A WIRELESS INTERFERENCE-AWARE INTERNET-OF-THINGS GATEWAY PLACEMENT FRAMEWORK WITH GENETIC ALGORITHM APPROACH

By

KONG ZAN WAI

A dissertation submitted to the Department of Computer and Communication Technology,
Faculty of Information and Communication Technology,
Universiti Tunku Abdul Rahman,
in partial fulfillment of the requirements for the degree of
Master of Science (Computer Science) in

March 2022

ABSTRACT

A WIRELESS INTERFERENCE-AWARE INTERNET-OF-THINGS GATEWAY PLACEMENT FRAMEWORK WITH GENETIC ALGORITHM APPROACH

Kong Zan Wai

IoT (Internet-of-Things) gateways are deployed together with sensor nodes to facilitate manageability, and operational cost of the IoT system. Gateway placement optimization is implemented to strategically placing the IoT gateways, aiming to fulfil different technical requirements on top of minimizing the number of gateway. However, there is no existing gateway placement scheme that considers all the factors of number of gateways, sensor nodes coverage, lateral bound (inter-gateway) connections, redundancy for fault tolerance and dynamic changes of sensor nodes' location.

Therefore, this work proposes a framework to optimized gateway placement that considers all the aforementioned factors. The solution takes the layout of sensor nodes as input and generates a set of proposed IoT gateway locations. The framework generates the solution using genetic algorithm. Our experimental results show that solution can be generated with relatively low processing power even for a relatively wide search space. One of the contributions of this work is the formalization of the fitness function for genetic algorithm.

A series of simulations were designed and carried out to benchmark our framework against existing solutions with different evaluation criteria based on the consideration factors. Our framework gave promising results in terms of lower wireless network overlapping, minimized number of gateways required to cover all sensor nodes without compromising redundancies for fault-tolerance, and shorter overall distance of gateway movements required during the relocation due to the change of sensor nodes layout.

ACKNOWLEDGEMENT

First of all, I would like to express my greatest gratitude to my supervisor Dr. Ooi Boon Yaik for his guidance and patience, especially the endurance with my aberrantly slow pace and procrastination. I would like thank my co-supervisor, Dr. Liew Soung Yue, for all the supports and tolerance during the research period.

I would like to express my appreciation towards every personnel of Faculty of Information and Communication Technology, and Institute of Postgraduate Studies and Research in Universiti Tunku Abdul Rahman, for helping me out when I was in trouble and tolerating with the inconveniences caused by me.

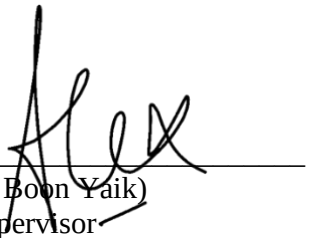
I would also like to thank my current company, Infologic Pte Ltd for supporting me through these years. Special thanks to my director Mr. Chow King Tock and manager Ms. Veronica Sunaly, for their encouragements and acknowledging my abilities.

Lastly, I would like to express my gratitude to my family, my fiancée and my best friends for the endless supports, not giving me pressure along the journey and be my beacon when I feel lost. It took longer than expected, but we are here.

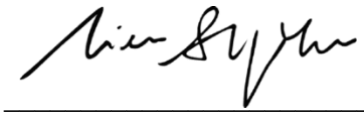
APPROVAL SHEET

This dissertation entitled **"A WIRELESS INTERFERENCE-AWARE INTERNET-OF-THINGS GATEWAY PLACEMENT FRAMEWORK WITH GENETIC ALGORITHM APPROACH"** was prepared by KONG ZAN WAI and submitted as partial fulfilment of the requirements for the degree of Master of Science (Computer Science) at Universiti Tunku Abdul Rahman.

Approved by:



(Dr. Ooi Boon Yaik) Date: 5/11/2021
Main Supervisor
Department of Computer Science
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman



(Dr. Liew Soung Yue) Date: 5/11/2021
Co-supervisor
Department of Computer and Communication Technology
Faculty of Information and Communication Technology
Universiti Tunku Abdul Rahman

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 05/11/2021

SUBMISSION OF DISSERTATION

It is hereby certified that **Kong Zan Wai** (ID No: **15ACM06682**) has completed this dissertation entitled "A WIRELESS INTERFERENCE-AWARE INTERNET-OF-THINGS GATEWAY PLACEMENT FRAMEWORK WITH GENETIC ALGORITHM APPROACH" under the supervision of Dr. Ooi Boon Yaik (Supervisor) from the Department of Computer Science, Faculty of Information and Communication Technology, and Dr. Liew Soung Yue (Co-Supervisor) from the Department of Computer and Communication Technology, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my dissertation in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

(Kong Zan Wai)

DECLARATION

DECLARATION

I Kong Zan Wai hereby declare that the dissertation/thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTAR or other institutions.

(KONG ZAN WAI)

Date 05/11/2021

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVATIONS

IoT	Internet-of-Things
AI	Artificial intelligence
GA	Genetic Algorithm
SSGW	Solution-Specific Gateway
IGW	Internet Gateway
Wi-Fi	Wireless Fidelity
CD	Coordinate Device
QoS	Quality of Service
LeTE	Low-end Transmission Equipment
ILP	Integer Linear Programming
LPWA	Low Power Wide Area
ICD	Interference cancellation and decoding
PGL	Pixel with Gray Levels

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Research Background

Internet-of-Things (IoT) is one of the most popular information technology domains in present day. 500 billion devices are predicted to be linked to the internet by year 2030 according to a report from Cisco [1]. With the rapid growing of IoT, 5G and AI, they are expected to work closely together to provide compelling services in both industrial and commercial area [2]. IoT is implemented and working in background within a lot of modern days applications [3], such as access control, public transit, industrial inspection, retail analysis, traffic system, public safety, logistic and so on. Most of these applications require continuous data collection from deployed sensor networks and convey of data to the rendezvous point such as cloud, where data analysis and post processing to more human-readable presentation shall be performed. In order to ensure the reliability of analytic outcomes and measurements, reliable IoT networks are crucial in order to prevent sporadic data collection. Scalability on the other hand, is also important for IoT network as it might be growing and altering over time after the deployment due to requirement changes of application.

As IoT network grows and comprise a large number of connected sensors nodes, deployment cost and manageability might get out of hand. The role of conventional edge devices has recently been elevated to address the aforementioned issue. Edge devices such as IoT gateways are now equipped with more hardware resources and able to run smarter applications, making them more capable than primitive access points. For example, they are able to

preprocess received data before sending to cloud, store backlogged data locally when the internet access is unavailable, and even take over certain tasks from the cloud. A noteworthy movement is that in Cisco introduced Fog Computing [4] a standard to extend computing to the edge network devices, to meet the needs of IoT. Similarly, NVIDIA also introduce their EGX platform [5] which facilitates AI production to move beyond the data centre and out to the edge layer. Through these years, There are several studies [6]–[8] that focus on the implementation of IoT gateway with different approaches on different applications, which proved its potential and feasibility.

In our previous work [9], we suggested the concept of collaborative IoT gateway, which is essentially a storage equipped single board edge computer that supports up to a minimum of 3 physical network interfaces. Each of the network interfaces has its communication role: northbound, southbound, and lateral bound. Northbound refers to connections between the cloud and gateway, southbound refers to connections between the sensor nodes and gateway; and the main novelty of this work, lateral bound, refers to the inter-gateway connections within the network.

As the center point of these 3 connections, IoT gateways certainly become the backbone [10], [11] elements of the IoT network infrastructure. I.e., the availability and quality of service of IoT gateways should create direct impact to the overall throughput of the IoT application. It is more crucial when mission-critical IoT systems are required in areas such as airfield [12] and surveillance [13]. According to [14], the common issues related to gateway

placements in an IoT network are: congestion, coverage, location, interference and distance.

An efficient gateway placement optimization scheme shall leverage the strength of collaborative IoT gateway and create a robust lateral bound network topology, subsequently provide better coverage to sensor nodes. For example, intersection region between coverage area of two or more gateways can provide redundancy to southbound devices, ultimately achieve better fault-tolerance. However, placing too many gateways at the same area might lead to opposite effects such as network congestion due to network interference [15], especially when sending large amounts of continuous data between the IoT gateways.

On top of that, IoT sensor network can be dynamic [4], [16], [17]. Sensor nodes could be added, removed, and reallocate periodically to improve the efficiency of immediate data collection. Relatively, IoT gateways should also be easily varied in terms of location to comply with the requirements from new sensor arrangement. To achieve that, the movement of existing deployed gateways to their new positions should be minimized to reduce the effort of IoT gateway relocation.

Gateway placement optimization is a subject that generally aims to address the mentioned requirements with the constraints such as number of gateway and the number of redundancies. It is an active research area with a considerable amount of published work to proposed the solution or algorithm to optimize the IoT gateway placements with different requirements and constraints on top of the basic connectivity and redundancies [18]–[21]. In this work, we propose a framework to find the optimal solutions of gateway

placement based on the locations of pre-locate sensor nodes, which the preliminary proof of concept was done and documented [22]. In our work, actuators are similar to sensor nodes from a communication perspective; thus, they are not explicitly distinguished, and both will be referred as sensor nodes unless specified otherwise. Based on our study, we managed to list out a series of consideration factors that need to be included into the gateway placement optimization scheme: - (1) Number of gateways, (2) Node coverage, (3) Lateral bound connection, (4) Redundancy and (5) Dynamic sensor node location. The more detailed problem statements are sorted out in the next subsection.

1.2 Problem Statement

According to our research, we found out that the consideration factors work relatively and extends to more issues when they have to be considered concurrently during the process of gateway placement optimization.

Firstly, to facilitate the connectivity of lateral bound network, gateways shall be place within each other's vicinity to communicate and create a strongly connected network. However, **placing IoT gateways too close together in a sensor network might cause wireless interference, ultimately affect overall network performance.** For applications where data continuity and short latency are crucial, this might be intolerable.

Secondly, to prevent single point of failure of sensor nodes' connection to cloud, they should be under coverage of more than one IoT gateway. However, **placing extra IoT gateway as redundancy will have adverse effects on the network performance and deployment cost.**

IoT sensor network can be dynamic and varies from time to time in terms of placement layout. Nevertheless, **existing gateway placement solutions have yet to accommodate changes due to dynamic sensor placement. Gateway placement will have to be agile to handle the dynamic nature.**

1.3 Objectives

The main objective of this research is basically to address the aforementioned problems, in a form of providing solutions to the multi-conditions gateway optimization problem with the following as basis:

1. To develop a framework to minimize the Wi-Fi interference between IoT gateway in the lateral bound network.
2. To design a technique to minimize the number of IoT gateway placement without sacrificing fault-tolerance by exploring the trade-off between time and sub-optimal solution using soft computing approach.
3. The design of the framework will accommodate dynamic sensors movement as consideration during computation of gateway placement scheme.

1.4 Research Contribution

The major contributions of this research are as follows:

1. The proposed gateway placement optimization framework takes multiple consideration factors into account, including number of gateways, redundancies, and network interference. The framework shall provide gateway placement layout for future IoT sensor network that is cost efficient, fault-tolerant and minimized interference.

2. The proposed framework also takes changes due to dynamic sensor layout as a consideration factor during the optimization. The optimized solution would reduce the effort to relocate gateways as sensor rearranged, which could continuously occur through the IoT system lifespan. The minimization of effort shall subsequently reduce the labor and time to reconfigure the gateway layout.
3. The proposed computational model is also highly scalable. It should be able to handle a wide range of problem size, which is the number of sensors, within acceptable computation time. In terms of software modification, the computational model should also be easily modified to include more consideration factor in future. In other words, the proposed model could work as a robust foundation and baseline model, which can be simply modified in future and branched into versions to adapt with different requirements.

1.5 **Dissertation Organization**

The remaining of this dissertation is organized as follow. Chapter 2 – Literature Review, Chapter 3 – Proposed Solution, Chapter 4 – System Implementation, Chapter 5 – Experimental Setup and Chapter 6 – Evaluation Results.

