A Survey on Deep Learning-based Methodologies for Solving Combinatorial Optimization Problems

Rahul Thapa [1]

Abstract

Combinatorial Optimization Problems (COPs) involve finding an optimal solution within a finite set of possible solutions. Generally, the solution set is large and hence, is not practical to be solved through exhaustive search. Researchers have proposed many exact as well as approximate algorithms for solving COPs. However, such approaches are not very scalable because most of the COPs are NP-hard. In this survey, we present various deep learning methodologies used to solve discrete COPs and compare their network architectures and results. We begin with an introduction of two very popular neural network-based approaches called Hopfield Network and Self Organizing Feature Maps to solve COPs. As the core focus of our survey, we present extensive research done on solving COPs using deep learning within the past ten years. We discuss both supervised as well as unsupervised learning methodologies. Finally, we describe some current areas of research as well as challenges within this field.

1. Introduction

Combinatorial optimization is a topic that consists of finding an optimal object from a finite set of objects. The set of possible solutions is discrete and generally defined by a set of restrictions. It is often hard to solve such problems because the solution set is too large for exhaustive search. Combinatorial optimization has important real-world applications such as optimizing the supply chain, determining the optimal way to deliver packages, allocating jobs to people in the best possible way, developing the best airline network of spokes and destinations. Some of the most popular COPs are Knapsack problem, Travelling Salesman Problems (TSP), Maximum Vertex Cover, and Satisfiability (SAT). In this paper, we discuss deep learning methodologies used to solve varieties of these problems. However, a lot of these approaches are targeted towards solving one particular COP i.e. TSP. Therefore, we provide a brief introduction along with the exact and approximate methods used to solve it.

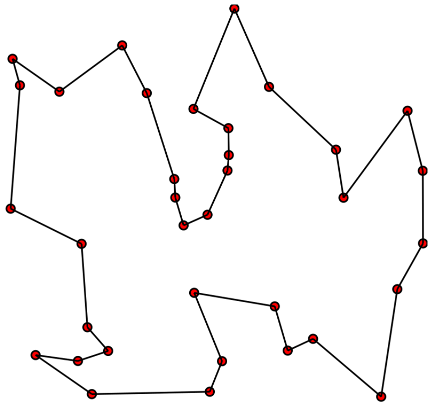

Figure 1. An optimal solution of a travelling salesman problem. The black line shows the shortest possible path that connects every red dot. Figure adapted from (wik, 2020)

The origin of TSP dates back to the 1800s, however, its mathematical forms were initiated in 1930 (Sarwar & Bhatti, 2012). In this problem, given a list of cities, one must find the shortest possible route that visits a set of cities all only once and returns to the original city, as shown in Figure 1. Some of the applications of TSP are delivering packages in a most cost-efficient way, drilling printed circuits, and picking order in a warehouse (Applegate et al., 2006; La Maire & Mladenov, 2012; Jünger et al., 1994). Even though the problem sounds fairly easy, it is not so easy to solve as it falls under NP-hard problems, even in two-dimensional Euclidean case (Padberg & Rinaldi, 1991; Bello et al., 2016).

Many exact and approximate algorithms have been proposed to solve TSP as surveyed in (Laporte, 1992). The exact dynamic programming algorithm for TSP has a time complexity of $\theta(2^n n^2)$. Some of the methods included in (Laporte, 1992) are the branch-and-bound method, integer

[1] Department of Computing Sciences, Villanova University, Villanova, PA. Correspondence to: Rahul Thapa <rthapa@villanova.edu>.

linear programming, and shortest spanning arborescence bound method. Because of exponential time complexity, it is not feasible to implement the exact algorithm for practical application. Christofides proposed an O(n^3) heuristic algorithm that involves computing a shortest spanning tree and a minimum cost perfect matching (Christofides, 1976; Bello et al., 2016; Laporte, 1992). Similarly, Lin-Kernighan-Helsgaun heuristic, which was extended from Lin-Kernighan heuristic, is a popular approximate search heuristic which can solve instances with hundreds of nodes (Bello et al., 2016; Helsgaun, 2000; Lin & Kernighan, 1973). This approach yields state-of-the-art results for symmetric TSP. Voudouris et al. (Voudouris & Tsang, 1999) used Guided Local Search (GLS) and Fast Local Search (FLS). GLS lied on top of local search heuristic and it was combined with the neighborhood reduction scheme of FLS.

It has been over four decades since researchers started using artificial neural networks to find an optimal solution to TSP (Smith, 1999). Hopfield and Tank designed one of the first neural networks to find an optimal path to TSP (Hopfield & Tank, 1985). They quantitatively demonstrated the computational power and speed of analog networks of neurons. However, (Wilson & Pawley, 1988) pointed out that this network was not very stable and reported unsatisfactory results even with a slight modification to the original algorithm. Many other researchers have expanded on this network (Aiyer et al., 1990). One of the most recent works on this network is done by (Sarwar & Bhatti, 2012) in which the authors compare the Hopfield Network with the heuristic algorithm.

Another main approach for solving TSP is by using Kohonen's Self-Organizing Feature Map (Kohonen, 1982). A vast majority of these approaches are based on elastic net method (Durbin & Willshaw, 1987) in which Kohenen's principles of self-organization are combined with the concept of the ”elastic band” containing circular rings of neurons (Durbin & Willshaw, 1987; Smith, 1999). Self-organizing nets have also been applied to solve other problems such as vehicle routing problem (Vakhutinsky & Golden, 1994). An overview of a wide range of its applications can be found in (Kohonen, 1990).

(Sarwar & Bhatti, 2012) showed that neural networks were still not able to yield comparable results to exact methods. Some of the reasons for a mediocre success of this field can be attributed to lack of powerful hardware resources and powerful neural network methodologies. This particular field remained stagnant for about a decade after 2000 (Bello et al., 2016). However, with the rejuvenation of deep learning (LeCun et al., 2015), researchers once again began applying artificial neural networks to solving COPs. This can be attributed to the advancement of some crucial deep learning methodologies such as sequence-to-sequence model (Sutskever et al., 2014), attention mechanism (Bahdanau et al., 2014), graph convolutional network (GCN) (Kipf & Welling, 2016), and deep reinforcement learning (DRL) (Mnih et al., 2015). One of the popular works is the Pointer Network introduced by (Vinyals et al., 2015). This network utilizes a recurrent neural network trained in a supervised manner to predict the sequence of visited cities. A simplified version of the Pointer Network for solving vehicle routing problems is presented in (Nazari et al., 2018). (Bello et al., 2016) significantly outperformed this supervised learning approach by using reinforcement learning and neural networks on a 2D Euclidean graphs. Similarly, (Khalil et al., 2017) solved optimization problems over graphs using graph embedding and deep Q-learning (DQN) algorithms (Mnih et al., 2015). (Li et al., 2018) presented a related work but differed in some key aspects. For example, (Li et al., 2018) used supervised learning instead of RL and GCN as their predictive model.

