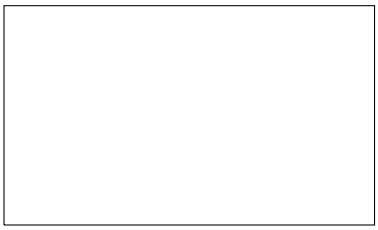
# Graphical Abstract

**Pro-active Component Image Placement in Edge Computing Environments**

Antonios Makris, Evangelos Psomakelis, Emanuele Carlini, Matteo Mordacchini, Theodoros Theodoropoulos, Patrizio Dazzi, Konstantinos Tserpes



# Highlights

**Pro-active Component Image Placement in Edge Computing Environments**

Antonios Makris, Evangelos Psomakelis, Emanuele Carlini, Matteo Mordacchini, Theodoros Theodoropoulos, Patrizio Dazzi, Konstantinos Tserpes

- Modeling the problem of proactive application images in Edge as a Minimum Vertex Cover problem
- Greedy implementation seems to offer the best tradeoff with respect to the number of allocated images and execution time

# Pro-active Component Image Placement in Edge Computing Environments

Antonios Makris[a], Evangelos Psomakelis[a], Emanuele Carlini[b], Matteo Mordacchini[c], Theodoros Theodoropoulos[a], Patrizio Dazzi[d], Konstantinos Tserpes[a]

[a]*Department of Informatics and Telematics, Harokopio University of Athens, Athens, Greece*
[b]*Institute of Information Science and Technologies, National Research Council (CNR), Pisa, Italy*
[c]*Institute of Informatics and Telematics, National Research Council (CNR), Pisa, Italy*
[d]*Department of Computer Science, University of Pisa, Pisa, Italy*

**Abstract**

Edge computing has attracted a lot of attention both from industry and academia in recent years and is considered as a key enabler for addressing the increasingly strict requirements of Next Generation applications. Contrary to Cloud computing, in Edge computing the computation are placed closer to the end-users into the so-called Edge, to facilitate low-latency and high-bandwidth applications and services that would not be feasible using cloud and far remote processing alone. However, the distributed, dynamic and heterogeneous environment in the Edge computing along with the diverse applications' requirements make service placement in such infrastructure a challenging issue. One important aspect of Edge computing is the management of the placement of the applications in the network system so as to minimize each application's runtime, given the resources of system's devices and the capabilities of the system's network. To this end, we propose an empirical experimental analysis, by comparing the results of different placements strategies and various edge communication networks. In particular, we model the problem of proactive placement of application images as a Minimum Vertex Cover problem. Our results demonstrate that the Greedy implementation seems to offer the best tradeoff in terms of performance, cost function and execution time.

*Keywords:* Edge Computing, Cloud Computing, Application Placement, Component Placement Algorithm, Proactive Image Placement, Graphs
*PACS:* 0000, 1111
*2000 MSC:* 0000, 1111

## 1. Introduction

The diffusion of Next Generation applications is leading to broad adoption of the Edge computing paradigm [1]. Running software at the edge of the network enables the adoption of utility computing for those applications left behind by the Cloud computing revolution. There are several peculiarities that characterize Next Generation applications [2]. One relevant aspect is that, differently from other applications that are currently being run on Edge computing platforms, the resource usage pattern adopted by Next Generation applications is primarily characterised by an "on-demand" behaviour. This means that a certain Edge is used by an application (runs an instance of such application, or some of its components) only when a certain amount of users are accessing the very same application from the area surrounded by such Edge.

Due to this unpredictable behaviour, Edge devices could be requested to run a large variety of applications over time. As a direct consequence, it emerges the need for resources at the Edge to access the sets of images of applications (regardless of their nature, e.g. traditional VMs, containers, Unikernels, etc.) to create running application instances when needed. From this regard, different approaches can be identified for the provisioning of such images to the Edge. The salient points that have to be taken into account when designing a solution to this problem can be summarized as:

- Average Transfer Time: how long does it take on average for the downloaders to get the image
- Total Transfer Cost: how costly is the bandwidth for the transferring
- Total Storage Cost: how costly is the storage of the images
- Update Cost: how costly is it to update the image of a service in case of changes

To analyze the problem, we can easily identify strategies at the opposite ends of the spectrum of the solutions to this problem. An *only-on-cloud*, fully centralised approach, in which the "authoritative copy" of each image is located on a centralized and globally reachable entity (such as a Cloud storage). The Edge devices always refer to such an entity to retrieve the corresponding data, and then store it locally in a purely ephemeral way. This is the simplest strategy to implement as there is only one copy per image to store and manage, but it suffers from high latency [57]. Indeed, the latency for accessing an average Cloud is reported to be from 2 to more than 3 times slower than accessing Edge servers [58]. In an *all-on-edge*, fully distributed

approach, each Edge device is provided in advance with a copy of all the application images. In this strategy, the transferring time is none (all the images are already on the Edge), but it requires expensive mechanisms to update and synchronize all the images and a significant amount of storage space on edge devices, since each of them should store all the applications' images. Finally, a *cloud-edge* solution in which some images are placed in a subset of edge nodes and others in the cloud. This strategy mixes the previous two. A tradeoff between transfer time and complexity in management is reached according to the nature of the edge resources.

This paper focuses on a *something-on-edge* solution. In particular, we consider the possibility of gathering the devices at the Edge into several distinct groups. It is then possible to consider maintaining a copy of each image for each Edge 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.). However, such interaction would require Edges to be mutually aware of their existence, to interact and collectively collaborate to optimize the system.

In the Cloud/Edge scenario, models for similar collaborations among heterogeneous entities have emerged in the Cloud Federation area. Indeed, Cloud federations have been demonstrated to be effective solutions for setting up a business offer characterised by vast pools of resources obtained by federating resources belonging to different providers, while enforcing global policies on resource usage, application management, and data deployment and, at the same time, are effective in revenue sharing [3]. Resource providers that participate in a federation, accept to delegate part of their sovereignty under the opportunity of enlarging their businesses and providing competitive offers to big customers, otherwise outside their reach. Federation can be organised in different ways, affecting the participating providers differently. Lighter forms of federation can be limited to common protocols for identity management and application representation. More advanced types of federation may require providers to adhere to management policies and to share data (e.g., monitoring information).

With respect to federations of Clouds, we target an even more distributed scenario in this paper. Specifically, we model a federation of Edge resources [4], where a centralized entity proactively decides when and where to place applications. Still, the images of those applications are distributed among the Edge resources. In other words, each Edge resource either owns the application image or has to download it from another Edge resource when assigned with the execution of the corresponding application. Figure 1 illustrates the aforementioned concept. However, on-demand download can be costly, as it requires to: (i) discovering who has the image of the application and (ii) transferring the application image locally. Essentially, the transferring time of an image depends on where these images are located and other parameters, such as the download strategy, the characteristics of the network, its topology, etc. Therefore, the main research question is: *Given an Edge federation, how*

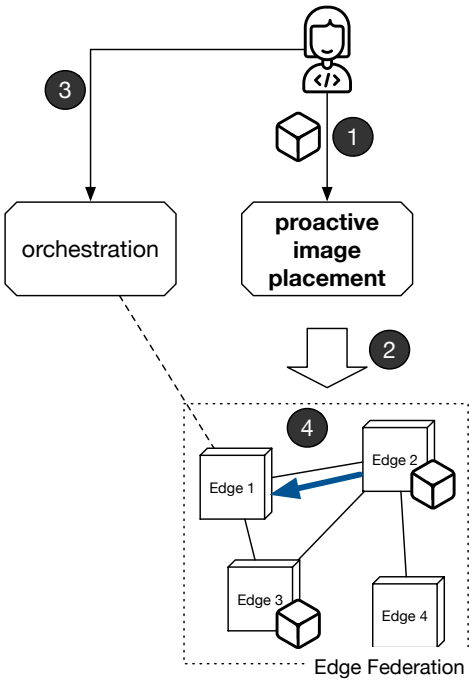



Figure 1: Envisioned system model. In step (1), a developer prepares an application to be run on the edge. In step (2), a proactive image placement service stores the image in a subset of the available edge resources. In step (3), an orchestration service decides Edge 1 as the best resource to run the application. In step (4), Edge 1 downloads the application from Edge 2.

*can we select the most suitable Edge nodes to which proactively place application images, to minimize the average transfer time to the other devices?*

We propose an empirical analysis to answer this question by comparing the results of different placement strategies and edge communication networks. It is worth mentioning that our contribution lies not in proposing a completely new approach but rather in evaluating existing solutions from a novel perspective. In particular, we model the problem of proactive placement of application images as a Minimum Vertex Cover (MVC) problem. We assume, without loss of generality, that a centralized controller knows the graph, and that the placement decision is single for each given edge federation and set of applications. We have considered several approximated implementations of the problem, and compared them with the optimal solution (when possible), aiming to determine which approach exhibits superior performance. We run these implementations against different network topologies, simulating various possible Edge Computing environments.

Experimental results demonstrate that the Greedy implementation offers the best tradeoff for the given problem, concerning the number of allocated images and execution time. Modeling the proactive placement problem as a MVC enables a concise and precise representation, effectively reducing the complexity of the problem formulation. MVC has a well established theoretical foundation in graph theory, with extensive research and efficient algorithms available for finding the MVC in various graph topologies. By leveraging this rich body of knowledge, we can benefit from proven techniques and algo-

rithms to solve the placement problem optimally or approximately. Additionally, MVC-based solutions offer resource optimization benefits. By selecting the minimum vertex cover, we ensure that the fewest number of nodes necessary to cover all the edges are involved, leading to efficient resource allocation. This reduces the computational and memory overhead associated with the placement process, improving overall system performance.

The rest of the paper is organized as follows. Section 2 serves as a literature review in the field of optimal image placement in Edge computing environments. Section 3 describes in detail the problem formulation and the proposed approach for the set cover problem. Section 4 introduces the Minimum Vertex Cover for solving the component image placement problem along with the various algorithm implementations. Moreover, Section 5 presents the experimental evaluation, the simulation methodology, and discusses the experimental results. Finally, Section 6 summarizes the merits of our work and highlights some perspectives that require further attention in the future.

## 2. Related Work

In terms of optimally placing images in Edge computing environments, there is no shortage of scientific endeavors. These scientific endeavors can be categorized based on the strategies and paradigms which are deployed, in order to tackle this rather demanding problem. It is of paramount importance to set some specific distinctions to properly examine the problem at hand, as well as the various attempts at solving it. Thus, the related work conducted in this particular field can be dissected in the following manner, which spans four distinct categories:

The *first distinction* is based on how the image placement problem is formulated from a technical perspective. The formalization of the image placement problem can be conducted in various ways. The first and most widely used, requires the use of Integer Programming. There are several variations of the Integer Programming paradigm as indicated in [5, 6, 7]. In [8], the authors examined the use of Integer Nonlinear Programming in the context of Fog Computing, while in [9], the use of Mixed Integer Linear Programming was suggested. Another approach to the formulation of the image placement problem, requires the implementation of Constrained Optimization, as shown in [10]. Constrained Optimization is a class of methods that are able to discover the best values for specific variables while taking into consideration some specified restrictions. Finally, according to [11], Markov Decision Processes [12], stochastic optimization [13], and general convex optimization are also some possible solutions to the challenge of properly formalizing the image placement problem.

The *second distinction* reflects how the optimization problem is framed within the context of the metrics utilized. There are several metrics that have been used across the scientific literature. In [14, 15, 16], the authors aimed to minimize the service's latency involved in the image placement process. As a matter of fact, low latency is considered integral in the context of delay-sensitive applications. As such, the majority of the scientific efforts that aim to tackle the image placement problem, utilize latency as their metric of choice. When contemplating the nature of Edge and Cloud infrastructures, the importance of minimizing operational costs becomes apparent. It is, therefore, safe to conclude that the cost of performing image placement is another measure that should be considered, as indicated in [17]. In addition, resource utilization is another notable metric. Some interesting ways of optimizing resource usage are explored in [18] and [19]. Last but not least, the congestion rate is another metric that is worth exploring. Yu et al. [20] considered the congestion ratio in the form of a potential minimum ratio between the flow and capacity of the link to accommodate service placement.

The *third distinction* concerns the way in which operational control will be implemented. Image placement processes can be carried out in a centralized, distributed, or federated manner. In the context of centralized control logic, decisions are made while considering all the infrastructure resources and deployment processes involved. The majority of scientific works devoted to tackling the image placement problem, leverage some sort of centralized control logic. In [21], this particular paradigm was explored. The distributed control logic, on the other hand, leverages multiple nodes in order to establish an orchestration and management layer, which can provide image placement functionalities. This distributed layer is capable of formulating strategies and making decisions based on information that is harvested on a local level. This enables the creation of a quite robust control plane, which is capable of facilitating the dynamic nature of Edge infrastructures. Up to this point, only a small number of scientific endeavors leverage distributed control planes. Wang et al. [22], introduced a Fog Computing architecture that relies on the underlying coordination of the various Fog nodes.

