POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Master's Degree Thesis

Beyond the TSP: On the transfer of existing deep learning architectures to other combinatorial problems

Supervisors

Prof. Michela MEO

Dr. Zied BEN HOUIDI

Prof. Dario ROSSI

Candidate

Matteo BOFFA

October 2021

Abstract

Solving combinatorial problems represents an everyday challenge for vital tasks such as logistics, resource allocation and optimization. In those contexts, machine learning, with its Reinforcement Learning and Graph Neural Networks branches, promises automation: give the machine input data and an evaluator to guide its learning process and, at the end of training, it will return feasible and high-quality outputs. Unfortunately, exploiting this potential is still not an easy matter: in practice, the research so far has only focused on tailored solutions, small environments or, with changing fortunes, on specific sub-classes of combinatorial problems such as the TSP and its variants.

The aim of this research is therefore assessing how and whether the latest deep learning findings in those fields also transfer to other practical problems. To this end, we set out to systematically transfer these architectures from the TSP problem to that of radio resource allocation in wireless networks.

Our results suggest that existing architectures are still incapable of capturing the structural aspects of combinatorial problems. Ultimately, this represents a sever setback for generalization purposes. We also demonstrate that reinforcing the structural representation of problems with a novel technique named Distance Encoding is a promising direction for obtaining multi-purpose autonomous solvers. In fact, the introduction of encoded distances as novel features improves the solutions when no other structural information is provided.

Acknowledgements

I would like to thank my supervisor, Prof. Michela Meo, for her invaluable assistance and patience throughout the development of this thesis. My sincere thanks also goes to my Huawei mentors, Prof. Dario Rossi, Dr. Zied Ben Houidi, Dr. Jonatan Krolikowski and Dr. Ovidiu Iacoboaiea. Your time with me and the way you made me feel so welcomed and included in the team is something I will remember forever: your suggestions will always be a treasure to me.

A special thank you also goes out to my colleagues Alessandro Baldo and Irene Benedetto, who have provided valuable insights and opinions during the process.

Last but not the least I would also like to thank all of my friends, family members for encouraging and supporting me whenever I needed them.

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

ML
machine learning

DL
deep learning

TSP
travel salesman problem

VRP
vehicle routing problem

SDVRP
split delivery vehicle routing problem

RL
reinforcement learning

SL
supervised learning

UL
unsupervised learning

AP
access point

STA

station

GNN

graph neural network

GAT

graph attention network

GGCN

gated graph convolutional network

GCN

graph convolutional network

CNN

convolutional neural network

MHA

multi head attention

DE

distance encoder

GCP

graph coloring problem

RAP

resource allocation problem

WLAN

wireless local area network

NN

neural network

NMT

neural machine translation

sequence-to-sequence

sequence to sequence

RNN

recursive neural network

Chapter 1

Introduction

1.1 Combinatorial problems in the real world

The importance of *combinatorial problems* in the real world is witnessed by the diversified fields of knowledge in which those are faced. Well known examples include finding the shortest path on a graph [1], crucial objective on logistic applications such as routing objects on a given topology (i.e. a commercial fleet between cities historically); scheduling and allocating resources, which are fundamental tasks on many industrial contexts [2]; even more practically, determining the structure and characteristics of molecules in chemistry and biology [3] [4]. The networking field is also rich of meaningful examples: routing packets in the network according to different objectives (i.e. minimum delay, shortest path) [5]; allocating the bandwidth on a contention-based protocol [6]; managing the radio-resources in a wireless network [7] are only some of the numerous examples which abound in the literature [8] [9] [10].

1.2 Current solving approaches

Typically, a combinatorial problem involves finding groupings, orderings or assignments of a discrete, finite set of objects, satisfying certain conditions or constraints. Due to the huge number of possible combinations from the solution space, solving the problem through complete enumeration is rarely an available option [11]: this is why two typologies of approaches, namely *exact methods* and *heuristics*, have been developed in the literature. The former, at the cost of possibly very high computational times, makes use of techniques like linear programming to obtain the optimal solution for the problem; contrarily, the latter favours resolution time, yielding though a feasible solution that cannot be guaranteed to be optimal [12]. However, many combinatorial problems are *NP-Hard*. This means that, for exact

methods, time scales exponentially with the numerosity; therefore, adopting this approach becomes practically unfeasible for larger instances. This is why so far heuristics represented a very popular alternative for the scientific research [13].

A parallel trend exploiting ML (machine learning) has also grown during the last decade [14]: this successfully overcame the pre-existing state-of-the-art solutions in fields like speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics [15]. Combinatorial problems also became object of ML's studies; particularly, several attempts in the literature have tried exploiting the graph structure of problems like the TSP (travel salesman problem) or the VRP (vehicle routing problem) [16] [17] [18] to automatically learn the policy getting the optimal solution. This approach, which is the one adopted by this research, presents several advantages on respect to handcrafted heuristics: among all, it is capable of replacing domain experts and can possibly generalize beyond training instances. [19].

1.3 The promise of "plug and play" ML solutions

As underlined by recent researches [20, 21, 22], machine learning and its Reinforcement Learning branch offer novel opportunities on respect to traditional solving methods. In order to replace handcrafted heuristics, they provide ML models capable of learning how to find the best strategy, either by supervision or through trial and error in a simulated environment. One challenge there is generalization, or how to transfer the knowledge acquired on an environment to similar but different ones. If successful, this resemblance to human intelligence, which is also able to *generalize beyond one's experiences* [23], should allow to build 'plug and play' solutions: with the only requirement of an evaluator and some generated input instances, the model should become able to get a meaningful representation of the problem and to pick meaningful decisions. In other terms, give the same machine examples of any problem and guide the learning evaluating the effectiveness of its actions; with training, it will hopefully learn to recognize the problems's important aspects and to output optimal solutions.

Machine learning promises automation: instead of constructing tailored and expect-demanding heuristics the goal is learning a "golden rule", namely a policy able to generalize to different contexts.

1.4 Limits of current ML architectures

Unfortunately, the revolutionary advantages of such an approach goes hand-in-hand with the complexity of implementing it. Obtaining a machine able not only to solve a particular problem, but also to learn the causal structures between constraints

in order to transfer to different environments is still a very hard [20] and partly unsolved task [24]. Like aforementioned, ML is currently able to achieve promising results which however are yet bounded to feature engineering [25] (an attempt to handcraft meaningful features exploiting technical knowledge of the problem), customized solutions [26], fixed-size networks [27] or related to small instances where the cost of training is affordable [17] [28].

Part of the prior literature, like [24] and [29], also studied generalization across problem size with varying success. However, even those pioneering works were bounded to sub-classes of combinatorial problems like the TSP and its variations. It is therefore not clear whether those novelties and findings also generalize for different combinatorial problems.

1.5 Proposal and contributions

The aim of this research is therefore to systematically evaluate whether the best architectural design choices for the TSP also transfer to different problems: in the framework of power and channel allocations, which are networking tasks of practical interest in terms of resource allocation, we test which are the key elements for the model to learn and which possible extensions might be helpful for a better generalization. The findings are then assessed for different sized networks in order to prove their robustness.

According to the obtained results, we state that current architectures are still affected by serious representation problems. Particularly, the state-of-the-art models are so far incapable of fully exploiting the existing structural notions of combinatorial problems, unless those are explicitly provided. This is indeed a limiting assumption and represent a severe impediment for generalization. In fact, we prove that current techniques, which follow the aforementioned two-step represent & choose approach, fails or under-perform when the representations is weak.

Finally, we propose the adoption of DE (distance encoder), a very recent proposal in the *ML* field. We first theoretically discuss how this tool aims at reinforcing the structural representation of the problem; then, we demonstrate that its role is beneficial when no other information are explicitly provided. This eventually constitutes the biggest achievement of the current work, since it enlightens new possibilities for future researches which seek for generalization patterns.

Chapter 2

Preliminary know-how

The main objective of this section is providing the reader the basic notions of the theoretical concepts handled in this research. We discuss more formally about graph problems, a way in which many combinatorial problems are usually represented; we describe the power and channel allocation problem, a fundamental task, for example, to optimally configure a network of IEEE 802.11 AP (access point)s; we then move into the ML world, defining concepts such as RL (reinforcement learning), GNN (graph neural network) and DE (distance encoder).

2.1 Graph, a Combinatorial Problems' representation

Graphs are fundamental representative aids in many real-world applications: atoms and bounds composing a molecule; social structures; cities and routes connecting them are simple and intuitive examples in which the information can be well represented using a graph-like structure [30] [31]. This is also the case for combinatorial problems, which is the reason why it is helpful to introduce some basic notions and nomenclatures.

In mathematics, the term *graph* refers to a set of vertices V and edges E such that $G = (V, E)$; in the most general case, a weight w_{ij} is also associated to the edge e_{ij} connecting node i and node j. We generally use the notions $|V|$ and $|E|$ referring to the cardinality of the nodes and edges sets, while $N(v_i)$ contains the *neighborhood* of vertex i, which is the set of vertices connected 1-hop away from v_i. The problems we analyse make use of *directed graphs*; this means that the edge $e_{i,j}$ connects node i (which therefore will be often called the *sending* node) to node j (which on the other hand will be the *receiver*) and that the reverse path $e_{j,i}$ could be different or not existing at all. Furthermore, both nodes and edges come *attributed*, meaning that one or more features are associated to them in the form of

vectors. **Fig 2.1** provides a visual example.

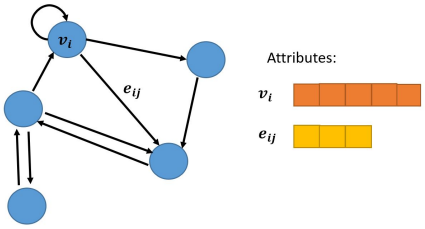


Figure 2.1: Graph example with notation

As we previously anticipated, the aim of this work is studying techniques valid for the TSP and the VRP on the power and channel assignments framework: it is therefore useful to briefly introduce those three examples as instances of graph problems.

2.1.1 The Travelling Salesman Problem

The TSP problem aims at finding a permutation τ of edges in a graph such that each node is visited once and only once and the total length of the path is minimum. In more formal terms, it will be:

$$L(opt) = min(\sum_{i}^{|V|} ||v_{\tau_i} - v_{\tau_{i+1}}||_2) \tag{2.1}$$

where $L(opt)$ is the optimal (shortest) length of the path, $|V|$ the set of vertices, v_{τ_i} the ith-vertex of the permutation τ and $v_{\tau_{i+1}}$ the (i+1)th. Notice that, in the context of the 2D Euclidean TSP, the edges weights usually represent the euclidean distances between nodes while the nodes themselves are often attributed by a tuple of (x, y) coordinates representing their position into the space. This is a NP-hard problem, meaning that finding the exact solution requires time which scales non polynomially with the size of the instances. It can be exactly solved through a TSP solver such as *Concorde* [32], while well known heuristics includes LKH [33], Christofides [34] and 2-opt [35].

2.1.2 The Vehicle Routing Problem

The VRP (vehicle routing problem) is a generalization of the TSP for which multiple routes must depart from a *depot* node, visit the other graph nodes and then return to the depot. The "vehicle" has a finite capacity D, while the visited nodes $i = 1...n$ have a demand $0 \leq d_i \leq D$ to satisfy. When a node is visited, its demand is subtracted by the vehicle's capacity and set to zero; after that, the

node can be no longer visited. Multiple routes can depart from the depot node; each of them must not exceed the capacity of the vehicle ($\sum_{i \in R_j} \delta_i \leq D$), where R_j are the nodes belonging to the route j. The goal is, like in the TSP, visiting all nodes with minimum cost. Without loss of generalization, the literature [17] [18] usually assumes a normalized capacity $D = 1$ and therefore also normalized demands $0 \leq d_i \leq 1$.

A generalization of the VRP itself exists, and is the so called SDVRP (split delivery vehicle routing problem): here, the nodes can be visited multiple times satisfying only a percentage of the original demand. This therefore introduces a novel feature, which is the *residual demand* $0 \leq \hat{d}_i \leq d_i$.

2.1.3 The Power and channel allocation problem

In the IEEE 802.11 WLAN (wireless local area network) context, choosing adequate levels of power and channels for a fleet of APs is a vital task to manage a scarce resource like radio-frequency. Those tasks are nowadays often performed through a Centralized Controller, which is in charge of both monitoring the APs and control their configuration [25]. At high level, three objectives are sought after when deploying a WLAN network. Those are guaranteeing optimal coverage to the users (which we will later call STA (station)) moving within the served area; maintaining low level of interference both within APs and with STAs; not "over-loading" a particular AP, but fairly distributing the amount of traffic across the entire fleet. **Fig 2.2** provides a visual representation of what we should consider computing a hypothetical *utility* for a generic (x, y) point.

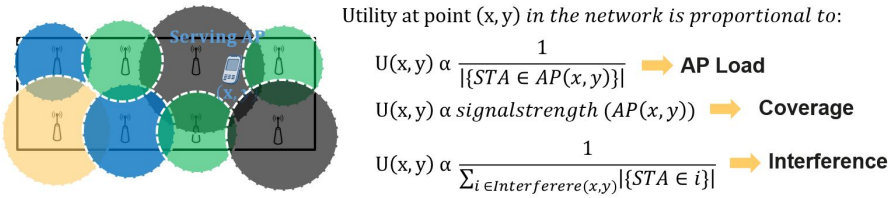


Figure 2.2: Example of power and channel assignment: colors represent the channels; diameters the powers. On the right, formulations to schematize the overall utility of the problem.

To what regards the relationships between power and channel assignment and the aforementioned objectives, the following might be helpful for the reader:

1. With a one-channel hypothesis, the higher the transmission power the better utility coming from coverage

2. With a one-channel hypothesis, the lower the transmission power the better utility coming from interference

3. More assigned channels implies less chances to interfere, guaranteeing more transmission power choices

4. The available channels represent a finite and possibly scarce resource; this basically means that we cannot afford to assign a distinct channel to each AP.

Most of the past literature treats the channel assignment problem as an instance of the GCP (graph coloring problem) [8] [36]. This is a well known graph problem, which on its simplest form consists on assigning colors to vertices such that neighbors do not share the same. The power assignment on the other hand is more generally a RAP (resource allocation problem), meaning that a resource has to be assigned in order to satisfy the constraints and minimize the overall cost of allocation.

According to the thesis' objective of studying to what extent machine learning architectures designed for the TSP and VRP problems transfer to this concrete power and channel allocation problem, we next provide background on existing machine learning techniques and architectures.

2.2 Machine Learning framework

As aforementioned, ML applications have dramatically increased during the last decade; increased computational resources have allowed those models to overtake the existing state of the art solutions in fields like speech recognition, machine translation, image captioning and other problems directly learning from raw data. Considering the vastity of the topic and the number of existing articles and books already providing a global view [15] [37], this background section only focus on the latest tools adopted by the literature to face graphs and combinatorial problems. Providing these basic knowledge here allow us to adopt a more pointed approach for the remaining of the paper.

2.2.1 Deep reinforcement learning

> *Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.*

RL consists on letting a *learning agent* interact with the external environment in order to fulfill a task: the agent is in charge of taking *actions* which will modify the *state* of the environment itself. Once committed an action, the agent will

observe the resulting *regret/reward* and modify its behaviour for the following choices according to how bad/good it did. **Fig 2.3** graphically schematizes the involved entities.

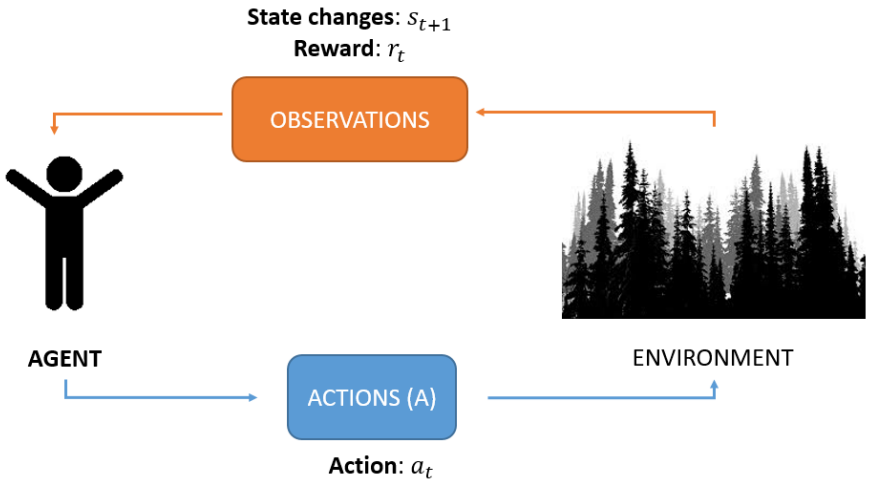


Figure 2.3: Learning cycle of RL

It is a branch of ML which differs from SL (supervised learning). In fact, for the latter, *labels*, namely "right answers", are generally provided by an external supervisor so that the model can learn to minimize the difference between its estimation and the correct choice; additionally, the model should become able to transfer the learned notions even on external environments never seen during training where no labels are available. However, for some cases it is practically expensive or infeasible to have both correct and general labels, and is therefore preferable for the agent to learn from its own experience [37]. At the same time, it differs from UL (unsupervised learning) since the primary objective of the latter is finding hidden structures and similarities on the data, while RL also aims at maximizing a reward function (or, symmetrically, minimizing a regret function) through its behaviour. **Fig 2.4** represents a toy-example.

It is also useful to formally define the elements composing RL [38]. Those are:

1. **Agent**: entity which interacts with the environment.
2. **Environment**: what is external to the agent; the context it "lives in".
3. **Action** a_t: how the agent interacts with the environment.
4. **Action space** A: space which contains all the possible choices that the agent can take. Can be discrete or continuous.
5. **State** s_t: observable response of the environment to the agent's action.

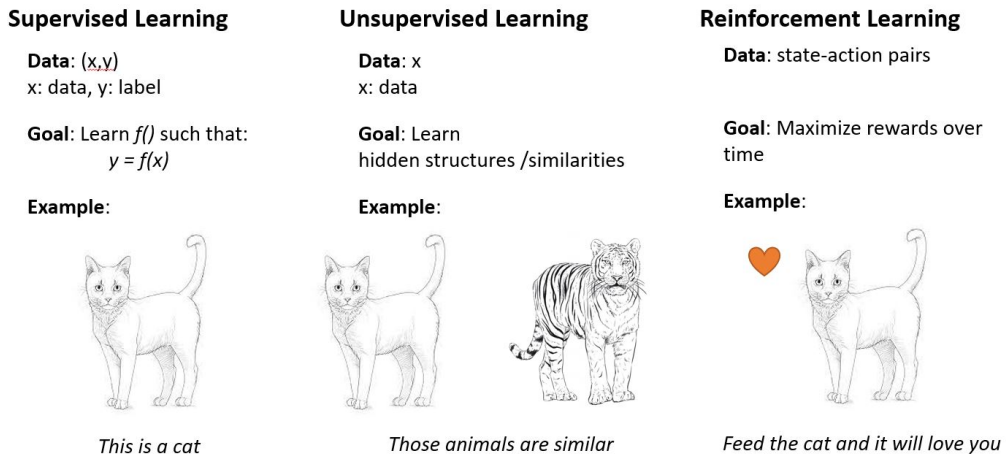


Figure 2.4: Different learning paradigms of ML

6. **Reward/regret** r_t: price/penalty which the agent receives according to the action it took. In the following we will consider the *reward* notation, but the cases are symmetric.

Like aforementioned, the aim of the agent is maximizing the total reward, which is:

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + ... + r_{t+N} + ... \tag{2.2}$$

Since this is clearly a diverging sum, it is often the case of introducing a discounting factor $0 < \gamma < 1$:

$$R_t = \sum_{i=t}^{\infty} \gamma^i r_i \tag{2.3}$$

On such a way, the sum becomes convergent and the model learns to estimate finished-time returns. R_t therefore evaluates the sum of discounted rewards starting from time t.

To express the the total future reward one could expect having taken an action a_t being on state s_t, it is also of help introducing the the so called Q-function:

$$Q(s_t, a_t) = E[R_t | s_t, a_t] \tag{2.4}$$

This basically represents the value which one could hopefully obtain being on a particular situation. A similar concept for example is familiar to chess players, which know that a given position might be winning or losing according to the current disposition of pieces and to the last opponent's move (the literature of games is dense of RL's applications [39] [40]). Clearly, if $Q(s_t, a_t)$ is given as an

"oracle" and there is no uncertainty (taking an action deterministically lead to a new state s_{t+1}), the best *policy* to adopt is:

$$\pi^*(s) = \underset{a}{argmax}(Q(s_t, a_t)) \tag{2.5}$$

Which is picking the action that returns the maximum reward.

Since however it is generally not the case of knowing the complete mapping of state and possible actions, two approaches have gained importance in the literature:

1. **Q-learning**: first estimating the Q-function (i.e.doing it with a Neural Network is known as *Deep Q-Learning*) and then obtaining the best policy similarly to 2.5.

2. **Policy learning**: directly inferring the policy with no need of a Q-function.

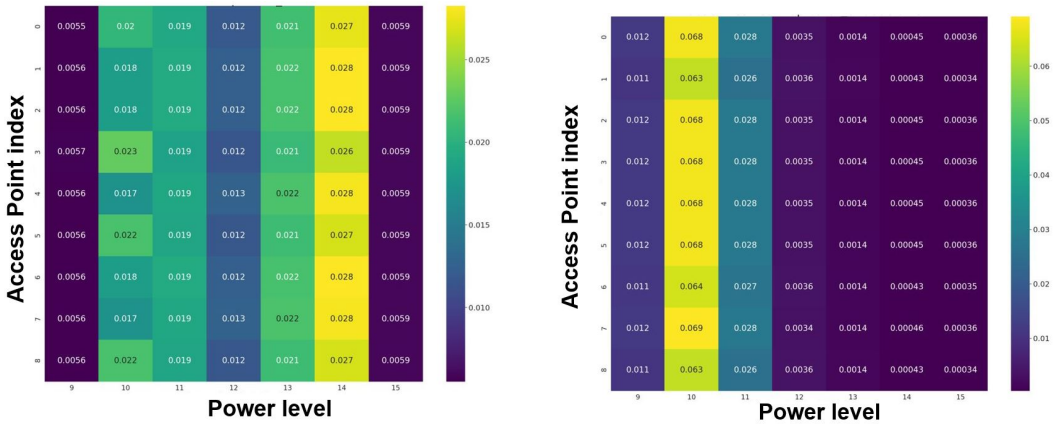
Of course, pros and cons exist for both methods: anyway, for the purpose of the research, it will be enough knowing that Policy Learning is generally chosen when the actions are picked from a numerous/continuous set or when the process involves randomness (i.e. the effect of an action might vary according to a random process) [41].