The remainder of this paper is organized into four sections. Section II discusses the use of neural networks to solve combinatorial optimization problems. This part specifically focuses on research done on this domain before 2000. Two main approaches we discuss under this section are Hopfield Neural Network and Self-Organizing Feature Maps based approaches. Section III focuses on the application of Deep Neural Networks to solving COPs. The majority of the paper concentrate on this section. Section IV discusses some current challenges in this field along with most recent works proposed to tackle those challenges. Finally, section V concludes the survey with a brief summary.

2. Neural Network based Approaches

Even though this paper is a survey on Deep Learning methodologies for solving COPs, it is crucial to understand how researchers started using artificial neural nets to solving COPs. In this section, we briefly discuss some famous neural network based approaches such as Hopfield Network and Self-Organizing Feature Maps. A lot of work done before the 2000s in this domain were influenced by or extended from these networks.

2.1. Hopfield Neural Network

Hopfield network was modeled initially to enable a content addressable memory (Smith, 1999) by John Hopfield in his 1982 seminal paper (Hopfield, 1982). He introduced a new way of modeling a system of neurons capable of performing computational tasks. In 1985, John Hopfield and David Tank extended the Hopfield Neural Network model to solve optimization problems (Hopfield & Tank, 1985). They selected weights and external inputs of their neural network that approximately represented the function to be

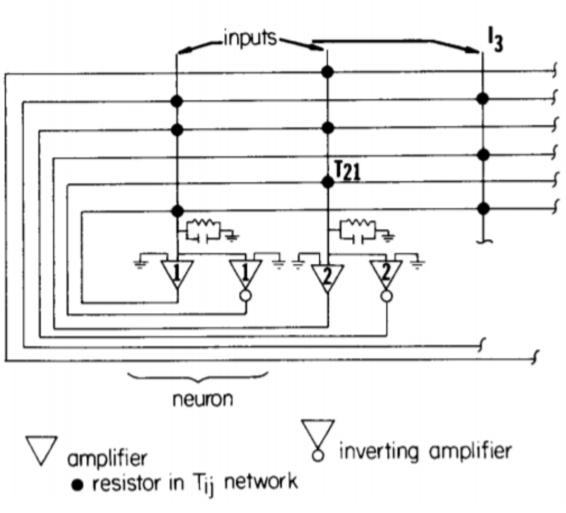

Figure 2. A simple analog representation of Hopfield and Tank Network that can solve optimization problems. The output of any neuron can potentially be connected to the input of any other neuron. Black circles represent resistive connections between inputs and outputs. Figure adapted from (Hopfield & Tank, 1985).

minimized and the desired states of the problem. For rapid and powerful solution techniques, they combined the analog nature of the neurons and the parallel processing of the updating procedure. The object function and energy function of the network were made equivalent which needed to be minimized i.e. the lowest energy state or stable states corresponds to the best path. The constraint of the optimization was included as penalty terms in the energy function. One pitfall of this approach was that these stable states might not correspond to optimal solutions, as the energy function might get stuck in one of many poor local minimums (Smith, 1999).

Hopfield and Tank's network was successfully applied to many optimization problems, however, their results for TSP gained most interest among the research community. They simulated a 10-city problem with 100 neurons (Hopfield, 1982). They chose the locations of the cities at random on the interior of a two-dimensional unit square. They claimed that they were able to find the appropriate general size of the parameters easily and that they used anecdotal exploration of parameter values to find a good operating point. However, they acknowledged that these points may not be optimal. They also acknowledged that the sensitivity of the results was due to the parameter settings of the network. For the 10 city problem, their model converged 16 out of 20 random starts to a valid tour. About 50% of the trials produced one of the two known shortest tours (Hopfield & Tank, 1985; Smith, 1999). However, their model did not do so great on larger sized inputs. For example, for a 30-city and 900 neurons problem, their results were fragmentary. They reported that the parameter choice appeared to be a more delicate issue for a larger network.

2.1.1. Limitations of Hopfield Neural Network

One of the most popular works that raised doubt on the reliability and validity of Hopfield and Tank's approach to solving COPs was (Wilson & Pawley, 1988) by Wilson and Pawley. While trying to imitate (Hopfield & Tank, 1985) approach to find the parameter settings for a larger problem size, they realized that the model did not converge to any valid solution. Even in a similar 10-city problem as proposed in (Hopfield & Tank, 1985), they were only able to converge 15 of the 100 random starts to valid tours which were not much better than random choices. Many of them got stuck into local minima. This result was much worse than what Hopfield and Tank reported (Wilson & Pawley, 1988). After trying some modifications such as change in parameter selection and increasing the value of the distance from a city to itself and not being able to get a significant improvement, they hypothesized that the root of the problem could be in the method, and not the operating point (Smith, 1999). There are some other works (Kamgar-Parsi, 1992), that have raised doubts on the stability of Hopfield and Tank's approach. (Lister, 1993) questioned the necessity of an energy function with a high degree of ultrametricity.

2.1.2. Hopfield Network Extensions

Despite various criticism of Hopfield Networks in solving COPs, various extension of this network had been proposed in solving TSP (den Bout et al., 1988; Gee & Prager, 1995; Yao et al., 1988; Lin et al., 1994). Extension of Hopfield Neural Networks involved two changes: First, modifying the energy function to eliminate the trade-off problems between valid and good solutions. Second, searching for ways to optimally select the penalty parameters while keeping the original energy function (Smith, 1999).

In (den Bout et al., 1988), the authors formulated a new TSP objective function which requires the setting of only one parameter to balance the importance of tour length and feasibility. This parameter is easy to determine from city distance. They also designed a new algorithm called Mean Field (MF) to handle the constraint that each city exists at one and only one tour position. This modification helped them find consistent valid and near-optimal solutions for a 30 city problem.

Another research around the same time (den Bout et al., 1988) used a modified penalty function to overcome the difficulty of consistently producing valid tours (Yao et al., 1988). With this modified network, they were able to generate 21/21 valid tours. They also formulated a fixed-parameter network whose connection matrix did not depend on problem data and could be fixed at hardware fabrication time for TSP.

