GNOSIS: Proactive Image Placement Using Graph Neural Networks & Deep Reinforcement Learning

Theodoros Theodoropoulos[1], Antonios Makris[1], Evangelos Psomakelis[1],
Emanuele Carlini[2], Matteo Mordacchini[2], Patrizio Dazzi[3] and Konstantinos Tserpes[1]
[1] *Department of Informatics and Telematics, Harokopio University of Athens, Greece*
[2] *Institute of Information Science and Technologies, National Research Council (CNR), Pisa, Italy*
[3] *Department of Computer Science, University of Pisa, Italy*

***Abstract*—The transition from Cloud Computing to a Cloud-Edge continuum brings many new exciting possibilities for interactive and data-intensive Next Generation applications, but as many challenges. Approaches and solutions that successfully worked in the Cloud space now need to be rethought for the Edge's distributed, heterogeneous and dynamic ecosystem. The placement of application images needs to be proactively devised to reduce as much as possible the image transfer time and comply with the dynamic nature and strict requirements of the applications. To this end, this paper proposes an approach based on the combination of Graph Neural Networks and actor-critic Reinforcement Learning. The approach is analyzed empirically and compared with a state-of-the-art solution. The results show that the proposed approach exhibits a larger execution times but generally better results in terms of application image placement.**

***Index Terms*—Edge Computing, Cloud Computing, Component Placement, Proactive Image Placement, Graph Neural Networks**

I. Introduction

The emergence of data-intensive and latency-aware (i.e., NextGen) applications is one of the primary drives for the diffusion of Edge Computing [1]–[3]. The promise of Edge computing is to facilitate the adoption of utility computing for those applications left behind by the Cloud computing revolution by enabling the allocation of applications close to users and data sources, addressing concerns such as latency, battery life, bandwidth costs, security, and privacy [4].

A microservice architecture typically characterizes these applications, which are composed of independently deployable and loosely coupled services [5], [6]. This organization enables the on-demand and dynamic behavior to support the movement of data and users. Such property implies that a microservice must be switched on only when a certain number of users are accessing it from the area around a specific edge resource.

Therefore, Edge devices are requested to run many microservices over time. As a direct consequence, it emerges the need for resources at the Edge to access the sets of images of microservices (regardless of their nature, e.g., containers, Unikernels, etc.) to create running application instances in a *proactive* fashion. Providing access to the images presents several challenges: (i) limiting the transfer time to comply with the dynamic behavior of the application, (ii) limiting the bandwidth cost to download the image, and (iii) complying with the possibly limited storage requirements of the edge resources. These challenges can be further exacerbated when the number of edge devices is high.

As a consequence, the download of images from a centralized repository is not always an option, due to possible bandwidth costs and transfer time. In this paper, we advocate for a distributed approach in which images are replicated in a subset of edge resources. In particular, we consider an approach that separates Edge resource into distinct groups, and assign an image to each group. The goal is to balance the storage usage with the latency induced by image transfer. Ideally, such groups of Edges should be (auto-)determined (and sized), depending on the actual availability of resources and the requirements of the application instances to be run (e.g., size, latency requirements, etc.).

To this end, we model the image placement to a Minimum Vertex Cover (MVC) on the network that connects the Edge nodes. We then propose GNOSIS, a learning approach based on the combination of Graph Neural Networks and Deep Reinforcement Learning. This solution combines the representation power of a Graph Neural Network (GNN) approach with the ability of actor-critic Reinforcement Learning to provide strong solutions. We extensively analyzed GNOSIS with different network topologies and sizes, and we compare it with a greedy approach at the state of the art [7]. The Greedy algorithm is a few orders of magnitude faster with all network configurations. However, the results show that GNOSIS outperforms the greedy algorithm in terms of vertex cover quality for random and small-world networks, whereas it falls behind for preferential attachment networks.

II. Related Work

Optimally placing application images at the Edge is a relevant problem [8], [9] in the dynamic and heterogeneous environment of the Cloud-Edge Continuum. To face this issue, some solutions are based on optimization techniques, like Integer Programming [10], [11], Markov Decision Processes [12] or stochastic optimization [13].

Apart from the techniques used to formulate the problem, many proposals consider that the main goal of this optimization should be aimed to minimize the latency experienced by the users of the services that have to be placed in the system. In Rossi et al. [14], the authors exploit Reinforcement Learning techniques for developing an extension of Kubernetes able to

deploy and replicate delay-sensitive containerized services in a geographically distributed system. Zavodovski et al. [15] propose ICON, a solution where autonomous containers collaborate to find the best (in terms of latency) allocation of their services. Other solutions of this kind are presented in [16]–[18].

A relevant point is that many of such works are based on a reactive placement of the services, i.e. placement strategies are exploited after one or more services have been requested. However, stringent Quality of Service (QoS) constraints coupled with the functional complexity of these solutions risk to make them not adequate for the needs of the Cloud-Edge Continuum. For this reason, in this work, we propose a novel *proactive* approach. Only few other approaches adopt this strategy. In [19], the authors propose a proactive mechanism based on Reinforcement Learning to optimize microservice placement and migration in the context of Mobile Edge Computing (MEC). Their approach is designed to handle the dynamic nature of the environment by anticipating future needs and adjusting placement and migration decisions accordingly. Deng et al. [20] proposes a proactive application deployment system consisting of three modules. An incentive module uses a Spontaneous Edge deployment algorithm to allow edge servers to compete in a two-stage game to win deployment rights and get paid. The other two modules adjust service prices and deployment intentions to maximize profits. Gonçalves et al. [21] propose a solution for a system where resource-constrained mobile devices have high mobility. Their proposed model is based on mobility predictions that induce a proactive migration of virtual machines to provide low-latency access to resources at the edge.