2 LITERATURE REVIEW

2.1 Gateway Placement Optimization

Gateway placement optimization is an existing research subject that aims to find the optimal locations of gateways based on a range of requirements. Gateway placement problems usually include basic requirements of minimizing number of gateway and still satisfying traffic demands [23], while modern gateway placement optimization research works would include more consideration factors or goals on top of those. A publication from Mnguni et al. [14] summarized more than 10 research works of gateway placement optimization algorithms for IoT over the past years. Despite the research works focused on IoT, there are diversities in terms of communication technology, objectives, constraints and problem size, relatively the approach taken to solve the problem are also broad.

For example, Wu et al. [23] proposed a gateway placement optimization solution for wireless mesh network. Implemented with graph theory, the algorithm aimed to find the minimum dominating sets with maximum weight among the connected mesh router networks, subsequently find the subset of routers to take over internet gateway roles, and finally assign the router attachments with load balancing as consideration factor.

On the other hand, a research work by Ahmed et al. [24] proposed to apply Genetic Algorithm (GA) to find the internet gateway locations among the connected mesh routers with the context of connectivity matrix of the mesh routers. The main objective of the research is to find near optimal solutions that

minimize the overall number of hops for mesh routers in a network to reach the nearest gateway.

Moreover, there were also several notable works reviewed through our research process, which shall be explained in the following subsections.

2.2 NewIoTGateway-Select

Karthikeya et al.[18] proposed NewIoTGateway-Select algorithm to find out optimal placement for Solution-Specific Gateways (SSGW) – edge devices that are equipped with multiple interfaces sensor, subsequently upgrading some of the SSGW to become Internet gateway (IGW). The main difference between SSGW and IGW is the latter has all the capabilities of former, but also comes with internet connectivity.

The proposed algorithm was implemented with greedy technique. It was designed to first finds out all the intersected coordinate devices (CD) pairs in terms of coverage range, and promote the most intersected ones to be SSGW until all CDs are covered by at least 1 SSGW. The step is then repeated again without the promoted CDs excluded to ensure each CD is connected to at least 2 different SSGW in order to achieve redundancies. The effective throughput of the each computed SSGW are computed, which is the product of load factor and link capacity. The link capacity is the maximum permissible load allowed on each network interface, and load factor is the ratio of actual load on the link as specific time. The SSGWs with highest effective throughput will be selected to take the role of IGW. To adapt this algorithm for our use case, the CDs can be taken as sensor nodes, where SSGW can be taken as IoT gateway. However, the algorithm does not include sensor nodes movement as consideration factor.

2.3 Device Selection Adaptive to QoS (DESAQos)

In addition, Gravalos et al. [20] suggested Device Selection Adaptive to QoS (DESAQos), a heuristic approach to find out internet gateway location and low-end transmission equipment (LeTE, which is similar to SSGW from previous work) among the IoT facilities (a group of data collection nodes, or sensor nodes). They formulated the gateway placement optimization problem into a multi-constraints integer linear program (ILP) problem, but the formulated ILP was too complex to be handled by usual ILP solver as the complexity is exponential.

Therefore, a solution with heuristic approach was proposed, which contains two parts which are the Initializion and clustering phase. Initialization phase involves computation of candidate locations of LeTE and gateway based on the existing facility centroid and Voronoi points, which are then served as the input for Clustering phase. The Clustering phase performs K-means clustering on the candidate locations to determine the locations of LeTEs and gateways.

The final outcome of the research work was proved to reduce the network installation cost without compromising the QoS. However, parameters such as the deployment price of gateway and outgoing traffic rate of facilities are required as an input of the algorithm, which we assumed to have no access to them in this work. Hence, it is not a feasible solution for our use case.

2.4 Efficient Data Collection for IoT Services in Edge Computing Environment

Furthermore, Maiti et al.[25] proposed Efficient Data Collection for IoT Services in Edge Computing Environment, which aimed to lower the overall service latency by transforming cloud-centric environment to edge-centric environment. The idea of mini-cloud was suggested in this work, which is a fog device in between the layer of cloud and gateways, expected to partially take over cloud role in IoT application. The mini-cloud locations are selected among the existing gateway, with the priority of minimizing the latencies between mini-clouds and gateways.

The authors first model the IoT gateways network as a graph with gateways as vertices, links between gateways as edges and propagation latencies as the edge weights. With this graph as basis, an n*n delay matrix which contains the shortest path latency between each pair of gateways. With the input of desired number *k* of mini-cloud and the delay matrix, the model shall go through k-means clustering algorithm, where the initial locations are selected from the gateways. But instead of averaging the distance of points with the cluster heads as in native k-means clustering algorithm, the centering is done based on the weight between the gateways and mini-cloud candidates. By the end of the algorithm, which is where convergence is occurred, the locations of *k* number of mini-cloud shall be computed.

To adapt this work to our scenario, we can take the mini-cloud as our IoT gateways, and the gateways as our sensor nodes. However, the reviewed work presume that the gateways are capable with multi-hop transmission, but our

sensor nodes are not. The mini-clouds also not defined to have lateral communications, where our IoT gateways do. Therefore, this algorithm is not suitable for our use case as well.

2.5 Improved Fast Search and Find of Density Peaks-based Fog Node Location of Fog Computing System

Similarly, a research work published by Yuan et al.[21] stated that fog node locations in fog computing network can be formulated as a clustering-based multi-constrained optimization problem. With the consideration factors of communication latency, fog node resource to cater with different types of nodes and the resource cost, they modified the existing algorithm of clustering by fast search and find by density peaks [26] that search for high density nodes based on their number of reachable neighbor nodes and the distance with other high-density nodes as cluster head, to compute the location of fog nodes within an IoT network.

The proposed clustering algorithm takes collections of nodes, along with their locations, acceptable node-fog latency, resource and resource cost as inputs, and giving outputs of cluster head locations, along with their required resource and resource costs. The algorithm involved processes of obtaining the weighted distances between all nodes, finding expected cluster sizes, iterations to find optimized cluster centers, compute required resources for cluster centers to server the cluster members and the cost.

As we can see, although this research work has a similar general goal as ours, which is to find the optimized location of IoT gateways (fog node), but the consideration factors are different. Besides, the expected input parameters of the

algorithm also vary with ours, which most of them are outside our research scope. Therefore, it can be concluded that the algorithm is also not suitable to be a solution of our problems.

2.6 Optimized Gateway Placement for Interference Cancellation for Transmit-Only LPWA Networks

Tian et al. [27] attempted to optimize gateway placement with the objective to minimize wireless interference within the transmit-only low power wide area (LPWA) network. The research work was built on top of the concept of capture effect and interference cancellation, which are the methods used to process collision packets due to wireless interference. The capture and interference cancellation process are to be done by gateways that are placed among the transmit-only sensor nodes, which is to resolve the collisions between packets from different connected sensor nodes. To ensure that the processes can be carried on, the gateways will need to be placed in the effective regions within the transmission range of sensor nodes.

Two algorithms were proposed to find the locations of gateways. Both algorithms involved the computation of the following properties for each sensor nodes with respect to the other nodes: 1) capture circle – the transmission region where capture process can be performed, 2) ICD crescent – the transmission region where interference and decoding can be performed outside of the capture circle. Afterwards, the optional points, which are the intersection points between capture circle and ICD crescents are located. These optional points are the candidates of gateway locations. To decide which are the final points to place a gateway, two algorithms were proposed.

The first algorithm is the Algorithm Weight Bipartite Graph (WBG). It generally creates a bipartite graph based on the optional points and the intersected sensor nodes, with the weight of the graph edges as the intersection type. With the greedy technique, the algorithm iterates to pick the optional point from the bipartite graph with highest weight (located within intersection of most capture circles and ICD crescents), and removed all the linked intersected sensor nodes, until all sensor nodes are removed from the graph. The picked points are then selected as the proposed gateway locations.

The second algorithm is the Algorithm of PGL (Pixel with Gray Levels). It is similar to the first algorithm, in terms of the greedy technique part. However instead of using the bipartite graph, the weight of the optional points is identified by the pixel greyscale darkness level. As the optional points are actually on the intersected regions of capture circle and ICD crescent, visualizing the regions with translucent colour shall create darker colors within the overlapped regions, where darker regions mean higher level of overlapping.

However, the research works does not have the same consideration factors as ours. Factors such as redundancies, and sensor layout change are not taken into accounts. Despite, the PGL approach of identifying level of overlapping based on greyscale level still inspired us as an evaluation methodology.

2.7 **Robust Gateway Placement for Scalable LoRaWAN**

Lastly, Loh et al. [28] suggested that the IoT gateway placement optimization as a geometric set cover problem, where gateways are disks and sensors are points. Each gateway should be assigned with a maximum number

of sensor (capacity), where the main task is to ensure that each sensor is covered by a gateway without exceeding the capacity.

The algorithm starts with providing the sensor node locations and candidate gateway locations as input. A Voronoi diagram based on the gateway locations that partitions the sensor nodes will be created. Each sensor nodes within the Voronoi geometry with a gateway as center shall be attached to that gateway, and the number of gateways shall be computed to ensure the condition of gateway capacity is not violated. If the condition is fulfilled, a selected gateway shall be removed from the set and the repeat the condition fulfilment validation process, until 3 gateways are removed but still fulfilling the requirements of not exceeding gateway capacity and all sensors are covered, OR no gateway is allowed to be removed.

The algorithm managed to solve a capacitated geometric set cover problem within a reasonable time based on real life Smart City scenario. However, the objective and consideration factors are quite different with ours. The work did not take fault tolerance with redundancies into account, and also not taking sensors relocation as a consideration criterion.

2.8 **Comparison between existing techniques**

Table 2.1 on next page concludes the existing technique, based on their model/algorithm, objective, and the research outcome.

#	Research	Model/ algorithm	Objective	Research outcome
1	A Genetic Approach for Gateway Placement in Wireless Mesh Networks [24]	Genetic Algorithm	Minimize the variation of (Mesh Router – Internet Gateway) MR-IG hop counts	Improved overall WMN performance by reducing MR-IG hop counts
2	Leveraging Solution-Specific Gateways for Cost-Effective and Fault-Tolerant IoT Networking [18]	Greedy technique	Reduce cost by minimizing number of gateways, replace IGW with SSGW when possible	Managed to proposed gateway placement locations with minimized number of gateways without single point failure
3	An Improved Fast Search and Find of Density Peaks-based Fog Node Location of Fog Computing System [21]	Improved K-means clustering technique from Clustering by fast search and find of density peaks [26]	Proposed an improved algorithm to locate fog node sites and determine resource for each fog node	Takes shorter time to locate fog nodes compared to existing algorithm, giving significant better service performance (shorter latency) with the trade-off of insignificant increment in cost.
4	Efficient Data Collection for IoT Services in Edge Computing Environment [19]	Binary knapsack	Lower the service latency compared to IoT with cloud environment	Service latency is lower with mini-cloud among IGW compare to cloud environment
5	Efficient Network Planning for Internet of Things with QoS Constraints [29]	Voronoi points, K-means clustering	Lower network installation cost without compromising QoS,	Gives near optimal installation cost compared to ILP Method
6	Optimized Gateway Placement for Interference Cancellation in Transmit-Only LPWA Networks [27]	Greedy technique	To find the optimum location of gateways for transmit-only LPWA networks,	Proposed WBG and PGL greedy technique based algorithms to find gateway locations with minimal interference
7	Robust Gateway Placement for Scalable LoRaWan [28]	Voronoi points, linear programming	To minimize gateway count, but ensuring that all sensors are covered without exceeding gateway capacity	Solve a capacitated geometric set cover problem within a reasonable time based on real life Smart City scenario and fulfilling the objective

Table 2.1 Comparison between existing techniques

2.9 Genetic Algorithm

As more consideration factors are added on top of primitive gateway placement optimization, the problem would get more complex due to the increment of variable number, variable scope, and variable diversity. A heuristic and global search approach would be preferable [30] to find a solution that does not need to be optimal at each requirement, good enough to fulfill the minimum requirements, wider search space, with a considerably short computational time. For our scenario, immediate solution is not the main priority, instead global search for sub-optimal solution is preferable. Therefore, locally search algorithms such as Greedy Technique (as implemented in [18], [27]) and Linear Programming (as implemented in [28]) is not a suit for this case. We find that Genetic Algorithm (GA) [31] with the mentioned characteristics is a good fit to implement the gateway placement optimization model as gateway placement does not require a precise solution. On top of that, GA is also highly scalable, thus modifications that need to be done in case of inclusion of new consideration factor could be done with less effort.