The most comprehensive study of various modifications in the energy function of the original Hopfield and Tank's model was done by (Lin et al., 1994). They evaluated six different variations to the energy function. One of the popular approaches that they included was Aiyer energy function(Aiyer et al., 1990). Aiyer et al. utilized the eigenvalues of weights to introduce adjusting constantly into the energy function and corrected the landing error in the valid sequence. Out of all six approaches, Aiyer energy function was able to converge to a valid solution largest number of times. On 10-city topology benchmarks, the convergence percentage of Aiyer energy function for most benchmarks was above 90%.

2.2. Self-Organizing Feature Map

The self-organizing feature map (SOFM) is an unsupervised neural network that learns from the pattern of the input data and organizes itself based on that (Kohonen, 1982). It differs from Hopfield Network in that it does not contain preset weights and does not require supervision to guide the network. In such a network, first, the input patterns of arbitrary dimensionality are converted into one or two-dimensional array of neurons as shown in Figure 3. Each component in the input vector x is connected to each of the neurons, and the weight W_{ij} transmits the input x_j toward the j^{th} neuron of the feature map (Smith, 1999). Learning is then achieved via adaptation of the weights connecting the input patterns to the array of neurons. The learning process comprises two stages: a competitive stage and an adaptive stage. On a competitive stage, a neuron that is closest to the input data called winning neuron (Smith, 1999) is identified. On the adaptive stage, the weights of the winning neuron and its surrounding neurons are adjusted (Kohonen, 1982).

2.2.1. Limitations of Self-Organizing Neural Network

(Bishop et al., 1997) discussed some significant limitations of the Self-Organizing Map. They claimed that the algorithm was at a disadvantage because it was not derived by optimizing an objective function. Similarly, in such maps, the neighborhood-preservation was also not guaranteed. Furthermore, the choice of how the neighborhood function should shrink over time was arbitrary during training.

2.2.2. Self-Organizing Feature Map Extensions

One of the most popular ways the principles of self-organization maps were applied to solving TSP was by merging it with the elastic net method. The elastic net was first proposed by Durbin and Willshaw in (Durbin & Willshaw, 1987) to solve TSP. The algorithm starts with k points, greater than the number of cities, lying on a hypothetical ”elastic band”. When the nodes are moved, the ”elastic band” stretches in such a way that minimizes the energy function (Smith, 1999; Durbin & Willshaw, 1987). Sometimes, this method is considered analogous to the gradient descent of the energy function. Durbin and Willshaw were able to achieve state-of-the-art solutions (Lin & Kernighan, 1973) at the time of its publication. Their solution was 19% better than the Hopfield and Tank's best solution in terms of solution quality (Smith, 1999). However, the limitation of this approach is that it can be applied only to Euclidean TSPs and related problems.

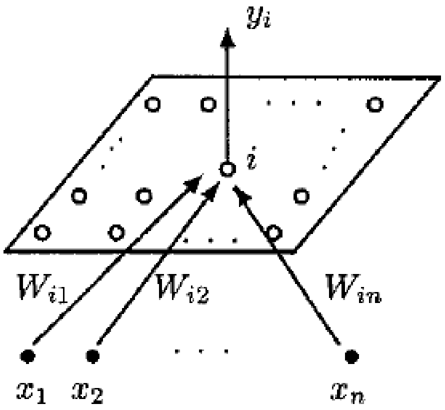

Figure 3. Mapping features of input vector x onto a two-dimensional array of neurons in Kohonen's SOFM. Adapted from (Smith, 1999).

A Self-Organization map approach to solving TSP, which is a very similar approach to the elastic method, was presented by (Fort, 1988) where they used a one-dimensional circular array of neurons and mapped it onto the TSP cities such that two neighboring points on the array were also neighbors in distance (Smith, 1999). There are some major differences, but the basic idea is the same, which even Fort agreed to in (Fort, 1988). However, the result that Fort obtained in a similar experiment done by (Durbin & Willshaw, 1987) was not as good.

Another group of researchers began experimenting by combining elastic net and SOFM to solve the TSP. One of the first approaches was by (Angeniol et al., 1988). They had a set of nodes joined together in a continuous ring. The nodes were free to move in a Euclidean plane iteratively until a complete tour was obtained when every city catches a node in the ring (Smith, 1999). They also included rules to create and delete nodes in their network. In a similar set of TSPs used by Hopfield and Tank (Hopfield & Tank, 1985), Angéniol et al. obtained a result that was comparable to the best result of (Durbin & Willshaw, 1987).

An alternative to the approach suggested by (Angeniol et al., 1988) is (Burke & Damany, 1992). Burke and Damany eliminated the need for the node creation and deletion rules. Their network is called the guilty net in which the number of nodes is equal to the number of cities. This network was inspired by the idea of conscience introduced in (DeSieno, 1988). This ensured the feasibility of the TSP tour as a minimum requirement, even if the network terminated prematurely (Smith, 1999). The results presented in (Burke & Damany, 1992) for TSP problems are poorer than the elastic net method. However, the authors noted that the number of iterations required was substantially lower. Burke later extended the guilty network called the vigilant net (Burke, 1994). This network was inspired by the vigilance of the adaptive resonance networks (Carpenter & Grossberg, 1987).

3. Deep Learning Methodologies

With the advent of deep learning, a significant impact has been made in various fields of machine learning such as object detection, speech recognition, and language translation (LeCun et al., 2015). The property of deep learning that makes it so influential is the deep neural networks that can automatically find compact low-dimensional representations or features of high-dimensional data. Through crafting inductive biases into neural network architectures, particularly that of hierarchical representations, machine learning practitioners have made effective progress in addressing the curse of dimensionality (Bengio et al., 2013). One field that was stagnant for over a decade, that has been rejuvenated by the advent of deep learning is solving COPs using artificial neural network. This section describes some popular supervised as well as unsupervised learning-based approaches in solving COPs.

3.1. Supervised Learning

One of the first attempts to tackle the COPs using deep learning methodology was proposed by (Vinyals et al., 2015). They introduced the concept of pointer network, which was inspired by sequence-to-sequence models (Sutskever et al., 2014). The Sequence-to-sequence learning method requires the size of the output dictionary to be fixed a priori (Vinyals et al., 2015). However, in COPs, the output sequence length is determined by the source sequence. Therefore, Vinyals et al. had to modify the sequence-to-sequence model to make it invariant to the length of the encoder sequence. They modified the attention mechanism of (Bahdanau et al., 2014) to create pointers to input elements as shown in Figure 4.