2.2.2 Guiding the learning

Policy Learning is about finding the best $P(a_t|s_t)$, that is the probability of adopting a behaviour a_t given a state s_t. The algorithm's learning is guided by reward, and consists on decreasing the probabilities of poor choices while increasing the chances of good ones. On its DL version, this is achieved through *back-propagation*, that is the common way through which a ML algorithm updates its inner weights once a complete episode (from first action taken to terminating condition) ends. Particularly, the back-propagated quantity is generally identified as:

$$Loss = -\log P(a_t|s_t)R_t \tag{2.6}$$

Where $\log P(a_t|s_t)$ is the so called *log-likelihood* of an action. This, at a high level, identifies "how much the model believes on its choices". Like we discuss on Section 3.1.4 and Section 3.1.5, at every decision step the neural network outputs a matrix of probabilities from which the next action is sampled. It can therefore happen that the model is very "convinced" about a particular set of choices (**Fig 2.5(b)**) or that, especially at the early learning phases, the probabilities are very similar (**Fig 2.5(a)**): intuitively, if the reward corresponding to a non-probable action is high, that behaviour must be reinforced increasing its probability of happening.



(a) Probabilities at early phases. Model is still uncertain and the probabilities are similar

(b) Probabilities at a more advanced stage. Model has gained confidence on its choices and seems to prefer low-powers allocations.

Figure 2.5: *Examples of probability-matrices returned by the neural network*

According to that, the weights update, which usually occurs according to the gradient rule, becomes:

$$W_{new} = W_{old} - LR\nabla Loss \tag{2.7}$$

$$W_{new} = W_{old} + LR\nabla \log P(a_t|s_t)R_t \tag{2.8}$$

Where LR is the adopted learning rate. The idea is that a negative reward, corresponding to a poor decision, subtracts importance to the weights which generated it; conversely, a positive reward increases their importance favouring that policy.

This method however is characterized by a couple of issues [42]:

- The reward's magnitude is problem-dependant and it could negatively influence the gradient steps (i.e. too big) "poisoning" the learning.
- In case of only positive rewards (i.e. length of a path on a graph) no actions would be penalized.

Those are the reasons why a term defined *baseline* is generally added to 2.9:

$$Loss = -\log P(a_t|s_t)(R_t - \mathbf{b}) \tag{2.9}$$

A baseline basically is a second term of comparison and, intuitively, represents the "average behaviour" of the current model. A positive difference of $(R_t - \mathbf{b})$ means that the latest solution provided by the model is doing better than the average,

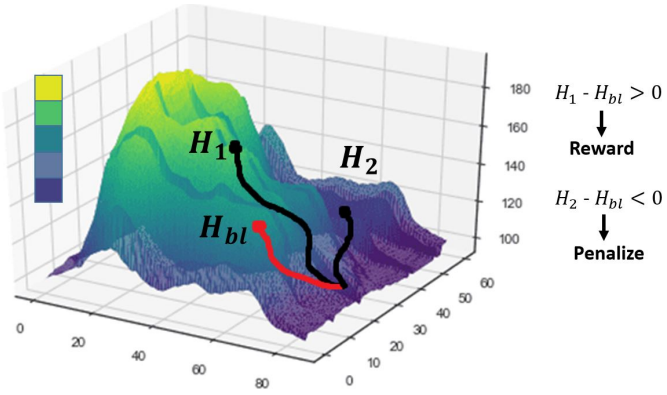


Figure 2.6: Policy role: the reward represents the altitude reached by the mountain tracks, baseline in red. The first solution (1) is rewarded, since it over-performs the baseline; the second solution (2) on the other hand is penalized. Notice that without a second term of comparison, the reward (altitude) would be always positive.

which is a behaviour that must be favoured; on the contrary, with a negative difference, the model is unable to achieve its average performance and its choices must therefore be discouraged. **Fig 2.6** provides a toy example.

While the previous Section 2.2.1 and Section 2.2.2 examined a learning approach and how to apply it, the followings introduce recent and wide-application tools for constructing a learning model. Having been greatly applied also in the context of combinatorial optimization, it is therefore important to present them.

2.2.3 Attention-based architectures

Original attention

In the ML field, the term *attention* first gain popularity with the almost in parallel researches of D. Bahdanau and M.T. Luong [43] [44]. Both of them were conducted in the NMT (neural machine translation) field, which aims at training a NN (neural network) to translate input sentences to different languages [45]. However, they soon became an integral part of compelling sequence modeling in various tasks (includinig combinatorial problems) and backbone of architectures like BERT [46] and GPT [47].

Attention's original intent was to enhance the representation capacity of NMT models, solving a potential scalability issue of pre-existing architectures. A NMT problem at the time was usually faced through a so-called sequence-to-sequence (sequence to sequence) scheme [48] [49]. In few words, a sequence-to-sequence model aims at outputting a set of items, given as input another set of objects (i.e. sentences on two different languages). Generally, this architecture can be further

divided in two blocks, namely an *Encoder* pile and a *Decoder* one. The goal of the first is to find a meaning-full and fixed sized representation of the input, a so called *context*; this is then passed to the second and processed to obtain the output. **Fig 2.7** shows the overall picture.

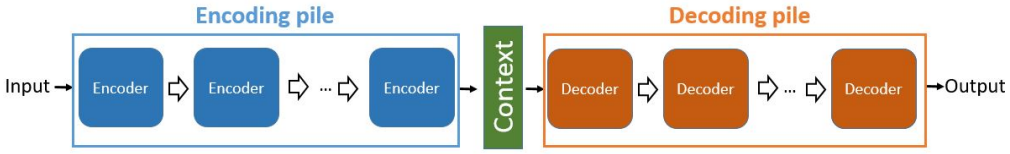


Figure 2.7: sequence-to-sequence model: the encoding pile (Encoder) gets as input a set of item and produces a context. This is then re-elaborated by the decoding pile (Decoder) which generates the output.

Like the researchers of [44] point out, the fixed-length *context* represents a bottleneck for longer input (i.e. longer sentences); the model might fail on identifying the dependencies between terms, resulting on worse overall performances. This is where attention is of help: the idea is that, instead of storing only the last-layer context, all the embeddings h_i, namely fixed-size manipulations of the input, are saved becoming active part in the decoding phase as well. Intuitively, instead of creating an artificial summary of the input, the *context* used before, it will be the model itself to understand where to "focus its attention". While **Fig 2.8** provides a visual example, the pipeline through which this is performed can be summarized like:

- Find fixed-size representations (embeddings) h_i of each input item i with the encoding pile.
- For each decoding block, compute a "score per input", which takes into account the current hidden state of the decoder s_t and all the embeddings from the encoder. Notice that s_t represents a sort of "memory", and is a typical element of RNN (recursive neural network) structures.
- Obtain the softmax of the scores.
- Multiply the scores by the softmax, in order to amplify the importance of elements with high score and shadow the others.
- Sum up the weighted vectors aggregating the resulting information to obtain an updated context c_t.

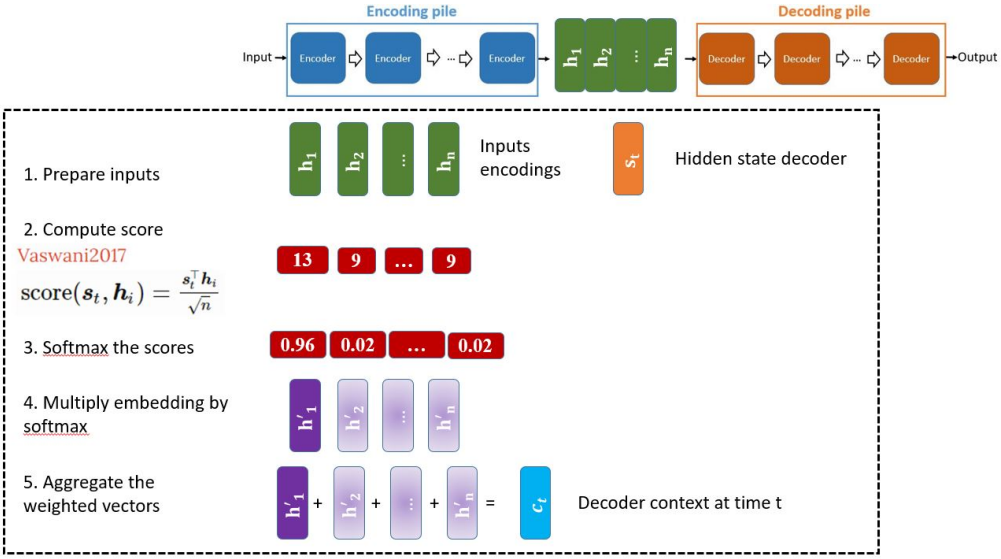


Figure 2.8: Attention mechanism using a *weighted Luong attention* from [50] (but there are other alternatives in the literature, like [43] and [44]): the context takes into account the importance of each input element in order to understand "where to focus attention".

Transformer

A Transformer is an encoder-decoder architecture first presented by the researchers of [50]. Its aim is overtaking the pre-existing recurrent cells (like the sequence-to-sequence presented in 2.2.3) with the motto *Attention is all you need*. In order to overcome the sequential behavior of previous models, the key is to substitute the hidden states of RNN units with attention-based mechanisms. In fact, the former, as **Fig 2.7** suggests, had to go over the input set one item at a time to improve the overall representation. The goal is therefore allowing parallelization, speeding up the learning process. Both the encoder and the decoder are built with identical blocks, stacked one on top of the other to create multiple-layer piles. Notice that each block do not share its weights with the others. The encoder's components are further divided into two layers, a so-called *Self-attention layer* and a *Feed Forward layer*; the decoder maintains the same modules, only adding a conventional *Attention layer* in the middle (like the ones used in the sequence-to-sequence) for determining what parts of the encoder's outputs to pay attention to. **Fig 2.9** provides a visual representation.

Particularly, instead of running a single item (hidden state s_t of the decoder block in **Fig 2.8**) against a set of inputs (embedded inputs in **Fig 2.8**) to determine where it should focus its attention, the *Self-attention layer* compares each element of

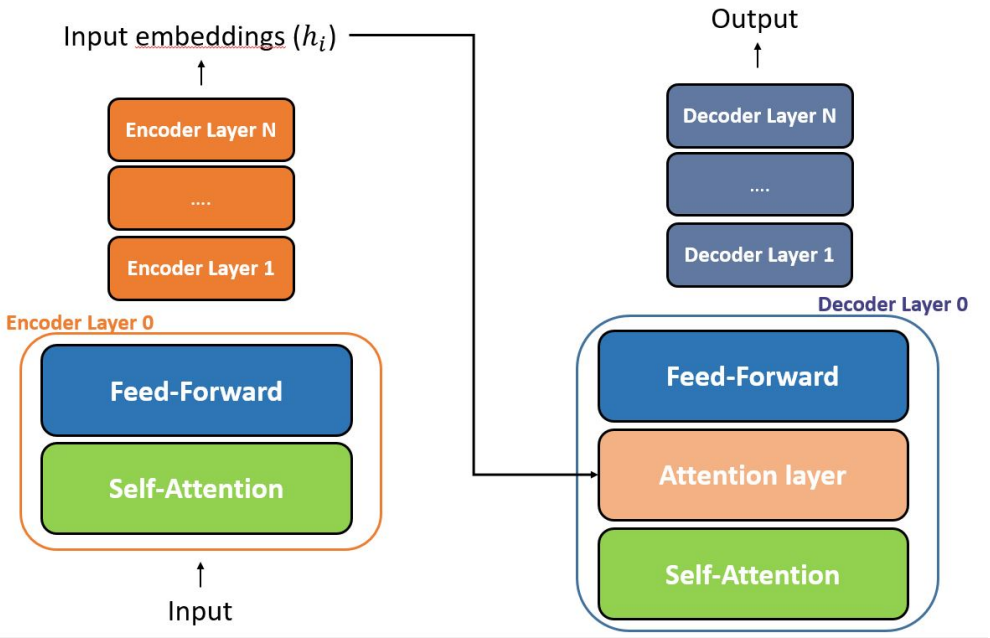


Figure 2.9: Transformer model: only feed-forward layers and attention-based ones are applied both in the encoder and in the decoder. Notice that each block has the same layers but does not share its weights with the others. This means that each of them is trained separately.

a set against all the others (including the element itself). On such a way, their initial representation, which at the beginning is uniquely self-describing, gets updated including notions of the relationships with the others. Practically, this is obtained running the attention mechanism multiple times changing the "subject under study" at every iteration. Introducing the notation used by [50], which will be useful for the remaining of the paper, the following pipeline is applied (**Fig 2.10(a)** and **Fig 2.10(b)** provides a visualization):

- For each element x_i of the input set, obtain a query (Q), a key (K) and a value (V) through learnable projections:

$$Q = W_q * x_i \tag{2.10}$$
$$K = W_k * x_i \tag{2.11}$$
$$V = W_v * x_i \tag{2.12}$$

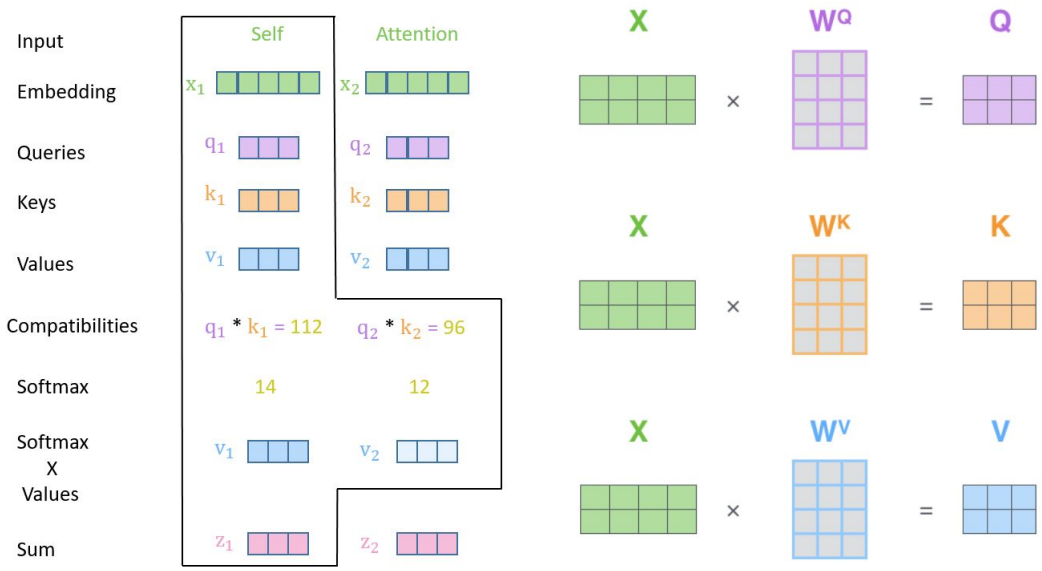
- Run the query of each element against the keys of the others (including the element itself) to obtain a *compatibility* of the element with the others. This is equivalent to the attention's score and is performed using the *weighted Luong*

attention previously mentioned:

$$c = \frac{Q * K}{\sqrt{d}} \tag{2.13}$$

Being $d = 64$ the dimension of the key vectors used in [50].

- Compute the softmax of each compatibility to obtain a smoothing factor.
- Multiply the results for the values V (including the value of the element itself), so that each of them is weighted according to its importance on respect to the element under study.
- Aggregate the results through summation and obtain new representation for the queried vector.



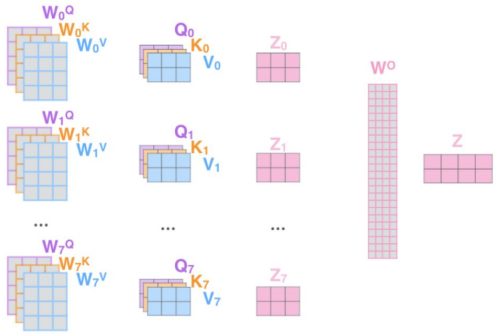
(a) Self-attention overall mechanism. In the example, the word "Self" gets the highest contribution on updating its own representation z_1.

(b) How query, values and keys are obtained. Notice that W_q, W_v and W_k are learnable parameters

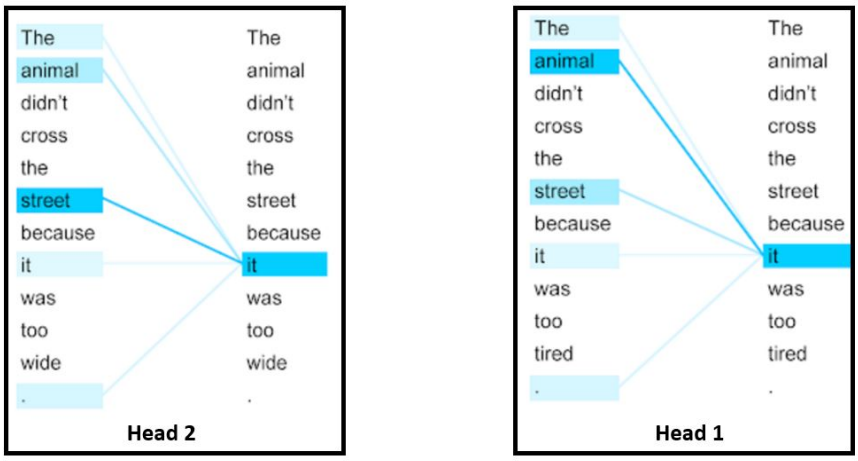
Figure 2.10: *Toy example of the self-attention procedure. Images from [51].*

Furthermore, since it is often the case that the highest contribution comes from the queried element itself (see example on **Fig 2.10(a)**, where "Self" is getting the maximum compatibility), a *Multi-head* scheme is adopted. This basically consists on using "different heads", namely different learnable matrices (i.e. with different

initialization) in order to obtain more versions of Q, V and K. Multiple heads guarantee "many points of view", which are eventually aggregated with a new learnable matrix W_z to obtain z_i. **Fig 2.11(a)** gives more details graphically, while **Fig 2.11(b)** is a toy examples which demonstrate the benefits of using many heads.



(a) Computations to achieve multi-head attention.



(b) Example in NMT: the model relates the pronoun *it* in the sentence to different objects according to the scores obtained from the different heads. For this specific case, it is up to W_z understanding that the second head is providing the right answer.

Figure 2.11: *Multi head attention. Images from [51].*

2.2.4 Graph Neural Networks

GNN have been firstly introduced with [52] and [53] to solve graph and nodes classification tasks. Their goal was fixing the general ML approach at the time

when facing a graph problem, which consisted on "squashing" the graph into flat vectors. Of course, that implied loosing meaningful information, like the one of the topology. GNNs are based on an information diffusion mechanism: each node, which as anticipated in Section 2.1 is a featured entity, periodically sends its current state to its surroundings on a *message-passing* fashion. The receiver stores its message together with the ones coming from the other neighbors; once the message passing is over, it eventually updates its state according to the new "notions" it received. This procedure is computed *until equilibrium*, stacking multiple blocks one on top of the other. At high level, the stopping condition generally indicates that, according to some criterion (i.e. few changes in the representation), keep propagating messages is not helpful anymore.

On more formal terms, one can write that:

$$h_i = f(x_i, h_j : j \to i) \tag{2.14}$$

Where h_i is the updated state of the receiving node, x_i is the initial self-describing feature of the receiver, $h_j : j \to i$ represent the set of i neighbors and f is a general function which aggregates those two information. Notice that, in general, the messages coming from the neighborhood can be weighted: this highlights that surrounding nodes might give different contributions to the receiving one (i.e. closer nodes might influence more than further ones). When the message passing is weighted, the mechanism is called *anisotropic*. This approach has been shown to outperform solutions in which no smoothing mechanism was applied [54] [55], and it is therefore appreciated by the researchers..

Nowadays GNN represent an important milestone for several tasks, including graph classification [55], unsupervised graph clustering [56] and graph representation learning [57] [58], which learns to represent graph nodes, edges or sub-graphs by low-dimensional vectors. The latter objective, which is commonly defined as *representation learning*, is then the one we mostly cover on the paper. For further information about other applications and details about the adopted methodologies, we refer to the research of [59], which provides a complete summary-article on those topics.

The original proposal of [53] enlarged the concept of RNN to the graph domain. For the state update, they proposed a multi-layer perceptron:

$$h_i = A\sigma(B\sigma(Ux_i + Vh_j)) \tag{2.15}$$

Where A, B, U and V are learnable projection, x_i the input features of the receiving node and h_i and h_j the hidden states of the receiving and sending node respectively. With time, more options, transferred from the fields of image processing and natural language processing, became available to represent the mapping function f: those included recurrent units ([60]), which are meant to act as "gates" for the message

passing; convolutional structures ([61] with *MoNet* and [54] with *GraphSAGE*); even attention-based mechanism to create the so called GAT (graph attention network) [62] [17].

For the scope of this research, we refer to a couple of anisotropic variants from the literature. Those are the aforementioned GAT architecture from [17] and a so called GGCN (gated graph convolutional network) model [28] [63], which refers to the works joining convolutional tools and graphs.

Graph Attention Network model

As previously mentioned, GAT architectures take inspiration from the attention mechanisms discussed on Section 2.2.3 to build an anisotropic message passing. Particularly, among the general framework presented in [62], one of the available options, which is the one used by researchers in [17], recalls the MHA (multi head attention) solution from the Transformer model of Section 2.2.3. In fact, the first layer of each GAT block is expressed as:

$$\hat{h_i} = BN^l(h_i^{l-1} + MHA(h_1^{l-1}, ..., h_n^{l-1})) \tag{2.16}$$

Where BN represent a Batch Normalization layer and MHA a multi-head-attention transformer layer. The number of heads for the MHA is set to $M = 8$. Like in the Transformer original example, a second layer, namely a feed-forward layer FF, is then following to create the updated state $h_i^{(l)}$ of each node:

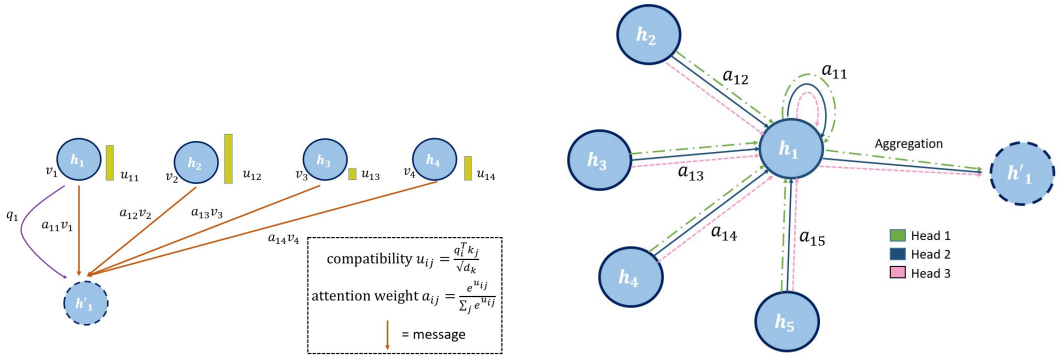
$$h_i^{(l)} = BN^l(\hat{h_i} + FF^l(\hat{h_i})) \tag{2.17}$$

Fig 2.12(a) provides an illustration of the weighted message passing, also recalling notations from the Transformer; for the technical details, we refer to the appendix A of research [17]. **Fig 2.12(b)** on the other hand exemplifies how the different heads must be interpreted.

As the reader can start noticing, no structural notions, which basically are information on how the topology is structured, are provided. Therefore, it is entirely up to the GAT to "discover", for example, how far is a node on respect to another. A variant slightly more informative in that sense also exists, and consists on preventing some nodes to participate the attention mechanism of some others according to a neighborhood masking (i.e. if neighbor, participate; mask otherwise).

