On the Difficulty of Generalizing Deep Reinforcement Learning Framework for Combinatorial Optimization

by

Mostafa Pashazadeh
B.Sc., Iran University of Science and Technology, 2005
M.Sc., Isfahan University of Technlogy, 2014

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

On the Difficulty of Generalizing Deep Reinforcement Learning Framework for Combinatorial Optimization

by

Mostafa Pashazadeh
B.Sc., Iran University of Science and Technology, 2005
M.Sc., Isfahan University of Technlogy, 2014

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Nishant Mehta, Department Member
(Department of Computer Science)

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Nishant Mehta, Department Member
(Department of Computer Science)

ABSTRACT

Combinatorial optimization problems on the graph with real-life applications are canonical challenges in Computer Science. The difficulty of finding quality labels for problem instances holds back leveraging supervised learning across combinatorial problems. Reinforcement learning (RL) algorithms have recently been adopted to solve this challenge automatically. The underlying principle of this approach is to deploy a graph neural network for encoding both the local information of the nodes and the graph-structured data in order to capture the current state of the environment. Then, a reinforcement learning algorithm trains the actor to learn the problem-specific heuristics on its own and make an informed decision at each state for finally reaching a good solution. Recent studies on this subject mainly focus on a family of combinatorial problems on the graph, such as the travel salesman problem, where the proposed model aims to find an ordering of vertices that optimizes some objective function. We use the security-aware phone clone allocation in the cloud as a classical quadratic assignment problem to study whether or not deep RL-based model is generally applicable to solve other classes of such hard problems. Our work contributes in two directions: First, we provide an analytical method that reduces the phone clone allocation problem to the traditional QP programming and evidence its superiority over heuristic algorithms with quality approximation solutions. Second, we build a powerful model that not only captures the node embedding in the context of graph-structured data but also provides valuable information related to the decision making. We then adopt a fitted RL algorithm to train the actor to make informed decisions. Extensive experimental evaluation shows that existing RL-based models

may not generalize to discrete quadratic assignment problems, where incrementally constructed solution is not an inherent requirement. Furthermore, we highlight the main features of problems that contribute to the success of applying RL algorithms.

Contents

Supervisory Committee ii

Abstract iii

Table of Contents v

List of Figures vii

Acknowledgements viii

Dedication ix

1 Introduction 1

1.1 Contributions . . . 3

1.2 Thesis Organization . . . 5

2 Background 6

2.1 Phone Clone Allocation . . . 6

2.2 Graph Embedding . . . 8

2.2.1 Pointer Network . . . 8

2.2.2 Graph Attention Network . . . 10

2.2.3 Structure2vector . . . 13

2.3 Approximate Solution . . . 14

2.3.1 Solution approaches . . . 17

3 QP-based Solution 19

3.1 Quadratic Programming . . . 19

3.2 Heuristic Algorithms . . . 23

3.3 Experiments . . . 24

3.3.1 Experimental setup . . . 24

3.3.2 Results 24

4 RL-based Solutions **26**
4.1 Q-learning 26
4.2 Policy Gradient 32
4.3 Experimental results 36
4.3.1 K-step Q-learning 36
4.3.2 Policy gradient 44

5 Conclusion and Future work **47**
5.1 Future Work 48

Bibliography **50**

List of Figures

Figure 2.1 A pointer network architecture [4]. 9
Figure 2.2 Attention based encoder [11]. For clarity, only messages received by node 1 are illustrated. 10
Figure 2.3 Message-passing graph attention network. 13

Figure 3.1 Relative hosts' capacities are set to $[0.2, 0.2, 0.2, 0.2, 0.2]$. 25
Figure 3.2 Relative hosts' capacities are set to $[0.1, 0.1, 0.2, 0.5, 0.4]$. 25

Figure 4.1 Q-learning model architecture. 27
Figure 4.2 Policy gradient model architecture. 33
Figure 4.3 Training progress over time, total number of phones = 40. . . . 37
Figure 4.4 Number of phones allocated to different hosts. 38
Figure 4.5 Total potential risk; number of hosts = 5. 38
Figure 4.6 Total potential risk; number of hosts = 5, and relative hosts' capacities are set to [0.1, 0.1, 0.2, 0.3, 0.3]. 39
Figure 4.7 Training progress over time, total number of phones = 100. . . 41
Figure 4.8 Number of phones allocated to different hosts. Number of phones = 100 41
Figure 4.9 Total potential risk; number of hosts = 5, and relative hosts' capacities are set to [0.1, 0.1, 0.2, 0.3, 0.3]. 42
Figure 4.10Training progress over time. 45

ACKNOWLEDGEMENTS

I would like to thank:

My Brother, for supporting me in the low moments.

Professor Kui Wu, for mentoring, support, encouragement, and patience.

Professors Nishant Mehta and Yang Shi, for their time and care in serving in the thesis examination committee.

DEDICATION

Just hoping this is useful!

Chapter 1

Introduction

Combinatorial optimization problems (COPs) on graphs appear in many real-world applications such as manufacturing layout design and DNA sequencing. Generally speaking, each such a problem has unique subtleties and constraints that prevent people from using a renowned optimization solver for a family of hard problems such as the Travel Salesman Problem (TSP) to address all COPs. This issue demands devising methods and examining heuristics specific to each individual COP. This process involves a lot of efforts to discover some structure in the combinatorial search space of the specific problem.

Recently, reinforcement learning (RL) algorithms have been successfully applied in solving hard problems. RL is an area of machine learning that trains a software agent to make the right decision in order to maximize the cumulative reward from the environment. RL can learn heuristics and structure of the problem on its own and often comes up with close to optimal solution. RL-based models consist of two main components: an encoder and a decoder. Once the encoder encodes the information from the environment, the decoder would compute the solution in real-time. Although solutions cannot be proven to be optimal, they get better when the problem's combinatorial space is further explored, and more inputs from problem instances are used to train the agent. To this end, RL algorithms manage to address diverse challenges, including quality solution and response time, and they are partially successful in dealing with scalability. [10] used RL to solve several hard COPs, including Minimum Vertex Cover (MVC), Maximum Cut (MAXCUT), and TSP. They trained a type of graph neural network (GNN) to represent the graph structure. We will further discuss GNNs on this matter in the next chapter. The GNN is then followed by Q-learning [22] to learn a greedy policy that incrementally builds up the solution

set, one element at a time. Experimental results in [10] show that this approach can achieve a promising approximation ratio (the ratio between RL's tour length and the optimal tour length in TSP). [17] solved the constrained TSP. They used an elegant GNN to embed both the local information and the underlying problem constraints to make informed decisions. Then, they employed a policy gradient algorithm [22] to learn the optimal policy. The results show that RL-based method outperforms approximation algorithms for small-scale TSP and achieves competitive solutions for large-scale constrained TSP. Furthermore, the trained models in [10] and [17] can handle problems of roughly an order of magnitude larger than the size of instances they were trained on. [16], [4], [7] and [18] also presented interesting ideas in this emerging field to solve hard problems such as Vehicle Routing Problem (VRP). They designed problem-specific frameworks to learn the policy that optimizes the objective function of the problem.

It is worth noting that [13] has introduced supervised learning to address hard problems. However, the difficulty of finding superior labels for instances of combinatorial optimization is a deterrent against leveraging supervised machine learning techniques across hard problems. Due to this reason, most successful machine learning algorithms for combinatorial optimization fall into the family of unsupervised learning.

We are interested in whether or not RL is generally applicable to solve other classes of hard problems faced in computer networks. Using the security-aware phone clone provisioning problem [23] as the motivating example, we investigate whether or not RL brings benefits and outperforms traditional optimization solutions. The idea of phone clones [8] is to build software clones of smartphones on the cloud that allows end users to upload resource-consuming tasks and backup data to the cloud. To be secure, phone clones that meet certain conditions should not be allocated to the same physical machine. Hence, security-aware allocation of clones to physical hosts in the cloud involves solving constrained discrete optimization problems with non-linear objective functions. Detailed problem formulation will be disclosed in Chapter 2.

Although phone clone allocation is identified as a hard problem, its nuance and circumstances differ from the class of problems mentioned above. As opposed to previous works that incrementally construct the final solution starting from a partial solution (tour), the phone clone provisioning improves on current provisioning of the clones by reallocating the phone clones in the cloud. The incremental approach allows previous works to adopt a mask function that excludes the partial solution from

search space and narrows the focus of the investigation. However, as we introduce RL-based solution to phone clone allocation problem in Chapter 4, it is irrelevant to exert any kind of masking procedure on the resources or restrict reallocating of some phone clones. Otherwise, we irrationally interfere in the learning process, which likely limits the algorithm's capability and turns it further away from a quality solution. Phone clone provisioning requires new architecture capable of capturing relevant information from the environment and learning a more sophisticated decision-making policy regarding the non-trivial objective function of the problem. The environment states embody the solution-specific features (current provisioning of the clones) and the problem-specific features (underlying communication graph between the phones). In order to measure the quality of the results obtained from the RL algorithm, we figure out an effective approximate solution by exploiting the problem's structure and circumstances. To achieve this goal, we first reduce the phone clone allocation problem to traditional QP programming and solve it by a general-purpose solver. This approach yields high-quality results, and its superiority is then evidenced by comparing it with heuristic algorithms. However, QP solver is a costly approach to our problem setting, only suitable for solving our problem in small scale.

1.1 Contributions

The main contributions of this thesis are threefold:

1. As the same high-level design of previous works cannot be directly applied to phone clone allocation on the cloud, we need a powerful model to tackle this issue. To this end, we design a well-grounded RL architecture with a GNN capable of concisely capturing and embedding both current provisioning of the clones and the underlying communication graph. The GNN is followed by an aggregation layer that looks over these embeddings and provides meaningful features of the current state to the decoder. Then, a Neural Network (decoder or actor) takes the state's feature map and learns a Q-function that approximates the action values. Action value here corresponds to the cost of assigning a phone clone to a host. We then train the RL model with k-step Q-learning algorithm that helps deal with the issue of delayed rewards, where the final reward is received by adding the attainable future reward to the immediate reward during a training episode.

2. We point out that RL-based approach, at least in its current incarnation, cannot generalize successfully to all types of COPs. This insight is a valuable note for researchers trying to promote RL-based solutions in COPs. While it is difficult to offer a theoretical proof on this claim, we offer empirical explanation by carefully analyzing the existing breakthrough in graph encoding techniques and successful RL algorithms on this matter. We identify the underlying features of problems where RL may be successful and explain why the lack of these features can compromise the success of RL as follows.

   (a) **Helper function:** Previous works such as TSP incrementally construct the solutions by adding one element at a time to the current partial solution (partial tour). With the aid of a problem-specific masking function, they reduce the combinatorial search space to a permutation of nodes currently not in this subset. Indeed, they approximate the action value of adding any node that has not been touched yet to the partial solution. As a result, the search space gets smaller as we move forward through the training episode. By reducing the search space, the helper function conducts the model to find a better solution.

   (b) **Diversified inputs:** In the TSP problem, the city coordinates as the input vectors carry diverse and abundant information, making the context vectors that get into the decoder profoundly distinguishable from one training iteration to another. The context vectors efficiently reflect the varying states of the environment. Rich context vectors conduct the decoder to trace the varying environment states across training iterations, and empower it to define more reliable and authentic state-action values. This issue highlights the role of the graph encoder to learn the problems' heuristics and generate distinctively different embeddings. On the other hand, the initial phone allocation vectors as the inputs in our problem setting are one-hot vectors from a very limited state pool. A one-hot vector is filled with zeros except for a one at the index of the assigned host (A one-hot vector in this example is a vector that every element is zero except for the one at the index of the assigned host). The setback of a limited state pool prompts many phone clones to share the same input vector (or the same host). This setback causes the context vectors to not remarkably contrast with each other as the environment state changes from one

training iteration to another; the encoder does not truthfully turn over the varying environment states to the decoder. As a result, the decoder is not adept enough to figure out meaningful and authentic action values throughout the training process. This issue will be clear as we explain the RL model architectures in the following chapters.

(c) **Trivial objective function:** In previous problem settings, the agent learns from a relatively light objective function, which is a simple summation over a quantitative feature of the nodes. However, in our problem setting, the agent should deal with an arduous nonlinear objective function.

3. We introduce a system model that represents the provisioning to phone clones in the cloud as an optimization problem. We then propose a quality approximate solution to the optimization problem that minimizes the risk of allocating the phone clones to the physical hosts. This method proves to be superior to the previous heuristic approaches to this problem.

4. For future research, we discuss possible directions and point out open challenges to tackle certain difficulties raised in applying RL to hard combinatorial problems.

1.2 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we review recent RL advances in solving combinatorial optimization problems, and highlight the features of the problems where RL is successful. We also review some traditional approximate methods for solving hard combinatorial problems. In Chapter 3, we introduce an analytical QP-based solution to the phone clone allocation problem and evidence its effectiveness. In Chapter 4, we present an RL-based solution, and explain the main barriers in the RL-based solution. We provide an intuitive explanation why the RL-based solution cannot keep up with the QP-based solution. Finally, Chapter 5 concludes the thesis and discusses future research.