The *fourth and final distinction* highlights the significance of our work. All the attempts mentioned so far focus on optimally placing images in Edge computing environments in a reactive manner. The placing strategies are invoked only after a particular service has been requested. Given the functional complexity, as well as the demanding Quality of Service (QoS) requirements that are associated with contemporary services, this type of reactive paradigm seems to be rather inadequate. Instead, in this work we focus on implementing a proactive approach. Up to this point, the topic of proactive image placement has largely remained unexplored by the scientific community, with only a couple of exceptions. In [23], the authors proposed a proactive microservice placement and migration approach that prefetches microservices based on the workflow dependency structure in the application microservice Directed Acyclic Graph (DAG). More specifically, a Reinforcement Learning based mechanism is employed to proactively deploy microservices on edge servers, considering the structure of the microservice graph application. This research work focused on linear workflow microservices, and the results revealed an average improvement of 28% in latency compared to traditional reactive approaches across various state-of-the-art MEC microservice benchmark models. In [24], the authors direct their efforts towards establishing a service placement and migration model that leverages mobility prediction in the con-

text of Mobile Edge Computing (MEC). More specifically, this paper focused on improving resource allocation and management for mobile devices connected to Fog computing infrastructure. It proposes a virtual machine (VM) placement and migration decision model based on mobility prediction. By moving the VM to a Fog node ahead of the user's route, the proposed approach significantly reduces the number of user migrations needed, with a potential decrease of nearly 50%. An Integer Linear Programming (ILP) model is introduced to optimize the VM placement within candidate cloudlets. Simulations indicated that this proactive VM migration approach effectively reduces migration along the user's path without compromising VM latency. Both of these works, despite being remarkable in their own right, do not delve into detailed analysis regarding the potential effects the various algorithmic approaches or network topologies may have on the proactive image placement problem, but instead focus on examining a set of rather specific solutions.

Beyond this classification and highlighting of distinctions between our proposed approach and existing works in the scientific literature, it is worth mentioning that an approach is often used when there is a need for finding a multi-objective solution with contrasting needs: the Pareto front. In fact, the problem we face encompasses two conflicting objective functions: average response time and total transfer cost. While minimizing response time necessitates frequent and proximate image transfers to requesting nodes, minimizing transfer costs requires less frequent transfers to distant nodes. Although the Pareto front is commonly employed to address similar multi-objective optimization problems, it poses specific challenges in our context. **Size and Representation Complexity.** The Pareto front can be extensive, making its computation and representation quite complex. In our case, it depends on multiple factors, such as node quantity, node locations, image demand, image popularity, transfer capacity, transfer cost, and network dynamics. These factors may vary spatially and temporally. Furthermore, the Pareto front does not account for preferences, which may involve additional criteria or constraints for selecting the most suitable solution. It treats all Pareto-optimal solutions as equally valid without determining the most desirable choice. In our scenario, specific preferences or constraints, such as quality of service, budget, or security, can significantly influence the selection of the optimal solution. **Sensitivity and Instability.** The Pareto front sensitivity to small variations in problem parameters or data, leads to unstable or inconsistent solutions. Minor changes in initial conditions can lead to distinct or even non-Pareto-optimal solutions. In our study, the dynamic nature of the network, affecting image demand, popularity, transfer capacity, and node locations, can cause rapid and substantial variations in the Pareto front. Considering these features of the context we target, we adopt an alternative approach. This method offers several advantages, including adaptability to network dynamics, incorporation of preferences, and the ability to approximate a set of Pareto-optimal solutions with a lower computational complexity.

To the best of our knowledge, our work is the first to propose to model the problem of proactive placement of application images as a Minimum Vertex Cover problem. We analyze a spectrum of algorithmic approaches, spanning from traditional ILP and Approximation methods to more sophisticated ones like Greedy and Genetic algorithms. Our exploration extends beyond mere theoretical propositions; it encompasses diverse network topologies, spanning from regular ones like Star, Binomial and Balanced trees to more real-world graphs such as Erdö-Rényi, Watts-Strogatz and Barabási-Albert. What sets our research study apart is the holistic evaluation we adopt, integrating a multitude of metrics to assess algorithmic efficiency. Additionally, our assessment spans beyond conventional factors like latency and operational costs; it spans a multi-dimensional spectrum, accounting for the object's volume, available link bandwidth, transfer time constraints, placement cost function, and node storage capacity. Finally, while most research revolves around reactive methodologies, this work aims to shift the focus towards a proactive paradigm.

## 3. Set Cover Problem

In this section, the proposed approach for the set cover problem is presented. Specifically, we first present the problem formulation, the constraints, and the optimization goal of the problem. Then, subsection 3.1 demonstrates the solution space, thus presenting the data placement architecture. Finally, subsection 3.2 introduces the transformation of set cover to an optimization problem.

**Problem formulation.** Locate the nodes on which one or more replicas of the data objects must be placed in order to achieve the constraint for all nodes $d$, while minimizing the cost function for the whole network.

**Constraint.** $V_{obj}/W_{sd} < K$

**Optimization Goal.** $min(F(K, Vobj, Rs))$

$R$ and $W$ are considered as constants for each specific time window $t_0 + K$, where $t_0$ is the starting time of our time window. That means that we are taking a snapshot of the network's state at time $t_0$, and this snapshot has a lifetime of $K$. We are considering the available bandwidth $W_{sd}$ as equal to $W_{ds}$, meaning that the direction of the edge is not affecting its available bandwidth. To formulate the feasible solution space, we are considering all combinations of data placement on nodes s, having one or more replicas of the data object, and pruning all combinations that fail to achieve the constraint set by $K$ for each node $d$. Each such combination is considered a feasible solution. As optimal solutions, we consider the ones that are feasible and minimize the cost function $F$ for all nodes involved in them. Main notations are summarized in Table 1.

### *3.1. Solution Space*

A solution is in essence a data placement architecture. It describes which nodes will be used as data storage nodes and how many replicas of the data object will be placed on the network. As a result, each solution ($i$) is composed of two parts;

Table 1: Notations

| Notation | Description |
|---|---|
| Node $s$ | Source node of a data object |
| Node $d$ | Destination node of a data object |
| $V_{obj}$ | Volume of data object |
| $R_s$ | Charge rate for data storage in node $s$ |
| $W_{sd}$ | Available link bandwidth between nodes $s$ and $d$ |
| $K$ | Time constraint for data object transfer $\forall$ node $d$ |
| $F$ | Cost function of data object placement on node $s$ |
| $N$ | Set of all nodes in the network |

a) the data placement architecture ($DP_i$) and b) the total cost ($TC_i$). The cost is derived using a function of the volume of the data object ($V_{obj}$), the cost rate of data storage in node $s$ ($R_s$), and the available storage capacity of node $s$ ($C_s$) for each node $s$ that is part of $DP_i$. For a solution to be considered feasible, it must preserve the constraint of data transfer time for all possible destination nodes $d$. This means that the time $V_{obj}/W_{sd}$, where $V_{obj}$ is the size (volume) of the object and $W_{sd}$ the available bandwidth of the connection, must be always less than the transfer time threshold $K$ for every node $d$ in the network $N$ and for at least one node $s$ that takes part in $DP_i$.

This is described in a mathematical way in function:

$$V_{obj}/W_{sd} < K \mid \exists s \in DP_i, \forall d \in N \tag{1}$$

For the solution $i$ which is defined as:

$$\begin{aligned} DP_i &= [Node_0, Node_1, \ldots, Node_s] \text{ and} \\ TC_i &= \sum_s F(V_{obj}, C_s, R_s) \mid \forall s \in DP_i \end{aligned} \tag{2}$$

### 3.2. Optimization Problem

We formalize the problem we want to solve as an optimization problem, i.e. in terms of objective function and constraints. In order to achieve that, the objective function and the restraints need to be adjusted in a form more fitting to linear optimization logic. The objective function can be considered a function of only the activated nodes and the volume of the image being transferred because, as discussed, all the other variables ($R, W$) are considered constants for the duration $K$ we are examining. The new objective function, which is also our minimization target is described as follows:

$$min(\sum_n (A_n * V)) \mid A_n \in \{0, 1\}, \forall n \in N \tag{3}$$

This new objective function employs a new set of variables $A_n$ which are binary, taking the values of 1 or 0, describing the activated or deactivated state of a node $n$ respectively. In our case, activation means that node $n$ is holding a replica of the image and is able to distribute it to other nodes that are connected to it. The next step in defining a linear optimization problem is defining the set of restraints that need to be applied which are the following:

$$\exists n \in N : A_n * V/W_{nd} < K \mid \exists d \in D \tag{4}$$

Equation 5 describes the restraint posed by the transfer time threshold. It states that if the optimization algorithm decides to activate a node $n$, it must ensure that the transfer time of the image from that node, given by the division of the image's volume $V$ and the available bandwidth $W_{nd}$ between the activated node $n$ and at least one destination node $d$ in the list of destination nodes $D$ that are requesting the image, must be under the threshold $K$. If the node is not activated the value of $A_n$ would be 0 so the restraint would hold true anyway.

Trying to implement "at least one" in a practical linear optimization algorithm is almost impossible. For this reason, we need to reformat this function, viewing it from different angles that encompass it. The rewritten format denotes that for each destination node $d$ the total transfer time of images from all nodes $n$ must be more than zero, ensuring that at least one of the nodes $n$ is activated and able to serve the image to the destination node $d$. This is described in the following function:

$$\sum_n (A_n * V/W_{nd}) > 0 \mid \forall n \in N, \forall d \in D \tag{5}$$

Moreover, we need to ensure that the activated nodes are transferring the image following as close as possible to the time threshold $K$. This means that we have to relax the constraint of the time threshold, reducing it to a "should" rule instead of a "must" rule. This relaxation practically means that we expect some slight violations of the constraint to happen, translating into possible QoS violations in a real-life scenario. Since our target is the minimization of the function we can just include the minimization of total transfer time, which ensures that the transfer time will be kept as low as possible, covering, in most cases, our need for a transfer time lower than the threshold $K$. In order to mathematically depict this change we have to reformat our objective function, having the transfer time included in it as follows:

$$\begin{aligned} min(\sum_n (A_n * V) + \sum_n \frac{A_n * V}{W_{nd}}) \\ A_n \in \{0, 1\}, \forall n \in N, \forall d \in D \end{aligned} \tag{6}$$

Now that the objective function has two factors in it, another problem arises; we need to balance these two factors in order to have an equal impact on the final value of the function. This problem arises from the fact that transfer times are usually counted in milliseconds while volume is counted in megabytes or gigabytes, which means that a change in the first factor of the function would greatly affect the final value while a change in the second factor would have only a slight effect. For that reason, we decided to remove the volume value from the first factor, applying only the sum of $A_n$ variables as a weight on the total transfer time in the network. This means that if a solution achieves the same or less total transfer time while using fewer nodes as image sources then it will be preferred over the others. The final objective function is the following:

$$min(\sum_{n} A_n + \sum_{n} \frac{A_n * V}{W_{nd}}) \quad (7)$$
$$A_n \in \{0, 1\}, \forall n \in N, \forall d \in D$$

## 4. Minimum Vertex Cover for Proactive Image Placement

In this section, MVC is introduced for modeling and solving the problem of proactive image placement. A minimum vertex cover constitutes an optimal solution to the vertex cover (VC) problem, as it is able to obtain a vertex cover of minimum size in a given graph. Several approximated implementations are considered and compared with the optimal solution (when possible). The network is treated as a graph where each node can (i) host an application image, (ii) require an application image or (iii) do both or none of the above.

### *4.1. Preliminaries*

Given an undirected graph $G = (V, E)$, a vertex cover is a subset of vertices $S \subseteq V$, such that for each edge $(u, v) \in E$, either $u \in S$ or $v \in S$, or both. That is, each vertex "covers" its incident edges, and a vertex cover for $G$ is a set of vertices that covers all the edges in $E$ [25]. The optimization version of the VC problem is to find a vertex cover of minimum size in a given graph, so called minimum vertex cover [26]. MVC is a prominent combinatorial optimization problem with many real-world applications, including network security [27], scheduling, feature selection [28], industrial machine assignment, and applications in sensor networks such as monitoring link failures, facility location and data aggregation [29]. MCV is also closely related to many other graph problems. For instance, the independent set problem [30] is similar to the MVC problem, as it defines a maximum independent set and vice versa.

Vertex cover is a classic NP-complete graph problem [30], thus finding an optimal solution for the MVC is hard to approximate. Specifically, it is NP-hard to approximate the optimal solutions for MVC within any factor smaller than 1.3606 [31]. Therefore, this problem has been extensively studied over the years and many approximated algorithms have been proposed as an alternative to construct vertex cover in different ways. Several algorithms are presented below to solve the VC problem.

### *4.2. Algorithms For Solving Minimum Vertex Cover Problem*

#### *4.2.1. Approximation*

Approximation algorithm takes as input an undirected graph $G$ and returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover [26]. The 2-approximation algorithm is known for about 40 years, and it is utilized for general graphs. This algorithm exploits the fact that the higher out degree of a vertex in the graph, the more likely it is to be in the minimal vertex cover. The algorithm works by arbitrarily picking an edge in the graph, adding its nodes to the solution, and then removing the edge and all of its neighboring edges from the graph. This is done because the nodes added to the solution now cover all the edges, so they do not need to be considered. By repeating this process until each edge of the graph is removed, the algorithm guarantees that the solution will cover all edges in the graph by design. Due to its structure, it is not particularly complex and therefore fast compared to the other algorithm options. However, it does not give exact solutions except for particular cases, which can be disadvantageous depending on the application of it.