Gated Graph Convolutional Network model

While GAT takes inspiration from attention, a GCN (graph convolutional network) mimes the behaviour of a CNN (convolutional neural network). Considering an



(a) Illustration of weighted message passing using a dot-attention mechanism. Only computation of messages received by node 1 are shown for clarity. The formulas and notations are very similar to the Transformer ones.

(b) An illustration of MHA (with K = 3 heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations.

Figure 2.12: *GAT methods*

example coming from computer vision, is the equivalent of trying to synthesize the feature of one pixel (i, j) studying its surroundings (i', j'). Like explained on Section 2.1, since on a graph context it is the topology which defines the meaning of neighborhood, we thus obtain:

$$h^{l+1} = f_{G-CNN}(h_i^l, h_j^l : j \to i) \tag{2.18}$$

With h_i being the receiving node, h_j the sending one and $h_j \in N(i)$. A possible example of f_{G-CNN} is proposed by the researchers of [64]. In fact, they obtained an anisotropic message passing exploiting an edge-gating mechanism:

$$h_i^{l+1} = ReLU(\sum_{j \to i} \eta_{ij} \circ V^l h_j^l) \tag{2.19}$$

With $ReLU$ being the rectified linear unit and defining the edge gating system like:

$$\eta_{ij} = \sigma(A^l h_i^l + B^l h_j^l) \tag{2.20}$$

Where V, A and B are linear projections. This formulation was then improved by other researchers in [65], which explicitly added the receiving node h_i to 2.19 obtaining:

$$h_i^{l+1} = ReLU(U^l h_i^l + \sum_{j \to i} \eta_{ij} \circ V^l h_j^l) \tag{2.21}$$

This, similarly to a central pixel updating on a classical convolutional network, underlines the importance of the receiving node. For what regards the initialization, it generally is $h_i^0 = \dot{x}_i$, being $\dot{x}_i$ a fixed size embedding of the nodes features.

Eventually, another improvement is provided by the studies of [16] which, among all, explicitly added edge features and processed the gates with a softmax function such that:

$$h_i^{l+1} = h_i^l + ReLU(BN(U^l h_i^l + \sum_{j \to i} \eta_{ij} \circ V^l h_j^l)) \tag{2.22}$$

With the gates computed like:

$$\eta_{ij}^l = \frac{\sigma(e_{ij}^l)}{\sum_{j \to i} \sigma(e_{ij}^l) + \epsilon} \tag{2.23}$$

and with the edge embedding being:

$$e_{i,j}^{l+1} = e_{i,j}^l + ReLU(BN(A^l h_i^l + B^l h_j^l + C^l e_{ij}^l)) \tag{2.24}$$

With U, V, A, B and C linear projections, BN a batch normalization layer, ϵ a small number to avoid divisions by 0 and $ReLU$ the rectified linear unit. Notice that, similarly to h_i^0, e_{ij}^0 is an initial embedding of the edges feature. A useful schema is provided by the work of [28] and is reported on **Fig 2.13**. Furthermore, on its appendix this paper also provides a useful summary of existing GNN models to which we referred for this section.

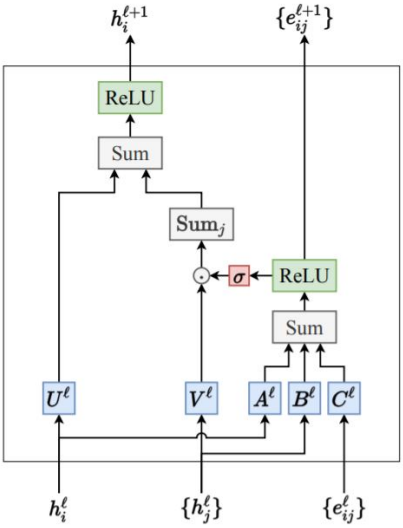


Figure 2.13: Gated Graph Convolution Network

To conclude, it is interesting to notice that, on respect to the GAT counterpart, a GGCN provides much more structural information. In fact, it explicitly includes

the concepts of neighborhood and edges using a distance matrix to weight the messages. Theoretically, this model should be therefore more capable of taking into account not only the nodes attributes, but also the topology.

2.2.5 Distance Encoding

We conclude the background chapter introducing a very recent tool called DE (distance encoder). This was first introduced by the papers of [66] [67], which underlined potential issues of current GNN models. Those are said to be *unable of establishing correlation between nodes representation* [67]; with a practical example provided by [66], *if node attributes are all the same, then for any node in a r-regular graph GNN will output identical representation.* In practise, they state that current GNN models mostly rely only on nodes: there is therefore a serious risk, with possibly severe impacts over the results, that their relationships cannot be as exploited as they should be. The work of [67] reinforces the concept with a graphical example which we report on *Fig 2.14*: since the two components of the graph, that represents the so called *foodweb*, are isomorphic (contain the same number of graph vertices connected in the same way), the model is unable to answer the query «Which one is more likely the predator of Pelagic Fish, Lynx or Orca?».

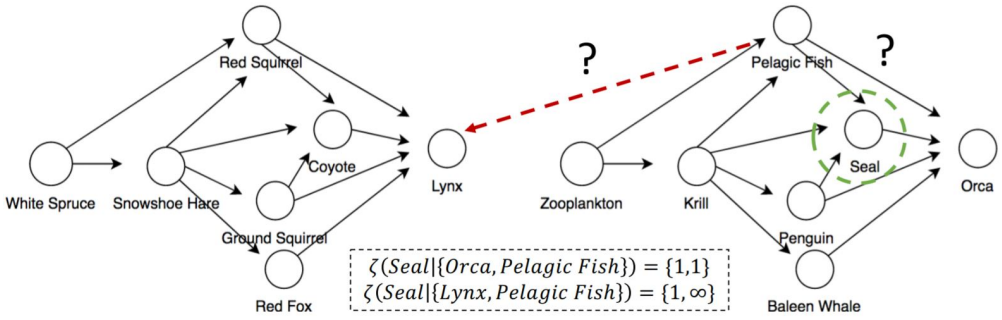


Figure 2.14: Foodweb example: since the two components are isomorphic, the final embedding of nodes is the same and the model is unable to correctly fulfil its quest. A representation which also reports that the actual distance between Lynx and Pelagic Fish is infinite, like the dotted box reports, is still missing from current GNN architectures.

The aforementioned cases lead the researchers of [66] to formulate the so called DE. Since current models are incapable of capturing the relationships between nodes, which are generally pieces of information embedded in the graph's edges, DE attempts to transfer the edge information to the nodes themselves. On such a way, a novel artificial feature is added to the nodes information which, differently from the edges ones, GNN can treat.

This is practically achieved as following. First, for each node u belonging to the graph, we must compute its DE:

$$D = \zeta(u|v, A)|v \in S \tag{2.25}$$

Where A is the adjacency matrix, S is the set of nodes we consider neighbors of u (can be $N(u)$ or a smaller fraction) and ζ characterize some "distance" between node u and node v. To evaluate the latter, the article proposes different options:

- Shortest path distance between v and u.
- Personalized PageRank scores (concept used by *Google Search* and originally presented in [68]: basically, each node has a weight signifying its importance in the graph and you move from a node to another favouring "heavier" nodes).
- Hitting time of the random walk from u to v.

The DEs of each node basically is a NxN distance-matrix, being N the number of nodes; once obtained it, we aggregate by columns ($dim = 1$) so that each node holds only one measurement "summarizing" its relationship with the others. We therefore move from an $N \times N$ space to a $N \times 1$ one, obtaining:

$$\zeta(u|S, A) = AGGR_{dim=1}(D) \tag{2.26}$$

$\zeta(u|S, A) = X_S$ represents a new per-node structural information which can be aggregated with the input "raw features" (i.e. current power and channel information for the power and channel assignment) or used by itself. An aggregation's example is the *concatenation*, which leads to:

$$X = (X_{raw}||X_S) \in \mathbb{R}^{\mathrm{N\times(d+k)}} \tag{2.27}$$

Being d the column-dimension of the raw features and k the column-dimension of the encoded-distance ($k = 1$ on 2.26, but there might be more general cases).

In practise, we also proposed a second way to aggregate the novel structural information with the "old" raw features. Once obtained the NxN distance-matrix, we do not directly move to a $N \times 1$ summarized vector, but we first expand the raw features to an NxN matrix as well. At this point, we immediately aggregate raw and structural info like in 2.26 to obtain a $NxNxF$ structure, where F is the number of raw and structural features ($F = (1 + 1) = 2$ in the example). Finally, we summarize the NxN matrix info into the usual $Nx1$ vector; notice that, after some reshaping, we obtain again vector X of 2.27. We thought this second algorithm could be useful since, both in 2.26 for the first method and on the $NxNxF$ structure for the second, a linear transformation $L = WX_s + b$ is usually applied so that the model can learn how to handle the inputs. For the second attempt, the model is immediately provided both structural and raw information for the transformation, while this is not the case for the first one. Eventually **Fig 2.15** provides a scheme of the two methods.

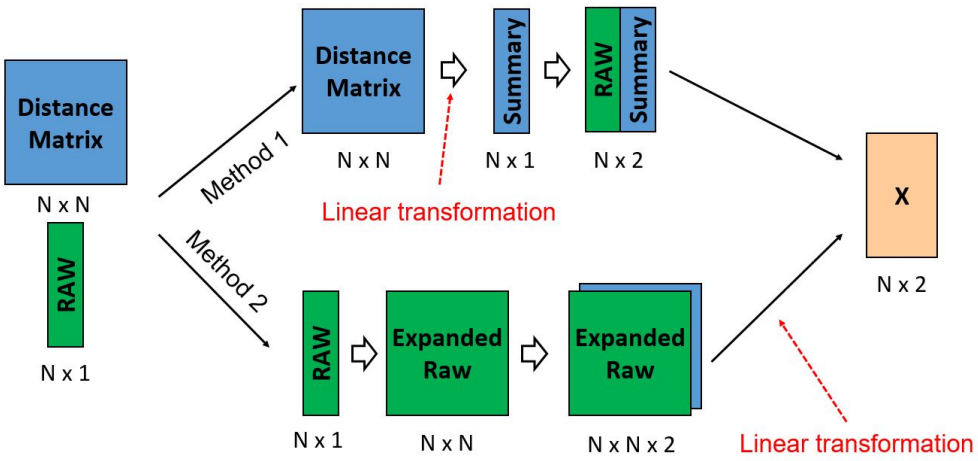


Figure 2.15: Two methods for applying DE; notice that both aims at producing a final vector X comprehending both the novel structural info and the "old" raw ones. However, the second applies the linear transformation having both raw and structural features at disposal.

Chapter 3

Related Work

The purpose of this chapter is to analyze the evolution of instruments and techniques that led to the most recent deep learning breakthroughs and generalizations regarding combinatorial problems. As mentioned in Section 1.5, our aim is assessing which of them is capable of transferring from a sub-class of problems such as the TSP and the VRP to a resource allocation one: Section 3.2 therefore reports a summary of the main building blocks which are currently representing the state-of-the-art for those instances, while the following Chapter 4 discusses them in the power allocation context and eventually adds some novelties.

3.1 Toward the tested architecture

3.1.1 Pointer Network

The first attempt to tackle combinatorial problems using SL tools was the *Pointer Network* proposed by the work of [69]: this, as already mentioned on Section 2.2.3, successfully enhanced the pre-existing sequence-to-sequence approaches enabling analysis on variable-sized instances. A pointer network is an encoder-decoder RNN-based architecture: the inputs are converted into *codes* by an encoder, and fed to the following generative model. This, as shown on **Fig 3.1**, apply a content-based attention using the hidden-state of each decoding block to point back at the encoded input. When tested on a combinatorial problem like the TSP, the attention mechanism basically helped the neural network discovering the order in which the nodes should be visited (i.e. from the current decoding step, corresponding to the last visited node, which is the one receiving "more attention" for the next visit?).

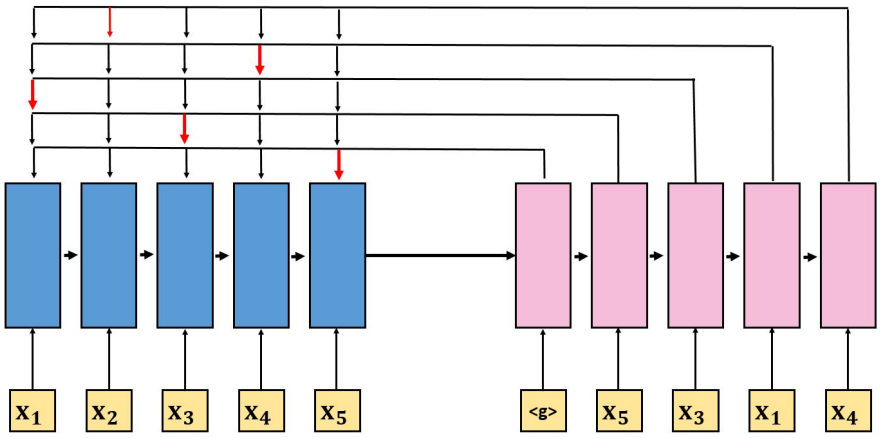


Figure 3.1: Pointer Network of [69]; each encoding block generates an embedding of the corresponding input element also considering the previous ones. During decoding, an attention mechanism point back to the input embeddings.

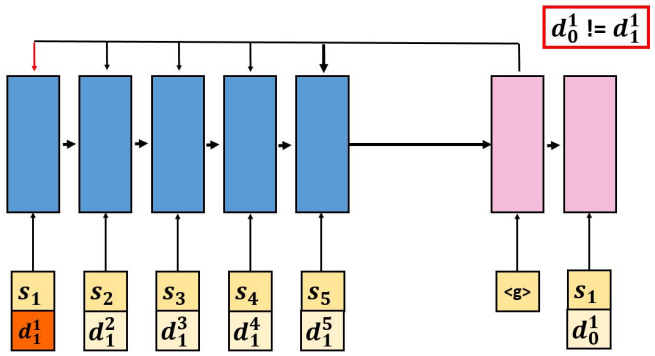
3.1.2 RL in combinatorial problems

While the neural network of [69] learned on a supervised manner, [70] adopted RL to leverage a similar model. In their research, the authors proposed a couple of learning techniques based on the RL-policy learning approach; those are the *Pre-training* and *Active Search* methods. While the first one makes uses of a training set to "fix" a policy, and then adopt it during the test-phase, the second involves no pretraining and directly learn on a single test instance starting from a random policy. In practise, they found that the best-performing is a mixture of the two; in Bayesian terms, pretraining is the equivalent of prior information which the model then use to face the test. Most importantly, they also defined a couple of behaviours the model can adopt while picking a solution: those are the *greedy behaviour*, which deterministically choose the most probable action, and the *sampling behavior*, which on the other hand sample the next action according to their probability density distribution.

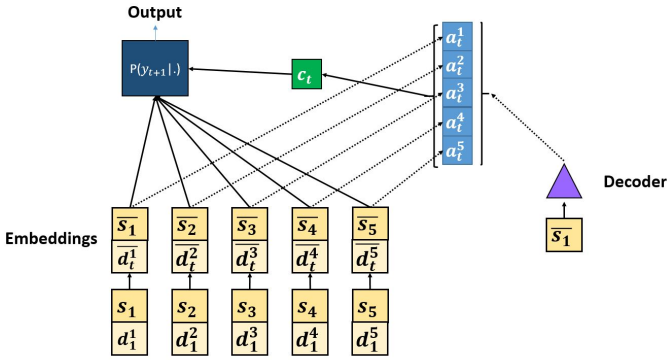
3.1.3 Handling dynamic features

The study of [18] represented an important milestone for ML application to combinatorial problems: firstly, according to the research of [71], they clearly stated that the those problems do not necessarily imply sequential order. This means that a RNN encoder model, which as mentioned before represents an important element of the Pointer Network and strictly works on a sequential manner, is actually unnecessary in some framework and can be easily substituted by less computationally-demanding structures (even simple linear projections). Secondly,

facing the VRP, a problem similar but slightly more complex than the TSP, they introduced the concepts of *dynamic* and *static* features. Like we stated in 2.1.2, the VRP adds a *demand* per node d_i^t to the simple TSP coordinates s_i, obtaining a new tuple features $x_t^i = (s_i, d_t^i)$: as stated above, a visit to a node fulfills its demand and sets it to zero. This means that the nodes are characterized by varying features which, as **Fig 3.2(a)** testifies, represent a severe limit for the pre-existing Pointer Network.



(a) VRP re-embedding issue: when a node is visited, its demand is set to zero and the whole Pointer Network must be recomputed to take further decisions.



(b) Solution proposed by [18]; no RNN architecture is used during encoding (and therefore no chain-rule exists if a node is modified) and the decoder only receives static features.

Figure 3.2: *Dynamic elements in VRP architecture*

Therefore, in order to update its hidden state h_i, [18] proposes an architecture (summarized in **Fig 3.2(b)**) in which the decoder only receives static elements s_i; in contrast, dynamic ones are part of the attention mechanism through which a new context is computed and the new decisions are made. This, referring to $\hat{x_t^i}$ as

the embedded features, can be expressed in the form:

$$a_t^i = a_t(\hat{x_t^i}, h_t) = softmax(u_t^i) \tag{3.1}$$

$$\text{where } u_t^i = v_a^t tanh(W_a[\hat{x_t^i}||h_t]) \tag{3.2}$$

And the context is obtained as:

$$c_t = \sum_i^M a_t^i \hat{x_t^i} \tag{3.3}$$

The probabilities are eventually obtained with a final attention layer between the input embedded features (static and dynamic) and the updated context c_t:

$$P(y_t + 1|Y_t, X_t) = softmax(\tilde{u}_t^i) \tag{3.4}$$

$$\text{where } \tilde{u}_t^i = v_c^t tanh(W_c[\hat{x_t^i}||c_t]) \tag{3.5}$$

3.1.4 The GAT and an autoregressive decoder

Even if researchers in [18] abandoned the sequential encoding of the Pointer Network, the obtained architectures were still unable of fully capturing the structural information provided by the graph representations. Particularly, each encoding only represented the referring node, and no neighborhood information could be inferred by the network. That is why, exploiting the researches of [50] and [62], the work of [17] was able to face problems like the TSP and the VRP using an attention-based graph neural network model, namely the GAT (graph attention network) already mentioned on Section 2.2.4.

The proposal is once more an encoder-decoder architecture trained with a RL paradigm. The input features depend on the faced problem (i.e. VRP nodes are characterized by a demand) but always include the points coordinates in terms of (x, y) position: as it is worth remembering from Section 2.2.4, a GAT architecture runs a multiple-layers self-attention mechanisms which aims at discovering the importance of each node for all the others. Hence, nothing but the input coordinates (and an optional masking procedure to disguise non-neighboring nodes) tells the network anything about the nodes' positions; no edge attributes, which embed the topology distance-matrix, are actually used, and coordinates become vital to get an adequate representation.

The layer-0 embeddings of fixed size $h_{dim} = 128$ entering the GAT are obtained as following:

$$h_{l=0}^i = W^x x_i + b^x \tag{3.6}$$

$$\text{where } x_i = [s_i||d_i||...] \text{ and is a concatenation of the input features} \tag{3.7}$$

The GAT structure is the one presented on Section 2.2.4 and makes use of 3 layers; the output consists on embeddings $h^i_{l=L}$ which do not represent only the node information anymore, but, thanks to the message-passing procedure, also its surroundings.

The decoding process is performed on a so called *autoregressive manner*; this indicates that each macro-episode, which is a run lasting until some stopping condition is met (i.e. all nodes visited), is faced through micro-actions regarding one node at the time. Basically, the decoder phase is a loop which starts with the decision for the first node and ends when all nodes have been visited/satisfied. At the end of the encoding-decoding process of each micro-event, a probability matrix containing the probabilities of visiting the next node is emitted; a decision is then picked according to the current policy (most probable if greedy, sampling otherwise) and the loop continues.

Furthermore, each micro-step recalls the idea of creating a context already cited for the work of [18]. Intuitively, taking track of the current circumstances is coherent with the sequential decisions described for the auto-regressive behaviour: those are iteratively modifying the contour conditions (i.e. the node which we are currently visiting) and must be taken into account for the future ones. To generate this context, [17] proposes to first evaluate a *graph summary* using the node embeddings from the GAT:

$$\hat{h_G} = \frac{1}{n}\sum_{i}^{n} h^i_{l=L} \tag{3.8}$$

Then, in order to take into account the decisions committed so far, to evaluate the context like:

$$h^N_{(c)} = \begin{cases} W_c(\hat{h_G}||h^{\pi_{t-1}}_{l=L}||h^{\pi_1}_{l=L}) & t > 1; \\ W_c(\hat{h_G}||v^l||v^f) & t = 1; \end{cases} \tag{3.9}$$

Where $||$ is the concatenation operator, W_c a learnable projection, N is the N-th step of the decoding loop, $h^{\pi_{t-1}}_{l=L}$ and $h^{\pi_1}_{l=L}$ the last and first visited node so far. The context of 3.1.4 is therefore always taking into account the "summarized graph", adding at the same time the information about the first visited node and the last one inserted when available. If they are not, learnable placeholders v_f and v_l are used.

The context $h^N_{(c)}$ acts like a fictitious node against which, similarly to [18], a self-attention mechanism from the other nodes is run. While **Fig 3.3** tries to give a

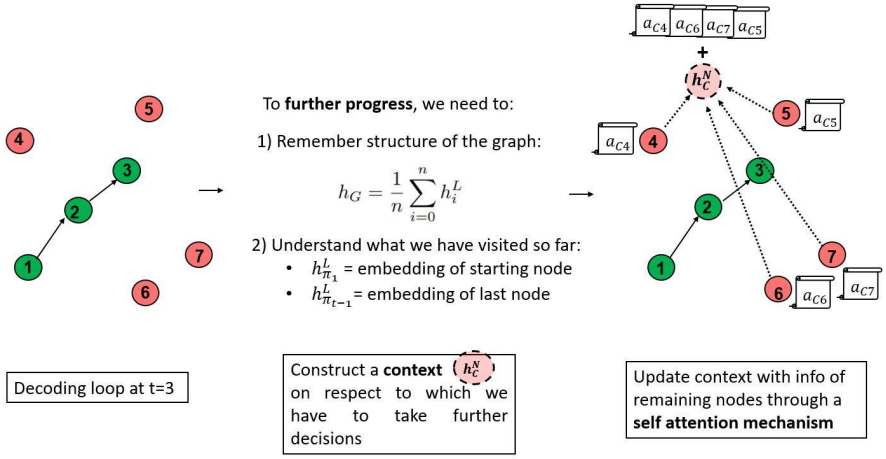


Figure 3.3: Decoder loop at timestamp 3: first we create a new context $h^{\hat{N}}_{(c)}$, then we run a self-attention mechanism between the remaining nodes and the context to update the latter.

visual representation, this, with a Transformer-like notation, is achieved evaluating:

$$q = W^q h_{(c)} \tag{3.10}$$
$$k_i = W^k h_i \tag{3.11}$$
$$v_i = W^v h_i \tag{3.12}$$
$$\tag{3.13}$$

And:

$$u_{(c),j} = \begin{cases} \frac{q^t_{(c)} k_j}{\sqrt{d_k}} \text{ if } j \neq \pi_{t'} \ \forall t' < t \\ -\infty \text{ otherwise;} \end{cases}$$

$u_{(c),j}$ are the compatibilities of each node with the context, and are used to obtain:

$$a_{(c),j} = \frac{e^{u_{(c),j}}}{\sum_j (u_{(c),j})} \tag{3.14}$$

Which are the weights to compute the updated context:

$$h^{N+1}_{(c)} = \sum_j a_{(c),j} v_j \tag{3.15}$$

Once the context is updated, similarly to 3.1.4, it is finally possible to achieve the output logits:

$$u_{(c),j} = \begin{cases} Ctanh(\frac{q^t_{(c)} k_j}{\sqrt{d_k}}) \text{ if } j \neq \pi_{t'} \ \forall t' < t \\ -\infty \text{ otherwise;} \end{cases}$$

Where $C = 10$ is a factor to clip the results between 10 and -10. The probabilities of visiting the next node are therefore:

$$p_i = p_\theta(\pi_t = i|s, \pi_{1:t-s}) = \frac{e^{u_{(c),j}}}{\sum_j (u_{(c),j})} \tag{3.16}$$

3.1.4 well-describes the importance of keeping track of a context: the next decision is taken giving the current state s (i.e. topology) and the trajectory $\pi_{1:t-s}$ (which in the TSP is summarized with first and last node only).