Chapter 2

Background

2.1 Phone Clone Allocation

We first introduce a system model that provides a mathematical perspective of the phone clones' security-aware allocation in the cloud. This solid view helps to recognize its nuance and inherent complexity in contrast with hard problems previously attacked by related works.

Smartphones often install quite a few useful applications that help users make the best out of daily life. To facilitate operating applications that are highly resource-consuming, the smartphones can leave them to the on-demand resource available on the cloud, upload the data and then get back the results which can be done by creating a software clone of the smart phone on the cloud [2, 14]. In phone clone allocation, we should be mindful of both security issues and the hosts' capacities. In practice, a phone clone can hack into others on the same host via covert channel [21, 25]. It is best to co-locate a phone to those closely connected with the phone rather than the strangers. However, due to the large number of end-users and the limited number of physical hosts, it is impossible to perfectly group the connected phone clones and isolate them from strangers. This cramp brings up the problem of security-aware provisioning of the phone clones [23].

Instead of intimate-stranger perspective on relationships among phone clones, we focus on a more general weighted version of the problem. We represent the communication history among mobile users with a weighted communication graph. A small weight implies poor communication between the endpoints and a high risk of attacks. We assume the system has m phone clones and n hosts in the cloud. We represent

the communication graph with an adjacency matrix $W = [w_{ij}]_{m\times m}$, where w_{ij} is a real value that models the tie between phone clones i and j. We denote the phone clone allocation matrix with $X = [x_{ij}]_{m\times n}$, where $x_{ij} = 1$ indicates that phone clone i is allocated to host j and $x_{ij} = 0$ otherwise. Given the adjacency matrix W and the allocation matrix X, the potential risk would be formulated as follows

$$\Upsilon = \frac{1}{2} tr(X^T \bar{W} X) \tag{2.1}$$

where $\bar{W} = [\bar{w}_{ij}]_{m\times m} = [1 - w_{ij}]_{m\times m}$ denotes the complementary adjacency matrix and $\bar{w}_{ij}$ denotes the potential risk between phone clones i and j. $tr(\cdot)$ denotes the trace of a matrix. For security-aware provisioning that keeps to the capacity constraints of hosts, we need to solve the following discrete optimization problem to minimize the risk of a phone clone allocation scheme.

$$\begin{aligned} & \min_{X} \ \frac{1}{2} tr(X^T \bar{W} X) \\ s.t. & \\ & x_{ij} \in \{0, 1\} \\ & \sum_{j=1}^{n} x_{ij} = 1, \qquad \text{for } i = 1, 2, \ldots m \\ & \sum_{i=1}^{m} x_{ij} \leq c_j, \qquad \text{for } j = 1, 2, \ldots n \end{aligned} \tag{2.2}$$

where c_j is the capacity of j-th host w.r.t. the maximum number of phone clones that it can host.

RL-based methods attack COPs through a twofold architecture composed of encoder and decoder. Encoder trains a GNN to learn the graph representation and the decoder trains a Neural Network (NN) with RL algorithm to make the right decision. In this problem setting, we train an RL model that starts with an initial solution and improves it iteratively. The counterpart in our model is to first learn node embeddings that encode the graph topology and the current provisioning of phone clones to figure out the cost of reallocating a phone to each host. Then, we look over the node embeddings through an aggregation layer and pass the outcome to the actor (decoder). The outcome of the aggregator, known as context vector, conducts the decoder to make an informed decision. The actor makes use of NN to learn a Q-function that

approximates the true value of allocating a phone to each host and accordingly define the optimal policy. This architecture is discussed in more detail in Chapter 4.

In the rest of this chapter, we keep in mind the requirements and circumstances of our problem as we go through recent architectures on this matter. This background review instructs us to work out an elegant architecture that is capable of capturing relevant information from environment states with respect to the requirements of our problem setting. Astute and thorough perception of environment is essential to develop a concrete model before we can take on the RL approach for constrained hard combinatorial optimization. We also highlight main features of the problems where RL is successful.

2.2 Graph Embedding

2.2.1 Pointer Network

Bello et al. [4] used a pointer network architecture to solve (2D Euclidean) TSP and KnapSack problems. Given a set of m cities $s = \{x_i\}_{i=1}^{m}$ where each $x_i \in R^2$ is a 2D coordinates of the i_{th} city, they adopted a graph level representation with recurrent neural network (RNN) [9] to read the inputs and encode them into a context vector. The decoder, made of RNN and an attention function, takes the context vector and calculates a distribution over the next city to be visited. Decoding proceeds sequentially, that is, once the next city is selected, it is fed to the next decoder step as input.

This method uses numerous problem instances as training samples and defines the tour length as the loss function. It then trains the model's parameters via policy gradient algorithm to minimize the loss at each trial. It achieves satisfying results on problem sizes of up to 100 nodes. However, there are four inherent differences between TSP and our problem setting that make it nonsensical to apply this model to the phone clone allocation problem.

- The underlying network graph in this problem is assumed to be complete, and graph topology is not incorporated in the encoder whereas in our case we model the communication graph among phone clones with an adjacency matrix.

- Inputs in our problem setting would be current allocation vectors, one-hot vector indicating the host to which a phone clone is assigned. The decoder looks over

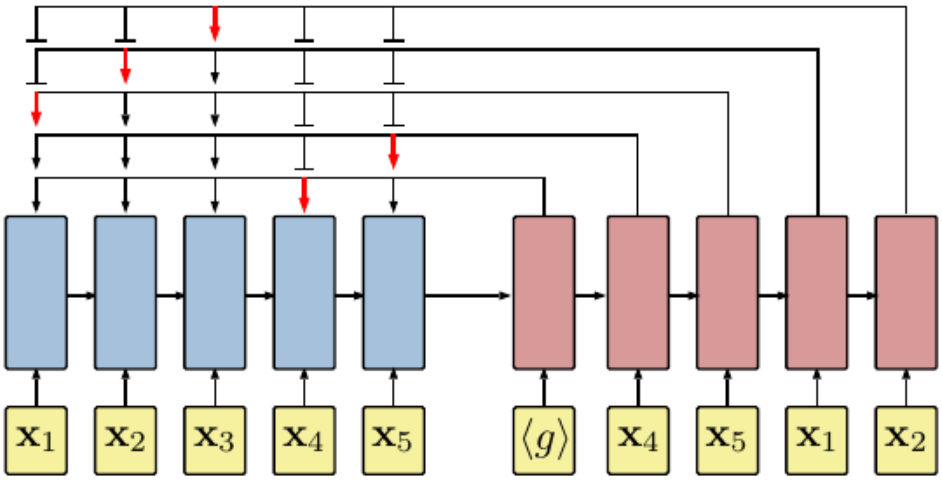


Figure 2.1: A pointer network architecture [4].

the node embeddings regardless of the order of the inputs to approximate the action values. As a result, the order in which the input vectors are fed into the encoder does not matter. In other words, a good encoder should be invariant to the permutation of input vectors so that changing the order of any two input vectors does not affect the model. This issue would be clear in the next chapter as we describe the architecture.

- The input vectors in TSP are diverse coordinates yielding profoundly distinguishable context vector from one training iteration to another. Rich context vectors reflect on varying states of the environment and instruct the decoder to define the state-action values more accurately and reliably. Whereas in our problem, there are few one-hot input vectors, corresponding to different hosts, shared between many phones; node embeddings may sound different, but they come from a limited pool. As a result, different allocation matrices lead to hardly distinguishable sets of embeddings (states) so as the decoder will not perceive correctly the varying state of the environment.

- The solution (tour in TSP) is built up incrementally which allows the decoder to use a helper function to mask the cities that were already visited. In other words, the search space gets smaller as the decoder moves forward through training iterations. Nonetheless, the essence of the phone clone allocation problem holds back unsupervised RL-based solution to rely on helper function for masking.

As we explain adopting the RL algorithm to solve this problem in Chapter 4, the counterpart of figuring out the next node (city) to be added to the partial solution (partial tour in TSP) is to decide the host to which we will assign the next phone. In this case, it is not a straightforward task to manually put kind of masking on the hosts' pool or limit the reallocation of some phones. Otherwise, irrational interferences that merely reduce the feasible domain of actions, push the decoder further toward a poor policy. In other words, the optimum policy should be learned by the RL framework itself. The decoder can learn about the potential risk of hosts through the reward function and keeps updating its approximate action values at each training iteration. The encoder is also tasked to encode the interactions between phones based on the underlying communication graph, and the current provisioning of phone clones at each iteration. The reward function and the encoder are the eyes of the decoder, which help the decoder to learn and advance its approximate action values in various situations.

2.2.2 Graph Attention Network

[11] and [24] push the idea forward and focus on developing a message-passing graph encoder that incorporates the graph topology and input features. The encoder can be leveraged across hard problems such as TSP, the vehicle routing problem (VRP), and the orienteering problem (OP) seamlessly, but the decoder needs to be customized to new circumstances. This approach contrasts with the previous architecture that employed a graph-agnostic sequence to sequence mapping. The encoder consists of several attention layers. Fig. 2.2 shows a general overview of message passing in the attention layer.

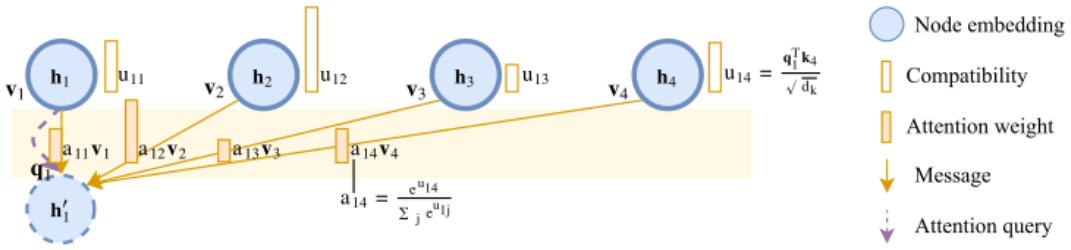


Figure 2.2: Attention based encoder [11]. For clarity, only messages received by node 1 are illustrated.

A problem instance is defined as a graph with m nodes, and the input features of i-th node is represented by vector x_i. The encoder first computes initial node embeddings from input features via a learnable linear projection. Then, each attention layer takes the node embeddings from the previous layer and computes new embeddings for the next layer. Each attention layer comprises two sublayers: a multi-head attention (MHA) layer and a node-wise fully connected (FC) layer. An attention head is roughly a weighted message-passing mechanism between the nodes of graph. Each sublayer also adds a residual connection to incorporate input features into outcoming node embedding. Each attention layer carries out computation as follows: for each node i, $h_i \in R^{d_h}$ denotes the node embedding taken from the previous layer. Each node first computes the key vector $k_i \in R^{d_k}$, value vector $v_i \in R^{d_v}$ and query vector $q_i \in R^{d_k}$ by projecting the node embedding as follows

$$q_i = \theta^Q h_i, \quad k_i = \theta^K h_i, \quad v_i = \theta^V h_i \tag{2.3}$$

where $\theta^Q \in R^{d_k \times d_h}$, $\theta^K \in R^{d_k \times d_h}$ and $\theta^V \in R^{d_v \times d_h}$ are learnable parameters. Then, the compatibility between the query q_i of node i and key k_j of node j is computed as

$$u_{ij} = \begin{cases} \frac{q_i^T k_j}{\sqrt{d_k}} & \text{if j non-adjacent to i} \\ -\infty & \text{otherwise} \end{cases} \tag{2.4}$$

Given the compatibilities u_{ij}, the attention weight a_{ij} between nodes i and j is computed using softmax function as $a_{ij} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{ij'}}}$. Then, the attention vector in node i (h'_i) is computed as the convex combination of messages received by node i, $h'_i = \sum_j a_{ij} v_j$. Node i also makes use of multiple attention heads to receive different types of messages from other nodes. It computes multiple attention vectors $h'_{i\gamma}$, $\gamma \in \{1 \dots \Gamma\}$ with different parameters where Γ is the number of attention heads and $h'_{i\gamma}$ is the output of the γ_{th} attention head. Then, these attention vectors are projected back to a single d_h-dimensional vector using learnable parameters $\theta^O_\gamma \in R^{d_h \times d_v}$. Finally, multi-head attention outcome for node i denoted by $\hat{h}_i$ is a linear function of attention vectors $\hat{h}_i = \sum_{\gamma=1}^{\Gamma} \theta^O_\gamma h'_{i\gamma}$.

The multi-head attention sublayer is followed by a node-wise fully connected sublayer that computes linear projections and applies the ReLU nonlinearity as follows

$$h_i^{new} = FC(\hat{h}_i) = \theta_1 \, relu(\theta_0 \hat{h}_i + \mu_0) + \mu_1 \tag{2.5}$$

where $\theta_1 \in R^{d_{fc} \times d_{fc}}$, $\theta_0 \in R^{d_{fc} \times d_h}$ and μ_1, $\mu_0 \in R^{d_{fc}}$ are learnable parameters.

It is noteworthy that a node embedding obtained from graph attention network with l attention layers embodies information about its l-hop neighbourhood as determined by the graph topology.

