Heuristic Edge Server Placement in Industrial Internet of Things and Cellular Networks

Shahrukh Khan Kasi, Mumraiz K. Kasi, Kamran Ali, Mohsin Raza,
Hifza Afzal, Aboubaker Lasebae, Bushra Naeem, Saif ul Islam, Joel J. P. C. Rodrigues

***Abstract*—Rapid developments in industry 4.0, machine learning, and digital twins have introduced new latency, reliability, and processing restrictions in Industrial Internet of Things (IIoT) and mobile devices. However, using current Information and Communications Technology (ICT), it is difficult to optimally provide services that require high computing power and low latency. To meet these requirements, mobile edge computing is emerging as a ubiquitous computing paradigm that enables the use of network infrastructure components such as cluster-heads/sink nodes in IIoT and cellular network base stations to provide local data storage and computation servers at the edge of the network. However, optimal location selection for edge servers within a network out of a very large number of possibilities, such as to balance workload and minimize access delay, is a challenging problem. In this paper, the edge server placement problem is addressed within an existing network infrastructure obtained from Shanghai Telecom's base station dataset that includes a significant amount of call data records and locations of actual base stations. The problem of edge server placement is formulated as a multi-objective constraint optimization problem that places edge servers strategically to balance between the workloads of edge servers and reduce access delay between the industrial control center/cellular base-stations and edge servers. To search randomly through a large number of possible solutions and selecting those that are most descriptive of optimal solution can be a very time-consuming process, therefore, we apply the genetic algorithm and local search algorithms (hill-climbing and simulated annealing) to find the best solution in the least number of solution space explorations. Experimental results are obtained to compare the performance of the genetic algorithm against the above-mentioned local search algorithms. The results show that the genetic algorithm can quickly search through the large solution space as compared to local search optimization algorithms to find an edge placement strategy that minimizes the cost function.**

***Index Terms*—Industrial Internet of Things (IIoT), Mobile edge computing, Edge server placement, Genetic Search, Data Mining.**

I. Introduction

THE previous decade has witnessed substantial growth in the Internet of Things (IoT) and mobile devices. A steep rise in the number of IoT and cellular devices is expected to continue in the future [1]. With mobile and IoT devices permeating in every field of life, services such as robotics, automation, assembly and production, machine intelligence, and virtual reality are becoming more and more widespread. These services, whether targeting IIoT in industry 4.0 [2]-[3] or infotainment and emergency services in cellular networks [4], [5]-[7] require high computing power and are sensitive to delays in communication networks [8].

IIoT promises to provide an agile and intelligent manufacturing process by proactively monitoring the state of the network using an enormous amount of data collected from information technologies and advanced sensors. The big data generated in a manufacturing process can be utilized using data mining techniques to predict and self-heal the outages, optimize the production process, and increase the lifetime of devices. However, due to the limited processing power and battery lifetimes of IoT and mobile devices, services such as intelligent processes (feed-back control system, predictive analysis via data mining or machine/deep learning models) and delay-sensitive applications (emergency systems) demand a change in current IoT and mobile technology architecture. To meet these requirements, cloud computing, a widely used computing paradigm, is used to deliver such services to mobile and IoT devices [9], [10], [11].

Cloud-centric architecture has surfaced as the predominant model of cloud computing in IoT. In the cloud-centric architecture, cloud platforms installed at the top layer provide virtually unlimited data storage and computational capacity. In the context of IoT, a large number of devices connected to a remote centralized cloud may introduce delay in the network due to the remote location of the cloud and processing of a large number of requests from IoT devices [12], [13].

To overcome the problem of large delays in cloud-centric computing, various network architecture variations and methods are proposed [14]. The key idea is to bring processing closer to the IoT devices, thus introducing distributed control systems. Edge computing provides a scalable solution to address the issues introduced by cloud-centric architecture by allowing data mining and processing at the edge of the network. Edge computing is implemented at the network infrastructure between the cloud and IoT devices as a computational layer where the virtualized application resources are leveraged.

Several edge architecture variations are proposed in literature including Cloudlet, Fog computing and Multi-access Edge Computing (MEC). In all of the above-discussed solutions, edge servers are placed near the edge of the network, providing a mid-tier between network devices and cloud. Edge servers, with high computation and data storage capabilities, are placed near the end devices to offer low latency and high throughput. However, there are several practical complexities in implementing edge solutions. Due to budget limitation and hardware requirements for edge servers, only a limited number of edge servers can be placed in a network. To optimally place

Shahrukh Khan Kasi is with the Department of Electrical and Computer Engineering Michigan State University, USA

Mumraiz K. Kasi, Hifza Afzal and Bushra Naeem are with the Faculty of Information and Communication Technology, BUITEMS, Quetta, Pakistan

Kamran Ali is with the Faculty of Science and Technology, Middlesex University London, UK

Mohsin Raza is with the Department of Computer Science, Edge Hill University, UK

Saif ul Islam is with the Department of Computer Science, Institute of Space Technology, Islamabad, Pakistan. e-mail: saiflu2004@gmail.com.

Joel J. P. C. Rodrigues is with the Federal University of Piauí (UFPI), Teresina - PI, Brazil and the Instituto de Telecomunicações, Portugal

edge servers in a network, out of a very large number of possible placement strategies, is a challenging problem.