All of the aforementioned solutions represent rather specific solutions for determined scenarios. A peculiar aspect that is not considered is the impact that the topology of the network could have on the final results. To overcome this issue, we propose a solution that is based on a learning approach that is able to adapt to the shape of the connections existing between entities at the edge. To best face this problem, the proactive placement of application images problem is formulated as a Minimum Vertex Cover problem. To the best of our knowledge, this work is the first one to propose such a model for proactive placement.

III. Problem Formulation

A. Minimum Vertex Cover and Proactive Image Placement

Vertex Cover (VC) is a category of NP-complete problems that describe the need to find paths in a vertex in order to "cross" all available edges in it [22]. The Minimum Vertex Cover (MVC) is a special category of VC problems that require us to find the optimal combination of edges that allow us to "cross" all of the edges in the vertex. In order to understand MVC a mathematical representation may provide more insight.

So, for the classic VC we can say that for an un-directed graph $G = (V, E)$, one solution of the VC problem would consist of a subset of vertices $S \subseteq V$ which ensures that $u \in S$ or $v \in S$, or both for every edge $(u, v) \in E$. This means that one vertex cover, which is one possible solution to the VC problem, is a set of vertices that includes all edges in E [23]. If values are added to the edges or vertices and the criteria of minimizing the total values in the subset S is added to the problem, then we are transitioning from a VC problem to an MVC problem.

The Proactive Image Placement problem can be modelled into an MVC problem by representing the network as the graph G. Each $v \in V$ is a node that either holds, requests, or is not interested in the application image we are trying to place. Each $e \in E$ represents a connection between two nodes that is assigned an available bandwidth. The MVC problem, in this form, is trying to identify the subset $S \subseteq V$ which is, in essence, a list of nodes that need to hold the image in order to minimize the data transfers in the network or/and the disk space used.

B. Set Cover Problem

By introducing a more clearly defined problem formulation and constraint we can create a set cover problem that describes the problem of proactive image placement in greater detail. The problem is the identification of the nodes on which one or more replicas of the application image must be placed, in order to achieve the following constraint for all nodes v, while minimizing the cost function for the whole network.

The constraint can be defined as $L_{image}/W_{sd} < T$, where L_{image} is the length of the image in bytes, the W_{sd} is the available bandwidth in the connection between source s and destination d and T is the time duration after which the image will be needed in the destination. The cost function can be defined as $F(T, L_(image), Rs)$ where T is the time threshold for the image transfer, $L_(image)$ is the length of the image in bytes and Rs is the cost of storage space in node s. A possible solution DP_i to the problem is a set of nodes that can hold the application image we are trying to place. Its total cost TC_i is calculated by the sum of the costs provided by the cost function for each node s that is included in DP_i.

The problem can be defined in a mathematical fashion using the function:

$$L_{image}/W_{sd} < T \mid \exists s \in DP_i, \forall d \in V \tag{1}$$

For the solution i which is defined as:

$$DP_i = [Node_0, Node_1, \ldots, Node_s] \text{ and} \tag{2}$$

$$TC_i = \sum_s F(L_{image}, C_s, R_s) \mid \forall s \in DP_i \tag{3}$$

C. Optimization Problem

The minimum vertex cover problem we defined can be converted into an optimization problem in order to fit into more optimization algorithms and methodologies. In order to convert it we need to define two things; an objective function and a set of constraints. Every optimization problem can be clearly defined using these two parts.

A useful tool in the definition of the necessary functions is the definition of a new set of binary values $A_v \forall v \in V$ where

$A_v \in \{0, 1\}$. These values are equal to 1 if the relevant node v holds a replica of the image or 0 if it doesn't. This enables us to define a constraint that forces the total transfer time for all destination nodes d to be greater than zero, ensuring that the destination nodes are getting the image as follows:

$$\sum_{n}(A_v * L_{image}/W_{vd}) > 0 \mid \forall v \in V, \forall d \in D \tag{4}$$

,where L_{image} is the length of the image in bytes, W_{vd} is the available bandwidth in the connection between source node v and destination node d and D is the set of destination nodes that request the image.