The decoder determines the next node to be visited, sequentially, one node per iteration. At each timestep $t \in \{1 \dots m\}$, the decoder takes the node embeddings h_is and graph embedding $\bar{h} = \frac{1}{m} \sum_{i=1}^{m} h_i$ from the encoder and the partial solution (tour) generated until then. The decoder first concatenates graph embedding and the embedding of the last visited node to create a context node. It then computes the attention weights between the context node and node embeddings that have not been visited yet using a single head attention mechanism, as explained before. In contrast with the encoder's attention mechanism, the decoder issues the query vector only from the context node, and key vectors and value vectors come from the unseen node embeddings. These attention weights are then considered to be the distribution over the next node that the decoder adds to the tour. This model uses a simple loss function that is the negative tour length and policy gradient algorithm to train the parameters. Experimental results show that this model brings benefits over the Pointer Network.

This model apparently provides a comprehensive message passing methodology and seems to be a good fit for a class of hard problems such as TSP, VRP and OR. However, there are some barriers that hold back applying this method in our problem.

- Encoder incurs overly extra computation that leaves an adverse impact on the performance when it comes to problems of larger size or other variants of hard optimization problems. Experimental results show that this model cannot scale up to solve larger problems, which endorses our claim.

- It is not a promising option for weighted graphs as it relies on queries and keys to realize weighted message passing rather than graph's weight matrix itself. Indeed, we applied graph attention network as an alternative solution to a variant of our problem with unweighted graph, but the outcome was not satisfying.

- As mentioned in previous section the essence of the phone clone allocation problem bars the RL-based solution to count on a helper function for masking. In this case, the optimum policy should be learned by RL framework itself through exploiting the problem's structure.

2.2.3 Structure2vector

[17], [10] and [19] introduced variants of message passing methodology which effectively reflect the graph-structured data from environment. [17] focused on constrained TSP and [10] addressed TSP, minimum vertex cover (MVC) and MAXCUT problems. They showed the ability to generalize to problems roughly by order of magnitude larger than those they were trained on. Since this methodology complies with the characteristics of our problem setting, we develop from that to solve the more complex problem of phone clone allocations.

Fig. 2.3 shows a general view of the message-passing graph attention network. Each node $i \in \{1, \ldots, m\}$ in the graph represents an embedding vector h_i^l where l

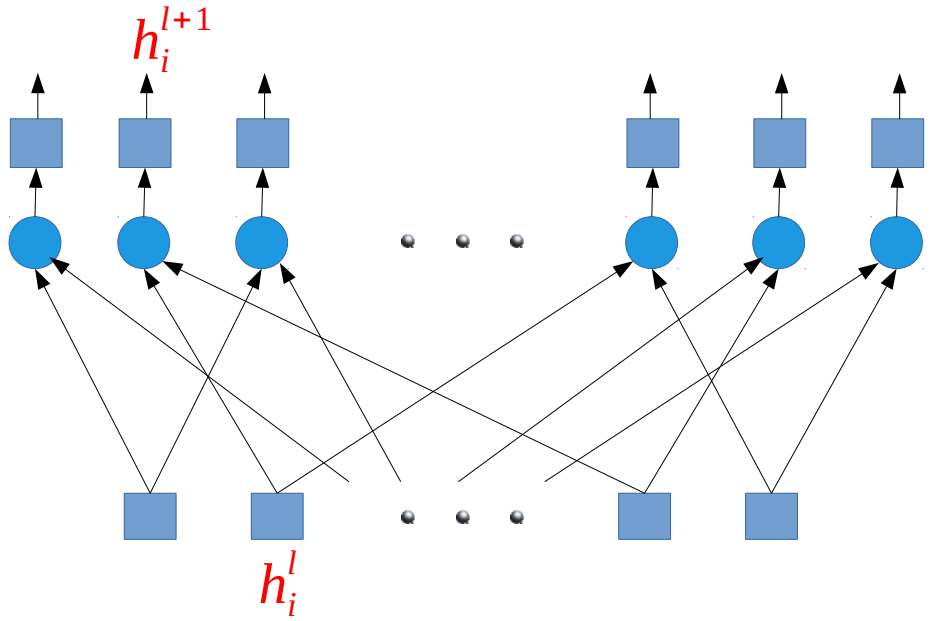


Figure 2.3: Message-passing graph attention network.

indicates the current layer. Preliminary embedding vector h_i^0 is initialized by mapping the corresponding input vector x_i to a larger dimension vector. During message passing phase, the embeddings are updated synchronously at each layer l according to

$$h_i^{(l+1)} \leftarrow F(x_i, \{h_j^l\}_{j \in N(i)}, \{w_{ij}\}_{j \in N(i)}) \tag{2.6}$$

where $N(i)$ is the set of neighbours of node i, w_{ij} is the weight of the edge between nodes i and j, and F is a generic nonlinear function such as a neural network. $h_i^{(l)}$ and $h_i^{(l+1)}$ are the inputs and outputs of layer l, respectively. Eq. (2.6) also implies

that message-passing graph encoder generally provides a residual path to incorporate the input feature to the final node embedding.

Once node embeddings are computed at the last layer, the decoder takes them as inputs and sequentially adds the next node to the partial solution (tour in TSP) in the order determined by its learned policy. The decoder in [10] uses a fitted Q-learning technique to learn a parameterized policy that aims to optimize the objective function of the problem instance (minimize the tour length in TSP). The main advantage of Q-learning algorithm is that it is mindful of delayed reward, which makes it a fitted approach for training our problem as well. In each training iteration of this algorithm, the embeddings from the encoder are updated according to the partial solution and problem-specific features (underlying graph) to capture the environment's changes after adding the most recent node to the partial solution. Distinctively different sets of node embedding that truly reflect the changing environment from one training iteration to another empowers the decoder to find meaningful action values. A big drawback of this method, similar to previous works, is that it relies on a masking function to reduce the search space. We want to avoid this problem-specific helper function because it is irrelevant to the circumstances of our problem.

2.3 Approximate Solution

There is a steady stream of literature on polynomial-time algorithms for certain classes of discrete optimization such as shortest paths, flows and circulations, and travel salesman problems. A well-known research topic on this matter is approximation algorithm, typically represented by the theory of linear programming, that finds close to optimal solutions. Nonetheless, phone clone allocation problem is considered to be a class of quadratic assignment problem (QAP) called constrained scheduling problems with interaction cost that additionally demand the feasible solution to stay within the constraint of hosts' capacities. Indeed, most hard combinatorial optimization problems on the graph are described as Boolean quadratic problems with assignment constraint. QAP is the most difficult combinatorial optimization problem that prohibits the common-sense rules intended to increase the probability of solving renowned hard problems and demands developing novel heuristics.

General quadratic assignment problem is essentially associating a set of m plants with a set of m locations so as to minimize a target quadratic cost function with respect to interaction between economic plants. The quadratic term in the cost

function comes from circumstances that links the cost of assigning a plant to a certain location given the allocation of the rest of the plants to the set of locations. If we denote plant-location pairing by an $m \times m$ matrix $X = (x_{ij})$ and define t_{ik} and d_{jl} as follows

$$x_{ij} = \begin{cases} 1 & \text{if plant i is assigned to location j} \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

$$\begin{cases} t_{ik} = & \text{total amount to be transported from plant i to plant k} \\ d_{jl} = & \text{unit transportation cost from location j to location l,} \end{cases} \tag{2.8}$$

then we can state the general quadratic assignment problem [12] as follows

$$min \left\{ \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{m} \sum_{l=1}^{m} t_{ik} d_{jl} x_{ij} x_{kl} : X \in \chi_m \right\} \tag{2.9}$$

where $\chi_m = \left\{ X \in R^{m \times m}, X = (x_{ij}) : \sum_i x_{ij} = 1, \sum_j x_{ij} = 1, x_{ij} \in \{0, 1\} \right\}$ is the feasible space of permutation matrices. The constraints account for each facility to be indivisible and has to be matched with exactly one location. A generalization of this problem can be stated as follows:

$$min \left\{ \sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{k=1}^{m} \sum_{l=1}^{m} w_{ijkl} x_{ij} x_{kl} : X \in \chi_m \right\} \tag{2.10}$$

where χ_m is as defined before and w_{ijkl} are arbitrary cost coefficients. To write this in matrix form, $X \in \chi_m$ is flattened row-wise to form the vector $x \in R^{m^2}$, i.e., the elements of x are ordered as $(x_{11}, \ldots x_{1m}, x_{21}, \ldots x_{2m}, x_{m1}, \ldots x_{mm})$. Denote $\psi_m = \left\{ x \in R^{m^2} : \sum_i x_{ij} = 1, \sum_j x_{ij} = 1, x_{ij} \in \{0, 1\} \right\}$. Assume $Q \in R^{m^2 \times m^2}$ to be an upper triangular matrix with zero-diagonal, i.e., $Q_{ik} \in R^{m \times m}$ for $1 \leq i \leq k \leq m$ and zero otherwise, and $Q_{ik} = (q_{jl})$ where

$$q_{jl} = \begin{cases} w_{ijkl} + w_{klij} & j \neq l \\ 0 & j = l. \end{cases} \tag{2.11}$$

It follows from long but straightforward calculation [20] that the general quadratic assignment problem can be reformulated as follows

$$min \left\{ cx + x^T Q x : x \in \psi_m \right\} \tag{2.12}$$

where $c \in R^{m^2}$ is generated from the operation of plant i at a given location j and arranged like vector x. In practice, Q represents the interaction cost between plant-location pairs according to the underlying graph and c allows to define the cost of the system being in the current state.

A variety of general quadratic assignment is scheduling resources with interaction cost. This class of problems arises when several activities struggle for simultaneous use of limited number of facilities. For example, when scheduling courses in a university, there might be several courses competing for the same time periods. The system faces an interaction cost when students find two or more desired courses allocated to the same time period. Since the problem of scheduling activities with interaction highly relates to the phone clone allocation in the cloud, we give a general mathematical statement of it. We have a set of activities $M = \{1, 2, \ldots, m\}$ and a set of facilities $N = \{1, 2, \ldots, n\}$ with $m > n$ and corresponding interaction cost as w_{ij}. We also define x_{ij} as follows

$$x_{ij} = \begin{cases} 1 & \text{if activity i is assigned to facility j} \\ 0 & \text{otherwise.} \end{cases} \tag{2.13}$$

The scheduling problem of minimizing the interaction cost can be formulated as follows

$$\begin{aligned} & min \sum_{i,k=1}^{m} \sum_{j=1}^{n} w_{ik} x_{ij} x_{kj} \\ & s.t. \\ & \sum_{j \in N} x_{ij} = 1 \qquad \text{for } i \in M \\ & x_{ij} \in \{0, 1\} \qquad \text{for } i \in M, j \in N \end{aligned} \tag{2.14}$$

It is worth noting that in our problem setting the risk of forming covert channels between phone clones in the same host accounts for the interaction cost.

It is not a straightforward task to modify an available approximate algorithm for a certain class of (constrained) QAP and apply it to every other problems [15]. Relative circumstances push for problem-specific approximation solution that fully exploits structural properties of a combinatorial optimization problem if present. To this end, broad understanding of solution approaches will invoke ideas and insights into each problem setting and provide general purpose tools to break it down and work out the approximate solution.

2.3.1 Solution approaches

The prevailing trend in approaching Prob. (2.10) is mixed zero-one formulations that restates it as linear problem. This approach also helps to lay foundation for advanced and creative solutions.

Defining $y_{ij}^{kl} = x_{ij}x_{kl}$, [12] offers the following mixed zero-one formulation of Prob. (2.10):

$$
\begin{aligned}
& min \ \sum_{i,j} c_{ij}x_{ij} + \sum_{i,j}\sum_{k,l} w_{ij}^{kl}y_{ij}^{kl} \\
s.t. & \\
& X \in \chi_m \\
& x_{ij} + x_{kl} - 2y_{ij}^{kl} \geq 0 \\
& \sum_{i,j}\sum_{k,l} y_{ij}^{kl} = m^2 \\
& y_{ij}^{kl} \in \{0,1\}
\end{aligned}
\tag{2.15}
$$

It is not hard to prove that (2.15) formulates the QAP correctly. Any lower bound for the linear programming is a lower bound for the corresponding QAP. Remarkable downsides of QAP linearization are huge number of new variables introduced by this method and huge number of constraints posed by the geometry of the problem. These overheads make the linearization relatively unpopular.

An alternative approximate solution is branch-and-bound algorithm. The idea of the branch-and-bound is to find the bounds of the cost function for certain subsets of the feasible set. Building up these subsets starts with the first dimension of the solution vector x and explores solutions with each of the possible values for that vector's dimension. The algorithm then carries on to the next dimension. This kind of search policy is similar to moving through a tree structure. The key aspect of this algorithm is to search the tree in the most efficient way as it might be time-intensive to evaluate the cost function at each leaf node. It follows an intelligent strategy that bounds the cost function at parent nodes and avoids expanding the calculation all the way to the leaf nodes. For instance, given subsets of the feasible set, S_1 and S_2, if the upper bound of the solutions from S_1 is lower than the lower bound of the solutions in S_2, then obviously it does not make sense to explore further the S_2. It keeps a stack of nodes that are not yet fully explored, called the open stack. At each step, the algorithm picks up a node from the open stack and expands it, or

evaluates it if it is a leaf node. If the node has children, the algorithm looks at the child nodes' lower bound and upper bound. If the child node's lower bound is lower than the global upper bound, then the child node is added to the open stack. If it is higher than the global upper bound, then it is discarded. Besides, it also updates the global upper bound found so far. If the node's upper bound is lower than the global upper bound, then the global upper bound will be replaced by the upper bound of the node. Calculating the lower and upper bounds often amounts to solving relaxed linear programming, making the whole process computationally friendly. Since the branch and bound algorithm provides better solutions with tighter bounds after each iteration, the ever-improving chain of solutions allows the algorithm to be stopped early with a quality approximate solution.