A pragmatic solution is to reuse the existing network infrastructure as edge servers (collocating edge servers with mobile network base stations or IoT cluster-heads or access points) [15]. Collocating edge servers with already existing network infrastructure reduces the number of possible edge server placement strategies. Moreover, to maintain low latency and high throughput, it is imperative to take user traffic patterns, the resulting edge server's workload, and access delay into consideration. Therefore, requiring for an edge placement solution to search through a large number of possible solutions and selecting those that are most descriptive of optimal solution which can be a very time-consuming process. Therefore, we apply the genetic algorithm and local search algorithms (hill-climbing and simulated annealing) to find the best solution in the least number of solution space explorations.

In edge computing, proximity to the edge server is desired for edge devices. It leads to reduced delay in the network. The resulting deployment of edge servers should also be performed in a way that the workload in the system is balanced. To the best of our knowledge, this is the first study that has used genetic algorithm and local search optimization techniques to provide a solution for the problem of edge server placement. The main contributions of the research are:

- The problem of edge server placement is addressed using existing network infrastructure. The low latency and workload balancing requirements in edge server placement strategies are formulated as a multi-objective constraint optimization problem.
- Genetic programming and local optimization algorithms (hill-climbing and simulated annealing) are applied to find the best solution. Experimental results are obtained using Shanghai Telecom's base station dataset to compare the performance of these optimization techniques.

The rest of the paper is organized as follows. We review the related work in Section II. The edge server placement problem is presented in Section III followed by the proposed genetic algorithm and local search optimization algorithms discussion in Section IV. The performance comparison of algorithms are discussed in Section V. Finally, the concluding remarks are provided in Section VI.

II. Related Work

Previous studies on the placement of edge servers have mostly focused on finding the candidate location for edge servers as cluster heads using clustering algorithms. In some of these works, k-means clustering was used along with mixed-integer quadratic programming [16], multi-objective constraint optimization problem [17], and mixed-integer linear programming [18].

In [19], the authors presented an edge provisioning algorithm that finds the ideal edge locations in physical networks. However, the authors have not discussed the problem of workload balancing between the edge servers that may lead to a higher workload for some edge servers while others remain unused. In [16], the authors use k-means algorithm in conjunction with mixed-integer quadratic programming to provide a solution to the edge placement problem. The authors proposed an approximate solution to the mixed-integer quadratic programming problem due to its complexity. However, the authors have stated that the proposed algorithm is not very efficient in terms of computational complexity.

The authors in [20] proposed a cloud assignment problem to optimally place the cloudlets in a wireless network. A multi-user and multi-cloudlet system were formulated using a queuing network followed by the assignment of cloudlets. Although their implementation is effective, the authors only consider workload balancing in their objective function.

Recently, in [21] the authors utilizing data mining techniques have proposed a non-dominated sorting genetic algorithm III (NSGA-III) to provide vehicular social media services with low latency and high reliability using cloud computing. The authors have focused on the placement of edge servers without considering the problem of finding optimal association between the connected base stations and edge servers.

In [17], authors have investigated the edge server placement problem in mobile edge computing environments for smart cities. They have formulated the edge server placement problem as a multi-objective constraint optimization problem that aims to balance the workload among edge servers and minimize the access delay between the mobile user and edge server. They have used mixed integer programming to find the optimal solution. Our proposed work extends the problem presented in [17] to investigate the working of genetic algorithm and local search optimization techniques in the context of edge server placement.

Search optimization techniques can be categorized into two main classes: 1) general-purpose heuristic optimization algorithms which are independent of the optimization problem, 2) heuristic approaches that are specifically designed for a mapping problem. Due to the limited applicability of the specifically designed algorithms and lack of interoperability of available solutions in diverse fields (healthcare, IIoT, cellular networks, vehicular networks, etc.), the proposed work focuses on the first class of heuristic algorithms due to their generalized applicability. Genetic algorithm, hill-climbing, and simulated annealing are three such widely used optimization techniques [22].

Genetic algorithms are stochastic search techniques that are inspired by the adaption in evolving natural systems. Genetic search algorithms' performance is as good as global search techniques, however, their convergence to the global optimum may take a longer time [22]. Hill climbing algorithm finds the global optimal solution only in convex space. However, most real-life problems are not convex. Simulated annealing offers a mechanism by which the major drawback in the hill-climbing algorithm is fixed by allowing the search space to include some bad solutions initially to make sure that the algorithm doesn't stick at a bad local optimum. Therefore, we propose a genetic algorithm for the edge server placement problem and compare its performance with hill-climbing and simulated annealing optimization techniques.

III. Computational Problem

The edge server placement problem can be described in the form of an undirected graph network $G = (V, E)$, where V represents the locations of B base stations (or cluster-heads in IIoT) and E is the weight of edge connection between base stations. Given the network G, a set of S edge servers are to be placed in B potential locations, that is, an edge server is restricted to be deployed with an already installed base

station/cluster-head to limit the search space for edge server placement. Since there is a one to one communication link between edge servers and base stations (as shown in Fig. 1), the access delay is defined in terms of euclidean distance ($\mathbf{d_b}$). Each base station/cluster-head processes a number of call/flow requests from a set of mobile users which is defined as the workload of a base station (t_b).