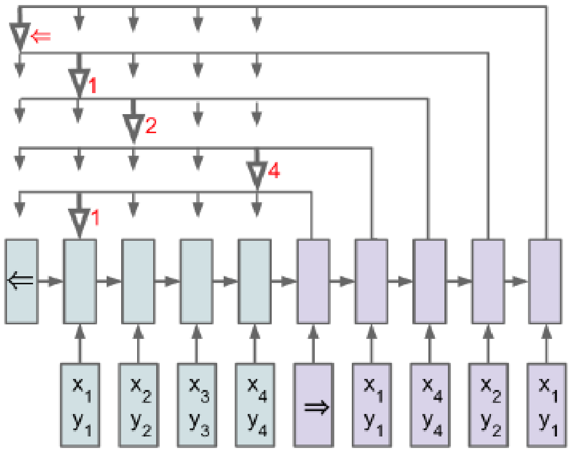

Figure 4. Pointer Network - The encoder RNN network, represented by blue boxes, covers the input sequence to a code that is then fed to the generating network represented by purple boxes. The output of generating network is a vector that modulates a content-based attention mechanism over input. Figure adapted from (Vinyals et al., 2015).

Given a training pair (P, C^P), the traditional sequence-to-sequence model computes the conditional probability $p(C^p|P;\theta)$ as

$$p(C^P|P;\theta) = \prod_{i=1}^{m(P)} p(C_i|C_1,...,C_{i-1},P;\theta) \qquad (1)$$

Here, $P = \{P_1,...,P_n\}$ is a sequence of n vectors and $C^P = \{C_1,...,C_{m(P)}\}$ is a sequence of $m(P)$ indices. An RNN with parameters θ is used to estimated the terms of the probability. This model uses softmax distribution over a fixed sized output dictionary to compute $p(C_i|C_1,...C_{i-1},P)$ in Equation 1 which can not be used for COPs because the size of the output dictionary is equal to the length of the input sequence. Therefore, Vinyals et al. used attention mechanism to model $p(C_i|C_1,...C_{i-1},P)$ as follows:

$$\begin{aligned} u^i_j &= v^T \text{tanh}(W_1 e_j + W_2 d_i) \quad j \in (1,...,n) \\ p(C_i|C_1,...,C_{i-1},P) &= \text{softmax}(u^i) \end{aligned}$$

Here, v, W_1 and W_2 are learnable parameters of the output model. Softmax normalizes the vector u^i to be the attention mask over the inputs i.e. pointers to the input elements. Their approach was specifically designed for problems whose outputs are discrete and correspond to the positions in the input.

Vinyals et al. used the same model architecture to perform experiments on 3 problem sets: Convex Hall (Graham, 1972; Jarvis, 1973), Delaunay Triangulation (Rebay, 1993), and TSP. All their models used a single layer LSTM trained with stochastic gradient descent. They used the convex hull as a baseline to understand the deficiencies of the traditional sequence-to-sequence model and build their pointer network. They checked the performance of their model for a convex hull experiment based on accuracy and area covered of the true convex hull. They noted that the area coverage achieved with the Ptr-Net was close to 100%. They observed that the order in which the inputs are presented matter during inference as it affected the performance when there was not attention mechanism. However, the attention mechanism allowed their decoder to look at the whole input at any time and hence, overcome this problem. They observed that the attention was pointing at the correct answer on the input side. The major contribution of this experiment was showing that their model had the key advantage of being inherently variable length. A single model trained on a variety of lengths performed quite well on all lengths it was trained on. They even noticed that their model even extrapolated to lengths that it had never seen before. They noted their result for n=500 which gave them a satisfactory result, while other models with LSTM or LSTM with attention could not even be used for any given input size that they had not been trained on.

For the experiment on TSP, they considered planner symmetric TSPs. Since their algorithm for Ptr-Net implemented an $O(n^2)$ algorithm, initially, they were not very clear if their model would be able to learn solely from data. To make sure that the model would not output invalid tours, they used beam search procedure for input size n > 20. As it is feasible to generate exact solutions for relatively small-sized TSP, they used three different algorithms A1, A2, and A3 to generate approximate solutions for large-sized TSP. They trained their Ptr-Net model on optimal data with 5 to 20 cities to see if they could generalize for larger values of n. They noted that their model performed pretty good for $n = 25$ and reasonably good for $n = 30$. However, their model broke for values higher than 40.

(Li et al., 2018) is another prominent work to solve COPs that harnesses graph property of some COPs such as TSP. They used GCN to predict the likelihood, for each vertex, of whether this vertex was part of the optimal solution. Li et al. extended from (Khalil et al., 2017), which was one of the first approaches to solving COPs in terms of a graph. We will discuss this in the next subsection. The approach outlined in Li et al. differs from the one proposed by Khalil et al. in 3 key respects.

First, they did not use reinforcement learning because they considered it a particularly challenging method. Rather, they used supervised learning and showed that they could achieve strong performance and generalization and could benefit from well-understood and reliable solvers. Second, they used GCN as opposed to Deep Q-learning (DQN) network with graph embedding in (Khalil et al., 2017). Finally, they designed and trained the network to synthesize a diverse set of solutions at once which was key to their approach and allowed them to rapidly explore the solution space.

Li et al. focused on four canonical NP-hard problems: Maximal Independent Set (MIS), Minimum Vertex Cover (MVC), Maximal Clique (MC), and Satisfiability (SAT). One key property of these problems is that they can be reduced to each other (Karp, 1972) in polynomial time. For example, MVC, MC, and SAT problems can all be represented as instances of the MIS problem. Therefore, this paper primarily focused on the MIS problem, although, the basic structure of the approach was general.

In Figure 5, we can see that there were 4 major parts of their network architecture: graph reduction, GCN, tree search, and local search. First was the graph reduction step in which they rapidly reduced a graph into a smaller one while preserving the size of the optimal MIS. This accelerated computation by focusing computation on the complex part of the graph (Li et al., 2018).