#### *4.2.2. Integer Linear Programming*

Integer Linear Programming (ILP) is a subset of Linear Programming algorithms that focuses on integer calculations. In our case, we are using a mix of integer and real number variables, resulting in a Mixed Integer Programming (MIP) solution. MIP and ILP are often considered the same and used intertwined. The basic principle of linear programming is that we need to define an objective function, which acts as a minimization or maximization target, and a set of constraints that define the limitations that are present in the problem. Using these limitations, the linear programming algorithm tests a great number of numerical combinations for its variables, calculating the result of the objective function at each point. This way, it can create a linear function that connects the values with the numerical result of the objective function. Of course, these calculations are extremely complex and they become more and more complex as more variables and constraints are added to the problem. More information about the exact constraints and objective function we employed can be found in subsection 3.2.

#### *4.2.3. Greedy*

The greedy algorithm is quite powerful and works well for a wide range of problems. More specifically, the greedy technique is used to solve optimization problems. A greedy algorithm always makes the choice that looks best at the moment, which tries to maximize the return on the basis of examining local conditions, with the assumption that the result will lead to a globally optimal solution [26]. It never reconsiders this decision in the future. However, generally, the greedy algorithm does not provide globally optimized solutions but merely provides a compromise by providing acceptable approximations. In this work, a clever greedy algorithm is employed which always takes the vertex highest degree.

#### *4.2.4. Genetic*

Genetic Algorithms (GAs) are heuristic search algorithms premised on the evolutionary ideas of natural selection, genetic change and endurance of the fittest of biological organisms [32]. The basic concept and techniques of these algorithms are inspired by the Darwinian theory of evolution such as crossover, mutation, and natural selection. As such, they represent an intelligent exploitation of a random search within a defined search space to solve a problem [33]. When exploration space is too large and the solution can be assigned some fitness value, then GAs thrive. VC problem may present numerous solutions which can be judged by assigning fitness value on the basis of to what extent the cover is minimum. Therefore, GAs can be applied for solving the VC problem.

More specifically, the evolution starts from a population randomly or heuristically initialized, called chromosomes that encode candidate solutions to a problem. The algorithm selects some parent chromosomes from the population set according to their fitness value [34], which are evaluated by a fitness function. The fittest chromosomes have more chances of selection for genetic operations in the next generation. Individuals are stochastically selected and modified by crossover and mutation operators to form a new population. In every generation, a new set of artificial creatures is created using bits and pieces of the fittest chromosomes of the old population. Although GA is probabilistic, in most cases it produces a better population compared to their parent population because selected parents are the fittest among the whole population set, and the worse chromosomes die off in successive generations. This process continues until the optimal solution is found or user-defined termination criteria are satisfied. The optimization mechanism of the GA system is based on four important features: (i) fitness function which is applied when children are choosing the next population, (ii) encoding scheme of the problem which represents the candidate solutions, (iii) crossover which is a genetic recombination process of segments between two pairs of chromosomes called parents in order to produce a new offspring, and finally (iv) mutation which maintains the genetic diversity between successive generations of population and prevents the over-similarity of the chromosomes in the population. After mutation and crossover, the next step is to choose the elite individual. If the best solution of the current generation is better than the elite individual of all previous populations, then it is saved as an elite individual.

## 5. Experimental Evaluation

This section presents the experimental evaluation in relation to various performance indicators. Extensive experiments are performed to investigate the properties of the proposed algorithms in the set cover problem and consequently in the proactive image placement. In the following, we describe the simulation methodology (subsection 5.1) and analyze the experimental results (subsection 5.2).

### *5.1. Simulation Methodology*

For the simulation of the networks, the image placement, and the image transfers between the nodes of the networks, a set of Python (3.6.9) scripts were developed and utilized. The NetworkX [35] Python package is exploited to create a wide variety of network topologies with different parameters. NetworkX also creates objects that allow the storage and easier manipulation of edge and node labels and characteristics such as the total capacity, the existence of an image replica, and the total usage of a network connection.

In detail, each network connection, modelled as an Edge, has two attributes, its available bandwidth, and its usage. A number of secondary values can be extracted from these two attributes, such as the percentage of bandwidth usage and the transfer time. Nodes have only one label that specifies the serial number or ID and an attribute that has a boolean value that becomes 1 if the node holds a replica of the image or 0 if it does not. In addition, 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. Using these network topologies and attributes, as well as the general parameters of the simulation, a large number of experiments is performed, utilizing a number of different algorithms in order to score and rank their performance under various conditions.

As mentioned in 4.2, the algorithms proposed and compared in this research work are: Approximation, Greedy, Genetic and ILP. It is proven that the computational requirements of Branch & Bound algorithm increase exponentially with the input size [36] and this significant cost is further exacerbated by large outlier ratios [37]. As a result, we decided to exclude this algorithm from our work.

The performance of the examined algorithms is evaluated in a wide variety of network graph topologies. Each topology is simulated in the graphical size spectrum, $V = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]$. However, due to the nature, the characteristics, the input parameters and the connectivity properties of each topology, the number of edges produced by the different number of vertices, varies considerably in each graph. Subsection 5.1.1 presents a detailed overview of the different graph topologies.

#### *5.1.1. Graph Topologies*

**Star graph**. The star graph $S_N$ of order $N$, also referred as "n-star" [38], is a complete bipartite graph, thus a tree with one internal node and $N$ leaves. In essence, a $S_N$ denotes a star graph on $N + 1$ nodes. The internal node has a vertex degree of $N - 1$ and the other $N - 1$ nodes have a vertex degree of 1 (end vertex). The star graph is hierarchical, vertex and edge symmetric, maximally fault tolerant, and strongly resilient [39, 40]. It has been known as a viable candidate for interconnecting a large number of processors in distributed systems [41]. The features of the star graph include a low degree of the nodes, small diameter, partitionability, symmetry, and a high degree of fault-tolerance [42].

**Balanced tree**. In order for a tree to be considered balanced, it is necessary that any given node which belongs to this specific tree has two sub-trees, whose heights differ by at most one. It has been proven that the height of any given balanced tree with $N$ nodes can not be greater than $1.44 \log N$ [43].

Various combinations of the branching factor and the height of the tree are considered in this work. More specifically, the different variants of the Balanced tree are the following: i) fixed $r = 2$ for different $h = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ - $BT - r_2$, ii) fixed $r = 3$ for different $h = \{1, 2, 3, 4, 5, 6\}$ - $BT - r_3$, iii) fixed $r = 5$ for different $h = \{1, 2, 3, 4\}$ - $BT - r_5$, iv) fixed $h = 2$ for different $r = \{1, 2, 4, 8, 16, 32\}$ - $BT - h_2$ and v) fixed $h = 4$ for different $r = \{1, 2, 4\}$ - $BT - h_4$. These variants produce significantly different graph structures, completely changing the number of vertices and edges as well as the connectivity between nodes. As the branching factor increases, we gradually decrease the height of the tree ($BT - r_3$ and $BT - r_5$), as extremely large graphs are generated. For instance, with a branching factor of

5 and a height of 8, a balanced tree with 488.281 vertices and 488.280 edges is created. Similarly, as the height increases, we gradually decrease the branching factor ($BT - h_4$), for the same reason.

**Binomial tree**. A binomial tree of order 0 consists of a single node. A binomial tree of order $N$ is defined recursively by linking two binomial trees of order $N - 1$, thus the root of one is the leftmost child of the root of the other.

**Erdö–Rényi random network**. Random graphs were introduced by Paul Erdös and Alfréd Rényi, who discovered that probabilistic methods were often useful in tackling problems in graph theory [44, 45]. As these types of networks were first introduced back in the 1950s, the computing power was almost insignificant compared to modern systems. As a consequence much of the modeling was based on relatively small "ordered" or "regular" networks, which are rare in the real world [46]. In their first article [45], Erdö–Rényi define a random graph as $N$ labeled nodes connected by $M$ edges, which are chosen randomly from $N(N-1)/2$ possible edges. In other words, the $G(N, M)$ model randomly selects a graph from the set of all possible graphs. An alternative and equivalent definition of a random graph is the binomial model. More specifically, the $G(N, p)$ model starts with $N$ nodes and connects each distinct node pair with probability $p$. In this work, we consider only the $G(N, p)$ model. Figure 2 illustrates an example of an Erdö–Rényi random graph of $N = 50$ nodes and a probability of $p = 0.2$. Nodes with higher degree are placed centrally, and those with lower degree away from the center.

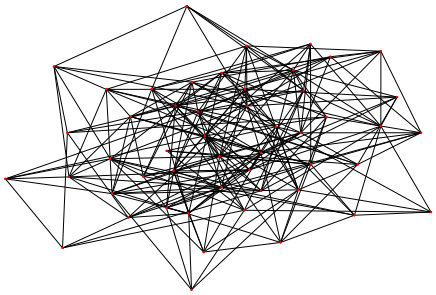


Figure 2: Undirected random network generated by the Erdö–Rényi model with $N = 50$ and $p = 0.2$

Based on the probability $p$ indicating an edge existing between any two nodes, three different variants of the Erdö–Rényi model namely, $p = 0.2$, $p = 0.5$ and $p = 0.7$, are considered in this work. As expected, the structure of the graph generated by each probability, varies significantly. Figure 3 demonstrates how the number of edges varies for the same number of vertices, between the three different probabilities; as $p$ increases, the number of edges also increases.

The uncorrelated Erdö–Rényi random graph model assumes that each pair of the graph's vertices are associated with equal and independent probabilities, thus treating a network as an assembly of equivalent units. However, real networks are fundamentally correlated systems, and in many respects, their topology deviates from the uncorrelated random graph model. The attention has shifted towards more advanced graph models, including the Internet and the World-Wide Web [47]. These are the systems that are defined as "real-world" networks or graphs. The proliferation of data has led to a disruption of activity toward understanding the general properties of real networks. Two classes of models have emerged from these efforts, commonly called "small-world" and "scale-free". Small-world networks aim to capture the clustering observed in real graphs and are heterogeneous 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.

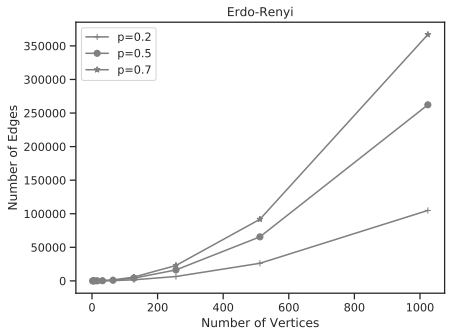



Figure 3: Number of edges and vertices generated by the Erdö–Rényi model, between the three different probabilities $p = 0.2$, $p = 0.5$ and $p = 0.7$.

**Watts–Strogatz small-world network**. Watts and Strogatz [48] proposed what has become known as the archetypical small-world network. The algorithm behind the model starts by creating an undirected ring lattice network,which is a ring of nodes with edges divided evenly between its $k_L$ closest left and right neighbors. $k_L$ demonstrates the degree of every node in the initial lattice. Then a process of random rewiring is applied, whereby each edge has an arbitrarily chosen probability $p$, of being re-wired. The algorithm rewires only one end of each edge, and traverses edges in a manner that ensures each node loses at most half of its edges. In addition, edges are only replaced, not added or removed and therefore the total number of edges and the mean degree is unchanged [49].

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.

Therefore in this work, we considered only one rewire probability $p = 0.5$ for three different degree values namely, $k_L = 2, k_L = 4$, and $k_L = 7$. Since the degree is, in essence, the number of edges between a node and other nodes in the graph, the structure of the graph generated by each degree value varies significantly. Figure 4 demonstrates how the number of edges varies for the same number of vertices, between the three differ-

ent degrees; as $k_L$ increases, the number of edges also increases.

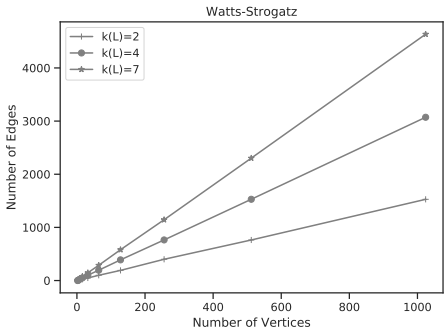



Figure 4: Number of edges and vertices generated by the Watts–Strogatz model, between the three different degree values $k_L = 2, k_L = 4$ and $k_L = 7$ for $p = 0.5$.

For the purposes of analytic treatment, the Watts–Strogatz model presents a number of problems. More specifically, the distribution of the total number of rewired connections is not completely uniform. For example, configurations with more than one connection between a particular pair of vertices are explicitly forbidden. This non-uniformity of the distribution makes an average over different realizations of the randomness hard to perform. Another problem is that the average distance between pairs of vertices on the graph is insufficiently defined [50].