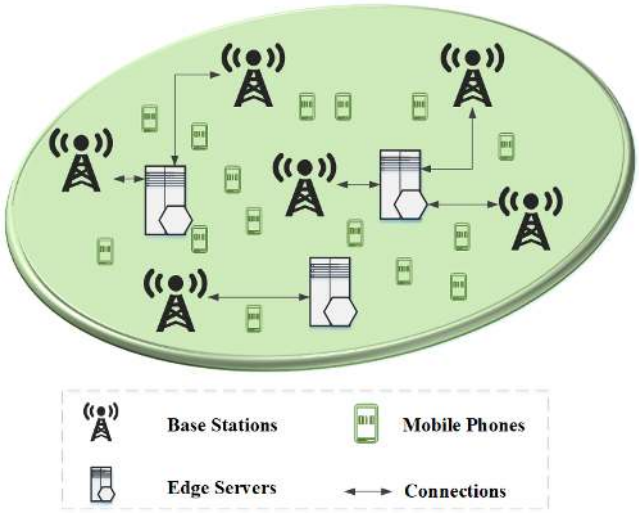


Fig. 1: Mobile edge computing system model.

The proposed work aims to provide optimized solutions for a given set of S edge servers to be placed in B possible locations. Edge servers are connected to a set of base stations such that: (i) the communication latency between base stations and edge servers is reduced, and (ii) workload is balanced between deployed edge servers. The constraints imposed on this optimization problem are:

- A base station can have only one connection with any of the edge servers.
- An edge server is collocated with a base station processing all mobile call/flow requests from the base station.

For a particular edge server placement strategy (ℓ), the workload of an edge server is defined as the sum of user requests offloaded from a set of base stations that are connected to s^{th} edge server, that is, $T_s(\ell) = \sum_b t_b$, where t_b is the workload of b^{th} base station. Similarly, delay of an edge server is defined as the sum of distances from a set of base stations that are connected to s^{th} edge server, that is, $D(\ell) = \sum_b \mathbf{d_b}$, where $\mathbf{d_b}$ is the euclidean distance matrix of b^{th} base station with the connected edge server. The workload balancing of edge servers ensures that no such situation arises where some of the edge servers are overloaded while others are underloaded. The workload balancing ($W(\ell)$) for a particular edge server placement strategy is defined as the standard deviation of the workload of each edge server in the mobile edge computing network, that is,

$$W(\ell) = std(T_j, T_k) \quad \forall j, k \in S$$

Therefore, the multi-objective cost of an edge server placement strategy is defined as:

$$cost(\ell) = \beta \times T_{normalized}(\ell) + (1 - \beta) \times D_{normalized}(\ell)$$

where $\beta \in \{0, 1\}$ is used to give weight to $T_{normalized}(\ell)$ in the cost function over $D_{normalized}(\ell)$

The optimization problem can be defined as,

1) find an edge server placement strategy (ℓ) such that $D(\ell)$ is minimized and,
2) for a given edge server placement scheme (ℓ), find the edge connections (x_{bs}, $\forall b \in B, s \in S$) for b^{th} base station to s^{th} edge server, such that the workload between edge servers is balanced.

Mathematically,

$$min\ cost(\ell),\ \forall\ \ell\ \in B$$

such that,

$$\sum_{s=1}^{S} x_{bs} = 1 \tag{1}$$

$$x_{b,s} \in \{0, 1\} \tag{2}$$

where constraint (1) and (2) ensures that there exists only one connection between a base station to an edge server.

The edge server placement problem in a mobile edge computing network is NP-hard [17], therefore we propose to find edge servers optimal locations using genetic algorithm and local search optimization techniques such as hill-climbing and simulated annealing. In [23], the authors have shown that network traffic follows periodic temporal and spatial statistical distribution law. This entails that the load statistical distribution at a certain duration of time window would remain the same. Therefore, the load statistical distribution can be utilized to choose the appropriate intervals at which the edge server placement is reconfigured based on the traffic patterns which is an interesting problem but beyond the scope of this work.

IV. Algorithmic Implementation

Genetic search algorithms are based on the principle of natural selection and genetics in which the decision variable of a search problem is encoded into a string of alphabets of a finite-length. These strings are referred to as chromosomes. A population can be attributed as the collection of chromosomes formed from different combinations of chromosomes. The population size, a predefined parameter, is an important factor that affects the scalability and performance of genetic search algorithms. A small population size may lead to suboptimal solution whereas a large population size will affect the convergence rate of the algorithm.

To evolve good solutions, a measure to distinguish between good and bad solutions is required. In the context of a genetic search algorithm, a fitness function is used to measure the relative fitness of the candidate solution. The standard genetic algorithm includes the generation of a population of random chromosomes followed by the assignment of fitness value to each of these chromosomes. The chromosomes that have a high fitness score are allowed to mate with each other and reproduce the children's population for the next generation. The process of mating is called reproduction and the genetic operations involved during reproduction are breeding and mutation.