In addition to solution techniques mentioned above, tabu search (a greedy-type trading algorithm), and graph and group theory are probably the most applicable approximate approaches. These approaches inspire mathematical initiatives techniques to develop creative solutions. For example, [5] reformulates the general Boolean quadratic assignment problem with linear equality constraint into a zero-one program with convex quadratic objective function. Then, it solves the reformulated problem by a branch-and-bound algorithm and provides a tight lower bound. [6] also offers semi-definite relaxation to solve this class of hard problems.

Inspired by these mathematical resources, in Chapter 3, we provide a QP-based solution to the phone clone allocation problem. We reduce this problem to typical quadratic programming (QP) and adopt general-purpose algorithms to solve the typical QP problem. To evidence the quality of the approximate solution, we also measure the benefits of the QP approach to reduce the total risk by comparing it to some quality greedy algorithms.

Chapter 3

QP-based Solution

In this chapter, we provide a quality approximate solution to problem (2.2). We reduce the discrete optimization problem (2.2) to a typical quadratic programming QP by exploiting the structure of the problem. Then, we solve it by general-purpose algorithms for QP. We also evidence the superiority of the QP approach in reducing the total risk by comparing it to some quality greedy algorithm. Note that this QP-based solution does not scale, which is reasonable due to the hardness of problem (2.2). This QP-based solution, however, provides us with an analytical reasoning to explain why the existing RL approach breaks when applied to our problem.

3.1 Quadratic Programming

We first look for probabilistic allocation rather than deterministic allocation matrix, that is, x_{ij} in (2.2) is considered to be probability of assigning the i_{th} phone to the j_{th} host. Furthermore, we flatten the allocation matrix $X \in R^{m \times n}$ column-wise, and denote the resulted vector by $x \in R^{mn}$. Given the vector x, we are able to rewrite the objective function of problem (2.2) as quadratic function, and build new optimization problem as follows

$$\begin{aligned} & \min_{x} \ \frac{1}{2} x^T \bar{W}' x \\ s.t. & \\ & x_i \geq 0 \\ & Bx = \overrightarrow{1} \\ & Gx \leq h \end{aligned} \tag{3.1}$$

where $\bar{W}' \in R^{mn \times mn}$ is a block diagonal matrix in which the diagonal elements are equal to the complementary adjacency matrix $\bar{W}$ in (2.2), m is the number of phones and n is the number of hosts. $\overrightarrow{1} \in R^m$ denotes all-one vector. Given an identity matrix of size m, we replicate it n times and then concatenate them horizontally to build matrix $B \in R^{m \times mn}$. The second constraint requires that the summation over allocation probabilities of any phone to be 1. $G \in R^{n \times mn}$ is a block diagonal matrix in which the diagonal elements are all-ones vector from $R^{1 \times m}$, and vector $h \in R^{n \times 1}$ is comprised of hosts' capacities. The last constraint requires that the summation of allocation probabilities of all the phones assigned to any host does not exceed the host's capacity.

Equations 3.2 - 3.4 illustrate how we build these arrays

$$\bar{W}' = \begin{bmatrix} \bar{W} & 0 & \cdots & 0 \\ 0 & \bar{W} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \bar{W} \end{bmatrix}_{mn \times mn} \tag{3.2}$$

$$B = \begin{bmatrix} 1 & 0 & \cdots & 0 & \cdots\cdots & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & \cdots\cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \cdots\cdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & \cdots\cdots & 0 & 0 & \cdots & 1 \end{bmatrix}_{m \times mn}, \quad l = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_{m \times 1} \tag{3.3}$$

$$G = \begin{bmatrix} 1\cdots1 & 0\cdots0 & 0\cdots0 & 0\cdots0 & 0\cdots0 \\ 0\cdots0 & 1\cdots1 & 0\cdots0 & 0\cdots0 & 0\cdots0 \\ 0\cdots0 & 0\cdots0 & 1\cdots1 & 0\cdots0 & 0\cdots0 \\ 0\cdots0 & 0\cdots0 & 0\cdots0 & 1\cdots1 & 0\cdots0 \\ 0\cdots0 & 0\cdots0 & 0\cdots0 & 0\cdots0 & 1\cdots1 \end{bmatrix}_{(n=5) \times mn}, \quad h = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix}_{(n=5) \times 1} \tag{3.4}$$

It is obvious that the objective functions of (2.2) and (3.1) are identical. Furthermore, if we flatten any feasible point of problem (2.2) column-wise, it will fulfill the requirements of problem (3.1), thus the feasible area of (2.2) is a subset of that of (3.1). The complement adjacency matrix $\bar{W}'$ is a symmetric matrix and we express

its determinant and trace in terms of eigenvalues,

$$det\,\bar{W}' = \prod_i \lambda'_i, \quad tr\,\bar{W}' = \sum_i \lambda'_i \tag{3.5}$$

Since a phone clone cannot attack itself, the diagonal elements and the trace of matrix $\bar{W}'$ are zeros. This means that $\bar{W}'$ is an indefinite matrix; the objective function of problem (3.1) has no minimum value and decreases indefinitely, if we ignore the constraints. The boundless objective function implies that the optimal solution occurs either at a corner or on the boundary of the feasible region. In cases where the optimal solution to problem (3.1) occurs in a corner of the feasible area, it would also be a feasible point for problem (2.2). Putting it all together, the solution to the discrete optimization problem (2.2) can be achieved by solving the typical quadratic optimization problem (3.1). On the other hand, if the optimal solution happens on the boundary of feasible area, we use the QP with rounding method to find a feasible solution to problem (2.2). In what follows, we prove when the optimal solution to problem (3.1) occurs in a corner, it is a feasible point for problem (2.2), i.e., it fulfills the first binary requirement of problem (2.2).

We prove with contradiction. We assume that the corner contains both binary (0 or 1) and real-valued elements (between 0 and 1), and we split it into two sections. The first part consists of all the binary elements, and the second part consists of real-valued elements termed by $x' \in R^{d_{x'}}$ ($d_{x'} \leq mn$). The second part should stay within the constraint of hosts' capacities which might be either a loose constraint (inequality) or a tight constraint (equality). Note that the loose constraints do not stop the solver algorithm from sliding freely around the corner, and if we point out a contradiction given only the tight constraints, it will also result in contradiction in cases where we have both tight and loose constraints (A tight constraint poses a stricter restriction on the movement of the solver algorithm around the corner than a loose constraint). Thus, to point out the contradiction, we only keep the tight constraint and discard the rest.

This analysis leads to the following constraints on x',

$$\begin{bmatrix} M_1 \\ M_2 \end{bmatrix} x' = \begin{bmatrix} \overrightarrow{1} \\ v_2 \end{bmatrix}, \quad 0 < x'_i < 1 \tag{3.6}$$

where $M_1 \in R^{d_1 \times d_{x'}}$ ($d_1 \leq m$) and $\overrightarrow{1} \in R^{d_1}$ uphold the constraint that the summation

over allocation probabilities of any phone should be 1. $M_2 \in R^{d_2 \times d_{x'}}$ ($d_2 \leq n$) and $v_2 \in R^{d_2}$ state that the summation of allocation probabilities of all the phone clones assigned to a host equals the host's capacity. It is obvious that we can move from x' in any direction $z \in R^{d_{x'}}$ and maintain the feasibility as long as z belongs to the null space of matrix $M = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}$, i.e., $Mz = 0$. If we prove that the null space of matrix M is not empty, this will be in contradiction with the fact that x' is on the corner of feasible area, which completes the proof.

We make use of the fact that it is impossible to move freely over a hyperpalane (null space of matrix M) if we stand on a vertex of the feasible area which is the intersection of some hyperplanes in the space. By the assumption we made so far, it is evident that $d_{x'} \geq 2d_2$ and $d_{x'} \geq 2d_1$. We break it down further into two scenarios:

1. $d_1 + d_2 < d_{x'}$, which in turn implies that the rank of matrix M is less than $d_{x'}$ and the null space of M is not empty.

2. $d_1 = d_2 = \frac{1}{2}d_{x'}$. In this scenario, matrix M_1 is made up of two diagonal matrix of size d_1 which are concatenated horizontally, and matrix M_2 is a $d_2 \times d_{x'}$ block diagonal matrix in which the diagonal elements are all-one vectors $\overrightarrow{1} \in R^{1\times 2}$. For instance, if $d_1 = d_2 = \frac{1}{2}d_{x'} = 3$, matrix M would be,

$$M = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{3.7}$$

   This shape of M yields a singular matrix with non-empty null space because any row of this matrix can be written as a linear combination of other rows.

Singular graph Singular complementary adjacency matrix reduce the chance of optimal solution falling into a corner. Although it is quite rare to come across a random graph with singular complementary adjacency matrix, we offer slight changes to the matrix $\bar{W}$ to tackle this issue without drifting too much away from the optimal solution. The adjacency matrix of the communication graph $\bar{W}$ can be restated by

eigenvalue decomposition as,

$$\bar{W} = \sum_{i=1}^{m} \lambda_i q_i q_i^T \tag{3.8}$$

where λ_i and q_i are eigenvalue and eigenvector, respectively. In case matrix $\bar{W}$ is singular, we add a tiny value ϵ to zero eigenvalue(s). The matrix $\bar{W}$ represents the underlying communication graph between the phones. The change to the communication graph caused by this manipulation is quite insignificant that does not turn the solver away from the optimal solution (allocation). However, this will help push the solver algorithm to settle on a corner of the feasible area, when the gradient of the objective function is not perpendicular to the boundary.

3.2 Heuristic Algorithms

To measure the benefits of reformulating the phone allocation problem as typical QP, we compare the performance of QP-based solution with quality greedy algorithms that were previously used to address this problem. These algorithms have shown the ability to find close to optimal solutions [23]. In what follows, we introduce two algorithms [23]: *maximum-conflict-first (MCF)* and *highest-degree-first (HDF)*. We define the node degree as the total weights of the edges crossing the node.

The main idea of *maximum-conflict-first (MCF)* is to allocate phone clones that have the most conflict (i.e., the least node degree in the communication graph) first as follows:

- Step 1: Sort phone clones in the ascending order of their node degree in the communication graph.
- Step 2: Select the phone clone i which has the least degree and has not been allocated.
- Step 3: Allocate the phone i to the host with the least potential risk of covert channels. When the number of assigned phone clones in a host reaches the capacity, it does not transfer any more phone clones to that host.
- Step 4: Repeat Steps 2-3 until the last phone.

The main idea of *highest-degree-first (HDF)* contrasts with that of MCF in that it allocates the least conflict (i.e., the highest node degree in the communication graph) first. Steps 3 and 4 of HDF are the same as those in MCF.

3.3 Experiments

3.3.1 Experimental setup

The solver algorithm's running time for typical QP optimization might be relatively longer than that of heuristic algorithm for phone clone allocation problem. A fix to this issue can be setting a limit on the solver algorithm's iterations yielding a quality approximation solution. We solve the typical QP by *trust-constr* and *SLSQP* algorithms available on *SciPy* library. Our analytical method outperforms heuristic algorithms in terms of potential risk. The *trust-constr* algorithm is the most appropriate for large-scale problems. It is also noteworthy that *cvxpot* software is not appropriate in this case because it restrictively deals with convex quadratic problems while problem (3.1) is non-convex. It also complains when the size of the problem grows larger.

To set up the experiments, we initialize the adjacency matrix of communication graph $W \in R^{m \times m}$ from a uniform distribution on the interval $[0, 1)$. A larger value of w_{ij} indicates a stronger tie between phone clones i and j. From matrix W, we build the block diagonal matrix $\bar{W}' \in R^{mn \times mn}$ as explained before. Furthermore, x_{ij} in the new formulation (3.1) is considered as the probability of assigning phone clone i to host j, and we start the solver algorithms with allocation matrix $X \in R^{m \times n}$ from a uniform distribution on the interval $[0, 1)$. We build the vector x in (3.1) by flattening the matrix X column-wise. To have a quantitative figure of gained performance over greedy algorithms, the comparison among these candidate algorithms is carried out in different scenarios where hosts have equal or unequal relative capacities. In the first scenario (Fig. 3.1), we set the same capacity for the hosts, and the total capacity of the system equals the number of phone clones. Actual capacities of the hosts are obtained by $m \times [0.2, 0.2, 0.2, 0.2, 0.2]$ where m is the number of phone clones and the second operand is the relative capacities. In the second scenario (Fig. 3.2), we consider the hosts with different capacities, and the cloud capacity is larger than the total number of phone clones.

3.3.2 Results

The following figures show the outcomes. According to Fig. 3.2 and Fig. 3.1, we observe that improvement in performance by QP programming is notable which validates our analytical approach.