**Barabási-Albert scale-free network**. Many "real world" networks include citation networks, the world wide web, and biological networks. These systems are constantly growing by the continuous addition of new nodes. For example, the world wide web grows exponentially over time with the addition of new web pages. Most real networks exhibit the preferential attachment property, such that the likelihood of connecting to a node depends on the current node's degree. For instance, much cited and well-known publications are more likely to be cited than less cited and consequently less known papers [46]. The origin of the power-law degree distribution in networks was first introduced by Barabási and Albert [51]. They proposed the scale-free network which is able to reproduce networks with "hubs", thus a small number of nodes with a much larger than average number of edges to/from other nodes, a property known as scale-free. As a result, the degree distribution is highly heterogeneous.

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 $m_o = m$). One cannot choose $m > m_o$ as then the first new node introduced cannot be assigned m edges. Thus, the initial network size $m_o$ determines the maximum mean degree of the network. When choosing the nodes to which the new node connects (preferential attachment), the probability $P$ that a new node will be connected to node $i$ depends on the degree $k_i$ of node $i$, such that:

$$P(k_i) = \frac{k_i}{\sum_j k_i}$$

The combination of network growth with this preferential attachment is what leads to a power-law degree distribution. In contrast to the small-world model, the distribution of degrees in a scale-free graph converges to a power-law when $N \to \infty$, which has been shown to be a combined effect of growth and the preferential attachment [52]. Thus, in the infinite time or size limit, the scale-free model has no characteristic scale in the degree size.

Figure 5 demonstrates an example scale-free network generated by the Barabási and Albert algorithm. Nodes with higher degree are placed centrally, and those with lower degree away from the center. Furthermore, a small number of high-degree "hubs" are presented.

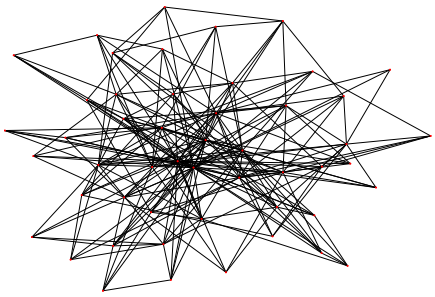


Figure 5: Barabási and Albert with $N = 20$ and $m = m_o = 5$

Three different variants of the Barabási and Albert model namely, $m = 1, m = 3$ and $m = 8$, are considered in this work. As expected the structure of the graph generated by each value $m$, varies significantly. Figure 6 demonstrates how the number of edges varies for the same number of vertices, between the three different values; as $m$ increases, the number of edges also increases.

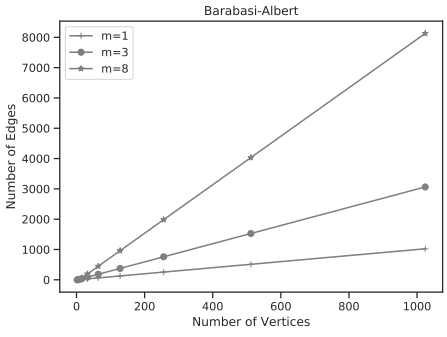



Figure 6: Number of edges and vertices generated by the Barabási-Albert model, between the three different values $m = 1, m = 3$ and $m = 8$.

As so many "real world" networks display degree distributions similar to the Barabási-Albert model, it remains one of the most well known and often used network generation methods.

### *5.1.2. Evaluation Metrics*

In order to evaluate the performance of each algorithm in solving the component image placement problem, four performance metrics are utilized: i) execution time ($ExT$), ii) approx-

imation ratio ($AR$), iii) cost function ($CF$) and iv) size of the VC ($VCS$).

Execution time refers to the total amount of time each algorithm requires to produce a solution. Approximation ratio is defined as the ratio of the VCS produced by an algorithm over the VCS produced by the optimal solution and is formulated as shown in Equation 8:

$$AR = \frac{VCS_{algorithm}}{VCS_{OPT}} \tag{8}$$

ILP algorithm is guaranteed to be the optimal solution in producing the minimum vertex cover for a given graph and, therefore, is utilized as the benchmark of correctness, by which the $AR$ seeks to evaluate all other algorithms with.

One of the most important metrics of evaluation is the cost function, which is also a minimization target in the ILP and the Genetic algorithm. The cost function is the same as the objective function (Function 7), as described in Section3.2. This function calculates a 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.

Finally, the last metric employed is the size of the VC, hence the size of vertices in it.

#### *5.1.3. Bandwidth Allocation*

When evaluating content distribution systems by means of simulation, it is of utmost importance to correctly mimic the bandwidth dynamics behaviour of the underlying network. Available bandwidth in P2P systems is affected by the presence of peers connected to the network via heterogeneous bandwidth links [53]. Flow-level network simulators implement algorithms, which are able to reproduce the bandwidth dynamics occurring in the transport protocol layer of a real network. Max-min fairness introduced in [54], for modeling additive-increase multiplicative-decrease congestion control protocols like TCP. In general, Max-min fairness 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 progressively each rate is increased equally until reaching the link's capacity. The simulated network is modelled as an undirected graph where Max-min fairness is able to provide a good approximation of the actual network behaviour.

Based on [55], the formal definition considers a set of sources $1, ..., n$ where each of them has $x_1, x_2, ..., x_n$ resource demands and a total capacity $C$. In the initial stage, the source with the smallest demand receives $\frac{C}{n}$ of the resource. The process continues until each source satisfies its demand. Otherwise, if its demand is not satisfied, the algorithm ensures, that each source will receives a demand no less than what any other source with a higher index obtained. Such an allocation is defined as Max-min fair, as it maximizes the minimum share of a source whose demand is not fully satisfied. The idea is that the rate of the minimum flow is maximized by the algorithm. Hence, small flows are given a higher relative priority. Only when a flow demands more than $\frac{C}{n}$, it faces the risk of having its bandwidth throttled by the algorithm. Generally, a low level of fairness policies provides high average throughput but low stability in the service quality. In comparison with the equal sharing policy, Max-min fairness is able to provide higher average throughput and better utilization of the resources. On the other hand, although maximum throughput resource management provides higher average throughput than Max-min fairness, it would result in the starvation of expensive flows.

During the simulations created in the context of the present work, the bandwidth allocation was conducted using the Max-min fairness algorithm. As a total bandwidth capacity, we considered the maximum bandwidth of the node's virtual network adapter, which is randomly set to either Ethernet (100MBps) or WiFi (25MBps). In a realistic edge network most communication, is made through WiFi since a plethora of smart devices and IoT equipment are utilized, so the Wifi network adapters were arbitrarily set to a 75% ratio while Ethernet adapters were assigned to the rest 25% of nodes.

### *5.2. Simulation Results*

This section provides the simulation results obtained by the different algorithms for the various network topologies. To this end, different sets of experiments were performed to evaluate the efficiency of each algorithm:

- Execution time analysis
- Approximation ratio analysis
- Cost function analysis
- Vertex Cover Set size analysis

As mentioned in subsection 5.1.1, different variants are considered for some network topologies, namely Balanced Tree, Erdö–Rényi, Watts–Strogatz and Barabási-Albert, derived from the input parameters of each model. Due to the extremely large number of experiments, we do not graphically represent the results of all the variants of the different models. For example, trying to visualize the experimental results for the different variations of the Balanced tree graph topology, requires sixteen additional graphical representations. As a result, one representative variation per model is considered, as shown in Table 2. Nevertheless, the detailed results for all variants of each model are presented and compared in Table 4. Improved tabular data representation can be accessed on Github[1], enhancing readability and clarity. Additionally, the repository hosts a Python script serving as a valuable resource for constructing and visualizing plots derived from the data analysis results. This script not only acts as a reference but also facilitates the generation of graphical representations, facilitating a more comprehensive understanding of data trends and insights. Due to lack of space, this table only presents experimental results for 64 and above vertices. As Balanced tree constructs a different number of vertices for the

[1] `https://github.com/AntonisMakris/FGCS_Proactive-Component-Placement`

different combinations of $r$ and $h$ compared to the other network topologies, its detailed results are presented separately in Table 5. As the ILP algorithm is extremely complex in calculating the objective function, as it evaluates a large number of numerical combinations for its variables, it fails to produce a solution to a large number of experiments performed for different variations per model, and is indicated in the aforementioned Tables as *NR*.

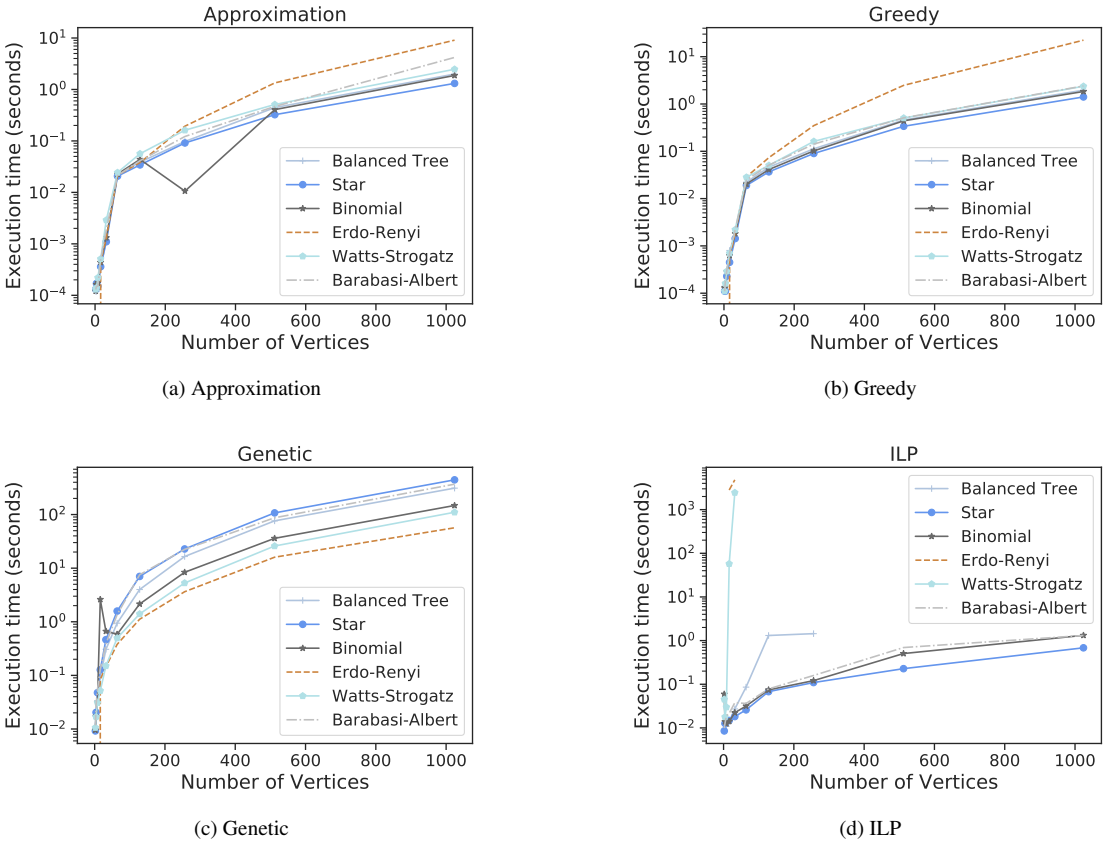
Table 2: Variations of the models considered for visualization

| Model | Variant |
|---|---|
| Balanced tree | $BT - r_2$ |
| Erdö–Rényi | $p = 0.2$ |
| Watts–Strogatz | $k_L = 2$ |
| Barabási-Albert | $m = 1$ |

*Execution time Analysis*. Figure 7 evaluates the execution time of each algorithm for the different models. As expected, the execution time of the examined models for each algorithm presents an upward trend as a higher number of vertices are selected. As the results suggest, Erdö–Rényi presents the highest execution time among all models for the Approximation, Greedy and ILP algorithms (Figure 7a, 7b and 7d respectively). Remaining models present similar behaviour for the same amount of vertices, increasing linearly the execution time when the number of vertices increases. Erdö–Rényi spent considerably more time than the other models to provide a solution for the Approximation and Greedy algorithm, especially as the number of vertices increases. In the Greedy algorithm, execution time follows this norm, however Erdö–Rényi present bigger fluctuations compared to the other models, especially for a number of vertices exceeding 256. For example, the execution time exhibited by the Greedy algorithm for Erdö–Rényi and Star graph with 1024 vertices is 22.33 and 1.40 (almost 16 times slower), while for Approximation is 9.07 and 1.31 (almost 7 slower), respectively. As shown in the case of ILP (Figure7d), Erdö–Rényi and Watts-Strogatz have extremely high execution times compared to the other models, followed by Balanced tree. Erdö–Rényi does not generate a connected graph for $N = 2, 4, 8$ and $p = 0.2$, thus no results are displayed. In addition, this model along with Watts-Strogatz fails to produce a solution for a number of vertices in excess of 64. The same applies to the Balanced tree for a number of nodes that exceeds 256. This is due to the fact that ILP is extremely complex, as it evaluates a large number of numerical combinations for its variables, in the calculation of the objective function. For example, the execution time exhibited by the ILP algorithm for Erdö–Rényi and Watts-Strogatz with 64 vertices is 4812.04 and 2453.32 seconds, while for Approximation is 0.0013 and 0.0029, respectively. Balanced tree also fails to produce a solution for a number of vertices in excess of 256 and spends considerably more time than the remaining models. The opposite behaviour is observed in the Genetic algorithm (Figure 7c), where Erdö–Rényi exhibits the lowest results, while the Star graph presents the highest execution times followed by Barabási-Albert and Balanced tree. The genetic algorithm requires significantly more time to produce a solution, regardless of the network model. As a matter of fact, the execution time for a Star graph with 1024 nodes is 443.81 seconds, while Greedy and Approximation require 1.31 and 1.40 respectively, thus almost 341 times faster. The same behaviour is observed in all models. As the results justify, execution times begin to differ significantly for all examined algorithms between the models, for a number of vertices that exceeds 128. Generally, the highest the number of vertices, the highest the execution time as the generated graph grows larger and more nodes are included in the VC set.

