presented in Bello et al. \cite{bello2016neural} to include a wider range of combinatorial optimization problems such as the Vehicle Routing Problem (VRP). They report that since the approach proposed in Bello et al. assumes that the system is static over time, it cannot be applied directly to problems such as VRP. In VRP, the demand changes over time in the sense that once a node has been visited, its demand becomes effectively zero \cite{nazari2018reinforcement}. Therefore, they propose a simpler approach to the Pointer Network \cite{vinyals2015pointer}. However, their model can efficiently handle both static and dynamic elements of the system. They argue that the RNN encoder used in \cite{bello2016neural} adds an extra complication to the encoder. Since there is no meaningful order to the input set in the combinatorial optimization problem, they claim that the RNN encoder is actually unnecessary. By omitting it, the approach can be made much more general. Therefore, in their model, they simply leave out the encoder RNN and directly use the embedded inputs instead of the RNN hidden states. Their policy model consists of an RNN decoder coupled with attention mechanism which forms a distribution over the feasible destinations that can be chosen at the next decision point.

\begin{figure\*}

\centering{

\includegraphics[scale=0.9]{graph-embedding.png}}

\caption{Illustration of the framework proposed in Khalil et al. \cite{khalil2017learning} as applied to an instance of Minimum Vertex Cover.Figure adapted from \cite{khalil2017learning}}

\label{fig:graph-embedding}

\end{figure\*}

One of the first approaches to solve the COPs using unsupervised deep learning over the graph was proposed by Khalil et al. \cite{khalil2017learning}. By this point, we have referenced this work multiple times and we have also already discussed some drawbacks and extensions to this approach. Since it is such a seminal work, we will discuss this approach in detail now. We will compare their approach and result in the works that preceded them. For the most part, we have already compared this approach with other similar approaches such as Li et al. \cite{li2018combinatorial} and Mittal et al. \cite{mittal2019learning}.

Khalil et al. \cite{khalil2017learning} proposes an end-to-end machine learning framework to solve COPs. They use deep graph embedding with reinforcement learning for automatically designing greedy heuristics. The learned policy behaves like a meta-algorithm that incrementally constructs a solution, with the action being determined by a graph embedding network over the current state of the solution. They exploit a common trait of real-world optimization problems | same optimization problem is solved again and again regularly, maintaining the same problem structure but differing in the data. Khalil et al. claim that other approaches proposed before theirs \cite{vinyals2015pointer, bello2016neural} are generic and do not reflect the combinatorial structure of graph problems effectively. Besides, these approaches require a huge number of instances to learn to generalize to new ones. \cite{bello2016neural} uses policy gradient for training which is not particularly sample-efficient \cite{khalil2017learning}.

They adopt a greedy meta-algorithm design, whereby a feasible solution is constructed by the successive addition of nodes based on the graph structure and is maintained to satisfy the problem's graph constraints. For their graph embedding network, they use structure2vec \cite{dai2016discriminative} to represent the policy in the greedy algorithm. This deep learning architecture factorizes the nodes in the graph, capturing the properties of a node in the context of its graph neighborhood. This allows the policy to discriminate among nodes based on their usefulness and generalizes to problem instances of different sizes. They assert that the prior works \cite{vinyals2015pointer, bello2016neural} use graph-agnostic sequence-to-sequence \cite{bahdanau2014neural} mapping that does not fully exploit graph structure.

They use fitted Q-learning to learn a greedy policy that is parametrized by the graph embedding network. Their framework allows the policy to optimize the objective function of the original problem instance directly. The main advantage of this approach is that it can deal with delayed rewards in a data-efficient way. Here, by delayed rewards, they mean the remaining increase in objective function value obtained by the greedy algorithm. In contrast, \cite{bello2016neural} which uses the policy gradient approach updates the model parameters only once for the whole solution. This process of graph embedding and greedy approach to add the best node to a partial solution is shown in Figure \ref{fig:graph-embedding}. It shows two iterations of the graph embedding to an instance of Minimum Vertex Cover.

The graph embedding network (structure2vec) defines the network architecture recursively according to an input graph structure. Node specific tags or features are aggregated recursively according to the graph's topology. After a few steps, the network will produce a new embedding for each node, taking into account both graph characteristics and long-range interactions between these node features. The parameters of the network are trained end-to-end using Q-learning. Off-policy reinforcement learning such as Q-learning can be more sample efficient than their policy gradient counterparts \cite{gu2016q}. They claim that this is because policy gradient methods require on-policy samples for the new policy obtained after each parameter update of the function approximator.

They train their model (S2V-DQN) on three problems: MVC, MAXCUT, and TSP. They compare their method with \cite{bello2016neural} (PN-AC) which does not make full use of graph structure. Their result showed that the PN-AC algorithm performs well on TSP as graph structure is not as important in a fully connected TSP problem. However, in MVC and MAXCUT, where graph information is crucial, their approach performs significantly better than PN-AC. In real-world instances, their model significantly outperforms all competing methods for MVC, MAXCUT, and TSP.

They made some important discoveries while examining the algorithms learned by S2V-DQN. For example, they noticed that S2V-DQN discovers an algorithm for MVC where nodes are selected to balance between their degrees and the connectivity of the remaining graph. Based on such results, they suggested that S2V-DQN may also be a good assistive tool for discovering new algorithms for lesser-known graph optimization problems.