Breeding is the process of cutting two chromosomes at a random point and then combining the two portions of different chromosomes whereas mutation is the process of flipping one or more variables in the chromosome randomly. In the proposed work, a chromosome is a combination of (i) the edge server locations ($\mathbf{l}$) and (ii) edge server connections ($\mathbf{x}$) to base

Algorithm 1 Genetic Search Algorithm

Input: f_{th}, G
Output: return the best chromosome

1: **x** ← random initial connections of base stations with edge servers
2: **l** ← random initial locations of edge servers
3: **population** = [**x**, **l**]
4: g ← set the generation counter
5: **while** true **do**
6: evaluate the fitness of all chromosomes in **population**
7: **if** fitness of any chromosome is greater than f_{th} **then**
8: append chromosome to **matingPool** and **elitePool**
9: apply breeding/crossover to chromosomes in **matingPool** and append to **children**
10: randomly mutate chromosomes in **children**
11: append **children** and **elitePool** to **population**
12: $g = g + 1$
13: **if** $g = G$ **then** $break$

stations/cluster-heads. The fitness function is made dependent on the cost function.

In the genetic algorithm (algorithm 1), an initial population with a size of p is generated by concatenating the randomly initialized edge server locations (**l**) and edge server connections (**x**). A fitness function is defined to assess the fitness of an individual chromosome in the population. Fitness function is made inversely proportional to the cost function. If the fitness function of a chromosome is greater than f_{th}, then these chromosomes are selected in the **elitePool** and **matingPool** population.

For the rest of the children population, we breed the chromosomes appearing in **matingPool** and append it to **children** population. The process of breeding is followed by mutation in which the chromosomes in **children** population are mutated with a small probability to enable search space exploration. The algorithm chooses the fittest chromosome in the population as the final state that signifies the individual with the highest fitness score. The algorithm stops execution when a predefined variable G number of iterations are reached. The complexity of genetic search algorithm is on the order of $O(G \times p)$.

The hill-climbing algorithm takes the current state as the input, where a state is defined as the concatenation of edge server locations **l** and edge server connections **x**. Through local transformations, the hill-climbing algorithm moves between neighboring states and evaluates the cost of each of the neighboring states. If the gain in the current state's cost and any of the neighboring state's cost is positive only then the algorithm moves to that neighboring state. This process is repeated until there are no better neighboring states.

In the hill-climbing algorithm (algorithm 2), both **l** and **x** are randomly initialized such that each base station has only one connection with an edge server. Random initialization step is followed by the computation of cost for the **current** state. From lines 6 – 8, the algorithm finds the **neighbor** states of the **current** state by flipping few of the elements in **x** and **l**. If the cost of any of the **neighbor** states is less than the cost of **current** state, then choose that state as the **current** state. This process is repeated until no further better **current** state can be obtained when compared with the **neighbor** states.

Algorithm 2 Hill Climbing Search

Input: $flips_x, flips_l, N$
Output: returns a local minima state

1: **x** ← random initial connections of base stations with edge servers
2: **l** ← random initial locations of edge servers
3: **current** = [**x**, **l**]
4: evaluate the cost of **current** state
5: **while** true **do**
6: append N neighbor states of **x** to $\mathbf{neighbor_x}$ by swapping $flips_x$ number of edge server connections
7: append N neighbor states of **l** to $\mathbf{neighbor_l}$ by swapping $flips_l$ number of edge server locations
8: **neighbor** = [$\mathbf{neighbor_x}$, $\mathbf{neighbor_l}$]
9: evaluate the cost of **neighbor** states
10: **if** cost of any **neighbor** state is less than the cost of current state **then**
11: set the **current** state to that neighboring state
12: **else** $break$

Algorithm 3 Simulated Annealing

Input: $flips_x, flips_l, N, maxSteps$
Output: returns a solution state

1: **x** ← random initial connections of base stations with edge servers
2: **l** ← random initial locations of edge servers
3: **current** = [**x**, **l**]
4: evaluate the cost of **current** state
5: s ← set the steps counter
6: **while** true **do**
7: append N neighbor states of **x** to $\mathbf{neighbor_x}$ by swapping $flips_x$ number of edge server connections
8: append N neighbor states of **l** to $\mathbf{neighbor_l}$ by swapping $flips_l$ number of edge server locations
9: **neighbor** = [$\mathbf{neighbor_x}$, $\mathbf{neighbor_l}$]
10: evaluate the cost of **neighbor** states
11: **if** cost of any **neighbor** state is less than the cost of current state **then**
12: set the **current** state to that neighboring state
13: **else**
14: Δcost ← difference of the cost of current and neighboring state
15: set the **current** state to the neighboring state with a probability $e^{\frac{-\Delta \text{cost}}{T}}$
16: $s = s + 1$
17: **if** $s = maxSteps$ **then**
18: $break$

Simulated annealing in principle is similar to hill-climbing with a minor modification. The simulated annealing methodology can be best described by the analogy of a heating system that is heated initially with a high temperature and then is left to cool down over time. Similar to this heating system, a simulated annealing algorithm initially allows more random movements in the neighboring search space and with reduction in temperature over time, the algorithm chooses neighboring states that have a lower or same cost than the current state.

In the simulated annealing algorithm (algorithm 3), the initial process remains similar to the hill-climbing algorithm with the exception that neighbor states are chosen based on

the acceptance probability of the state. If the cost of any of the **neighbor** states is less than the cost of **current** state, then the acceptance probability is 1, otherwise, it is determined by an exponential function with a sensitive parameter T. The algorithm stops execution when a predefined variable $maxSteps$ number of iterations is reached.