According to the TSP constraints, each node can be visited once and only once: this mean that, once a node is chosen, it can no longer be in the following iterations. This is practically obtained preventing the visited nodes to take part in the context update (set $u_{(c),j} = -\infty$ for all nodes j already in the tour) and masking their chances to be picked for the following step (set $p_i = 0$ for all nodes j already in the tour).

Finally, it is important to mention that the whole model is trained using RL. Particularly, a gradient-descent policy-based solution is adopted (according to a REINFORCE [72] gradient estimator):

$$\nabla L(\theta|s) = \mathbb{E}_{(\pi|s)}[(L(\pi) - b(s))\nabla log p_\theta(\pi|s)] \tag{3.17}$$

Where $L(\pi)$ is the loss associated to the path found, $b(s)$ is the baseline, θ is the current parameter ruling the policy. For what regards the baseline, a *Rollout* algorithm, which deterministically picks the most probable value, is implemented: this, as aforementioned, is meant to give the model's sampling procedure a second term of comparison to enhance the learning.

3.1.5 The Gated GCN and a one-shot decoder

Like mentioned on Section 2.2.4, another way of approaching the GNN infrastructure is through convolutional tools to create the so called GCN (graph convolutional network). That was the choice for researchers in [16], which faced the TSP problem as well adopting the GGCN described on Section 2.2.4. Similarly to the GAT model chosen by [17], nodes become aware of their surroundings through a "message passing" procedure, in which the information about them is spread throughout the network using multiple-layer architectures. Anyway, while for the GAT we did not have any structural information about the topology except the for coordinates and possibly a neighbors mask, the GCN gates make explicit use of the edge features. Particularly, given as input edge feature a weighted adjacency matrix (so called *distance matrix*) containing values d_ij, they obtain:

$$\beta_{ij} = A_{edge} d_i j + b_{edge} || A_{knn} d_{ij}^{knn} \tag{3.18}$$

Where A_{edge} and b_{edge} are used to obtain a $\frac{h}{2}$-size projection of the edge distances and the concatenated A_{knn} to obtain the other $\frac{h}{2}$. Notice that using the *K-Nearest-Neighbors* aggregator is a common technique for the TSP, where nodes are usually connected with others in the close proximity. The embedding β_{ij} is fed to the GCN together with the node embeddings (which are obtained similarly to [17]). While the mathematical steps are described on Section 2.2.4, it is important to stress once more that the gating mechanism obtained as:

$$\sum_{j \to i} (\eta_{ij} \circ V^l h_j^l) \tag{3.19}$$

is the crucial difference on respect to [17]. While there the messages are weighted according to the *attention* between the nodes (the coefficient c_{ij} of 2.13 in the original Transformer model) and it is entirely up to the model to discover what are those weights referring to, here [16], using the edge notions, are explicitly applying the concept of distance as weights. In other words, with the model of [17] the nodes must contain some structural information to discover and learn the message passing weights; with the one of [16], this is not the case since those are obtained exploiting the topology itself.

The other difference between [17] and [16] is that the second uses a non-autoregressive decoder: this means that the model is basically one-shooting the complete solution without taking intermediate steps. Therefore, after applying the GNN encoder, the node embeddings are passed through a Multi-Layer Perceptron model such that:

$$p_{ij} = W_2(ReLU(\hat{W}_1(h_G||h_i^L||h_j^L))) \tag{3.20}$$

Where W_1 and W_2 are learnable projections, h_G is the graph embedding obtained similarly to 3.1.4, h_i^L and h_j^L are respectively the last layer embeddings of node i and node j. The result is an heat-map over the edges over which one can:

- Greedily pick the most probable link starting from a random node: the algorithm ends when all the nodes are visited.
- Apply a beam-search, which means maintaining in memory the best B links and choose a path only when all nodes are covered. The concept of "best" can be declined as most probable path or the shortest.

An example coming from [16]'s paper is presented on **Fig 3.4**.

Also for this section we finally discuss the learning procedure: for this article, a SL approach seemed the more appropriate since more *sample efficient*. This means it requires less training examples to get the same results as the RL counterpart as SL gets *complete information about the problem*, while the latter only a *sparse reward*. However, this does not come for free: as SL needs high-quality solutions to

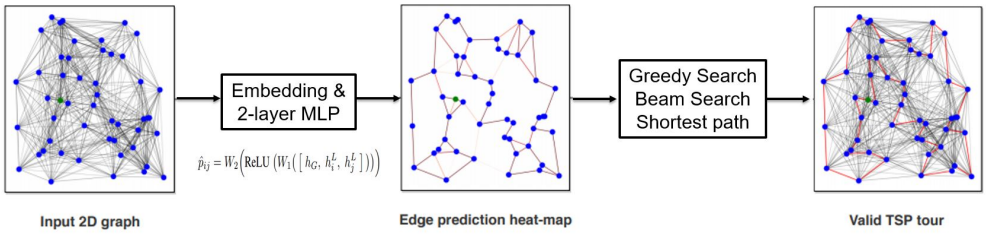


Figure 3.4: Non auto-regressive model for the TSP.

be trained, scaling to instances bigger then 100 nodes become practically infeasible since the exact solutions for such a high numerosity are often intractable to obtain.

3.1.6 A common benchmark

Very recently, [24] proposed an attempt to create a unified benchmark to test several design choices' generalization capacity. Understanding the importance of transferring the knowledge gained on smaller instances to larger ones, the researches collected most of the choices already cited in Section 3.1.4 and Section 3.1.5 to systematically check whether and how much each of them was able to provide the already mentioned inductive biases (Section 1.4). Beyond some interesting statements, like the greater generalization capacity of the GGCN or the predominance of autoregressive models over non autoregressive ones when scaling to larger instances, the key message of [24] is that *Learning the TSP requires rethinking generalization*. As it appears from this study, much work still have to be done and, most importantly, a new approach seeking for more general pattern rather than case-suited solutions must be preferred.

It is therefore with this attitude that we want to expand [16]'s work: while they attempted to scale "vertically", increasing the number of instances, we want to assess whether the latest state of the art tools, which we attempted to describe so far, are capable to move horizontally, namely from a combinatorial problem to another. Given a topology and some nodes feature, how difficult it is to transfer to other examples? And which is the cost in terms of gap with a case-specific solution like a tailored heuristic? Answering those questions and possibly introducing some novelties to enhance the transfer capacity is therefore the scope of this research.

3.2 Building blocks

As previously mentioned, this section describes the overall architecture which will be tested and implemented in the remaining of the paper. According to the

researches cited on Section 3.1, this will be an encoder-decoder model; the available options for the first will be faced on Section 3.2.1. Section 3.2.2 will on the other hand overlook the second.

3.2.1 Encoder pile

Layer 0 embeddings

Both [17] and [16] used a simple dense layer to move from a variable size input to a fixed size embedding. However, while [16] only faced the TSP, [17] also included the VRP and SDVRP, which as mentioned in Section 2.1.2 have slightly more complex initial features and actors involved. This second approach, which also generalize to the one-typology and one-feature only cases, is therefore more complete and is the one we present in the following.

First, the typologies must be handled separately aggregating their corresponding features:

$$x_{aggr1} = AGGR_{dim=1}(s_{j1}, d_{j1}) \tag{3.21}$$

$$x_{aggr2} = s_{j2} \tag{3.22}$$

Where j is a generic node, $1-2$ represent the first or second typology, $AGGR_{dim=1}$ is an aggregation operation (i.e. concatenation in the papers) performed on the first dimension, s_j the (x,y) coordinates and d_j the demand. Notice that, as **Fig 3.5(a)** testifies, the second dimension of both matrix is still of variable size (depends on the number of initial features).

The second step consists on obtaining a generic embedding h_j: first, each concatenated representation must be projected:

$$\hat{x_{aggr1}} = W_1(x_{aggr1}) + b_1 \tag{3.23}$$

$$\hat{x_{aggr2}} = W_2(x_{aggr2}) + b_2 \tag{3.24}$$

Where the two projections are different ($W_1 \neq W_2$ and $b_1 \neq b_2$) but projecting to the same hidden space $H_{dim} = 128$, which is the fixed-size mapping dimension. Eventually, another aggregation on dimension zero is computed to obtain:

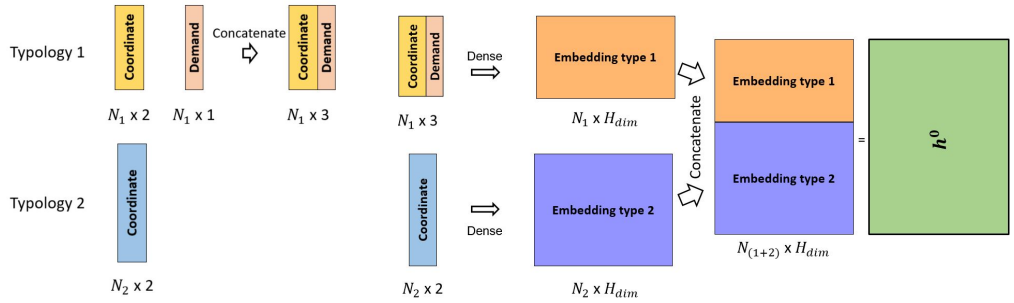
$$h_j = AGGR_{dim=0}(\hat{x_{aggr1}}, \hat{x_{aggr2}}) \tag{3.25}$$

Fig 3.5(b) completes the picture.

GNN encoder

At this point, the fixed-size projections of the input features obtained in 3.2.1 are fed to the GNN. It therefore is:

$$h_j^{l=0} = h_j \tag{3.26}$$



(a) Aggregating features for different typologies of nodes

(b) Obtain a generic embedding h_j from the two typologies

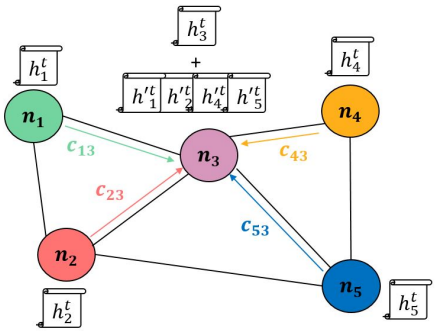
Figure 3.5: Encoder - Obtaining layer 0 embeddings

Where $l = 0$ represents the layer zero of the GNN multi-layer architecture and h_j the output of 3.2.1. The vectors are intuitively self-descriptive, which means that they each refer to only one node, and no notion of the surroundings has been added yet.

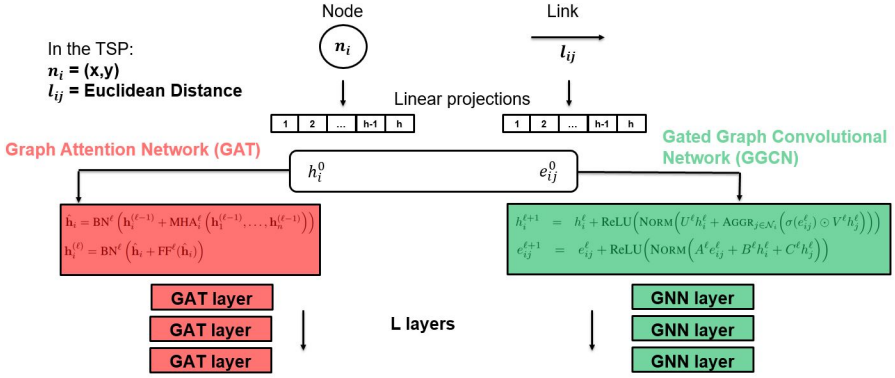
The following layers differs according to the chosen model: [16] uses a GGCN encoder, where the update functions are the ones described in Equation 2.22 and Equation 2.24. Research [17] on the other hand adopts a GAT encoder, which as aforementioned resembles the Transformer schema of Section 2.2.4 and uses the update function of Equation 2.16.

Without specifically reporting the formulas (the previous Section 2.2.4 and Section 2.2.4 already formally cover the topic), we think it might be useful for the reader to spend some more words on the comparison between a GNN architecture and a message-passing schema [73]. Basically, one could see the hidden state h_j of each node as a message and each layer as a round of message passing: at the beginning of each of round, every node sends its message to the connected neighbors (meaning that a link must exist between the sender and the receiver). In the most general case, the passage through the link involves some kind of smoothing/amplification of the message: the weighting coefficient can refer to a real physical quantity (i.e. smoothed according to the real distance separating the nodes or, on a Boolean manner, to the fact that the link itself exists or not) or to some learned relation between nodes (i.e. the attention mechanism). Eventually, the receiving node gets all the weighted-messages coming from the neighbors and somehow produces a synthesis which updates its previous state.

Each round r updates the $r - hop$ neighbors with new information: there is therefore no real stopping condition (no real limit for messages keeping flowing in the network), and the number of rounds (the number of layers) represents an hyper-parameter which must be carefully tuned.



(a) Message passing schema: each node j sends its hidden state (its own message) h_j to a receiving node. Those states are weighted by a link coefficient c_{ij} and eventually aggregated by the receiver h_i.



(b) The overall encoder schema with a TSP example: it is once more important to underline that while some models like the GGCN explicitly use a concept of "edge", others like the GAT have to learn the weighting coefficients looking a the nodes themselves.

Figure 3.6: Encoder pile and details on message passing

Fig 3.6(a) provides an example of message passing on respect to a single receiver; **Fig 3.6(b)** on the other hand summarize the entire encoder architecture according to the available choices.

3.2.2 Decoder pile

Both versions presented by [16] and [17], namely the non-sequential and sequential decoder, receive as input the final-layer embeddings $h_j^{l=L}$ from the encoder and construct the so called graph summary $\hat{h_G}$. Intuitively, this should give the model

a global view of the topology synthesizing its key elements; obtaining a good representation at this point, especially for the non-sequential approach where practically nothing else is given (Formula 3.1.5), is vital. On respect to 3.1.4, which only provided the average as aggregator, [24] tests more versions including *sum*, *max* and *avg* itself to assess the one providing more inductive biases.

The sequential approach starts the decoding loop at this point, whereas the non-sequential approach needs nothing else since the probability matrix is already generated. This, as already discussed on Section 3.1.4, consists on creating a context like in 3.1.4 and then on running a self-attention mechanism between it and the other nodes. The context should summarize the information about the previous steps (i.e."where did we get so far"), while the self-attention mechanism should conjugate the past trajectory with the remaining nodes (i.e. "since we are here, which is the next to visit?"). Also on this case, a probability matrix is then emitted to choose the following step. **Fig 3.7** provides a toy-example.

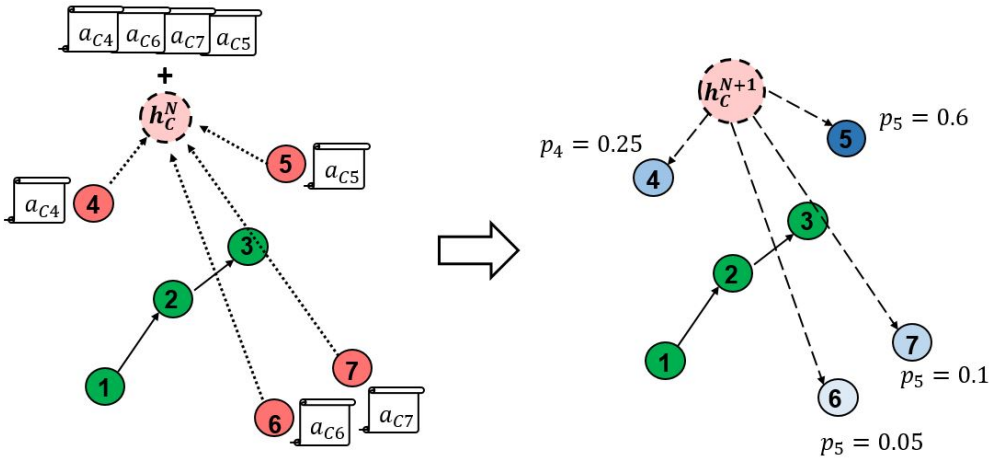


Figure 3.7: Sequential decoder's choices: once the context node is updated with the self-attention mechanism, it should be aware of 1) which is the trajectory so far (till node 3 in the example) 2) which node should be visited next according to the topology (node 5 is the most probable in the example)

About the decoding loop is also important to notice that, both for the TSP and the VRP, the graph and nodes embeddings represents "fixed elements" that can be computed only once before entering the loop. For example, in the TSP the nodes do not change when the agent choose an action: its coordinates remains the same, regardless on it having been visited or not. In practise, this means that no re-reembedding, namely recomputing the embeddings, is needed once an action is taken. Since this might not be the most general case facing a combinatorial problem, [17], which also faces the SDVRP version of the VRP, partly faced the issue: as underlined in Section 2.1.2, the SDVRP accepts residual demands, meaning that a

node can be visited multiple times without having to satisfy its demand in one-shot. This clearly creates a more dynamic context to handle, which they faced adjusting the decoder self-attention mechanism of 3.1.4 into:

$$q = W^q h_{(c)} \tag{3.27}$$

$$k_i = W^k h_i + W_d^k \hat{\delta}_{i,t} \tag{3.28}$$

$$v_i = W^v h_i + W_d^V \hat{\delta}_{i,t} \tag{3.29}$$

$$\tag{3.30}$$

Where the information about the residual demand $\hat{\delta}_{i,t}$ is added to the keys and values together with the node embeddings; on such a way, the dynamic information is not added to the initial set of features (which would lead to the need of re-embedding) but only during the decoding phase to "complete" the context.

Chapter 4

Transferring the problem

This section's goal is describing, from a theoretical point of view, which are the main differences between the problems faced so far (i.e. the TSP) and other combinatorial problems; how those differences impact the existing architecture.

Particularly, we will discuss more in details about the relationship between nodes and edge features, properly defining the aforementioned *structural information* a combinatorial problem yields. The latter indeed represent vital aspects which, as we will lately assert, current GNN models are yet incapable of fully capturing with severe limitation for the knowledge transferring.

Subsequently, we will also distinguish *static* features from *dynamic* ones and show that, while previous models always faced cases in which some static element (i.e. coordinates) was present to uniquely identify nodes, this is not the most general case. We will also demonstrate how the absence of static identifiers deeply impact the decoder's effectiveness eventually worsening the overall results.

Thereby, we will finally contextualize the recent DE technique as a solution for both problems: embedding the distances as novel nodes features can indeed solve the lack of structural information, providing in the meantime static information for the nodes themselves.

4.1 Structural information

4.1.1 In theory

According to the description provided on Section 2.1, a graph is characterized by two main components: those are the *nodes* v_i and the connecting *edges* e_{ij}. While the first represent single entities and their characteristics (i.e. demand of a store; polarity of a molecule; number of contacts of a social user), the second embed their relationships in terms of distances (i.e. distance between shops), typology of link

(i.e. type of boundary connecting the atoms) or even, in Boolean terms, whether they are connected at all (friends or not in the social).

Intuitively, for some real-life examples, one can infer notions about the relations only looking at the nodes themselves: for instance, if in the case of geographical distances between objects the coordinates are provided, estimating how distant two nodes are is relatively an easy task. For some other case, that estimation is still possible but slightly more complex: for example, given the properties of two molecules, it might not be that easy to guess how and whether they are connected. Finally, for some problem this is not possible at all: in the power and channel assignment problem, it is generally unfeasible to recover on real time the exact STA's position, and the only available information is how distant that is in terms of signal received by the APs. Therefore, like pictured by **Fig 4.1**, for those cases relying on nodes features only is not an option, and edges information gather vital importance. More generally, we can assert that, for those cases, edges represent the only element providing information about the structure of the problem: without them, one cannot obtain a full picture since the topology itself is not well represented.

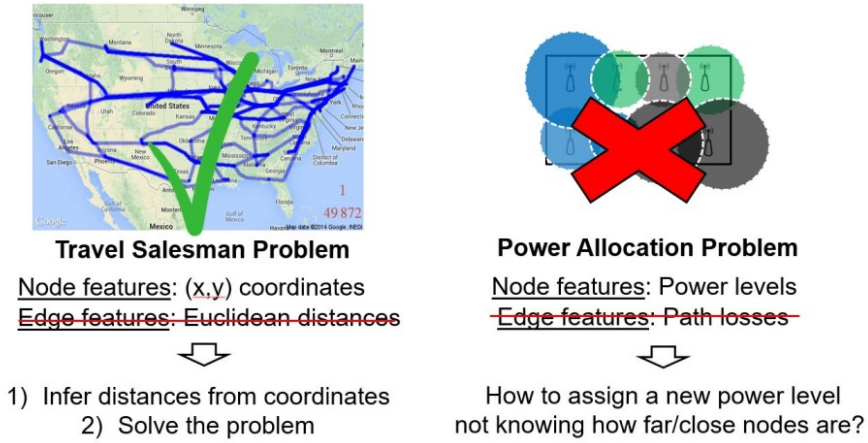


Figure 4.1: What would happen removing edges information from the TSP and from the power allocation problem

As we will prove on Section 5.2.1, the existing architectures, which have been tested on problems with strong structural information already as node features (both the TSP and the VRP make use of coordinates), fail or under-perform when those are available only as edge ones and cannot be inferred using nodes. This of course represents a severe limitation facing graphs contexts, where the topology is a crucial element, and a major setback for generalization purposes.

4.1.2 In practise

To demonstrate the importance of structural information, it might be of help looking back at the encoder proposals of Section 3.2.1 and face a toy-example in which no nodes features are provided and the information only relies on the edges. As we will later explain, despite seeming an extreme example, it is actually not so far from the power allocation case we want to face.