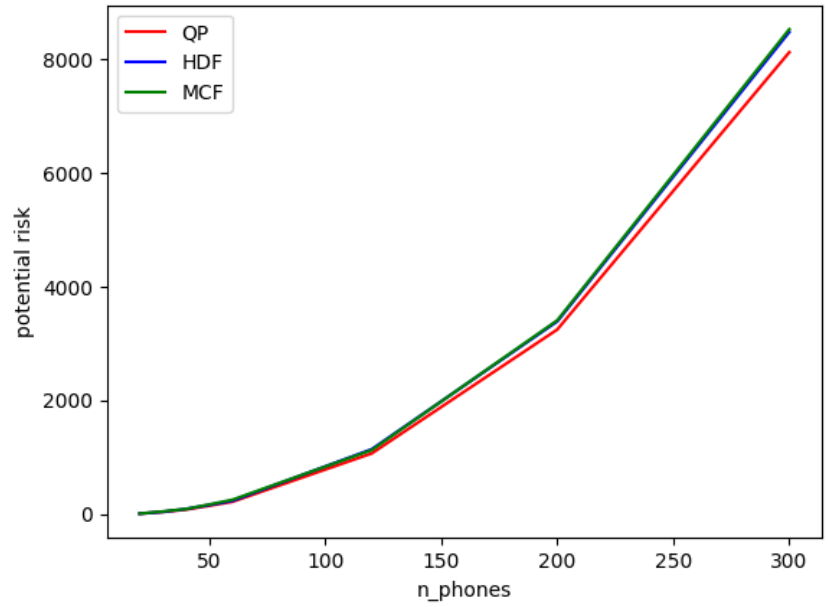


Figure 3.1: Relative hosts' capacities are set to $[0.2, 0.2, 0.2, 0.2, 0.2]$.

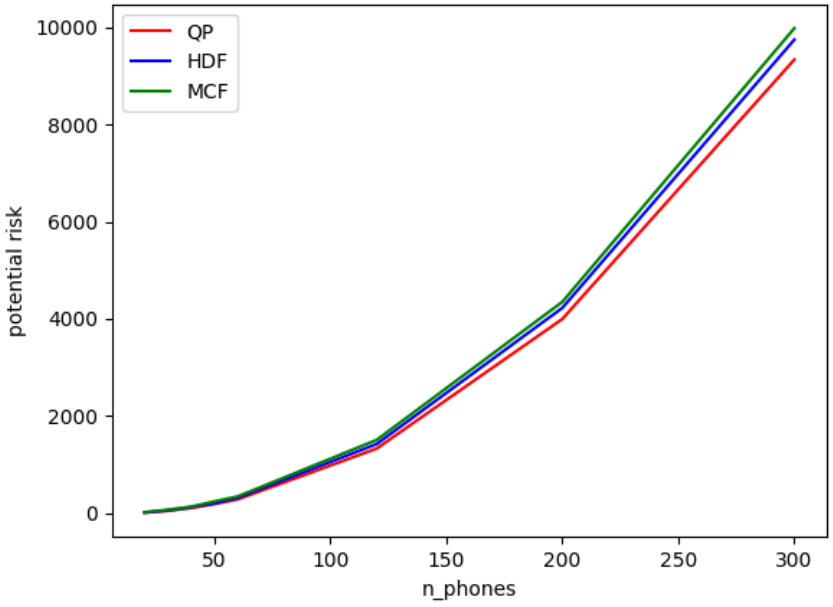


Figure 3.2: Relative hosts' capacities are set to $[0.1, 0.1, 0.2, 0.5, 0.4]$.

Chapter 4

RL-based Solutions

In this chapter, we leverage reinforcement learning (RL) to solve combinatorial optimization problems. Regarding the hard nature of these problems, RL looks like a natural candidate to make decisions in a more principled way. We detail a methodology to integrate RL and combinatorial optimization. We build a framework composed of Graph Neural Network (GNN) and Deep Q-network (DQN) to address the phone clone allocation problem, i.e., problem (2.2). Our important finding is that despite RL's success in solving some hard combinatorial optimization problems [4, 10], the power of RL in its current incarnation is limited if the problem under consideration does not have desired features, which will be disclosed in this chapter.

4.1 Q-learning

To evidence our argument about limits of RL for constrained combinatorial optimization, we first build a solid model that can inherit the problem characteristics and effectively reflect the combinatorial structure of the graph. The model consists of an encoder and a decoder both parameterized with trainable coefficients. Fig. 4.1 shows an overview of the architecture.

This model can be initialized in any state (not necessarily feasible) and seeks to improve on any proposed solution. The encoder, essentially a variant of message-passing graph network, receives the current allocation matrix (solution) as a set of allocation vectors x_i, $i \in \{1, \ldots, m\}$. It first maps $x_i \in R^n$ (recall that n denotes the number of physical hosts) into a higher dimensional vector to produce initial node embedding $h_i^0 \in R^{d_h}$. These linear transformations share weights across all allocation

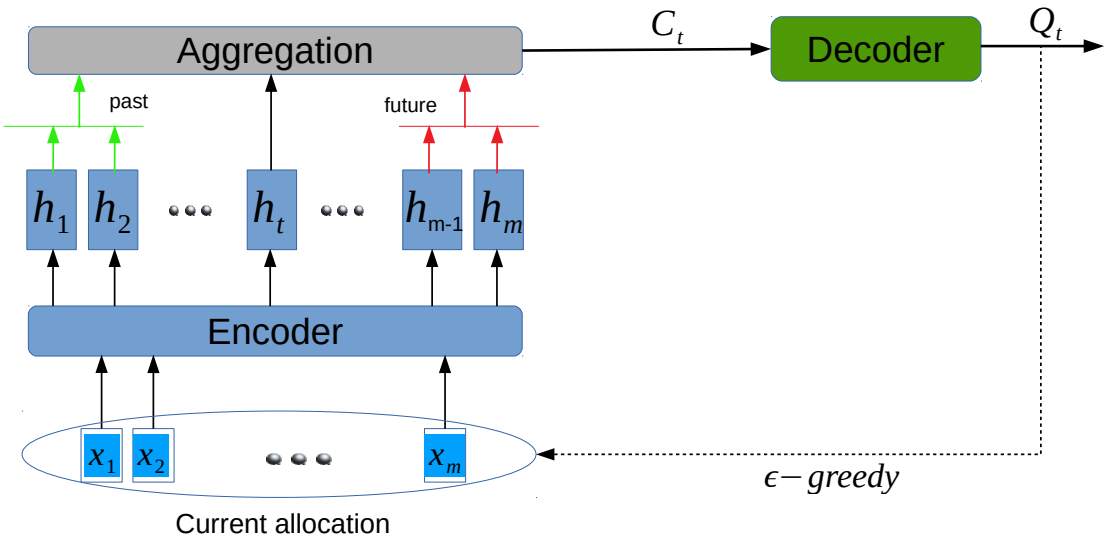


Figure 4.1: Q-learning model architecture.

vectors. We build several attention layers upon initial embedding. Each attention layer updates node embeddings repeatedly according to

$$h_i^{(l+1)} = relu(\theta_3 x_i + \theta_2 \sum_{j=1}^{m} \bar{w}_{ij}\, relu(\theta_1 h_j^{(l)} + \mu_1)), \tag{4.1}$$

where $\theta_3 \in R^{d_h \times n}$, $\theta_2, \theta_1 \in R^{d_h \times d_h}$ and $\mu_1 \in R^{d_h}$ are trainable parameters, $relu$ function introduces nonlinear operation and $\bar{w}_{ij} = 1 - w_{ij}$ is the complementary of the weighted edge between nodes i and j. $h_i^{(l)}$ and $h_i^{(l+1)}$ are the inputs and outputs of layer l, respectively. The attention layers let each node embedding to incorporate further graph structured data about the interaction between other nodes. To make the message passing process more powerful, Eq. (4.1) adds a pre-pooling operation represented by parameters θ_1 and μ_1 followed by $relu$ non-linearity function before looking over node embeddings (we have implemented the model with and without this pre-pooling operation and reported the best results). Also we found it useful to incorporate a residual path from initial embeddings $h_i^{(0)} = \theta_3 x_i$ at each attention layer.

Once the final embedding for each node is computed after L recursions, they get to the actor as the environment state. The actor constructs new solution sequentially, reallocates a phone clone per timestep (iteration), and improves on current solution. At each iteration $t \in \{1, \ldots, m\}$, it first aggregates node embeddings that were re-

allocated before, with respect to communication history between the phone at hand and reallocated phones. This is done through the aggregation layer on top of the encoder, and the outcome is termed left context vector c_t^{left}. It then aggregates node embeddings that are going to be reallocated in the rest of the episode. The outcome is termed right context vector c_t^{right}.

$$c_t^{left} = \sum_{t'=1}^{t-1} \bar{w}_{tt'} h_{t'}^{(L)}, \quad c_t^{right} = \sum_{t'=t}^{m} \bar{w}_{tt'} h_{t'}^{(L)}, \tag{4.2}$$

where $h_{t'}^{L}$ is the embedding of node t' from last attention layer. We set the number of iterations of each training episode to be the multiples of the number of phones m. The reason we set the length of episode equal to the multiples of m is first, to give each phone equal chance to be reallocated and second, there is no definite termination state for this problem.

These context vectors provide the decoder in the next phase with broad insight of the current state of the environment. Decoder later receives these context vectors as input. It then processes them through separate channels to acquire a better view of the environment state, and accordingly takes action, i.e., allocates phone clone t (the phone clone currently at hand) to a host following a learned optimal policy π. During the training phase the ever-improving policy π is set to find the true action value, and constantly improves as the decoder goes through further training episodes. We use neural network as parameterized Q-function to approximate the state-action values as follows.

$$Q(c_t; \Theta) = \theta_6 relu(c_t), \quad c_t = [\theta_5 c_t^{left}, \theta_4 c_t^{right}], \tag{4.3}$$

where c_t is called context vector and $[\cdot, \cdot]$ is the concatenation operation. $\theta_4, \theta_5 \in R^{d'_h \times d_h}$, and $\theta_6 \in R^{n \times 2d'_h}$ are trainable parameters. $Q(c_t; \Theta)$ is an n-dimensional vector that represents the values of allocating current phone to any of hosts. It depends on a set of 6 parameters $\Theta = \{\theta_i\}_{i=1}^{6}$. The parameter set Θ will be trained, and it is expected to find the true state-action values. Accordingly, decoder makes decision with respect to the state-action values, i.e., $a_t = argmax\, Q(c_t; \Theta)$ where a_t determines the host to which the current phone clone will be allocated. Breaking down the context vector into c_t^{left} and c_t^{right} brings the benefit to allow the decoder to figure out more authentic and reliable action values at each iteration.

We deploy k-step Q-learning algorithm to train this model. The key aspect of

k-step Q-learning is that it is mindful of the issue of delayed reward where the final reward of interest to the agent is received by adding attainable future reward to immediate reward during a training episode. In our problem setting, the true value of an action is only revealed after several subsequent phone clone allocations, e.g., the optimal policy may allocate the current phone clone to a relatively high risk host while planning to displace some problematic phone clones from this host later in the training episode. We wait k steps before measuring each action value to collect more accurate estimate of future rewards.

At each training iteration t, actor makes either greedy decision or random decision with ϵ-probability yielding corresponding reward and the next state

$$a_t = \begin{cases} \text{random host} & w.p.\ \epsilon \\ argmax\, Q(c_t; \Theta) & \text{otherwise} \end{cases} \tag{4.4}$$

This ϵ-greedy policy aims to set a balance between exploration and exploitation, i.e., between exploring the search space and greedy decision to get the maximum return. The hyperparameter ϵ gradually fade away so that at the end of the training process the actor will make greedy decision according to true action values.

After each decision, the node embeddings gets updated based on the new allocation matrix and the underlying communication graph to reflect the change in the problem environment after reallocating the most recent phone.

We call this architecture policy network, and to alleviate the stability issue during training we build an identical architecture called target network [22]. Policy network is the one that computes the $Q(c_t; \Theta)$ to allocate the phone clones accordingly, and learn to improve during training. Meanwhile, target network computes a counterpart $\hat{Q}(c_t; \Theta)$ termed target action values which is used to calculate the loss. The target network keeps its parameters unchanged most of the time and updates them with the policy network's parameters every so often. At each training iteration t, the loss is defined as the absolute gap between action value $Q^{\pi}(c_t; \Theta)$ and the expected action value. The expected action value is the accumulated reward over the next k iterations $r_{t:t+k}$ plus the target action value $max\, \hat{Q}(c_{t+k}; \Theta)$.

$$\begin{aligned} loss &= (Q^{\pi}(c_t; \Theta) - (r_{t:t+k} + \gamma\, max\, \hat{Q}(c_{t+k}; \Theta))^2 \\ where \quad r_{t:t+k} &= \sum_{t'=0}^{k-1} r_{t+t'} \end{aligned} \tag{4.5}$$

where $Q^{\pi}(c_t;\Theta)$ represents the value of action taken with respect to ϵ-greedy policy over the set of action values denoted by $Q(c_t;\Theta)$. γ is a discount factor, and reward r_t at iteration t is defined as the cut back in total potential risk and penalty term after new phone clone is reallocated. We impose the penalty term for infeasible solution on the reward function to push the actor toward a feasible solution with respect to the constraints in (2.2). It was observed that the hosts with capacity less than or equal to the average value m/n are always working with full capacity no matter which algorithm is adopted (reinforcement learning algorithm or heuristic algorithm), and the phones are allocated as evenly as possible to the rest of the hosts. Thus, given the constraint of hosts' capacities, we set a desired distribution of phones over the hosts and define the penalty term as the Kullback–Leibler (KL) divergence between the actual distribution, obtained from ongoing allocation matrix, and the desired distribution. Although KL divergence is a non-negative value, the penalty term can be either rewarding (positive) or punishing (negative). Since we define the penalty term as the cut back in KL divergence after a new phone allocation, after k iterations, the intermediate terms would be canceled out and the outcome amounts to the gap between the last KL term and the first KL term yielding either positive or negative outcome. In a sense, the penalty term measures how the actual distribution becomes closer to the desired one after the last allocation compared to k steps earlier. We also tune a regularizer hyperparameter that helps to reach a balance between potential risk and penalty term. This formulation is valid until the last iteration. By definition we set value function to 0 at the state induced by last iteration.