GA is a search heuristic that was inspired by the theory of evolution. It imitates the process of natural selection, where the fittest individuals (genes) are chosen for reproduction to produce offspring of the next generation. In GA, a gene in a chromosome commonly refers to a variable in a solution to the problem; and a population refer to a pool of chromosomes. Generally, the implementation of GA shall consist of the following steps:

a) **Initialization** – A population of chromosomes (solution) with randomly generated genes (variables) is initialized

b) **Selection** – Two chromosomes (solution) are randomly selected from the population

c) **Crossover** – Part of the genes (variables) from the selected chromosomes (solutions) were extracted and combined to yield a new chromosome (new solution)

d) **Mutation** – With a slight chance, a gene, or genes (variables) of the new chromosome (solution) shall be mutated randomly (assign with new random value)

e) **Fitness Evaluation** – The newly generated chromosomes (solution) go through an evaluation process to assess the suitability of solving the problem

2.10 Summary

Gateway placement optimization is still a relevant research subject, as we are able to find publications from the past few years. However, the researches have different goals to meet, mainly because of the diversities of requirements and objectives to pursue. The fields of focus for the reviewed works spanned through different domains, such as cost efficiency, computing performance, system reliability, quality of service and so on.

There are works that have intersected fields of interest as us, but none of them solve the same set of problems as ours. It is also worth noting that none of the reviewed works has taken changes due to dynamic sensor network into account during the optimization. Nonetheless, the work from Karthikeya et. al. [18], which will be referred as NewIoTGateway-select has the closest solution

for the requirements to us, thus it will be taken as a benchmark object in the as documented in the latter chapters.

3 PROPOSED SOLUTION

As we decided to implement the framework using Genetic Algorithm, the next step would be to formulate the gateway optimization problem and adapt to the algorithm. In this chapter, we will focus on the problem formulation with the requirements based on our research objectives and the methodology to model the problem as the input to our computational model.

3.1 Problem formulation

In this work, it is presumed that the gateway optimization problem is within a finite spatial plane with a height *h* and width *w*, where sensor nodes are distributed around the plane, resembling the sensor network in a monitored field. In real life scenario, the sensor node locations should be able to be recorded on a map or floorplan.

For instance, Figure 3.1 shows a shopping mall floor plan with the locations of sensor nodes of a crowd counting project. With the aid of the floorplan, the sensor node locations can be converted into a collection of XY coordinates, which would be the input of the framework. The framework is also expected to take the desired number of gateways, and original gateway coordinates (obtainable with the same method as sensor nodes) as inputs to yields a solution of proposed gateway layout solution. The detailed workflow of the framework is described in Figure 3.2.

Figure 3.1 Example of sensor layout on mall floorplan

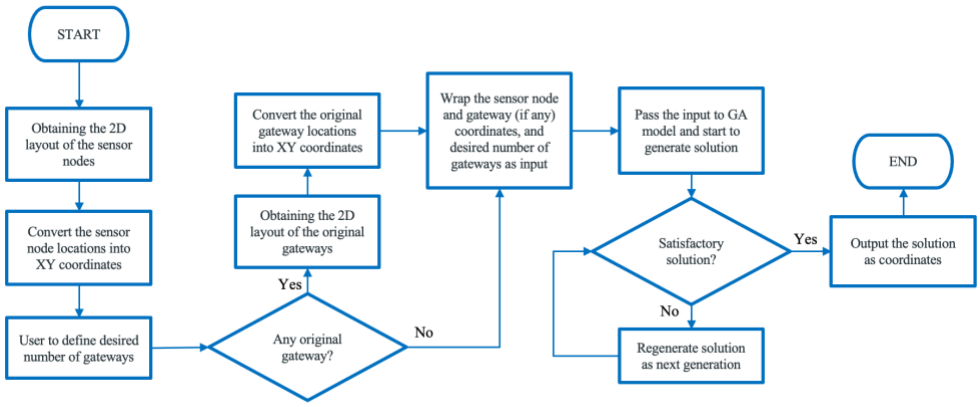


Figure 3.2 Workflow of proposed framework

Based on our research objectives, we mainly aim to design a framework to provide solutions for a multiple-conditions gateway placement optimization problem. The framework should be able to yield solutions that 1) minimize the Wi-Fi interference between IoT gateway in the lateral bound network, 2) minimize the number of IoT gateway without compromising fault tolerance, and 3) adaptive to dynamic sensor movement.

To formulate the gateway placement optimization problem, we listed out a series of factors that must be considered during the problem solving based on our scopes and objectives as illustrated in Figure 3.3:

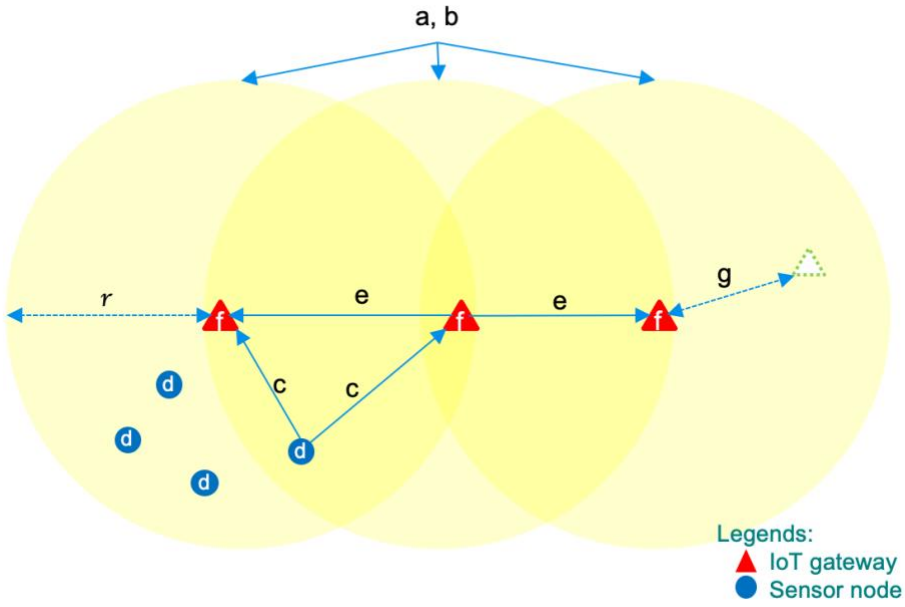


Figure 3.3 Illustration of consideration factors

a) Average coverage of an IoT gateway for sensor nodes (southbound).

   $A_{south} = \pi r^2$

b) Average distance of an IoT gateway lateral bound (in radius) for IoT gateway collaboration.

   $A_{lateral} = \pi r^2$

c) Number of gateways a node can use to provide redundancy, i.e. the number of gateways a node can connect to.

d) Number of sensor nodes under the coverage of a gateway.

e) Number of collaborative IoT gateway within the average lateral bound distance set in (b)

f) Number of gateways desired.

g) Distance between the original and proposed gateway positions (if any)

3.2 **Proposed model**

Based on our framework structure, the output of the model (solution) is expected to be the layout of gateways within the 2D spatial plane. Therefore, the GA model of our framework shall be expressed as:

- Chromosome (solution): 2D layout of gateways
- Gene (variables): The location of each gateway of the layout

On this basis, we model the gateway placement optimization to comply with the steps of a GA model, where the details are explained as follows:

3.2.1 Initialization

The purpose of initialization is to create an initial population of N chromosomes as an entry point of the computation. The properties of the chromosomes should be random. In our model, the chromosome is the gateway layout on a 2D spatial plane. In general, the initialization process shall generate a pool of random layouts of gateways.

The values that can be randomized in a set of gateways are:

- Number of gateways in the layout
- Location of each gateway

At the end of the initialization, the model is expected to create a defined number of gateway layouts, where each layout shall contain a randomized number of gateways, and each gateway shall have a random location within the spatial plane.

3.2.2 Selection

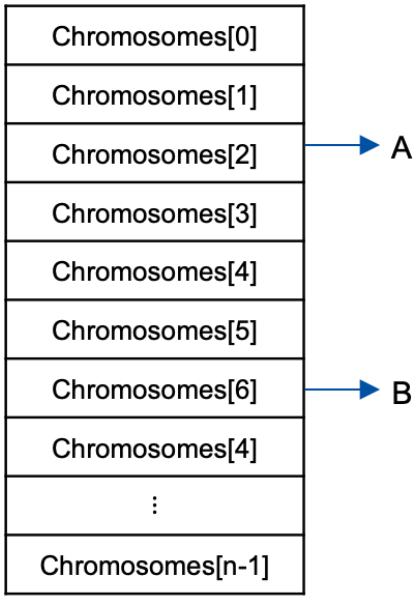


Figure 3.4 Illustration of Selection

During the selection process, two chromosomes in the population shall be randomly chosen from the population to go through the crossover process, repeated until a new generation of population is formed. In our model, it will be presented as randomly selecting two different gateway layouts from the current population (illustrated in Figure 3.4) which will be taken as inputs of crossover (explained in next subsection). The process will be iterated with different random layouts for a defined number of times.

3.2.3 Crossover

Every pair of selected gateway layout from the selection process shall go through the crossover process. For each pair of layouts, says $layout_a$ and $layout_b$, a portion from each layout is extracted and joint to yields a new layout (visualized in Figure 3.5 below). In our model, we proposed to take the left portion of $layout_a$ and right portion of $layout_b$ to go through the concatenation.

However, instead of cropping each layout by exactly half, a random percentage of $layout_a$ and $layout_b$ (with total of 100%) shall be taken to increase the randomness of GA model. For example, if the randomized percentage is 30%, then 30% of $layout_a$ from the left and 70% of $layout_b$ from the right shall be combined and produce a new gateway layout.

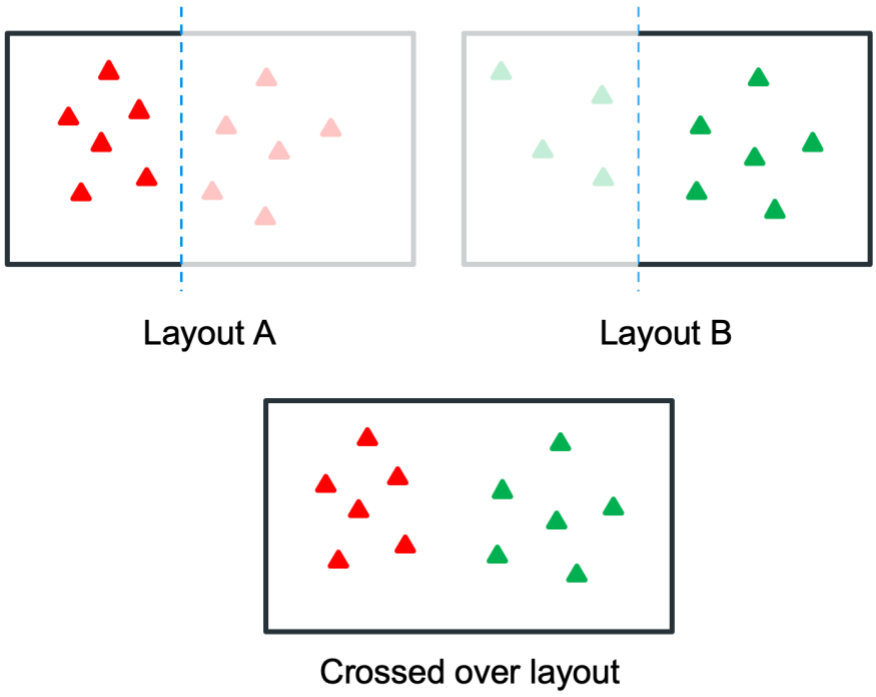


Figure 3.5 Illustration of Crossover

3.2.4 Mutation

By a small chance, the product of crossover shall get to go through mutation process. The mutation is done by randomly tweaking the genes in the chromosome. In our model, we proposed to have 3 types of mutation that involve the process of adding, removing and updating the genes to/from the chromosomes, as illustrated in Figure 3.6.