The first issue arises from the fact that both the GGCN and the GAT architectures requires initial nodes embeddings h_i^0 as inputs, while our toy-example, with no nodes features to apply the formula of 3.2.1, does not have any. The simplest way to overcome the problem and start the encoding pile is therefore using a common padded vector (i.e. a vector of ones) $\tilde{x_i}$ as a placeholder in place of the missing features x_i:

$$h_i^0 = W(\tilde{x_i}) + b \tag{4.1}$$

At this point, the GAT option is already hopeless: as the theory of Section 2.2.4 suggests, not including the edge features and with the attention mechanism focusing only on meaningless representations (placeholder's projections), there are no chances it can learn anything and emit good embeddings h_i^L. Therefore, more in general, it is very unlikely that the model can obtain good performances.

Something more, thanks to the explicit use of edges on the weighted message passing, can be on the other hand obtained with the GGCN counterpart. With a very practical intuition, it is like having people repeating the same sentence in a room: each of them, even not knowing the positions of the others, can roughly estimate it according to how much the voice is smoothed by the distance. According to the process described in Section 2.2.4, with the toy-example case, the placeholders' projection departing from each node are smoothed using the learnable edge embeddings as gates:

$$\text{incoming message} \propto \sum_{j \to i} \eta_{ij} \circ V^l h_j^l \tag{4.2}$$

Where $V^l h_j^l$ are the messages departing from neighbors and η_{ij} the edge-dependant gating-weigh factor. Like **Fig 4.2** visually exemplifies, even if starting with the initial same projections h_i^0, each of them is modified by greater or smaller quantities according to the closeness on respect to the other nodes; on such a way, the final representation h_i^L might hopefully be different for each node and representative of its surroundings.

However, while obtaining representative embeddings on this way is theoretically possible, in practise, as the results of Section 5.2.5 will demonstrate, the structural information are too weak and the model under-perform on respect to examples in which coordinates are provided.

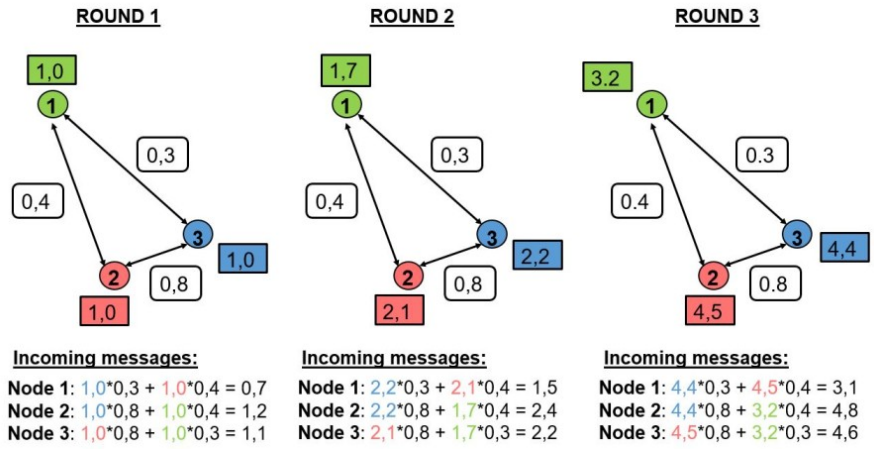


Figure 4.2: An oversimplified toy-example (with fictitious numbers) of GGCN encoder when only edge features are available. Already at round 3 the algorithm has produced three different embeddings h_i^3: notice that h_1^3 is the smallest (node 1 is further from the others) while h_2^3 and h_3^3 are similar (and so are node 2 and node 3 structurally).

4.2 Static and dynamic features

4.2.1 In theory

On Section 2.2.1 we have described how, in RL, an agent interacts with the environment committing actions which modify its states. If we contextualize this learning paradigm for a combinatorial problem, the best-fitting entity to represent the environment is clearly the graph itself: however, not all the edges and nodes features might be part of the state the agent can modify. For example, in the TSP the nodes are definitively not shifted because their coordinates are not modified: it is the agent travelling across the graph, and therefore for such a problem it is the agent's position the state's best candidate. In other words, we could say that coordinates in the TSP are *static features*, which are not modified even if the agent picks a new action. On respect to Section 4.1 previously discussed, it is interesting to notice that, since for combinatorial problems the topology is generally unchanged, most of the structural information represent static elements.

As presented on Section 3.1.3, something slightly more complex occurs for the VRP: there, since the agent's aim is satisfying the nodes requests, their *demand* represents a dynamic element which can be zeroed after a model's action (i.e. visit that node). Same happens in Section 3.1.4 facing the SDVRP, where the demand feature is split in *original demand* (static) and *current demand* (dynamic). However, static information in terms of coordinates exist on both cases and is

therefore possible to:

1. Obtain fixed representations h_i^L of the problem, which are immutable and actions-independent.

2. In the decoding phase, add, mostly on a sequential manner, a new "context" representing the current state of the environment (i.e. providing information about the current demands).

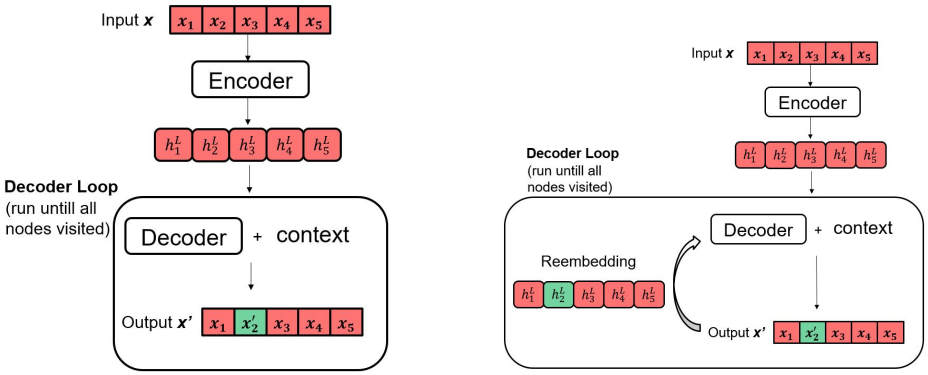
Unfortunately, it is not a general assumption to always have at disposal some static node features. A practical example comes from the power and channel allocation problem: as we will discuss on Section 4.4, on its simplest (and most general) formulation, the nodes are described only by the power and channel information. Those, as the name of the problem itself suggests, also represent the elements which the agent's actions will modify to ensure its objectives (the ones listed on Section 2.1.3). Powers and channels are hence dynamic features, and no fixed embeddings can be obtained with them. Existing architectures, which generally use the encoding pile only once, fails at tackling those cases; dynamic features therefore represent another issue that the current models have partly or not faced yet.

It is finally important to underline that the previous considerations are valid only for sequential approaches. Those, like the ones of Section 3.1.4, are models in which the agent takes one decision at the time until all the nodes are visited. For non-sequential approaches on the other hand, like the ones of Section 3.1.5, all the decisions are taken together and defining static and dynamic features is useless. However, since the first, according to [24], provide more inductive biases and are the more adapt for generalization, it is important to underline which possible findings are limiting their performances.

4.2.2 In practise

Similarly to Section 4.1.2, it could be of help facing a toy-example using the state-of-the-art encoder-decoder infrastructures proposed by the literature. This time, nodes will be represented by dynamic features only: this means that the agent, picking an action and modifying the state, will also modify the initial input.

In the first instance, it is clear that computing the embedding only once, like previous infrastructures did, is no longer possible: as **Fig 4.3(a)** testifies, once that the first change is applied to the original state, the agent is practically facing a different problem. In fact, the original input vector x, which generated the embeddings h_i^L used to pick the first decision a_1, has changed to x': this means that the encoder's representation h_i^L are no longer valid and new ones, as **Fig 4.3(b)**, must be generated re-embedding the problem.



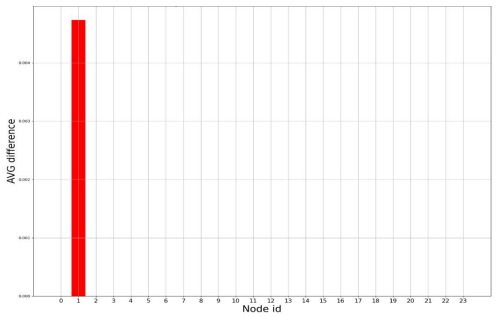
(a) Handling dynamic features as previous models: since a node, after the first round of decoding loop, is modified, the input embeddings h_i^L are no more consistent.

(b) How to handle dynamic features: once an action modifies a node, re-embedding is applied to obtain consistent embeddings for the next decoding iteration.

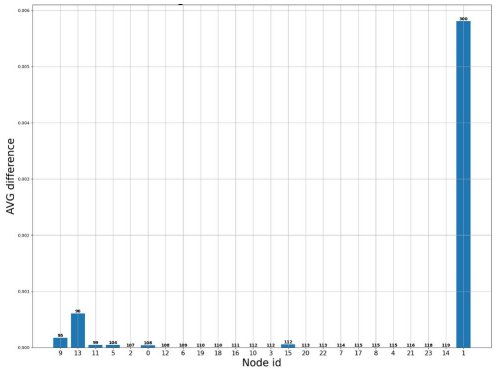
Figure 4.3: Handling dynamic features: only the first round of decoding loop (decision about one node only) is shown for clarity. Notice that, in case of static input features, the agent would modify the context and not the initial features (no re-embedding would be needed).

Furthermore, even re-embedding during the decoding loop might not be enough for not "poisoning" the decoder. Suppose that, similarly to [17] and [24] for the TSP, the embeddings of the last two visited nodes are used as a decoding context. Imagine, for the sake of clarity, that we have already picked decisions about three nodes and that our current context is therefore $[h_4^L, h_6^L]$, being h_4^L and h_6^L the last visited and $L = 2$ the number of encoding layers. We now choose that our next decision regards node 1: since we are facing a problem with dynamic features only and according to what was explained before, our action changes node 1 initial features from x_1 to x_1' and re-embedding must be therefore applied to maintain consistent embeddings. However, since using a GNN encoder, the modification occurred to node 1 propagates through the network also affecting the surroundings (**Fig 4.4** provides an example from the power assignment problem). Intuitively, it is like throwing a rock in the pond: the immediate perturbations only regards a point, but the waves soon propagates.

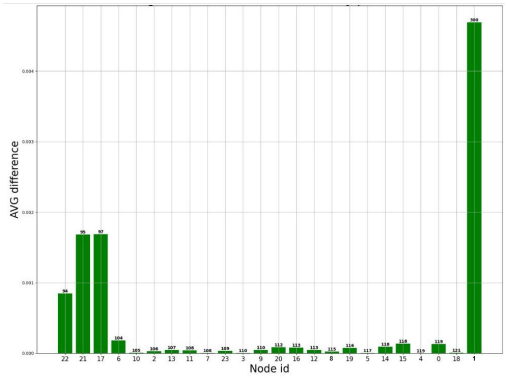
This means that the new context after the decision on x_1 is $[h_6'^L, h_1^L]$ with $h_6'^L \neq h_6^L$ since also affected by the re-embedding. **Fig 4.5** might help visualizing the concept: if the nodes embeddings are not fixed, the decoder's context, which should help the model keeping track the sequence of decisions, is no longer of help. This will also be proved on Section 5.2.1, where we will demonstrate that, facing the power allocation problem, the decoder worsen the results instead of improving



(a) Average difference of the embeddings before entering the GNN: only the changing node is involved.



(b) Average difference of the embeddings after one layer of GNN: besides node 1 itself (higher bar), change has propagated and mainly affected the closest neighbors



(c) Average difference of the embeddings after two layers of GNN: change has propagated further, including nodes not involved in the first round.

Figure 4.4: Example coming from the power allocation problem: AP 1 is reassigned to a new power and, using a 2-layer GNN encoder, the 'novelty' is spread across the network. It is important to notice that the node ids in the x-axis are sorted with increasing order according to the path-loss distance from node 1 (node 1 is the last one since its distance to itself is set to infinity), while the y-axis represent the average difference, computed across the 128-dimensional embedding vectors, before and after the change.

them.

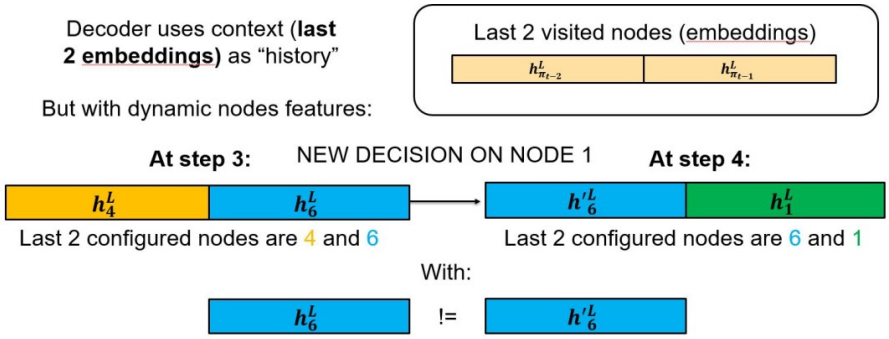


Figure 4.5: Decoder issue with re-embedding: since every embedding is involved, the contexts as history is no longer of help since not coherent throughout the decoding loop.

4.3 Distance Encoder: a multi-purposes solution

Summarizing the issues faced so far, we stated that:

- Current encoding architectures are unable to represent problems unless strong structural information are provided in the form of nodes features.
- Current encoder-decoder schemes only holds if static information are provided.

Clearly, both of them represent limiting hypothesis which prevent transferring to more general combinatorial problems or severely limit the model's effectiveness. This is why, on Section 2.2.5, we presented the recent DE (distance encoder) technique proposed by the researches of [66] and [67]. As already mentioned, this tool basically tries performing an edge-node transformation: in fact, like **Fig 4.6** testifies, given a node, it first aggregates all the information about its connected edges; then, it creates a novel node feature, also descriptive of its contour. On mathematical terms, being N the number of nodes, it reduces a NxN distance matrix into a $Nx1$ vector of features.

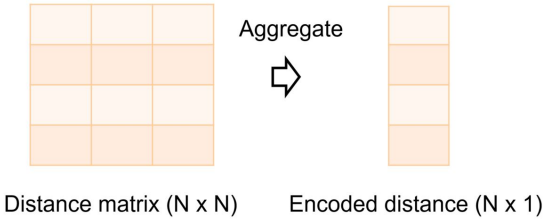


Figure 4.6: Distance Encoding edge-node transformation

If we consider the example of **Fig 4.7**, a possible implementation could be

averaging all the edges contributions: in the example's case, an high value would represent an isolated node; a low value, a central one.

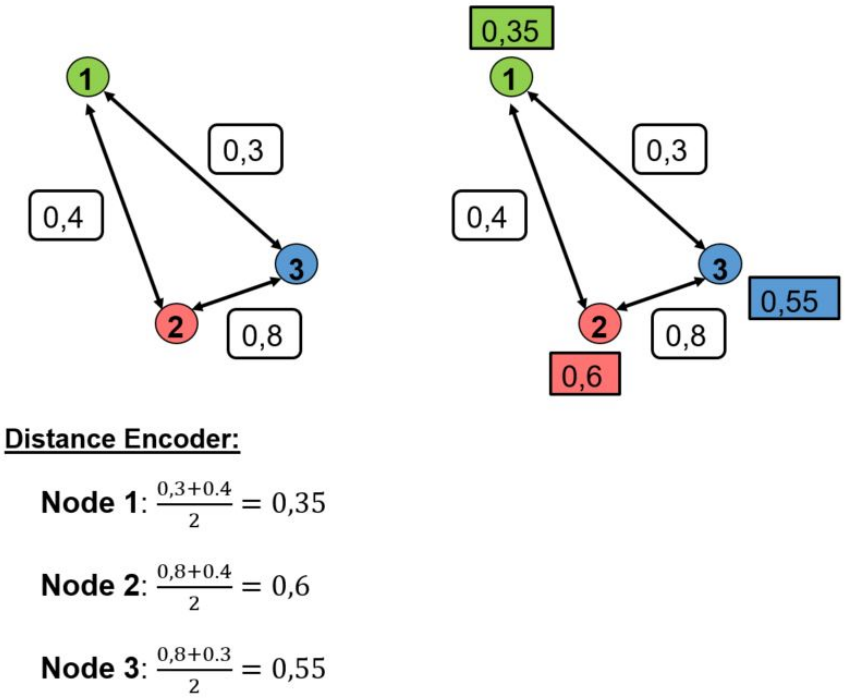


Figure 4.7: Distance Encoding over-simplified example: new nodes information have been obtained exploiting the knowledge on the edges

Since such an easy application is generally not enough for improving the learning, usually the edges features are at least linearly transformed like 4.3 before applying the aggregator:

$$x_{DE} = AGGR(W * x) \tag{4.3}$$

Being W a learnable projection.

If we consider again the generalization impediment of Section 4.1, we can now observe that with DE all the graph notions are eventually exploited: in fact, while with previous architectures edges were not taken into account at all by the GAT and were only used as gates by the GGCN, now they can be used to construct new nodes representation reinforcing the structural information on the problem. When facing cases with no nodes information, the GAT becomes again an option since nodes, with the new feature provided by DE, can finally "help" learning about the topology. Furthermore, also the GGCN gets strengthened since the

nodes themselves gain importance: the use of weighted message passing, which with placeholder represented the only opportunity for the model to learn starting from a dummy initialization, now becomes again only an optional enhancement.

On the other hand, referring to the concern of Section 4.2, DE can produce a static per-node identifier. As we previously mention, it is often the case that structural information are also static ones: this is also true for DE, which is basically able to create novel action-independent features. Like in the work of [17], it is therefore possible to first use the new static information to build the encoded representation of the problem; then, with fixed and action-independent embeddings, add the dynamic information as a context for the decisions. On such a way, the proposal of **Fig 4.3(b)** evolves into **Fig 4.8**, solving the decoder's inconsistencies of **Fig 4.5** while improving the overall speed of the algorithm (no need for re-embedding).

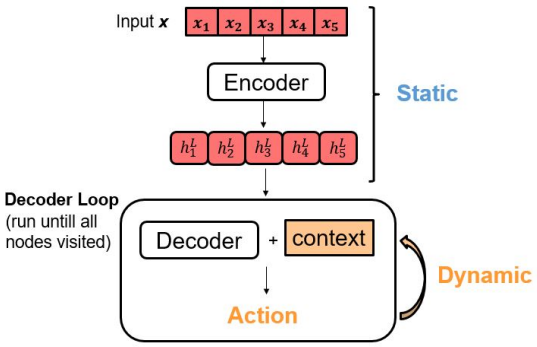


Figure 4.8: Encoder-Decoder scheme when using DE to obtain static nodes features. The encoder is action-independent and therefore no re-embedding is required at the end of each decoding iteration.

It is important to conclude mentioning that while the role of DE on improving the structural representation of the problem will be object of Section 5.2.3 analysis, its capability of solving the encoder-decoder malfunctions when no static information are available currently represents a qualitative evaluation only. Proving this second statements might represent an important step for future analysis.

4.4 A practical use-case: the power allocation problem

This section is meant to introduce the use-case problem we want to try transferring the state-of-the-art techniques on. This is the power and channel allocation problem, already mentioned on Section 2.1.3: particularly, without loss of generality for the

purpose of the research, we face a sub-case which only takes into account the power levels' configuration. As we demonstrate, this problem is particularly adapt to prove the limits of current architecture: no structural information are provided in terms of nodes features; it comprehends dynamic elements only.

The following Section 4.4.1 and Section 4.4.2 describe its inputs and outputs, while the in-between architecture is the one described on Section 3.2 and is summarized on **Fig 4.9**. Notice that, being the most promising, only the *sequential approach* of [17] is being tested.

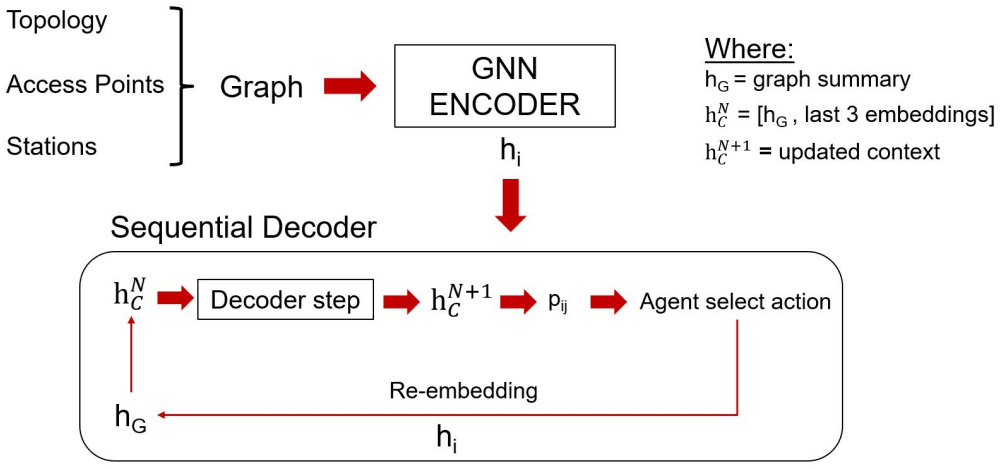


Figure 4.9: Overall architecture for the power allocation problem: notice that, as already present on Section 4.4.1, since the nodes features are dynamic-only, re-embedding during the decoding phase is necessary.

On respect to the research of [17] we also maintained the same learning approach (RL) and the same baseline (Rollout technique).

Section 4.4.3 finally provides a more technical overview of the model's tree. This is meant to show the reader an high-level picture of the "flow" of passages performed by the encoder-decoder architecture when facing the power allocation problem.

4.4.1 Input problem

Like introduced on Section 2.1.3, the power management problem consists on allocating radio resources in wireless networks. Its aim is finding the best APs' configuration to optimally serve a set of STAs while reducing the possible interference.

The two typologies of nodes, the serving and served ones respectively, are therefore AP (access point) and STA (station). Those, on the problem's simplest version, are stationary: this means that, throughout an episode, neither the APs nor the STAs will change their location.

About positioning, like previously mentioned, for this problem not having at disposal the nodes coordinates is generally a realistic assumption: on a real-world scenario, while the APs placements are generally given, the real-time users (STAs) positions in the space are unknown. Anyway, another measure of distance exists in the field of wireless networks: the path-loss (or *path-attenuation*). This calculates the power density reduction of the electromagnetic wave propagating through the space. In practise, according to the definition given on Section 4.1, path-losses are the equivalent of Euclidean distances for the TSP and represent the nodes' relationships: it is therefore coherent to use them as edge features. Still, like **Fig 4.10** schematizes, it is only feasible to evaluate the AP-AP and the AP-STA path-losses, while the STA-STA are generally not available: this is not a problem since, having to configure the APs power decisions according to the STAs placements, it is not that important for a user to know where the others are (i.e. STA-STA measurements). The unknown quantities are therefore padded with very high values of path-losses (300 dB) as if they were not connected at all: the same is done for each AP on respect to itself (no self-links are allowed).