V. Results

In this section, the experimental evaluation of performance of genetic algorithm and local optimization algorithms such as hill-climbing and simulated annealing for minimizing cost function (see Eq .1) are discussed. The dataset used for the experimentation is provided by Shanghai Telecom [16][24][25]. The Shanghai Telecom dataset contains more than 7.2 million records of accessing the Internet through 2766 base stations (that have been placed in a geographically diverse manner) from 9481 mobile phones. The data set contains 4.6 million call records and 7.5 million flow records of about 10 thousand mobile users during six successive months. Each call/flow record contains the detailed start time and end time of accessing the base station for each mobile user.

Fig. 2: Distributions of base stations in Shanghai [16][24][25].

These records have been used to quantify the workload of a base station. Shanghai is one of the densely populated cities of the world, making it a perfect dataset to analyze the placement of edge servers in a dense network. Fig. 2 shows the distribution of base stations where each node denotes a base station in Shanghai, China.

This work has been implemented in MATLAB. Before the implementation of proposed algorithms, we pre-process the data in order to obtain delay and workload metrics for each base station. The list of parameters used during the experiments are given in Table 1. Our results answer the following questions:

A. How does the performance (in terms of convergence) of hill-climbing, simulated annealing, and genetic search algorithm compare against each other?
B. How generalizable are the results?
C. Do these local search algorithms with a small size of neighbor search space perform as good as if the size of neighbor search space is increased?
D. What is the impact of using the different number of edge servers on the performance of these local search algorithms?
E. How sensitive is the cost function to different values of β?

TABLE I: Simulation parameters.

Symbol	Parameter Name	Parameter Value
B	Number of base station	2766
S	Number of edge servers	10
$neighbors$	Size of neighbor search space	4
β	Beta	0.5
$Seed$	Seed for random generator	99
$flips_x$	Number of flips in neighboring edge server connections	10
$flips_l$	Number of flips in neighboring edge server locations	2

A. How does the performance (in terms of convergence) of hill-climbing, simulated annealing, and genetic search algorithm compare against each other?

In this experiment, we simulate the performance of local search optimization techniques by setting the simulation parameters as shown in Table 1. The simulations are repeated for a fixed number of iterations for both simulated annealing and genetic search algorithms. However, for the hill-climbing algorithm, if the cost of any of the neighboring states is not smaller than the current state's cost then the algorithm execution is halted, and the algorithm returns the current state as the final state.

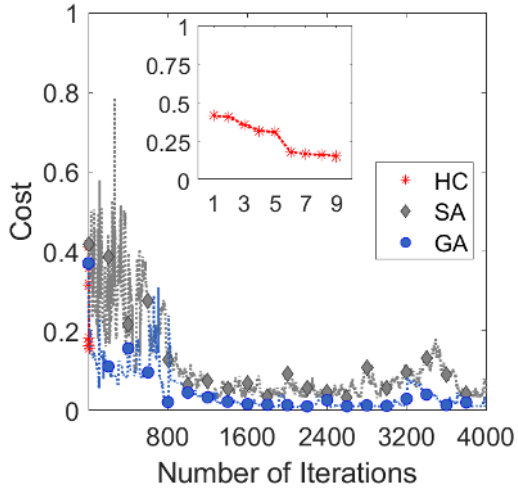
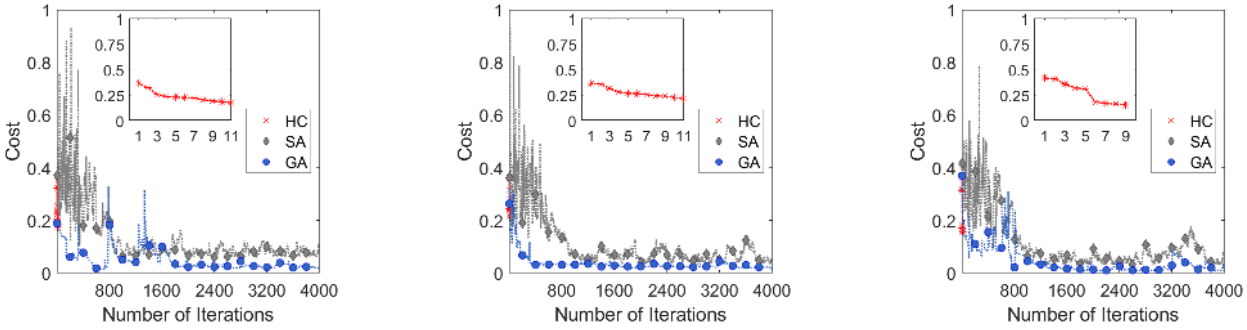


Fig. 3: Performance comparison of hill-climbing, simulated annealing and genetic search algorithms.