1. **Adding** – a new gene with randomized value shall be generated and append to the chromosome. I.e. a new extra gateway with random location shall be generated and added to the crossed-over layout of gateways.
2. **Removing** – an existing gene shall be selected randomly and removed from the chromosome. I.e. an existing gateway from the crossed-over layout of gateways shall be randomly selected and removed from the layout.
3. **Updating –** The properties of a random existing gene in a chromosome shall be assigned with a random new value. I.e. a random existing gateway from the crossed-over set of gateways shall be randomly relocate.

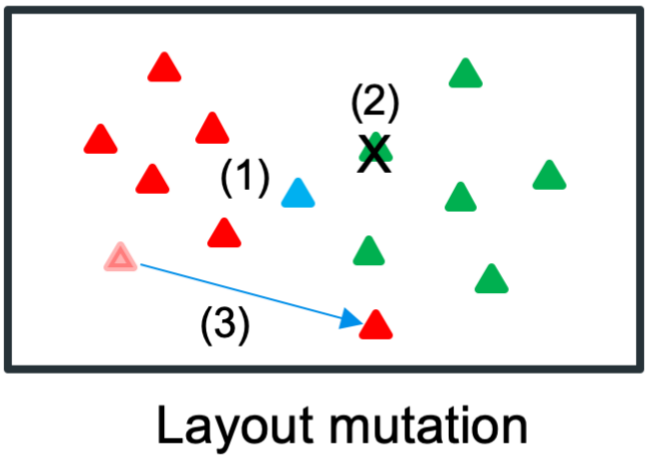


Figure 3.6 Illustration of Mutation

3.2.5 Fitness Evaluation

For each of the chromosomes produced by crossover (mutated and not mutated), it must go through a fitness evaluation process. For our case, the fitness evaluation is about assessing the performance of gateway layouts based on the

input of the model, which are the sensor nodes layout, number of desired gateways in the layout and the original gateway layout (if any).

To evaluate the fitness of the gateway layout, we have come out with a list of evaluation criteria corresponded to our consideration factors, namely:

a) Number of gateways a node can use to provide redundancy
b) Number of sensor nodes a gateway can cover
c) Number of collaborative IoT gateway within the average lateral bound
d) Number of gateways desired
e) Distance between original (if any) and proposed gateway positions

With the system of rewards and penalties, scores shall be rewarded or deducted based on the fulfillment of each criterion. Table 3.1 presents the grading system of the fitness evaluation of our model:

Score is rewarded	Consideration Factor	Score is deducted
• For each node connected to a gateway • Reward gradually decrease to avoid too much overlapping ** Extra score if node covered by exact number of gateways configured by user	Number of gateways a node can use to provide redundancy	• For each uncovered node
• For each node a gateway covers • Gradually increases as the number of nodes under the coverage of the gateway increases	Number of sensor nodes a gateway can cover	• For each unconnected gateway
• For each pair of gateways with intersected coverage • Reward decreases as the number of intersections of each gateway increases	Number of collaborative IoT gateway within the average lateral bound	• No change
• If the number of gateways is less than the user-defined number of gateways.	Number of gateways desired	• If the number of gateways is greater than the user-defined number of gateways.
• No change	Distance between original (if any) and proposed gateway positions	• Directly proportional to the distance between original and generated gateway positions.

Table 3.1 Score system based on consideration factors

With all the gateway layouts evaluated with a score, they will be ranked accordingly. The layout with the highest score shall be taken as the latest solution. However, if the user opts not to stop at current generation, the top performers from the current population shall be extracted as the candidates of next generation of population. The new generation shall go through the selection, crossover, mutation, and fitness evaluation process again. The whole cycle shall be looped until the user stops it, which is usually when the acceptable solution is yielded.

3.3 **Summary**

In this chapter, we presented the high-level overview of the framework, consisting of the inputs, model and outputs. The gateway placement optimization problem formulated based on the consideration factors. Genetic algorithm is selected as the baseline of the computational model due to the nature and complexity of the gateway placement optimization problem. The methodology of genetic algorithm was studied and explained. With the consideration factors as basis, the purposed solution is modelled around the steps in a generic GA life cycle: initialization, selection, crossover, mutation, fitness evaluation. The proposed solution shall be implemented into a software program which is explained in the next chapter – System Implementations.

4 SYSTEM IMPLEMENTATION

In the previous chapter, we explained on the multiple-conditions gateway placement optimization problem formulation and modeling to adapt with Genetic Algorithms as a solution. To realize our proposed solution, we have implemented the design as a software program. In this chapter, the programming concepts of the implementation will be presented in detail. The program was written in C# with .NET Framework. However, the explanations of the modules will be presented in the form of abstraction.

4.1 Basic elements

According to the proposed solution, the expected input of the gateway placement optimization model are the desired number of gateways, sensor node distribution layout, and original gateway layout (optional), and the expected output is a solution of proposed gateway layout. Technically, a sensor/gateway layout is a composite of individual nodes or gateways with their locations on a 2D spatial plane. The location on a 2D spatial plane can basically be represented as an XY cartesian coordinate. With this concept, the gateway and sensor node can be form into objects as represented in Figure 4.1:

```
Sensor
{
    X: int
    Y: int
}
```

```
Gateway
{
    X: int
    Y: int
}
```

Figure 4.1 Structure of Sensor and Gateway objects

Relatively, a layout as a collection of gateways or sensor, can be represented as a list of gateway or sensor objects (as illustrated in Figure 4.2).

```
SensorLayout
{
    Members: Sensor[]
}
```

```
GatewayLayout
{
    Members: Gateway[]
    Score: int
}
```

Figure 4.2 Structure of SensorLayout and GatewayLayout object

These structures will be working as the basis elements through the entire implementations, where the gateway layout structure also represent as the chromosome in the genetic algorithm model.

4.2 Genetic Algorithm Implementation

In this section, the implementation of the GA-based gateway placement optimization model shall be explained. The model as proposed in the previous chapter, shall follow the generic live cycle of generic GA, which consist of the modules of initialization, selection, crossover, mutation and fitness evaluation. The overview of the life cycle is basically implemented as the following logic in Figure 4.3:

```
GatewayPlacementOptimizationModel
Inputs:
desiredGatewayNumber: int,
sensorLayout: SensorLayout,
originalGatewayLayout: GatewayLayout
Output:
```

```
solution: GatewayLayout
{
  var population = Initialization(POPULATION_SIZE)
  While !Stop
    var newGeneration

    For i = 1 to maxCandidateCount
        var parents = Selection(population)
        var child = Crossover(parents)
        child = Mutation(child)
        child.Score = FitnessEvaluation(child,
                      sensorLayout
                      originalGatewayLayout)
        newGeneration.Add(child)
    End For

    newGeneration = SortByHighestScore(newGeneration)
    population = newGeneration.Slice(0,
                 population.count)
    solution = newGeneration.First()
  End While
}
```

Figure 4.3 Pseudocode of proposed GA model

Based on the pseudocode, the model is taking 3 inputs, and giving 1 output. The program starts by creating an initial population with the Initialization function. In a loop that only stops at user command, a new generation is created based on the current population, which begin with Selection from the current population. For each of the selected "parents" from selection, they serve as the input of Crossover which would produce a child. The child would be taken as input for Mutation, where it might return the same or mutated child. Next, the child shall go through Fitness Evaluation and assigned with a score, subsequently added to the pool of new generation candidates.

The steps are repeated until a defined number of candidates are created. The candidates will then be sorted by their score from highest to lowest, where the top N candidates will be taken to replace the current population, and the top candidates will be used as the current solution. Based on the nature of GA, the solution might be replaced through the generations. Once the user stops the program, the solution from current generation will be taken as the final solution.

Next, we will explain on how each of the GA function works in detailed.

4.2.1 Initialization

The Initialization is a function with one input which is the size of population, and shall return a collection of GatewayLayout. The function as described in the pseudocode in Figure 4.4, starts by creating an empty population, and repeatedly adding newly generated GatewayLayout object to it until it reaches the defined size, eventually return it. Each of the generated GatewayLayout shall contain an array of Gateway object with randomized X and Y value, in its Members property, where the array length equals to defined number of user desired gateway count.

```
Initialization
Input:
// number of chromosomes in the population
populationSize: int
{
  // create an empty population
  var population: GatewayLayout[]

  For i = 1 to populationSize
    // create an empty layout
    var layout : GatewayLayout
```

```
    // create a number of gateway layouts
    // based on user defined number of desired gateway
    For j = 1 to USER_DESIRED_GATEWAY_COUNT
      var member : Gateway

      //with randomized X Y coordinates
      member.X = random(1, MAX_WIDTH)
      member.Y = random(1, MAX_HEIGHT)
      layout.Members.Add(member)
    End For

    // add the layout to population
    population.Add(layout)
  End For

  return population
}
```

Figure 4.4 Pseudocode of Initialization

4.2.2 Selection

The Selection function (Figure 4.5) takes a collection of GatewayLayout as input, with the return value of two GatewayLayout objects. It is a relatively simple function which randomly selects two items from the input GatewayLayout array and returns them as an array of 2 elements.

```
Selection
Input:
population: GatewayLayout[]
{
  var max = population.Count

  // randomly pick two layout from the population
```

```
  var parentA = GatewayLayout[random(0,max)]
  var parentB = GatewayLayout[random(0,max)]

  // wrap the two parents and return
  var parents = { parentA , parentB }

  return parents
}
```

Figure 4.5 Pseudocode of Selection

4.2.3 Crossover

The function of Crossover is to extract the genes from the two parents GatewayLayout objects and combine them into a child. Based on our proposed solution, the Crossover should take the left portion of layout A and right portion of layout B with random percentages that sum up to 100%. In our implementation as shown in the pseudocode in Figure 4.6, instead of using percentage, a random point between the width of the 2D spatial plane is generated as a slicing point. Any Gateway object in layout A that has an X value smaller than the slicing point, and any gateway object in layout B that has greater or equal value as the slicing point, shall be extracted and merged as a new chromosome. Although the approach looks different on paper, theoretically it should be the same as what we have proposed in last chapter.

It is also notable that the implemented Crossover function has a prevention mechanism to not returning a child chromosomes (GatewayLayout) with empty genes (0 Gateway).

```
Crossover
Input:
```

```
parents: GatewayLayout[2]
{
  // create a child with no member
  var child: GatewayLayout


  // to ensure that the eventual child member
  // is not empty
  While child.Members is empty


    // generate a horizontal slicing point within
    // the layout
    var slicingPoint = Random(0, MAXWIDTH)


    // extract the left side of layoutA and
    // right side of layoutB based on slicing point
    // inside the where() function is a lambda
    // expression to compare the X value of member with
    // slicing point
    var genesA = parents[0].Members.
                   where(g => g.X < slicingPoint)
    var genesB = parents[1].Members
                   where(g => g.X >= slicingPoint)


    // combine the two genes into one and assign
    // to child
    child.Members = genesA.Join(genesB)
  End While


  return child
}
```

Figure 4.6 Pseudocode of Crossover

4.2.4 Mutation

The Mutation function block aims to tweak the genes of a chromosomes with a slight chance. As proposed in the previous chapter, the mutation block is

expected to contain three types of operations, which are add, remove and update a gene (Gateway) in a chromosome (GatewayLayout).

In our implementation (refer to Figure 4.7), the three operations are mutually exclusive. The input chromosome has individual chances to go through the operations, ranging from neither, any or all. To decide whether the chance is hit, a random number is generated between 1-100, if the number is smaller than the mutation rate (e.g. 3 for 3%), the corresponding block shall be run.