*Approximation Ratio analysis*: Figure 8 evaluates the approximation ratio achieved by each algorithm for the different models. ILP is employed as the benchmark of correctness, by which the *AR* seeks to evaluate all other algorithms with. As the results suggest, Erdö–Rényi and Watts-Strogatz present the highest approximation ratio among all models for each algorithm, followed by the Balanced tree. As already mentioned, these models fail to produce a solution for a number of vertices in excess of 64 and 256, respectively. In fact, the differences in the approximation ratio of the different models in relation to the rest are significantly large. For example, the approximation ratio exhibited by the Greedy algorithm (Figure 8b) for Erdö–Rényi and Barabási-Albert with 32 vertices is 3.5 and 1.1 (3.1 times larger), respectively. Erdö–Rényi maintains extremely high approximation ratio values even with small number of vertices, compared to the other models. Watts-Strogatz presents lower values than Erdö–Rényi, but remain high in all algorithms. Balanced tree has lower values compared to Erdö–Rényi and Watts-Strogatz, although the differences with the remaining models are significant. For example, the approximation ratio exhibited by the Genetic algorithm for Balanced tree and Barabási-Albert with 256 vertices is 1.8 and 1.26, respectively. Star graph and Binomial tree present exactly the same approximation ratio in all algorithms. In Greedy and Genetic, the *AR* obtained for the different number of vertices is the best possible, i.e. 1, while in Approximation it is 2. Approximation algorithm (Figure 8a) presents the highest approximation ratio values on all models, followed by Genetic and Greedy. For example, the approximation ratio exhibited by Approximation, Genetic, and Greedy for Barabási-Albert with 1024 vertices is 1.7, 1.38 and 1.02, respectively. Barabási-Albert holds the lowest *AR* in the Approximation algorithm and one of the lowest along with the Star and Binomial tree in Greedy. In the Genetic algorithm (Figure 8c), *AR* shows slightly higher values but still remains low.

*Cost function analysis*: Figure 9 evaluates the cost function of each algorithm for the different models. As the results suggest, Binomial Tree and Watts-Strogatz exhibit relatively small cost function values as the number of vertices increases for the Approximation algorithm (Figure 9a), while Star graph presents the highest values followed by Balanced Tree, Erdö–Rényi and Barabási-Albert models. For example, the cost function obtained by the Approximation algorithm for Star graph and Binomial Tree with 1024 vertices is 168392 and 1024 respectively, thus almost 164 times larger. The cost function obtained with Barabási-Albert, the second most "expensive" model in Approximation, for the same amount of nodes




(a) Approximation

(b) Greedy

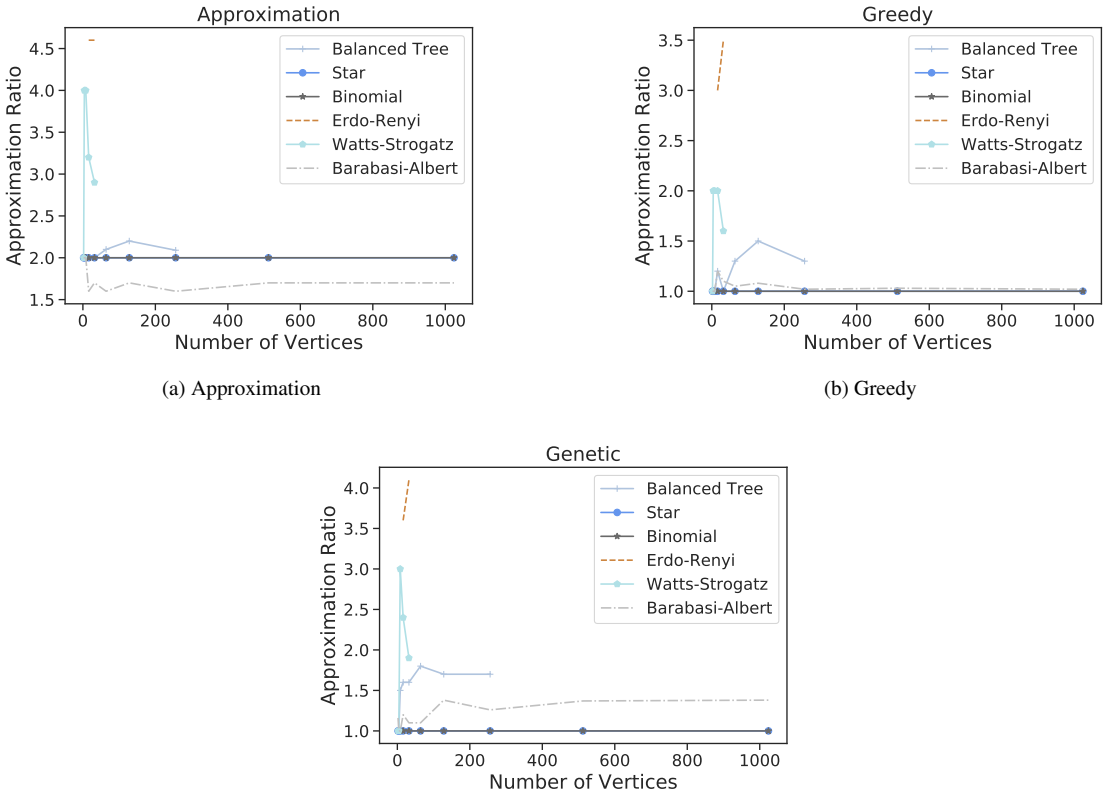
(c) Genetic

(d) ILP

Figure 7: Execution time of each algorithm for the different models

is 81269, which is almost half when compared to Star graph. It is noteworthy, that the cost function of Star graph presents large values in all algorithms, even with a small number of vertices. This claim becomes more apparent in the Approximation and ILP algorithms. In Greedy and Genetic algorithm (Figure 9b and Figure 9c respectively), Erdö–Rényi demonstrates the highest cost function values as the number of vertices increases, followed by Star graph. However, for these two algorithms, the differences in terms of cost function between the different models are not as significant as the number of vertices increases. As the results suggest, cost function values begin to differ between Erdö–Rényi and the remaining models, for a number of vertices that exceeds 256. In fact, this differentiation is more evident in the Genetic algorithm. As an example, the cost function exhibited by the Genetic algorithm for Erdö–Rényi and Star graph with 1024 vertices is 16359885 and 168556 respectively, thus almost 97 times larger. Genetic algorithm shows higher cost function values on all network models compared to Greedy, which exhibits slightly smaller results. ILP (Figure 9d) presents higher values than Genetic on all models, except Barabási-Albert and Binomial tree. In most cases, however, the differences between these algorithms are not significant. On the contrary, Approximation presents lower cost function values for all network models. For instance, the cost function obtained by the Genetic, ILP, Greedy and Approximation algorithm for Barabási-Albert with 512 vertices is 39449, 36950, 36592 and 25169 respectively. As already mentioned, Erdö–Rényi does not generate a connected graph for $N = 2, 4, 8$, and in addition, this model along with Watts-Strogatz fails to produce a solution for a number of vertices in excess of 64. The same applies to the Balanced tree for a number of nodes that exceeds 256.

*VC set size analysis*: Figure 10 evaluates the Vertex Cover set produced by each algorithm for the different models. As the results demonstrate, the Star graph produces the minimum VC set as the image is placed on the internal central node. For the Approximation algorithm (Figure 10a), Binomial Tree, Erdö–Rényi and Watts-Strogatz present almost the same VC set followed by Balanced Tree and Barabási-Albert which exhibit smaller sets. In fact, Barabási-Albert presents the smallest VC set for the different algorithms (after the Star graph), for almost all different numbers of vertices. As expected, the VC set size increases linearly as the number of vertices increases. In Greedy and Genetic algorithm (Figure 10b and Figure 10c respectively), Erdö–Rényi presents the largest VC sets, followed by Watts-Strogatz. As an example, the VC set produced by the Greedy algorithm for Erdö–Rényi and Barabási-Albert with 1024 vertices is 998 and 321 respectively, thus almost 3 times larger. In addition, Binomial and Balanced tree exhibit slightly different sets. The approximation algorithm presents the highest VC set in all network models for the different number of vertices, while ILP (Figure 10d) shows the lowest results followed by Greedy. The genetic algorithm presents intermediate results.




(a) Approximation

(b) Greedy

(c) Genetic

Figure 8: Approximation of each algorithm for the different models

For instance, the VCs set produced by the ILP, Greedy, Genetic, Greedy, and Approximation algorithm for Barabási-Albert with 512 vertices is 158, 217, 163, and 2729 respectively.

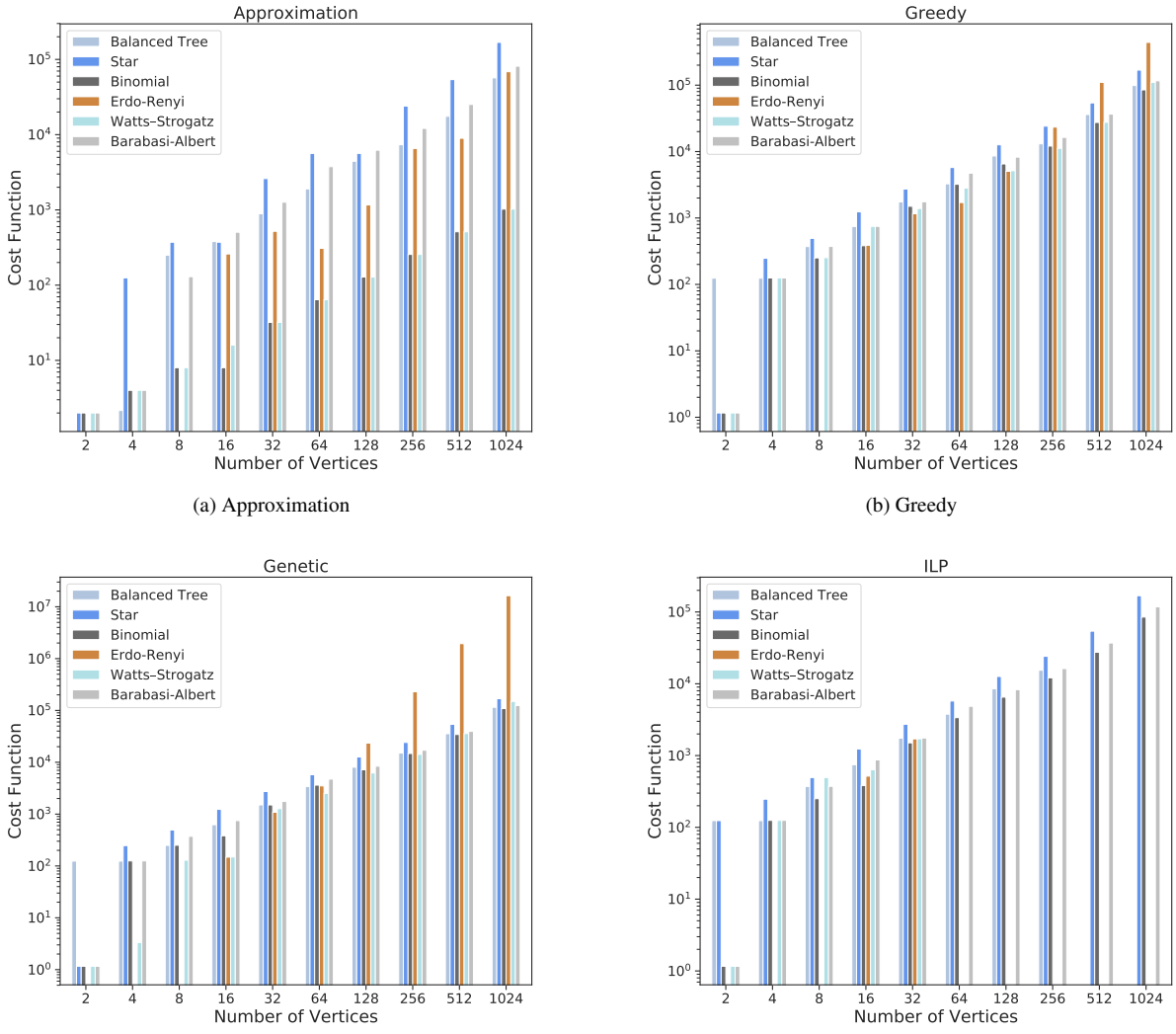
*5.2.1. Discussion*

In this subsection, a brief overview of the experimental simulation results and the final insights are presented, to summarize the performance evaluation of each algorithm.