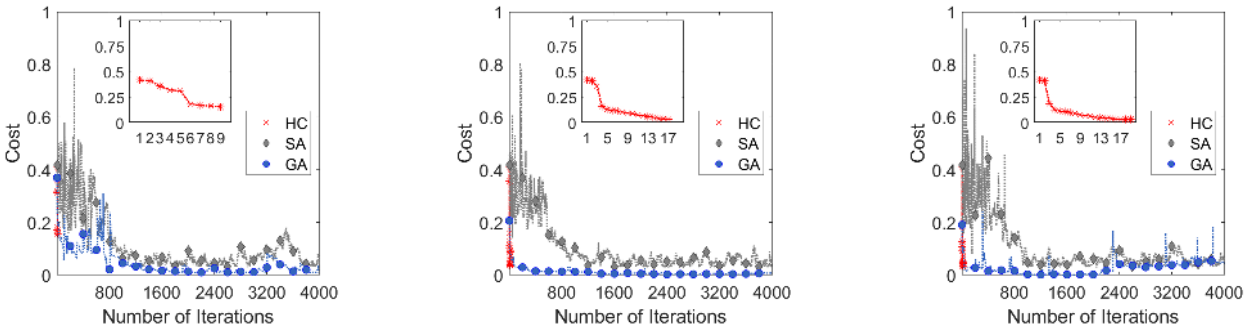
In Fig. 3, the cost values of the current state's returned by local search optimization algorithms across the number of iterations can be seen. Ideally, we would want the cost to be minimized optimally in the first iteration. However, doing so would mean having the global knowledge of the optimal state which is not practical and is extremely computationally expensive. Therefore, we anticipate that for local search optimization techniques, the cost and cost variations will reduce with the increasing number of iterations.

In Fig. 3, we show the results when the normalized cost for the initial state is around 0.5. As expected, the cost for all three local optimization techniques is following a negative trend as the algorithm matures in its execution. However, as anticipated and observed from the results shown in Fig. 3-6, the performance of simulated annealing and genetic search algorithms is far superior to hill climbing in terms of the search for a final state that gives the lowest cost. The mediocre performance of the hill-climbing algorithm can be ascribed to



(a) $Seed = 12$ (b) $Seed = 55$ (c) $Seed = 99$

Fig. 4: Performance comparison of hill-climbing, simulated annealing and genetic algorithms with varied seed numbers used for random value generation and initial state selection.



(a) $neighbors = 4$ (b) $neighbors = 8$ (c) $neighbors = 12$

Fig. 5: Performance comparison of hill-climbing, simulated annealing and genetic algorithms for the varying size of neighbor search space.

its stringent requirements of moving to a new state only if its cost is lower than the current state thus not allowing the search to look for better local optima's than the current one.

Both simulated annealing and genetic search algorithm perform fairly well, that is, it reduces the cost to almost zero starting from higher initial state cost. Cost can not be reduced to zero since the cost function is dependent on delay and workload balancing, there will always be non-negative cost value for any state. However, the nearest the obtained cost for the final state is to zero, the better the placement strategy for edge servers. In Fig 3, we can observe that genetic search convergence to its final state is much faster than simulated annealing. Also, the attained cost value using genetic search is lower than the cost value attained using simulated annealing as shown in Fig. 3-6.

The superior performance of genetic search algorithm can be attributed to the fact that it maintains a proper balance between the state exploration and exploitation. Unlike hill-climbing and simulated-annealing optimization techniques, genetic search algorithm maintains a population of best solutions (chromosomes) that through evolutionary techniques are modified to evolve to a better solution.

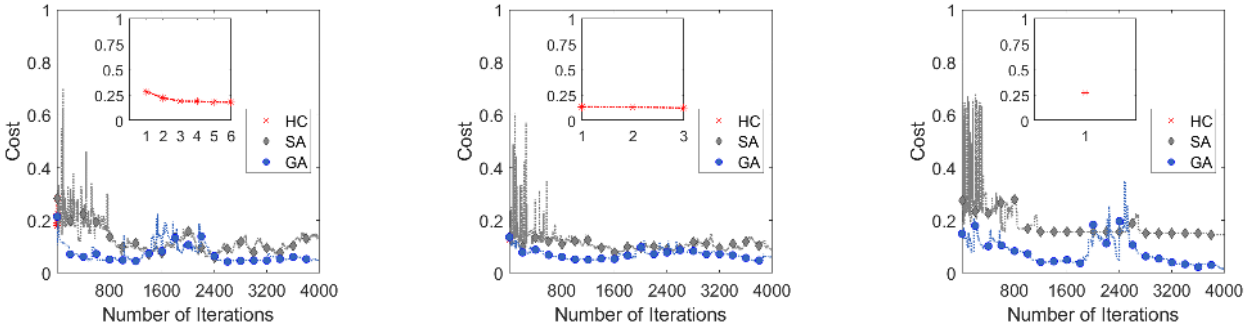
B. How generalizable are the results?

In this experiment, we set the simulation parameters as shown in Table 1. However, we use different seeds for each of the subfigure shown in this section to show the convergence behavior with hill-climbing, simulated annealing, and genetic search algorithms. Ideally, we would expect that for all these experiments the final state returned by these local search optimization algorithms be the same. However, due to the use of different seeds, the initial state is different as well as the values for random parameters are different which gives an entirely different search space to these algorithms. Nonetheless, the work is applicable in different application areas (such as IIoT) and can offer suitable cost minimization.

In Fig. 4, we can observe that for different seed numbers used for random value generation and initial state selection, all these algorithms can reduce the cost. However, similar to what had been observed in Fig 3, the performance of simulated annealing and genetic search transcends the performance of hill-climbing in terms of cost reduction and converging to better suboptimal cost value. Another significant observation from Fig. 4 is the faster convergence of the genetic search algorithm than other algorithms. Not only genetic search algorithm converges faster to a lower-cost state but the attained cost value is also less than the other two algorithms. Therefore, we claim that the performance of hill-climbing, simulated annealing, and genetic search algorithms is generalizable for different initial states and randomness involved in the simulations.