For addition operation, a new Gateway object with randomized X and Y value within the 2D spatial plane dimensions shall be generated and append to the layout's Members property. The removal operation shall randomly pick a Gateway object from the layout's Members property and remove it from the array. The update operation shall randomly pick a Gateway object from the layout's Members property and assign it with a random X and Y properties.

```
Mutation
Input:
child: GateawayLayout
mutationRate : Int between 1-100
{
  // Mutation rate
  If Random(1, 100) < MUTATION_RATE
    // Create a new Gateway object with random XY
    var newMember : Gateway
    newMember.X = random(1, MAX_WIDTH)
    newMember.Y = random(1, MAX_HEIGHT)
    // Append to existing members
    child.Members.Add(newMember)
  End If

  // Mutation rate
```

```
  If Random(1, 100) < MUTATION_RATE
    // To prevent 0 Gateway upon removal
    If child.Members.count > 1
      // randomly pick a gateway and remove
      var index = Random(0, child.Members.count - 1)
      child.Members.Remove(child.Members[index])
    End If
  End If

  // Mutation rate
  If Random(1, 100) < MUTATION_RATE
    // randomly pick a gateway
    var index = Random(0, child.Members.count - 1 )
    var x = random(1, MAX_WIDTH)
    var y = random(1, MAX_HEIGHT)
    // assign with new randomized value
    child.Members(child.Members[index]).X = x
    child.Members(child.Members[index]).Y = y
  End If

  return child
}
```

Figure 4.7 Pseudocode of Mutation

4.2.5 Fitness Evaluation

The Fitness Evaluation function (pseudocode in Figure 4.8) is the essence of our GA model. With the inputs of candidate gateway layout, sensor layout and original gateway layout, the function mainly aims to grade the GatewayLayout object based on multiple consideration factors as per described in section 3.2.5. The evaluation process is divided into two parts, which are rewards and penalties.

The evaluation score starts with 0, increase/decrease according to the rewards and penalties. The first consideration factor is the number of gateways a node can use to provide redundancy. In our implementation, we find out the number of connectable gateways for each sensor node by computing the distance between all gateways and sensors, subsequently compare with the defined coverage radius. For each gateway a sensor can connect to, a defined score factored by the number of connected gateways will be added, resulting with gradually lower rewards with respect to redundancy saturation. The final number of connectable gateways for each sensor are also recorded, where each achievement of exact redundancy number as defined by user shall be given extra points. On the other hand, heavy penalty will also be incurred if for every node that has insufficient redundancy.

The second consideration factor is the number of sensor nodes a gateway can cover. Based on the computed number of connectable gateways for each sensor node, for each sensor node a gateway can connect to, defined score multiply with the number of connected nodes will be given, resulting with gradually higher score for each sensor nodes under coverage. The final number of covered sensors are recorded, and penalties will be given for each gateway that has 0 node under coverage.

The next consideration factor of the fitness evaluation is the number of collaborative IoT gateway within the average lateral bound. Among the gateway layout itself, the connectivity between all gateways are computed, again by computing the distance sand compare against the defined coverage radius, where lower than means within the communication vicinity. For each gateway, every

intersection with other gateway is entitled with a score reward, which is factored by of the intersected target, resulting with gradually decreasing rewards.

Number of desired gateways is an expected number of gateways in the solution gateway layout defined by user. The evaluation function shall award score for every unit of gateway less than the defined number, but deducted for every extra one in the input GatewayLayout. The last consideration factor during fitness evaluation is the distance between original and proposed gateway positions, which is omittable if no original gateway layout is provided. A lumpsum computed by the moving penalty calculator function (Get Moving Penalty – explained in details in next subsection) will be deducted from the total score and there goes the fitness score of the chromosome (GatewayLayout).

```
FitnessEvaluation
Input:
candidate : GatewayLayout
sensorLayout : SensorLayout
originalGatewayLayout : GatewayLayout
{
  var gateways = candidate.Members
  var gatewayCount = gateways.Count
  var sensors = sensorLayout.Members
  var sensorCount = sensors.Count

  var score : int
  // number of gateways each sensor can connect to
  var sensorConnectedIndex : int[]
  // number of sensors each gateway can cover
  var gatewayCoverageIndex : int[]
  // number of gateways a gateway can connect to
  var lateralConnectionIndex : int[]
```

```
// initialize records with 0s
For i = 0 to sensorCount
  sensorConnectedIndex.Add(0)
End For
For i = 0 to gatewayCount
  gatewayCoverageIndex.Add(0)
End For
For i = 0 to gatewayCount
  lateralConnectionIndex.Add(0)
End For


For i = 0 to gatewayCount
  For j = 0 to sensorCount
    If(Distance(gateways[i], sensors[j]) <
        COVERAGE_RADIUS)
      // increment of sensor coverage count
      ++gatewayCoverageIndex[i]
      // increment of gateway connectivity count
      ++sensorConnectedIndex[j]
      // Score given for each gateway a sensor
      // can connect to, decrease gradually
      // to prevent overcrowding
      Score += NODE_COVERAGE_REWARD /
               sensorConnectedIndex[j]
      // Score given for each sensor a gateway
      // can cover, increase linearly
      Score += NODE_COVERING_REWARD *
               gatewayCoverageIndex[i]


    End If
  End For


  For k = i + 1 to gatewayCount
    If Distance(gateways[i], gateways[k] <
        COVERAGE_RADIUS)
      // increment of lateral bound connectivity
      // for both gateways
```

```
        ++interGatewayConnectedIndex[i]
        ++interGatewayConnectedIndex[k]
        // reward score for each intersection
        // reward decrease gradually
        // to prevent overcrowding
        score += GATEWAY_INTERSECT_REWARD /
                 lateralConnectionIndex[i];
      End If
    End For

    // Score given for each exact redundancy achieved
    Foreach index in sensorConnectedIndex where
            index equals to USER_DEFINED_REDUNDANCY
      score += EXACT_REDUNDANCY_REWARDS
    End Foreach

    // Heavy penalty for each sensor without
    // sufficient redundancy
    Foreach index in sensorConnectedIndex where
            index is less than USER_DEFINED_REDUNDANCY
      score -= REDUNDANCY_PENALTY
    End Foreach

    // Penalty for each gateway without any
    // sensor nodes under coverage
    Foreach index in gatewayCoverageIndex where
            index is 0
      score -= NOT_COVERING_PENALTY
    End Foreach

    // Moving penalty
    var movingPenalty = GetMovingPenalty(
                        candidate, originalGatewayLayout)
    score -= movingPenalty
    // Score rewarded for each gateway less compared
    // to user defined number, but deducted
    // for each extra
```

```
    var gatewayCountReward = (USER_DESIRED_GATEWAY_COUNT
                              - gatewayCount) *
                              GATEWAY_COUNT_REWARD
    score += gatewayCountReward

  end For
  return score
}
```

Figure 4.8 Pseudocode of Fitness Evaluation

4.2.6 Get Moving Penalty

The Get Moving Penalty (Figure 4.9) sub-function is used to compute the reduction score incurred by the movements of gateways between two layouts. The idea of gateway moving is based on the case of user having to physically move the gateways from original locations of existing layout to new locations of proposed layouts due to the changes of sensor layout change. The moving penalty is basically the effort required to move the existing gateways to their new locations, which is the total travel distance.

To compute the total distance, all the distances between the gateways in original layout and new layout will be computed and stored in a list along with their indices (the length of list is expected to be number of gateways in original layout times number of gateways in new layout). The list will be sorted ascendingly by the distances and the shortest combinations of original-new gateway locations shall be taken with the Greedy approach. Eventually, the sum of the taken combinations' distances is return as the moving penalty.

With this implementation, if either of the layouts has empty members (gateways), it shall return 0. If there is difference between the number of

gateways in the layouts, the difference will be considered as addition or removal to/from deployment, not contributing to the moving penalty.

```
GetMovingPenalty
Input:
layoutA: GatewayLayout
layoutB: GatewayLayout
{
  var gatewaysA = layoutA.Members
  var gatewaysB = layoutB.Members

  // distance mapper, an array of object with
  // properties of distance, and indices of
  // gateways
  var distanceMapper : {distance : int,
                        indexA: int,
                        indexB: int}[]
  var penalty = 0. // initial penalty is 0

  // find out all the distances between gateways
  For i = 0 to gatewaysA.Count - 1
      For j = 0 to gatewaysB.Count - 1
        var distance =
            Distance(gatewaysA[i],gatewaysB[j])
        // record the distance to mapper
        distanceMapper.Add({distance, i, j})
      End For
  End For

  // sort the array based on distance
  distanceMapper =
        distanceMapper.SortByDistanceAscending()

  // while the mapper still contains record
  While distanceMapper.Count > 0
```

```
    // take the curent shortest distance
    // and add to penalty
    penalty += distanceMapper[0].distance
    i = distanceMapper[0].indexA
    j = distanceMapper[0].indexB
    // remove all records which has
    // either gateway of the removed record
    distanceMapper.RemoveWhere( d =>
      d.indexA equals to i OR
      d.indexB equals to j
      )
  End While
  return penalty
}
```

Figure 4.9 Pseudocode of Get Moving Penalty

4.2.7 Distance

```
Distance
Input:
nodeA: Gateway or Sensor
nodeB: Gateway or Sensor
{
  // Pythagoras theorem
  distance = sqrt(
       (nodeA.X – nodeB.X)² + (nodeA.Y – nodeB.Y)²)
  return distance
}
```

Figure 4.10 Pseudocode of Distance

The distance sub-function (Figure 4.10) returns the distance between two nodes, with any combinations of gateway and sensor. The function basically utilized the Pythagoras theorem $a^2 + b^2 = c^2$ to compute distance between two points, with the aid of XY coordinates in each node.

4.3 Realization

The implementations described above were written into a software with C# programming language with .NET Framework. The program consists of a graphic user interface where the proposed solution of current generation is presented to user in the graphical 2D spatial plane.

The information such as current generation and number of gateways are also shown, and there are controls where user can insert the source sensor nodes layout and original gateway layout (in .json file format), and buttons to start and stop the computation. Upon stopping the computation, the last generated solution shall be exported as a .json file.

4.4 Summary

In this chapter, the implementation of the proposed solution is explained in detail majorly in the form of abstractions. The model with native GA cycle steps is described and presented in pseudocodes which programmed according to the designs in the proposal. However, there were some parts where adaptation works are required to formulate the problem into a more computable form. The codes were written into a desktop program where GUI is available for user to have a more interactive experience.

5 EXPERIMENTAL SETUP

5.1 Simulation Setup

A set of simulations is designed and run to evaluate the performance of the proposed framework against other techniques to propose a solution for gateway placement optimization problem. The simulation data is derived and transformed from an existing real-time crowd counting application, where sensors are located around a shopping mall to detect visitors flow. The simulation is setup and run with the following basis:

- Two IoT-gateways should be placed within the lateral bound communication vicinity to ensure communicability.
- The average coverage of the southbound and average distance of the lateral bound are set the same because based on our actual implementation, both are using 2.4GHz Wi-Fi.
- To ensure that each sensor node is covered by redundancy k=2, each location is placed with 2 gateways for the uniform arrangements.

In general, each technique would be given with a same series of problem inputs: the sensor node locations in a 2D spatial plane; and expected to yield a solution for each problem: the gateway locations.

For implementation, each problem is modeled as a file that consists of a list of json node objects (refer to Figure 5.1) with randomized X and Y coordinates within a designated range. Every technique is implemented into programs written in C# language, which would read the input file, propose a solution based on the deserialized input and consequently record the solution as another file that contains a list of json gateway objects with computed X and Y coordinates as

output. For uniform arrangements, the implemented program shall always return a constant solution that is hardcoded but sufficient to fulfill the mentioned requirements (to ensure that each sensor node is covered by redundancy k=2, each location is placed with 2 gateways). The simulation is basically to iteratively running every program of particular technique with the array of input files until all inputs have a proposed solution from each technique. The outputs are subsequently analyzed and evaluated based on different criteria, the simulation to evaluate each criterion are elaborated in following subsections.