The second part was the graph convolutional network (Karp, 1972) f which can perform dense prediction over a graph with pairwise edges. The f generated probability maps from the reduced graph, which encoded the likelihood of each vertex being in the optimal solution. The reason that they did not generate a binary vector, 0 and 1, but a real-valued vector in $[0, 1]^N$ was because simply rounding the real values to 0 or 1 might violate the independence constraints. The probability maps were used to iteratively label the vertices until all vertices were labeled. In this setup, f was used as a heuristic function for a greedy search algorithm for MIS. Given a graph G, the search algorithm labeled a batch of vertices with 1 or 0 recursively. First, they sorted all the vertices in descending order based on $f(G)$ and iterated over the sorted list in order and labeled each vertex as 1 and its neighbor as 0. This process stops when the next vertex in the sorted list is already labeled as 0. They removed all the labeled vertices and the incident edges from G and obtained a residual graph. This residual graph was fed as an input to f which gave a new likelihood map. This process was repeated iteratively.

The third part was the tree search. Li et. al. noted that just graph reduction and GCN could confuse the network when there were multiple optimal solutions for the same graph. In other words, the solution space is multimodal and many different modes may be encountered during training (Li et al., 2018). To enable the network to differentiate be-

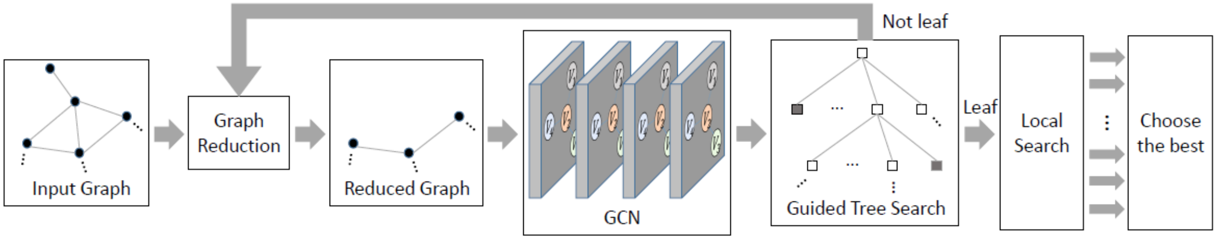

Figure 5. The Deep learning architecture of the algorithm defined in Li et. al. Graph Convolutional Network is used to generate the multiple probability maps which are used to iteratively label the vertices until all vertices are labeled. Labeling is generated by the guided tree search and refined by rapid local search. Figure adapted from (Li et al., 2018).

tween different modes, they extended the structure of f to generate multiple probability maps. To generate diverse high-quality probability maps, they used the hindsight loss (Li et al., 2018). Producing multiple diverse probability maps also helped them to explore multiple solutions with each run of f. The basic idea of a tree search algorithm is that one maintains a queue of incomplete solutions and randomly chooses one of them to expand in each step. When they expanded an incomplete solution, they used M probability maps to spawn M new more complete solutions, which were added to the queue. This process is akin to the breadth-first search algorithm, rather than depth-for search, which they noted, did not yield diverse solutions (Li et al., 2018). To this end, the expanded tree nodes were kept in a queue and one was selected at random in each iteration for expansion.

The last step was a local search which helped them refine the candidate solutions produced by tree search. They specifically used a 2-improvement local search algorithm (Andrade et al., 2012; Feo et al., 1994).

They ran the experiments on SAT Competition 2017 (Balyo et al.), BUAA-MC (Xu et al., 2007), SNAP Social Network (Leskovec & Krevl), and Citation networks (Sen et al., 2008). They mainly compared their result with the approach outlined by (Khalil et al., 2017), which we discuss in detail in the next section. From now, approach described in (Khalil et al., 2017) is referred to as S2V-DQN, following their terminology. Li et al. also listed the performance of a classic greedy heuristic, referred to as Classic (Papadimitriou & Steiglitz, 1998). For both S2V-DQN and Classic, they also analyzed an enhanced version by including graph reduction and local search. They referred to those networks as S2V-DQN+GR+LS and Classic+GR+LS.

On the SAT dataset, Li et al.'s model solved all 1000 instances whereas S2V-DQN and Classic could not solve a single problem instance. They hypothesized that the reinforcement learning procedure might not have discovered fully satisfying solutions during training in S2V-DQN. It discovered solutions that were close but struggled to get to the optimum, consistent with the result reported in (Khalil et al., 2017). However, with the enhancement mentioned above, S2V-DQN and Classic models solved 89 and 79 instances, which was still not comparable to their model. On the BUAA-MC dataset, S2V-DQN, even with enhancement, could not find any solution, whereas their approach solved 62.5% of the instances. Moreover, their model was only trained on synthetic SAT graphs from a different dataset which showed that their approach generalized well across datasets and problem types (Li et al., 2018).

Finally, they also reported results on large-scale real-world graphs. Their approach outperformed all other baselines on all graphs. They also found out that S2V-DQN did not perform as well as Classic when the graph size was larger than 10,000 vertices. They hypothesized that S2V-DQN did not generalize well large graphs (Li et al., 2018). This approach showed that their approach also generalized from synthetic graphs to real graphs and from graphs with roughly 1,000 vertices to graph with 100,000 nodes and more than 10 million edges. They claimed that their approach may be discovering the universal motifs present in graphs across datasets and scales.

(Mittal et al., 2019) extended from (Khalil et al., 2017) and (Li et al., 2018) in order to adequately address the aspect of scalability to billion-sized graphs. They claimed to be the first learning-based approach that could be applied to massive real networks without compromising on quality. They proposed a deep reinforcement learning framework called GCOMB to learn algorithms that could solve combinatorial optimization problems over graphs at scale. They essentially addressed two main drawbacks of (Khalil et al., 2017) and (Li et al., 2018). First, was the scalability issue. They noted that the primary focus of Li et al. and Khalil et al. were to obtain a solution as optimal as possible (Mittal et al., 2019). Efficiency studies were limited to networks containing only hundreds of thousands of nodes. However, real-life networks contain millions of nodes and billions of edges. They claimed that the non-scalability of the framework proposed in (Li et al., 2018) aroused due to their design and framework. As described earlier, following the tree-search, GCN was repeated on a reduced graph which continued iteratively. The authors claimed that there were two bottlenecks in this process. First, tree-search was time-consuming on large graphs and second GCN was called repeatedly on iteratively reduced graphs, which did not scale well for large real-life networks.

The second drawback was the generalizability of real-life combinatorial problems. They claimed that the non-scalability of Li et al. arose from its non-generalizability to a larger class of COPs. (Li et al., 2018), as discussed earlier, proposed a learning-based heuristic for the Maximal Independent Set Problem (MIS). When the combinatorial problem was not MIS, Li et al. pointed out that one should map the problem to MIS and then apply the algorithm. Even though this approach worked well for varieties of problems, for problems that could not be mapped to MIS, the efficacy was compromised as shown by (Mittal et al., 2019).