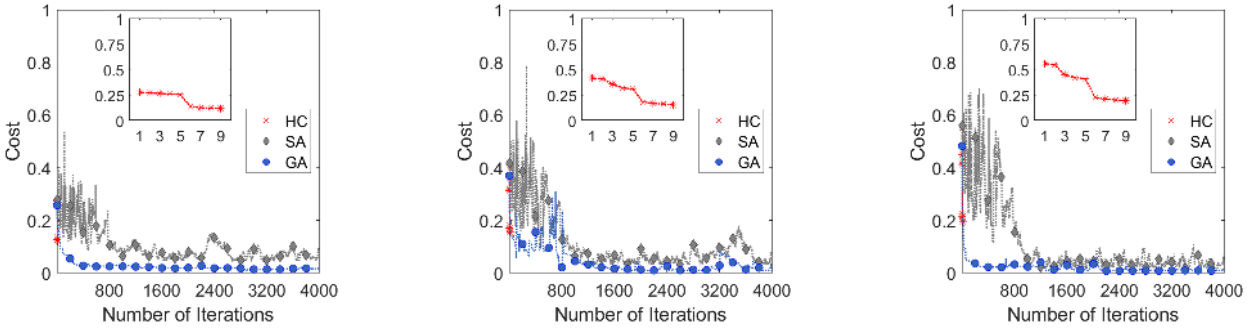
C. Do these local search algorithms with a small size of neighbor search space perform as good as if the size of neighbor search space is increased?

In this experiment we vary the size of the neighbor search



(a) $S = 20$ (b) $S = 50$ (c) $S = 100$

Fig. 6: Performance comparison of hill-climbing, simulated annealing and genetic algorithms for different number of edge servers.



(a) $\beta = 0.30$ (b) $\beta = 0.50$ (c) $\beta = 0.70$

Fig. 7: Performance comparison of hill-climbing, simulated annealing and genetic algorithms for varying β values.

space to show the convergence behavior with hill-climbing, simulated annealing, and genetic search algorithms. Ideally, we would expect that for a large neighbor search space all the algorithms will converge faster since it has now a larger search space to look for the edge placement strategy that reduces the cost. However, using a large neighbor search space is computationally expensive. Therefore, to assess the performance of an algorithm, if the cost reduction with the small neighbors search space is almost similar to the cost reduction with the large neighbors search space then we can claim that the algorithm can reduce the cost to the best suboptimal value irrespective of neighbor search size.

In Fig. 5, we compare the performance of hill-climbing, simulated annealing and genetic algorithms for the different sizes of neighbor search space. It can be observed from the results shown in Fig. 5 that variation in the size of neighbor search space does not majorly affect the performance of simulated annealing and genetic search algorithms, as all variations converge to low-cost final states for a specific algorithm. However, major improvements can be observed in the performance of the hill-climbing search algorithm when the size of the neighbor search space is increased. The final state cost attained with hill-climbing algorithms approaches to the cost attained with simulated annealing and genetic search algorithm as the size of neighbor search space is increased to 8 and beyond. Therefore, we can claim that the edge server placement with simulated annealing and genetic search algorithm provide best results even with the small size of neighbor search space. Whereas, hill-climbing improves the performance in terms of cost reduction albeit with added complexity of a larger size of neighbors search space.

D. What is the impact of using the different number of edge servers on the performance of these local search algorithms?

In Fig. 6, we compare the performance of hill-climbing, simulated annealing and genetic algorithms for the different number of edge servers (S) to be placed in the network. We know that the total delay in the network will increase if the number of edge servers in the network is increased. Therefore, we anticipate that the final state cost attained in the network with a large number of edge servers will be greater than the cost attained in a network with a small number of edge servers.

It can be observed, as anticipated, from the results shown in Fig. 6 that the increase in the number of edge servers affects the final state cost values attained for the network. For a few numbers of edge servers, that is S less than 50, simulated annealing performs almost as good as the genetic algorithm. However, when S is increased beyond 50, the difference in the performance of simulated annealing and genetic algorithms is significant. Another significant observation is the exceptional performance of the genetic search algorithm in comparison to hill-climbing and simulated annealing algorithms. From our observations in Fig 3–6, we can claim that the genetic search algorithm for all the simulation variations we have observed exceeds the performances of hill-climbing and simulated annealing for the edge server placement problem.

E. How sensitive is the cost function to different values of β?

In this experiment, we vary the values of β to show the behavior of optimization algorithms when the weights of delay and workload balance are altered in the cost function. Note that setting $\beta < 0.5$ means that the edge placement strategy is prioritizing the reduction in workload balancing cost whereas $\beta > 0.5$ means that the edge placement strategy is prioritizing the reduction in network access delay.

Ideally, we would expect similar convergence behavior throughout the different values of β which is evident from the results shown in Fig. 7. All algorithms reduce the cost in the edge placement problem and the best performance in terms of cost reduction is achieved by the genetic algorithm followed by the simulated annealing algorithm.