The objective function needs to take into consideration both the total transfer time of data inside the network and the amount of space being occupied by replicas of the image, as discussed in the minimum vertex cover problem formulation. Since it is impossible to define a constraint that forces the time threshold limitation for each data transfer we are integrating the transfer time to the objective function in order to achieve the least possible total transfer time, limiting as much as possible the violations to the time threshold. This means that the objective function would take the following form:

$$min(\sum_{n}(A_n * L_{image}) + \sum_{n}\frac{A_n * L_{image}}{W_{nd}}) \\ A_n \in \{0, 1\}, \forall n \in N, \forall d \in D \tag{5}$$

The problem that arises in this case is that the two factors of this objective function; the amount of space being occupied and the total transfer time, are using different scales. The space is calculated in gigabytes or even terabytes while the time is calculated in milliseconds or seconds. This causes an imbalance since a slight difference in the first scale would greatly affect the function while the same change at the second scale would have minimal effect. To counteract this effect we decided to remove the L_{image} value from the first scale, taking into consideration only the total number of nodes that hold a replica of the image and using this number as a weight on the total transfer time. The adjusted function takes the following form:

$$min(\sum_{n}A_n + \sum_{n}\frac{A_n * L_{image}}{W_{nd}}) \\ A_n \in \{0, 1\}, \forall n \in N, \forall d \in D \tag{6}$$

IV. The GNOSIS Approach

The GNOSIS approach to solving the MVC problem that was explored in the previous section is based on the use of Graph Neural Networks & Deep Reinforcement Learning in order to perform combinatorial optimization.

A. Graph Neural Networks

Graph Neural Networks (GNNs) are a type of neural network that operate on graph-structured data, allowing for the learning of node and graph-level representations. They can be used for a variety of tasks, including node classification, graph classification, link prediction and accurate resource usage prediction [24]. Here we will focus on explaining GNNs for graph-level representations.

Let us consider an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. The graph can be represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$, where n is the number of nodes. The elements of A denote the presence or absence of edges between nodes, such that $A_{ij} = 1$ if there is an edge between nodes i and j, and $A_{ij} = 0$ otherwise. In addition, we assume that each node i has an associated feature vector $x_i \in \mathbb{R}^d$, where d is the number of features.

The goal of GNNs is to learn a function that maps the graph G and its associated node features $x_{ii \in V}$ to a vector representation of the entire graph. This can be achieved by defining a sequence of neural network layers, each of which aggregates information from neighboring nodes and updates the node representations.

At each layer k, the node representations are updated according to the following formula:

$$h_i^{(k)} = \sigma\left(\sum_{j \in \mathcal{N}(i)}\frac{1}{c_{ij}}W^{(k)}h_j^{(k-1)} + b^{(k)}\right) \tag{7}$$

,where $h_i^{(k)}$ is the updated representation of node i at layer k, $\mathcal{N}(i)$ is the set of neighboring nodes of node i, c_{ij} is a normalization constant, $W^{(k)}$ and $b^{(k)}$ are the trainable weight matrix and bias term at layer k, and σ is an activation function such as the rectified linear unit (ReLU).

After K layers, the final representation of the graph can be obtained by applying a readout function to the node representations, such as summation or max-pooling:

$$h_G = \rho\left(\sum_{i \in V}h_i^{(K)}\right) \tag{8}$$

,where h_G is the final representation of the graph and ρ is the readout function.

The GNN is trained by minimizing a loss function that depends on the final graph representation and the true label or target associated with the graph, using backpropagation and stochastic gradient descent. GNNs allow for the learning of graph-level representations by recursively aggregating information from neighboring nodes through multiple layers of neural network operations. This approach can be used for a variety of tasks, including combinatorial optimization problems.

B. The Actor-Critic Algorithm

Actor-critic is a class of reinforcement learning algorithm that combines the advantages of both policy-based and value-based methods. It consists of two neural networks: an actor network, which learns the policy, and a critic network, which estimates the value of the policy.

When leveraging temporal difference (TD) learning approaches, the critic network learns to estimate the expected cumulative reward by iteratively updating its value function based on the observed rewards. Specifically, at each time step

t, the critic updates its value estimate $V_\phi(s_t)$ using the TD error:

$$\delta_t = R_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{9}$$

,where R_t is the observed reward at time step t, γ is the discount factor, and s_t and s_{t+1} are the current and next states, respectively. The TD error represents the difference between the predicted reward and the actual reward, and is used to update the critic's value estimate:

$$V_\phi(s_t) \leftarrow V_\phi(s_t) + \alpha\delta_t \tag{10}$$

,where α is the corresponding learning rate.

In the actor-critic algorithm with TD learning, the actor network learns the policy by maximizing the expected cumulative reward, which is estimated using the critic's value function. The policy update is based on the advantage function A_t, which represents the advantage of taking action a_t in state s_t compared to following the current policy. The advantage function is defined as:

$$A_t = \delta_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t) \tag{11}$$

Finally, the policy update is then given by:

$$\theta \leftarrow \theta + \beta\nabla_\theta \log \pi_\theta(a_t|s_t)A_t \tag{12}$$

,where θ are the parameters of the actor network, β is the learning rate, $\log \pi_\theta(a_t|s_t)$ is the log-probability of taking action a_t in state s_t according to the policy, and ∇_θ is the gradient operator.

C. Graph Neural Networks & Deep Reinforcement Learning for Combinatorial Optimization

Actor-critic algorithms have shown promising results in solving combinatorial optimization problems. By combining actor-critic with GNNs, we can leverage the power of GNNs to represent graph structures and use actor-critic RL to provide solutions.

One may represent a combinatorial optimization problem as a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Each vertex represents a decision variable, and each edge represents a constraint between two decision variables. Let $x \in 0, 1^n$ be a binary decision vector, where $n = |V|$, such that $x_i = 1$ if vertex i is selected and $x_i = 0$ otherwise. The goal is to find an optimal decision vector x^* that maximizes an objective function $f(x)$, subject to the constraints encoded in the graph.