To address these issues, (Mittal et al., 2019) proposed an end-to-end prediction framework called GCOMB. GCOMB first generated node embeddings through GCN which encoded the effect of a node on the budget-constrained solution set. After that, these embeddings were fed into a neural network to learn a Q-function and predict the solution set. (Khalil et al., 2017) also used reinforcement learning. However, unlike them, Mittal et al. used supervised learning. They claimed that because of using supervised learning, they were able to make higher quality predictions (Mittal et al., 2019). The difference from (Li et al., 2018) was that Mittal et al. used deep reinforcement learning instead of tree-search to learn and predict the combinatorial nature of the problem.

They applied their model on Influence Maximization (IM) and showed that they performed better than Li et al. Moreover, they also improved the state-of-the-art algorithm built specifically for IM. However, GCOMB was about 100 times faster. They also benchmarked GCOMB on billion sized networks which GCOMB finished in less than a minute. The quality produced by GCOMB was on par with GCN-TreeSearch while being way faster.

3.2. Unsupervised Learning

Most successful machine learning techniques fall under supervised learning where a mapping from training labeled inputs to outputs is learned. However, (Bello et al., 2016) 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 use 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 was by (Bello et al., 2016). They presented a framework to tackle a popular COP called TSP using neural networks and reinforcement learning, specifically policy gradients (Williams, 1992). (Bello et al., 2016) followed the approach of Vinyals et al. to generalize beyond a pre-specified graph size. (Vinyals et al., 2015) made use of a set of non-parametric softmax modules, resembling the attention mechanism from (Bahdanau et al., 2014). Vinyals et al. was further inspired by (Sutskever et al., 2014) as described in the supervised learning section above. We do not go into the detail of sequence-to-sequence learning or pointer network again in this section, however, we outline some crucial differences between these two approaches.

(Vinyals et al., 2015) proposed training a pointer network using a supervised loss function comprising of conditional log-likelihood (Vinyals et al., 2015). 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., 2016) claimed 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. proposed model-free policy-based reinforcement learning (RL) to optimize the parameters of a pointer network because they believed 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 was the expected tour length. They used a well-known REINFORCE algorithm (Williams, 1992) to formulate the gradient of their network. They also introduced an auxiliary network called an actor-critic (Joel et al., 2002) which they used as a parametric baseline to estimate the expected tour length found by their policy. They believed that it typically improved learning. The critic was 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 comprised 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., 2016).

Bello et al. considered two search strategies called sampling and active search. Under sampling, they simply sampled multiple candidate tours from their stochastic policy and selected the shortest one. Under active search, they refined the parameters of the stochastic policy during inference to minimize the tour length on a single test input. They claimed that this approach proved to be competitive when starting from a trained model, however, it also produced satisfying solutions when they started from an untrained model. They refered to these approaches as RL pretraining-Active Search and Active Search respectively.

They tested their model's performance with 3 benchmark tasks: Euclidean TSP20, 50, and 100. They compared their model against 3 different baselines: Christofides (Christofides, 1976), vehicle routing solver OR-Tools (goo), and optimality (Applegate et al., 2006). They noted that training with RL significantly improved over supervised learning (Vinyals et al., 2015). All their methods comfortably surpassed Christofides' heuristic (Christofides, 1976). They also found out that both of their greedy approaches were time-efficient and just a little worse than optimality. Besides, they found out that searching at inference time proved crucial to get closer to optimality. However, that came at the cost of running times. They concluded that RL pretraining-Sampling and RL pretraining-Active Search were the most competitive neural combinatorial optimization methods as they recovered the optimal solution in a significant number of their test cases. For a small solution space, they reported that RL pretraining-Sampling outperformed RL pretraining-Active Search in both quality and speed. However, for larger solution space, RL pretraining-Active Search proved to be 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 yielded solutions that were just 1% less than optimal on average and Active Search solved all instances of KnapSack problems of size 50, 100, and 200 optimally.

(Nazari et al., 2018) generalized the framework presented in (Bello et al., 2016) to include a wider range of combinatorial optimization problems such as the Vehicle Routing Problem (VRP). They reported that since the approach proposed in Bello et al. assume that the system was 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 (Nazari et al., 2018). Their model could efficiently handle both static and dynamic elements of the system. They argued that the RNN encoder used in (Bello et al., 2016) adds an extra complication to the encoder. Since there is no meaningful order to the input set in the combinatorial optimization problem, they claimed that the RNN encoder was actually unnecessary. By omitting it, the approach can be made much more general. Therefore, in their model, they simply left out the encoder RNN and directly used the embedded inputs instead of the RNN hidden states. Their policy model consisted of an RNN decoder coupled with attention mechanism which formed a distribution over the feasible destinations that could be chosen at the next decision point.

One of the first approaches to solve the COPs using unsupervised deep learning over the graph was proposed by (Khalil et al., 2017). At this point, we have already 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 discuss this approach in detail now. We 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., 2018) and (Mittal et al., 2019).

(Khalil et al., 2017) proposed an end-to-end machine learning framework to solve COPs. They used deep graph embedding with reinforcement learning for automatically designing greedy heuristics. The learned policy behaved like a meta-algorithm that incrementally constructed a solution, with the action being determined by a graph embedding network over the current state of the solution. They exploited 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. claimed that other approaches proposed before theirs (Vinyals et al., 2015; Bello et al., 2016) were generic and did not reflect the combinatorial structure of graph problems effectively. Besides, these approaches required a huge number of instances to learn to generalize to new ones. (Bello et al., 2016) used policy gradient for training which is not particularly sample-efficient (Khalil et al., 2017).