VI. Conclusions

Edge computing is an emerging computing paradigm that facilitates in providing computational and storage resources to mobile and IoT devices. To offload the processing workload from the edge devices such that the workload is balanced and access delay is minimized, the edge server placement problem is formulated as a multi-objective constraint optimization problem. Genetic algorithms and local search algorithms were then used to find an edge server allocation strategy. The experimental evaluation of the proposed work using Shanghai's Telecom dataset proves that the genetic algorithm quickly reaches to a solution state such that it reduces access delay in the network and maintains the workload balance between edge servers with local information. While the evaluation is carried out using Shanghai's Telecom dataset, the suitability of the proposed work is well suited for IIoT, especially in cluster-based sensory networks within industry 4.0.

In the future works, edge server placement problem can be designed such that the computing capacities of edge servers are not the same and there is an upper bound defined on the computing capacities of edge servers. Another aspect of future works includes the extension of current work to a reinforcement learning approach enabling an edge server to optimally choose its location given the traffic demands and user's mobility pattern. The proposed work can be further extended by addressing adaptive optimization selection for the case-based scheme. Also, the suitability of proposed work can further be increased by including reliability as a primary-objective, thus enabling ultra-reliable low latency communication (URLLC) for critical IIoT and industry 4.0 applications.

Acknowledgement

This work is partially supported by FCT/MCTES through national funds and when applicable co-funded EU funds under the project UIDB/EEA/50008/2020; and by Brazilian National Council for Scientific and Technological Development (CNPq) via Grant No. 309335/2017-5.

References

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2017-2022 white paper," *Cisco*, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-738429.html

[2] J. Zhang, C. Xu, Z. Gao, J. J. Rodrigues, and V. Albuquerque, "Industrial pervasive edge computing-based intelligence iot for surveillance saliency detection," *IEEE Transactions on Industrial Informatics*, 2020.

[3] C. Wang, G. Yang, G. Papanastasiou, H. Zhang, J. Rodrigues, and V. Albuquerque, "Industrial cyber-physical systems-based cloud iot edge for federated heterogeneous distillation," *IEEE Transactions on Industrial Informatics*, 2020.

[4] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 5595–5637, 2017.

[5] G. Smart, N. Deligiannis, R. Surace, V. Loscri, G. Fortino, and Y. Andreopoulos, "Decentralized time-synchronized channel swapping for ad hoc wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 10, pp. 8538–8553, 2015.

[6] R. Zhao, R. Yan, Z. Chen, K. Mao, P. Wang, and R. X. Gao, "Deep learning and its applications to machine health monitoring," *Mechanical Systems and Signal Processing*, vol. 115, pp. 213–237, 2019.

[7] J. Lee and J. Lee, "Preallocated duplicate name prefix detection mechanism using naming pool in ccn based mobile iot networks," *Mobile Information Systems*, vol. 2016, 2016.

[8] J. Lee, D. Kim, and J. Lee, "Zone-based multi-access edge computing scheme for user device mobility management," *Applied Sciences*, vol. 9, no. 11, p. 2308, 2019.

[9] S. Challita, F. Zalila, C. Gourdin, and P. Merle, "A precise model for google cloud platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 177–183.

[10] G. Fortino, C. Savaglio, C. E. Palau, J. S. de Puga, M. Ganzha, M. Paprzycki, M. Montesinos, A. Liotta, and M. Llop, "Towards multi-layer interoperability of heterogeneous iot platforms: The inter-iot approach," in *Integration, interconnection, and interoperability of IoT systems*. Springer, 2018, pp. 199–232.

[11] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, and G. Fortino, "Autonomic computation offloading in mobile edge for iot applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[12] C. Stergiou, K. E. Psannis, B.-G. Kim, and B. Gupta, "Secure integration of iot and cloud computing," *Future Generation Computer Systems*, vol. 78, pp. 964–975, 2018.

[13] G. Fortino, W. Russo, C. Savaglio, W. Shen, and M. Zhou, "Agent-oriented cooperative smart objects: From iot system design to implementation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 11, pp. 1939–1956, 2017.

[14] K. Bilal, O. Khalid, A. Erbad, and S. U. Khan, "Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers," *Computer Networks*, vol. 130, pp. 94–120, 2018.

[15] T. Lähderanta, T. Leppänen, L. Ruha, L. Lovén, E. Harjula, M. Ylianttila, J. Riekki, and M. J. Sillanpää, "Edge server placement with capacitated location allocation," *arXiv preprint arXiv:1907.07349*, 2019.

[16] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Software: Practice and Experience*, 2019.

[17] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 160–168, 2019.

[18] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 787–796, 2018.

[19] H. Yin, X. Zhang, H. H. Liu, Y. Luo, C. Tian, S. Zhao, and F. Li, "Edge provisioning with flexible server placement," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1031–1045, 2016.

[20] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725–737, 2015.

[21] X. Xu, B. Shen, X. Yin, M. R. Khosravi, H. Wu, L. Qi, and S. Wan, "Edge server quantification and placement for offloading social media services in industrial cognitive iov," *IEEE Transactions on Industrial Informatics*, 2020.

[22] P. R. Norvig and S. A. Intelligence, *A modern approach*. Prentice Hall, 2002.

[23] D. Lee, S. Zhou, X. Zhong, Z. Niu, X. Zhou, and H. Zhang, "Spatial modeling of the traffic density in cellular networks," *IEEE Wireless Communications*, vol. 21, no. 1, pp. 80–88, 2014.

[24] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, 2019.

[25] J. Xu, S. Wang, B. K. Bhargava, and F. Yang, "A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3538–3547, 2019.