The parameters of policy network get updated at each iteration to minimize the loss. As the gap fades away, the policy network gets closer to the true action values and subsequently the optimal policy. Algorithm 1 concisely explains the process. As illustrated in the algorithm, we train the model several episodes, each with a random graph (the weight of the edge between two arbitrary nodes comes from a uniform distribution over $(0, 1)$) or a graph generated through stochastic block model [1]. We also evaluate the model's training progress with the validation dataset (sample graphs and random initial allocation matrix) at the end of each episode. Moreover, regarding the sample efficiency of the algorithm, instead of throwing away past samples, we record the obtained tuple from each iteration $(c_i, a_i, r_{i:i+k}, c_{i+k})$ in the replay memory. We run the optimization step on every iteration which picks a random batch of size B from the replay memory to train the ever-improving policy via stochastic gradient descent method.

Algorithm 1: k-step Q-learning

Input : number of episodes Z, and replay batch B
Output: Θ

Initialize experience replay memory M to capacity N;
Initialize agent network parameters Θ;
for *episode = 1 ... Z* **do**
  $g \sim \text{SampleGraph}(G)$;
  $s \sim \text{SampleSolution}(S)$;
  for $i = 1 \dots m$ **do**
    Compute the context vector c_i;
    $a_i = \begin{cases} \text{random host} & w.p.\ \epsilon \\ argmax\, Q(c_i; \Theta) & \text{otherwise} \end{cases}$;
    if $i \geq k$ **then**
      Add tuple $(c_{i-k}, a_{i-k}, r_{i-k:i}, c_i)$ to M
    end
    Sample random batch from replay memory $B \overset{i.i.d}{\sim} M$;
    Update Θ by Adam over the loss (4.5) averaged over B
  end
end

It is noteworthy that this architecture is also mindful of scalability, i.e., the number of parameters is independent of problem size. It can generalize to problems larger than it was trained on.

4.2 Policy Gradient

As an alternative solution, we could have allocated the phone clones simultaneously rather than sequentially. The motivation is that each node embedding from graph encoder seems to acquires a broad insight of the communication graph topology and the current solution (allocation matrix) so that the decoder can make the right decision about all phones at once. For instance, according to the message passing between the i_{th} phone and others, it would learn how they feel toward a certain host, whether they discourage the phone from going into the host or not, and how favourable or unfavourable would be to allocate the phone to a certain host. It will also learn complementary information about the interaction between other phones that help the phone make the right decision. For instance, assume phone i is discouraged by a stranger (stranger is a phone clone with no background communication with phone i) from going to a host, but the stranger itself was alarmed about going to the host by others. According to this complementary information that phone i has learned, it may not take the stranger seriously when it comes to making decisions about the host. Actually, the learning procedure is more sophisticated, and each phone obtains further information that we cannot provide a tangible interpretation for. It is noteworthy that we also encounter kind of intangible knowledge from Convolutional Neural Networks (CNNs) as we cannot decipher feature maps (activation values) of the deeper layers.

Fig. 4.2 shows a general view of the model architecture. This model can be initialized in any state and seeks to improve on any proposed solution. We use the same powerful encoder like the one adopted in Q-learning because, to the best of our knowledge, it is the *best* fit for our problem setting. Furthermore, we build a node-wise fully connected layer followed by a softmax function on top of the encoder to map the output logits to the hosts. The encoder computes a vector of logits per phone which encompasses the potential risk of landing on different hosts and the inclination of other phones. Each node will go to the host with the highest probability. For this architecture the number of parameters does not depend on the problem size, however, experimental results show that it is not capable of generalizing to problems

larger than it was trained on.

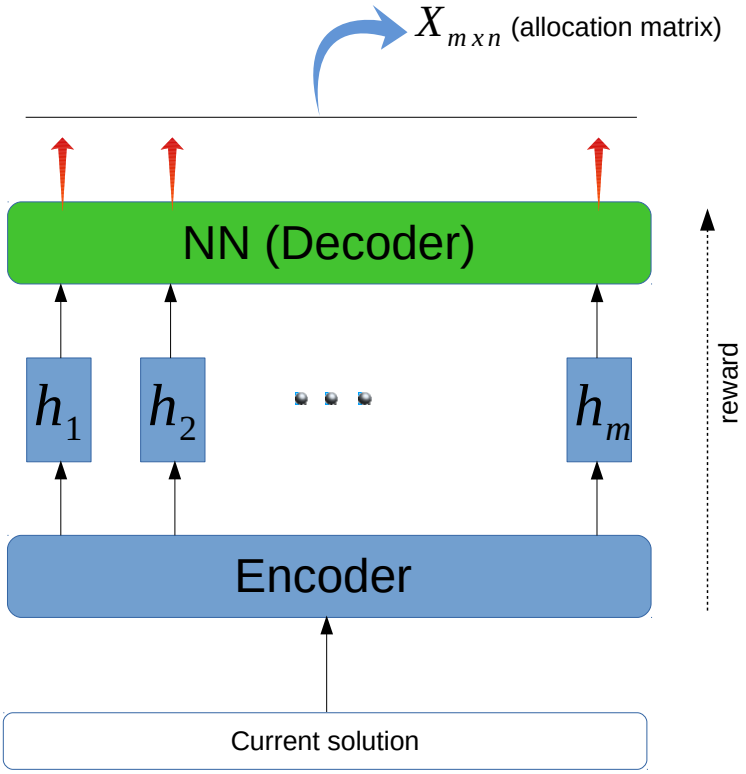


Figure 4.2: Policy gradient model architecture.

Given the communication graph g and the current allocations s, the model computes probability distribution $\pi_\Theta(a/g, s)$ from which it takes action a with the highest probability, and reallocates all the phone clones at once. π is the parameterized policy, and Θ is the model's trainable parameters. After reallocating the phones, the reward $r(a)$ is the cut back in potential risk (2.2). We define the objective function as the average reward $\Omega(\Theta) = E_{\pi_\Theta(a/g,s)} r(a)$ and the underlying principle of this approach is to constantly increase the objective function. In previous works [11, 10, 16] that adopt RL-based model to solve a certain class of hard problems such as TSP, the goal is to find the best order of actions by keeping eyes on future rewards, i.e., it might sacrifice reward at the moment with the aim of gaining more reward in the future. In contrast, the optimum policy for our problem setting is capable of constantly improving the solution. Indeed, here, it does not make sense at all that the optimum policy sets back, making an inferior allocation with the aim of finding a better solution in future. The optimal policy is expected to improve on the current solution, and does

not undergo back and forth. As a result, the right way to improve the policy is to take the direction suggested by obtained reward per training step rather than long-term return.

We maximize the objective function $\Omega(\Theta)$ and learn the optimal parameters Θ by gradient descent using REINFORCE gradient estimator with baseline b

$$\nabla\Omega(\Theta) = E_{\pi_\Theta(a/g,s)}[(r(a) - b)\nabla log\pi_\Theta(a/g, s)] \tag{4.6}$$

The baseline is set to lessen the gradient variance and accelerate the learning process. A common choice for baseline is exponential moving average $b = \chi$ with a decay factor β. In the first iteration $\chi = r(a)$ and then it gets updated as $\chi = \beta\chi + (1 - \beta)r(a)$ in subsequent iterations.

At each training step, we draw a batch ($=B$) of i.i.d random graphs and random allocation matrices (solutions) from the training dataset. Then, we take action based on the most current policy, and update the policy parameters according to the direction suggested by the reward (4.6). We approximate the gradient $\nabla\Omega(\Theta)$ by Monte Carlo sampling. Algorithm 2 concisely explains the process.

Algorithm 2: Iteration reward

Input : number of epochs E, steps per epoch Z , batch size B
Output: Θ

Initialize policy network parameters Θ;
for *epoch = 1 . . . E* **do**
  for *step = 1 . . . Z* **do**
    $g_l \sim$ SampleGraph(G) for $l \in \{1 \dots B\}$;
    $s_l \sim$ SampleSolution(S) for $l \in \{1 \dots B\}$;
    $a_l \sim$ TakeAction($(g_l, s_l), \pi_\Theta$) for $l \in \{1 \dots B\}$;
    $\nabla L = \frac{1}{B}\sum_{l=1}^{B}[(r_l - b_l)\nabla_\Theta \log \pi_\Theta(a_l|(g_l, s_l))]$;
    $\Theta \sim Adam(\Theta, \nabla L)$
  end
end

To make a comprehensive examination, we also worked out a different scenario in which the model itself learns the right trajectory and comes to improve the solution constantly. The idea is to start with a random solution, create a short sequence of actions termed trajectory τ with probability $\pi_\Theta(\tau/g, s)$ by running the policy π. Then, we obtain the accumulated reward over the trajectory $r(\tau)$, and move the policy forward according to the direction suggested by the trajectory return. The objective function that we seek to maximize in this scenario is defined as average return $\Omega(\Theta) = E_{\pi_\Theta(\tau/g,s)} r(\tau)$. Algorithm 3 concisely explains the process.

Algorithm 3: Trajectory return

Input : number of epochs E, steps per epoch Z , batch size B
Output: Θ

Initialize policy network parameters Θ;
for *epoch = 1 . . . E* **do**
  for *step = 1 . . . Z* **do**
    $g_l \sim \text{SampleGraph}(G)$ for $l \in \{1 \dots B\}$;
    $s_l \sim \text{SampleSolution}(S)$ for $l \in \{1 \dots B\}$;
    $\tau_l \sim \text{SampleRollout}((g_l, s_l), \pi_\Theta)$ for $l \in \{1 \dots B\}$;
    $\nabla L = \frac{1}{B} \sum_{l=1}^{B} [(R_l - b_l) \nabla_\Theta \log \pi_\Theta(\tau_l | (g_l, s_l))]$;
    $\Theta \sim Adam(\Theta, \nabla L)$
  end
end

Because problem (2.2) is a hard problem and we do not know the optimal solution in advance, algorithms 2 and 3 may hit and pass the optimal solution. However, they are expected to learn and make a null decision in such a situation. The null decision means generating a new solution which is the same as the current solution; otherwise, it will receive negative reward.

4.3 Experimental results

4.3.1 K-step Q-learning

Getting a Q-learning algorithm to work is pretty challenging and takes much effort to tune the hyperparameters. We found out that a very deep graph encoder does not show improvement on the outcome. Thus, we construct a reasonable size graph encoder comprised of a mapping function and two attention layers. We tried out three values $32, 64, 128$ regarding the node embeddings' dimension, and $d_h = 64$ came out with the best result. Regarding the hidden dimension of decoder, we observed that $d'_h = 64$ was actually the most suitable size. Through the course of training, we decrease the exploration hyperparameter ϵ as follows

$$\epsilon = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) * e^{\frac{-steps_{done}}{\epsilon_{decay}}}, \tag{4.7}$$

where $steps_{done}$ represents the number of training steps, $\epsilon_{start} = 0.9$ and $\epsilon_{end} = 0.05$ are the initial and final values of ϵ, and $\epsilon_{decay} = 200$ is the pace of decay throughout the training process. We also found that the episode length equal to the number phones leads to better result compared with other setups. The number of hosts is set to 5. For the constrained problem, the hosts' relative capacities are set as $[0.1, 0.1, 0.2, 0.3, 0.3]$, respectively. Setting the hyperparameter k (delay) higher than 2 worsened the result, and relatively small discount factors resulted in a lower risk. Insignificant delay and discount factor means that the optimal model tends to take on a greedy policy when allocating the phone clones sequentially. Besides, model parameters get updated at each step via a stochastic gradient descent method. We used the Adam optimizer for this purpose as it showed improvement over the SGD optimizer. Also, the random batch size B for Adam optimizer during training is set to 128.

Initializing the parameters was also a challenging issue because there was a huge gap between the action values calculated by the decoder and the reward returned by the environment at each training iteration. To fix this issue and get the model to start learning, we introduce a new hyperparameter that scales the reward values without loss of generality. We tune this hyperparameter to reach a balance between these two terms so that the Q-learning algorithm can take into account both terms when it calculates the expected action value (4.5).