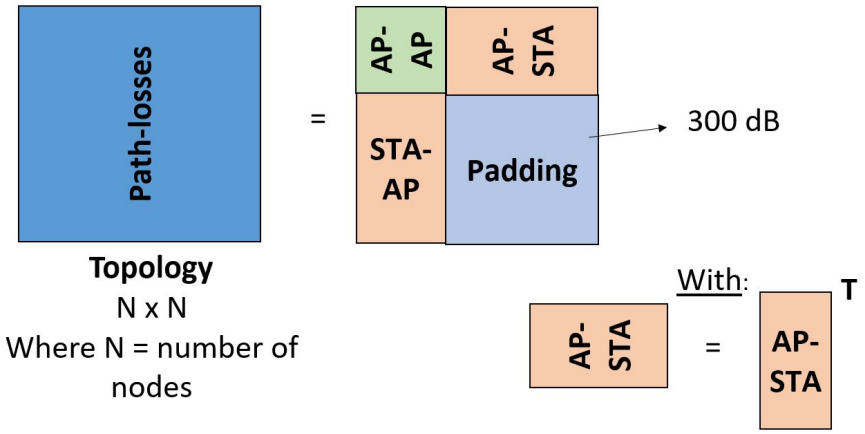


Figure 4.10: Path-losses: notice that those also represent the graph weighted topology.

For what regards the nodes features on the other hand, we know that each AP is characterized by its current power level. Those powers, like explained on Section 4.2, also represent the environment's state the agent interacts with: this means that, having no other static feature, APs are represented by a dynamic feature only. STAs are on the other hand more complicated. They indeed constitute crucial elements of the problem since power levels are adjusted according to their placements; however, we do not know their physical positions and there are no other meaningful characteristic to represent them. Like in the toy-example of Section 4.1.2, all the (structural) information only lays on the edges (path-losses),

while there is nothing describing each STA on its own: therefore, exactly like in that example, placeholders will be used to adopt current architectures. Intuitively, this is the equivalent of recognizing there is a non-omissible entity (STA) which however, with the current techniques, we cannot represent. **Fig 4.11** eventually summarizes the problem's input.

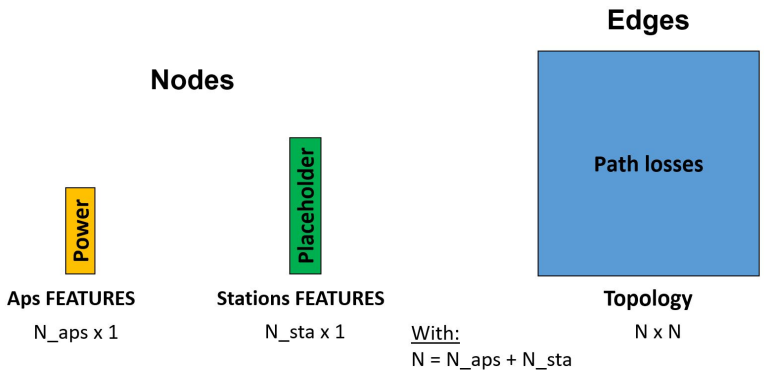


Figure 4.11: Input features for the power allocation problem: notice that $N =$ number of nodes, while N_{aps} and N_{sta} are the number of APs and STAs respectively

4.4.2 Output problem

Another important difference of the power allocation problem on respect to the TSP regards the desired output. While, according to the formulation presented in 3.1.4, the TSP objective is picking decisions on which node to visit next, the power allocation problem requires something slightly more complex. In fact, at each round of the decoding loop, the agent has to pick both a new AP to configure and one of the available power levels. Therefore, instead of the N-dimensional vector of probabilities p_i of the TSP, where N represents the number of nodes, we actually need a $N_{ap} \times P$ matrix of probabilities $p_{i,j}$, where P represents the number of available power levels and N_{ap} the APs numerosity (only nodes involved in the decision process). **Fig 4.12** provides a visual representation.

It is important to notice that, as it was happening for the TSP (Section 3.1.4), once a node is "visited" it can no longer be modified in the following iterations. This means that, if an AP has already been configured, it cannot be modified anymore in the remaining of the decoding loop. In practise, for the following loops:

- It will not take part in the decoding self-attention mechanism since its compatibility with the context is set to $-\infty$
- Its chances of being chosen by the decoder are zeroed.

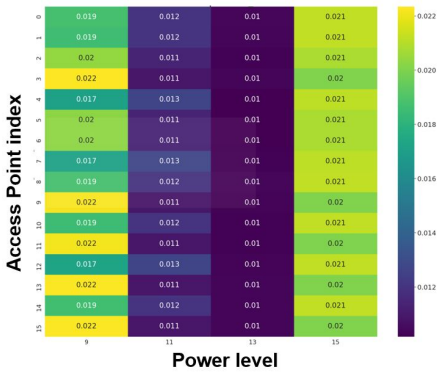


Figure 4.12: Heatmap of the matrix of probabilities $p_{i,j}$. Notice that the sum over the elements of the entire matrix must return 1 according to the law of total probabilities.

4.4.3 An overview of the model's tree

Like aforementioned, this section, with the aid of a series of images, aims at providing the reader a more in-depth overview of the encoder-decoder model applied to the power allocation problem. Particularly, exploiting the back-propagation flow of the gradient, it is an attempt of showing the practical sequence of passages performed by the machine from the input data to the final output. Notice that all the following images are only meant to provide an high-level overview of the overall architecture: it is therefore non-necessary for the reader to focus on the single boxes reporting the passages described in the sections before (i.e. sum of vectors; divisions...).

Fig 4.13 shows the main methods to obtain the h_0 embeddings from the input data. Like explained on Section 3.2.1, two linear projections $W(x) + b$ are applied; W represents the *weights* while b the *biases*. Their outputs are the APs and STAs projections respectively; those are finally concatenated to obtain a unique vector h_0.

Fig 4.14 on the other hand shows the first GGCN layer. The left side of the tree concerns the nodes manipulation: the receiving and the sending nodes are obtained trough linear projections U and V. The right side on the other hand elaborates the edge-embeddings for the edge-gating mechanism: particularly, according to what explained on Section 2.2.4, the edge-embeddings are updated using notions of the receiving and sending nodes (projections A and B respectively) and the ones of the edges themselves (C projection). Those gates are then used for the weighted message passing: a *max* aggregator is used to obtain a synthesis of the received messages. Both nodes and edges are finally normalized before moving to the next layer.

Fig 4.15 completes the picture about the encoder, providing a visualization of

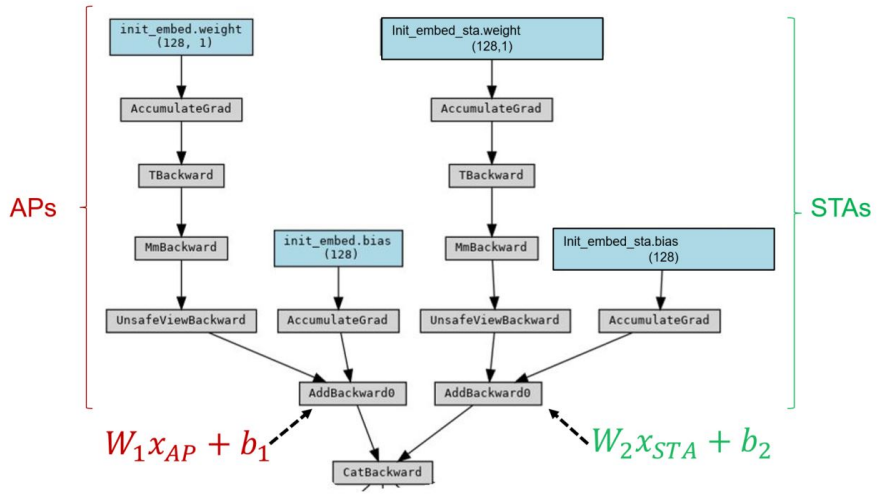


Figure 4.13: How to obtain the h_0 embeddings.

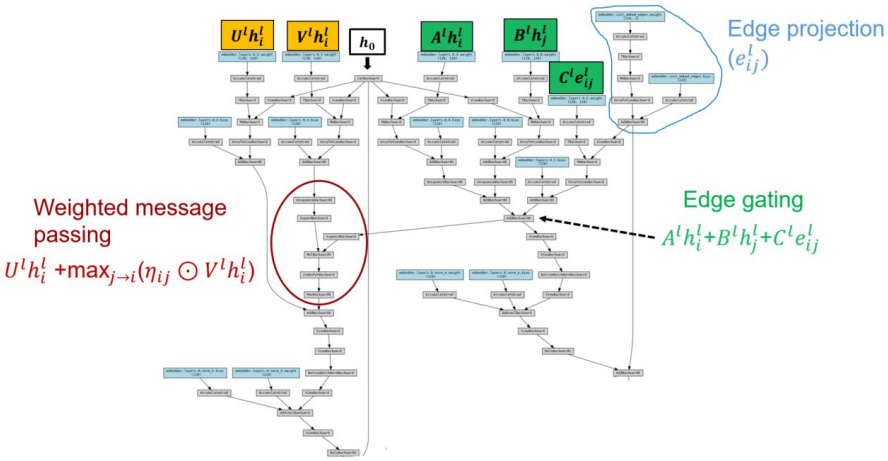


Figure 4.14: Encoder detail: How to obtain the h_1 embeddings.

the three GGCN layers. Notice that those are equal but do not share weights (each of them is trained separately).

Finally, **Fig 4.16** shows the decoder tree; without going again into the details already explained on Section 2.2.4, it is interesting to notice how, facing a case with four APs, four decoding branches are clearly visible. Those are justified by the use of a sequential decoder, which requires to create a new context every time a new decision is picked. Notice that the tree becomes larger when more APs have to be configured (we do not report a larger example for graphical reasons).

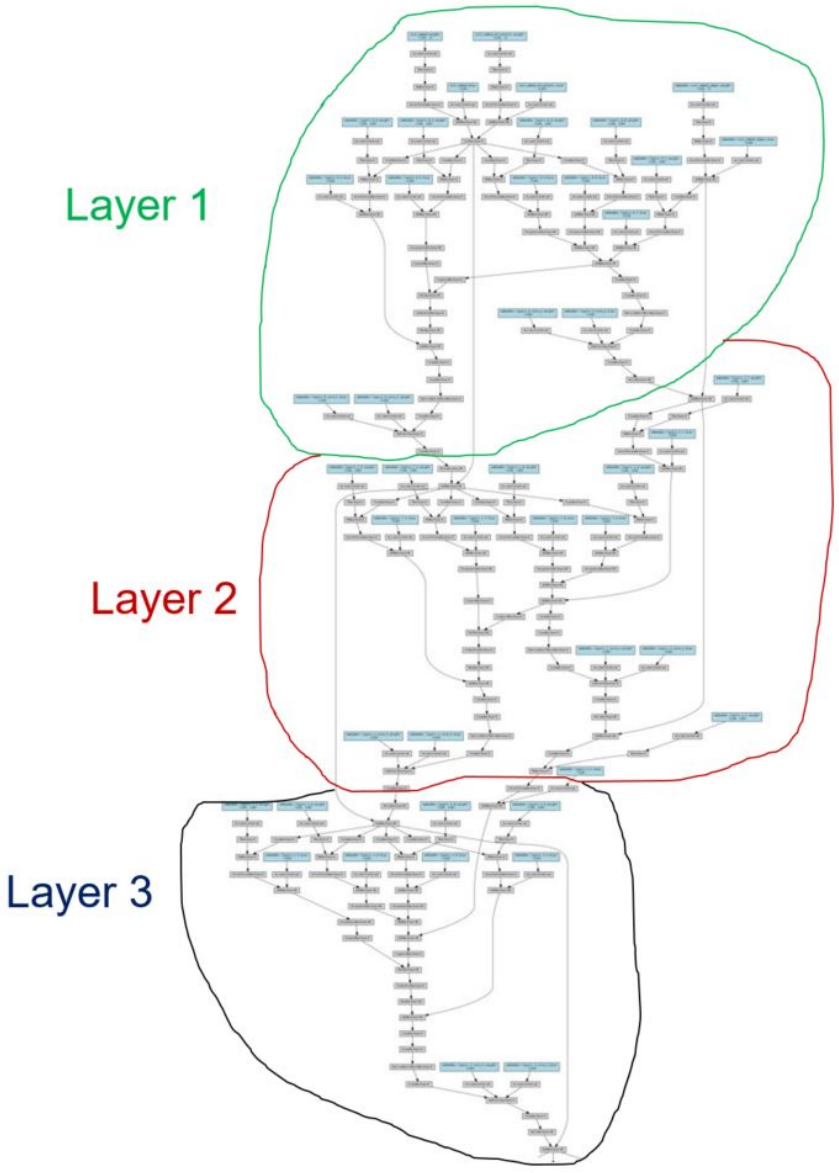


Figure 4.15: Encoder full picture: three identical layers having the structure of **Fig 4.14**. Again, it is not important for the reader to focus on the single boxes, while it is the overall picture which matters.

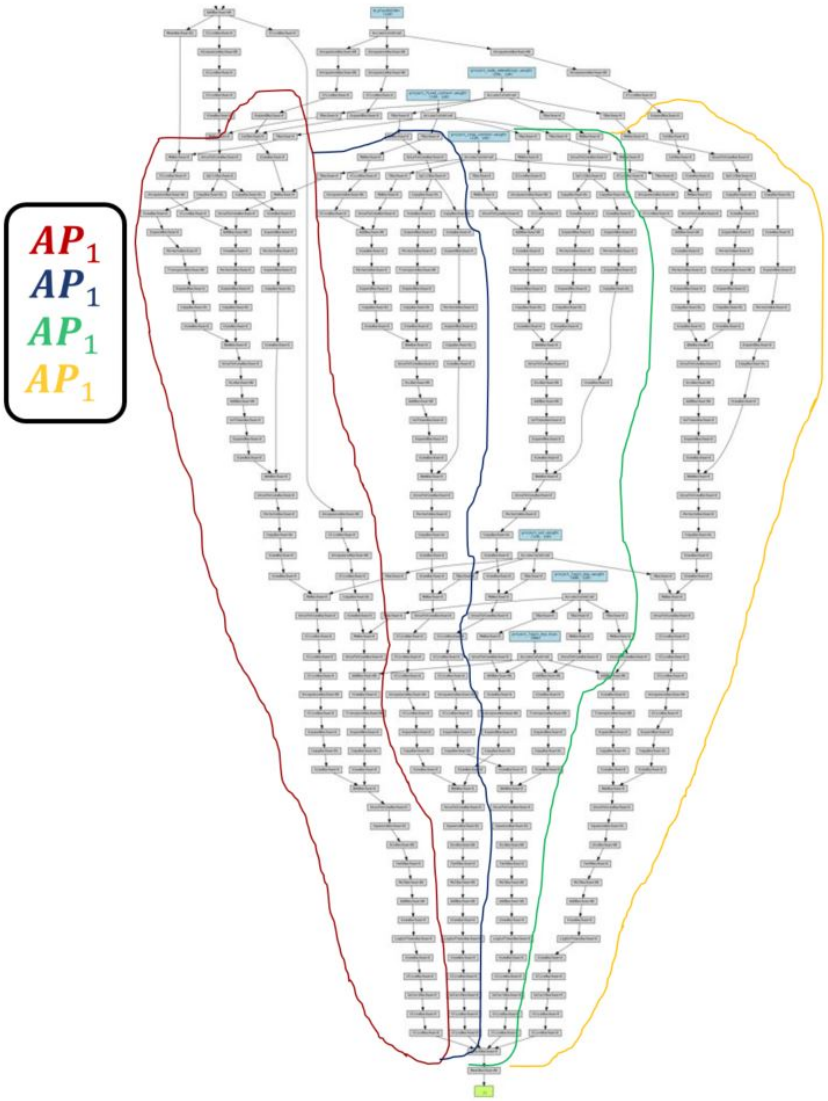


Figure 4.16: Decoder full picture: four branches are visible and correspond to the four APs to configure.

Chapter 5

Results and findings

This chapter reports the experimental results of the thesis.

Section 5.1 introduces some preliminary concepts: it presents the evaluator, a crucial element for RL; explains how the dataset are generated; spends some words on the optimal results and on the heuristic used when the datasets were too large; describes the so called *Multisampling*, a technique used during the testing phase; establishes which metric are adopted for evaluation.

The following Section 5.2.1 on the other hand shows the results when applying the best choices for the TSP to other problems such as the power allocation problem; those empirically testifies the issue presented on Section 4.2 concerning the decoder's utility when handling dynamic features only.

Section 5.2.2 reinforces the concept: with an architecture relying on the encoder only, the decoder's issues are solved and the results get better.

Section 5.2.3 refers to the second issue stated on Section 4.1 about structural information. Particularly, with two examples of Section 5.2.3 and Section 5.2.3, it proves that those are generally useful for enhancing the model's comprehension of the problem. Furthermore, Section 5.2.5 demonstrates that this also extends back to the TSP case: if we remove the coordinates, the model which makes use of DE performs better than the one with traditional gating system only.

5.1 Before starting

Like aforementioned, the following subsections list and briefly describe some introductory key concepts before moving to the obtained results. The first one, 5.1.1, introduces the evaluator and explains how it is used.

5.1.1 On the evaluator

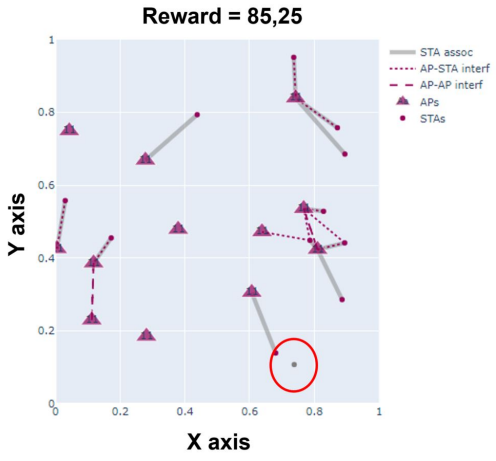
On Section 2.2.1 we stated that RL's learning is guided by rewards or regrets. This means that it must exists an entity "judging" the model's behaviour and discouraging "bad actions": that is the so called evaluator. Referring to the overall objectives, on the power allocation problem we generally do not want to increase the chances to interfere; we want to cover all the demanding STAs; we do not want to overload an AP. The reward U the model gets from its actions is therefore:

$$U \propto \frac{1}{|STA \in AP|} \text{ (\textbf{AP load})} \tag{5.1}$$

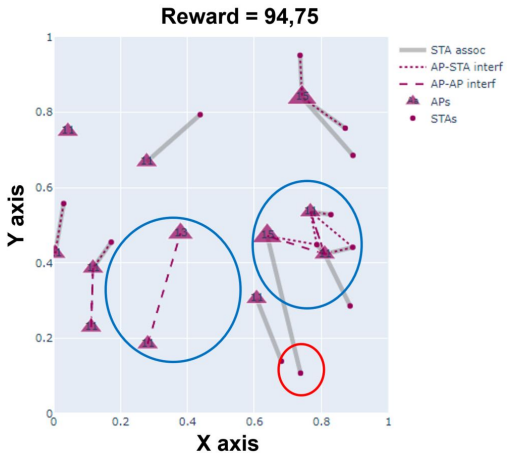
$$U \propto signalstrenght(AP) \text{ (\textbf{Received power})} \tag{5.2}$$

$$U \propto \frac{1}{\sum_{i \in Interfere} |STA \in i|} \text{ (\textbf{Interference})} \tag{5.3}$$

Fig 5.1(a) and **Fig 5.1(b)** also provides a use-case which might clarify the judging criteria.



(a) A practical example: the model has adapted a conservative behaviour to limit the interference and all the APs have been set to a medium-low power level (range from 9 to 15). Anyhow, one STA (red circle) was left uncovered.



(b) Same use-case of 5.1(a). Guided by the rewards, the model has now picked also some higher choices. Despite an increase of the overall interference (blue circles), all the STAs are now covered and the reward has greatly improved.

At the end of each decoding loop, when all the APs have been reconfigured, the evaluator is queried and emits a score rating the current environment's state. If this, according to what explained on Section 2.2.2, is positive, it encourages that model's behaviour increasing the probabilities of the picked actions; otherwise, those are penalized not to be picked again in the future.

Notice that, for a matter of consistency with the TSP original code, which uses the path-lengths as regrets and aims at minimizing them, in practise we also use $regret = -reward$ to penalize bad behaviours.

5.1.2 On the datasets

For what regards the adopted dataset, it is important to specify that this first approach to the problem only makes use of simulated data. This implies that, for creating the datasets, we also need a simulator accepting parameters to vary the environment's configuration in terms of number of APs, number of STAs per AP and number of neighbors.

These settings require wise and balanced choices. In fact, creating a very dense environment would over-stress the importance of interference; consequentially, the model would be induced to mostly assign low power preferences reducing the overall noise. On the other hand, sparser environments are expected to present many high power assignments to cover isolated nodes. Of course, both cases are feasible and could exist on a real-world scenario: however, they clearly represent extreme cases and, more importantly, they both accept "easy solutions". The latter are particularly unwanted since they might complicate our capabilities of evaluating the model's performances: unfortunately, even an architecture completely unable to represent the problem (i.e. unaware of the distance between nodes) can learn to set everything to the minimum power level available.

As shown on **Fig 5.1**, another characteristic which the simulator accepts as a parameter is how to geometrically dispose the nodes: the available options involve *Poisson Point Process* (PPP), *Square* and *Hexagonal*. It is generally a good choice to dispose the STAs according to a PPP so that the uncertainty regarding the users position is represented by the random process; on the other hand, placing the APs allows more freedom of choosing and there are no particular reasoms for preferring one option over the others.

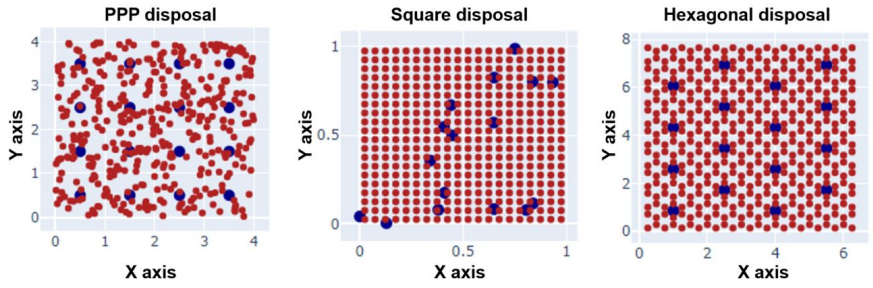


Figure 5.1: Example showing the possible nodes disposals; here, the STAs are displayed according to a PPP; on a square grid; hexagonally on respect to the APs.

Finally, as it is of common use in the field of ML, the simulator can handle batches: those allow the model to face multiple problem instances in parallel eventually speeding up the learning. Batches of size 32 were chosen for this task.

Table 5.1 reports the configurations which will be discussed in the following results sections.