To apply actor-critic algorithms with GNNs, one has to define a policy function $\pi_\theta(a|s)$ that takes as input a state representation s and outputs a probability distribution over the action space a. We parameterize the policy function with θ. The state representation s is obtained by feeding the graph structure and the current decision vector x through a GNN. The actor and critic are updated using the policy gradient and the TD learning algorithms, respectively. The policy gradient updates the policy parameters θ using the gradient of the expected reward with respect to θ. The TD learning updates the critic parameters ϕ by minimizing the mean squared error between the estimated and actual values.

The actor-critic with GNNs algorithm for combinatorial optimization is described in Algorithm 1.

Algorithm 1 Actor-critic with GNNs algorithm for combinatorial optimization

Begin
1.Initialize the policy and critic parameters θ and ϕ.
2.For each episode do:
3. Initialize the decision vector x randomly.
4. Feed the graph structure and x through the GNN to obtain the state representation s.
5. Sample an action a from the policy distribution $\pi_\theta(a|s)$.
6. Obtain the next state s' and reward r by applying the action a to x.
7. Update the critic parameters ϕ using TD learning.
8. Compute the policy gradient and update the policy parameters θ.
9. Repeat steps 4-8 until a stopping criterion is met.
10. Obtain the next state s' and reward r by applying the action a to x.
11.End For.
12.Return the decision vector x^* that maximizes $f(x)$.
End

V. Experimental Evaluation

In this section, the experimental evaluation is presented in relation to various performance indicators. The examined algorithms are subjected to a rigorous investigation through extensive experiments in the set cover problem, which in turn sheds light on their properties and impacts on proactive image placement. In the following, we describe the simulation methodology (subsection V-A) and analyze the experimental results (V-D).

A. Simulation Methodology

To simulate different network topologies, the image placement and the image transfers between the nodes of the networks, Python (3.6.9) scripts were developed and utilized. The NetworkX [25] Python package is exploited to create a wide variety of network topologies with different parameters. In addition, NetworkX creates objects that facilitate the storage and manipulation of node and edge labels and characteristics, such as total capacity, image replication, and network connection usage.

Each network connection, modelled as an Edge, is characterized by two attributes: its available bandwidth and its usage. From these attributes, several secondary values can be derived, such as the percentage of bandwidth usage and the transfer time. Each node is labeled with a unique serial number/ID and has an attribute with a boolean value that indicates whether it holds a replica of the image (1) or not (0). Additionally, a number of parameters are set regardless

of the networks that affected each set of experiments, such as image size and maximum available bandwidth for Ethernet and WiFi connections.

In the simulations carried out in this study, bandwidth allocation was accomplished using the Max-min fairness algorithm. Max-min fairness [26] tries to maximize the bandwidth allocated to the flows with minimum share, thus guaranteeing that no flow can increase its rate at the cost of a flow with a lower rate. Initially, all flow rates are set to zero and then each rate is gradually increased equally until the link's capacity is reached. The simulated network is represented as an undirected graph, and Max-min fairness can provide a reliable estimate of the network's actual behavior. As a total bandwidth capacity, the maximum bandwidth of the node's virtual network adapter is considered, which is randomly set to either Ethernet (100MBps) or WiFi (25MBps). In a realistic Edge network most communication, are made through WiFi since a plethora of smart devices and IoT equipment are utilized, therefore the Wifi network adapters were arbitrary set to a 75% ratio while Ethernet adapters were assigned to the rest 25% of nodes.

B. The Greedy algorithm

Leveraging various network topologies and associated attributes, along with the simulation's general parameters, a multitude of experiments were conducted, utilizing two different algorithms in order to score and rank their performance under various conditions. Specifically, we compare GNOSIS (described in Section IV) with a Greedy algorithm.

The Greedy algorithm is a powerful method that can effectively solve a variety of optimization problems. It works by always selecting the option that appears best at the moment, attempting to maximize the return based on local conditions, assuming that this will lead to a globally optimal solution [7]. In the case of the minimum vertex cover problem, the Greedy algorithm starts with an empty set of vertices S, then chooses an arbitrary edge e in the graph, adds both endpoints of e to S, and finally removes all edges from the graph that are covered by the vertices in S. If the graph is empty, S is returned as the minimum vertex cover, otherwise, the process starts again by choosing another arbitrary edge e. At each step, the algorithm chooses the edge with the fewest number of uncovered vertices and adds both endpoints to the vertex cover set. Thus, it guarantees that the set of vertices will cover as many edges as possible.

The Greedy algorithm does not always produce an optimal solution as in some cases, the algorithm may get stuck in a local optimum and fail to find the global optimum. However, for the minimum vertex cover problem, the Greedy algorithm is known to produce a solution that is at most twice the size of the optimal solution. This bound is known as the 2-approximation guarantee.