The most recent work that exploits the structure of the problems defined over a graph is done by Barrett et al. \cite{barrett2019exploratory}. Barrett et al. claim that learning a policy that directly produces a single, optimal solution is often impractical because of the complexity of many combinatorial problems. Therefore, they propose an exploratory combinatorial optimization (ECO-DQN) in which an agent seeks to continuously improve the solution by learning to explore at test time. This is different than S2V-DQN and approaches following it which incrementally construct solutions one element at a time as described in \cite{khalil2017learning}.

\section{Challenges \& Current work}

Despite promising results shown by deep learning methodologies to solving COPs, there are still multiple challenges. A lot of these challenges are mostly because of the nature of the problems themselves, most of which are NP-hard. Bengio et al. \cite{bengio2018machine} discusses three main current challenges in this field.

First is finding a feasible solution. Bengio et al. \cite{bengio2018machine} claim that algorithms learned by deep learning methodologies do not give any guarantee in terms of optimality or feasibility. This is because the algorithm is learning a heuristic rather than learning the solution. Like any heuristic algorithm, there is always a risk of not reaching a close enough optimal solution or even feasible solutions. Bengio et al. suggest that because neural networks are so complex, they should be designed in such as way that it does not break the differentiability of COPs, to make sure that network can generate feasible solutions.

Second is scaling to larger problems. Even though some recent works are dealing with this issue, there has not been a promising result. If a model trained on instances up to a certain size COP is evaluated on larger instances, the challenge exists in terms of generalization \cite{bengio2018machine}. Previous works to solve TSP using deep learning, while attempting to solve larger instances observe degrading performance as size increases much beyond the sizes seen during training \cite{vinyals2015pointer, bello2016neural, khalil2017learning}. Training on larger sized instances can be computationally expensive.

Third, even though previous work has been successful in generalizing their approach to different well-studied COPs such as Knapsack and TSP \cite{bello2016neural}, it is still a challenging task to generalize such approaches on lesser-known COPs. Smith-Miles and Bowly \cite{smith2015generating} claims that often, a new algorithm is claimed to be superior by showing that it outperforms a state-of-the-art approach on a set of well-studied instances.

Finally, it is also hard to generate data for some COPs. For the application that we are well aware of, it is not very hard to collect data. However, if we are not very familiar with a COP and do not have much historical data, it is a challenge to train a policy to find a solution to such problems. Bengio et al. \cite{bengio2018machine} suggests generalizing such problems to a known family of instances. Even so, \cite{bengio2018machine} hints that it is hard to generate problems that capture the essence of real applications because these COPs are high dimensional, highly structured, and troublesome to visualize \cite{bengio2018machine}. Generating just the graphs itself is a complicated task.

\section{Discussion \& Conclusion}

Despite all these challenges, we believe that deep learning has a lot of potential in the domain of solving NP-hard COPs. Some recent works have shown promising results, especially because of the use of DRL. For example, Mittal et al. \cite{mittal2019learning} designed a deep reinforcement learning model called GCOMB and trained on billion-sized graphs, showing some promising results in dealing with the scalability issue. Barrett et al. \cite{barrett2019exploratory} designs an exploratory combinatorial optimization using deep Q-learning (ECO-DQN) to train an agent to continually improve the solution by learning to explore at test time. Most of the previous deep learning methodologies construct the solution subset incrementally, adding one element at a time \cite{khalil2017learning, li2018combinatorial}. However, the irreversible nature of these approaches prevents the agent from revising its earlier decisions. Given the complexity of the problem, it may be necessary to revisit the earlier decisions, which Barrett et al. \cite{barrett2019exploratory} effectively using ECO-DQN.

In this paper, we have shown some prominent work done to solve combinatorial optimization problems. We briefly discuss some exact and approximate algorithms to solve such a problem. However, because most of these problems are NP-hard, such exact and heuristic-based algorithms are not feasible for large-sized COPs. This led researchers to look into machine learning to let the algorithm figure out the essential heuristic to find the optimal solutions. We discussed some pioneer works based on Hopfield neural network \cite{hopfield1985neural} and Self-Organizing Maps based approaches \cite{kohonen1982self}. Even though the field looked promising, some major setbacks drove researchers away from this field from the turn of $21^{st}$ century.

With the advent of deep learning, researchers started using artificial neural networks to tackle this problem with promising results. Researchers have used both supervised and unsupervised learning approaches to find the optimal solution to COPs. Some of the prominent supervised learning approaches include Vinyals et al. \cite{vinyals2015pointer} and Li et al. \cite{khalil2017learning}. Vinyals et al. use Pointer Networks which is based on sequence-to-sequence learning with attention mechanism whereas Li et al. use Graph Convolutional Network (GCN). The field of unsupervised learning to solve COPs has been heavily influenced by DRL based approaches such as Deep Q-Learning (DQN). Some of the prominent works are Bello et al. \cite{bello2016neural} who extend Pointer Networks with the reinforcement learning algorithm and Dai et al. \cite{khalil2017learning} who uses graph embedding network with Q-learning. We also present some challenges in this field at the moment, which includes a lack of scalability, feasibility, and methods to generate training data. Finally, by discussing some promising recent work being done to tackle some of these problems using deep reinforcement learning, we recommend the research community to focus more on DRL methodologies to solving COPs.

\section\*{Acknowledgment}

We would like to thank Dr. Benjamin Mitchell for giving us helpful guidance and impetus to start this project. The authors would also like to thank Subash Nepal for crucial guidance in the process of writing and structuring the paper.