Config name	AP number	STA per AP	Number neighbors	Placement	Available power levels
9sq_7pl	9	6	3	Square	[9,10,11,12,13,14,15]
14ppp_4pl	14	2	3	PPP	[9,11,13,15]
16hex_4pl	16	2	3	Hexagonal	[9,11,13,15]
25ppp_7pl	25	3	4	PPP	[9,10,11,12,13,14,15]
28ppp_4pl	28	2	3	PPP	[9,11,13,15]
32ppp_7pl	32	12	6	PPP	[9,10,11,12,13,14,15]

Table 5.1: Configs' parameters

5.1.3 On the optimum and on the heuristic

This subsection describes how the exact or approximated solutions for the configs of Table 5.1 were obtained. Notice that those are practically useful for validating purposes; in fact, metrics such as the average optimality gap can be computed only with the optimal solution (or at least an approximation) at disposal. On the other hand, as mentioned on Section 2.2.1, they are not necessary during training since adopting a RL approach.

The exact results were obtained through complete enumeration: basically, a script brute-forces all the possible combinations of power assignment and eventually keep the best of them. Of course, this is extremely costly in terms of time resources (scales exponentially with the numerosity) and, on our case, was affordable only for the configs with 9, 14 an 16 APs. This is why a *local search* heuristic was also used to approximate the results for the others. Particularly, PowerLS is a heuristic that outputs a locally optimal power configuration with respect to the reward function. Given an initial configuration, PowerHeur varies the power level for each AP separately within a specified range, while keeping the remaining configuration constant. The best individual power level for each AP, together with its overall reward gain, are recorded. After an iteration over all APs, the best improvement is compared with a simultaneous change of all APs into their individual directions, and the process is repeated for the best configuration found. If no further improvement can be achieved, the algorithm terminates.

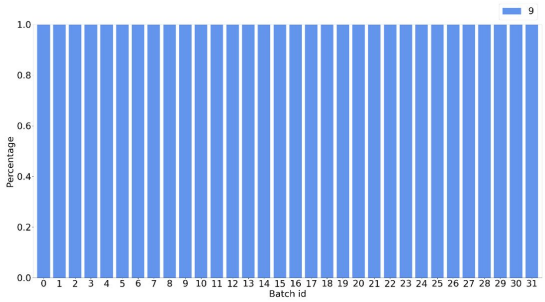
5.1.4 On Multisampling

Like explained on Section 4.4.2, every decoding loop emits a matrix containing the probabilities p_{ij} of choosing an AP i and a power level j; the decision is then taken

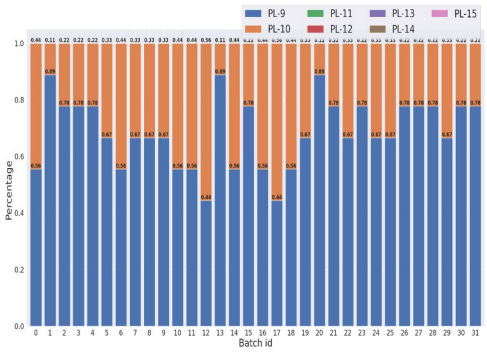
sampling from the available options.

Since the process involves a random process, during evaluation it might be helpful to allow multiple samplings: the idea is to repeat the decoding loop several times and eventually only keep the best combination. On such a way we guarantee the model more chances to explore the solution space and coming up with better solutions.

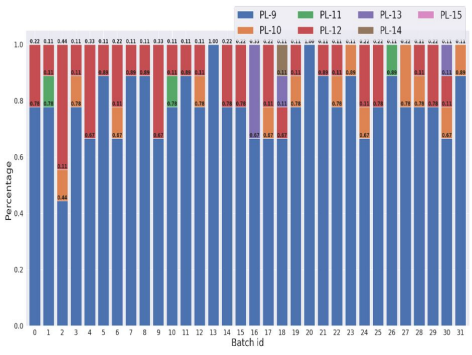
Fig 5.2 shows, in the power allocation problem framework, what happens in practise picking decisions greedily (most probable); allowing 10 samples; allowing 100 samples. Interestingly, as **Fig 5.2(a)** testifies, the model is confident on choices that minimize the overall interference; the most probable guesses in fact consist on setting all the APs to the minimum power level available. If we start sampling and allow 10 attempts, **Fig 5.2(b)** demonstrates that the model gets more "intrepid" and also picks more various combinations. Finally, letting the model pick 100 solutions like on **Fig 5.2(c)**, the solutions get very diversified since also less probable actions are chosen.



(a) Greedy behaviour: model is very conservative on limiting the overall interference.



(b) 10 attempts allowed: the solutions are more diverse since the model is also picking less probable actions.



(c) 100 attempts allowed: the solutions' space is much more explored and more power levels are eventually picked.

Figure 5.2: Greedy vs Multisampling

Fig 5.3 on the other hand practically anticipates how results are affected adopting one policy on respect to another. This highly depends on which model is actually used: the encoder-decoder version of the GGCN for instance, which is the state-of-the-art model presented by researches in Section 3.1.6, tends to produce probability-matrix very peaked around a particular choice. This basically means that, no matter of how many samples are allowed, the solution is always the same because the same actions are always taken. On the other hand, an encoder-only GGCN version using DE, which we will present on Section 5.2.3, highly benefits from having more attempts at disposal.

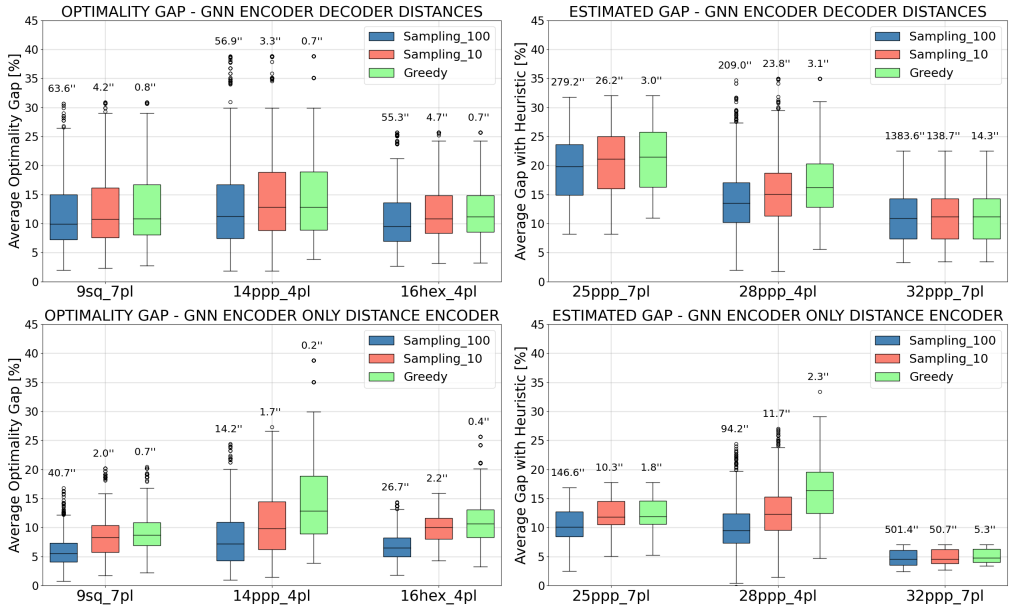


Figure 5.3: Optimality gap on configs adopting a greedy policy; a 10-samples policy; a 100 samples policy. Results highly depends on typology of model adopted.

5.1.5 On the metrics

It is finally important to introduce the metric used to validate the results. Notice that each attempt (model with a particular setting) was run with different seeds both during training and during test in order to assess its robustness: more in details, there were 5 training seeds and 10 validation seeds. This leads to:

$$|\text{Result per attempt}| = |\text{Seed training}| * |\text{Seed validation}| * \text{Batch size} \tag{5.4}$$

$$1600 = 10 * 5 * 32 \tag{5.5}$$

Which also means:

$$|\text{Result per attempt per batch}| = |\text{Seed training}| * |\text{Seed validation}| \tag{5.6}$$

$$50 = 10 * 5 \tag{5.7}$$

The two metrics we therefore adopt are:

1. **Optimality Gap / Estimated Optimality Gap**: basically, each batch result is compared to its exact/approximated solution in order to obtain a gap:

$$gap_i = \frac{\hat{x_i} - x_i*}{x_i*} \tag{5.8}$$

   where $\hat{x_i}$ is the model's solution for batch i and x_i* the exact/approximated one. A box-plot containing the gaps for all the batches and seeds is then plotted: an higher box implies an higher gap (and therefore a worse performance). On some occasions, a *random decision maker* is also used to test how the proposed alternatives behave on respect to random solutions; for fairness on respect to the multisampling, each batch gets $|\textit{Seed training}| * |\textit{Seed validation}| = 50$ random guesses and only the best is kept.

2. **Inter-attempt difference**: used for a more detailed comparison between two attempts. It measures whether one configuration is better than the other, and is obtained subtracting the model's result for the first attempt ($\hat{x_{i,1}}$) to the one of the second ($\hat{x_{i,2}}$):

$$diff = \hat{x_{i,1}} - \hat{x_{i,2}} \tag{5.9}$$

   A box-plot containing the differences for all the batches and seeds is then plotted: if the box is around zero, the difference is marginal. An interval of confidence around the mean is also computed to assure that the difference is significant (i.e. one model is in fact better than the other).

5.2 The results

It is now time to discuss the thesis results. The first sections can be interpreted as the *pars destruens*: in fact, as shown on Section 5.2.1, transferring with minimum changes results in a failure since previous architectures do not solve the issues stated on Section 4.1 and Section 4.2 regarding structural information and dynamic feature. On Section 5.2.2 we therefore start a simplification process to systematically analyse which elements are really beneficial for the problem's representation; the conclusion

is that the decoder's context is of no help, confirming the theoretical statements of Section 4.2.

The remaining sections on the other hand constitute the so called *pars costruens* and aim at fixing the structural information problem of Section 4.1. This is meant to create more robust basis for future works, finally enhancing the graph representation.

5.2.1 Transferring with minimum changes: Encoder-Decoder architecture

As anticipated, the first attempt consists on trying transferring the auto-regressive architecture proposed on Section 3.1.6 with the minimum changes required by the different problem. Those have already been cited on Section 4.4 and mainly regards the different inputs and outputs of the latter.

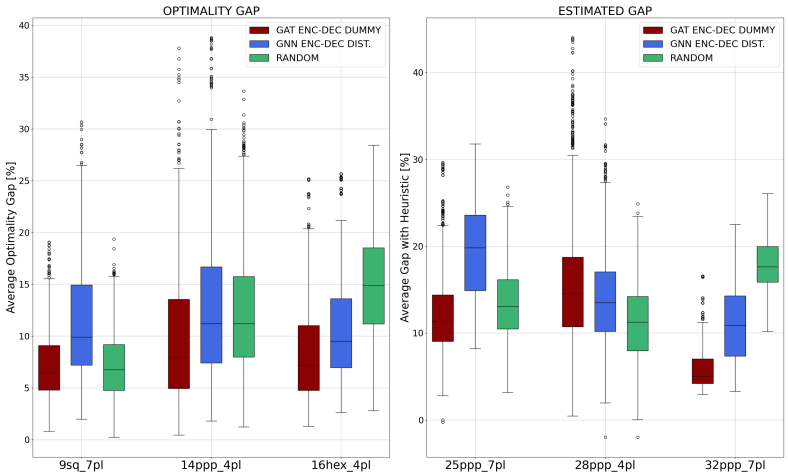


Figure 5.4: Optimality gaps of an encoder-decoder GAT architecture; of an encoder-decoder GGCN one; of a random decision maker

Fig 5.4 shows the optimality gaps of the two encoder-decoder architectures (one using a GAT encoder; the other a GGCN) and of a random decision maker on the configuration files of Table 5.1. Notice that we added the labels *dummy* and *dist.* (distance) to recall that while the GGCN uses the latter for the weighted message passing procedure, this is not the case for the GAT which therefore has no knowledge of the APs and STAs position.

Only in two of them, the 16*hex_4pl* and the 32*ppp_7pl*, the proposed models behave better than random; for other two, the 9*sq_7pl* and the 28*ppp_4pl*, both models even do worse. This is clearly a symptom of malfunctioning and testifies that neither architectures have been able to get a meaningful representation of the

problems.

If we just compare the GAT and GGCN results as **Fig 5.5** then, we can also notice that the GAT counterpart is obtaining better results for approximately all configs. This is again counter-intuitive if we refer to our statements of Section 2.2.4: the attention-based mechanism of the GAT encoder in fact, contrarily to the GGCN one, does not make use of edge features which, in the power allocation problem, embed the only information about the nodes' distances. Therefore, this suggests that either neither models are able to handle distances or that actually some architectural element is confusing and "poisoning" the remaining of the work; the following Section 5.2.2 further analyses whether it is this second case.

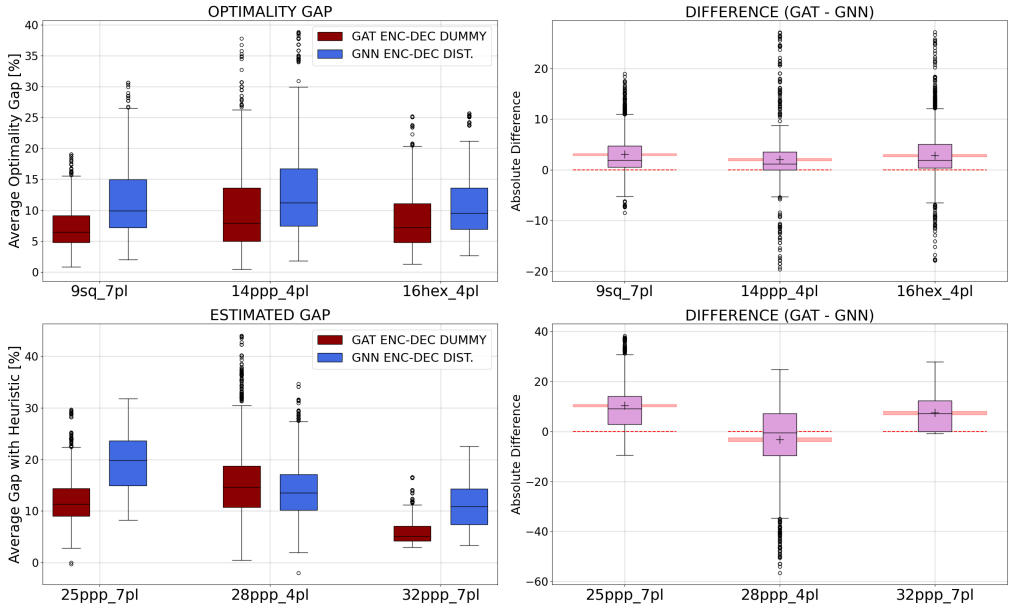


Figure 5.5: On the left, optimallity gaps of an encoder-decoder GAT architecture and of an encoder-decoder GGCN one; on the right, their inter-attempt difference.

5.2.2 Is it a decoding issue?

Considering the negative results of Section 5.2.1, it clearly appears that the current state-of-the-art architectures are unable to capture the essential aspects of the problem. It is therefore useful to systematically analyse which components are in fact helping the representation. Since, from the statements of Section 4.2, we know that the decoder might have issues with a problem yielding dynamic features only, start working with an *Encoder only* model seems to be a good idea. This implies that, instead of applying the self-attention mechanism described on Section 3.1.4, for these attempts we obtain the probabilities of choices directly from the nodes

embeddings.

Notice that, in addition to the considerations of Section 4.2, it is also reasonable to point out that, for the power allocation problem, the context is indeed already included in the nodes embedding. In fact, since re-embedding is needed every time an AP is modified, each decision is already taken on an updated environment: hence, a specific passage to revise the nodes embeddings, as it was required for the TSP, appears to be redundant. What might be still missing with an Encoder-only version is an "history" summarizing the previous choices: however, to solve the issue of Section 4.2, this seemed like a necessary simplification.

Fig 5.6 shows the results comparing the best encoder-decoder architecture of Section 5.2.1 with the GAT and GGCN Encoder-only versions. It clearly appears that, in general, removing the decoder is beneficial for all models: all configs, except for the 32*ppp_7pl*, exhibit improvements.

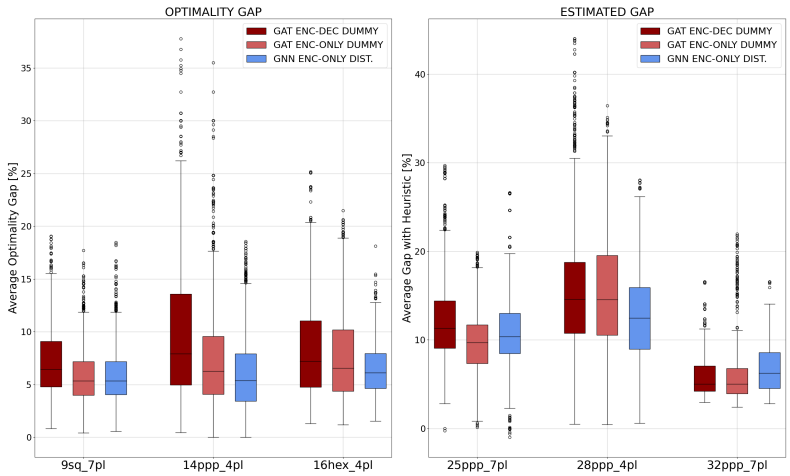


Figure 5.6: Results showing the best encoder-decoder architecture so far against two encoder-only versions. Those are eventually able to outperform the first on all cases except one.

It is still unclear why the GAT version sometimes achieves results comparable to those of the GGCN. Intuitively, although the GGCN explicitly uses distances in the edge-gating mechanism, this could still not be sufficient on a scenario where nodes features are still predominant, but not meaningful enough. The following Section 5.2.3 will therefore analyse whether anything better can be achieved enhancing their structural information.

5.2.3 Reinforcing the structural information

This section represents the beginning of the "pars costruens", namely an attempt to reinforce the representation power of the model providing a path for future researches. With the simplified architecture of Section 5.2.2, we want the machine to autonomously capture the essential aspects of the problem: this, on the first instance, means a model both able to handle nodes and edges information, which therefore does not require the boost of strong but not general features like coordinates.

Section 5.2.3 starts with a cheating attempt: would the performances improve if coordinates were at disposal? This basically aims at assessing whether a structural knowledge comparable to the TSP one would be beneficial for the learning.

Section 5.2.3 replaces the coordinates with encoded distances exploiting the DE mechanism of Section 2.2.5: this provides an instrument possibly suiting all problems with a graph representation, making the first step toward a solution for the current architectures lacks.

A cheating attempt

As aforementioned, we firstly want to assess the importance of structural information testing the model in the same conditions of the TSP. Since we are not using real data, the simulator can deceitfully provide the APs and STAs coordinates together with the others information.

Fig 5.7 shows the results comparing a "dummy" GAT encoder, a GAT version using coordinates and a GGCN encoder with distances for the usual message-weighting procedure.

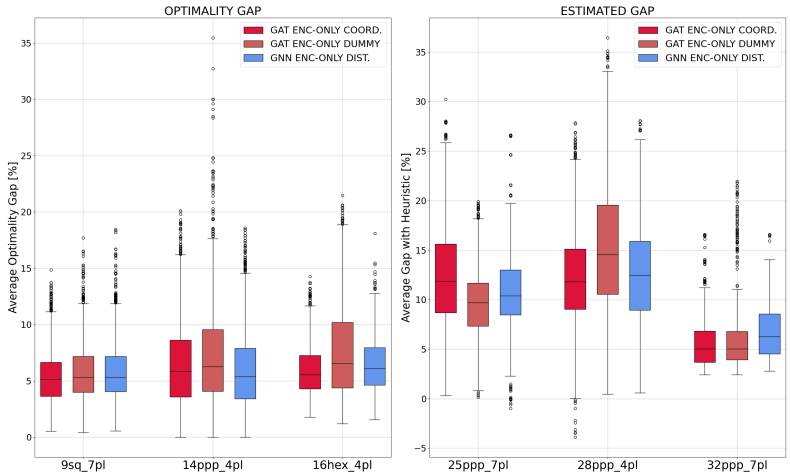


Figure 5.7: Results showing the best encoder-decoder architecture so far against two encoder-only versions. Those are eventually able to outperform the first on all cases except one.

Clearly, coordinates are beneficial for all configs but one, both reducing the results median and standard deviation. This confirms the statements of Section 4.1 and demonstrate that for graph problems being able to locate nodes is fundamental for improving the overall performances.

Applying Distance Encoding

Unfortunately, the real world is not a simulator we can query for more information: the power allocation problem lacks the nodes coordinates and therefore requires alternatives.

Particularly, DE might be a valid one: with every node getting a new feature encoding the distances on respect to the others, it could be that the model is eventually able to exploit the topology information so far unused. **Fig 5.8** visually provides the comparison between the GGCN encoder with distances, the GAT with coordinates of **Fig 5.7** and two GGCN encoder using the different versions of DE presented in Section 2.2.5.

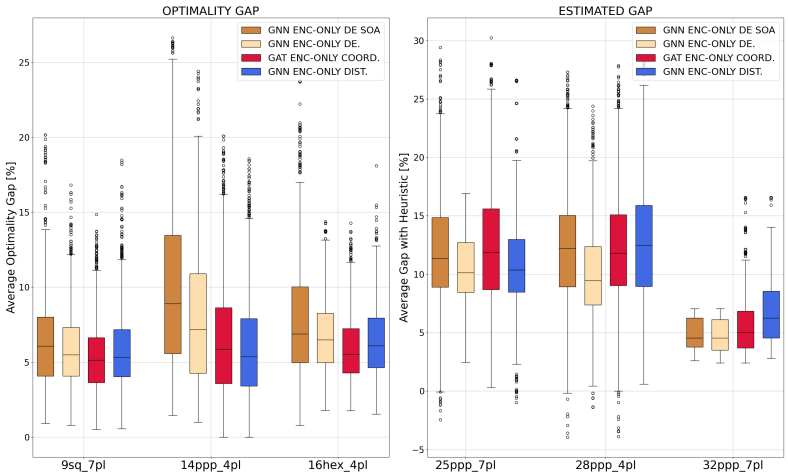


Figure 5.8: Two DE attempts against the GAT encoder with coordinates and a GGCN encoder using distances. The revised DE version successfully outperforms the other versions on bigger configs; on the other hand, it performs comparably to the GGCN encoder for smaller ones.

While the State-Of-the-Art counterpart generally bad-behaves, the revised one obtains results comparable to the classical GGCN with distances on smaller configs and outperforms the other options for larger ones.

If we do not consider the GAT versions with coordinates, which cannot fairly be implemented, **Fig 5.9** demonstrated that the revised DE attempt represents our best option. In fact, it successfully reduces the variance on smaller configs; it both reduces the variance and the average behaviour on bigger ones.

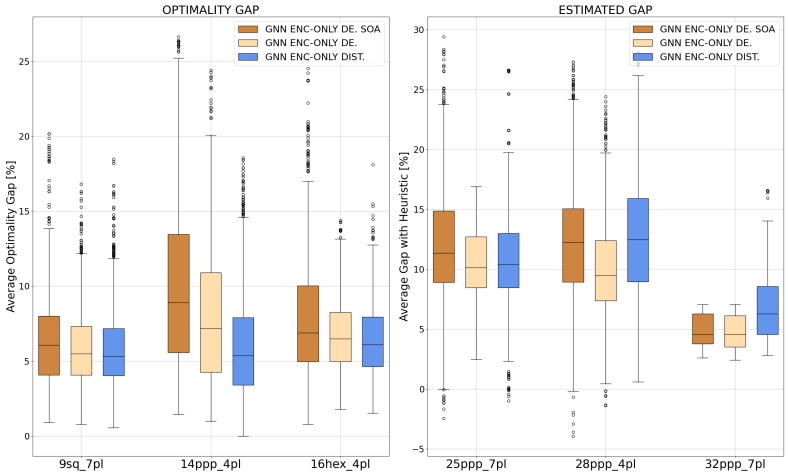


Figure 5.9: If we remove the GAT encoder with coordinates, the revised GGCN with encoded distances represents our best guess.