We consider two scenarios to construct problem instances. In the first scenario, we draw random graphs in which the weight of the edge between two arbitrary nodes comes from a uniform distribution over $(0, 1)$. We train the model on communication graphs with 20 and 40 phone clones for both unconstrained and constrained problems (unconstrained problem means that the hosts' capacity constraints are relaxed). Fig. 4.3 shows the training progress over time that evidences the policy improvement during the course of training, i.e., the learning is going on in practice. It shows the return, the cut back in the potential risk and the penalty term, over training episodes. The return is computed at the end of each episode by getting the average over a batch (=128) of problem instances. Moreover, Fig. 4.4 shows the number of phone clones allocated to the hosts by RL-based model. We can see that they fairly stay within the last constraint of the problem (2.2). Trained models are then generalized and tested on a range of graph sizes, from 20 nodes to 80 nodes. For each problem size, we test the model on a batch (= 64) of graphs and report the averaged result. Fig. 4.5 and Fig. 4.6 show the potential risk obtained from RL-based methods for both constrained and unconstrained scenarios. RL-based approach outperforms the random solution; however, the potential risk obtained from RL-based approach is higher than the risk from analytical approach explained in Chapter 3. The gap between them widens as the problem size increases. This suggests that RL-based methods does not bring benefit over QP-solvers for our combinatorial optimization problem.

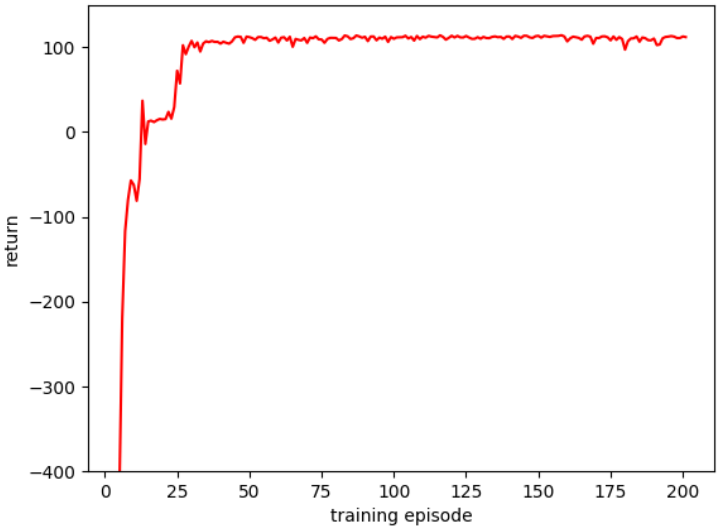
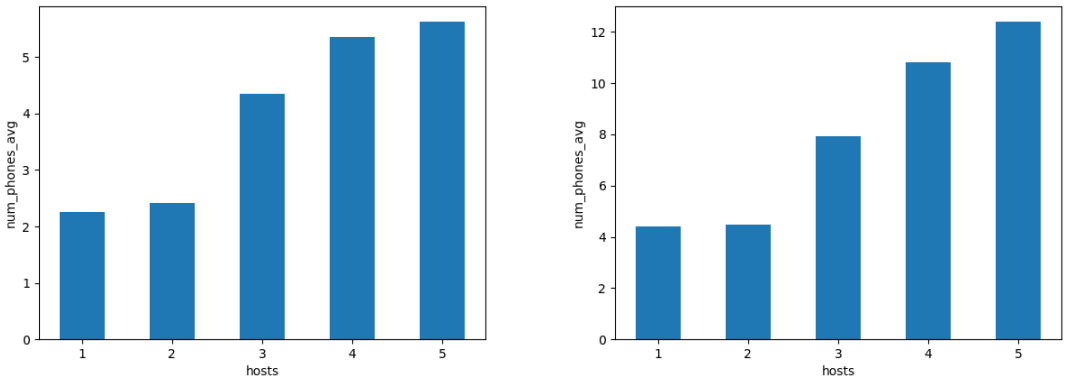


Figure 4.3: Training progress over time, total number of phones = 40.



(a) Total number of phones = 20. (b) Total number of phones = 40.

Figure 4.4: Number of phones allocated to different hosts.

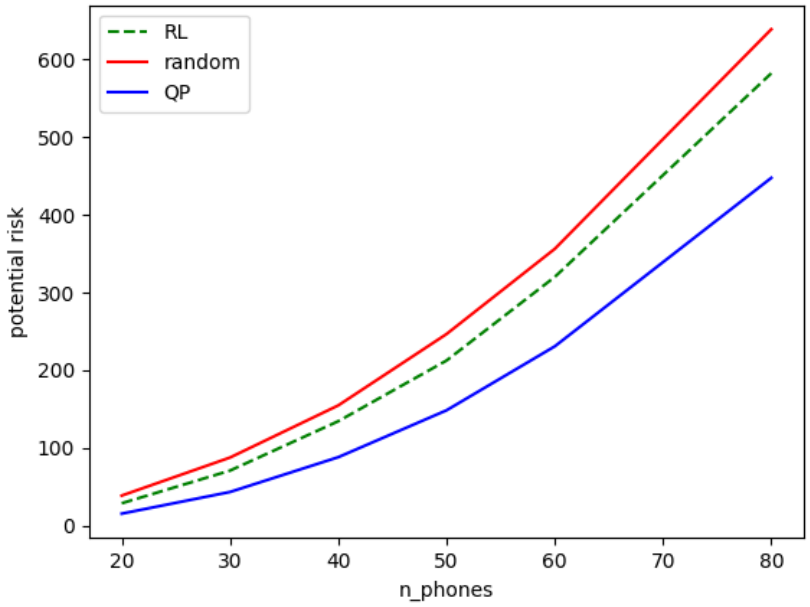


Figure 4.5: Total potential risk; number of hosts = 5.

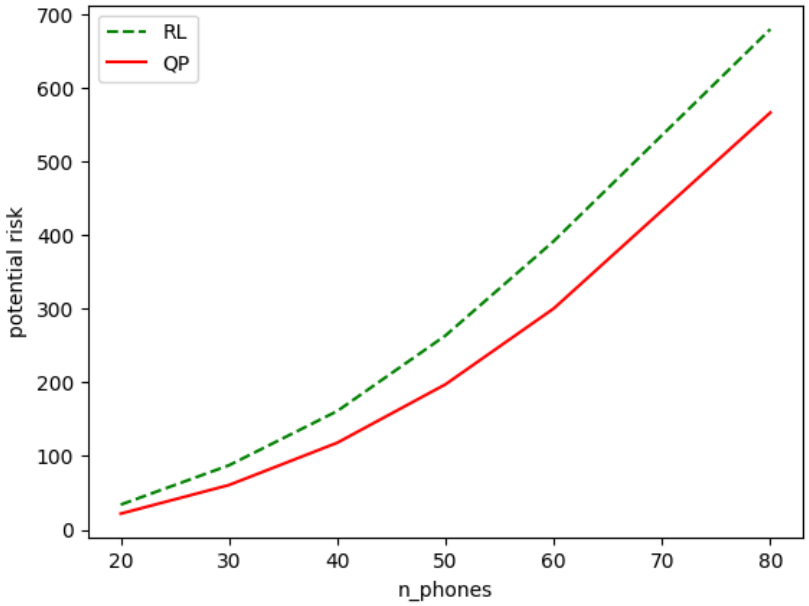


Figure 4.6: Total potential risk; number of hosts = 5, and relative hosts' capacities are set to [0.1, 0.1, 0.2, 0.3, 0.3].

In the second scenario, we use stochastic block model (SBM) to generate real world communication graphs of different sizes with 4 clusters. To construct the graphs in our experiments, we use the *SparseBM* which is a python module for handling sparse graphs with clusters. The module requires two parameters, number of nodes in each cluster and the connection probabilities within/between clusters. In our experiment, the nodes are evenly distributed among the clusters. $C = [c_{ij}]_{4\times 4}$ represents the connection probability as follows

$$C = \begin{bmatrix} 0.5 & 0.018 & 0.006 & 0.031 \\ 0.018 & 0.37 & 0 & 0 \\ 0.006 & 0 & 0.55 & 0.12 \\ 0.031 & 0 & 0.12 & 0.43 \end{bmatrix} \tag{4.8}$$

where the diagonal elements are the interconnection probabilities within the clusters and $c_{ij}, i \neq j$ is the connection probability between clusters i and j. We train the model on communication graphs with 100 phones for the constrained problem. Fig. 4.7 reports on the training progress. It shows the policy network is gaining more return over training episodes that accounts for the fact that the policy improves with training. The return, as mentioned earlier, is the cut back in the potential risk and the penalty term. Fig. 4.8 shows the number of phone clones allocated to the hosts by RL-based model which fairly keeps to the constraints regarding the maximum number of the phones that each host could serve. We then evaluate its performance on communication graphs of different sizes ranging from 40 nodes to 100 nodes. For each problem size, the model is tested on a batch (= 64) of graphs and the averaged results are reported. Fig. 4.9 shows the potential risk obtained from RL-based method for the constrained phone allocation. As we can see RL-based method cannot outperform QP-solvers although it gets closer to QP-performance in the SBM scenario.

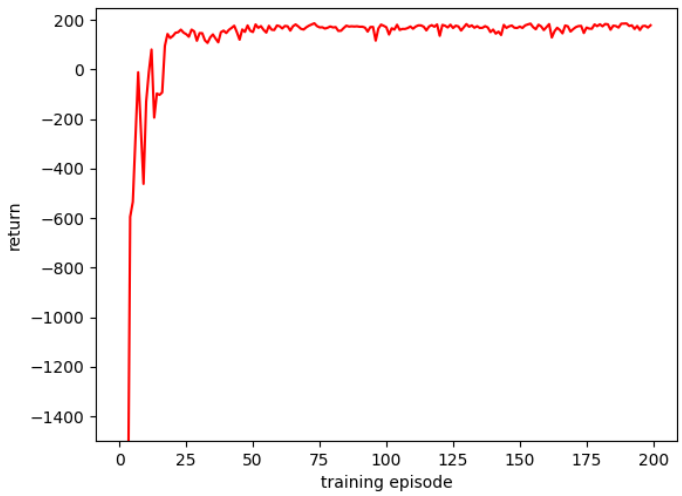


Figure 4.7: Training progress over time, total number of phones = 100.

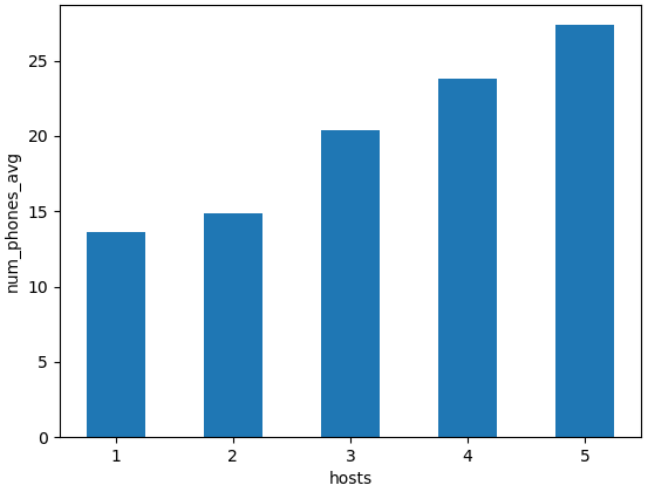


Figure 4.8: Number of phones allocated to different hosts. Number of phones = 100

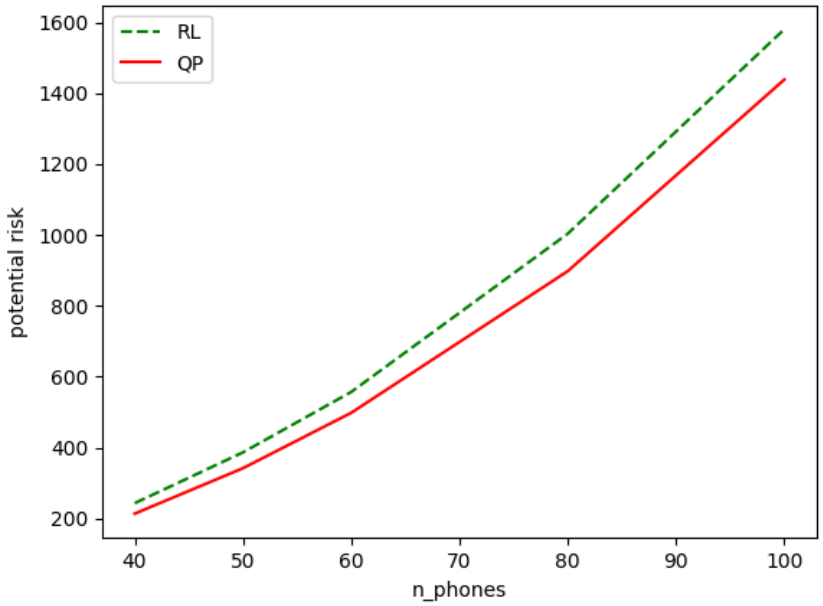


Figure 4.9: Total potential risk; number of hosts = 5, and relative hosts' capacities are set to [0.1, 0.1, 0.2, 0.3, 0.3].