C. Network topologies

The performance of the examined algorithms is evaluated over three different network graph topologies. Each topology is simulated at various sizes, ranging from 64 to 512 vertices, including $V = [64, 128, 256, 512]$. However, the number of edges in each graph varies significantly due to the unique characteristics, input parameters, and connectivity properties of each topology, resulting from the different number of vertices. The network topologies utilized in this research work are:

Erdö–Rényi random network. The concept of random graphs was introduced by Paul Erdös and Alfréd Rényi, who found that probabilistic methods were effective in solving graph theory problems [27], [28]. Due to the limited computing power available when these network types were first introduced in the 1950s, much of the modeling was focused on relatively small "ordered" or "regular" networks, which are infrequent in real-world scenarios [29]. An alternative and equivalent definition of a random graph is the binomial model, in which the $G(N, p)$ model starts with N nodes and connects each distinct node pair with probability p. Two different variants of the Erdö–Rényi model, which are based on the probability p indicating an edge between any two nodes, have been considered in this work: $p = 0.2$ and $p = 0.5$. As expected, the structure of the graph generated by each probability, varies significantly; as p increases, the number of edges also increases.

The uncorrelated Erdö–Rényi random graph model assumes that every pair of vertices in a graph has equal and independent probabilities, thus treating the network as a collection of equivalent units. However, real networks are fundamentally correlated systems, and their topology often diverges from that of the uncorrelated random graph model. The attention has shifted on developing more sophisticated graph models, with particular emphasis on "real-world" networks like the Internet and the World-Wide Web. To understand the general properties of such networks, two classes of models have emerged: "small-world" and "scale-free". Small-world networks aim to capture the clustering observed in real graphs and are inhomogeneous in that the pattern of connectivity between nodes is relatively localized. Scale-free networks present inhomogeneity in the "degree" of nodes (i.e., the number of connections a node has to other nodes) and reproduce the power law degree distribution present in many real networks.

Watts–Strogatz small-world network. Watts and Strogatz [30] proposed what has become known as the archetypical small-world network. The algorithm behind the model begins by constructing an undirected ring lattice network that consists of a ring of nodes with edges evenly distributed between its k_L nearest left and right neighbors. The value of k_L represents the degree of each node in the initial lattice. Then a process of random rewiring is applied where each edge has a probability p of being re-wired. The algorithm only rewires one end of each edge and traverses edges in a way that ensures that each node loses at most half of its edges. It is important to note that edges are only replaced, not added or removed, thus the total number of edges and the mean degree is unchanged. By varying p, it is easily proved that only a small number of rewires is required to produce a low average

path length while maintaining a high clustering coefficient. In fact, for $p = 0$ in the small-world model, a regular graph is preserved while for $p = 1$ a random graph is generated, which differs from the uncorrelated random graph only slightly. For intermediate values of p, Watts-Strogatz produces a small-world network, which captures the high clustering properties of regular graphs and the small characteristic path length of random graph models. Hence, we only focused on a single rewiring probability of $p = 0.5$ for two distinct degree values: $k_L = 2$ and $k_L = 4$. As the degree of a node represents the number of edges connecting it to other nodes in the graph, the resulting graph structure differs significantly for each degree value; as k_L increases, the number of edges also increases.

Barabási-Albert scale-free network. Barabási and Albert [31] proposed the scale-free network which is able to reproduce networks with "hubs", where a few nodes have significantly more connections to/from other nodes than the average, a property known as scale-free. As a result, the degree distribution is highly inhomogeneous. Since numerous real-world networks exhibit degree distributions similar to the Barabási-Albert model, it continues to be one of the most renowned and frequently employed network generation methods. The algorithm that produces a Barabási-Albert scale-free network of size N, starts with a small number of nodes m_o and then $N - m_o$ nodes are introduced sequentially into the network, where each node connects to/from $m \leq m_o$ existing nodes (it is typical to choose $m_o = m$). It is not possible to select $m > m_o$ as then the first new node introduced cannot be assigned m edges. Therefore, the initial network size m_o dictates the maximum mean degree of the network. Two different variants of the Barabási and Albert model have been considered in this work: $m = 1, m = 3$. As expected the structure of the graph generated by each value m, varies significantly; as m increases, the number of edges also increases.

D. Simulation Results

This section presents the simulation results for the different network topologies obtained by the two examined algorithms. Three performance metrics are used to assess the effectiveness of each algorithm in addressing the component image placement problem. The metrics that are considered are the following: i) execution time (ExT) ii) cost function (CF), and iii) size of the vertex cover (VCS). Execution time refers to the total amount of time each algorithm requires to generate a solution. One of the most important metrics of evaluation is the cost function, which calculates the cost based on the number of image replicas placed on the network as well as the transfer delays, in order to share the image between all network nodes. The cost function is the same as the objective function (Function 6), as described in III-C. Finally, the last metric employed is the size of the vertex cover, hence the size of vertices in it.

As discussed in subsection V-A, different variants are considered for each network topology, based on the input parameters of the corresponding model. However, due to the substantial number of experiments and space limitations, it is not feasible to visually present all the variations of the different network topologies. Consequently, only one representative variation is considered for each model: $p = 0.2$ for Erdö–Rényi, $k_L = 2$ for Watts–Strogatz and $m = 1$ for Barabási-Albert. Nevertheless, the detailed results for the remaining variant of each model are presented in Table I.