Finally, if we now compare, as in Section 5.2.1, a random decision maker with the best models we finally achieved, we can also assess that this is always over-performed by the latter. This is in fact proved on **Fig 5.10**.

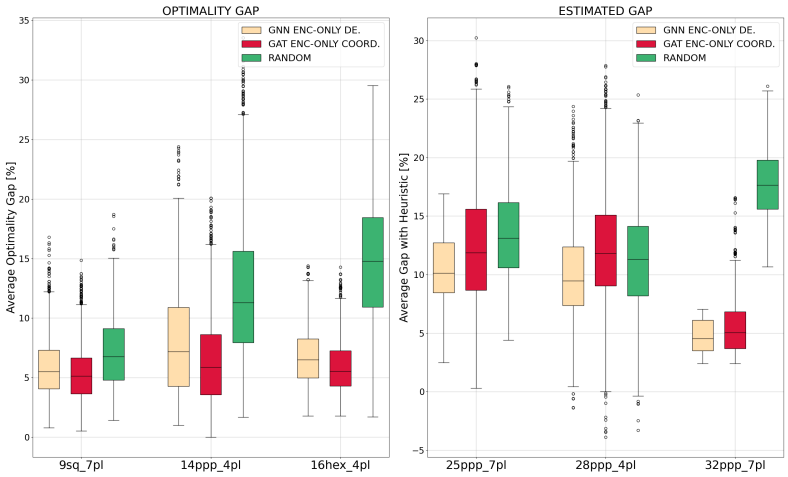


Figure 5.10: The random decision maker is always outperformed by the revised DE architecture

5.2.4 Summing up the results

It is eventually useful to present a joint plot summarizing all the obtained results: this is represented on **Fig 5.11**, where each bar represents the average gap value

across all attempts (grouping batches and seeds) and each line its standard deviation.

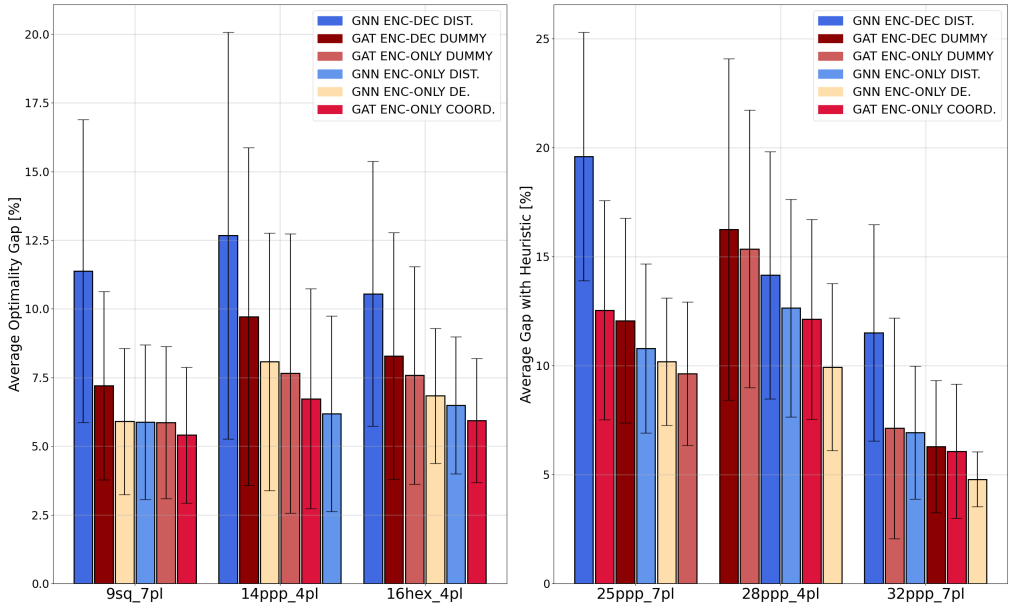


Figure 5.11: Ordered attempts for each configuration file.

As it can be observed, despite some exceptions which might be worth deepening in future researches, there is a common trend. Both versions with an encoder-decoder architecture generally bad-behave and occupy the last step of the ranking; next, the dummy GAT encoder-only version, which poorly behaves since incapable of fully exploiting the graph notions; finally, the first three positions are usually occupied by the encoder-only versions of the GAT with coordinates, of the GGCN with distances and of the GGCN with DE.

5.2.5 Can it transfer back to the Travel Salesman Problem?

Section 5.2.3 confirmed that enhancing the structural information of problems suffering the issue of Section 4.1 improves the model's overall performances. Furthermore, we stated that DE is a generic tool, valid for all types of graph problems. This section therefore try "transferring back" those concepts to the TSP problem: if there we remove the nodes coordinates and only rely on DE to replace them, how do the results are affected?

Fig 5.12 presents a comparison between the original GGCN with coordinates, a GGCN and a GAT versions using DE, a GGCN with distances to weight the messages and finally a "dummy" GAT.

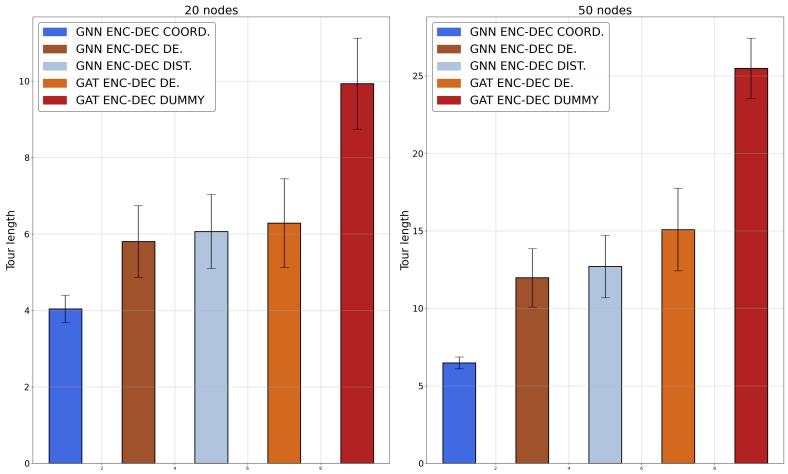


Figure 5.12: "Transferring back" to the TSP

According to the theoretical statements of previous sections, the "dummy" GAT version, which makes no use of the topology notions and cannot use coordinates, does not stand a chance against the others. However, reinforcing the structural information immediately boosts up the results: in fact the GAT version using DE almost halves the TSP path-length.

Interestingly, all GGCN versions perform better than this last attempt. This is quite intuitive, since they all consider edges info in the weighted message passing between nodes; as also observed in the research of [24], this is beneficial in graphs contexts where the attention mechanism of the GAT is of lesser effects. Furthermore, it is proved that the DE version of the GGCN is capable of outperforming the distance-only counterpart also for the TSP. This testifies once more the benefits of enhancing the structural information of the nodes embeddings. Finally, the comparison between the GGCN version with coordinates and the DE; this is indeed important to remind the preliminary nature of this study, which represents the first step towards a promising generalization breakthrough but still under-performs against solid but less generic representations like coordinates.

Chapter 6

Conclusions

This research examined the current potential of RL applied to combinatorial problems. Particularly, we tested its generalization capabilities when transferring the latest architectures, valid for a specific subset of combinatorial instances like the TSP, to other examples such as the power allocation problem.

The results proved that finding neural models capable of extracting good features from any graph is still an unresolved task. In fact, all the existing architectures are generally highly node-dependant and unable to fully exploit the edges' attributes: in the last instance, considering the edges' importance on representing the nodes' relationships, this severely limits the model's representation power.

We also showed that the lack of static elements like coordinates prevents the decoder from successfully creating a context: the experiments in fact testify that a model which only uses the encoder's representations performs better than all the available encoder-decoder alternatives. This is in contrast with the usual trends in ML, and must therefore be a results restraining-factor as well.

We then presented DE, a recently proposed technique to transform edge features into novel nodes ones. This is meant to enhance the structural information provided to the model, solving the aforementioned issues: in fact, DE could potentially improve the model's representation power while providing static information to the decoder. The numerical results testify that both the power allocation problem and the TSP benefit from its use; however, despite the improvements, it is still incapable of being as representative as coordinates would be if available.

To conclude, this study can hopefully bring a twofold advantage to future researches: firstly, it should point out that current architecture, even if promising and successful on a subset of combinatorial tasks, still struggles at generalizing beyond them; secondly, that further improvements in the field of graphs require enhancing the models' representation capabilities. For this second point, DE represents a fascinating alternative which anyway still requires a deeper study for completely solving the current lacks.

Bibliography

[1] J. K. Lenstra and A. H. G. Rinnooy Kan. «Some Simple Applications of the Travelling Salesman Problem». In: *Operational Research Quarterly (1970-1977)* 26.4 (1975), pp. 717–733. ISSN: 00303623. URL: http://www.jstor.org/stable/3008306 (cit. on p. 1).

[2] Horacio Yanasse. «A review of three decades of research on some combinatorial optimization problems». In: *Pesquisa Operacional* 33 (Apr. 2013), pp. 11–36. DOI: 10.1590/S0101-74382013000100002 (cit. on p. 1).

[3] J et al. Ellman. «Combinatorial thinking in chemistry and biology». In: *Proceedings of the National Academy of Sciences of the United States of America* 94 (1997), pp. 2779–82. DOI: doi:10.1073/pnas.94.7.2779 (cit. on p. 1).

[4] Rafael Gómez-Bombarelli et al. «Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules». In: *ACS Central Science* 4.2 (Jan. 2018), pp. 268–276. ISSN: 2374-7951. DOI: 10.1021/acscentsci.7b00572. URL: http://dx.doi.org/10.1021/acscentsci.7b00572 (cit. on p. 1).

[5] F. Kuipers, P. Van Mieghem, T. Korkmaz, and M. Krunz. «An overview of constraint-based path selection algorithms for QoS routing». In: *IEEE Communications Magazine* 40.12 (2002), pp. 50–55. DOI: 10.1109/MCOM.2002.1106159 (cit. on p. 1).

[6] Ayoub Bahnasse, Fatima Ezzahraa Louhab, Hafsa Ait Oulahyane, Mohamed Talea, and Assia Bakali. «Smart bandwidth allocation for next generation networks adopting software-defined network approach». In: *Data in Brief* 20 (2018), pp. 840–845. ISSN: 2352-3409. DOI: https://doi.org/10.1016/j.dib.2018.08.091. URL: https://www.sciencedirect.com/science/article/pii/S2352340918309430 (cit. on p. 1).

[7] Ashish Raniwala, Kartik Gopalan, and Tzi-cker Chiueh. «Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks». In: *ACM SIGMOBILE* (2004) (cit. on p. 1).

[8] Taehoon Park and Chae Y Lee. «Application of the graph coloring algorithm to the frequency assignment problem». In: *Journal of the Operations Research society of Japan* (1996) (cit. on pp. 1, 7).

[9] Navid Naderializadeh, Jaroslaw Sydir, Meryem Simsek, and Hosein Nikopour. «Resource Management in Wireless Networks via Multi-Agent Deep Reinforcement Learning». In: *arXiv:2002.06215* (2020) (cit. on p. 1).

[10] Francesc Padró, Javier Ozón, and Departament Aplicada. «Graph Coloring Algorithms for Assignment Problems in Radio Networks». In: (Aug. 1995) (cit. on p. 1).

[11] Holger H. Hoos and Thomas Stützle. «1 - INTRODUCTION». In: *Stochastic Local Search*. Ed. by Holger H. Hoos and Thomas Stützle. The Morgan Kaufmann Series in Artificial Intelligence. San Francisco: Morgan Kaufmann, 2005, pp. 13–59. ISBN: 978-1-55860-872-6. DOI: https://doi.org/10.1016/B978-155860872-6/50018-4. URL: https://www.sciencedirect.com/science/article/pii/B9781558608726500184 (cit. on p. 1).

[12] R. Gary Parker and Ronald L. Rardin. «1 - Introduction to Discrete Optimization». In: *Discrete Optimization*. Ed. by R. Gary Parker and Ronald L. Rardin. Computer Science and Scientific Computing. San Diego: Academic Press, 1988, pp. 1–10. ISBN: 978-0-12-545075-1. DOI: https://doi.org/10.1016/B978-0-12-545075-1.50006-7. URL: https://www.sciencedirect.com/science/article/pii/B9780125450751500067 (cit. on p. 1).

[13] Antoine François, Quentin Cappart, and Louis-Martin Rousseau. *How to Evaluate Machine Learning Approaches for Combinatorial Optimization: Application to the Travelling Salesman Problem*. 2019. arXiv: 1909.13121 [cs.AI] (cit. on p. 2).

[14] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006 (cit. on p. 2).

[15] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016 (cit. on pp. 2, 7).

[16] Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. *An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem*. 2019. arXiv: 1906.01227 [cs.LG] (cit. on pp. 2, 21, 31–36).

[17] Wouter Kool, Herke van Hoof, and Max Welling. «Attention, Learn to Solve Routing Problems!» In: *arXiv:1803.08475* (2018) (cit. on pp. 2, 3, 6, 19, 28, 29, 31, 32, 34–37, 44, 48, 49).

[18] MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. «Reinforcement Learning for Solving the Vehicle Routing Problem». In: *NeurIPS*. 2018 (cit. on pp. 2, 6, 26–29).

[19] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. *Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon*. 2020. arXiv: 1811.06128 [cs.LG] (cit. on p. 2).

[20] Guozhi Lin, Jingguo Ge, Yulei Wu, Hui Li, Tong Li, Wei Mi, and Yuepeng E. «Network Automation for Path Selection: A New Knowledge Transfer Approach». In: *2021 IFIP Networking Conference (IFIP Networking)*. 2021, pp. 1–9. DOI: 10.23919/IFIPNetworking52078.2021.9472841 (cit. on pp. 2, 3).

[21] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. *Reinforcement Learning for Combinatorial Optimization: A Survey*. 2020. arXiv: 2003.03600 [cs.LG] (cit. on p. 2).

[22] Sergio Barrachina-Muñoz, Alessandro Chiumento, and Boris Bellalta. *Stateless Reinforcement Learning for Multi-Agent Systems: the Case of Spectrum Allocation in Dynamic Channel Bonding WLANs*. 2021. arXiv: 2106.05553 [cs.NI] (cit. on p. 2).

[23] Peter W. Battaglia et al. *Relational inductive biases, deep learning, and graph networks*. 2018. arXiv: 1806.01261 [cs.LG] (cit. on p. 2).

[24] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, Thomas Laurent, and Xavier Bresson. *Learning TSP Requires Rethinking Generalization*. 2020. arXiv: 2006.07054 [cs.LG] (cit. on pp. 3, 33, 37, 43, 44, 70).

[25] Ovidiu Iacoboaiea, Jonatan Krolikowski, Zied Ben Houidi, and Dario Rossi. «Real-Time Channel Management in WLANs: Deep Reinforcement Learning versus Heuristics». In: *2021 IFIP Networking Conference (IFIP Networking)*. 2021, pp. 1–9. DOI: 10.23919/IFIPNetworking52078.2021.9472828 (cit. on pp. 3, 6).

[26] O. Jeunen, P. Bosch, M. V. Herwegen, K. V. Doorselaer, N. Godman, and S. Latré. «A Machine Learning Approach for IEEE 802.11 Channel Allocation». In: *2018 14th International Conference on Network and Service Management (CNSM)*. 2018 (cit. on p. 3).

[27] K. Nakashima, S. Kamiya, K. Ohtsu, K. Yamamoto, T. Nishio, and M. Morikura. «Deep Reinforcement Learning-Based Channel Allocation for Wireless LANs With Graph Convolutional Networks». In: *IEEE Access* (2020) (cit. on p. 3).

[28] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. *Benchmarking Graph Neural Networks*. 2020. arXiv: 2003.00982 [cs.LG] (cit. on pp. 3, 19, 21).

[29] Iddo Drori et al. *Learning to Solve Combinatorial Optimization Problems on Real-World Graphs in Linear Time*. 2020. arXiv: 2006.03750 [cs.LG] (cit. on p. 3).

[30] M. Gori, G. Monfardini, and F. Scarselli. «A new model for learning in graph domains». In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942 (cit. on p. 4).

[31] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. *Gated Graph Sequence Neural Networks*. 2017. arXiv: 1511.05493 [cs.LG] (cit. on p. 4).

[32] David L. Applegate, Robert E. Bixby, Vašek Chvatál, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006. ISBN: 9780691129938. URL: http://www.jstor.org/stable/j.ctt7s8xg (cit. on p. 5).

[33] S. Lin and B. Kernighan. «An Effective Heuristic Algorithm for the Traveling-Salesman Problem». In: *Oper. Res.* 21 (1973), pp. 498–516 (cit. on p. 5).

[34] Nicos Christofides. «Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem». In: 1976 (cit. on p. 5).

[35] Shen Lin. «Computer solutions of the traveling salesman problem». In: *The Bell System Technical Journal* 44.10 (1965), pp. 2245–2269. DOI: 10.1002/j.1538-7305.1965.tb04146.x (cit. on p. 5).

[36] M. Seyedebrahimi, F. Bouhafs, A. Raschellà, M. Mackay, and Q. Shi. «SDN-based channel assignment algorithm for interference management in dense Wi-Fi networks». In: *EuCNC*. 2016 (cit. on p. 7).

[37] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018 (cit. on pp. 7, 8).

[38] Alexander Amini. *MIT Introduction to Deep Learning*. Youtube. 2021. URL: https://www.youtube.com/watch?v=93M1l_nrhpQ (cit. on p. 8).

[39] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG] (cit. on p. 9).

[40] István Szita. «Reinforcement Learning in Games». In: *Reinforcement Learning: State-of-the-Art*. Ed. by Marco Wiering and Martijn van Otterlo. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 539–577. ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_17. URL: https://doi.org/10.1007/978-3-642-27645-3_17 (cit. on p. 9).

[41] Elliot Waite. *Policy Gradient Theorem Explained - Reinforcement Learning*. Youtube. 2020. URL: https://www.youtube.com/watch?v=cQfOQcpYRzE (cit. on p. 10).

[42] Pieter Abbeel. *Deep RL Bootcamp Lecture 4A: Policy Gradients*. Youtube. 2017. URL: https://www.youtube.com/watch?v=S_gwYj1Q-44 (cit. on p. 11).

[43] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL] (cit. on pp. 12, 14).

[44] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL] (cit. on pp. 12–14).

[45] Nal Kalchbrenner and P. Blunsom. «Recurrent continuous translation models». In: *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* 3 (Jan. 2013), pp. 1700–1709 (cit. on p. 12).

[46] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL] (cit. on p. 12).

[47] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL] (cit. on p. 12).

[48] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL] (cit. on p. 12).

[49] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL] (cit. on p. 12).

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. «Attention is All you Need». In: *NeurIPS*. 2017 (cit. on pp. 14–16, 28).

[51] Jay Alammar. *The Illustrated Transformer*. Git Hub. 2020. URL: https://jalammar.github.io/illustrated-transformer/ (cit. on pp. 16, 17).

[52] M. Gori, G. Monfardini, and F. Scarselli. «A new model for learning in graph domains». In: *IEEE International Joint Conference on Neural Networks*. 2005 (cit. on p. 17).

[53] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. «The Graph Neural Network Model». In: *IEEE Transactions on Neural Networks* (2009) (cit. on pp. 17, 18).

[54] Will Hamilton, Zhitao Ying, and Jure Leskovec. «Inductive Representation Learning on Large Graphs». In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 1024–1034. URL: http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf (cit. on pp. 18, 19).

[55] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG] (cit. on p. 18).

[56] Zhengdao Chen, Xiang Li, and Joan Bruna. *Supervised Community Detection with Line Graph Neural Networks*. 2020. arXiv: 1705.08415 [stat.ML] (cit. on p. 18).

[57] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. «A Survey on Network Embedding». In: *IEEE Transactions on Knowledge and Data Engineering* 31.5 (2019), pp. 833–852. DOI: 10.1109/TKDE.2018.2849727 (cit. on p. 18).

[58] William L. Hamilton, Rex Ying, and Jure Leskovec. *Representation Learning on Graphs: Methods and Applications*. 2018. arXiv: 1709.05584 [cs.SI] (cit. on p. 18).

[59] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. «Graph neural networks: A review of methods and applications». In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.01.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651021000012 (cit. on p. 18).

[60] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. «Gated Graph Sequence Neural Networks». In: *arXiv:1511.05493* (2015) (cit. on p. 18).

[61] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. *Geometric deep learning on graphs and manifolds using mixture model CNNs*. 2016. arXiv: 1611.08402 [cs.CV] (cit. on p. 19).

[62] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. «Graph Attention Networks». In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=rJXMpikCZ (cit. on pp. 19, 28).

[63] X. Bresson and T. Laurent. «An Experimental Study of Neural Networks for Variable Graphs». In: *ICLR*. 2018 (cit. on p. 19).

[64] Diego Marcheggiani and Ivan Titov. *Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling*. 2017. arXiv: 1703.04826 [cs.CL] (cit. on p. 20).

[65] Xavier Bresson and Thomas Laurent. *Residual Gated Graph ConvNets*. 2018. arXiv: 1711.07553 [cs.LG] (cit. on p. 20).

[66] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. *Distance Encoding: Design Provably More Powerful Neural Networks for Graph Representation Learning*. 2020. arXiv: 2009.00142 [cs.LG] (cit. on pp. 22, 46).

[67] Haoteng Yin, Yanbang Wang, and Pan Li. *Revisiting graph neural networks and distance encoding from a practical view*. 2020. arXiv: 2011.12228 [cs.SI] (cit. on pp. 22, 46).

[68] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999. URL: http://ilpubs.stanford.edu:8090/422/ (cit. on p. 23).

[69] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. «Pointer Networks». In: *NeurIPS*. 2015 (cit. on pp. 25, 26).

[70] Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. «Neural Combinatorial Optimization with Reinforcement Learning». In: *arXiv:1611.09940* (2016) (cit. on p. 26).

[71] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. «Order matters: Sequence to sequence for sets». In: *ICLR*. 2016. URL: http://arxiv.org/abs/1511.06391 (cit. on p. 26).

[72] Ronald J. Williams. «Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning». In: *Mach. Learn.* 8.3–4 (May 1992), pp. 229–256. ISSN: 0885-6125. DOI: 10.1007/BF00992696. URL: https://doi.org/10.1007/BF00992696 (cit. on p. 31).

[73] Microsoft Research. *An Introduction to Graph Neural Networks: Models and Applications*. Youtube. 2020. URL: https://www.youtube.com/watch?v=zCEYiCxrL_0 (cit. on p. 35).