Ablative analysis

We conclude that unlike the previous successful reinforcement learning applications to hard problems, existing RL framework fails to work well in the phone clone allocation problem. To find out the cause of the fault, we traced the model's outcome all the way back to the node embeddings, inner calculations inside each layer, and the inputs. We discovered that the encoder's final node embeddings are not distinguishable enough while the environment state changes from one iteration to another, and therefore the decoder cannot trace the varying environment states. As a result, it is difficult for the decoder to figure out meaningful and authentic action values throughout the training process. This shortcoming is caused by few one-hot input vectors corresponding to different hosts being shared between many phones. In this case, node embeddings may sound different, but they actually based on a limited pool of hosts. In contrast, in previous problems such as TSP where the RL-based method thrives, the context vectors distinctively differ from one training iteration to another since the input vectors carry diverse and abundant information. As a result, they thoroughly turn over to the decoder the change in the environment after the most recent allocation. Informative context vectors empower the decoder to figure out the state-action values more accurately. This issue shows the importance of the graph encoder to learn the problems' heuristics and generate exemplary embeddings.

Furthermore, in previous works, RL model adds one element at a time to the current partial solution. This incremental approach allows the decoder to use a mask function that removes the partial solution from the search space and narrows the focus of the investigation. By reducing the search space, the helper function conducts the model to find a better solution. Besides, the mask function rules out the impact of the nodes that have already been touched from the context vectors which helps the decoder to discriminate through subsequent iterations. Nevertheless, the helper function is a problem-specific asset, e.g. in TSP, we need to figure out the next node (city) to be added to the partial solution (partial tour). Similarly, in our problem we need to decide the host to which we will assign the next phone. In this case, it is not a straightforward task to manually mask the hosts' pool or limit the reallocation of some phones (e.g. those with poor communication background). It is irrational to naively use a masking function since it limits the feasible domain of actions, worsening the solution. In essence, the mask function is irrelevant to the circumstances of our problem because incrementally constructing an order of nodes is not an inherent

requirement. We believe, RL model itself should evolve as it goes through more training instances. The decoder can improve its approximate action values in various situations by looking at the potential risk involved in each decision and context vector that acts as the surrogate for the environment.

Moreover, in previous problem settings, the policy network learns from a relatively trivial objective function, which is a simple summation over a quantitative feature of the nodes. However, in the phone allocation problem, the RL algorithm has to minimize an arduous nonlinear cost function through interaction with the environment.

Computation complexity

To have a fair comparison between RL and QP approaches, we measured the run time of these algorithms on a SBM graph with 100 nodes. We trained the RL model on Compute Canada clusters with the following configuration: 16 intel CPU cores, 64G RAM, and one NVIDIA P100 GPU. We did a grid search on more than 100 set of hyper-parameters which took about 3 days to discover the optimal set of hyper-parameters (about 3 hours per each set). The model is then tested with the same configuration except for the GPU. Given a problem instance with 100 nodes, it takes less than a second for the RL model to construct a new solution at the test time. The reason that the RL model is such fast is that the code carries out fully vectorized computation which substantially speeds up the process as opposed to *for loops*. Besides, in the test phase, the model goes through only one episode and does not need to keep track of the chain of gradients for back-propagation. The QP algorithm is tested on the same configuration and it takes about 10 seconds for the QP solver to land on the optimal solution. These empirical results reveals that the RL-based approach is significantly faster than QP-based approach (up to two orders of magnitude). It is also noteworthy that a practical downside of the QP algorithm is that it hardly converges on the graphs with random topology.

4.3.2 Policy gradient

Fig 4.10 shows the training progress for policy gradient approach. The return, i.e., the reduction in potential risk, is computed at the end of each episode by getting the average over a batch (=128) of problem instances. Unfortunately, the experimental results show that this model fails to learn a promising policy. It is indeed slightly superior to the random solution, but worse than the QP-based solution. The same

reasons that keep the Q-learning algorithm from developing a powerful model result in this setback. Particularly, because of the inherent weakness of graph encoder, node embeddings are not diverse enough so that many of the phone clones are prone to go to few hosts. To make it worse, the trained model does not generalize to problem instances of larger size. Consequently, we have to keep numerous trained models for a wide range of phone clones in the cloud, which puts additional overhead on the system.

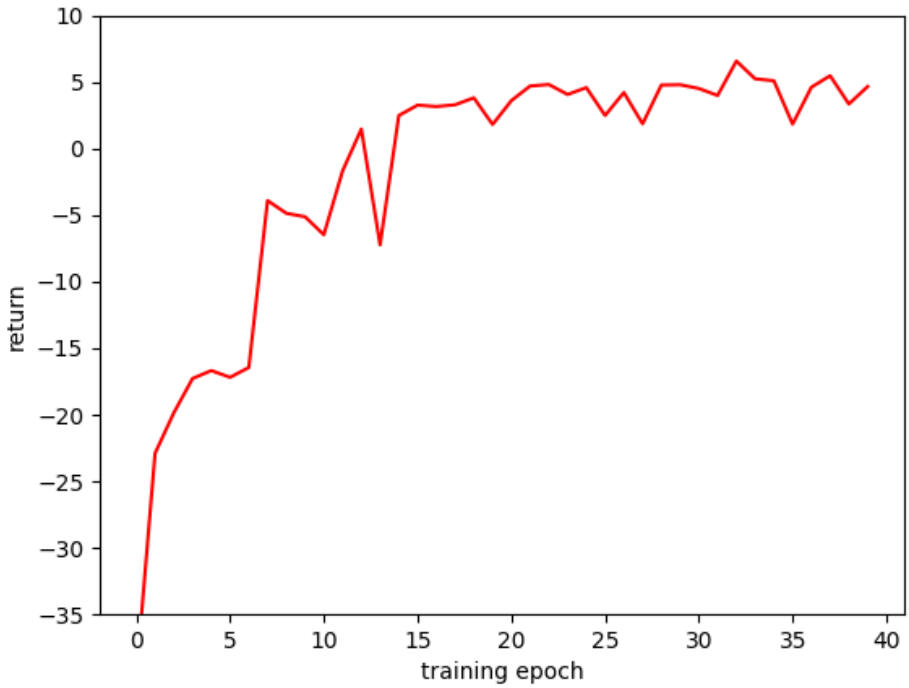


Figure 4.10: Training progress over time.

Overall, many combinatorial optimization problems that arise in computer science have unique nuance and circumstances that keep people from generalizing existing RL-based models for a known family of problems such as TSP to address other classes of hard problems. This insight serves as a cautious note that people should not be over optimistic about the power of RL for tackling complex combinatorial optimization problems.

Chapter 5

Conclusion and Future work

In this thesis, we study the challenge of applying existing success of deep reinforcement learning framework to solve hard combinatorial optimization on graph including quadratic assignment problem investigated in this thesis. We use the security-aware phone clone allocation in the cloud as motivation example to investigate whether or not deploying existing RL framework on quadratic assignment problem outperforms the traditional approximation algorithms.

To gain a better understanding on the difficulty and the structure of the problem, we provide an analytical solution to the problem that involves solving a typical QP problem with general-purpose solvers. This method shows remarkable improvement in the results compared with previous successful approximate algorithms on this matter. In the rest, before we extend RL's application to handling our problems and discuss its limits, we proposed an end-to-end unsupervised machine learning framework, which is capable of learning the heuristics of the problem in the context of graph structured data. Then, we train the model with a fitted reinforcement learning algorithm to make informed decisions based on the graph structured data that reflect the environment. Extensive experimental evaluation shows that the existing RL-based models are not effective enough to generalize to the quadratic assignment problem, where incrementally constructing an order of vertices is not an inherent requirement for obtaining the optimal solution.

In what follows, we highlight the main features of problems where RL is successful, and the lack of such features causes the RL algorithm to underperform in our problem setting.

- Incrementally constructed solution in previous works such as TSP allows the

decoder to mask the partial solution. Getting rid of partial solution reduces the search space and helps the decoder to find a better solution.

- Diversity of input vectors leads to readily distinguishable context vectors from encoder. Informative context vectors in the next phase conduct the decoder to achieve accurate state-action values for varying state of the environment.

- The objective function is relatively simple (e.g., the total length of a tour). The simple objective function facilitates the convergence of the training process. In contrast, the phone clone allocation problem involves an arduous nonlinear objective function.

5.1 Future Work

As evidenced by our investigation, due to the inherent complexity of the quadratic assignment problem, building an RL model that can directly produce a satisfactory solution is extremely challenging, if not impossible. Due to the irreversible nature of most RL models, the actor cannot revisit and revise its previous decisions. It is likely necessary due to the complexity of the quadratic assignment problem. Thus, a promising future line of work is to enable the actor to reconsider and change its earlier decision. However, this is a long and arduous process, and simply allowing the actor to revise its former decision does not automatically fix the issue. For the actor to exploit this freedom of going back and revising the former decisions, it is essential to provide it with augmented information from the environment. Augmented and rich input vectors also make the states during training distinctively different in the actor's view; resolving the major downside of the untraceable environment states. Some valuable information to incorporate into the input vectors are the potential risk that the phones currently incur, the potential risk of the phone's host, and steps since the phone clone was last reallocated.

To the best of our knowledge, existing RL models for solving combinatorial optimization problems do not offer an *effective* mechanism to revoke/revise previous decisions. The only work that has adopted a mechanism to address the irreversible nature of the existing methods and revise the previous decisions is [3] by Barrett et al. They worked on the Maximum Cut problem and applied this mechanism along with ensemble technique and an intermediate reward procedure to mitigate the problem of locally optimum state. This work, however, only brought marginal performance

improvement over [10]. After K round of message passing via the encoder, a read-out function predicts the action value of adding or removing each vertex from the solution subset. To apply this idea to our problem setting, we need a twofold policy: Q_1, we need a scoring mechanism to find the apt phone to reallocate, Q_2, we have to compute the risk of allocating the attained phone to different hosts. Besides, the decision should be feasible with respect to hosts' capacities. To achieve these goals, a more sophisticated architecture and rewarding procedures such as hierarchical design as mentioned in the following paragraph seems pivotal. Considering the marginal benefit of this method over [10] and the arduous overhead in restructuring this idea to be applicable in our problem, we consider it as future work.

One potential direction that might improve the performance of RL is to design hierarchical architecture in the decoder to break down the complex task into several simple tasks with separate reward functions. Particularly, the lower layer of the hierarchy can be tasked with determining the "tricky" phone clone regardless of whether it was already reallocated in the ongoing episode. The higher layer can reallocate the phone clones based on the latent data received from the previous layer. This architecture can continue by adding a layer in the middle of the hierarchy to push the final decision toward the feasible set of actions concerning the problem constraints by defining a suitable reward function as mobilizer in this phase. This potential architecture may resolve possible adverse competitions between the penalty term and the potential risk in defining the state-action value in the RL model. Besides, this middle layer would help to get away from tuning the hyperparameter that is supposed to set a balance between these two counterparts. To harness this utility, designing reasonable reward functions and strategy for training the model is another future path toward further improving the model. Furthermore, each training episode should continue until no remarkable improvement could be made rather than keeping to a certain number of iterations.

These lines of work deserve future investigation to see if discrete quadratic assignment problems can be finally tackled by a combination of deep graph network and RL algorithms. We leave this open challenge for future research.

Bibliography

[1] Edoardo M Airoldi, David M Blei, Stephen E Fienberg, and Eric P Xing. Mixed membership stochastic blockmodels. *Journal of machine learning research*, 9(Sep):1981–2014, 2008.

[2] Marco V Barbera, Sokol Kosta, Alessandro Mei, and Julinda Stefa. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *2013 Proceedings Ieee Infocom*, pages 1285–1293. IEEE, 2013.

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

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

[5] Alain Billionnet, Sourour Elloumi, and Marie-Christine Plateau. Improving the performance of standard solvers for quadratic 0-1 programs by a tight convex reformulation: The qcr method. *Discrete Applied Mathematics*, 157(6):1185–1197, 2009.

[6] Alain Billionnet and Éric Soutif. An exact method based on lagrangian decomposition for the 0–1 quadratic knapsack problem. *European Journal of operational research*, 157(3):565–575, 2004.

[7] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. *arXiv preprint arXiv:2006.01610*, 2020.

[8] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314, 2011.

[9] Yu Jin and Joseph F JaJa. Learning graph-level representations with recurrent neural networks. *arXiv preprint arXiv:1805.07683*, 2018.

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

[11] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[12] Eugene L Lawler. The quadratic assignment problem. *Management science*, 9(4):586–599, 1963.

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

[14] Fangming Liu, Peng Shu, Hai Jin, Linjie Ding, Jie Yu, Di Niu, and Bo Li. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless communications*, 20(3):14–22, 2013.

[15] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European journal of operational research*, 176(2):657–690, 2007.

[16] Hao Lu, Xingwen Zhang, and Shuang Yang. A learning-based iterative method for solving vehicle routing problems. In *International Conference on Learning Representations*, 2020.

[17] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*, 2019.

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

[19] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[20] Manfred W Padberg and Minendra P Rijal. *Location, scheduling, design and integer programming*, volume 3. Springer Science & Business Media, 2012.

[21] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 199–212, 2009.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[23] Seyed Yahya Vaezpour, Rui Zhang, Kui Wu, Jianping Wang, and Gholamali C Shoja. Swap: Security aware provisioning and migration of phone clones over mobile clouds. In *2014 IFIP Networking Conference*, pages 1–9. IEEE, 2014.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[25] Zhenyu Wu, Zhang Xu, and Haining Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 159–173, Bellevue, WA, 2012. USENIX.