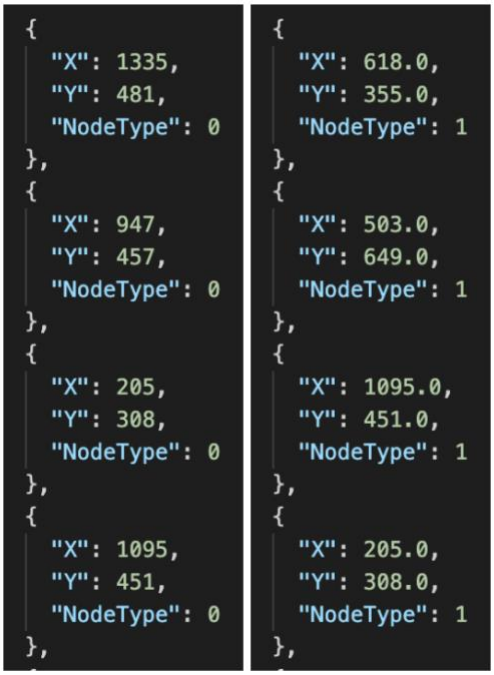


Figure 5.1 Snippet of input and output file content

5.1.1 Network Overlapping Region

To evaluate the wireless interference caused by network overlapping of the proposed framework, it was compared against the **uniform arrangements of hexagonal** [32] (Figure 5.2) and **square arrangements** [33] (Figure 5.3) where the gateways are placed within the communication vicinity. These arrangements were shown to be the most efficient patterns to cover an area with

least overlapping. Besides, it is also compared with the existing solution NewIoTGateway-Select by Karthikeya et al.[18], which we have reviewed in Chapter 2. We generate 30 problems as input with 50 sensor nodes each and collect the output solutions by each technique.

A profiling program is written to take the solution file as input and return the coverage of area in percent for each layer of overlapping. The program would create a visual of 2D spatial plane with white background overlaid with the gateway according to their coordinates in the solution. Each gateway is surrounded by a circle with constant radius and translucent coloured fill that indicates the wireless coverage of it. Technically, when two translucent circles overlap, the intersected area shall have a lower transparency (i.e. more saturated colour) and the more layer of overlapping shall result with even lower transparency. Ultimately, each layer of overlapping would be indicated with different colour.

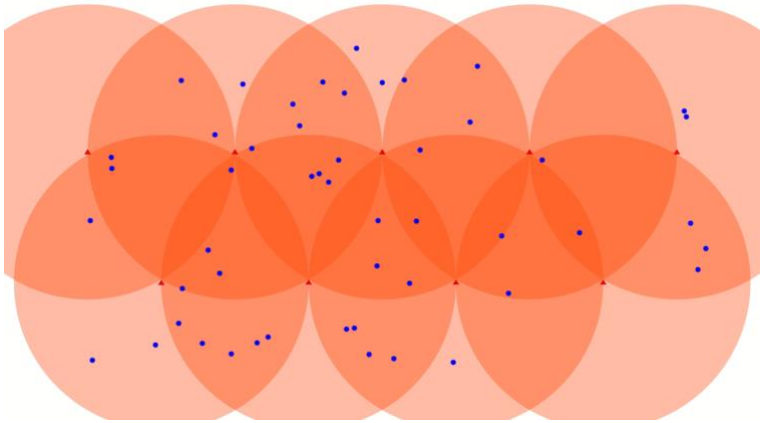


Figure 5.2 Sample of 2D spatial plane visual with hexagonal gateway arrangement

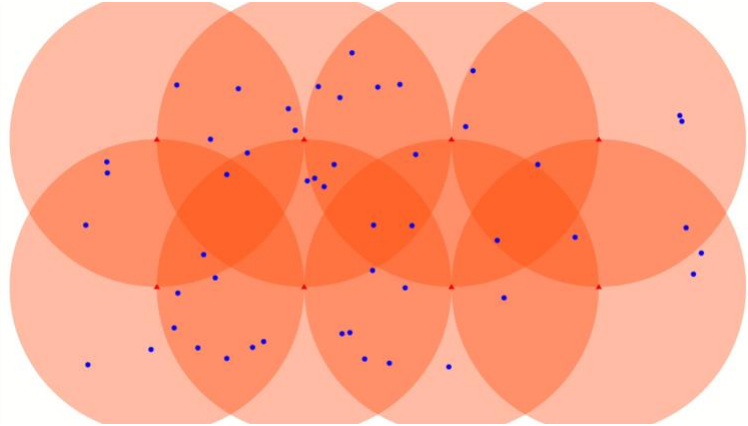


Figure 5.3 Sample of 2D spatial plane visual with square gateway arrangement

The coverage area by different layer of overlapping is acquired from this visual of 2D spatial plane using pixel counting based on the distinct colour of each layer. The distribution of each layer of overlapping in percentage is then calculated and recorded in a table. To better illustrate the overlapping layer, we have come out with a formula to determine the overlapping index as follow:

$$i_o = \sum_{l=1}^{L} l \times A_l \qquad (1)$$

Where i_o indicates the overlapping index, l indicates the overlapping layer, L indicates the maximum overlapping layer and A_i indicates the area percentage of the overlapping layer.

5.1.2 Number of Gateways Required and Network Resiliency

In this simulation, the proposed framework is compared against the **hexagonal arrangement with max coverage and minimum overlapping** [32] which is a most efficient uniform arrangement in terms of area coverage with least overlapping area, and the reviewed work of NewIoTGateway-Select [18] to evaluate the number of gateways required and network resiliency of the

proposed solution. Unlike previous simulation, the hexagonal arrangement has its gateways placed further to maximize the coverage (Figure 5.4). The 3 techniques generate the solution based on the same set of problems from last simulation. For the first part of evaluation, the number of gateway in each solution is extracted and recorded.

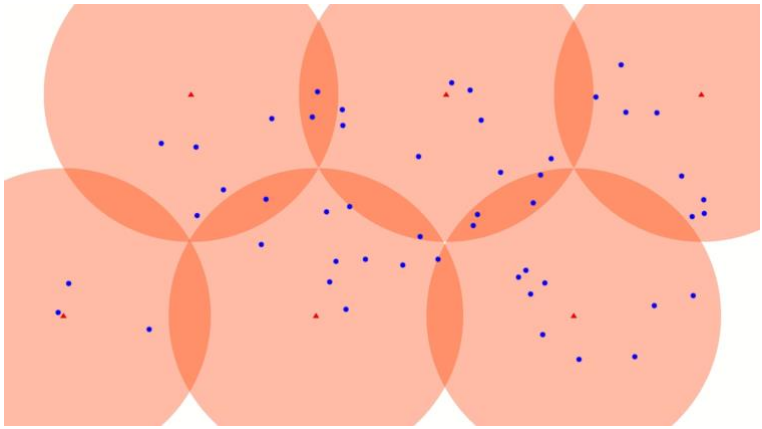


Figure 5.4 Sample of 2D spatial plane visual with hexagonal gateway arrangement with maximized coverage

For the second part of the evaluation, which is the network resiliency, we run fault injection tests on each collected solution based on their problem. A program is written to take the problem (list of sensor nodes) and solution (list of gateway) as input and creates a connectivity matrix of every sensor node with gateways. The gateways are randomly marked as remove one by one until at least one node is not connected to any gateway. All collected solutions and their corresponding problem shall go through the test for 30 times and the number of gateway removed for each iteration, which also indicates the level of fault tolerance is recorded.

5.1.3 Movement required due to change of sensor layout

In this simulation, the proposed framework is compared against NewIoTGateway-Select [18]. The intention of this simulation is to imitate the scenario of rearranging a set of deployed gateways to cater for a new sensor nodes layout, subsequently evaluate the overall movement required for each gateway. To start the simulation, 30 problems with 50 sensor nodes each are generated and serve as inputs for the programs implemented by the 2 techniques to compute respective solutions.

For these 30 problems, says $P_1 – P_{30}$, we regard a single problem as the original sensor nodes layout and its next problem (circularly) as the new sensor nodes layout. For example, P_3 is the new layout of P_2, P_4 is the new layout of P_3, P_1 is the new layout of P_{30} and so on. In such event, we can assume that the differences in positions between the solution of a single problem and its next problem is the movement incurred by the change of sensor node layout.

The program implemented with our proposed framework would be executed again with the same 30 problems as input, but for this time, it would take another input, which is the solution from last problem so that it can take the original layout into consideration to compute a new solution in order to minimizes the movement between original and new gateway locations.

To determine the movement required from original gateway layout to new gateway layout, all distances between gateways in both layout are computed and added into a list along with the two gateways. The element with shortest available distance is popped from the list, followed by removing all other elements containing any of its two gateways. This process is repeated until the list is empty. The sum of distances from the popped elements is recorded as the

result of movement required between original and new gateway layout. Besides, the difference in the number of gateway between the original and new gateway layout is also recorded.

5.1.4 Computational Time

In this simulation, the proposed framework is compared against NewIoTGateway-Select [18] from the perspective of computational time. The programs implemented based on the 2 techniques attempt to solve problems with different sizes (number of nodes of 20 – 80 with the multiple of 20) and iterate for 30 times for each size. For NewIoTGateway-Select, the recording time is the time taken to compute a solution; while the recording time for the proposed framework is the time taken to complete 5000 generations.

5.2 **Summary**

To summarize this chapter, a series of experiments consisting of 4 sets of experiments were designed and setup to benchmark the proposed framework based on different criteria. The simulations involved benchmarking the performances of the proposed framework against existing techniques.

The evaluation criteria include: (1) Network overlapping region – to evaluate the wireless interference caused by network overlapping within IoT gateways; (2) Number of Gateways Required and Network Resiliency – to evaluate the performance of minimizing number of required gateway and fault tolerance; (3) Movement required due to change of sensor layout – to assess the performance of minimizing the gateway movement required from original to new gateway layout due to sensor layout changes; (4) Computational Time – To benchmark the computational speed of the framework to produce a solution.

The simulations shall be executed, and the results are presented in the next chapter.

6 EVALUATION RESULTS

6.1 Simulation Results

The simulation as described in the previous section is executed, the output results are collected and organized for analysis. The following subsections show the simulation outcomes and our findings

6.1.1 Network Overlapping Region

The collected results of the output from the simulation to evaluate the network overlapping region are sorted out and organized as Table 6.1 below, containing the mean percentage of area distribution with different layer of overlapping, along with their standard deviation (σ). With our suggested formula, we also calculated the overlapping index (i_o) based on the results. According to the data in the table, a graph shown in Figure 6.1 is also generated to give a better visualization to the data.

Layer of overlapping	GA Framework		NewIoTGateway -Select		Hexagonal		Square	
	Mean percentage	σ	Mean percentage	σ	Mean percentage	σ	Mean percentage	σ
1	25.76	7.28	19.15	4.45	0.00	0.00	0.00	0.00
2	52.19	10.56	30.61	4.91	41.94	0.00	37.53	0.00
3	12.36	4.06	23.07	3.22	0.00	0.00	0.00	0.00
4	6.97	5.86	17.03	3.28	40.00	0.00	35.14	0.00
5	2.12	3.28	7.66	3.54	0.00	0.00	0.00	0.00
6	0.58	1.94	2.12	1.97	11.14	0.00	19.91	0.00
7	0.01	0.08	0.34	0.78	0.00	0.00	0.00	0.00
8	0.00	0.00	0.03	0.14	6.92	0.00	7.42	0.00
Total	100.00		100.00		100.00		100.00	
i_o		209.29		271.25		366.09		394.49

Table 6.1 Average distribution of layer of overlapping in percentage and overlapping index

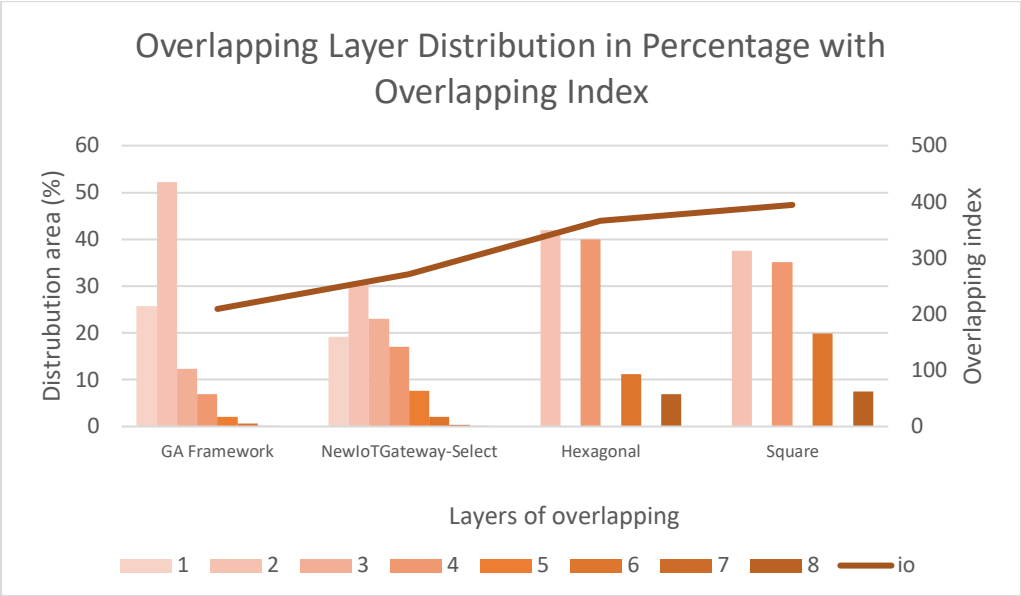


Figure 6.1 Overlapping Layer Distribution in Percentage with Overlapping Index

From the results, we can see that the hexagonal and square uniform arrangements have their distributions over the even number layer of overlapping between 2 to 8. This is due to 2 gateways are placed on each position to ensure the redundancy of k-2. Around half of the covered area of for NewIoTGateway-Select is with 1 and 2 layers of overlapping, while the rest are covered with higher layer of overlapping. For the proposed GA framework, almost 80% of the area is covered by 1 and 2 layers of overlapping, while most of the remaining is covered with layers of 3 to 5. Majority of the covered area of proposed GA framework is below 5 layers of overlapping, while the uniform arrangements have a significant portion of the coverage make up with 6 and 8 layers.