They adopted a greedy meta-algorithm design, whereby a feasible solution was constructed by the successive addition of nodes based on the graph structure and was maintained to satisfy the problem's graph constraints. For their graph embedding network, they used structure2vec (Dai et al., 2016) 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 asserted that the prior works (Vinyals et al.,

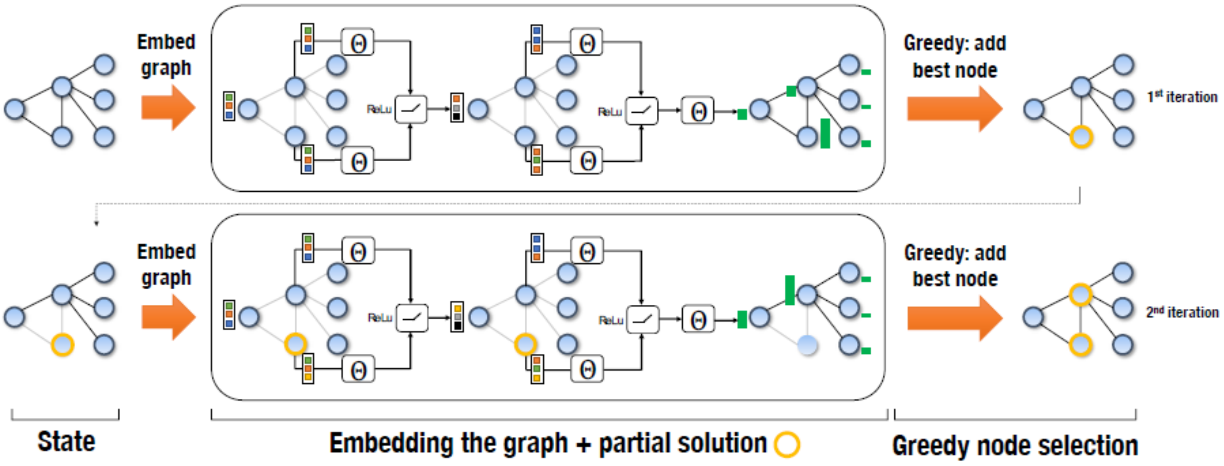

Figure 6. Illustration of the graph embedding and greedy node selection framework proposed in Khalil et al. as applied to an instance of Minimum Vertex Cover. Figure adapted from (Khalil et al., 2017)

2015; Bello et al., 2016) used graph-agnostic sequence-to-sequence (Bahdanau et al., 2014) mapping that did not fully exploit graph structure.

They used fitted Q-learning to learn a greedy policy that was parametrized by the graph embedding network. Their framework allowed 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 meant the remaining increase in objective function value obtained by the greedy algorithm. In contrast, (Bello et al., 2016) which used the policy gradient approach updated 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 6. It shows two iterations of the graph embedding to an instance of Minimum Vertex Cover.

The graph embedding network (structure2vec) defined the network architecture recursively according to an input graph structure. Node specific tags or features were aggregated recursively according to the graph's topology. After a few steps, the network produced 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 were 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 (Gu et al., 2016). They claimed that this was because policy gradient methods require on-policy samples for the new policy obtained after each parameter update of the function approximator.

They trained their model (S2V-DQN) on three problems: MVC, MAXCUT, and TSP. They compared their method with (Bello et al., 2016) (PN-AC) which did not make full use of graph structure. Their result showed that the PN-AC algorithm performed 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 performed significantly better than PN-AC. In real-world instances, their model significantly outperformed 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 discovered an algorithm for MVC where nodes were 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. (Barrett et al., 2019). 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 (Khalil et al., 2017).

4. Challenges

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., 2018) discuss some of these persistent challenges in this field.

First, finding a feasible solution to COPs is not so trivial. (Bengio et al., 2018) 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 they do not break the differentiability of COPs, to make sure that network can generate feasible solutions.

Second, scaling the proposed methodologies to solve larger instances of COPs is still a challenging task. 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 (Bengio et al., 2018). 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 (Vinyals et al., 2015; Bello et al., 2016; Khalil et al., 2017). Training on larger sized instances can also be computationally expensive.

Finally, even though previous work has been successful in generalizing their approach to different well-studied COPs such as Knapsack and TSP (Bello et al., 2016), it is still a challenging task to generalize such approaches on lesser-known COPs. Smith-Miles and Bowly (Smith-Miles & Bowly, 2015) asserted 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.

5. Discussion and Summary

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., 2019) 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. (Barrett et al., 2019) designed an exploratory combinatorial optimization using deep Q-learning (ECO-DQN) to train an agent and continually improved the solution by learning to explore at test time. Most of the previous deep learning methodologies constructed the solution subset incrementally, adding one element at a time (Khalil et al., 2017; Li et al., 2018). 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 is addressed using ECO-DQN (Barrett et al., 2019).

In this paper, we surveyed some prominent work proposed to solve COPs using deep learning methodologies. We briefly discussed 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 (Hopfield & Tank, 1985) and Self-Organizing Maps based approaches (Kohonen, 1982). Even though the field looked promising, some major setbacks drove researchers away from this field after the turn of 21^{st} century.

With the advent of deep learning, researchers started using artificial neural networks once again 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., 2015) and (Khalil et al., 2017). Vinyals et al. used Pointer Networks which is based on sequence-to-sequence learning with attention mechanism whereas Li et al. used 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., 2016) who extended Pointer Networks with the policy gradient algorithm and (Khalil et al., 2017) who used graph embedding network with Q-learning. We also presented some challenges in this field at the moment, which includes a lack of scalability, feasibility, and methods to generate training data. Finally, in light of 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.

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.

References

Vehicle routing — or-tools — google developers. URL `https://developers.google.com/optimization/routing`.

Travelling salesman problem, Apr 2020. URL `https://en.wikipedia.org/wiki/Travelling_salesman_problem`.

Aiyer, Sreeram VB, Niranjan, Mahesan, and Fallside, Frank. A theoretical investigation into the performance of the hopfield model. *IEEE transactions on neural networks*, 1(2):204–215, 1990.

Andrade, Diogo V, Resende, Mauricio GC, and Werneck, Renato F. Fast local search for the maximum independent set problem. *Journal of Heuristics*, 18(4):525–547, 2012.

Angeniol, Bernard, Vaubois, Gael De La Croix, and Le Texier, Jean-Yves. Self-organizing feature maps and the travelling salesman problem. *Neural Networks*, 1(4): 289–293, 1988.

Applegate, David L, Bixby, Robert E, Chvatal, Vasek, and Cook, William J. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Balyo, Tomas, JH Heule, Marijin, and Matti, Jarvisalo. Sat competition 2017. URL `https://baldur.iti.kit.edu/sat-competition-2017/`.

Barrett, Thomas D, Clements, William R, Foerster, Jakob N, and Lvovsky, Alex I. Exploratory combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1909.04063*, 2019.

Bello, Irwan, Pham, Hieu, Le, Quoc V, Norouzi, Mohammad, and Bengio, Samy. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Bengio, Yoshua, Lodi, Andrea, and Prouvost, Antoine. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*, 2018.

Bishop, Christopher M, Svensén, Markus, and Williams, Christopher KI. Gtm: A principled alternative to the self-organizing map. In *Advances in neural information processing systems*, pp. 354–360, 1997.