As suggested by the results, the Genetic algorithm exhibits the highest execution times for all network models, except Erdö –Rényi and Watts-Strogatz, where ILP presents the highest results. This is due to the fact that applications of Genetic algorithms in the optimizations of complex problems can lead to a substantial computational effort as a result of the repeated evaluation of the objective function(s) and the population-based nature of the search. A substantially large number of generations might be required in some cases, to find the optimum value of the objective function [56]. ILP on the other hand, is extremely complex as it evaluates a large number of numerical combinations for its variables in the calculation of the objective function. The high execution times presented by the Genetic algorithm, are accompanied by high-cost function values. A typical example is the fact that Genetic algorithm presents the highest cost function values for the Binomial tree and the Barabási-Albert model. In the remaining models, ILP presents the highest cost function results followed by the Genetic algorithm. The differences in cost function between Genetic and the other algorithms are in some cases quite significant. For instance, the cost function value obtained by the Genetic, Greedy and Approximation algorithm for Erdö–Rényi with 1024 vertices is 16359885, 440389 and 68618 respectively. In other words, 280 and 37 times larger when compared to the Approximation and Greedy, respectively.

Although the Approximation algorithm presents the lowest cost function values and execution times in all models, it exhibits the largest VC sets. This practically means, that this algorithm is able to produce a solution faster than other algorithms, however, this solution will contain a significantly larger number of nodes. Therefore, there is a tradeoff between the total amount of time each algorithm requires to produce a solution and the number of nodes considered in the graph. In addition, the Approximation algorithm acquires the highest approximation ratio in all models followed by the Genetic algorithm.

In terms of the size of the VC set, the Genetic algorithm shows higher results compared to the Greedy and ILP algorithms for the various models. Greedy algorithm has the lowest approximation ratio in all models and comes after ILP in terms of VC set size. This algorithm achieved lower cost function values compared to Genetic and ILP and in combination with short execution times and small VC sets, proves to be one of the most efficient algorithms. ILP returns the lowest VC sets and therefore requires the minimum number of nodes storing the




(a) Approximation

(b) Greedy

(c) Genetic

(d) ILP

Figure 9: Cost function of each algorithm for the different models

image, however in some cases, it presents prohibitively large execution times. In addition, ILP fails to produce a solution for the Erdö–Rényi Watts-Strogatz and Balanced tree models, for a number of vertices in excess of 64 for the first two models and 256 for the last one. Finally, the Genetic algorithm presents better results compared to the Approximation algorithm in terms of VC set size but requires longer execution times.

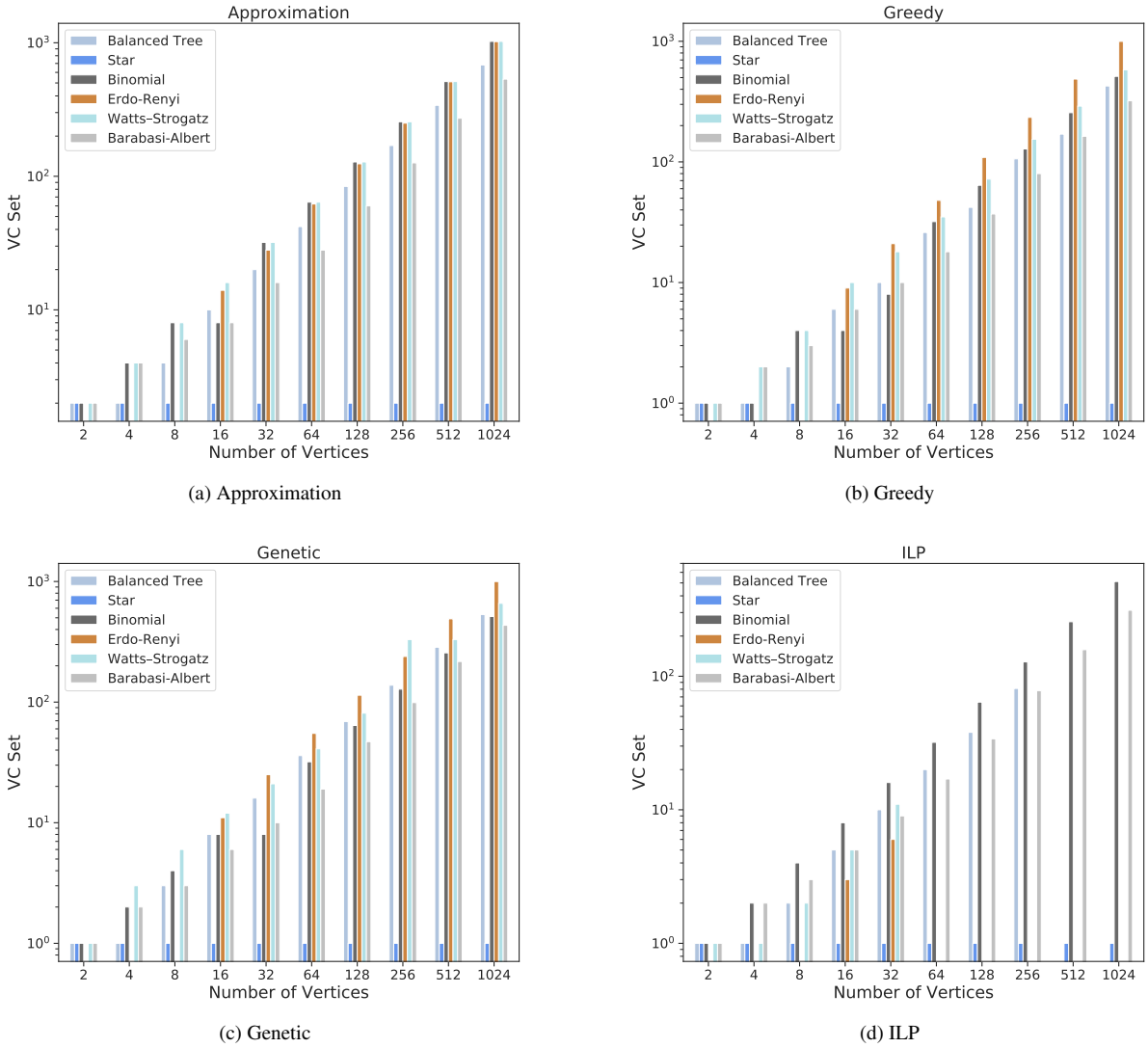
Table 3provides a comprehensive summary of the results based on the different evaluation metrics. ILP has been omitted from this table due to its inability to produce solutions for certain network topologies, as previously noted. Since the approximation ratio metric becomes inapplicable in those specific cases, we also exclude it from the table.

## 6. Conclusion

The geographical distribution of Edge resources assumes a central role in the management of Edge Computing platforms, allowing applications to be deployed near the end-users. Such placement is usually driven by scheduling decisions, which achieve to optimize various performance indicators, such as network performance, and keeping data as close as possible to its source. Enforcing scheduling decisions often means transferring and deploying, potentially large, application images into a large number of Edge resources. Therefore, it is important to provide strategies that proactively transfer such images in a way that minimizes their download time when requested, while at the same time minimizing the number of transfers.

In this paper, we model the problem of proactive placement of application images as a Minimum Vertex Cover problem. We compare several placement strategies against different network topologies, simulating various possible Edge computing environments. Our observations suggest that the Greedy implementation offers the best tradeoff in terms of performance (i.e. number of allocated images), cost function, and execution time.

As a further improvement, we plan to compare our current




(a) Approximation

(b) Greedy

(c) Genetic

(d) ILP

Figure 10: Vertex Cover set of each algorithm for the different models

proactive placement solution derived from a traditional model-driven formulation with other data-driven solutions. In particular, we are interested in exploring machine learning algorithms, such as Graph Neural Networks and Reinforcement Learning, and comparing them in terms of execution time and performance.

## Acknowledgement

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] 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.

[2] A. Younis, B. Qiu, D. Pompili, Latency-aware hybrid edge cloud framework for mobile augmented reality applications, in: 2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON), IEEE, 2020, pp. 1–9.

[3] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, G. Righetti, Cloud federations in contrail, in: Euro-Par 2011: Parallel Processing Workshops: CCPI, CGWS, HeteroPar, HiBB, HPCVirt, HPPC, HPSS, MDGS, ProPer, Resilience, UCHPC, VHPC, Bordeaux, France, August 29–September 2, 2011, Revised Selected Papers, Part I 17, Springer, 2012, pp. 159–168.

[4] X. Cao, G. Tang, D. Guo, Y. Li, W. Zhang, Edge federation: Towards an integrated service provisioning model, IEEE/ACM Transactions on Networking 28 (3) (2020) 1116–1129.

[5] H.-J. Hong, P.-H. Tsai, 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. doi:10.1109/APNOMS.2016.7737202.

Table 3: Summary of results

| Network Model | Algorithm | ExT | CF | VCS |
|---|---|---|---|---|
| Erdos-Renyi | Genetic | Highest | Highest | Moderate |
| | Approximation | **Lowest** | **Lowest** | Highest |
| | Greedy | Moderate | Moderate | **Lowest** |
| Watts-Strogatz | Genetic | Highest | Highest | Moderate |
| | Approximation | **Lowest** | **Lowest** | Highest |
| | Greedy | Moderate | Moderate | **Lowest** |
| Balanced Tree | Genetic | Highest | Highest | Moderate |
| | Approximation | **Lowest** | **Lowest** | Highest |
| | Greedy | Moderate | Moderate | **Lowest** |
| Binomial Tree | Genetic | Highest | Highest | **Lowest** |
| | Approximation | **Lowest** | Lowest | Highest |
| | Greedy | Moderate | Moderate | Lowest |
| Star | Genetic | Highest | Highest | **Lowest** |
| | Approximation | **Lowest** | **Lowest** | Highest |
| | Greedy | Moderate | Highest | **Lowest** |
| Barabasi-Albert | Genetic | Highest | Highest | Moderate |
| | Approximation | **Lowest** | Moderate | Highest |
| | Greedy | Moderate | **Lowest** | **Lowest** |


[6] M. I. Naas, P. R. Parvedy, J. Boukhobza, 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. doi:10.1109/ICFEC.2017.15.

[7] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, P. Leitner, Optimized iot service placement in the fog, Serv. Oriented Comput. Appl. 11 (4) (2017) 427–443. doi:10.1007/s11761-017-0219-8.
URL https://doi.org/10.1007/s11761-017-0219-8

[8] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, J. P. Jue, Qos-aware dynamic fog service provisioning, CoRR abs/1802.00800 (2018). arXiv:1802.00800.
URL http://arxiv.org/abs/1802.00800

[9] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, G. Pavlou, On uncoordinated service placement in edge-clouds, in: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2017, pp. 41–48. doi:10.1109/CloudCom.2017.46.

[10] F. Ait Salaht, F. Desprez, A. Lebre, C. Prud'homme, M. Abderrahim, Service placement in fog computing using constraint programming, in: 2019 IEEE International Conference on Services Computing (SCC), 2019, pp. 19–27. doi:10.1109/SCC.2019.00017.

[11] F. A. Salaht, F. Desprez, A. Lèbre, An overview of service placement problem in fog and edge computing, ACM Computing Surveys (CSUR) 53 (2020) 1 – 35.

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

[13] M. Neely, Stochastic network optimization with application to communication and queueing systems. Springer Nature, 2022.

[14] K. Velasquez, D. P. Abreu, M. Curado, E. Monteiro, Service placement for latency reduction in the internet of things, Annals of Telecommunications 72 (2017) 105–115.

[15] M. Taneja, 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. doi:10.23919/INM.2017.7987464.

[16] G. Lee, W. Saad, 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. doi:10.1109/ICC.2017.7996574.

[17] H. R. Arkian, A. Diyanat, A. Pourkhalili, Mist: Fog-based data analytics scheme with cost-efficient resource provisioning for iot crowdsensing applications, J. Netw. Comput. Appl. 82 (2017) 152–165.

[18] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, Towards qos-aware fog service placement, in: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), 2017, pp. 89–96. doi:10.1109/ICFEC.2017.12.

[19] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, A. V. Vasilakos, Fog computing for sustainable smart cities: A survey, ACM Comput. Surv. 50 (3) (jun 2017). doi:10.1145/3057266.
URL https://doi.org/10.1145/3057266

[20] R. Yu, G. Xue, X. Zhang, Application provisioning in fog computing-enabled internet-of-things: A network perspective, in: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications, 2018, pp. 783–791. doi:10.1109/INFOCOM.2018.8486269.