According to the overlapping index, the proposed GA framework is noticeably lower than the other 3 techniques. It is around 23%, 43% and 47% lower than the NewIoTGateway-Select, hexagonal and square arrangements respectively. This shows that most of the covered area of the solutions yielded

by GA Framework are with low layer of network overlapping, which should relatively prevent heavy wireless interference within the lateral-bound network.

6.1.2 Number of Gateways Required and Network Resiliency

From this simulation, the collective of tolerable number of gateway failure before at least a sensor node is uncovered is recorded. The proposed GA framework is evaluated along with the hexagonal arrangement with maximum coverage and NewIoTGateway-Select. The results from a total of 30 x 30 fault injection tests are organized and compiled as the following Table 6.2 and further presented as a combo chart in Figure 6.2.

	Proposed GA Framework		**NewIoTGateway -Select**		**Hexagonal (Max Coverage)**	
	Mean	**σ**	**Mean**	**σ**	**Mean**	**σ**
Average number of gateway failure before a sensor node is out of coverage	3.3	1.0	4.4	1.5	4.5	1.4
Average number of deployed gateway	9.0	0.8	12.3	1.2	12.0	0

Table 6.2 Average number of gateway failure with average number of deployed gateway

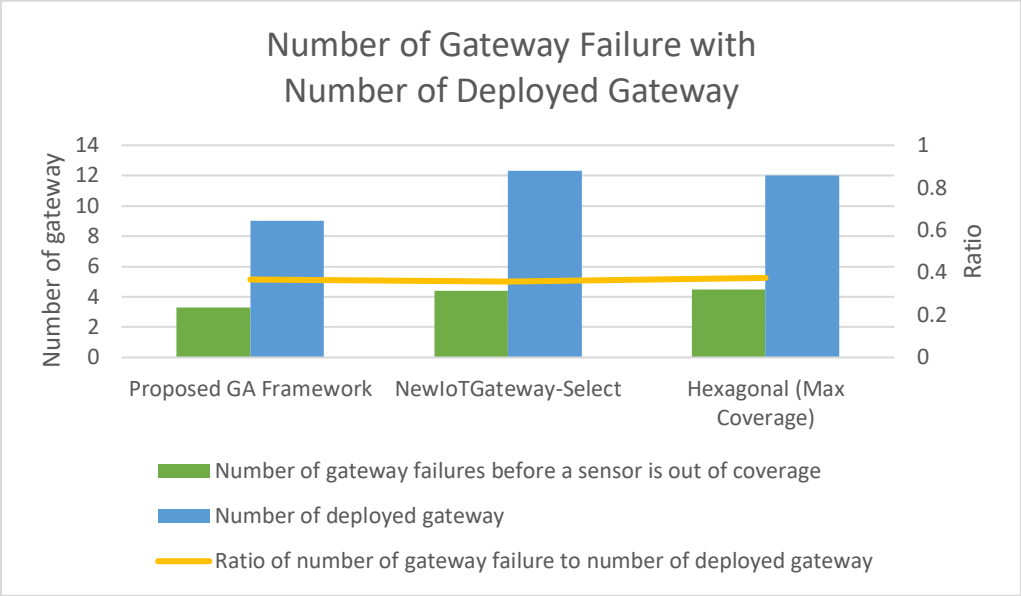


Figure 6.2 Number of gateway failure with number of deployed gateway

Based on the results, we found that our proposed GA framework has a lower tolerance to gateway failure compared to the other techniques, which is around 25% lower. However, the number of gateways deployed is also 25% lower than the other 2 approaches. When we put these 2 sets of number together, we discover that the ratio of number of gateway failure to number of deployed gateways are almost the same through all three techniques, which is around 36%. Therefore, although the GA framework has a tradeoff of lower tolerance to failure in terms of unit of gateway with the reduction of deployed gateways, it actually maintains the tolerance from the perspective of correlation.

6.1.3 Movement required due to change of sensor layout

One of our objectives in this research is to accommodate dynamic sensors movement as consideration during computation of gateway placement scheme, ultimately to reduce overall gateway movement incurred by the change of sensor nodes movement. In this simulation, we find out the total distances

between new and original gateway placement schemes computed by different techniques.

It is notable that we also include the result of our proposed GA framework without the input of original gateways locations. We exclude uniform arrangements from this simulation because we assume that uniform arrangements are static and there is not movement regardless of the sensor nodes changes. The results of the simulation is organized and compiled as Table 6.3 below and presented as a bar chart in Figure 6.3.

	NewIoTGateway-Select		**Proposed GA framework (without input of original gateways)**		**Proposed GA framework (with input of original gateways)**	
	Movement	**Gateway difference**	**Movement**	**Gateway difference**	**Movement**	**Gateway difference**
Average	1815.10	0	1450.40	0	796.03	0
σ	570.79	1.8	389.66	1.3	233.60	1.3

Table 6.3 Average gateway movement required due to change of sensor nodes layout

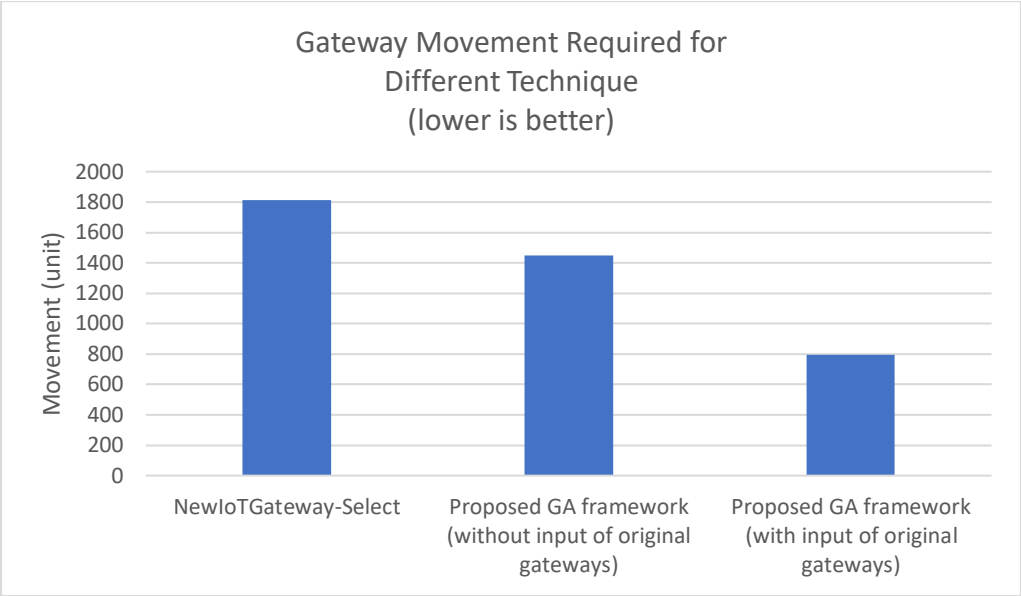


Figure 6.3 Gateway movement required for different technique

Based on the results, the average movement of the 3 techniques over the 30 output solution is 1815.10, 1450.40 and 796.03 units respectively. From the numbers, we find that the movement of NewIoTGateway-Select is around 25% more than the proposed GA framework without input of original gateways, and double of the proposed GA framework with the input of original gateways. From the number of gateway difference, we can see that the average difference are all 0 with a standard deviation of 1.3 to 1.8.

According to the findings, it can be concluded that the proposed GA framework with the input of original gateway locations, is able to drastically reduce the movement of gateway during the event of changes in sensor nodes layout without the need of adding many extra gateways.

6.1.4 Computational Time

The programs implemented with proposed GA framework and NewIoTGateway-Select are set to run in to solve gateway placement optimization problem with different problem size (number of sensor nodes) on

a Windows 10 machine installed with an Intel i7-7700HQ processor and 16GB RAM. The following Table 6.4 and the graph in Figure 6.4 show the average time taken for NewIoTGateway-select to produce a solution and for proposed GA framework to complete 5000 generations.

Number of Sensor Nodes	Computational Time (s)			
	Proposed GA Framework		NewIoTGateway-Select	
	Mean	σ	Mean	σ
20	6×10^{-3}	2×10^{-3}	16.00	0.52
40	15×10^{-3}	4×10^{-3}	31.90	1.3
60	29×10^{-3}	6×10^{-3}	51.93	2.84
80	49×10^{-3}	11×10^{-3}	79.37	1.85
100	74×10^{-3}	17×10^{-3}	102.10	2.88

Table 6.4 Average computational time for different problem size

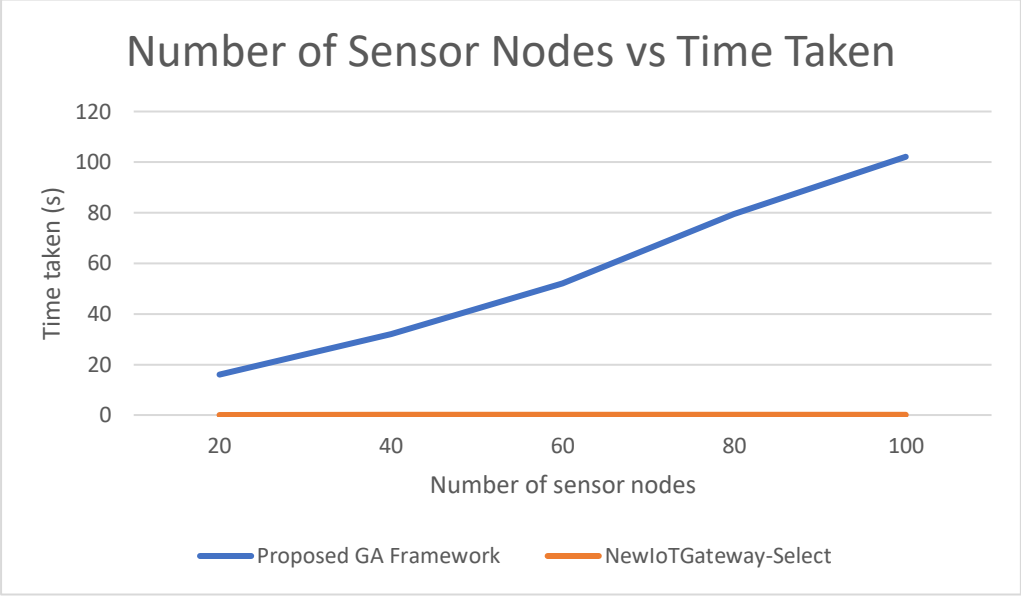


Figure 6.4 Number of sensor nodes vs time taken

According to the results, we can see that the GA Framework has its computational time increases linearly with the problem size, and take around 100 seconds to for problem with 100 sensor nodes. For NewIoTGateway-Select,

it is almost immediate. However, considering that the program is only executed when there is a change in the setup, the execution time is reasonable.

6.1.5 Summary

Based on the evaluation results, we are able to come out with a number of findings regarding the GA framework. The framework is able to give solutions for gateway placement optimization problems with the input of list of sensor nodes with X and Y coordinates, and the solutions include gateway redundancy of k=2 for each sensor nodes.