Burke, Laura I. Neural methods for the traveling salesman problem: insights from operations research. *Neural Networks*, 7(4):681–690, 1994.

Burke, Laura I and Damany, Poulomi. The guilty net for the traveling salesman problem. *Computers & Operations Research*, 19(3-4):255–265, 1992.

Carpenter, Gail A and Grossberg, Stephen. Art 2: Self-organization of stable category recognition codes for analog input patterns. *Applied optics*, 26(23):4919–4930, 1987.

Christofides, Nicos. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

Dai, Hanjun, Dai, Bo, and Song, Le. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pp. 2702–2711, 2016.

den Bout, Van et al. A traveling salesman objective function that works. In *IEEE 1988 International Conference on Neural Networks*, pp. 299–303. IEEE, 1988.

DeSieno, Duane. Adding a conscience to competitive learning. In *IEEE international conference on neural networks*, volume 1, pp. 117–124. Institute of Electrical and Electronics Engineers New York, 1988.

Durbin, Richard and Willshaw, David. An analogue approach to the travelling salesman problem using an elastic net method. *Nature*, 326(6114):689–691, 1987.

Feo, Thomas A, Resende, Mauricio GC, and Smith, Stuart H. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, 42 (5):860–878, 1994.

Fort, JC. Solving a combinatorial problem via self-organizing process: An application of the kohonen algorithm to the traveling salesman problem. *Biological cybernetics*, 59(1):33–40, 1988.

Gee, Andrew H and Prager, Richard W. Limitations of neural networks for solving traveling salesman problems. *IEEE Transactions on Neural Networks*, 6(1):280–282, 1995.

Graham, Ronald L. An efficient algorithm for determining the convex hull of a finite planar set. *Info. Pro. Lett.*, 1: 132–133, 1972.

Gu, Shixiang, Lillicrap, Timothy, Ghahramani, Zoubin, Turner, Richard E, and Levine, Sergey. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

Helsgaun, Keld. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

Hopfield, John J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8): 2554–2558, 1982.

Hopfield, John J and Tank, David W. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

Jarvis, Ray A. On the identification of the convex hull of a finite set of points in the plane. *Information processing letters*, 2(1):18–21, 1973.

Joel, Daphna, Niv, Yael, and Ruppin, Eytan. Actor–critic models of the basal ganglia: New anatomical and computational perspectives. *Neural networks*, 15(4-6):535–547, 2002.

Jünger, Michael, Thienel, Stefan, and Reinelt, Gerhard. Provably good solutions for the traveling salesman problem. *Zeitschrift für Operations Research*, 40(2):183–217, 1994.

Kamgar-Parsi, Behzad. Dynamical stability and parameter selection in neural optimization. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pp. 566–571. IEEE, 1992.

Karp, Richard M. Reducibility among combinatorial problems. In *Complexity of computer computations*, pp. 85–103. Springer, 1972.

Khalil, Elias, Dai, Hanjun, Zhang, Yuyu, Dilkina, Bistra, and Song, Le. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6348–6358, 2017.

Kipf, Thomas N and Welling, Max. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kohonen, Teuvo. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43 (1):59–69, 1982.

Kohonen, Teuvo. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

La Maire, Bert FJ and Mladenov, Valeri M. Comparison of neural networks for solving the travelling salesman problem. In *11th Symposium on Neural Network Applications in Electrical Engineering*, pp. 21–24. IEEE, 2012.

Laporte, Gilbert. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *nature*, 521(7553):436–444, 2015.

Leskovec, Jure and Krevl, Andrej. Snap datasets: Stanford large network dataset collection. URL `http://snap.stanford.edu/data/`.

Li, Zhuwen, Chen, Qifeng, and Koltun, Vladlen. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pp. 539–548, 2018.

Lin, Shen and Kernighan, Brian W. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

Lin, Wei, Delgado-Frias, JG, Pechanek, Gerald G, and Vassiliadis, Stamatis. Impact of energy function on a neural network model for optimization problems. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 7, pp. 4518–4523. IEEE, 1994.

Lister, Raymond. Annealing networks and fractal landscapes. In *IEEE International Conference on Neural Networks*, pp. 257–262. IEEE, 1993.

Mittal, Akash, Dhawan, Anuj, Manchanda, Sahil, Medya, Sourav, Ranu, Sayan, and Singh, Ambuj. Learning heuristics over large graphs via deep reinforcement learning. *arXiv preprint arXiv:1903.03332*, 2019.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Nazari, Mohammadreza, Oroojlooy, Afshin, Snyder, Lawrence, and Takác, Martin. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pp. 9839–9849, 2018.

Padberg, Manfred and Rinaldi, Giovanni. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

Papadimitriou, Christos H and Steiglitz, Kenneth. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

Rebay, Stefano. Efficient unstructured mesh generation by means of delaunay triangulation and bowyer-watson algorithm. *Journal of computational physics*, 106(1):125–138, 1993.

Sarwar, Farah and Bhatti, Abdul Aziz. Critical analysis of hopfield's neural network model and heuristic algorithm for shortest path computation for routing in computer networks. In *Proceedings of 2012 9th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*, pp. 115–119. IEEE, 2012.

Sen, Prithviraj, Namata, Galileo, Bilgic, Mustafa, Getoor, Lise, Galligher, Brian, and Eliassi-Rad, Tina. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Smith, Kate A. Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34, 1999.

Smith-Miles, Kate and Bowly, Simon. Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113, 2015.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Vakhutinsky, Andrew I and Golden, BL. Solving vehicle routing problems using elastic nets. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 7, pp. 4535–4540. IEEE, 1994.

Vinyals, Oriol, Fortunato, Meire, and Jaitly, Navdeep. Pointer networks. In *Advances in neural information processing systems*, pp. 2692–2700, 2015.

Voudouris, Christos and Tsang, Edward. Guided local search and its application to the traveling salesman problem. *European journal of operational research*, 113(2): 469–499, 1999.

Williams, Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Wilson, GV and Pawley, GS. On the stability of the travelling salesman problem algorithm of hopfield and tank. *Biological Cybernetics*, 58(1):63–70, 1988.

Xu, Ke, Boussemart, Frédéric, Hemery, Fred, and Lecoutre, Christophe. Random constraint satisfaction: Easy generation of hard (satisfiable) instances. *Artificial intelligence*, 171(8-9):514–534, 2007.

Yao, Wang et al. Alternative networks for solving the traveling salesman problem and the list-matching problem. In *IEEE 1988 International Conference on Neural Networks*, pp. 333–340. IEEE, 1988.