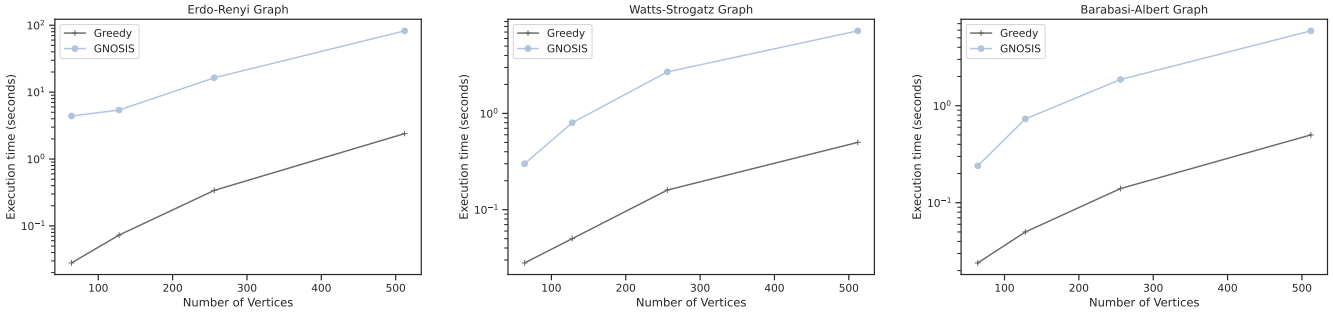
Execution time analysis. Figure 1 evaluates the execution time of each algorithm for the different network topologies. As the results indicate, the execution time of both examined algorithms is significantly higher for Erdö–Rényi graphs (Figure 1a) compared to other network topologies, especially as the number of vertices increases. As an illustration, when considering 512 vertices, the GNOSIS algorithm demonstrates an execution time of 82 seconds for Erdö–Rényi graphs and 5.9 seconds (almost 14 times slower) for Barabási-Albert graphs, whereas for the Greedy algorithm, the execution times are 2.4 seconds and 0.5 seconds (almost 5 times slower), respectively. Both algorithms exhibit the lowest execution times for Barabási-Albert graphs. As the results suggest, the GNOSIS algorithm demonstrates higher execution times compared to the Greedy algorithm for all network topologies. Both algorithms display a similar trend across the various network topologies, wherein the execution time increases linearly with an increase in the number of vertices. In general, as the number of vertices increases and more nodes are added to the vertex cover set, the execution time also increases as the generated graph becomes larger.

Cost function analysis. Figure 2 evaluates the cost function of each algorithm for the different network topologies. The results indicate that for Erdö–Rényi graphs, the Greedy algorithm results in the highest cost function values, whereas for Barabási-Albert graphs, the GNOSIS algorithm yields the highest cost function values. For both algorithms, as the number of vertices increases, Watts-Strogatz graphs demonstrate comparatively lower cost function values. To illustrate, when analyzing the Watts-Strogatz graph with 512 vertices, the cost function obtained by the Greedy algorithm is 27571, whereas, for Barabási-Albert and Erdö–Rényi, the same algorithm generates cost function values of 36592 and 110230, respectively. In addition, as the number of vertices increases, there is no significant difference in the cost function values between the two examined algorithms for Watts-Strogatz graphs. It is worth noting that in both Erdö–Rényi and Watts-Strogatz graphs, the GNOSIS algorithm demonstrates lower cost function values in comparison to the Greedy algorithm. The reduction in the cost function obtained using the GNOSIS algorithm is particularly remarkable in the case of Erdö–Rényi graph, as the decrease is nearly half with an increase in the number of vertices.

Vertex Cover Set size analysis. Figure 3 evaluates the vertex cover set produced by each algorithm for the different network topologies. As expected, the vertex cover set's size increases in a linear fashion with an increase in the number of vertices. As the results suggest, compared to other network topologies, GNOSIS algorithm generates a significantly smaller vertex cover set for Barabási-Albert graphs. The Greedy algorithm

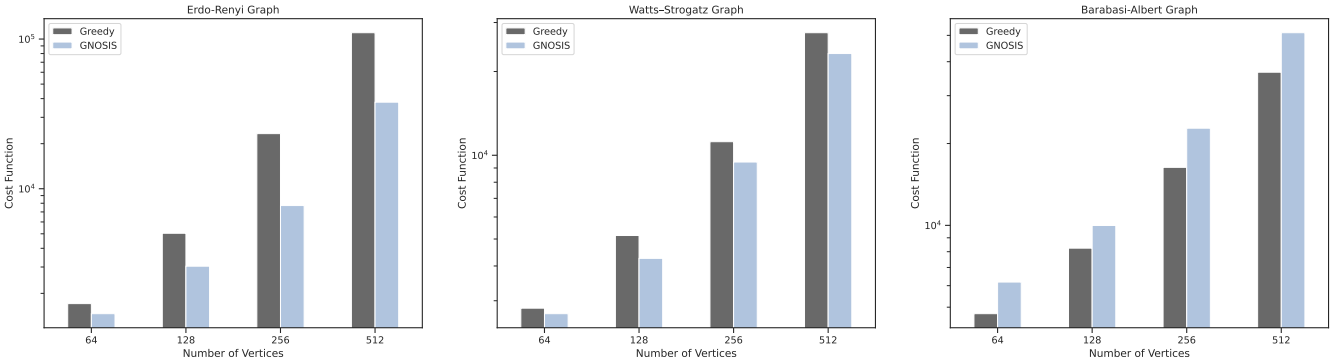
TABLE I: Detailed results for the remaining variant of each model