[21] H.-J. Hong, P.-H. Tsai, A.-C. Cheng, M. Y. S. Uddin, N. Venkatasubramanian, C.-H. Hsu, Supporting internet-of-things analytics in a fog computing platform, in: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2017, pp. 138–145. doi:10.1109/CloudCom.2017.45.

[22] P. Wang, S. Liu, F. Ye, X. Chen, A fog-based architecture and programming model for iot applications in the smart grid, ArXiv abs/1804.01239 (2018).

[23] K. Ray, A. Banerjee, 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. doi:10.1109/SEC50012.2020.00010.

[24] D. Gonçalves, K. Velasquez, M. Curado, L. Bittencourt, E. Madeira, Proactive virtual machine migration in fog environments, in: 2018 IEEE Symposium on Computers and Communications (ISCC), 2018, pp. 00742–00745. doi:10.1109/ISCC.2018.8538655.

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

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

[27] S. Cai, J. Lin, C. Luo, Finding a small vertex cover in massive sparse graphs: Construct, local search, and preprocess, Journal of Artificial Intelligence Research 59 (2017) 463–494.

[28] X. Xie, X. Qin, Dynamic feature selection algorithm based on minimum vertex cover of hypergraph, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2018, pp. 40–51.

[29] V. Kavalci, A. Ural, O. Dagdeviren, Distributed vertex cover algorithms for wireless sensor networks, arXiv preprint arXiv:1402.2140 (2014).

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

[31] I. Dinur, S. Safra, On the hardness of approximating minimum vertex cover, Annals of mathematics (2005) 439–485.

[32] J. H. Holland, Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence, MIT press, 1992.

[33] K. Kotecha, N. Gambhava, A hybrid genetic algorithm for minimum vertex cover problem., in: IICAI, 2003, pp. 904–913.

[34] D. E. Goldberg, Genetic algorithms in search, optimization, and machine learning. addison, Reading (1989).

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

[36] C. Roucairol, A parallel branch and bound algorithm for the quadratic assignment problem, Discrete applied mathematics 18 (2) (1987) 211–225.

[37] A. P. Bustos, T.-J. Chin, Guaranteed outlier removal for point cloud registration with correspondences, IEEE transactions on pattern analysis and machine intelligence 40 (12) (2017) 2868–2882.

[38] F. Harary, Graph theory, Addison Wesley, Reading, Massachusetts, 1969.

[39] S. B. Akers, The star graph: An attractive alternative to the n-cube, in: Proc. Int'l Conf. Parallel Processing., 1987, 1987.

[40] S. B. Akers, B. Krishnamurthy, A group-theoretic model for symmetric interconnection networks, IEEE transactions on Computers 38 (4) (1989) 555–566.

[41] S. Latifi, E. Saberinia, X. Wu, Robustness of star graph network under link failure, Information Sciences 178 (3) (2008) 802–806.

[42] S. Latifi, On the fault-diameter of the star graph, Information Processing Letters 46 (3) (1993) 143–150.

[43] D. E. Knuth, The Art of Computer Programming: Combinatorial Algorithms, Part 1, 1st Edition, Addison-Wesley Professional, 2011.

[44] E. N. Gilbert, Random graphs, The Annals of Mathematical Statistics 30 (4) (1959) 1141–1144.

[45] P. Erdös, A. Rényi, On random graphs publ, Math. debrecen 6 (1959)

Table 4: Detailed results for all variations of each model

| | | | | Approximation | | | | Greedy | | | | Genetic | | | | ILP | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | V | E | ExT | AR | CF | VCS | ExT | AR | CF | VCS | ExT | AR | CF | VCS | ExT | CF | VCS |
| Barabasi Albert | m=1<br>m=3<br>m=8 | *64* | *63* | 0.021 | 1.64 | 3752.51 | 28 | 0.0247 | 1.05 | 4732.42 | 18 | 1.366 | 1.11 | 4734.80 | 19 | 0.0363 | 4854.30 | 17 |
| | | | *183* | 0.020 | 2.2 | 1498.54 | 48 | 0.0263 | 1.6 | 2519.53 | 34 | 0.4941 | 2 | 2702.77 | 42 | 0.3088 | 2284.42 | 21 |
| | | | *448* | 0.023 | *NR* | 977.61 | 56 | 0.0259 | *NR* | 2087.64 | 45 | 0.4238 | *NR* | 3829.13 | 52 | *NR* | *NR* | *NR* |
| | | *128* | *127* | 0.041 | 1.76 | 6217.07 | 60 | 0.0509 | 1.08 | 8244.56 | 37 | 7.533 | 1.38 | 8393.66 | 47 | 0.0782 | 8280.75 | 34 |
| | | | *375* | 0.040 | *NR* | 3446.53 | 92 | 0.0431 | *NR* | 5974.57 | 66 | 7.554 | *NR* | 7554.51 | 83 | *NR* | *NR* | *NR* |
| | | | *960* | 0.043 | *NR* | 3200.37 | 108 | 0.0633 | *NR* | 6119.13 | 89 | 1.2152 | *NR* | 12317.88 | 103 | *NR* | *NR* | *NR* |
| | | *256* | *255* | 0.122 | 1.61 | 12096.63 | 127 | 0.142 | 1.02 | 16346.34 | 80 | 22.392 | 1.26 | 17062.14 | 99 | 0.1587 | 16260.24 | 78 |
| | | | *759* | 0.118 | *NR* | 9991.89 | 176 | 0.1271 | *NR* | 15948.10 | 127 | 7.6216 | *NR* | 20213.48 | 146 | *NR* | *NR* | *NR* |
| | | | *1984* | 0.105 | *NR* | 9175.27 | 228 | 0.1946 | *NR* | 23828.37 | 182 | 3.4397 | *NR* | 54844.29 | 202 | *NR* | *NR* | *NR* |
| | | *512* | *511* | 0.468 | 1.72 | 25169.55 | 272 | 0.509 | 1.03 | 36592.49 | 164 | 87.12 | 1.37 | 39449.04 | 217 | 0.6933 | 36950.95 | 158 |
| | | | *1527* | 0.406 | *NR* | 33823.91 | 379 | 0.4602 | *NR* | 61257.26 | 264 | 28.48 | *NR* | 79522.86 | 330 | *NR* | *NR* | *NR* |
| | | | *4032* | 0.478 | *NR* | 47196.73 | 440 | 0.8199 | *NR* | 100365.45 | 358 | 11.742 | *NR* | 233526.26 | 392 | *NR* | *NR* | *NR* |
| | | *1024* | *1023* | 4.192 | 1.71 | 81269.18 | 534 | 2.407 | 1.02 | 116151.27 | 321 | 368.45 | 1.38 | 124501.55 | 433 | 1.310 | 117625.16 | 312 |
| | | | *3063* | 1.833 | *NR* | 144784.55 | 732 | 2.053 | *NR* | 250635.65 | 517 | 115.00 | *NR* | 332645.54 | 634 | *NR* | *NR* | *NR* |
| | | | *8128* | 2.145 | *NR* | 157996.72 | 904 | 3.2084 | *NR* | 423542.24 | 701 | 58.1213 | *NR* | 936801.81 | 791 | *NR* | *NR* | *NR* |
| Watts-Strogatz | $k_L = 2$<br>$k_L = 4$<br>$k_L = 7$ | *64* | *99* | 0.0243 | *NR* | 64.0 | 64 | 0.0283 | *NR* | 2821.53 | 35 | 0.4967 | *NR* | 2514.96 | 41 | *NR* | *NR* | *NR* |
| | | | *194* | 0.0244 | *NR* | 64.0 | 64 | 0.0238 | *NR* | 1442.30 | 47 | 0.3114 | *NR* | 3196.69 | 48 | *NR* | *NR* | *NR* |
| | | | *287* | 0.0236 | *NR* | 64.0 | 64 | 0.0271 | *NR* | 1570.44 | 51 | 0.312 | *NR* | 3618.70 | 54 | *NR* | *NR* | *NR* |
| | | *128* | *191* | 0.0564 | *NR* | 128.0 | 128 | 0.0504 | *NR* | 5147.98 | 72 | 1.399 | *NR* | 6212.21 | 81 | *NR* | *NR* | *NR* |
| | | | *389* | 0.0436 | *NR* | 128.0 | 128 | 0.048 | *NR* | 3495.86 | 94 | 0.9190 | *NR* | 7416.55 | 99 | *NR* | *NR* | *NR* |
| | | | *580* | 0.0476 | *NR* | 128.0 | 128 | 0.052 | *NR* | 3104.33 | 102 | 0.8005 | *NR* | 8444.13 | 108 | *NR* | *NR* | *NR* |
| | | *256* | *400* | 0.1614 | *NR* | 256.0 | 256 | 0.162 | *NR* | 11187.54 | 154 | 5.2896 | *NR* | 14206.98 | 165 | *NR* | *NR* | *NR* |
| | | | *764* | 0.1164 | *NR* | 256.0 | 256 | 0.1259 | *NR* | 8555.45 | 188 | 3.0085 | *NR* | 19147.82 | 198 | *NR* | *NR* | *NR* |
| | | | *1144* | 0.0973 | *NR* | 256.0 | 256 | 0.1267 | *NR* | 9418.7 | 206 | 2.4849 | *NR* | 31716.450 | 209 | *NR* | *NR* | *NR* |
| | | *512* | *763* | 0.5105 | *NR* | 512.0 | 512 | 0.5051 | *NR* | 27571.52 | 290 | 25.899 | *NR* | 35845.13 | 330 | *NR* | *NR* | *NR* |
| | | | *1529* | 0.434 | *NR* | 512.0 | 512 | 0.4891 | *NR* | 35829.82 | 368 | 10.986 | *NR* | 79931.84 | 389 | *NR* | *NR* | *NR* |
| | | | *2303* | 0.4402 | *NR* | 512.0 | 512 | 0.5464 | *NR* | 38984.10 | 408 | 8.841 | *NR* | 124304.65 | 416 | *NR* | *NR* | *NR* |
| | | *1024* | *1528* | 2.4698 | *NR* | 1024.0 | 1024 | 2.368 | *NR* | 109849.12 | 580 | 110.31 | *NR* | 149304.38 | 659 | *NR* | *NR* | *NR* |
| | | | *3072* | 1.848 | *NR* | 1024.0 | 1024 | 2.1536 | *NR* | 140270.02 | 742 | 58.156 | *NR* | 316953.56 | 789 | *NR* | *NR* | *NR* |
| | | | *4635* | 1.9649 | *NR* | 1024.0 | 1024 | 2.973 | *NR* | 155346.43 | 817 | 38.55 | *NR* | 530868.49 | 840 | *NR* | *NR* | *NR* |
| Erdo-Renyi | p=0.2<br>p=0.5<br>p=0.7 | *64* | *394* | 0.0216 | *NR* | 307.76 | 62 | 0.0288 | *NR* | 1708.70 | 48 | 0.3792 | *NR* | 3498.15 | 55 | *NR* | *NR* | *NR* |
| | | | *1003* | 0.0276 | *NR* | 385.08 | 62 | 0.0363 | *NR* | 1027.26 | 58 | 0.7371 | *NR* | 7651.58 | 59 | *NR* | *NR* | *NR* |
| | | | *1417* | 0.0128 | *NR* | 518.44 | 62 | 0.034 | *NR* | 1200.11 | 59 | 1.462 | *NR* | 10101.84 | 60 | *NR* | *NR* | *NR* |
| | | *128* | *1612* | 0.0379 | *NR* | 1162.52 | 124 | 0.0736 | *NR* | 5041.98 | 109 | 1.116 | *NR* | 23480.76 | 114 | *NR* | *NR* | *NR* |
| | | | *4054* | 0.0699 | *NR* | 1431.88 | 126 | 0.1522 | *NR* | 5343.53 | 120 | 2.3257 | *NR* | 68028.0 | 122 | *NR* | *NR* | *NR* |
| | | | *5712* | 0.076 | *NR* | 1965.96 | 126 | 0.190 | *NR* | 4722.90 | 123 | 3.512 | *NR* | 95802.12 | 124 | *NR* | *NR* | *NR* |
| | | *256* | *6522* | 0.1936 | *NR* | 6552.64 | 250 | 0.3496 | *NR* | 23343.71 | 234 | 3.635 | *NR* | 229235.27 | 239 | *NR* | *NR* | *NR* |
| | | | *16331* | 0.3777 | *NR* | 5514.58 | 254 | 1.018 | *NR* | 23919.62 | 247 | 9.323 | *NR* | 618366.53 | 248 | *NR* | *NR* | *NR* |
| | | | *22951* | 0.4973 | *NR* | 256.0 | 256 | 1.34 | *NR* | 22429.10 | 250 | 13.38 | *NR* | 868933.56 | 252 | *NR* | *NR* | *NR* |
| | | *512* | *26207* | 1.3412 | *NR* | 8951.86 | 510 | 2.479 | *NR* | 110230.25 | 486 | 16.01 | *NR* | 1942119.08 | 490 | *NR* | *NR* | *NR* |
| | | | *65574* | 2.867 | *NR* | 21632.86 | 510 | 7.719 | *NR* | 106116.31 | 502 | 41.71 | *NR* | 5165041.22 | 501 | *NR* | *NR* | *NR* |
| | | | *91762* | 3.526 | *NR* | 30068.60 | 510 | 14.17 | *NR* | 103960.13 | 505 | 56.11 | *NR* | 7271924.85 | 507 | *NR* | *NR* | *NR* |
| | | *1024* | *104927* | 9.074 | *NR* | 68618.70 | 1020 | 22.3394 | *NR* | 440389.58 | 998 | 56.66 | *NR* | 16359885.61 | 999 | *NR* | *NR* | *NR* |
| | | | *262255* | 26.75 | *NR* | 85500.24 | 1022 | 58.05 | *NR* | 381167.11 | 1015 | 171.46 | *NR* | 41944465.43 | 1015 | *NR* | *NR* | *NR* |
| | | | *366916* | 31.10 | *NR* | 1024.0 | 1024 | 82.47 | *NR* | 355593.74 | 1018 | 280.11 | *NR* | 59156072.22 | 1018 | *NR* | *NR* | *NR* |
| Binomial Tree | | *64* | *63* | 0.0227 | 2 | 64.0 | 64 | 0.0204 | 1 | 3217.71 | 32 | 0.585 | 1 | 3623.29 | 32 | 0.0317 | 3371.40 | 32 |
| | | *128* | *127* | 0.0438 | 2 | 128.0 | 128 | 0.0412 | 1 | 6510.32 | 64 | 2.1467 | 1 | 7171.00 | 64 | 0.0742 | 6510.04 | 64 |
| | | *256* | *256* | 0.0106 | 2 | 256.0 | 256 | 0.102 | 1 | 12099.31 | 128 | 8.377 | 1 | 14766.39 | 128 | 0.1210 | 12099.31 | 128 |
| | | *512* | *511* | 0.406 | 2 | 512.0 | 512 | 0.441 | 1 | 27427.64 | 256 | 36.002 | 1 | 34354.79 | 256 | 0.505 | 27427.64 | 256 |
| | | *1024* | *1023* | 1.87 | 2 | 1024.0 | 1024 | 1.836 | 1 | 84872.029 | 512 | 147.58 | 1 | 108433.52 | 512 | 1.32 | 84872.03 | 512 |
| Star | | *64* | *63* | 0.0207 | 2 | 5630.31 | 2 | 0.019 | 1 | 5752.19 | 1 | 1.5824 | 1 | 5752.19 | 1 | 0.0259 | 5752.19 | 1 |
| | | *128* | 127 | 0.0342 | 2 | 5630.31 | 2 | 0.03680 | 1 | 12639.10 | 1 | 7.015 | 1 | 12639.10 | 1 | 0.0677 | 12639.10 | 1 |
| | | *256* | 255 | 0.0913 | 2 | 24003.90 | 2 | 0.0900 | 1 | 24125.78 | 1 | 22.81 | 1 | 24125.78 | 1 | 0.1094 | 24125.78 | 1 |
| | | *512* | 511 | 0.3236 | 2 | 53895.35 | 2 | 0.3381 | 1 | 54017.23 | 1 | 107.36 | 1 | 54017.23 | 1 | 0.2276 | 54017.23 | 1 |
| | | *1024* | *1023* | 1.312 | 2 | 168392.52 | 2 | 1.4089 | 1 | 168556.29 | 1 | 443.81 | 1 | 168556.29 | 1 | 0.6835 | 168556.29 | 1 |