The framework is also able to provide solutions that has lower overlapping index, where most of the overlapping region is below 5 layers. The solutions also minimize the number of gateways to cover all sensor nodes, but the tolerance of failure in terms of gateway unit is also lower. However, it still able maintain the ratio of failed gateway to deployed gateway to be close with other techniques.

For the required movement of gateway due to change of sensor nodes layout, the GA framework shows good performance when it is fed with the information of original gateway locations as input during the computation. The outputs give significantly better results compared to the existing work. Nevertheless, the computational time of the GA framework is shown notably longer compared to NewIoTGateway-Select, but considering that the it is only executed when there is a change in the setup, the execution time is still reasonable.

7 CONCLUSION AND FUTURE WORK

7.1 Conclusion

To sum up the project, this research has developed a framework to find solutions for gateway placement optimization problems, with the considerations of wireless interference, number of gateways, redundancies, and gateway relocations due to changes of sensor nodes layout. In this research, we have presented the common implementations of deploying IoT gateway in IoT sensor networks for better cost efficiency, and store and forward capabilities. This research is built on top of the basis of collaborative IoT gateway which the lateral bound connection which shall facilitate the implementations of gateway redundancies.

It is realized that although deploying extra gateways can provide redundancies to sensor nodes internet connection, placing too many gateways eventually cause signal overlapping and network interference, also having impact to deployment cost. Besides, IoT sensor network can be dynamic and differ from time to time, gateway placement has to be agile to handle the dynamic nature. Gateway placement optimization is the problem that we tried so solve, which involves finding optimal gateway locations to place gateways in order to fulfill multiple conditions. In this research we have reviewed a number of existing works aimed of solving gateway placement optimizations with different types of consideration factors. Although none of the works have the same objectives as ours, we still manage to find one that is relatively close to our requirements and selected it to be our benchmarking target.

We proposed a gateway placement framework, which takes the sensor node locations, number of gateway desired and original gateway locations as input, and shall produce the proposed gateway locations (layout). Based on our requirements, we have come out with a list of consideration factors and formulate them into gateway placement optimization problem. As more consideration factors are added on top of primitive gateway placement optimization, the problem would get more complex due to variable number variable scope and variable diversity. Genetic algorithm is selected to implement our optimization model in the proposed framework as a global search approach is more preferable to find a solution that does not need to be optimal at each requirement, good enough to fulfill the minimum requirement and with wider search space.

The gateway placement optimization model in our framework is designed with genetic algorithm as basis. Problems and requirements are formulated to adapt the steps in a genetic algorithm lifecycle, which are initialization, selection, crossover, mutation and fitness evaluation. The proposed solution is implemented into software programming logics accordingly, and written in a modern programming language to be compiled as a software. The written software program, along with another software programs created based on the benchmarking solution, are used to go through a series of simulations for performance evaluations.

The simulations are designed to evaluate the performance of our gateway placement framework based on several evaluation criteria, namely: network overlapping region, number of gateways required and resiliency of the network, movement required due to change of sensor layout, and the computational time.

The framework is evaluated against uniform gateway placement, and also the solutions by an existing research work. The evaluation outcomes of the framework are promising for each evaluation criterion, but due to the nature of genetic algorithm, there are still some space of improvement in terms of the computational speed compared to the existing work that was implemented with greedy technique.

At the end of this research, we can conclude that all of the research objectives are achieved, which are 1) to develop a framework to minimize the Wi-Fi interference between IoT gateway in the lateral bound network, 2) to design a technique to minimize the number of IoT gateway placement without sacrificing fault-tolerance and 3) the design of the framework will accommodate dynamic sensors movement as consideration during computation of gateway placement scheme.

7.2 Future Work

GA is known to have a very high scalability. The future works can take advantage of this nature by tweaking the fitness evaluation module, our GA framework can be further improved to adapt with a wider range of gateway placement problem. More considerations factor can be injected on top of the current set to cater with new requirements and constraints. For instance, restricted area to place a gateway, multiple wireless communication technologies, load balancing and so on.

On top of that, there are still rooms of improvement for the computational speed of the GA framework. The current model was implemented with single threaded programming. For future work, the model can be implemented with the concept of multi-threaded programming to greatly improve the computational

speed and raise the upper bound of problem size that can be solved within a time frame.

REFERENCES

[1] Cisco, “Internet of Thing,” 2016.

[2] D. Wang, D. Chen, B. Song, N. Guizani, X. Yu, and X. Du, “From IoT to 5G I-IoT: The Next Generation IoT-Based Intelligent Algorithms and 5G Technologies,” *IEEE Commun. Mag.*, vol. 56, no. 10, pp. 114–120, 2018.

[3] NVIDIA, “Metropolis - Video Analytics & Applications.” [Online]. Available: https://www.nvidia.com/en-sg/autonomous-machines/intelligent-video-analytics-platform/. [Accessed: 01-Dec-2019].

[4] B. Flavio, R. Milito, J. Zhu, and A. Sateesh, “Fog Computing and Its Role in the Internet of Things,” *Proc. First Ed. MCC Work. Mob. Cloud Comput.*, pp. 13–16, 2012.

[5] NVIDIA, “NVIDIA EGX: Accelerating Edge Computing for AI at the Edge | NVIDIA.” [Online]. Available: https://www.nvidia.com/en-sg/data-center/products/egx-edge-computing/. [Accessed: 01-Dec-2019].

[6] M. Hemmatpour, M. Ghazivakili, B. Montrucchio, and M. Rebaudengo, “DIIG: A Distributed Industrial IoT Gateway,” *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 755–759, 2017.

[7] S. Shirmohammadi, W. T. Chai, B. Y. Ooi, and S. Y. Liew, “Taxi-sharing: A wireless IoT-gateway selection scheme for delay-tolerant data,” *I2MTC 2018 - 2018 IEEE Int. Instrum. Meas. Technol. Conf. Discov. New Horizons Instrum. Meas. Proc.*, pp. 1–6, 2018.

[8] T. Adesina and O. Osasona, “A Novel Cognitive IoT Gateway Framework: Towards a Holistic Approach to IoT Interoperability,” *IEEE 5th World Forum Internet Things, WF-IoT 2019 - Conf. Proc.*, pp. 53–58, 2019.

[9] B. Y. Ooi, Z. W. Kong, W. K. Lee, S. Y. Liew, and S. Shirmohammadi, “A collaborative IoT-gateway architecture for reliable and cost effective measurements,” *IEEE Instrum. Meas. Mag.*, vol. 22, no. 6, pp. 11–17,

2019.

[10] S. Ivanov and E. Nett, “Achieving Fault-Tolerant Network Topology in Wireless Mesh Networks,” pp. 1–24, 2012.

[11] S. K. Datta, C. Bonnet, and N. Nikaein, “An IoT gateway centric architecture to provide novel M2M services,” *2014 IEEE World Forum Internet Things*, pp. 514–519, 2014.

[12] M. Kumari, A. Kumar, and R. Singhal, “Design and Analysis of IoT-Based Intelligent Robot for Real-Time Monitoring and Control,” *2020 Int. Conf. Power Electron. IoT Appl. Renew. Energy its Control. PARC 2020*, pp. 549–552, 2020.

[13] A. Mijuskovic, R. Bemthuis, A. Aldea, and P. Havinga, “An Enterprise Architecture based on Cloud, Fog and Edge Computing for an Airfield Lighting Management System,” *Proc. - IEEE Int. Enterp. Distrib. Object Comput. Work. EDOCW*, vol. 2020-Octob, pp. 63–73, 2020.

[14] S. Mnguni, P. Mudali, A. M. Abu-Mahfouz, and M. Adigyn, “A Review On Gateway Placement Algorithms on Internet of Things,” *Proc. Int. Conf. Intell. Sustain. Syst. ICISS 2019*, pp. 479–484, 2019.

[15] J. Zhang, G. Han, and Y. Gui, “An interference-aware cognitive WLAN for high density wireless environment,” *Int. Conf. ICT Converg. 2015 Innov. Towar. IoT, 5G, Smart Media Era, ICTC 2015*, no. 2014, pp. 586–588, 2015.

[16] M. Kim, S. Park, and W. Lee, “Ping-pong free advanced and energy efficient sensor relocation for iot-sensory network,” *Sensors (Switzerland)*, vol. 20, no. 19, pp. 1–18, 2020.

[17] N. Battat, H. Seba, and H. Kheddouci, “Monitoring in mobile ad hoc networks : A survey,” *Comput. NETWORKS*, vol. 69, pp. 82–100, 2014.

[18] S. A. Karthikeya, J. K. Vijeth, and C. S. R. Murthy, “Leveraging Solution-Specific Gateways for Cost-Effective and Fault-Tolerant IoT Networking,” *IEEE Wirel. Commun. Netw. Conf. (WCNC 2016) - Track 4 - Serv. Appl. Bus.*, 2016.

[19] P. Maiti, J. Shukla, B. Sahoo, and A. K. Turuk, “Efficient Data

Collection for IoT Services in Edge Computing Environment," *2017 Int. Conf. Inf. Technol.*, pp. 101–106, 2017.

[20] I. Gravalos, P. Makris, K. Christodoulopoulos, and E. A. Varvarigos, "Efficient Network Planning for Internet of Things with QoS Constraints," *IEEE Internet Things J.*, pp. 1–6, 2018.

[21] X. Yuan, Y. He, Q. Fang, X. Tong, C. Du, and Y. Ding, "An Improved Fast Search and Find of Density Peaks-based Fog Node Location of," *2017 IEEE Int. Conf. Internet Things IEEE Green Comput. Commun. IEEE Cyber, Phys. Soc. Comput. IEEE Smart Data*, pp. 635–642, 2017.

[22] Z. W. Kong, T. B. Tan, B. Y. Ooi, and S. Y. Liew, "Interference-aware Wireless Internet of Things Gateway Placement Scheme," *Proc. - Int. Conf. Comput. Inf. Sci. Sustain. Tomorrow with Digit. Innov. ICCOINS 2021*, no. July, pp. 201–206, 2021.

[23] W. Wenjia, L. Junzhou, and Y. Ming, "Gateway placement optimization for load balancing in wireless mesh networks," *Proc. 2009 13th Int. Conf. Comput. Support. Coop. Work Des. CSCWD 2009*, pp. 408–413, 2009.

[24] A. M. Ahmed and A. H. A. Hashim, "A Genetic Approach for Gateway Placement in Wireless Mesh Networks," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 15, no. 7, p. 9, 2015.

[25] P. Maiti, J. Shukla, B. Sahoo, and A. K. Turuk, "QoS-Aware Fog Nodes Placement," *2018 4th Int. Conf. Recent Adv. Inf. Technol.*, pp. 1–6, 2018.

[26] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science (80-.).*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[27] H. Tian, M. A. Weitnauer, and G. Nyengele, "Optimized gateway placement for interference cancellation in transmit-only LPWA networks," *Sensors (Switzerland)*, vol. 18, no. 11, pp. 1–23, 2018.

[28] F. Loh, D. Bau, J. Zink, A. Wolff, and T. Hossfeld, "Robust Gateway Placement for Scalable LoRaWAN," pp. 71–78, 2021.

[29] I. Gravalos, P. Makris, K. Christodoulopoulos, and E. A. Varvarigos,

"Efficient Network Planning for Internet of Things with QoS Constraints," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3823–3836, 2018.

[30] J. Brownlee, "Local Optimization Versus Global Optimization." [Online]. Available: https://machinelearningmastery.com/local-optimization-versus-global-optimization/. [Accessed: 01-Apr-2021].

[31] K. F. Man, K. S. Tang, and S. Kwong, "Genetic algorithms: Concepts and applications," *IEEE Trans. Ind. Electron.*, vol. 43, no. 5, pp. 519–534, 1996.

[32] Y. Kamer, G. Ouillon, and D. Sornette, "Barycentric fixed-mass method for multifractal analysis," *Phys. Rev. E*, vol. 88, no. 2, p. 022922, Aug. 2013.

[33] "Optimal Packing." [Online]. Available: https://datagenetics.com/blog/june32014/index.html. [Accessed: 13-Dec-2020].