	#of Nodes	Metrics	m=3			p=0.5			k=4	
			Greedy	**GNOSIS**		**Greedy**	**GNOSIS**		**Greedy**	**GNOSIS**
	64	ExT	0.0263	0.88		0.0363	1.77		0.0238	0.5
		CF	2519.53	2422		1027.26	861.89		1442.30	1552.34
		VCS	34	35		58	59		47	47
Barabási-Albert	128	ExT	0.0431	4.9	Erdö–Rényi	0.1522	6.09	Watts-Strogatz	0.0480	1.7
		CF	5974.57	5421.32		5343.53	2084.31		3495.86	2756.91
		VCS	66	75		120	117		94	90
	256	ExT	0.1271	6.69		1.018	27.43		0.1259	3.65
		CF	15948.10	9859.43		23919.62	17568.33		8555.45	7249.61
		VCS	127	177		247	242		188	181
	512	ExT	0.4602	18.43		7.719	126.43		0.4891	11.24
		CF	61257.26	40930.19		106116.31	63837.23		35829.82	26619.32
		VCS	264	350		502	496		368	353



(a) Erdö–Rényi (b) Watts–Strogatz (c) Barabási-Albert

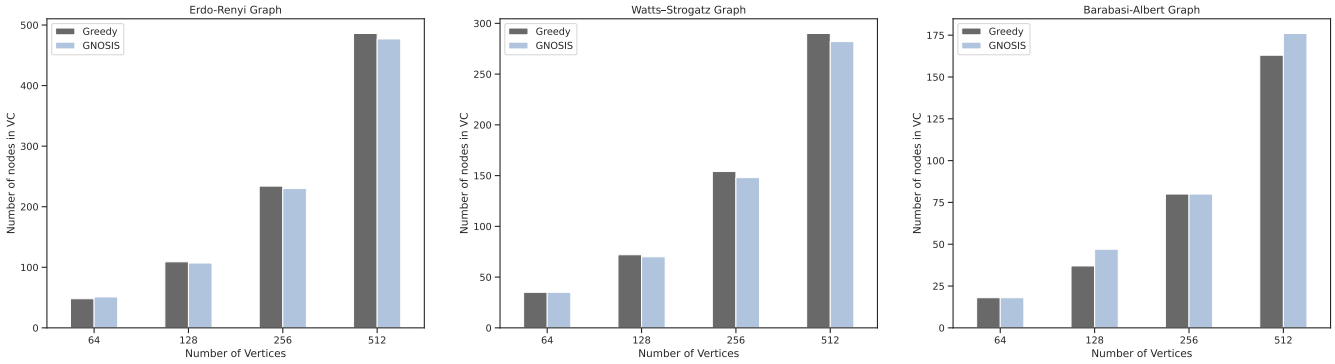
Fig. 1: Execution time of each algorithm for the different network topologies



(a) Erdö–Rényi (b) Watts–Strogatz (c) Barabási-Albert

Fig. 2: Cost function of each algorithm for the different network topologies (the lower the better)

exhibits the same behavior as well. As an illustration, when considering Barabási-Albert graphs with 512 vertices, the GNOSIS algorithm generates a vertex cover set size of 176 while for Erdö–Rényi and Watts-Strogatz graphs, the sizes are 477 and 288, respectively. Both algorithms produce the largest vertex cover sets for Erdö–Rényi graphs, followed by Watts-Strogatz. Although the Greedy algorithm yields a smaller vertex cover set for Barabási-Albert, the GNOSIS algorithm



(a) Erdö–Rényi (b) Watts–Strogatz (c) Barabási-Albert

Fig. 3: Vertex Cover set of each algorithm for the different network topologies

produces superior results for the other two network topologies.

Discussion. A concise summary of the experimental simulation results and key insights is provided to summarize the performance evaluation of each algorithm. As the results suggested, the GNOSIS algorithm demonstrates higher execution times compared to the Greedy algorithm for all network topologies. The reason is that GNOSIS involves training a neural network on the graph data and making predictions for each vertex in the graph. As the graph structure becomes more complex, GNOSIS captures more intricate relationships between the vertices which can lead to more accurate solutions but it also requires more computation. The high execution times presented by the GNOSIS algorithm result in higher cost function values for Barabási-Albert graphs compared to the Greedy approach. However, for the remaining network topologies, the cost function values are lower when using the GNOSIS algorithm. In terms of the size of the vertex cover set, the GNOSIS algorithm generates larger results than the Greedy algorithm for Barabási-Albert graphs. However, similar to the cost function, the GNOSIS algorithm generates smaller vertex cover set sizes for the other network topologies.

VI. Conclusion

The placement of container and VMs images is particularly relevant in the context of Edge Computing. The dynamic nature of Edge applications calls for short cold startup times, and the ability to download application images as fast as possible is a core requirement. In this paper, we modeled the problem as a Minimum Vertex Cover and propose GNOSIS, an approach that combines actor-critic RL with graph neural networks. We evaluated GNOSIS with different graph topologies and sizes, and we compared it with a Greedy state-of-the-art approach. From the analysis of the results, it emerges that, compared with the Greedy algorithm, GNOSIS has higher execution times but generally better MCV scores.