Table 5: Detailed results for the various combinations of $r$ and $h$ in Balanced Tree

| | | | Approximation | | | | Greedy | | | | Genetic | | | | ILP | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | V | E | ExT | AR | CF | VCS | ExT | AR | CF | VCS | ExT | AR | CF | VCS | ExT | CF | VCS |
| | *3* | *2* | 0.0001 | 2 | 2.16 | 2 | 0.0001 | 1 | 124.04 | 1 | 0.0176 | 1 | 124.04 | 1 | 0.0123 | 124.04 | 1 |
| | *7* | *6* | 0.0001 | 2 | 250.08 | 4 | 0.0002 | 1 | 371.28 | 2 | 0.0338 | 1.5 | 249.40 | 3 | 0.0109 | 371.28 | 2 |
| | *15* | *14* | 0.0003 | 2 | 379.86 | 10 | 0.0008 | 1.2 | 745.11 | 6 | 0.094 | 1.6 | 624.23 | 8 | 0.0143 | 745.73 | 5 |
| | *31* | *30* | 0.0012 | 2 | 886.69 | 20 | 0.0015 | 1 | 1746.77 | 10 | 0.3064 | 1.6 | 1506.15 | 16 | 0.0271 | 1745.06 | 10 |
| $BT - r_2$ | *63* | *62* | 0.0221 | 2.1 | 1903.46 | 42 | 0.0223 | 1.3 | 3269.75 | 26 | 0.9414 | 1.8 | 3400.336 | 36 | 0.0859 | 3770.035 | 20 |
| | *127* | *126* | 0.0387 | *2.2* | 4433.25 | 84 | 0.0461 | 1.5 | 8583.57 | 42 | 3.997 | 1.8 | 8012.42 | 69 | 1.309 | 8527.22 | 38 |
| | *255* | *254* | 0.0977 | *2.09* | 7342.22 | 170 | 0.113 | 1.3 | 13074.54 | 106 | 16.49 | 1.7 | 15039.79 | 138 | 1.443 | 15417.41 | 81 |
| | *511* | *510* | 0.444 | *$NR* | 17542.49 | 340 | 0.4588 | *$NR* | 36131.69 | 170 | 76.28 | *$NR* | 35542.64 | 285 | *NR* | *NR* | *NR* |
| | *1023* | *1022* | 1.99 | *$NR* | 56812.17 | 682 | 1.96 | *$NR* | 98694.95 | 426 | 310.96 | *$NR* | 114932.21 | 532 | *NR* | *NR* | *NR* |
| | *4* | *3* | 0.0001 | 2 | 125.04 | 2 | 0.0001 | 1 | 246.92 | 1 | 0.0288 | 1 | 246.92 | 1 | 0.0494 | 246.92 | 1 |
| | *13* | *12* | 0.0002 | 2 | 500.32 | 6 | 0.0005 | 1 | 865.96 | 3 | 0.1165 | 1.3 | 744.08 | 4 | 0.0345 | 865.96 | 3 |
| | *40* | *39* | 0.0023 | 2 | 1876.69 | 20 | 0.0037 | 1.2 | 2492.73 | 12 | 0.7098 | 2.7 | 2389.69 | 27 | 0.0744 | 2501.24 | 10 |
| $BT - r_3$ | 121 | 120 | 0.0370 | 2.14 | 6420.473 | 60 | 0.0422 | 1.07 | 9074.63 | 30 | 6.3380 | 2.4 | 8743.99 | 68 | 0.1327 | 9100.39 | 28 |
| Balanced Tree | 364 | 363 | 0.2024 | 2.16 | 17648.53 | 182 | 0.2686 | 1.3 | 24555.07 | 111 | 51.4963 | 2.35 | 26776.00 | 198 | 0.7695 | 26934.29 | 84 |
| | 1093 | 1092 | 1.88 | *$NR* | 96751.76 | 546 | 2.59 | *$NR* | 144493.70 | 273 | 634.18 | *$NR* | 146385.61 | 582 | *$NR* | *$NR* | *$NR* |
| | 3280 | 3279 | 19.10 | *$NR* | 867756.66 | 1640 | 17.73 | *$NR* | 1204059.16 | 1002 | 6402.57 | *$NR* | 1322459.91 | 1632 | *$NR* | *$NR* | *$NR* |
| | *6* | *5* | 0.0001 | 2 | 370.80 | 2 | 0.0001 | 1 | 492.68 | 1 | 0.0454 | 1 | 492.68 | 1 | 0.0543 | 492.68 | 1 |
| | *31* | *30* | 0.0017 | 2 | 1747.77 | 10 | 0.0015 | 1 | 2235.79 | 5 | 0.5683 | 2 | 2117.91 | 10 | 0.051 | 2117.12 | 5 |
| $BT - r_5$ | *156* | *155* | 0.057 | 2 | 9918.3 | 52 | 0.0548 | 1.15 | 11833.80 | 30 | 13.21 | 3.03 | 12028.23 | 79 | 0.2183 | 11922.99 | 26 |
| | *781* | *780* | 1.22 | 2 | 65712.08 | 260 | 0.9831 | 1 | 81913.69 | 130 | 438.62 | 2.85 | 81903.43 | 371 | 1.2135 | 81913.69 | 130 |
| | *3906* | *3905* | 28.66 | *$NR* | 1639072.88 | 1302 | 29.08 | *$NR* | 1982558.40 | 755 | 9756.42 | *$NR* | 2069177.82 | 1838 | *$NR* | *$NR* | *$NR* |
| | *3* | *2* | 0.0001 | 2 | 2.16 | 2 | 0.0001 | 1 | 124.04 | 1 | 0.0291 | 1 | 124.04 | 1 | 0.0215 | 124.04 | 1 |
| | *7* | *6* | 0.0003 | 2 | 250.08 | 4 | 0.0001 | 1 | 371.28 | 2 | 0.0588 | 1.5 | 249.4 | 3 | 0.0215 | 371.28 | 2 |
| | *21* | *20* | 0.0016 | 2 | 999.63 | 8 | 0.0021 | 1 | 1365.94 | 4 | 0.28 | 2.75 | 1250.06 | 11 | 0.053 | 1365.94 | 4 |
| $BT - h_2$ | *73* | *72* | 0.0297 | 2 | 5267.55 | 16 | 0.0321 | 1 | 6010.91 | 8 | 3.73 | 3.75 | 5910.03 | 30 | 0.0797 | 5899.04 | 8 |
| | *273* | *272* | 0.1307 | 2 | 23233.58 | 32 | 0.1701 | 1 | 24589.89 | 16 | 48.68 | 5.75 | 24543.01 | 92 | 0.188 | 24509.76 | 16 |
| | *1057* | *1056* | 1.894 | 2 | 168954.13 | 64 | 2.044 | 1 | 174364.72 | 32 | 849.31 | 9.75 | 174644.72 | 312 | 1.185 | 174364.72 | 32 |
| | *5* | *4* | 0.0001 | 2 | 4.16 | 4 | 0.0002 | 1 | 247.92 | 2 | 0.0441 | 1.5 | 126.04 | 3 | 0.0286 | 247.92 | 2 |
| $BT - h_4$ | *31* | *30* | 0.0046 | 2 | 886.69 | 20 | 0.0044 | 1 | 1746.77 | 10 | 0.488 | 1.6 | 1506.15 | 16 | 0.0647 | 1745.06 | 10 |
| | *341* | *340* | 0.272 | 2 | 19748.00 | 136 | 0.2904 | 1 | 26427.00 | 68 | 75.13 | 2.58 | 26083.26 | 172 | 0.6814 | 25981.85 | 68 |

290–297.
[46] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Reviews of modern physics 74 (1) (2002) 47.
[47] I. J. Farkas, I. Derényi, A.-L. Barabási, T. Vicsek, Spectra of" real-world" graphs: Beyond the semicircle law, in: The Structure and Dynamics of Networks, Princeton University Press, 2011, pp. 372–383.
[48] D. J. Watts, S. H. Strogatz, Collective dynamics of 'small-world'networks, nature 393 (6684) (1998) 440–442.
[49] B. J. Prettejohn, M. J. Berryman, M. D. McDonnell, Methods for generating complex networks with selected structural properties for simulations: a review and tutorial for neuroscientists, Frontiers in computational neuroscience 5 (2011) 11.
[50] M. E. Newman, D. Walls, M. Newman, A.-L. Barabási, D. J. Watts, Scaling and percolation in the small-world network model, in: The Structure and Dynamics of Networks, Princeton University Press, 2011, pp. 310–320.
[51] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, science 286 (5439) (1999) 509–512.
[52] A.-L. Barabási, R. Albert, H. Jeong, Mean-field theory for scale-free random networks, Physica A: Statistical Mechanics and its Applications 272 (1-2) (1999) 173–187.
[53] F. L. Piccolo, G. Neglia, The effect of heterogeneous link capacities in bittorrent-like file sharing systems, in: 2004 International Workshop on Hot Topics in Peer-to-Peer Systems, IEEE, 2004, pp. 40–47.
[54] D. Bertsekas, R. Gallager, Data networks, Athena Scientific, 2021.
[55] S. Keshav, S. Kesahv, An engineering approach to computer networking: ATM networks, the Internet, and the telephone network, Vol. 1, Addison-Wesley Reading, 1997.
[56] A. A. Javadi, R. Farmani, T. P. Tan, A hybrid intelligent genetic algorithm, Advanced Engineering Informatics 19 (4) (2005) 255–262.
[57] B. Charyyev, E. Arslan, M. H. Gunes, Latency Comparison of Cloud Datacenters and Edge Servers, 2020 IEEE Global Communications Conference, IEEE, 2020, pp. 1-6.
[58] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, X. Liu, From Cloud to Edge: a First Look at Public Edge Platforms, Proc. of the 21st ACM Internet Measurement Conference, ACM, 2021, pp. 37–53.