While a Greedy approach can provide a simple and efficient solution, it often falls short in terms of optimality as it considers locally optimal choices at each step without considering the global structure of the graph. Consequently, it lack the ability to adapt to different scenarios with complex graphs, learn from the data and may yield suboptimal solutions. In contrast, GNNs and actor-critic RL methods leverage the graph structure, generalize to unseen graphs and learn effective strategies to make globally informed decisions. This allows them to achieve better performance and potentially find optimal or near-optimal solutions, making them more effective for solving the minimum vertex cover problem.

While the results look promising, we plan to perform further evaluation of GNOSIS under different networks and versus other approaches. In particular, we intend to extend the types of topologies considered (e.g. balanced tree networks) and the comparison approaches for solving MVC, such as genetic meta-heuristics and Integer Linear Programming solutions.

Acknowledgment

The research leading to these results received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016509 (project CHARITY) and No 871793 (project ACCORDION). The paper reflects only the authors' views. The Commission is not responsible for any use that may be made of the information it contains.

References

[1] A. Makris, A. Boudi, M. Coppola, L. Cordeiro, M. Corsini, P. Dazzi, F. D. Andilla, Y. G. Rozas, M. Kamarianakis, M. Pateraki *et al.*, "Cloud for holography and augmented reality," in *2021 IEEE 10th International Conference on Cloud Networking (CloudNet)*. IEEE, 2021, pp. 118–126.

[2] I. Korontanis, K. Tserpes, M. Pateraki, L. Blasi, J. Violos, F. Diego, E. Marin, N. Kourtellis, M. Coppola, E. Carlini *et al.*, "Inter-operability and orchestration in heterogeneous cloud/edge resources: The accordion vision," in *Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge*, 2020, pp. 9–14.

[3] T. Theodoropoulos, A. Makris, A. Boudi, T. Taleb, U. Herzog, L. Rosa, L. Cordeiro, K. Tserpes, E. Spatafora, A. Romussi *et al.*, "Cloud-based xr services: A survey on relevant challenges and enabling technologies," *Journal of Networking and Network Applications*, vol. 2, no. 1, pp. 1–22, 2022.

[4] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[5] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for iot using docker and edge computing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 118–123, 2018.

[6] A. Makris, K. Tserpes, and T. Varvarigou, "Transition from monolithic to microservice-based applications. challenges from the developer perspective," *Open Research Europe*, vol. 2, p. 24, 2022.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[8] J. Darrous, T. Lambert, and S. Ibrahim, "On the importance of container image placement for service provisioning in the edge," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–9.

[9] K. Kaur, F. Guillemin, and F. Sailhan, "Container placement and migration strategies for cloud, fog, and edge data centers: A survey," *International Journal of Network Management*, vol. 32, no. 6, p. e2212, 2022.

[10] H.-J. Hong, P.-H. Tsai, and C.-H. Hsu, "Dynamic module deployment in a fog computing platform," in *2016 18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016, pp. 1–6.

[11] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand, "ifogstor: An iot data placement strategy for fog infrastructure," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017, pp. 97–104.

[12] M. L. Puterman, "Markov decision processes: Discrete stochastic dynamic programming," in *Wiley Series in Probability and Statistics*, 1994.

[13] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," 2010.

[14] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with kubernetes," *Computer Communications*, vol. 159, pp. 161–174, 2020.

[15] A. Zavodovski, N. Mohan, S. Bayhan, W. Wong, and J. Kangasharju, "Icon: Intelligent container overlays," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018, pp. 15–21.

[16] K. Velasquez, D. P. Abreu, M. Curado, and E. Monteiro, "Service placement for latency reduction in the internet of things," *Annals of Telecommunications*, vol. 72, pp. 105–115, 2017.

[17] M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 1222–1228.

[18] G. Lee, W. Saad, and M. Bennis, "An online secretary framework for fog network formation with minimal latency," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[19] K. Ray, A. Banerjee, and N. C. Narendra, "Proactive microservice placement and migration for mobile edge computing," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 28–41.

[20] S. Deng, Y. Chen, G. Chen, S. Ji, J. Yin, and A. Zomaya, "Incentive-driven proactive application deployment and pricing on distributed edges," *IEEE Transactions on Mobile Computing*, 2021.

[21] D. Gonçalves, K. Velasquez, M. Curado, L. Bittencourt, and E. Madeira, "Proactive virtual machine migration in fog environments," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018, pp. 00 742–00 745.

[22] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[23] T. F. Gonzalez, *Handbook of approximation algorithms and metaheuristics*. CRC Press, 2007.

[24] T. Theodoropoulos, A. Makris, I. Kontopoulos, J. Violos, P. Tarkowski, Z. Ledwoń, P. Dazzi, and K. Tserpes, "Graph neural networks for representing multivariate resource usage: A multiplayer mobile gaming case-study," *International Journal of Information Management Data Insights*, vol. 3, no. 1, p. 100158, 2023.

[25] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[26] D. Bertsekas and R. Gallager, *Data networks*. Athena Scientific, 2021.

[27] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959.

[28] P. Erdös and A. Rényi, "On random graphs publ," *Math. debrecen*, vol. 6, pp. 290–297, 1959.

[29] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[30] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[31] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.